

# Working Papers

## QR '94

### The Eighth International Workshop on Qualitative Reasoning about Physical Systems



#### Program Committee:

Bert Bredeweg

Brian Falkenhainer

Boi Faltings

Yumi Iwasaki

Takashi Kiriya, Program Vice Chair

Riichiro Mizoguchi

Hiroshi Motoda

Toyoaki Nishida, Workshop Chair

Peter Struss

Tetsuo Tomiyama, Program Chair

Daniel S. Weld

Nara-Ken New Public Hall, Nara, Japan

June 7-10, 1994



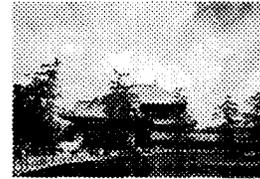
# Working Papers

## QR '94

### The Eighth International Workshop on Qualitative Reasoning about Physical Systems

#### Program Committee:

Bert Bredeweg  
Brian Falkenhainer  
Boi Faltings  
Yumi Iwasaki  
Takashi Kiriyama, Program Vice Chair  
Riichiro Mizoguchi  
Hiroshi Motoda  
Toyoaki Nishida, Workshop Chair  
Peter Struss  
Tetsuo Tomiyama, Program Chair  
Daniel S. Weld



**Nara-Ken New Public Hall, Nara, Japan  
June 7-10, 1994**

#### With financial support from:

American Association for Artificial Intelligence  
Foundation for Nara Institute of Science and Technology  
International Joint Conferences on Artificial Intelligence, Inc.



Nara Convention Bureau

#### In corporation with:

Information Processing Society of Japan  
Japanese Society for Artificial Intelligence  
Nara Institute of Science and Technology  
The Institute of Electronics, Information and Communication Engineers  
The Japan Society for Precision Engineering  
The Society of Instrument and Control Engineers



## Preface

International Workshop on Qualitative Reasoning about Physical Systems (QR) is an annual forum for a small group of researchers (approximately 50) to get together in an intimate atmosphere to survey recent results in qualitative reasoning and to debate new approaches. Since the first workshop in 1987, the workshop site has alternated between the United States and Europe every year.

This year we meet in Nara, Japan, celebrating the community's escape from a simple flip-flop behavior and embarkment in a more complex behavior. Interestingly, this transition from the simple to the complex coincides the recent shift of the community's concern from simplistic toy problems into complex real world problems. In fact, we will have a couple of exciting demonstrations: Mita's self-maintenance copier (physical demonstration) and IBM's mechanism design and analysis using configuration spaces (video-taped demonstration).

We have received 53 submissions. The levels of submitted papers were very high, all addressing interesting issues related to qualitative reasoning. Due to the unfortunate limitation of spatio-temporal resources, we have selected 14 papers for oral presentation and 20 papers for poster presentation, as the most promising papers for stimulating important discussion. This volume contains the written version of these 34 papers.

Roughly, the papers can be classified into three categories. The first is concerned with modeling, a continuing interest of the community. The second is on integration of qualitative reasoning techniques with numerical or/and mathematical techniques. The third addresses task-level reasoning, application of qualitative reasoning techniques to engineering problem solving.

We owe a great debt to the Program Committee for their significant efforts on reviewing submitted papers. We thank American Association for Artificial Intelligence, Foundation for Nara Institute of Science and Technology, International Joint Conferences on Artificial Intelligence, Inc., Mita Industrial, Co. Ltd., and Nara Convention Bureau, for financial support.

Toyoaki Nishida  
Tetsuo Tomiyama  
Takashi Kiriyaama  
May 17, 1994

## Table of Contents

RAPPER: The Copier Modeling Project David Bell, Daniel Bobrow, Brian Falkenhainer, Markus Fromherz, Vijay Saraswat, and Mark Shirley	1
Automatic Construction of Accurate Models of Physical Systems Elizabeth Bradley	13
Modeling the Influence of Non-Changing Quantities Bert Bredeweg, Kees de Koning, and Cis Schut	24
Abstraction Framework for Compositional Modeling Diane Chi and Yumi Iwasaki	36
Model Decomposition and Simulation Daniel J. Clancy and Benjamin Kuipers	45
A Distance Measure for Attention Focusing and Anomaly Detection in Systems Monitoring Richard J. Doyle	55
Prediction Sharing Across Time and Contexts Oskar Dressler and Hartmut Freitag	63
Topology-based Spatial Reasoning Boi Faltings	69
A Semi-Quantitative Physics Compiler Adam Farquhar and Giorgio Brajnik	81
Physical Model Generation in Thermal Engineering Problems described by Partial Differential Equations Donal P. Finn and Pádraig Cunningham	90
Polynomial-time Compilation of Self-Explanatory Simulators Kenneth D. Forbus and Brian Falkenhainer	98
Using Qualitative Physics to Build Articulate Software for Thermody- namics Education Kenneth D. Forbus and Peter B. Whalley	106
Integrating Qualitative Simulation for Numerical Data Fusion Methods Yang Gao and Hugh F. Durrant-Whyte	114
Compositional Modeling for Complex Spatial Reasoning Tasks Kyungsook Han and Andrew Gelsey	124
A Theoretical Analysis of Thought Experiments David Hibler	136
QCMF: A Tool for Generating Qualitative Models from Compartmental Structures L. Ironi and M. Stefanelli	144

Integration of Real-Time System Prototyping with Qualitative and Quantitative Reasoning-Based Parameter Tuning Methods Kiyoshi Itoh	156
Reasoning in Logic about Continuous Systems Benjamin J. Kuipers and Benjamin Shults	164
Context-Dependent Causal Explanations Maria Lee and Paul Compton	176
Qualitative Reasoning with Spatially Distributed Parameters Monika Lundell	187
Qualitative Behavior Hypothesis from Device Diagrams N. Hari Narayanan, Masaki Suwa, and Hiroshi Motoda	197
Visual Reasoning with Graphs Yusuf Pisan	205
Learning Qualitative Models for Systems with Multiple Operating Regions Sowmya Ramachandran, Ray J. Mooney, and Benjamin J. Kuipers	212
An Investigation on Domain Ontology to Represent Functional Models Munehiko Sasajima, Yoshinobu Kitamura, Mitsuru Ikeda, Shinji Yoshikawa, Akira Endou, and Riichiro Mizoguchi	224
A Generic Harness for the Systematic Generation of Multiple Models Q. Shen, R. R. Leitch, and A. D. Steele	234
Model Abstraction for Testing of Physical Systems Peter Struss	246
Supporting Creative Mechanical Design Kun Sun and Boi Faltings	256
Qualitative Reasoning of a Temporally Hierarchical System based on Infinitesimal Analysis Hiroshi Tanaka and Shusaku Tsumoto	266
Constructing Functional Models of a Device from its Structure Sunil Thadani and B. Chandrasekaran	276
A History-oriented Envisioning Method Takashi Washio	286
Activity Analysis: The Qualitative Analysis of Stationary Points for Optimal Reasoning Brian C. Williams and Jonathan Cagan	295
Macroscopic Interpretation of Microscopic Models Kenneth Man-kam Yip	302
Examination of Deep Knowledge in Knowledge Compilers Shinji Yoshikawa, Akira Endou, Yoshinobu Kitamura, Munehiko Sasajima, Mitsuru Ikeda, and Riichiro Mizoguchi	310
Intelligent Computing About Complex Dynamical Systems Feng Zhao	318
<b>Author Index</b>	<b>325</b>



# RAPPER: The Copier Modeling Project

David Bell      Daniel Bobrow      Brian Falkenhainer      Markus Fromherz  
                         Vijay Saraswat                      Mark Shirley\*

May 6, 1994

## Abstract

The programme of research of model-based diagnosis is based on several assumptions, foremost of which is the availability of component-based models of the system being diagnosed. This assumption has been fairly innocuous for diagnosis of networks of combinatorial devices, for which the construction of models with appropriate levels of discriminatory power has been easy. A priori, however, there is no reason to believe that it will be possible to easily construct models of appropriate generality, power and usability for complex, real-time, computational electro-mechanical systems.

This paper reports on a project we started three years ago to construct a component-based model for the input document-handler of a photocopier. Our goal was to develop a model that was rich enough to allow the generation of diagnostic trees with the same discriminatory power as existing documentation (RAPs, or Repair Action Procedures), which are by and large constructed manually. This goal has been met in principle, but we have learnt several useful lessons along the way.

## 1 Introduction

Model-based diagnosis has been a central area of research for members of our group for several years.<sup>1</sup>

For some time it has been recognized that the central challenge in model-based diagnosis is developing appropriate models of structure and behavior for the systems under consideration. To be useful for diagnosis, such models should be *accurate* — they should be discriminating enough to correctly identify exactly those states of the system that are in fact consistent with the given observations. To be usable in practice, they should be *abstract* — so that relevant reasoning can be performed with appropriate time- and space-efficiency. To be economical to produce, they should be *veridical* — they should be designed to capture the physics of the device being modeled so that

---

\*Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304, U.S.A.; Name@parc.xerox.com

<sup>1</sup>Our group, the Scientific and Engineering Reasoning Area at Xerox PARC, includes David Bell, Danny Bobrow, Johan de Kleer, Brian Falkenhainer, Markus Fromherz, Olivier Raiman, Vijay Saraswat, Mark Shirley and Brian Williams.

they could be reused in all those settings in which the device would be used. Global behavior should be derived from local interaction between components. This is usually called the “No function in structure principle” [1].

Models with these properties have been easy to construct for networks of electronic devices. But will it be possible to construct such models with reasonable effort for other real-world designed systems? In particular, it is natural for us to ask this question for the very complex engineered devices about which detailed information is available to us, namely, reprographics machines. Typically, these machines consist of hundreds of electrical, mechanical, electro-mechanical and computational components — sensors and effectors — involved in moving sheets of paper through mechanical pathways, using complex xerographic processes to transfer optical images (or bit-patterns) onto marks on paper, and applying various physical transformations to streams of paper (stapling, stitching, shrink-wrapping, . . .) — while keeping the entire process under computational control.

Fortunately, there already exists extensive diagnostic documentation for such machines, principally in the form of detailed diagnostic trees called RAPs (Repair Analysis Procedures). This gives us a pre-existing standard for coverage and accuracy that we could try to match. The question we asked ourselves, therefore was:

*For a complex, real-world, computational, electro-mechanical system, is it possible to systematically create an abstract and veridical model SD of structure and behavior that is such that the diagnostic tree generated by SD is query-equivalent to existing diagnostic documentation?*

In more detail: How much detail would be necessary, particularly when describing signal and error propagation? At what level of abstraction would we have to model continuous processes (e.g. air-knife)? How detailed would information about geometry and spatial organization and interaction have to be? How detailed would information about temporal interactions between sub-systems have to be? Which ontologies for space and time would be appropriate? Could the models plausibly be generated as a byproduct of design? Could models be constructed explicitly at a level of effort that would be justified when compared to the existing approach to creating service documentation?

This paper is a preliminary report on our work in answering these questions. We chose the Recirculating Document Handler (RDH) subsystem of a copier in wide use — a major reason was that this subsystem is the only subsystem that handles customer input documents, and is relatively isolated from other components that are concerned with imaging and moving marked sheets. In brief, we have built an abstract, qualitative model that is detailed enough to produce diagnostic trees that match existing documentation. But we have learnt several important lessons along the way, chief among them being the importance of understanding both current work-practice and existing processes for informal model construction, and the wider organizational context in which re-engineering of current work processes (such as that implicit in the model-based diagnosis approach) is to be done.

The rest of this paper is as follows. We first briefly describe the particular subsystem we modeled. We then consider the particular modeling viewpoint we adopted, and discuss some representational issues and the diagnostic algorithm we employed. Finally, we step back and review some lessons we learnt (are learning) from this (ongoing) work.

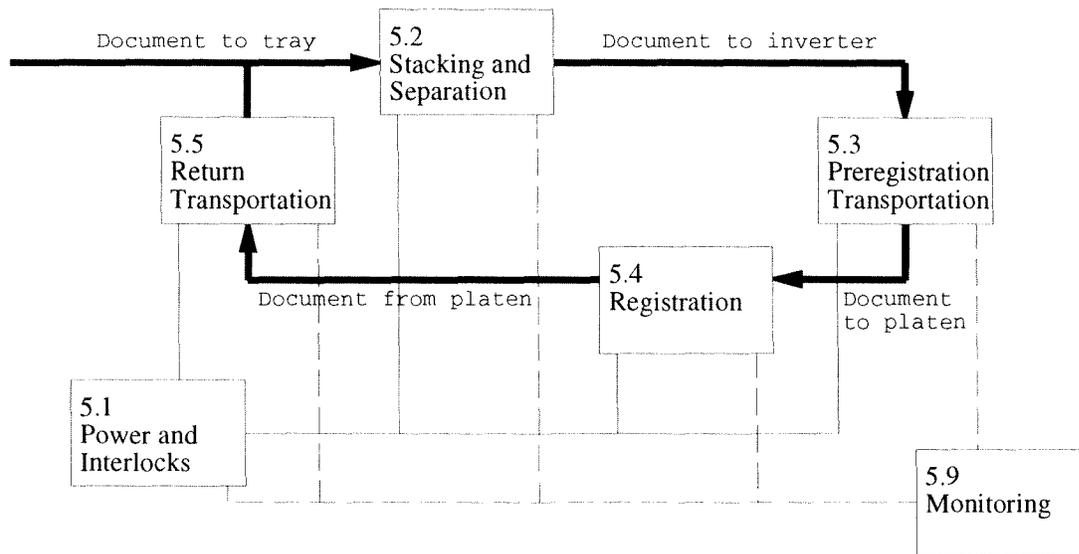


Figure 1: Schematic view of the Recirculating Document Handler

## 2 The task

The RDH for the particular machine we examined is responsible for feeding documents one at a time through a series of rolls from the tray to the platen glass, registering them on the platen for exposure, and moving them back to the document tray (Figure 1).

The system is broken up into several subsystems.<sup>2</sup> The first subsystem handles power and interlocks. It provides motor drive to subsequent chains. It contains primarily the interlock mechanism and the AC motor that supplies torque to the rest of the document handler. The interlock mechanism ensures that the left-cover, right-cover and the input station are all latched shut; only then is power supplied to the motor. It contains the mechanical components for interlocking (a counter-balance and a magnetic catch) — these control interlock switches which electronically signal whether or not the latch is in place.

The second subsystem handles document stacking and separation in re-circulating mode. Its principle operations are to sense the presence of a document stack in the RDH (and report the height of the stack), achieve sheet separation using the air knife<sup>3</sup> and provide vacuum to transport the bottom-most sheet into the next chain. It contains solenoids, (position and light) sensors, valve assemblies, vacuum blower assembly, vacuum transport assembly, and stack height sensor.

The next subsystem handles the preregistration transportation of paper from the input tray (starting at the takeaway roll) to the platen. In simplex mode, it simply transports the paper around the paper-path loop so as to provide the paper to the platen face-down. In duplex mode, this chain is responsible for inverting the paper and providing the paper “face-up” to the platen. It consists primarily of roll stations, the inverter assembly (for duplex sheets) consisting of an inverter roll,

<sup>2</sup>We concentrate only on those subsystems that come into play during the operation of the *recirculating* document handler. The document handler also provides the functions of transporting computer forms and sheets fed from a side-feeder.

<sup>3</sup>A side-ways flow of air that flutters and thus separates top sheets from the bottom one.

an inverter gate and two different output paper paths, a bi-directional roll station (to handle both simplex and duplex), and forward and reverse drive assembly.

The fourth subsystem is responsible for driving the document onto the platen glass, stopping the document on the glass in the proper position for exposure, and after exposure driving the document off the glass into the return transport. It uses several main component assemblies. The document drive assembly drives the paper up to the registration fingers which stop the document. The clamp plate assembly then clamps the document in this registered position. Once registered and clamped, the fingers retract, clearing the paper path for future transport. After exposure, the clamp plate moves up allowing the document drive assembly to drive the document off the glass.

The fifth subsystem handles the return transportation of paper from the platen to the input tray, without curling the document. It consists of a document exit roll station, an exit roll drive clutch, decision gates, baffles, and a static eliminator.

Finally, an electronic subsystem monitors the state of the signals received from the switches and sensors within the document handler. If it detects a fault, the document handler operation is stopped and a message is displayed on the control panel.

Overall, the system we model consists of about 160 components, with approximately 40 different component types. The components are pneumatic, mechanical, electro-mechanical, electrical and computational and have time-varying behavior.

**The diagnostic task** Innumerable things can go wrong in this complex process. Sheets of paper may not separate properly, may jam during transport, may not register correctly (producing skewed copies), or may stick together because of electro-static charge.

When things go wrong a service technician is called. Part of his job is to *diagnose* the machine by making a series of tests on the system to refine his hypotheses until the faulty component(s) have been identified. In this task, the service technician is guided by a thick binder of *Repair Analysis Procedures* (RAPs). As the copier's builtin monitoring software reports a fault code in case of a malfunction, there is one RAP for each fault code. Starting from this initial symptom, a RAP includes a sequence of tests and observations for the service technician to follow, culminating in some suggested repair actions.

Currently, RAPs are generated manually by systematic experimentation. Our goal is to construct a qualitative model of the copier which would be adequate to either interactively guide a service technician through the repair process or automatically generate the Repair Analysis Procedures.

### 3 Model and Methodology

Diagnosis of time-varying systems is a complex problem. Our task is aided by the *sequential* and *cyclic* nature of activity in the RDH.

**“Critical event” modeling.** The correct overall functioning of the RDH is generally determined by the behavior of each component at certain “critical” periods, typically determined by the time that a sheet of paper usually passes a particular position in the path. Further, interactions between successive cycles of paper through the RDH are very rare and can be ignored to the first degree of approximation. This allows us to adopt the following modeling style. We model *one* “typical

pass" of a "typical" sheet through the system, and for the typical pass, state the deviations from normality of various component behaviors at the critical time-events (cf.,[7]).

For instance, a segment of a paper-path takes on as value a "sheet". A sheet may be the value *none* (if in fact no sheet is transmitted during the prototypical pass) or *doc(P)*, where P is a complex description with attributes *size*, *dep\_time* and *arr\_time*. The normal value for *size* is one indicating a cleanly separated sheet of paper passing through that paper-path segment during the prototypical pass. It may be *shingled* if in fact two sheets stuck together were passed in. The values that *dep\_time* and *arr\_time* take on include *normal*, *early\_1*, *early\_2*, *late\_1* and *late\_2*, indicating the time (relative to normal) at which the document departed from the station at the beginning of the path, and arrived at the station at the end of the path. In addition, *arr\_time* can take on the value *jam* to indicate that paper did not leave the station at the end of the path.

Other types of entities in the system (e.g., representing torque, fluid-path, sensor signals, electrical wires, etc.) could also be modeled as taking on values in a discrete set. (For example, a sensor signal is one of *step*, *clear*, *blocked*, or *pulse(T)*, where T specifies relatively a possible delay (*none*, *early*, *late*.)

Such a representation decision eliminates the need for explicit representation of time and complex temporal constraints. In some cases, a sheet passes the same device more than once (e.g., inverting roller for a duplex sheet). This is handled by having that component deal with a tuple of values, one for each moment of interaction between the device and the paper during the prototypical pass. However, genuine repetitive actions (e.g., the comparison between multiple passes) cannot be modeled. <sup>4</sup>

**Classifying behavior** For the task of diagnosis, we are interested in identifying and therefore modeling both correct and incorrect behavior. As an example, a roller might transport a sheet of paper with normal or low speed. With each behavior, we associate the (qualitative) states or *modes* of the process consistent with that behavior. For example, the roller behavior just described might be due to the roller being in normal or worn mode.

Correct and incorrect behavior is modeled for those cases only where the input is "in band", i.e. for each component, we make a deliberate choice of which values are accepted as valid input values. For all other inputs, we make the simplifying assumption that no information about the behavior of the device can be gained in these situations, and therefore usually provide some default behavior only. This is in line with another decision, namely the *single-fault assumption* that we will see at most one component in a fault mode. For the task at hand, this significant assumption is appropriate. Whenever an error is encountered as a sheet moves around the paper path, the physical system is shut down, thus masking any "down-stream" faults. This strongly sequential nature of the artifact allows an analysis of most problems as single independent faults.

**Encapsulating complex processes.** Since our task-objective was to be RAP-equivalent, we did not have to create detailed models of the internal working of most components; rather coarse

---

<sup>4</sup>This still allows us to handle some kinds of faults that arise only on multiple passes of a document through the system. For example, suppose that a large number of copies are being made of a given document. The state of the platen glass is such that a small amount of extra static charge builds up on the sheet on each pass. If the charge crosses a threshold then it can cause the sheet to stick and be detected as late at the tray exit sensor. This kind of multi-cycle interaction can still be tackled through the "prototypical-pass" approximation.

qualitative models sufficed. For example, the actual functioning of the air-knife involves a complex interaction between a stack of paper, air blowing on the stack to separate the sheets of paper, and a vacuum system that sucks the bottom sheet down onto a moving belt. To describe the *correct* functioning of such a system (e.g., in terms of air pressure, suction effects, the physics of paper separation, etc.) would be a significant undertaking in physical modeling. But for the task at hand, we are only interested in the (correct and incorrect) interactions of a process with its environment, *regardless* of its internal structure.

Similarly, the start-up circuit for an AC motor exhibits complex dynamics, which did not have to be modeled in detail. For our purposes it was sufficient to encapsulate it as a single “component” whose input/output behavior was equivalent to the RAP’s view of the subsystem.

**Hierarchical modes.** Because of the sheer size of the system being modeled, we found it very convenient to associate hierarchical modes with subsystems. In general the mode of a subsystem was some function of the mode of its components; typically, it was just a tupling. This meant that the overall system had a mode that took on as value a tree of mode-values. This required us to handle the implementation of single-fault assumptions with care — just because the mode for one device was known to be ok, it was not possible to instantiate the modes for all other devices, since the shape of the mode-tree (i.e., the shape of the tree underlying other sub-systems) may not yet have been determined.

**Implementation and system-level issues.** The model is implemented in an early version of the language `cc(AL)`, a concurrent constraint language [8] with the “attribute-value” constraint system and naive arithmetic. Terms of the form `P[dep_time >> normal]` designate an av-list identical to `P` except that the value of the attribute `dep_time` is `normal`. Accessor notation (`P.dep_time`) can be used for obtaining values at attributes. Limited but useful constraint-solving over av-lists is allowed. AV-lists proved invaluable in passing, accessing and updating complex aggregate data-structures between processes.

Each component type is represented by a predicate whose clauses describe the possible behaviors for all modes of the component and for “out of band” inputs. Interconnection axioms establish local component interactions via shared *node* variables. Figure 2 shows a simple example, the model of a roll station.

A simple compiler translates `cc(AL)` programs into (Sicstus) Prolog, using delay primitives to control evaluation of non-ground arithmetic expressions. The diagnostic algorithm is implemented as an invocation of the model, under a pro-active single-fault assumption (which is not in pure Prolog). This version of the compiler does not generate code that does constraint propagation at run-time. Hence, in essence the run-time behavior is that of Prolog-style depth-first generate-and-test search.

## 4 Diagnosis in RAPPER

Given an observed fault code which indicates faulty behavior, we can query the RDH model for all possible modes of components consistent with this observation. As explained above, we make the single-fault assumption.

**Roll Station.** If a roll station receives no document or low torque, then it does not pass any paper through. In other (in-band) cases, its behavior is determined by whether it is ok or worn. If more than one sheet of paper is simultaneously received (shingled), then a jam is asserted. Otherwise, the sheet is transferred normally (if the device is ok) and with an incremental delay otherwise.

```

Mode is {ok, worn_roller}
ROLL_STATION(Mode,      Torque, PaperIn, PaperOut) */

roll_station(_M,      _T,      none,      none).
roll_station(_M,      Torque, doc(P[arr_time >> jam]), none):- low_torque(Torque).
roll_station(worn_roller, normal, doc(P[size >> shingled, arr_time >> jam]), none).
roll_station(ok,      normal, doc([size >> shingled, arr_time >> jam]), none).
roll_station(worn_roller, normal, doc(P[size >> one]), doc(P_o[size >> one])):-
    delayed_doc(P,P_o).
roll_station(ok,      normal, doc(P[size >> one]), doc(P_o[size >> one])):-
    transferred_doc(P,P_o).

low_torque(low).
low_torque(none).

transferred_doc(P[arr_time >> A, dep_time >> A], P[dep_time >> A, arr_time >> _New]).
delayed_doc( P[arr_time >> A, dep_time >> A], P[dep_time >> Dep, arr_time >> _New]) :-
    delayed(A,Dep).

delayed(late_2,late_2).
delayed(late_1,late_2).
delayed(normal,late_1).
delayed(early_1,normal).
delayed(early_2,early_1).

```

Figure 2: A sample component model: Takeaway Roll Station

**Diagnostic Algorithm** Concretely, given a symptom, all hypotheses  $W$  consistent with this symptom can be computed. A hypothesis  $w \in W$  is the triple  $\langle p_w, N_w, M_w \rangle$  describing a specific fault, where  $p_w$  is the probability of hypothesis  $w$ ,  $N_w$  a set of probe points  $x$  with their values  $v$ , and  $M_w$  a set of components  $c$  with their modes<sup>5</sup>  $m$ .  $p_w$  is defined by  $p_w = \prod_{(c=m) \in M_w} p_{c=m}$ , where  $p_{c=m}$  is the prior probability that component  $c$  is in mode  $m$ . Probe points  $x$  have domain  $V_x = \{v \mid w \in W \wedge (x = v) \in N_w\}$ .

Given the hypotheses consistent with the symptom, we want to rank the probes according to how well they can distinguish between the hypotheses, and lead to a diagnosis. Currently, we use a standard Shannon entropy calculation [3]. When used in an iterative algorithm that chooses probe points, accepts their values, and removes inconsistent hypotheses, this correspond to a one-step lookahead [2]. Given set  $W$  and a probe point  $x$ , the expected entropy is defined by

---

<sup>5</sup>A hypothesis contains all component modes, not just the faulty one.

$$H_e(x, W) = \sum_{v \in V_x} p(W_{x=v}) H(W_{n(x=v)})$$

where  $W_{x=v}$  are the hypotheses consistent with  $x$  being  $v$ ,  $W_{n(x=v)}$  is the respective normalized set,  $p(W)$  is the total probability of the hypotheses  $W$ , and  $H(W)$  is the entropy of hypotheses  $W$  defined by  $H(W) = -\sum_{w \in W} p_w \ln p_w$ .

Based on  $W$  and  $X$ ,  $X_r$  is a list of probe points ranked with respect to their expected entropy. The algorithm is parameterized such that subsets of  $W$  ("leading hypotheses" based on  $p_w$ ) and  $X$  ("relevant probe points" based on  $V_x$ ) are used to compute  $X_r$ .

We have also implemented a variant of this diagnostic algorithm that takes into account the hierarchy of the model's structural description. Essentially, instead of using hypotheses  $W$ , an abstraction  $A_l$  of  $W$  at subsystem level  $l$  is used to rank probe points. Diagnosis starts at an abstract level and adaptively proceeds to a lower level only when no discrimination among hypotheses can be performed at this level. The idea of this variation is that initially those probe points are preferred that rule out entire subsystems.

**Interactive Diagnosis** In interactive diagnosis, the algorithm starts from the symptom and computes an initial set of hypotheses  $W$  with a ranked probe list  $X_r$ . The technician chooses one of the top-ranked probes  $x \in X_r$ , makes a measurement  $x = v$ , and reports the observed value back to the system. The diagnostic algorithm then removes all hypotheses inconsistent with this observation from  $W$ , recomputes  $X_r$ , and iterates. The algorithm halts if only one hypothesis is left, indicating the faulty component.

**RAP Generation** With a similar algorithm, RAPs can be generated by automating probe selection and anticipating all possible probe values. Instead of having the user choose probe points, the first probe point  $x$  in  $X_r$  is taken, and then all possible values are "envisioned" instead of reacting just to the one measured by the user. Thus, the generator spans, at each chosen probe point, a diagnostic subtree with  $|V_x|$  branches, one branch for each possible value in  $V_x$ . For each branch of the subtree, the generator proceeds as in the interactive diagnostic algorithm, removing inconsistent hypotheses and ranking the remaining probe points, before generating the next subtree.

This procedure results in one diagnostic tree per fault code, where internal nodes are probe points, branches are transitions to subsequent probe points based on possible probe values, and leaves are diagnoses.

When producing RAPs from the diagnostic trees, a range of formatting options are available. For example, it is useful to order the branches of each node such that shorter paths come first. Also, similar measurements of the same types of components can be summarized, especially if they appear in a degenerate subtree which simply checks each component in a suspect set. Another formatting option is to add *focus* information, which is a statement that tells the user on which subsystem all further hypotheses of a diagnostic subtree will focus (see below). All these operations result in skeletal RAPs, which are basically annotated, ordered diagnostic trees. Figure 3 lists a typical skeletal RAP (excerpt).

RAP generated for fault code *a069*:

*Measure probe po3.*

Is the value of po3 normal?

**Yes** (All suspects are now in 'chain-5-3'.)

*Measure probes [cm,cn,cp,co] in 'chain-5-3'.*

Is the value of any of them float?

**Yes** Diagnoses:

cm: cm in 'chain-5-3' is open

cn: cn in 'chain-5-3' is open

cp: cp in 'chain-5-3' is open

co: co in 'chain-5-3' is open

**No** (co is v5dc, cp is dc-com, cn is dc-com, cm is blocked)

Diagnosis: doc-to-platen in 'chain-5-3' is obstructed

**No** (po3 is none)

(All suspects are now in 'chain-5-4'.)

*Measure probe reg-finger-position in 'chain-5-4'.*

Is the value of reg-finger-position in?

**Yes** Diagnosis: pwb-5-4 in 'chain-5-4' is bad

**No** (reg-finger-position is out)

*Measure probes [at,as] in 'chain-5-4'.*

Is the value of any of them float?

**Yes** Diagnoses:

at: at in 'chain-5-4' is open

as: as in 'chain-5-4' is open

**No** (as is dc-com, at is v24dc)

Diagnoses:

gear in 'chain-5-4:doc-drive-assembly' is bad

drive-rolls in 'chain-5-4:doc-drive-assembly' is bad

clutch in 'chain-5-4:doc-drive-assembly' is stuck-open

Figure 3: A skeletal RAP

## 5 Evaluation and Future Work

**Coverage of RAPs.** The evaluation of coverage is mostly very encouraging — the generated RAPs cover the diagnoses of existing RAPs fairly well. Missing diagnoses can often be traced back to inadvertent omissions in the model. In a few cases, the generated RAPs also show diagnoses that are missing from the existing RAPs.

The existing RAPs also often summarize a series of diagnoses for similar components, and contain intermediate, general hints (abstract diagnoses). Both features can be achieved in the generated RAPs. However, the existing RAPs are clearly richer in their explanations, and also contain “catch-all” diagnoses and references to other RAPs, to be applied when all else fails.

Observations obviously missing from the generated RAPs are those about “non-systemic” events (e.g., loose or burned objects), missing components, and observations that are functions of the device running in diagnostic mode. The existing RAPs also emphasize a different set of observations, which may be due to a different probe cost model.

One area not addressed by the generated RAPs is descriptive documentation. The existing

RAPs contain background information on symptoms, components, functions, and general servicing information strategically placed throughout the procedures. This information is meant to outline the context and point to further explanations, or to related RAPs. Often, RAPs also contain schematic diagrams and references to the descriptive documentation (another thick binder). This could be somewhat improved by using keywords and an appropriately structured database of canned texts, but the human analysts clearly add important value to the decision trees.

**Simple modeling techniques suffice.** For diagnostic models of mechatronic systems, the 80/20 rule seems to apply — simple models can go a long way.

In retrospect, while the “critical event” modeling style we adopted considerably simplified the task of generating the model, it may not have been as useful as we thought. To understand what counted as deviation from normality for a particular device, we had to understand what counted as abnormal for devices farther downstream: this required envisioning the component’s behavior in context, and characterizing the histories of the signals in and out. We were helped by the fact that the subsystem has one focus of interest, namely the sheet moving through the system, but this often proved awkward.

We are exploring the relative merits of different modeling styles in terms of both simulation speed and naturalness of the description. This includes using discrete event-based models of devices in either a traditional event-driven simulator or a temporal constraint system. It is possible that the smooth integration of event-driven simulation ideas in this context will require the use of certain non-monotonic features, such as “safe defaults”. We are also exploring whether it is possible to automatically derive critical events from a simulation, and hence generate the critical event model.

Nevertheless, by and large, we feel we understand the technical issues involved in designing modeling languages to allow the representation of models of electro-mechanical systems at the level of detail necessary for constructing most RAPs. Simple constraint languages, perhaps augmented with real-time features (as we are currently investigating) seem to be adequate.

Perhaps the most surprising lesson seems to be: *General mechanisms for reasoning about space, time, spatial and temporal interactions may not be necessary for useful diagnostic models of real-world devices.* Clearly, this will have to be further substantiated – at least in the reprographics domain – as we work on other subsystems of a reprographic engine. But to date we have been surprised by how far we could go with extremely simple representations.

**Sometimes simple inference techniques suffice.** Currently, very little control is exercised over the search tree; nevertheless under a single-fault assumption the entire fault tree for a single symptom can be generated in a few seconds on a SparcStation. For larger diagnosis problem, it may well be necessary to focus on a few probable diagnosis (as in, for example, Sherlock [4]), as opposed to exploring all diagnoses simultaneously. Regardless, a better conceptual understanding is needed of combining probabilistic information with constraint-based computation.

**Understanding the larger organizational setting is crucial.** An unexpected lesson we are learning is that the development, delivery and deployment of knowledge- or model-based diagnostic systems is seriously affected by larger organizational dynamics. In the case of Xerox, diagnostics involves several different groups of people — product and design engineers, service analysts,

service technicians, management, researchers developing computational diagnostic systems — who may not share a common background about field service and diagnosis. An appreciation of the organizational dynamics between these groups is essential to understanding the appropriate role of technology in the work-place.

When we started the project we were by and large unaware of the crucial role played by *service analysts*. These are a group of people — distinct from the engineers who work on developing products — charged with assimilating all the sources of information (if any) about the given product to produce documentation to be used by service technicians. As we ploughed through the various sources of documentation about the RDH — principles of operation, Block Schematic Diagrams, RAPs, parts lists, system and module operation descriptions — we began to gain a better understanding of the complex assimilation task performed by service analysts (who manually produce RAPs currently), and of the tools that may be of use in aiding them in their task. (In some cases, service analysts may not even have access to some of the documentation — they may just have a “Box from Heaven” whose modes of correct and incorrect functioning they try to infer by literally “pulling wires” and observing resultant symptoms.)

Equally, in order to make a diagnostic system useful for *service technicians*, a series of additional issues on top of modeling and diagnostic algorithms have to be addressed. Among these are explanation, learning and adaptation, graceful degradation, and integration with complementary techniques [6].

We realized that tools need to be integrated into actual work practice, rather than causing “task interference”; tools should be a time saving practice for work *already* being done rather than appear to be additional work. This requires studying current work practice and working with practitioners to evolve new work methods.

In particular, it is important to present model-based diagnosis technology as an incremental build on the existing work practice. To do this, the new technology must support all the deliverables the old technology delivered that are still considered essential:

**Network models (pathways of interaction) should be built.** We’ve seen that analysts already build Block Schematic Diagrams (BSD’s), they are kept up to date, and are used by field service technicians for deductive diagnosis. Guidelines for building BSD’s need to be refined (e.g. to capture all of the pathways of interaction we would like).

In addition, service analysts are also responsible for producing a “Principles of Operation” document. How can model-based reasoning techniques be used to aid in the task of generating such documents?

**Component models should extend FMEA-like models.** Failure Mode Effects Analysis (FMEA), Fault Insertion and Failure Analysis are three current practices that develop a qualitative understanding of the machine behavior. The modes of each component are identified, and the resulting effects are captured. We should understand what additional benefits are provided by the more explicit and detailed compositional models we have been developing, and how such a modeling activity can be introduced as an extension of current practice.

**‘Just-In-Time’ modeling should be supported.** In subsequent projects we have found that service analysts develop a more accurate knowledge of the machine over time through envisionment,

tests where bad modes are inserted into a machine and behaviors observed, and through feedback from field experience. In addition, components of the machine are constantly being changed and/or updated. This makes model development an evolving process, with analysts recording their models right before the moment they are needed. Modeling is far from being the one-time process that design engineers engage in and hand-off to customers downstream. What kinds of model-analysis and maintenance tools need to be developed to aid in this task?

**Future work** These and many other related questions are being taken up in our continuing work on understanding the engineering and product development process within Xerox. We have been developing a Xerox-wide modeling and simulation infra-structure, a methodology for representing and reasoning with real-time and hybrid systems and for using constraint-based models to generate control software, and a suite of tools to enhance current work-practice in smart service (model-based diagnosis, FAST/FMEA tools, descriptive documentation, "tips" data-base, . . . ). We expect to report on these activities in subsequent papers.

## Acknowledgement

The work reported here has benefited immeasurably from interactions with Johan de Kleer, Brian Williams and Olivier Raiman. Special thanks are also due to Bob Easterly, and Ken Kahn for valuable discussions and comments.

## References

- [1] Johan de Kleer, John S. Brown, "A Qualitative Physics Based on Confluences", *Artificial Intelligence*, vol. 24, 1984; also in: [9], pp. 88–126.
- [2] Johan de Kleer, Olivier Raiman, Mark H. Shirley, "One Step Lookahead is Pretty Good", in: *Proc. Second Int. Workshop on Principles of Diagnosis*, Milano, Italy, October 1991, pp. 136–142.
- [3] Johan de Kleer, Brian C. Williams, "Diagnosing Multiple Faults", in: *Artificial Intelligence*, 32, 1987.
- [4] Johan de Kleer, Brian C. Williams, "Diagnosis with Behavioral Modes", in: *Proc. 11th IJCAI*, 1989, pp. 1324–1330.
- [5] Ken D. Forbus, "Qualitative Physics: Past, Present and Future", in: H. E. Shrobe and the American Association for Artificial Intelligence (Eds.), *Exploring Artificial Intelligence*, Morgan Kaufmann, 1988, pp. 239–296; also in: [9], pp. 11–39.
- [6] Markus P. J. Fromherz, Mark H. Shirley, "Supporting Service Technicians: Model-Based Diagnosis in Context", in: *Proc. Workshop on AI in Service and Support at AAAI'93*, Washington, DC, July 1993, pp. 59-69.
- [7] Walter Hamscher, "Temporally Coarse Representation of Behavior for Model-Based Troubleshooting of Digital Circuits", in: *Proc. 11th IJCAI*, Detroit, MI, August 1989.
- [8] Vijay A. Saraswat, *Concurrent Constraint Programming Languages*, MIT Press, 1993.
- [9] Daniel S. Weld, Johan de Kleer (eds.), *Readings in Qualitative Reasoning about Physical Systems*, Morgan Kaufmann, 1990.

# Automatic Construction of Accurate Models of Physical Systems

Elizabeth Bradley\*  
University of Colorado  
Department of Computer Science  
Boulder, CO 80309-0430  
lizb@cs.colorado.edu

## Abstract

This paper describes some preliminary work on a program that builds ordinary differential equation (ODE) models of target systems from user-supplied hypotheses, observations, and specifications. Its implementation exploits symbolic and qualitative reasoning whenever possible, only resorting to low-level numeric methods if absolutely necessary. ODE theory and domain-specific rules are used to combine hypotheses, to check evolving models against the observations modulo the resolution inherent in the specifications, to remove unnecessary terms, and to synthesize new terms from scratch if need be. This tool has been designed to use sensors and actuators in an *input-output* approach to modeling real physical systems.

## Introduction

One of the most powerful analysis tools in existence — and often one of the most difficult to create — is a good model. Expert model-builders typically construct hierarchies of successively subtler representations that capture the salient features of a physical system, each incorporating more physics than the last. At each level in the hierarchy, the modeler assesses what properties and perspectives are important and uses approximations and abstractions to focus the model accordingly. The subtlety of the reasoning skills involved in this process, together with the intricacy of the interplay between them, has led many of its practitioners to classify modeling as “intuitive” and “an art (Morrison 1991).” This paper presents some preliminary ideas about a computer program, currently under development, that is an attempt to automate a coherent and useful part of this art. Such a tool is of obvious practical importance in science and engineering: as a corroborator of existing models and designs, as a medium within which to instruct newcomers, and as an intelligent assistant, whose aid allows more time and creative thought to be devoted to other demanding tasks.

The modeling program described here works with ordinary differential equations (ODEs), linear or non-

linear, with multiple variables; it combines powerful mathematical formalisms with domain-specific notions — Kirchhoff’s laws for electronic circuits, for instance, or force balances for mechanics — to allow the type of “custom-generated approximations” (Weld 1992) that are lacking in many existing automated modeling programs. The current incarnation of the program is written in Scheme and Maple; its implementation combines traditional numerical analysis methods, such as simulation and nonlinear regression, with symbolic computation, and will soon incorporate qualitative simulation (Kuipers 1986).

The program builds ODE models from three types of information: mathematical precepts that are true for all ODEs, such as the definition of an equilibrium point; rules that apply in individual domains, such as Kirchhoff’s voltage law for electronic circuits; and highly specific information about the individual target system, entered by a user. Hypotheses may conflict and need not be mutually exclusive, whereas ODE and domain rules are always held to be true. The ODE and domain rules are currently hard-coded in the program; we ultimately envision incorporating a few dozen of the former in total, plus half of a dozen of the latter for each supported domain, and allowing users to augment and modify both types. The specific information about the target system is presented to this modeling tool in a variety of forms and formats:

- the user’s hypotheses about the physics involved
- observations, interpreted and described by the user, symbolically or graphically, in varying degrees of precision
- physical measurements made directly and automatically on the system

Because the physical interface will include both sensors and actuators, this tool will be able to take an *input-output* approach to modeling, another novel and powerful feature.

This research project has two goals; one is immediate and concrete, while the other is far-reaching and less well-defined. The first is to create a program that autonomously constructs mathematical models using the

---

\*Supported by NSF NYI #CCR-9357740

same kinds of inputs that a human expert would use. The second is a first cut at “mental modeling” (Bobrow 1984): to understand what matters in a model and what qualitative and quantitative properties are affected by the requirements of the situation and the knowledge of the user.

## What the Program Does

A model is an abstraction of someone’s understanding of a particular system. It is based on observations, depends critically on the physics background of its architect, incorporates some notion of what quantities matter (*scope*<sup>1</sup>), and adapts to changing scales (*resolution*). Because exact descriptions are often unknown and/or inappropriate, modeling almost always involves abstractions and approximations. *Refinement* increases the *order* of a model, adding terms, sharpening approximations, and shrinking “black box” boundaries. In *simplification*, the dual of refinement, unnecessary terms are removed. Refinement occurs when a model does not match observations and must be improved. Simplification occurs when a model is more complex than necessary. Both operations are governed by the required accuracy and resolution.

Figure 1 shows the connections between the tool described in this paper and the system to be modeled. The program represented by the right-hand block in this figure constructs an ODE model of a target system based on information entered by a user; if the model is to be based on direct observations of a physical system, the program will obtain additional information via sensors, actuators, a hardware I/O channel, and data acquisition software. The *control vector*  $\vec{u}$  represents the actuator inputs. The *observation vector*  $\vec{x}$  is a set of variables that represent the system state — or at least parts of it that are observable and/or interesting.<sup>2</sup>

The program’s output is an ordinary differential equation, of the form  $f(\vec{x}, t) = 0$  (or  $f(\vec{x}, t) = \gamma(t)$ , if the system is driven), that matches the observations to within the prescribed resolution. The program builds this model by mapping the domain rules encoded in its knowledge base onto the user’s hypotheses, checks it against the observations modulo the precision inherent in the specifications, and refines or simplifies it, as necessary, using a collection of techniques that are outlined in the later sections of this paper.

Figure 2 shows an example of how one might instruct the program to build a model of the damped

pendulum. The user first sets up the problem, hypothesizes four different force terms, gives three observations about  $\theta$ , and specifies the required resolutions. The details and implications of each part of this syntax are covered in the rest of this section; the next section describes how the program uses that information to build a model.

The initial problem setup requires several steps. The first specifies the domain and instantiates its associated rules. (*autonomous <force>*) tells the program to apply all *<force>* rules blindly, and not to require the right-hand side of the model to include any particular forcing function. The next two lines identify *<theta>* as a state variable that is a *coordinate* associated with a *point* (the bob). The modeling process works most efficiently if all of the important state variables are identified in the *state-variables* statement, but the program does incorporate a few techniques, addressed in a later section of this paper, that allow it to construct a model, in some cases, even if the user omits important state variables. *Redundant* state variables — ones that play no role in the model — increase the size of the search space but otherwise present no problems.

*Hypotheses* are ODE fragments (single terms, currently) whose variables are (1) elements of the observation vector and (2) special keywords that provide the connection to domain and ODE rules. The *<time>* keyword is common to all domains; keywords specific to the *mechanics* domain are *<force>* and *<energy>*, with the associated domain rules (*point-sum <force> = 0*) and (*point-change <energy> = 0*).<sup>3</sup> The electronics domain uses the keywords *<current>* and *<voltage>*, which work with the domain rules (*point-sum <current> = 0*) and (*loop-sum <voltage> = 0*). Manipulation of these point and loop constructs places some interesting requirements on the internal representations of *coordinates*, as the program must be able to infer connections, cutsets, etc. Note that the concepts of loop and point sums are not only appropriate for these examples, but also generalizable well beyond mechanics or electronics. Finally, its syntax and setup make the program easily extensible to other paradigms (e.g., *volume-change*, etc.) via simple syntax extensions and new rules that tap into those extensions, much as (*point-sum <force> = 0*) works with hypotheses that include the *<force>* keyword.

Multiple hypotheses about a single effect can — *and should* — exist; the program will automatically deter-

<sup>1</sup>The terminology of (Weld 1992) will be adopted here, identified with sans serif font upon first appearance of each term.

<sup>2</sup>Note that the components of  $\vec{x}$  may or may not represent a unique function of the internal system state, and that measuring any given quantity may not be possible (i.e., no appropriate sensor may exist). Observability, controllability, and reachability issues will not be addressed here.

<sup>3</sup>A body-centered inertial reference frame is assumed here, together with coordinates that follow the formulation of classical mechanics (Goldstein 1980), which assigns one coordinate to each degree of freedom, thereby allowing all equations to be written without vectors. A *conjugate momentum* is associated with each coordinate; this concept is useful in symmetry identification and symbolic model matching.

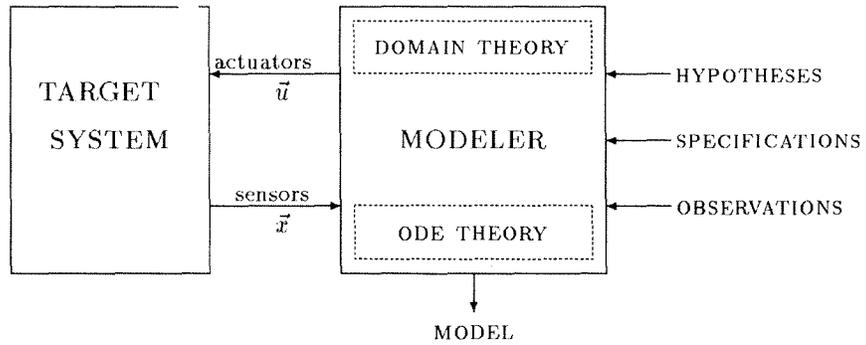
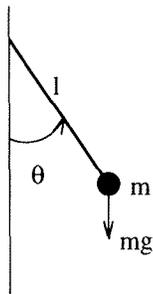


Figure 1: Structure of the modeling tool



```
(find-model
  (domain mechanics)
  (autonomous <force>)
  (state-variables (<theta>))
  (point-coordinate <theta>)
  (hypotheses
    (<force> (* (constant A1 ()) (deriv (deriv <theta>))))
    (<force> (* (constant A2 ()) (sin (* (constant A5 ()) <theta>)))
    (<force> (* (constant A3 ()) (deriv <theta>)))
    (<force> (* (constant A4 ()) (square (deriv <theta>))))
  )
  (observations
    (<theta> (linear (1 0) <theta> (range -.05 .05)))
    (<theta> (asymptote (eqn 0)
      (at <time> *infinity*
        (range 0 *infinity*)))
    (<theta> (numeric (<theta> <time>) ((0 .1234) (.1 .1003) ... )))
  )
  (specifications
    (mesh-width <time> absolute 1e-6 (0 120))
    (mesh-width <theta> absolute 1e-3 (0 (* 2 pi))))
  )
)
```

Figure 2: Instructing the program to model the damped pendulum

mine which one is appropriate. Some other modeling programs, e.g., (Falkenhainer & Forbus 1991), define *groupings* of terms in such situations, such as a set of hypotheses about friction. Mutual exclusivity constraints are then imposed within each group, greatly reducing the complexity of the refinement process. We have chosen *not* to use such groupings, for two reasons: (1) to minimize the restrictions on the models that the program can choose and (2) to minimize the high-level conceptual processing required of the user.

*Observations* describe the behavior of a single element of the observation vector, either in the time domain or in any state-space projection.<sup>4</sup> Unlike hypotheses, observations may not conflict. They have two potential sources: the user and the sensors. User observations may be *descriptive*, *graphical*, or *numeric*. The former use special descriptive keywords, the second are sketches drawn on a computer screen with a mouse, and the third simply specify data points. Descriptive keywords — *concave*, *monotonic*, *oscillation*, *linear with [slope, intercept]*, etc. — closely resemble terms in qualitative physics(QP)(Weld & de Kleer 1990). The user's sketches will be processed — curve fitting, interpolation, recognition of linear regions, and so on — using Maple functions, and the results will be used in the same way as descriptive observations. Observations from the hardware I/O channel will be treated much like graphical observations, but at a higher confidence level. Finally, observations of any form must encode the range in which they are valid; the endpoints of these ranges are akin to QP's *landmarks*.

Observations guide the modeling process in a fundamental way. A model constructed by a human expert matches, minimally, a particular set of observations: the model builder does no more work than necessary to effect the match, and does not try to anticipate extensions or further developments until forced to do so by model failure or requirement escalation. The automatic modeler described here does exactly the same thing: at all times, the program attempts to establish the match with the minimum of work, using information at as high a level as possible to do so. The most important implications of this concern the determination of the coefficients **A1**, **A2**, etc. in figure 2. A descriptive observation often places only qualitative requirements or bounds on those coefficients, while matching a model against a detailed numeric observation usually requires exact coefficient values out to some number of significant figures. Moreover, a single observation, even a qualitative one, can contain information about many different variables — if, for example, it concerns a state variable that appears in all of the terms in the model. Such an observation would significantly focus the model by forcing the eval-

uation of several coefficients in several different parts of the differential equation. Thus neither the number nor the characteristics — qualitative, quantitative, etc. — of observations required for the construction of a successful model are necessarily related to the number of undetermined coefficients or state variables; these requirements are complicated and possibly nonlinear functions of the terms involved. Finally, a higher-level and more obvious implication of the role of observations in the modeling process is that neither a program nor a human expert can construct a model if no observations are given.

A *specification* concerns any function of any number of observation vector elements; it prescribes range and resolution limits for that quantity and specifies whether the resolution is absolute or relative. The **mesh-width** statements in figure 2, for instance, instruct the modeler to impose microsecond and milliradian accuracy over 120 seconds of system evolution. All observation processing is based on the ranges and resolutions given in the specifications; if only the range  $x_2=[3, 5]$  is of interest, asymptotes at  $x_2 = 40$  are immaterial. Landmarks that overlap can be combined, and any effect that occurs on a smaller scale than the one specified can be subsumed into the intervals around it (e.g., a nanosecond glitch during a millisecond-resolution run). It is important to note that specifications implicitly govern the level of abstraction that the modeler enforces: sharpening the **mesh-width** will typically force the modeler to account for lower-level effects and add terms to the ODE, given a fixed set of observations.

This set of inputs was consciously chosen to mimic the information that an expert designer uses when he or she constructs a model. The motivation for this choice is that, in a project whose secondary aim is to understand the human problem-solving process, the program should emulate the process that a human expert would follow, insofar as possible. Moreover, a tool designed for human use should interface smoothly with human skills, reasoning and communication patterns. This choice, and its justification, are specific to this particular project. The debate about whether or not computer problem-solving processes should *in general* emulate their human equivalents has a long and somewhat contentious history, to which we plan no contribution.

## How The Program Works

The program encodes a body of general knowledge about ODEs — how to recognize, locate, and quantify equilibria, basins of attraction, integrability, periodicity, etc. — and domain-specific knowledge like (**point-sum <force> 0**). Together, the ODE and domain rules, operating on the inputs described in the previous section, govern how hypotheses and models are combined, tested, ruled out, augmented, and simplified. Throughout the process, tasks are performed

<sup>4</sup>This program will perform no frequency-domain reasoning.

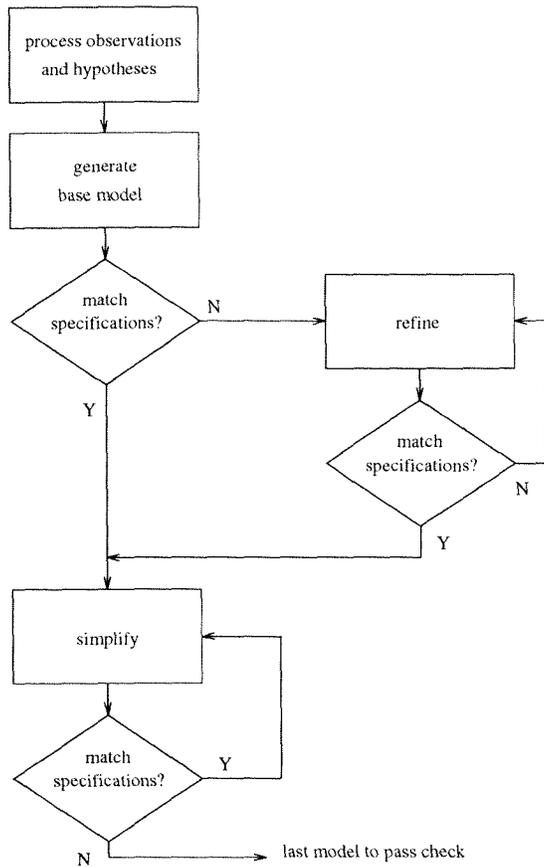


Figure 3: Flowchart of the modeling program

at the highest, most qualitative level possible; symbolic techniques are attempted first and numeric ones only as a last resort. Finally, the modeling process is not passive: the program can use its actuators to manipulate the system and *actively test* the evolving model.

A set of candidates for the program's first attempt at a model is constructed by mapping the domain rules onto the hypotheses — e.g., using  $\langle \text{force} \rangle = 0$  to combine  $\langle \text{force} \rangle = A_1 \ddot{\theta}$  and  $\langle \text{force} \rangle = A_2 \sin A_5 \theta$  into the model  $A_1 \ddot{\theta} + A_2 \sin A_5 \theta = 0$ . Most of the resulting candidate models can quickly be ruled out using rough symbolic rules and techniques; for example, unless a state variable is observed to be constant, the model must include its derivative. The simplest remaining model — the *base model* — is then passed to the check/refine/simplify loop, as schematized in figure 3. This model is checked against the specifications and observations; if it passes, the program then attempts simplification, removing each term in succession and checking the resulting ODE. If the consistency check fails, the program *refines* the ODE, using the domain rules to introduce successive hypotheses from the user's list. As a last resort, the refiner calls upon general ODE theory and purely mechanical methods, such as power-series expansion, to

synthesize terms from scratch. The refined model undergoes one last round of simplification that finds and removes any newly-superfluous terms, and the final product is returned to the user.

This control flow design has a variety of interesting implications. The simplify/refine loop allows the program to move sideways through the search tree of models, recovering from bad choices and making globally good moves that require one locally bad intermediate step. We could also have chosen to loop more than once, which would increase the width of the program's lateral reach through the search space, and, ultimately, allow it to find the *provably minimal model* in that space. However, such a search would have the standard complexity problems (Winston 1992) and a secondary goal of this project is to produce a "good enough" answer in minimal time.

The four blocks of figure 3 — and the mechanics and implications of failure in each — are described in the next four subsections. The following section describes the as yet unimplemented hardware interface and the mechanics of input-output modeling, and is followed by a brief discussion of related work.

## Generating the First Model

The base model generator uses domain rules to combine hypotheses, then ascertains which of those combinations (models) are consistent with the user's observations. The idea here is to produce a preliminary solution to an exponentially complex problem very quickly; to do so, the base-model generator uses *only* symbolic manipulation and stops short of an exhaustive check of all possible combinations of hypotheses. When the answer that it produces is suboptimal or even incorrect — a not-unlikely occurrence because of its quick-and-dirty techniques — the refiner and simplifier act as a safety net, as described in the penultimate paragraph of the previous section.

The hypothesis list is first sorted using a simple-minded complexity metric, discussed later in this paper. The base-model generator then constructs a one-term model from the first hypothesis in the list and checks it against the observations. Symbolic linear algebra, for instance, can be used to establish that the candidate model  $A_1 \ddot{\theta} = 0$  is inconsistent with an observed initial condition  $\theta(0) = 1.2$  and an equilibrium point  $\theta(\infty) = 0$ . If the check fails, the process is repeated using the next entry in the hypothesis list, and so on. If none of the one-term models passes, the base model generator then starts testing two-term combinations.<sup>5</sup> If all two-term models fail, the program proceeds to three-term models, and so on. The first ODE in this succession that is not implausible, given the observations, is then passed to the consistency checker.

<sup>5</sup>Note that *combination* need not imply *sum*; the combination operator is implicit in the operative domain rule.

## Checking Models Against Observations

The consistency checker compares the behavior of an ODE to a set of observations using the specifications as guidelines for how closely to enforce the match. When operating upon descriptive or graphical observations, the consistency checker reasons symbolically when possible, falls back on numerical simulation if necessary, and recognizes different types of behavior (e.g., chaotic, periodic) in both numeric and symbolic results. With numeric observations, the check proceeds directly to the numeric phase.

As mentioned previously, the evolving precision of the model's coefficients follows the procession of the ODE through the phases of the consistency check. If a symbolic check is coefficient-independent, none of the  $A_i$  need be determined. If the check proceeds all the way to a point-by-point comparison of Runge-Kutta output to the data in a **numeric** observation, data-point interpolation and Newton-Raphson are used to compute all undetermined coefficients. Many cases will fall between these two extremes, yielding partially determined coefficients. For example, to match a model against an observed oscillation, the program might symbolically construct a root-locus plot and determine the coefficient values that cause the system poles to cross into the right-half plane — e.g., ( $A_1$  greater-than 0).

The symbolic techniques used to check a model against a qualitative observation are similar to those used by the base model generator — in fact, the latter are a subset of the former. For instance, if the observations included a description of an asymptote, the consistency checker would use symbolic differentiation and L'Hôpital's rule to take the appropriate limit and corroborate the observation. Regions where the system is linear or where a particular polynomial has been specified — or found by a curve-fitting procedure — can be checked using the first few terms of a symbolically constructed Taylor series. The QSIM program (Kuipers 1986) performs qualitative simulation of *qualitative differential equations* (QDEs); given an ODE $\leftrightarrow$ QDE translator (Crawford, Farquhar, & Kuipers 1990), QSIM could be used to do some of these first-cut symbolic consistency checks.

A numeric consistency check of a model entails a comparison of a mechanical integration of the ODE, computed using Runge-Kutta (Press *et al.* 1988), and a set of point-by-point observations. As mentioned above, this comparison is preceded by a numeric fit of model coefficients to numeric data; if it was invoked by a non-numeric observation, the check may entail geometric recognition of high-level behavior patterns — such as chaos, thresholds, or periodic orbits — in the simulation data. One easy — and proven (Bradley 1992) — way to perform this type of geometric recognition is to superimpose a variable-resolution grid over the system's state space, discretize the trajectories into lists of grid squares, and analyze the patterns in those

lists. This ties in well to the input format proposed earlier: specifications and observations can be used to set up the parameters of the mechanical integration and the geometry of the grid, assuring adequate accuracy and resolution in the results, attained with the minimum computation.

If any model fails a check, a failure report is generated; this report contains the model, the violated observation(s), the data (qualitative or quantitative) that caused the violation, perhaps a simple interpretation derived from those data, and *the model's full genealogy* — all branches in the search tree that were traversed during the search. This information could be used by the other program modules — a form of *discrepancy-driven refinement* (Addanki, Cremonini, & Penberthy 1991; Weld 1992) — in a variety of ways: to back intelligently out of blind alleys, to avoid duplicating previously performed checks, or to pick up at some appropriate midpoint in the event of a restart.

## Simplification: Reducing a Model to Lowest Order

The simplifier identifies a candidate term for removal, deletes it, and then invokes the consistency checker on the remaining ODE. If the check fails, the simplifier replaces that term and removes another. This process is repeated until some ( $n-1$ )-term “sub-model” of the  $n$ -term model passes the check, or until all  $n-1$ -models have been tested. In the latter case, the model cannot be further simplified and is passed on as a final result. If any sub-model does pass the check, the entire process is repeated, testing all  $n-2$ -term subsets of that ODE, and so on.<sup>6</sup>

Like the consistency checker, the simplifier takes a symbolic-first approach to identification of candidate terms for removal. It uses some of the same symbolic techniques as do the base model generator and the checker; the main area of overlap is in ODE theory (e.g., asymptote recognition and symbolic differentiation). It also uses some symbolic techniques that would not be useful in the checker, such as pole-zero analysis and conjugate momentum/symmetry recognition. If symbolic reasoning cannot identify a good candidate for removal among the terms in the ODE, the simplifier then turns to less-intelligent techniques, such as removing the “simplest” terms (see below), or terms whose coefficients are much smaller than the others in the model. These choices are somewhat *ad hoc*; it is not at all clear that we gain much from either beyond a defined order of attack. Another approach would be to give the user some leverage by weighting hypotheses according to order of entry. Each of these tactics has advantages and disadvantages; each shifts a different

<sup>6</sup>This raises a subtle issue: sometimes adding or removing *pairs* of terms (or larger groupings) works where successive single-term removal does not: consider coriolis and centripetal forces, which must appear together in a rotating frame. We plan to investigate this in the near future.

amount of the burden of rigor from the simplifier to the checker.

Ordering terms or hypotheses logically is difficult because no satisfying simplicity metric exists. The modeling literature contains no good solutions, and this author has no better suggestions. The somewhat-arbitrary metric that we use is based on number and complexity of terms and derivatives:  $x + 3 \triangleright x$ ;  $x^n \triangleright x^{n-1}$ ;  $\dot{x} \triangleright x$ ; etc., where  $\triangleright$  denotes “more complex than.” This is unsatisfying for a variety of reasons, not the least of which is how to resolve ties between equations like  $x + x^2$  and  $x^3 + 4$ . One of the seminal papers in qualitative modeling uses a similar metric (Falkenhainer & Forbus 1991) — and gives similar disclaimers about its unsatisfying nature. A more-recent paper (Nayak 1992) defines model simplicity heuristically, terming a model that is “more approximate” or that “uses fewer phenomena” as more simple.

### Refinement: Adding Terms to a Model

The refiner uses ODE and domain rules, observations, and hypotheses to add terms to a model, keeping track of previous attempts in order to avoid duplication of effort. It first draws upon the user’s hypotheses, then synthesizes ODE terms from scratch using power-series expansions if no successful hypothesis-based model can be found. These power-series methods are a powerful safety net: a one- or two-term expansion in  $\theta$  and  $\dot{\theta}$  would regenerate every form of friction found in freshman physics texts — and a few more besides. Moreover, these methods actually allow the program to create new state variables — an important feature if the observation vector is smaller than the true state vector. Like the other program modules, the refiner uses physics concepts before less-intuitive mechanical techniques and follows symbolic reasoning with more-expensive numeric approaches.

The refiner’s first task is to prune the sorted hypothesis list, removing any that were used in the base model. The first hypothesis in the sorted list is then added to the base model and the checker is called on the new ODE. If that model fails, the refiner removes the previously introduced term and successively tries the rest of the hypotheses in the ordering, one at a time. If no one-term addition causes the model to match the observations, the refiner then tries *pairs* of terms, and so on. The first successful model is returned as the refiner’s result. If the refiner exhausts the list of hypotheses before finding an adequate model, the power series methods outlined in the previous paragraph are invoked. These expansions are subject to a user-specified limit  $p$ ; if no match has been established after the refiner tries the  $p^{\text{th}}$  term in the expansion, the program admits failure and requests additional hypotheses from the user.

Simplicity metric difficulties, discussed at the end of the previous section, also affect the refiner. The metric proposed here may not be rigorous, but it at least

provides a starting point. A much more intelligent approach would be to sort the hypotheses according to their behavior, using the same symbolic and numeric techniques used in the checker to make a rough comparison to the observations, all modulo specification precision. For instance, if an observed voltage converged to zero, a hypothesis that caused that value to diverge to infinity should be lower on the list of things to try. Note that such a hypothesis should *not* be removed from the list altogether; it may well appear in the ultimate model, moderated by another term (e.g.,  $x - \frac{x^3}{6} + \frac{x^5}{120} \dots$ ). This *sorting by behavior assessment* may be computationally expensive enough to negate its advantages. Moreover, serious problems can arise when one attempts to extend conclusions about the parts of a system to conclusions about its whole (e.g., closed-loop versus open-loop dynamics in a feedback system). However, the advantages of ruling out some of the hypotheses may dominate, making this approach attractive. Another possibly useful alternative would be, again, to use the entry order verbatim. A half-dozen informal interviews have shown that most users do indeed list physics hypotheses in order of perceived simplicity,<sup>7</sup> so this may not be a bad idea.

The two automatic term-synthesis techniques to be used here are both derived from power-series expansion. *Canonical perturbation theory* (Goldstein 1980, chapter 11) creates new parameters and *Frobenius’s method* (Morrison 1991, page 187) creates new state variables, each via expansion in the appropriate quantity. One could also synthesize new state variables using delay coordinates (Gouesbet & Maquet 1992), but doing so would vastly complicate the symbolic algebra. Because the lower-order terms of power-series expansions are likely to match (up to a coefficient) simple physics hypotheses, the modeler will use simple symbolic techniques to avoid duplication of effort, eliminating any power series expansion terms that also appear in a user hypothesis.

The combination of all of this machinery allows the program to preferentially check the user’s hypotheses, then synthesize and explore new state variables and parameters if necessary, possibly concocting model terms that do not resemble any of the given hypotheses. It can even construct models in the absence of any hypotheses, if the underlying physics admits a power-series description. The ODE and domain rules, together with symbolic reasoning and qualitative behavior descriptions, will allow this tool to reason at a very high level indeed. For example, if a system was observed to be chaotic, but only one state variable was specified, the power series methods would be invoked automatically to introduce a new state variable before the program even attempted to build a base model.<sup>8</sup>

<sup>7</sup>The exception is serious experts, who sometimes add the simplest terms only as an afterthought.

<sup>8</sup>Three state-space dimensions are a necessary condition

This automatic modeling tool will be able to adapt the scope of the model on the fly, inferring “state variables” that are internal (e.g., voltages inside a black box) or that were omitted by oversight (e.g., low-amplitude, low-frequency vibration of a bench by a lab’s HVAC system). These quantities are diffeomorphically related to the true system state, so they can be used to draw some rigorous conclusions. Again, formal observer theory is not a goal of this project, so the proposed program will not be able to solve this problem in all situations, and this research will include only limited study of the relationships between inferred and true state variables.

### Incorporating Physical Measurements

An important feature of the physical link between modeler and system is that data will be able to flow across it in *both* directions, making the modeling process an *active* one — in all parts of the procedure described in the previous three sections. This has a number of important implications. Among other things, the program could autonomously exercise the target system in order to verify or augment the user’s observations; it could even construct and check observations that transcend its user’s knowledge of physics. For instance, the boundaries of the basins of attraction of the ODE and the target system could be compared using different-energy kicks, even if the user knew nothing about dynamics and basin structure. Obviously, this presents some dangers: if the program is free to use the sensors, the actuators, and the full breadth of its own (significant) knowledge base, the only controls on the sophistication — and intricacy — of the resulting model are the resolution (**mesh-width**) and the expansion limit  $p$ . To address this problem, we have introduced another parameter, **max-level**, that allows the user explicit control over the number of terms that the program may use in the model.

Physical measurements will be treated exactly like graphical user observations, from the program’s point of view, with one important distinction: in the event of a conflict, the former will be trusted over the latter. Measurements will be processed and translated into the syntax of descriptive user observations and then used in exactly the same ways — as targets for symbolic comparisons. The measurement processing algorithms that extract this qualitative information from the sensor data will essentially duplicate the approach to behavior recognition outlined in the discussion on checking models against observations.

With the hardware data channel in place, the program could potentially be used not only to model a physical system, but also to debug designs — and even to validate and verify devices ostensibly constructed according to a known design. For example, if a particular 2.5K $\Omega$ , 1/4 Watt resistor burned up every time a

for chaos.

device was turned on, one could place probes across its terminals, set up an observation that specifies “voltage over 25 volts,” and observe discrepancies between the resulting model and what was intended. Needless to say, one would want to isolate the sensors from potential damage during such an experiment.

### Related Work

Modeling research spans many fields, from the cognitive science-related branch of AI (Langley *et al.* 1987) through dynamic systems (Gershensfeld & Weigend 1993) and control theory (Astrom & Eykhoff 1971) to qualitative reasoning (QR) (Bobrow 1985). Some of the earliest QR/modeling work (Falkenhainer & Forbus 1991) built upon a fixed base of hypotheses, instantiated only those that were appropriate to answer a given query, and chose between them with a truth maintenance system. The GoM approach (Adanki, Cremonini, & Penberthy 1989) is similar in many regards to (Falkenhainer & Forbus 1991), but represents the space of possible models as a directed graph of models where edges between nodes (models) are approximations.<sup>9</sup> Another approach (Weld 1992) adapts models to problems using *model sensitivity analysis*, which formalizes and exploits the effects of parameter changes in the construction of the model. The combinatorial explosion involved in limit checking (e.g., the pendulum’s asymptote to  $\theta = 0$ ) can be mitigated (Kuipers 1987) by decomposing and abstracting time into a hierarchy of scales. Rules for determining the behavior of a composite device can be derived from the models of its constituent components (Kleer & Brown 1984).

This is an extremely active research area; many good papers by many other groups, as well as many other papers by the cited groups, have appeared in the past four years. The state of the art in this field is particularly well-summarized in a recent article by Weld (Weld 1992), which is also the source of much of the terminology used in this document. Concepts common to this paper and the bulk of the QR/modeling literature include avoidance of unnecessary terms, model refinement driven by failure at a “lower” modeling level, and reasoning that proceeds at as a high level of abstraction as possible.

### Status and Discussion

This paper is a description of the first stages of work-in-progress on a highly ambitious task. To date, we have only built a small, but functional — and hopefully representative — subset of the program.<sup>10</sup> This subset incorporates a few instances of each technique, providing a quick check of the whole symbolic/numeric

<sup>9</sup>(Falkenhainer & Forbus 1991)’s propositional reasoning also uses a digraph, but it is used implicitly and constructed somewhat differently.

<sup>10</sup>hence the mix of verb tenses in this paper.

paradigm and the overall control flow (hierarchy of hypotheses, domain rules and ODE rules; control flow between modules, and so on). Most importantly, it has allowed us to test and refine the syntax and the use of the various types of user inputs.

This first version, constructed<sup>11</sup> with the aid of Reinhard Stolle, allows one-term hypotheses involving the keywords `<force>` and `<time>`, sorts them according to the simplistic metric proposed in a previous section, incorporates one rule (`(point-sum <force> 0)`) in one domain (`mechanics`), has a reduced vocabulary of qualitative terms (`above`, `below`) and a correspondingly small repertoire of symbolic and numeric techniques to verify them, and has been tested on two-dimensional systems with numeric observations, such as figure 2 with all but the `numeric` observation omitted. Even this limited exercise has turned up some interesting problems. Determining the coefficients for the numerical integrator run required linear interpolation of the data points to produce  $m$  nonlinear equations in the  $m$  unknown constants, which were then solved with Newton-Raphson. In general, this problem — known as *parameter estimation* — is solved with much more sophisticated techniques like Kalman filtering (Kalman 1960), and is the topic of a rich body of literature (Sorenson 1985). Our simplistic solution spawned a second problem: a failure to match a Runge-Kutta run against an observation could mean either that the model was inadequate or that our parameter estimation algorithm was inaccurate. We solved this by imposing an artificially high accuracy on the latter, but this is not a good general solution. The `mechanics` domain syntax has grown far closer to the Lagrangian formulation than we had originally envisioned because its formalized structure is so useful (for example, the coordinate/momentum pair for each degree of freedom, related by a derivative, that suffice to describe the system completely and contain *symbolically extractable* information about symmetries and conserved quantities as well).

Figure 4 depicts a modeling run on a simplified version of the pendulum of figure 2, shown at the top of figure 4, that includes only two hypotheses and one numeric observation. The base-model generator is bypassed in this run because the only observation is numeric. The refiner rules out all one-term models via symbolic methods, then uses the force-balance rule to map the two hypotheses into a two-term candidate model, upon which the checker is invoked. The check proceeds directly to the numeric phase, the parameter estimator computes the coefficients, and the point-by-point comparison succeeds. Since both “sub-models” of this model have already failed the check, this model and its coefficients are returned as the final result. We have also experimented with adding white noise to the numeric data and changing the resolution in the spec-

ifications, with predictable results. The program still finds the right model if the added noise is small compared to the resolution, and fails when it is not, as the repertoire of hypotheses does not allow it to model the noise.

Later versions will use the hardware I/O channel, incorporate more symbolic and qualitative methods, be tested with underspecified modeling tasks and sketchier observations, and use more domain rules in multiple domains that operate on multiple, heterogeneous keywords. We expect the first and the last to be the hardest and most interesting of these tasks, particularly since our aim is emphatically not to build a tool that is tuned for one particular application domain. Beyond that, further development will entail the exploration of different implementation paradigms and techniques: sorting hypotheses by behavior or by order of entry, allowing multi-term hypotheses, etc.

## Summary

The nascent program described here uses general mathematical theory as a foundation, adds concise and powerful domain-specific rules, and funnels user-specified hypotheses through general ODE theory and domain-specific rules to generate “appropriate” models — models that are well-matched to the task at hand. It exploits intelligent, high-level techniques like symbolic manipulation from the outset, carrying them as far as possible through each phase of the work, and using them to make the type of quick, overall assessment that a human expert uses in the first stages of model-building. The program then resorts to lower-level, less-intuitive methods to complete the analysis and synthesis processes. The chosen set of inputs and the way that they are used closely resembles the process one finds documented on any designer’s scratch paper: parts of equations, rough sketches, scratched-out forays up analytical blind alleys, and an overall progression of ideas and abstractions from simpler to more complex.

This mixture of exact and approximate techniques and precise and heuristic knowledge is powerful, but has one important disadvantage: it makes formal, rigorous analysis of the necessary conditions for modeling success very difficult to come by. This is not a pure mathematician’s tool, though pure mathematics certainly contributed to the plan for its implementation. It is a design tool that is intended to be *useful*: to find a “good enough” solution with minimum time and effort. This tradeoff motivated many of the design choices described here, notably the control flow between the simplifier and the refiner.

Though the examples here are drawn from the domains of mechanics and electronics, many other potential application areas exist. The general framework described here has been designed to smoothly accommodate rules from many domains, and the program itself has been written to be easily extensible. It should

<sup>11</sup>both code and ideas

The invocation:

```
(find-model
  (domain mechanics)
  (autonomous <force>)
  (state-variables (<theta>))
  (point-coordinate <theta>)
  (hypotheses
    (<force> (* (constant a ()) (deriv (deriv <theta>))))
    (<force> (* (constant b ()) (sin <theta>))))
  (observations
    (<theta> numeric (<theta> <time>) ((0 .1234) (.1 .1003) ... )))
  (specifications
    (mesh-width <time> absolute 1e-6 (0 120))
    (mesh-width <theta> absolute 1e-3 (0 (* 2 pi)))))
```

The transcript:

```
Trying to find model for
hypotheses = ((* a (deriv (deriv <theta>)))
              (* b (sin <theta>)))
with max level = 2.

Trying to find model at level 2...
Checking model
(model ((= (+ (* (constant b ()) (sin <theta>))
              (* (constant a ()) (deriv (deriv <theta>)))) 0))).
Checking model
(model ((= (+ (* (constant b ()) (sin <theta>))
              (* (constant a ()) (deriv (deriv <theta>)))) 0)))
numerically.
Checking
((= (deriv <theta>) d<theta>)
 (= (deriv d<theta>) / (- 0 (* 3. (sin <theta>))) 2.))
against data.

((model
  ((=
    (+ (* (constant b ()) (sin <theta>))
      (* (constant a ()) (deriv (deriv <theta>))))
    0)))
  ((a 2.) (b 3.)))
```

Figure 4: A modeling run on the damped pendulum

be obvious that the definitions and solutions presented here are preliminary and will necessarily undergo much development and refinement as this program is developed, but the preliminary results have been encouraging — and some of the early problems have been subtle, rich, and rewarding to think about and to solve.

## References

- Addanki, S., Cremonini, R., and Penberthy, J. S. 1989. Reasoning about assumptions in graphs of models. In *Proceedings IJCAI-89*. Detroit, MI.
- Addanki, S., Cremonini, R., and Penberthy, J. S. 1991. Graphs of models. *Artificial Intelligence* 51:145–178.
- Astrom, K. J., and Eykhoff, P. 1971. System identification — a survey. *Automatica* 7:123–167.
- Bobrow, D. G. 1984. Qualitative reasoning about physical systems: An introduction. *Artificial Intelligence* 24:1–5.
- Bobrow, D. G., ed. 1985. *Qualitative Reasoning about Physical Systems*. Cambridge MA: M.I.T. Press.
- Bradley, E. 1992. *Taming Chaotic Circuits*. Ph.D. Dissertation, M.I.T.
- Crawford, J. M., Farquhar, A., and Kuipers, B. J. 1990. QPC: a compiler from physical models into qualitative differential equations. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-90)*.
- Falkenhainer, B., and Forbus, K. D. 1991. Compositional modeling: Finding the right model for the job. *Artificial Intelligence* 51:95–143.
- Gershenfeld, N. S., and Weigend, A. S. 1993. The future of time series. In *Time Series Prediction: Forecasting the Future and Understanding the Past*. Santa Fe, NM: Santa Fe Institute Studies in the Sciences of Complexity.
- Goldstein, H. 1980. *Classical Mechanics*. Reading MA: Addison Wesley.
- Gouesbet, G., and Maquet, J. 1992. Construction of phenomenological models from numerical scalar time series. *Physica D* 58:202–215.
- Kalman, R. E. 1960. A new approach to filtering and prediction problems. *J. Basic Eng.* 82D:35–45.
- Kleer, J. D., and Brown, J. S. 1984. A qualitative physics based on confluences. *Artificial Intelligence* 24:7–83.
- Kuipers, B. J. 1986. Qualitative simulation. *Artificial Intelligence* 29:289–338.
- Kuipers, B. J. 1987. Abstraction by time scale in qualitative simulation. In *Proceedings AAAI-87*, 621–625. Seattle, WA.
- Langley, P., Simon, H. A., Bradshaw, G. L., and Zytkow, J. M., eds. 1987. *Scientific Discovery: Computational Explorations of the Creative Process*. Cambridge, MA: MIT Press.
- Morrison, F. 1991. *The Art of Modeling Dynamic Systems*. New York: Wiley.
- Nayak, P. 1992. Causal approximations. In *Proceedings AAAI-92*.
- Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. 1988. *Numerical Recipes: The Art of Scientific Computing*. Cambridge U.K.: Cambridge University Press.
- Sorenson, H. W. 1985. *Kalman Filtering: Theory and Application*. IEEE Press.
- Weld, D. S., and de Kleer, J., eds. 1990. *Readings in Qualitative Reasoning About Physical Systems*. San Mateo CA: Morgan Kaufman.
- Weld, D. S. 1992. Reasoning about model accuracy. *Artificial Intelligence* 56:255–300.
- Winston, P. H. 1992. *Artificial Intelligence*. Redwood City CA: Addison Wesley. Third Edition.

# Modelling the Influence of Non-Changing Quantities

Bert Bredeweg  
Kees de Koning  
Cis Schut

Department of Social Science Informatics (S.W.I.)  
University of Amsterdam, Roetersstraat 15  
1018 WB Amsterdam (The Netherlands)  
Telephone: +31-20-525 6788, Telefax: +31-20-525 6896  
E-mail: bert@swi.psy.uva.nl

April 24, 1994

## Abstract

*In qualitative modelling, information is lost by abstracting from quantitative formulae. We show that when the behaviour of two similar systems is compared, non-changing quantities from these formulae can have a significant influence on the qualitative prediction. We propose the addition of a new ontological primitive for representing these influences in qualitative models, and provide a calculus for exploiting this primitive in the reasoning process. Augmentation with the new primitive enhances the power of the qualitative simulator, resulting in a more appropriate prediction of behaviour, and also improves the explanation capacities of the model. The latter feature is of major importance for tutoring systems using qualitative reasoning.*

# 1 Introduction

A behaviour description generated by qualitative simulation consists of a set of states modelling qualitatively distinct behaviours of the simulated device. The notion of change is the key concept for generating such a description. In qualitative reasoning basically, two ways of dealing with changes have been investigated: values and (in-)equalities. In particular in the early days of qualitative reasoning research, distinct states of behaviour were defined as having different values for quantities and/or different values for their derivatives. Finding a state transition implied searching for a quantity whose derivative was *plus* or *minus* so that it would adopt a higher or lower value in its quantity space [3, 6]. In later publications, reasoning with (in-)equalities became more important (see for instance [9]). Looking for state transitions became more complex and included reasoning steps such as

if  $A = B$  and  $\Delta A = plus$  and ( $\Delta B = 0$  or  $\Delta B = minus$ ) then  $A > B$ .

In the approach we are using, both techniques for reasoning with changes can be employed (cf. [1]). Still we had severe trouble in modelling the behaviour of the balance system (see Figure 1)<sup>1</sup>. Certain changes in the behaviour of this system cannot be represented adequately. This

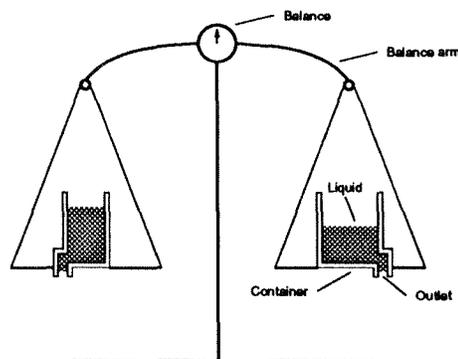


Figure 1: A Balance Problem

behaviour depends on 'influences' of quantities that don't change themselves, but still have a significant effect on how the behaviour of the system evolves. For example, if the heights of the water columns in both containers are equal (and therefore also the pressures at the bottom and thus the flow rates) then the containers will lose water with equal rates. If the widths of the containers are also equal, both containers will be empty at the same time and the height of the water columns will stay equal while emptying. However, a problem occurs if the widths of the containers are unequal. In that case, the container with the smallest water column will become empty first. In order to reach that state of behaviour the heights of the columns have to become unequal first. The height of smallest column will become lower (see Figure 2, transition from State 3 to State 4). In other words, the change in height depends, among others, on the width of the column. The width has a significant effect on how the behaviour changes, even though it does not change itself. This feature cannot be modelled adequately by current

---

<sup>1</sup>The problem is to predict the behaviour of balances with containers on each balance arm. Both containers are assumed to be equal in weight. Depending on the difference in the mass of the liquid contained by the containers, one balance arm may be heavier than the other. Therefore, after releasing it from the starting position the balance may change its position. Through outlets near the bottom of the containers the liquid gradually flows out of the containers. Depending on the pressure at the bottom, the flow rates may be different. As a result, the balance may move to a different position, because the difference in weight between the two balance arms changes. Eventually, when both containers are empty, the balance will reach an equilibrium.

techniques for qualitative reasoning that use influences and proportionalities as the basis for determining changes, because the ‘influence’ of the non-changing quantity is not captured by these dependencies.

We present a technique that can be used for modelling the influence of non-changing quantities. After providing some background on our research in Section 2, the problem we are facing is explained in more detail (Section 3). In Section 4, we discuss how the problem can be solved by using quantitative mathematics. Section 5 presents a reformulation of the mathematically oriented solution in qualitative terms, such that it can be added to current approaches to qualitative reasoning. Finally, in Section 6 we briefly summarise our results and discuss the consequences of them for qualitative reasoning.

## 2 Teaching Qualitative Reasoning

It is widely recognised that for solving physics problems a careful qualitative analysis of the problem situation is essential (cf. [8, 2, 7, 5]). A typical difference in the problem solving behaviour of experts and novices is the large amount of time an expert spends on the qualitative analysis. The novice is more likely to skip this phase and start computing formulae right away. One of the problems for physics tutors is to make sure that novices start with a qualitative analysis before they select mathematical formulae. In other words, teaching qualitative reasoning is an essential step in teaching physics.

The *Balance Tutor* [4] is our first realisation of a teaching environment that coaches students in analysing the behaviour of physical devices using qualitative knowledge. The teaching environment uses an on-line qualitative simulator (GARP) that implements a problem solving capacity similar to the traditional approaches to qualitative reasoning [1]. For teaching qualitative reasoning, two requirements must be fulfilled by the simulator:

- The simulator must predict the behaviours that are manifested by the real system. This means that the simulator may not neglect any states of behaviour, but also that it may not predict any spurious (non-existing) behaviours.
- The knowledge used by the simulator for predicting the states of behaviour should facilitate explanation of why certain behaviours occur whereas others do not. This implies that the simulator must be able to provide a causal account of why the behaviour evolves in a certain direction.

The knowledge representation we use does not completely satisfy these two constraints. The shortcomings are not just weaknesses in our simulator, but result from lacking reasoning capabilities in current qualitative reasoning techniques. In the next section we will first elaborate on the knowledge representation that we use and then explain the problem of dealing with non-changing quantities in more detail.

## 3 Knowledge Representation

For reasoning about the balance system we use the notion of processes as the prime cause of changes (cf. [6]). The (direct) changes imposed on a system by influences are propagated by proportionalities (indirect changes). In addition to these two causal relations, corresponding values are defined between magnitudes of specific quantities. Non-causal dependencies ( $<$ ,  $\leq$ ,  $=$ ,  $\geq$ ,  $>$ ) may be used for representing inequalities between quantities. Note that inequalities are not the same as correspondences; two quantities can be equal but still have non-corresponding qualitative values, or vice versa.

Similar to the component oriented approach [3] and the process oriented approach [6], GARP uses the notion of *model fragments* for modelling the behaviour of some real-world system. All model fragments have associated with them a set of conditions under which they are applicable, and a set of consequences that are given once their conditions hold. Typically, conditions specify required objects, inequalities between quantities, and/or specific values that quantities must have. Consequences, on the other hand, usually introduce influences and proportionalities between quantities, although inequalities can also be specified as a consequence.

In order to reason about the behaviour of the balance system, model fragments are needed for the containers containing liquid, the liquid flow out of the containers, the position of the balance (depending on the mass difference between left and right), and the movement of the balance (depending on the flow difference between left and right). In the *scenario*, or input system, the balance system is, apart from its physical structure, defined by the quantities *Height* ( $H_L, H_R$ ), *Volume* ( $V_L, V_R$ ), *Width* ( $W_L, W_R$ ), and *Flow* ( $F_L, F_R$ ). For purposes of clarity, we will not discuss the fully corresponding quantities *Amount* and *Mass* (corresponding with *Volume*), *Pressure* (corresponding with *Height*), *Position* (corresponding with  $V_L - V_R$ ), and *Movement* (corresponding with  $F_L - F_R$ ). Furthermore, we assume that the containers are rectangular, and for the moment ignore the depths of the containers by assuming that they are equal.

In the example shown in Figure 2, the following inequalities initially hold:  $H_L > H_R$ ,  $V_L = V_R$  and  $W_L < W_R$ . All quantities have the initial value *plus*. When presenting this

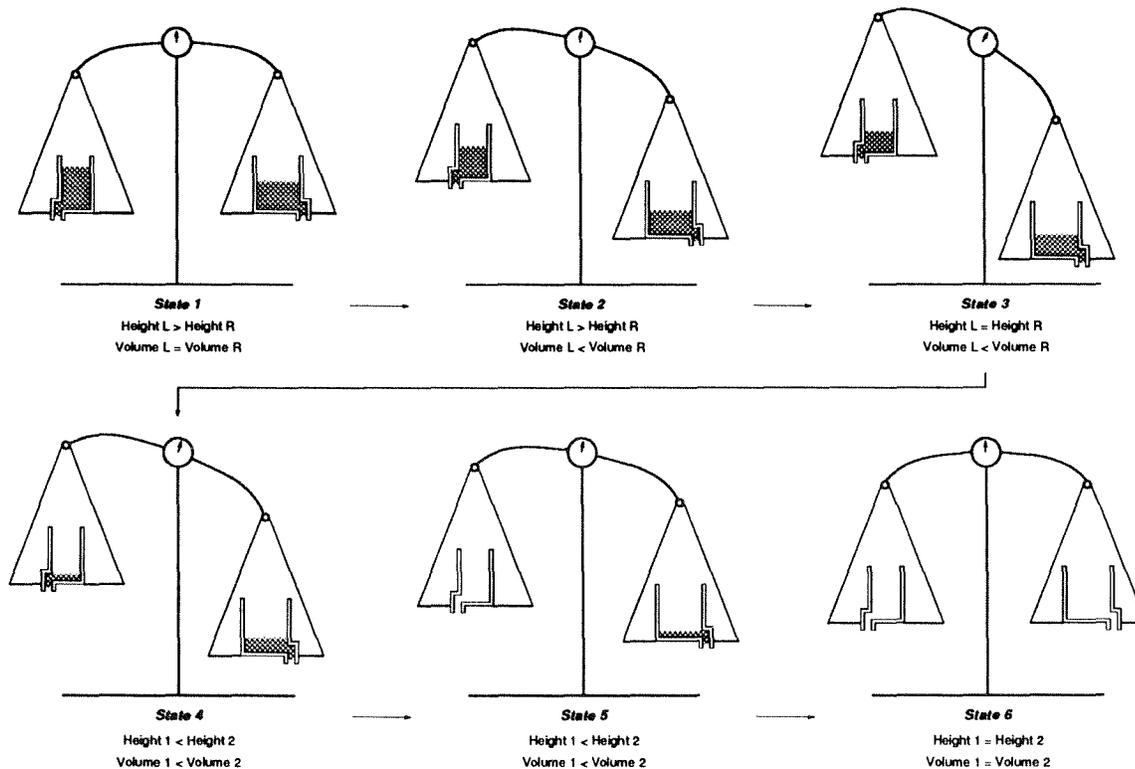


Figure 2: Behavioural Description of a Balance Problem

system to the qualitative simulator, a causal structure is produced for the different states of behaviour, as shown in Figure 3. The height of the column determines the flow rate. The flow rate influences the volume. Changes in the volume propagate into changes of the height. Changes in the width also propagate into changes in height. This causal structure is applicable

for the liquid columns on both sides of the balance. However, this model of the balance system

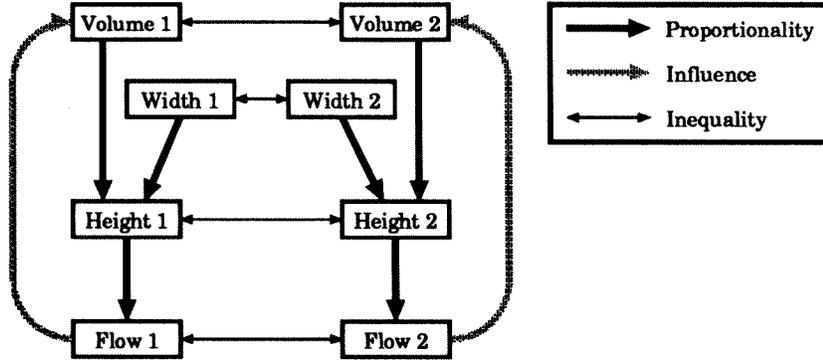


Figure 3: Causal Dependencies and Inequalities for the Balance Problem

does not allow for the derivation of all required states as shown in Figure 2. In particular the transition from State 2 to State 3 can not be derived.

The transition from State 1 to State 2 is derived because  $H_L > H_R$  yields  $F_L > F_R$ , and consequently  $\delta V_L > \delta V_R$ . The volumes decrease with different rates, and will thus become unequal. The decrease rates of the heights, however, can not be derived. This is caused by the way in which the effect of the width on the liquid is represented. The height of the liquid is represented as proportional to the volume (which is decreasing) as well as to the width (which is constant). Because the relation between *Volume*, *Height* and *Width* is divided into two proportionalities, the effect of *Width* on the derivative of *Height* is lost.

The next section discusses this problem in a mathematical manner, providing the basis for introducing a new primitive relation type and its calculus.

## 4 Mathematical Solution

State transitions are induced by (in)equalities between the volumes  $V$  on each side of the balance system and between the heights  $H$  of the liquid columns. The relation between  $V$  and  $H$  is mediated by the area of the container. Since we assume rectangular containers with equal depths, only the width  $W$  is relevant:

$$V = W \times H \quad (1)$$

The flow rate  $F$  is defined as the decrease of the volume per second, thus  $F = -\frac{d}{dt}V$ . However, to avoid the confusion caused by the negative derivative, we simplify this formula to

$$F \sim \delta V \quad (2)$$

The relation between the flow and the height of the liquid column is

$$F = \alpha\sqrt{H}$$

where  $\alpha$  is determined by various quantities that are assumed to be equal for both liquids, like the viscosity, the area of the outlet, and the gravitational constant. We therefore omit  $\alpha$ , yielding

$$F \sim \sqrt{H} \quad (3)$$

The change of the volume is equal to the width of the liquid column times the change of its height.

$$\delta V = W \times \delta H$$

This can be rewritten as

$$\delta H = \frac{\delta V}{W} \quad (4)$$

These equations allow us to infer the state sequence that describes the behaviour of the balance. In each state, Equation 3 is used to deduce the ratio of the flow rates, and Equation 2 to deduce the ratio of the volume derivatives. Equation 4 is used to derive the ratio of the derivatives of the heights. The initial state is characterised by the inequalities  $V_L = V_R$ ,  $H_L > H_R$ , and  $W_L < W_R$ . We describe the first three state transitions in detail. (see also Figure 2).

**State 1 → State 2** We use  $H_L > H_R$  to deduce  $F_L > F_R$  (Equation 3), and consequently  $\delta V_L > \delta V_R$  (Equation 2). Because the decreases in the volumes are unequal, their ratio changes from  $V_L = V_R$  to  $V_L < V_R$  in the next state.

**State 2 → State 3** Since  $W_L < W_R$  and  $\delta V_L > \delta V_R$ , we derive that  $\delta H_L > \delta H_R$ , using Equation 4). State 2 therefore terminates by the heights becoming equal. Another possibility would be a termination in which the left container becomes empty: because  $\delta V_L > \delta V_R$ , and  $V_L < V_R$ , we can derive that  $V_L$  may become zero. However, this requires  $H_L < H_R$ , and no direct transition is possible from  $H_L > H_R$  to  $H_L < H_R$  (the intermediate ratio  $H_L = H_R$  can not be skipped).

**State 3 → State 4** In State 3,  $H_L = H_R$  holds, so  $F_L = F_R$  (Equation 3), and thus  $\delta V_L = \delta V_R$  (Equation 2). Because  $W_L < W_R$  also holds,  $\delta H_L > \delta H_R$  is derived using Equation 4, causing a transition to State 4.

Apparently, there are no problems in deriving the behaviour of the balance in the mathematical model. However, as observed in Section 3, the transition from State 2 to State 3 cannot be derived in the qualitative model, because some of the required information is lost in the abstraction process. A solution for this problem is presented in the next section.

## 5 Qualitative Solution

The mathematical solution presented in the previous section reveals the limitations of qualitative reasoning with respect to the influences of non-changing quantities. In quantitative formulae, the relations between different variables as well as constants can easily be represented. But constant values are abstracted from in building qualitative models, because they are not actively involved in the causal quantity interactions within the system, and hence do not directly influence the qualitative values of quantities. The ‘hidden’ constant in proportionalities and correspondences influences the *quantitative* value of the resulting quantity, however, and may become relevant in inequality reasoning, where the quantitative values of quantities are compared.

### 5.1 Representation

Consider again the balance example, and the representation of the quantities as given in Figure 3. The reason why the quantitative model is capable of predicting the change in the height equation while the qualitative model is not, can be attributed to the fact that in the latter the relation between *Volume* and *Height* is independent from the relation between *Volume* and

*Width*. Moreover, the correspondence between the width and the height will not induce any change in the height because the width has a constant value. In the qualitative model, information is lost on the interrelation between the three quantities *Width*, *Volume*, and *Height*. This loss of information can be compensated by explicitly representing that *Volume* is the product of *Width* and *Height*.

One option is to define a qualitative version of the (quantitative) multiplication relation for representing formulae in qualitative models. The formula  $V = W \times H$  can now straightforwardly be represented as `mult(V,W,H)`<sup>2</sup>. In order to support larger multiplications (for instance, when the depth ( $D$ ) is taken into account, the formula expands to  $V = W \times H \times D$ ), the representation can be defined to allow embedded multiplication relations, or intermediate variables can be added. Incorporating the depth will then result in respectively `mult(V,mult(W,H),D)` or `mult(V,X,D)`, `mult(X,W,H)`. We do not elaborate upon the exact representation of larger multiplications here.

Another option is to enrich the ontology for qualitative reasoning with a new primitive that we will call *relation modification*. This enables us to express explicitly that the value of a non-changing quantity affects another relation. Considering the balance example again, we can now express that the width affects the influence of the volume on the height. This may be represented as

```
prop_pos(Prop1, Volume, Height)
mod_neg(Width, Prop1)
```

The positive proportionality relation `Prop1` between *Volume* and *Height* states that an increase (decrease) in the volume causes an increase (decrease) in the height, whereas *Width* acts as a negative modifier of `Prop1`: the larger *Width*, the smaller the influence of *Volume* on *Height*. Representing more than one modifier for a proportionality is now easily represented by defining more than one `mod_pos` or `mod_neg` relation for the same proportionality.

For the purpose of incorporating the influence of non-changing quantities, there is no principal difference between these two representations. A formula  $A = C \times B$ , where  $C$  is a constant, can equivalently be represented as `mult(A,C,B)` or as `prop_pos(Prop1, B, A)`, `mod_neg(C, Prop1)`. The difference is in the explicit notion of causality present in the latter representation, as is discussed in Section 6. In the remainder of this section, we adopt the latter representation for presenting the calculus.

## 5.2 Calculus

In designing a calculus for dealing with relation modifiers, it is important to keep in mind that these modifiers are only relevant in situations where two similar processes are compared. As long as a single causal sequence of quantity dependencies is considered (as is the case with causal prediction of behaviour for a solitary system), these modifiers can be omitted. This is exactly what happens in the current definition of a proportionality:  $A = \alpha \times B$  is abstracted to  $A \propto B$ . When comparing two similar processes, however, it is not always possible to abstract from constant values. As soon as corresponding constant values are different, they may effect the relation between other corresponding quantities in the system. Thus, our calculus assumes the existence of two similar processes. In qualitative reasoning terms, similar processes are modelled by (sets of) instances of the same generic model fragments. This fact is exploited explicitly in our calculus.

---

<sup>2</sup>To stress the fact that multiplications are only relevant for (quantitative) inequality reasoning, the alternative representation `equal(V,mult(W,H))` could be used.

We start with the simplest case, in which there is a single proportionality with a single modifier in each of the systems. Let  $A_1, B_1, \dots$  be the quantities of one system, and  $A_2, B_2, \dots$  the corresponding quantities in the other system. Let the following relations hold<sup>3</sup>:

```
prop_pos(Prop1, A1, C1)
mod_neg(B1, Prop1)
prop_pos(Prop2, A2, C2)
mod_neg(B2, Prop2)
```

Then the calculus presented in Table 1 is used to compute the effect of the modified correspondences on the relation between  $\delta C_1$  and  $\delta C_2$ <sup>4</sup>. For reasons of clarity, some boundary cases are

Combining prop_pos and mod_neg				
	$B_1 < B_2$	$B_1 = B_2$	$B_1 > B_2$	$B_1 ? B_2$
$\delta A_1 < \delta A_2$	$\delta C_1 ? \delta C_2$	$\delta C_1 < \delta C_2$	$\delta C_1 < \delta C_2$	$\delta C_1 ? \delta C_2$
$\delta A_1 = \delta A_2$	$\delta C_1 < \delta C_2$	$\delta C_1 = \delta C_2$	$\delta C_1 > \delta C_2$	$\delta C_1 ? \delta C_2$
$\delta A_1 > \delta A_2$	$\delta C_1 > \delta C_2$	$\delta C_1 > \delta C_2$	$\delta C_1 ? \delta C_2$	$\delta C_1 ? \delta C_2$

Table 1: A Calculus for Processing Modified Proportionalities

omitted (for instance, if one allows modifiers to be *zero*). These can be defined in a similar way. By the same token, analogous calculi are defined for **prop\_neg**'s and **mod\_pos**'s.

This calculus is sufficient only for computing the influence of a single modifier on a single proportionality for  $C$ . Consequently, we have to expand our calculus to incorporate multiple modifiers and multiple proportionalities. We do this by first combining multiple modifiers for each proportionality, then computing the proportionalities one by one, and finally combining the different proportionalities.

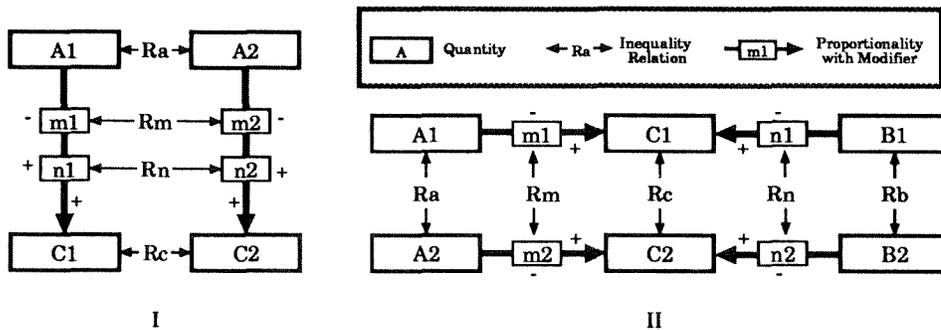


Figure 4: Multiple Proportionalities and Modifiers

This is best explained by means of examples. In Figure 4-I, a schematic representation is given of the following relations:

```
prop_pos(Prop1, A1, C1)
mod_neg(m1, Prop1)
```

<sup>3</sup>Because the systems are modelled by instances of the same generic model fragments, the relations  $\text{Prop}_1$  and  $\text{Prop}_2$  are abstractions of the same quantitative formula.

<sup>4</sup>The column for  $B_1 ? B_2$  is added because combined modifiers can be ambiguous, as will become clear below. In subsequent tables, similar columns are omitted.

`mod_pos(n1, Prop1)`  
`prop_pos(Prop2, A2, C2)`  
`mod_neg(m2, Prop2)`  
`mod_pos(n2, Prop2)`

In addition there are inequality dependencies between the quantities ( $R_a$ ,  $R_m$ ,  $R_n$ , and  $R_c$ ). We now want to calculate the *combined influence* of the inequality relations  $R_m$  and  $R_n$  on  $R_c$ . This is done by using a simple combination calculus, as depicted in Table 2. The right-side table applies to our example, because we want to combine a `mod_neg` with a `mod_pos`. Modifier pairs that have opposite inequality relations can be combined (e.g., “>” and “<” yields “>”, where the resulting relation is one between `mod_neg`’s), and equal modifier pairs can be omitted (e.g., “>” and “=” yields “>”). For proportionality relations with more than

	<	=	>
<	<	<	?
=	<	=	>
>	?	>	>

Similar relations (e.g., `mod_pos` with `mod_pos`)

	<	=	>
<	?	>	>
=	<	=	>
>	<	<	?

Opposite relations (e.g., `mod_pos` with `mod_neg`); resulting relation is of type defined in the top row)

Table 2: Calculi for Combination

two modifiers, the calculus can be applied incrementally. The result of this addition is that all proportionalities can be represented as having exactly one modifier, because non-modified proportionalities can be seen as having one modifier which is equal in both systems. Similarly, the different proportionalities for one quantity can now be combined by adding the inequality relations between modifiers and quantities. Consider the relations in Figure 4-II, illustrating the relations

`prop_pos(PropA1, A1, C1)`  
`mod_neg(m1, PropA1)`  
`prop_pos(PropB1, B1, C1)`  
`mod_neg(n1, PropB1)`  
`prop_pos(PropA2, A2, C2)`  
`mod_neg(m2, PropA2)`  
`prop_pos(PropB2, B2, C2)`  
`mod_neg(n2, PropB2)`

The two proportionality relations influencing  $C$  are calculated separately using the calculus from Table 1<sup>5</sup>. This time, we do not calculate the actual inequality between  $\delta C_1$  and  $\delta C_2$ , but we calculate the *relative influence* of each proportionality. For example, if the left side proportionality relations in Figure 4-II would have caused  $R_c$  to change from “=” to “>” provided it was the only proportionality affecting  $C$ , then we can say that the relative influence on  $R_c$  is “>”. When we have done this for all proportionalities affecting  $C$ , then we can add them conform the calculus in Table 2.

<sup>5</sup>When the subscripts for a quantity are omitted, the statement holds for both instances.

Summarising, modified proportionalities are dealt with by computing the effect of the modifier on the proportionality using the calculus in Table 1. In the case that multiple modifiers exist for one proportionality, the inequality relations between corresponding modifiers are combined by using the calculus in Table 2. If more than one proportionality affects the same quantity, the relative influences of the inequalities involved are combined by the latter calculus.

### 5.3 The Balance Problem Revisited

Exploiting the augmentation we proposed in the previous section, we will now show that we can predict the problematic state transition in the balance problem (Figure 2, State 2 to State 3).

In State 2, we have  $H_L > H_R$ ,  $V_L < V_R$ , and  $W_L < W_R$ . The problem was that the transition from  $H_L > H_R$  to  $H_L = H_R$  was not found. With our new primitive, this can be modelled adequately. We redefine the proportionality relations between the volumes and the heights as in Section 5.1.

```
prop_pos(Prop_L, V_L, H_L), mod_neg(W_L, Prop_L)
prop_pos(Prop_R, V_R, H_R), mod_neg(W_R, Prop_R)
```

Given  $H_L > H_R$ , and thus  $\delta V_L > \delta V_R$ , we use  $W_L < W_R$  to derive that  $\delta H_L > \delta H_R$  (see Table 1).  $\delta H_L > \delta H_R$  together with  $H_L > H_R$  yields the desired transition to  $H_L = H_R$ .

Now suppose the problem is extended by taking into account the depths ( $D_L, D_R$ ) of the containers as well. Again considering the same situation as in State 2, we can now derive that if (for example)  $D_L > D_R$ , the resulting combined modification, and hence the state transition, is ambiguous:

**if  $W_L < W_R$  and  $D_L > D_R$  then  $M_L ? M_R$**

Here  $M_L, M_R$  are the resulting combined modifiers for both sides. Applying these modifiers on the proportionality relations between volumes and heights yields  $H_L ? H_R$  (see Table 1).

If, on the other hand,  $D_L = D_R$  or  $D_L < D_R$ , the same transition to  $H_L = H_R$  is found (see the left table in Table 2):

```
if W_L < W_R and D_L = D_R then M_L < M_R
if W_L < W_R and D_L < D_R then M_L < M_R
```

Applying these combined modifiers yields

**if  $\delta V_L > \delta V_R$  and  $M_L < M_R$  then  $\delta H_L > \delta H_R$**

Although the balance problems may seem to be a toy domain, the observed problem occurs in many situations. On the one hand, processes may be compared in real-life situations. Also, comparisons of predicted behaviour for different values of relevant constants can be exploited for tuning equipment. On the other hand, it is often found in teaching situations; a lot of physics problems in text books are based on the comparison of two similar processes. For example:

- “Two liquids with different heat capacities are heated; which one boils first?”
- “Two different masses slide down a hill. Compute the difference in friction.”
- “The front brakes of a car are unequally worn out. In what direction will the car turn if you brake?”

By taking into account the influence of constant values, a better understanding of the process is possible.

## 6 Discussion and Concluding Remarks

We showed that non-changing quantities may have a significant influence on the prediction of behaviour of physical systems. When comparing similar processes, which regularly occurs in teaching situations as well as in real-life applications, current QR techniques are not capable of modelling the desired behaviour. We presented a new ontological primitive that enlarges the scope of qualitative reasoning by modelling the changes in behaviour that result from influences of non-changing quantities.

Two implementations have been discussed. They differ with respect to the explicit representation of the multiplication and the addition of modifiers. The latter is explicit with respect to its role in the prediction of causal behaviour, whereas the multiplication relation can also be used for other (teaching) purposes, for instance providing general background knowledge about the relations between the different quantities. That is, it may be useful to teach a student the multiplication relation explicitly, and not only its causal (more implicit) consequences. On the other hand, knowing which quantity is the modifier (thus by explicit representation) simplifies the realisation of causal explanations for a teaching system.

The extension proposed here is a small, but nevertheless important step. In the development of QR research, the power of qualitative reasoning has increased by gradually lowering the level of abstraction. First, only (qualitative) quantity values were used. An important improvement was the introduction of inequality reasoning, which employs the quantitative values of quantities. We follow this line one step further by introducing the modified proportionality, facilitating the exploitation of constants in the comparison of similar behaving systems. When predicting the behaviour of solitary systems these constants can be omitted, but when comparing similar systems they can influence the behaviour significantly.

In the context of our current research project on teaching qualitative reasoning, the new primitive will be employed to facilitate better explanation and cognitive diagnosis. Especially in tutoring situations, comparison of similar systems is very useful for bringing about a better understanding of the relative influences of different quantities on a system.

## Acknowledgement

We would like to thank Jaap Kamps for providing useful comments on earlier drafts of this paper, and for some of the drawings.

## References

- [1] B. Bredeweg. *Expertise in qualitative prediction of behaviour*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, March 1992.
- [2] M.T.H. Chi, P.J. Feltovich, and R. Glaser. Categorization and representation of physics problems by experts and novices. *Cognitive Science*, 5:121–152, 1981.
- [3] J.H. de Kleer and J.S. Brown. A qualitative physics based on confluences. *Artificial Intelligence*, 24:7–83, 1984.
- [4] K. de Koning and B. Bredeweg. The balance tutor. Technical report, University of Amsterdam, July 1993.
- [5] R. Elio and P.B. Sharf. Modeling novice-to-expert shifts in problem-solving strategy and knowledge organization. *Cognitive Science*, 14:579–639, 1990.

- [6] K. D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.
- [7] D. Gentner and A. L. Stevens, editors. *Mental Models*. Lawrence Erlbaum, Hillsdale, 1983.
- [8] J. H. Larkin, J. McDermott, D. P. Simon, and H. A. Simon. Models of competence in solving physics problems. *Cognitive Science*, 4:317–345, 1980.
- [9] R. Simmons. Commonsense arithmetic reasoning. In *Proceedings of the AAAI-86*, pages 118–124, 1986.

# Abstraction Framework for Compositional Modeling

Diane Chi and Yumi Iwasaki

Knowledge Systems Laboratory

Stanford University

701 Welch Road, Bldg. C, Palo Alto, CA 94304  
chi@ksl.stanford.edu and iwasaki@ksl.stanford.edu

## Abstract

Abstractions transform the representation of a complex problem into a simpler, more manageable form. Many researchers have proposed methods of abstraction for various types of problems in specific problem areas, but it is difficult to compare these methods. The representation formalisms of the abstraction methods vary widely, and even the definition of abstraction changes from one group to another.

In this paper, we present a framework for characterizing various abstraction relations in the context of compositional modeling. Our framework classifies abstractions along two dimensions: the *method* used to transform the representation and the *representational element* to which the method is applied. We limit our discussion to the model fragments used in compositional modeling so that we may precisely define each abstraction method and analyze their representational and computational consequences. Such a framework is an important step towards automatic model formulation as well as automatic generation model fragments.

## Introduction

Abstraction is an essential concept in modeling complex phenomena. For a given phenomenon, there are many possible abstraction levels at which it can be modeled. There is no single “correct” level of abstraction, since any model is necessarily an abstraction and the goodness of the abstraction depends on one’s goal, i.e. the problem one is trying to solve by constructing the model. For a model to be useful, it must be at the appropriate level of abstraction, which means it must contain enough information to answer the given question with sufficient precision and accuracy but without containing too much unnecessary detail.

A host of different methods of abstraction have been proposed in many problem areas such as modeling (Iwasaki & Simon 1994; Amador & Weld 1990; Williams 1991), planning (Amarel 1981; Fikes, Hart, & Nilsson 1972; Sacerdoti 1974; Knoblock 1989), learning (T. M. Mitchell 1990; Minton 1988; Knoblock 1990; Giordana & Saitta 1990), and theorem-proving (Giunchiglia & Walsh 1992; Plaisted 1981; 1986). However, it is difficult to compare the computational and

representational implications of each abstraction technique. For one thing, the representation formalisms of the methods vary widely. Furthermore, the abstraction methods themselves range from the simple act of deleting elements from the representation to using a completely different ontology, seeming to render the term “abstraction” as a catch-all term for any transformation of a representation. We note however, that all transformation techniques deemed as abstraction share a common goal of simplifying the problem representation with the intention of simplifying the problem solving process as a result with some metric of simplicity.

Giunchiglia and Walsh introduced a theory of abstraction based on work in theorem-proving and planning (Giunchiglia & Walsh 1992), where they informally defined abstraction as “the process of transforming the representation into another form that is simpler to handle yet retains the desirable characteristics of the original problem”. In this paper, we adopt this general definition and consider various types of abstraction transformations in the context of compositional modeling. In particular, we present a framework for characterizing various abstraction relations among model fragments and their components.

Our motivation for this work is to extend our model formulation algorithm to include all types of abstraction methods on model fragments. We have been working on a technique for automatically formulating a model that is appropriate for answering a given query. Iwasaki and Levy reported on an approach based on the relevance and irrelevance of knowledge that characterizes modeling assumptions underlying model fragments and selects among them to formulate an adequate model for a given query (Iwasaki & Levy 1994). This relevance-based approach is appropriate when the difference between model fragments can be adequately characterized by the difference in the aspects of the situation that are considered *relevant* (and are thus included in the model fragment). We found that though most cases of abstraction relations among models can indeed be characterized by such differences, there are cases where such differences are awkward to capture

in this manner. Such cases include situations where a model fragment is abstracted using simpler mathematical relations and situations where one quantity that is hard to measure accurately is approximated by another whose value is easy to obtain. Thus, a broader framework for characterizing relations among alternative descriptions is needed to determine how to extend our current model formulation mechanism.

Our framework classifies abstractions along two dimensions: the method used to transform the representation and the aspect of the representation to which the method is applied. The methods are aggregation, elimination, and approximation. Generally speaking, aggregation replaces a set of elements (of the same type) in the representation by one aggregate element. Elimination removes selected elements from the representation. Approximation replaces an element by another that is deemed "close" to the original by some measure of closeness.

These three general methods are applied to different types of representational elements. In the compositional modeling paradigm, a model consists of model fragments, which in turn consist of conditions and consequences. Both conditions and consequences consist of relations on quantities.<sup>1</sup> The three general methods are applied to each of these representational elements with various computational implications. Limiting our discussion to model fragment abstraction allows us to define each abstraction method precisely. It also allows us to analyze their representational and computational consequences in concrete terms.

Though many specific abstraction techniques have been presented, only a few general theories of abstraction have been proposed. Giunchiglia and Walsh classify abstractions into Theorem Decreasing (TD) and Theorem Increasing (TI) abstractions based on whether the abstract representation has less theorems (and thus more interpretations) than the original representation or vice versa (Giunchiglia & Walsh 1992). Their framework examines the properties of logical theories (which correspond to models in our context) where one is an abstraction of the other. However, the classification of all abstractions as TD or TI is too coarse to provide much insight into the effects of different abstraction models on the consequences of simulation. Furthermore, we find that some common types of model transformations that may be considered as types of abstraction are not TD nor TI.

Our classification approach more closely resembles the approach taken by Struss in his theory of model simplification and abstraction. He for-

mally defined abstraction, approximation, and simplification, and analyzed some representational consequences of such transformations (Struss 1991). Our three general methods—aggregation, elimination, and approximation—generally correspond to his definitions of abstraction, simplification, and approximation. One important difference is that while Struss considers relations among (complete) models, we study abstraction relations among *model fragments*, which must be combined to compose a complete model. The fact that a the problem of model fragment abstraction is much less constrained than abstraction of complete models makes it more difficult to prove general theorems about the representational and computational implications of performing abstractions on them.

This document is organized as follows: In the remainder of this section, we briefly describe the compositional modeling paradigm and our approach to model formulation. The second section presents our abstraction framework and a small example of a graph of model fragments related by the abstraction relations described in the framework. The final section discusses the implications of our work.

## Compositional Modeling and Relevance-based Model Formulation

In this section, we briefly describe the compositional modeling paradigm for representing physical knowledge and predicting behavior (Falkenhainer & Forbus 1991), as well as our method for model formulation based on relevance (Iwasaki & Levy 1994).

A physical situation is modeled as a collection of model fragments.<sup>2</sup> Each model fragment represents some aspect of a physical object or a physical phenomenon. A model fragment is composed of conditions and consequences. The conditions specify the individuals that must exist and the requirements they must satisfy for the phenomenon to occur. The consequences specify the functional relations among the attributes of the objects that are entailed by the phenomenon.

If there exists individuals  $a_1, \dots, a_n$  that satisfy the operating conditions of a model fragment  $M$  at time  $t$ , we say that an instance of  $M$  is active at that time. We will call  $a_1, \dots, a_n$  the *participants* of the instance of  $M$ . We denote the particular instance by  $M(a_1, \dots, a_n)$ .

The prediction mechanism on model fragments works generally as follows: For a given situation, the system identifies the model fragments whose conditions hold as the active model fragment instances. In each state, this set of active model fragments forms the *simulation model*. The simulation model gives rise to equations that must hold among variables as a consequence of the phenomenon taking place. The prediction mechanism uses the equations to determine the next state

<sup>2</sup>We will use the definition of model fragments as given in (Farquhar *et al.* 1993).

<sup>1</sup>By *quantities*, we mean any kind of attribute of a model fragment. Quantities may be numerical or non-numerical attributes. In this paper, we assume that the range of a quantity forms some kind of a metric space, though this restriction is not strictly necessary for application of most of the abstraction methods discussed in the section on the Abstraction Framework.

of the simulation. Each state has a simulation model along with a set of variable values and predicates that hold. The prediction mechanism outputs a sequence of states. If prediction is performed qualitatively, the output can be represented as a graph. Each path through the graph from the initial state represents a possible behavior of the system. Such a path is called a *trajectory*.

Our model formulation approach described in (Iwasaki & Levy 1994) consists of making two choices. The first choice is deciding what phenomena to represent in the model. The second choice is selecting the model fragment(s) to include in the model from the set of all possible model fragments. The set of possible model fragments includes different descriptions of the modeled phenomena using different modeling assumptions. The first choice is made by backward chaining through the possible causal influences on the variables of interest to the user. The second is made by reasoning about the modeling assumptions necessary to answer the given query. To facilitate this choice, model fragments in the knowledge base are organized into structures called *assumption classes*. An assumption class is a graph of model fragments representing different ways to describe the same phenomenon. The language of relevance and irrelevance of knowledge (Levy 1993) is the basic language used to represent the modeling assumptions underlying different model fragments in an assumption class. Our goals for developing the abstraction framework presented in this paper are (1) to expand this language, especially in the direction of being able to characterize a broader class of abstraction relations more precisely, and (2) to facilitate selection among model fragments by enabling analysis of the representational and computational implications of using different abstract model fragments.

The following section presents our abstraction framework, which provides a classification of ways to operate on each type of representational element to produce a more abstract version. We consider only transformation techniques that do not add new information to the representation, since it is debatable whether transformations that add information are abstractions at all. Also, we do not consider cases where the abstract representation employs a completely new ontology of the domain unless such an ontology is actually a product of applying one of the abstraction methods discussed below.

## Abstraction Framework

We classify all abstraction methods into three general classes, namely elimination, approximation, and aggregation.

- **Elimination** is the removal of all references to some selected elements of the representation except in cases where such removal results from applying approximation or aggregation.

- **Approximation** replaces an element by another element that is less accurate, but closely resembles the original element. Approximation applies only to elements in some metric space that can be used to establish a measure of similarity.
- **Aggregation** involves grouping related elements into aggregates and representing the problem in terms of the aggregates.

The general abstraction methods can be applied to the representational elements of the knowledge base and predicted behaviors. Table 1 summarizes the applicability of the methods to each type of element.<sup>3</sup> A table entry “s” indicates that the method applies to a single element (i.e. a single quantity, a single model fragment, etc.), while “m” indicates that the method applies to multiple elements. As shown in the table, aggregation requires more than one element of a particular type, while the other methods apply to individual elements.

In the following, we provide detailed explanations of each entry in the knowledge base part of the table. As described in the previous section, a knowledge base consists of model fragments composed of conditions and consequences. Conditions and consequences are defined in terms of relations among quantities. Although we discuss abstraction of each type of representational element individually, we must note that abstractions of different elements are not independent. Abstraction of one type of element often necessitates abstraction of another type of element as discussed below.

## Quantity Abstraction

A quantity  $Q_1$  of a model fragment  $MF$  is formally defined as a mapping from the set of all instances of  $MF$  to a set of functions that map time to actual values. In other words, if  $MF_1$  is an instance of  $MF$ , then  $QF_{11}$  is a function of  $Q_1$  and  $MF_1$  such that  $QF_{11}(t)$  for some time  $t$  is the value of the quantity  $Q_1$  of the model fragment  $MF_1$  at the time  $t$ . We distinguish abstraction of the range of a quantity<sup>4</sup> and abstraction of the quantity function ( $Q_1$ ) itself.

**Range Abstraction.** When abstracting the range of a quantity, one changes the set of possible values for the quantity. Elimination and approximation abstract individual values in the range of a quantity. Aggrega-

<sup>3</sup>For the sake of completeness, the table includes behavior abstraction, since the general methods can also be applied to the representation of behaviors. In this paper however, we limit our discussion to the abstraction of model fragments.

<sup>4</sup>We are actually referring to the range of the the function  $QF_{11}$  returned by the quantity function applied to a model fragment instance. We will call it the *range* or *value range* of the quantity for brevity unless the meaning is not clear from the context.

Repr. Element  Abstr. Method	Knowledge Base					Behavior	
	Model Fragment					State	Traj.
	Quantity		Cond.	Conseq.			
	Range						
Elimination	s	s	s	s	s	s	s
Approximation	s	s	s	s	-	-	-
Aggregation	m	m	m	m	m	m	m

Table 1: Applicability of Abstraction Methods to Elements of the Representation

tion partitions the range into subsets so that the values are represented in terms of the subsets.

**Range elimination** removes an individual value or a set of values from the range. The removal of selected values is often motivated by knowledge that certain values are unattainable or highly unlikely for the given quantity. For instance, in a model fragment representing a thruster component of a jet propulsion system, one may eliminate all negative values in the range of `pressure-differential`, which is defined as the difference between the input pressure and the output pressure. Elimination of all negative values amounts to making the assumption that there is no possibility of reverse pressure. Such an abstraction can reduce the computational effort of a prediction mechanism by pruning unlikely behaviors.

**Range approximation** replaces a value (or a sub-range of values) in the value range of a single quantity with another value (or subrange of values) that is somehow “good enough” for the given problem. Examples of range approximation include changing the precision for quantity values and idealization of quantities. If one changes the precision of numerical values by rounding them all to the second decimal place, a set of values is replaced by a single approximate value. *Idealization* is an extreme example of such an approximation where one replaces the entire value range of a quantity with a single extreme value. The approximation of the interactions between solid objects as frictionless surfaces is an example of idealization. Range approximation results in an abstract representation that produces less accurate numerical predictions.

**Range aggregation** occurs by grouping elements in the range into subsets and using the subsets to represent values. The grouping creates either regular or uneven intervals. Discretization of a continuous range of a quantity into intervals bounded by landmark values, as in qualitative calculus, is an example of range aggregation. Range aggregation results in less precise predictions. When the quantity abstracted is time, range aggregation results in temporal abstraction.

**Quantity Function Abstraction.** We now consider abstraction of quantity functions.

**Quantity elimination** removes a quantity from the representation. For example, consider a model fragment representing a tank with two pressure sensors provided for redundancy. If the original model fragment has two pressure quantities corresponding to each of the sensor readings, one can abstract the model fragment by eliminating one of the pressure quantities. This generates a more compact model fragment; however, as in this example, quantity elimination may eliminate the redundancy present in the actual device and result in a less robust model.

**Quantity approximation** replaces a quantity with a similar quantity where the value of the new quantity might be easier to obtain. For instance, consider a detailed model fragment of a fuel tank including the quantity `amount-of-fuel`. If the exact measurement of the remaining amount is difficult to obtain due to the structure of the tank, but the initial amount and the history of the fuel consumption are available, one might create another model fragment that contains the quantity `computed-fuel-amount` based on these variables to replace `amount-of-fuel`. However, if the approximation is only good under some assumptions, then a model consisting of the approximate model fragment might fail to reflect reality when the assumptions do not hold. For instance, in the case of `amount-of-fuel`, if there exists a large leak between the tank and the component that actually consumes the fuel, `computed-fuel-amount` will not adequately approximate `amount-of-fuel`.

**Quantity aggregation** replaces a set of quantities with a new representative quantity. There are a variety of ways to define the representative, including summing and averaging. For a tank having two pressure sensors that monitor the pressure within the tank, one may replace the quantities `pressure-reading-from-sensor-a` and `pressure-reading-from-sensor-b` with an `average-pressure-reading` quantity whose value is the average of the original two quantities. Quantity aggregation reduces the number of quantities, but it may or may not affect the prediction accuracy depending on the other equations in the simulation model.

Since quantities are used to state the conditions

and consequences of model fragments; abstraction of a quantity necessitates the modification of conditions and consequences that reference the abstracted quantity. In ABSTRIPS, abstraction layers are defined in this manner (Sacerdoti 1974). The planning operators in STRIPS are comparable to model fragments with the applicability conditions and the consequences (consisting of ADD and DELETE lists) of the operators corresponding to the conditions and consequences of model fragments. ABSTRIPS ranks the predicates used in specifying the conditions and consequences of the planning operators according to their criticality. To define an abstract planning space, abstract operators are defined by ignoring less critical predicates in the conditions and consequences of the original operators.

### Condition Abstraction

The *conditions* part of a model fragment is a list of atomic formulae, which is an implicit conjunction of the conditions.

**Condition elimination** removes selected conditions from the list of conditions in a model fragment. Condition elimination may occur as a necessary consequence of quantity abstraction, but it is also performed independently when a condition is deemed uncritical for some purpose. Statistical information may motivate such elimination. If a condition is known almost never to fail, one may decide to eliminate the condition to produce a simpler model. We also include in this category replacement of a condition  $C_1$  by another condition  $C_2$  that is strictly weaker than  $C_1$ , since  $C_1$  is logically equivalent to  $C_1 \wedge C_2$  if  $C_1 \rightarrow C_2$ .

**Condition approximation** replaces a condition with another similar condition, which is simpler in some respect. For instance, the activation condition for a tank model fragment may require that ( $> \text{pressure-differential } 1e-5$ ). In an abstract model, we might replace this condition with ( $> \text{pressure-differential } 0$ ). Such replacement might occur as a necessary result of range approximation, or it might result as an independent decision.

**Condition aggregation** replaces a set of conditions with an aggregate condition. For instance, one may replace the conditions ( $= \text{inflow-1 outflow-1}$ ) and ( $= \text{inflow-2 outflow-2}$ ) with ( $= (+ \text{inflow-1 inflow-2}) (+ \text{outflow-1 outflow-2})$ ). Condition aggregation reduces the number of conditions in a model fragment, but the new conditions may be more complex syntactically. The aggregate condition is often weaker than the conjunction of the original set of conditions.

### Consequence Abstraction

The *consequences* part of a model fragment is a list of atomic formulae, which is interpreted as an implicit conjunction. Consequences cannot contain embedded conditions.

**Consequence elimination** removes selected consequences from model fragments. As with condition elimination, consequence elimination may occur as a necessary result of quantity elimination. This category includes replacement of a consequence by another that is strictly weaker than the original.

**Consequence approximation** replaces a relation in the consequence of a model fragment with a simpler relation. For instance, one might replace a complex equation with a simpler, approximate equation. Examples of such approximations include equilibration, exogenization (Iwasaki & Simon 1994), and piecewise linear approximation.

Equilibration is applicable to the consequence equations of one model fragment representing some mechanism that restores equilibrium much quicker than other mechanisms in the system. In this situation, one can regard the fast mechanism as acting instantaneously. The equilibration operation replaces a dynamic equation representing a fast mechanism by its respective equilibrium equation. For example, using a model fragment for a pressure sensor, one might assume that ( $= \text{sensed-pressure pressure-reading}$ ). This assumption treats the reading of the pressure on the sensor as an instantaneous process, giving the pressure at the current instant. Determining the *pressure-reading* is considered a fast mechanism compared to other processes in the model. Similarly, exogenization—replacement of a slow mechanism by a constant equation—is also a type of consequence approximation.

Piecewise linear approximation may be used to simplify complex equations given in the consequences of a model fragment. In piecewise linear approximation, one approximates a non-linear equation by pieces each of which can be approximated by a linear equation.<sup>5</sup>

**Consequence aggregation** combines a set of consequences into an aggregate. For instance, two consequences of the model fragment representing a heat exchanger are the conservation of mass equations for the hot and cold flows. In an abstract model, one might combine them into a single conservation of mass equation. Consequence aggregation decreases the number of equations in a model fragment, but the new consequences may be weaker than the conjunction of the original consequences.

### Model Fragment Abstraction

Methods of abstracting model fragments include eliminating selected model fragments and aggregating a set of model fragments into one model fragment. Model fragment abstraction is often motivated by knowledge

<sup>5</sup>If the formal definition of a model fragment does not allow conditional consequences as it is not in CML, each linear piece will have to be represented by separate model fragments. This is a rare case where abstraction of one model fragment results in a set of model fragments.

of their structural, functional, or statistical relationships.

**Model fragment elimination** removes the model fragments representing certain elements in the problem description. Often, functional or statistical knowledge motivates model fragment elimination. For instance, consider a device which contains a secondary valve as a backup in cases of failure of its primary valve. One may choose to eliminate the model fragment for the secondary valve, based on the knowledge that the secondary valve functions solely as a backup. Statistical knowledge about a population of individuals may also motivate model fragment elimination. If it is known that 99.9% of a certain population of carbon molecules have the molecular weight of 24, one may choose to eliminate all model fragments representing carbon molecules having other weights.

**Model fragment aggregation** replaces a set of model fragments with an aggregate model fragment. Again, functions and population statistics are common bases for performing such an abstraction. Aggregation reduces the total number of model fragments and allows one to ignore the unnecessary details.

Structural or functional aggregation replaces a set of model fragments representing components by a model fragment that represents their structural or functional super-component. Aggregation of a nearly decomposable dynamic system (Simon & Ando 1961) is an example of model fragment aggregation if we view each mechanism represented by an equation in the original system as a model fragment.

Amador and Weld discuss population abstraction in (Amador & Weld 1990). Amador and Weld model population systems on three levels: the individual level, the aggregate level, and the macro level. The aggregate level contains a collective description of individual level properties. Applying statistical operators, such as *Summation*, *Mean*, *Max*, and *Min*, to the attribute values of individuals in the population generates probabilistic descriptions of the individuals. The modeler uses these statistical operators on all the attributes of all the members of a population to create a representative model fragment. For instance, the attribute values of a representative may be the summation of the attribute values for its individual members.

## Example

We illustrate our abstraction framework with a simple example of model fragments representing a hot air balloon. Figure 1 gives the formal specification for the model fragment *hot-air-balloon*.

The hot air balloon is composed of an *envelope* (the balloon), *burner* (to heat the air in the envelope), *basket*, *temperature-sensor1*, *temperature-sensor2*, *pressure-sensor1*, and *pressure-sensor2*. The sensors measure the conditions within the envelope. The quantities *temperature1*, *temperature2*, *pressure1*, and

```
(defModelFragment hot-air-balloon
:quantities
  (temperature1, temperature2,
   pressure1, pressure2,
   balloon-temp, balloon-pres,
   balloon-vol, balloon-mass,
   compressibility-factor,
   universal-gas-const)
:conditions
  ((< (Abs (- temperature1 373K))
       1e-5)
   (< (Abs (- temperature2 373K))
       1e-5)
   (< pressure1 101.3kPa)
   (< pressure2 101.3kPa))
:consequences
  ((= balloon-temp
    (Avg temperature1 temperature2))
   (= balloon-pres
    (Avg pressure1 pressure2))
   (= balloon-vol
    (/ (* compressibility-factor
          universal-gas-const
          balloon-temp
          balloon-mass)
        balloon-pres))))
```

Figure 1: The *hot-air-balloon* Model Fragment

*pressure2* indicate the readings from the respective sensors. Table 2 specifies abstraction methods that are performed on *hot-air-balloon* and the abstracted representational element. For each abstraction, the table indicates the portion of the original representation that is abstracted and specifies its abstract representation. The abstract model fragments are named to correspond with the graph of model fragments in Figure 2. The graph illustrates the relationship between the original and abstract representations of *hot-air-balloon*. The *hot-air-balloon* is an aggregate model fragment, representing the collection of its component model fragments.

## Discussion

The framework categorizes abstraction relations among model fragments according to the type of transformation performed on one model fragment to produce another model fragment.

It would be ideal if one could make general statements about the representational consequences of applying these transformations on model fragments using such general theories of abstraction as the one proposed by Giunchiglia and Walsh or the one proposed by Struss. Unfortunately, it is impossible to simply apply either of these general theories to abstractions of

Abstraction Type	Original Representational Elements	Abstract Representational Elements	Abstract Model Fragment Name
Quantity Elimination	temperature1 temperature2	temperature1	balloon-with-single-temp-sensor
Quantity Aggregation	temperature1 temperature2	avg-temp	balloon-with-sensor-averager
Condition Elimination	(< pressure1 101.3kPa) (< pressure2 101.3kPa)	(< pressure1 101.3kPa)	balloon-with-single-pres-cond
Condition Approximation	(< (Abs (- temperature1 373K 1e-3))	(= temperature1 373K)	balloon-with-approx-temp-cond
Condition Aggregation	(< pressure1 101.3kPa) (< pressure2 101.3kPa)	(< (+ pressure1 pressure2) 202.6kPa)	balloon-with-agg-pres-cond
Consequence Approximation	(= balloon-vol (* compressibility-factor universal-gas-const balloon-temp balloon-mass (inv balloon-pres)))	(= balloon-vol (* quantity-contained ideal-gas-const balloon-temp (inv balloon-pres)))	balloon-assuming-ideal-gas
Model Fragment Aggregation	envelope burner basket temperature-sensor-t1 temperature-sensor-t2 pressure-sensor-p1 pressure-sensor-p2	hot-air-balloon	original-hot-air-balloon

Table 2: Abstraction of the hot-air-balloon Model Fragment

model fragments. Giunchiglia and Walsh's theory compares two logical theories and does not easily extend to comparisons among model fragments. Performing the types of transformations discussed in this paper on a model fragment produces a new set of model fragments. A knowledge base in the compositional modeling framework is a set of model fragments and does not constitute a consistent logical theory. A knowledge base can contain logically inconsistent model fragments though different subsets of the knowledge base may give rise to logically consistent models that are useful in different situations. Struss' theory also does not serve our purpose since his theory presumes a complete model, and in our case, we need a way to characterize differences among potentially relevant fragments of models before they are assembled into a complete model.

Nevertheless, if we changed our problem and made strong assumptions that (1) we are starting from a set of model fragments that constitute a complete and

logically consistent model, and (2) an abstraction operation performed on one model fragment will be performed consistently throughout the set, we could make the following general observations:

- Elimination generally results in TD abstraction. It falls in the category of simplification as defined by Struss but is not necessarily a representational transformation<sup>6</sup> since the result may no longer be a model (Struss 1991).
- Approximation is neither TD nor TI. It is a representational transformation, and is a type of simplification as defined by Struss.
- Aggregation is TD. It is a representational transformation.

<sup>6</sup>Intuitively, a transformation is a representational transformation if it results in a model that covers all the situations covered by the original model; however, it may make different kinds of distinctions from the original.

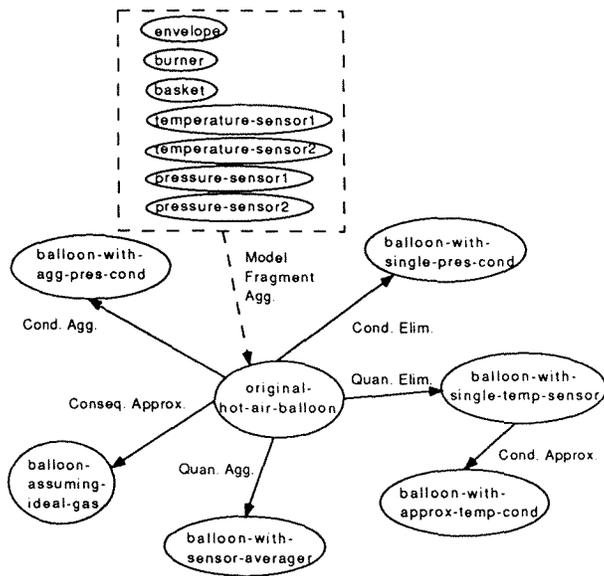


Figure 2: Graph of Model Fragments

Our immediate motivation for this work is to expand our model formulation method by using this framework to precisely characterize a broader class of abstraction relations among model fragments. Our next step is to analyze various types of assumptions underlying the transformations discussed in this paper. We must then decide how to represent such assumptions, what additional information can be specified about the query, and how to reason with them in order to choose an appropriate level of abstraction for each model fragment.

Further work is needed to predict the computational consequences of abstraction in terms of the effects they have on the results of behavior prediction. One observation that can be made is that abstraction of the representation as discussed in this paper does not necessarily result in a simpler prediction. In particular, condition abstraction generally results in a weaker condition, which could result in more model fragments becoming activated during simulation.

Though our immediate goal is to incorporate the abstraction framework into the model formulation mechanism and use further analysis results to guide model fragment selection, this framework is an important step towards the automatic generation of abstract model fragments. The proposed framework has the advantage of being easy to operationalize once one selects the aspect of the representation to abstract. We plan to explore this possibility of automatic abstraction of model fragments in the future.

### Acknowledgments

The first author was supported by the National Physical Science Consortium Fellowship.

This research was sponsored by the Advanced Re-

search Projects Agency, ARPA Order 8607, monitored by NASA Ames Research Center under grant NAG 2-581; and by NASA Ames Research Center under grant NCC 2-537.

### References

- Amador, F. G., and Weld, D. 1990. Multi-level modeling of populations. In *Proceedings of the Fourth International Workshop on Qualitative Physics*.
- Amarel, S. 1981. On representations of problems of reasoning about actions. In Webber, B. L., and Nilsson, N. J., eds., *Readings in Artificial Intelligence*. Los Altos, CA: Morgan Kaufmann.
- Falkenhainer, B., and Forbus, K. 1991. Compositional modeling: Finding the right model for the job. In *Artificial Intelligence*, Vol. 51, pp. 95-143.
- Farquhar, A.; Bobrow, D.; Falkenhainer, B.; Fikes, R.; Forbus, K.; Gruber, T.; Iwasaki, Y.; and Kuipers, B. 1993. A compositional modeling language. Knowledge Systems Laboratory Technical Report KSL-93-53, Stanford University, Stanford, California.
- Fikes, R. E.; Hart, P.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. In *Artificial Intelligence*, 251-288.
- Giordana, A., and Saitta, L. 1990. Abstraction: A general framework for learning. In *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*.
- Giunchiglia, F., and Walsh, T. 1992. A theory of abstraction. To appear in the *Journal of Artificial Intelligence*.
- Iwasaki, Y., and Levy, A. 1994. Automated model selection for simulation. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*.
- Iwasaki, Y., and Simon, H. 1994. Causality and model abstraction. In *Artificial Intelligence*. to appear.
- Knoblock, C. A. 1989. A theory of abstraction for hierarchical planning. In Benjamin, P., ed., *Proceedings of the Workshop on Change of Representation and Inductive Bias*. Boston, Mass: Kluwer.
- Knoblock, C. A. 1990. Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence*. Los Altos, CA: Morgan Kaufmann.
- Levy, A. Y. 1993. *Irrelevance Reasoning in Knowledge Base Systems*. Ph.D. Dissertation, Stanford University, Stanford, CA.
- Minton, S. 1988. *Learning Search Control Knowledge: An Explanation Based Approach*. Academic Publishers, Hingham, MA.
- Plaisted, D. 1981. Theorem proving with abstraction. In *Artificial Intelligence*, Vol. 16, pp. 47-108.
- Plaisted, D. 1986. Abstraction using generalization functions. In *Proceedings 8th Conference on Automated Deduction*, 365-376.

Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. In *Artificial Intelligence*, Vol. 5, pp. 115-135.

Simon, H. A., and Ando, A. 1961. Aggregation of variables in dynamic systems. In *Econometrica*, volume 29.

Struss, P. 1991. A theory of model simplification and abstraction for diagnosis. In *Working notes of the 5th International Workshop on Qualitative Reasoning*.

T. M. Mitchell, R. M. Keller, S. K.-C. 1990. Explanation-based generalization: A unifying view. In Shavlik, J. W., and Dietterich, T. G., eds., *Readings in Machine Learning*. San Mateo, CA: Morgan Kaufmann. 435-451.

Williams, B. C. 1991. Critical abstraction: Generating simplest models for causal explanation. In *Working notes of the 5th International Workshop on Qualitative Reasoning*.

# Model Decomposition and Simulation

Daniel J. Clancy and Benjamin Kuipers

Department of Computer Sciences

University of Texas at Austin

Austin, Texas 78712

clancy@cs.utexas.edu and kuipers@cs.utexas.edu

## Abstract

Qualitative reasoning uses incomplete knowledge to compute a description of the possible behaviors for dynamic systems. For complex systems containing a large number of variables and constraints, the simulation frequently is intractable or results in a large, incomprehensible behavioral description. Abstraction and aggregation techniques are required during the simulation to eliminate irrelevant details and highlight the important characteristics of the behavior. The total temporal ordering of unrelated events provided by a traditional state-based qualitative representation is one such irrelevant distinction. Model decomposition and simulation addresses this problem.

Model decomposition uses a causal analysis of the model to partition the variables into tightly connected components. The components are simulated separately in the order dictated by the causal analysis beginning with causally upstream components. Information from the simulation of causally upstream components is used to constrain the behavior of downstream components. If a feedback loop exists between components or a set of components are acausally related, then a concurrent simulation is performed for these components. A truth maintenance system is used to record and retract assumptions made during this concurrent simulation.

Model decomposition provides a general architecture which separates the method of simulation from the model decomposition algorithm. This architecture can be used to introduce alternative abstraction techniques to eliminate other irrelevant distinctions.

## 1 Introduction

Qualitative simulation derives a description of the possible behaviors of a dynamic system from a structural representation. The structural representation details the relationship between the variables within the system through constraints. Traditionally, a

state-based approach [Forbus, 1984, Kuipers, 1986, De Kleer and Brown, 1984] is used to describe the device behavior by a set of qualitative states of the system and the transitions between these states. Each qualitative state contains a complete description of the system at either a time point or over a time interval. This level of description provides a complete temporal ordering of events within each behavior. An event occurs when a variable crosses a landmark value or alters its direction of change. The combinatoric complexity of providing a total ordering of events is a general problem encountered within artificial intelligence when trying to reason about the physical world [Hayes, 1985].

For many qualitative reasoning tasks, this level of detail is irrelevant. Branching on unrelated temporal distinctions significantly increases the complexity of the simulation. In addition, this branching makes it more difficult to interpret the behavioral description thus obscuring the relevant behavior.

Williams [1986] was the first to investigate methods of eliminating this complete temporal ordering of events within a simulation with the Temporal Constraint Propagator (TCP). TCP forsakes the traditional state-based approach and independently describes the behavior of each variable over time as a set of variable *histories*. The relevant temporal relations between events are described separately. A temporal ordering of events is only provided when their interaction affects the value of other quantities. Using TCP to reason about physical systems, however, requires a significant amount of work by the modeler. Furthermore, it is unclear how this system can be extended to incorporate advances in other qualitative reasoning paradigms.

Other techniques [Clancy and Kuipers, 1993, Fouché and Kuipers, 1991] have eliminated the irrelevant temporal correlation of events by combin-

ing behaviors and states into a more abstract representation. These abstraction techniques, however, determine the relevance of a correlation between events after the distinction is made in the simulation (i.e. the abstraction is performed in a post-processing fashion), and thus do not sufficiently address the complexity problem. Coiera [1992] eliminates these distinctions by superimposing qualitative predictions from two causally-unrelated processes on a single downstream variable. He does not address how these techniques can be applied to more complicated causal interactions.

Model decomposition bridges the gap between a state-based simulation and a history-based approach. A causal analysis is used to partition the variables into tightly connected components. Each component is simulated separately using a standard state-based simulation, providing a total ordering only for the events within each component. Temporal correlation between events in different components is only provided when necessary to constrain the resulting set of behaviors.

## 2 Model Decomposition - An Overview

Model decomposition uses a divide and conquer approach to the simulation of a qualitative model. The variables within the model are partitioned into components so that closely related variables are contained within the same partition. Each component is simulated independently. The interaction between components is modeled using shared, or *boundary*, variables.

The variables are partitioned using a causal analysis [Iwasaki and Simon, 1986, Iwasaki, 1988] of the constraint network. A sub-model is created for each partition containing the constraints between these *within-partition variables*. *Boundary variables* are causally upstream or acausally related variables directly connected to a within-partition variable through a constraint but not included in the partition. These variables are used to relate the behavior of connected components and are handled specially during the simulation of a sub-model. Constraints connecting within-partition variables and boundary variables are included in the sub-model.

Each sub-model is simulated independently, generating a behavioral description for the within-partition variables. Information about the behavior of the boundary variables is used to constrain the simulation and is obtained from the behavioral description generated by simulating the upstream components

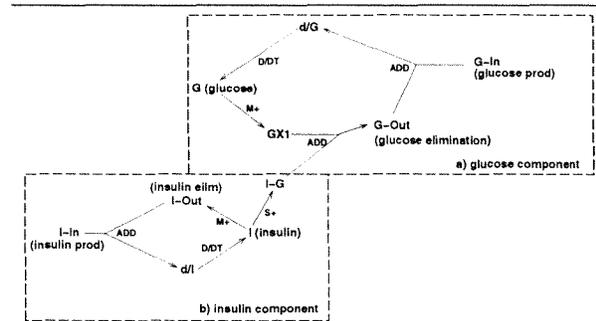


Figure 1: Simplified Glucose Insulin Regulatory System Model

A QSIM model of the human Glucose-Insulin Regulatory System (GIR) which contains two connected feedback loops corresponding to the glucose (a) and insulin (b) regulatory systems.

- The nodes in the graph are variables and the links are QSIM constraints. The arrows indicate the direction of causality derived via a causal analysis of the constraint network.
- The insulin and glucose production (I-In and G-In respectively) are assumed to be constant in this version of the model.

Model decomposition divides the model into two components (identified by the boxes) related by the intermediate variable I-G. A sub-model is created for each component. Each sub-model is simulated separately. Temporal correlation of events is only provided for variables within the same component.

- Since the insulin component is causally upstream, it is simulated prior to and independent of the glucose component. I-G is dependent in this model. I-G's behavior is completely determined during this simulation.
- The glucose sub-model includes I-G as an independent variable. Its behavior is constrained by the results from the simulation of the insulin sub-model. The behavioral description of the glucose component is prevented from branching on distinctions in the value of I-G.

containing these variables. This description guides the behavior of the boundary variables during the simulation of the downstream sub-model. The behavioral description generated during the simulation of a sub-model branches only on distinctions in the values of the within-partition variables. Distinctions in the boundary variables are eliminated through abstraction.

The order of simulation for the individual sub-models is determined by a causal analysis starting with the causally upstream components. If a feedback loop exists between the components or two components are acausally related, then a more complicated simulation strategy must be employed in which the components are simulated concurrently.

### 3 A Model of the Glucose Insulin Regulatory System

Figure 1 shows a constraint network modeling the human Glucose-Insulin Regulatory system (GIR). This model was developed by Ironi and Stefanelli [1993] using the Qualitative Compartmental Modeling Framework (QCMF) designed to assist the user in formulating models of a pathophysiological system and in analyzing their behaviors. The GIR model contains two separate feedback loops. One controls the amount of glucose (G) within the body and the other controls the amount of insulin (I). The feedback loops are connected through the intermediate variable I-G which relates the amount of insulin to the rate of glucose elimination.

Figure 2 shows the results from a simulation of the GIR model using model decomposition. The GIR model is divided into two separate sub-models corresponding to the two feedback loops related by the intermediate variable I-G. This variable is included in both sub-models; however, its behavior is completely determined by the insulin feedback loop. I-G is a dependent variable in this system and an independent variable in the glucose system.

Since the insulin sub-model is causally upstream, it is simulated prior to the glucose component, deriving a complete behavioral description for the variables within the insulin feedback loop including the variable I-G. Next, the glucose sub-model is simulated. The behavior of I-G, determined during the simulation of the insulin component, is used to constrain the possible values of I-G during this simulation. Branches caused by distinctions in the value of I-G are eliminated via abstraction during the simulation of the glucose component.

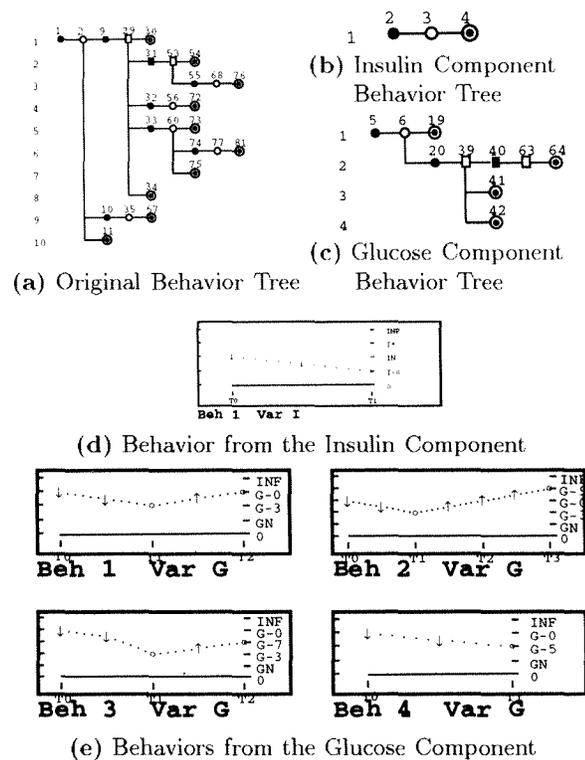


Figure 2: Simulation of the GIR Model  
A normal QSIM simulation of the GIR model using chatter box abstraction [Clancy and Kuipers, 1993] results in 10 behaviors (a). Chatter box abstraction selectively eliminates distinctions in the derivative of the chattering variables thus making the simulation tractable. Chattering regions are identified by a square within the behavior tree (e.g. states 29 and 53).

- Within the 10 behaviors generated, there are four distinct behaviors for the amount of glucose and only a single unique behavior for the amount of insulin.
- The behavior tree provides a complete temporal ordering of the glucose component and the insulin component events even though they are unrelated. These distinctions add complexity to the simulation and obscure the relevant behaviors of the individual components.

Model decomposition generates a behavior tree for each component (b & c) describing the same behaviors as the standard QSIM simulation. No temporal correlation is provided between these behaviors.

- The simulation of the insulin component is performed first and results in a single unique behavior in which the amount of insulin decreases from its initial value and then reaches steady (d).
- Simulation of the glucose component results in four unique behaviors (e). The behaviors are distinguished by the final amount of glucose with respect to the amount at the beginning of the simulation. The behavior is constrained by the results from the simulation of the insulin component.

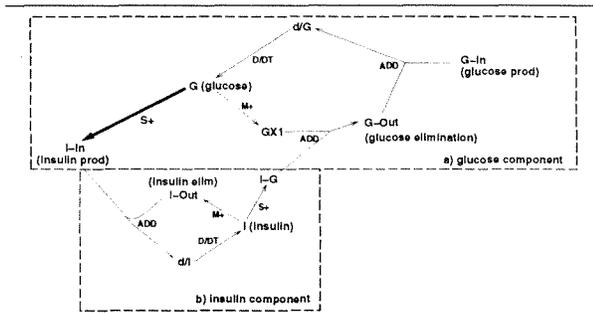


Figure 3: More Complex Model of the Glucose-Insulin Regulatory System

This version of the glucose-insulin regulatory system includes a feedback connection from the glucose component to the insulin component. An S+ constraint relates the amount of glucose to the insulin production rate. I-In is no longer constant.

- The rate of insulin production (I-In) is now included within the glucose sub-model since it is a dependent variable with respect to this component.
- The simulation of the glucose component depends upon the value of I-G while the insulin component depends upon the value of G.
- Since neither component is causally upstream, they must be simulated concurrently.
- During the concurrent simulation, the simulation of the other component is assumed to be complete. States are marked inconsistent if a state cannot be found in the other behavior tree satisfying the constraints relating the components. If a state satisfying this constraint is created later, the inconsistent state is then marked consistent and simulated.

### 3.1 A More Complex Model

Figure 3 contains a more complex model of the GIR system. In this model, the rate of insulin production depends upon the amount of glucose. Thus, a feedback loop exists between the two components. The rate of insulin production is a dependent variable with respect to the glucose component. No causal ordering exists between the connected components, so they must be simulated concurrently. The simulation of each sub-model depends upon the behavior of a variable in the other component. The glucose component depends upon I-G and the insulin component depends upon G.

A concurrent simulation alternates between extending the behavioral description of each component. Since the components are simulated concurrently, the behavior of the boundary variables is incomplete during the simulation of an individual sub-model. A truth maintenance system is used to address this

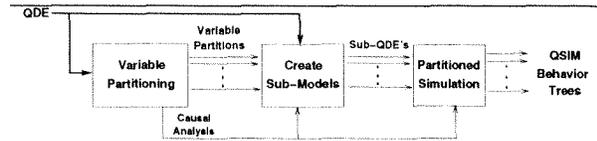


Figure 4: Overview of the Model Decomposition Algorithm

**Variable Partitioning** generates a causal analysis of the model and partitions the variables based upon this analysis. The partitioning identifies tightly connected components within the model to minimize irrelevant temporal correlations.

**Sub-Model Creation** takes the variable partitions as input and generates a sub-model for each partition. The causal analysis is used to determine which causally upstream variables affect each sub-model.

**Partitioned Simulation** determines the order of simulation for the sub-models from the causal analysis and performs the simulation generating a behavior tree for each sub-model.

problem. During the simulation of a sub-model, it is assumed that the simulation of related components is complete and the behavior of the boundary variables is known. If a state is marked inconsistent because the boundary variables do not satisfy the constraints relating the components, a *dependency* is recorded. A dependency records information about the inconsistent state and the condition not satisfied by the incomplete behavioral description of the related component. If this condition is met as the simulation is extended, then the state marked inconsistent is reinserted into the behavior tree and simulated. The details of this algorithm are addressed in the next section.

## 4 The Algorithm

The model decomposition and simulation algorithm is based on the QSIM [Kuipers, 1984, Kuipers, 1994] qualitative reasoning paradigm. It can be adapted to work with other qualitative simulators. It accepts a QSIM QDE as input and generates a set of behavior trees. Each behavior tree corresponds to one of the variable partitions. Figure 4 provides an overview of the three main components of the algorithm: variable partitioning, sub-model creation, and model simulation.

### 4.1 Variable Partitioning

Model decomposition requires a partitioning of the variables into components. In general, this parti-

tioning should combine variables which are tightly connected within the constraint network and separate variables whose behavior is unrelated. The ability to perform the simulation and eliminate temporal correlations is independent of the variable partitioning selected. Model decomposition and simulation provides the same soundness guarantees as a standard QSIM simulation [Kuipers, 1986]. The partition selected only affects the complexity of the resulting simulation and the level of temporal correlation provided.

We use an adaptation of Iwasaki and Simon's [Iwasaki and Simon, 1986, Iwasaki, 1988] causal ordering algorithm to obtain a causal analysis of the model. In certain situations, the partitioning can be derived directly from the causal ordering as in the model in figure 1. In other situations (like the more complicated example in figure 3), selecting the appropriate partitioning can be more difficult. A set of heuristic rules may be required to guide the selection of the partition. A general purpose algorithm to automatically choose the best partitioning is still being developed. Currently, the partitioning is performed by hand. Automatically identifying an effective partitioning will facilitate the integration of these abstraction techniques with automatic model building and query answering systems [Rickel and Porter, 1994].

## 4.2 Creating Sub-Models

A sub-model is created for each variable partition describing the relationships between the variables of the partition. This sub-model, or *sub-QDE*, is used to simulate the component. There are two types of variables contained within each sub-model.

**Within-partition variables** are the variables belonging to the partition. These are the variables of interest for this sub-QDE.

**Boundary variables** are non-partition variables that are related to a within-partition variable through a constraint and are causally upstream or acausally related to that variable. These variables are independent with respect to the sub-QDE. Variables which are causally downstream from a within-partition variable are excluded because these variables act as dependent variables.

A sub-QDE is comprised of the constraints within the main QDE which contain only within-partition and boundary variables. Since the sub-model is concerned with the behavior of the within-partition variables, only those constraints which restrict these

Glucose Component	Insulin Component
<b>Within-Partition Variables</b> G, d/G, G-In, G-Out, Gx1, I-In	<b>Within-Partition Variables</b> I, d/I, I-Out, I-G
<b>Boundary Variables</b> I-G	<b>Boundary Variables</b> I-In
<b>Constraints</b> (M+ G Gx1) (ADD Gx1 I-G G-Out) (ADD d/G G-Out G-In) (D/DT G d/G) (Constant G-In)	<b>Constraints</b> (M+ I I-Out) (S+ I I-G) (ADD d/I I-Out I-In) (D/DT I d/I) (S+ I G)

Figure 5: Partitioning of the More Complex GIR Model

The GIR model in figure 3 is partitioned into a sub-model for the insulin feedback loop and one for the glucose feedback loop. The simulation of each component will derive the behaviors for the within-partition variables. The behavior of the boundary variable is obtained from the simulation of the other component and used to constrain the simulation.

- In the glucose sub-model, I-G is the only boundary variable. It is related to G-Out through an ADD constraint and is causally upstream from G-Out. I-In is not included as a boundary variable since it is causally downstream.
- In the insulin sub-model, I-In is the only boundary variable. It is related to I-Out and d/I through an ADD constraint.

values are included. Figure 5 shows this partitioning for the more complex example discussed above. All constraints within the main QDE are included in at least one sub-QDE. If there is any causal ordering between variables from different partitions which share a constraint, then this constraint is only included in the causally downstream sub-model. Thus, constraints are included in multiple sub-QDE's only when the variables within the constraint are acausally related and belong to different components.

## 4.3 Partitioned Simulation

A separate QSIM simulation is performed for each sub-model. Each simulation generates a state-based behavioral description for the within-partition variables of the sub-model. The description of a boundary variable's behavior generated by an upstream sub-model is used to guide the simulation of a downstream component. Branches due to distinctions in the values of the boundary variables are eliminated through abstraction. Two or more sub-models must be simulated concurrently when a feedback loop exists between these components. The following three sections elaborate on each of these steps.

### 4.3.1 Boundary Variable Behavior Guide

During the simulation of a component, boundary variables are viewed as exogenous variables whose behavior is completely determined by the upstream components to which they belong. The constraints which restrict the behavior of these variables are contained within these upstream sub-models. The constraints which exist between the boundary variables and the within-partition variables in the current sub-model serve to restrict the behavior of the within-partition variables. By restricting the behavior of the boundary variables to match the description provided by the upstream sub-models, no constraining power is lost with regards to the within-partition variables. (i.e. The behaviors generated for the within-partition variables are the same ones produced by the standard QSIM algorithm.)

The boundary variables of a sub-model are grouped into sets according to the components to which they belong. A *guide tree* is provided for each set describing the behavior of the boundary variables within the set. These trees are derived during the simulation of the upstream components.

The guide trees are used to determine the valid successor values for the boundary variables during the simulation. A mapping is maintained between states in the current simulation and matching states within the guide trees. A state  $S_g$  within a guide tree *matches* a state  $S$  in the current simulation if and only if:

- $S_g$  *includes*  $S$  (defined below) with respect to the boundary variables,<sup>1</sup> and
- the predecessor of  $S$  matches either  $S_g$  or the closest ancestor of  $S_g$  that differs from  $S_g$  in the value of at least one current boundary variable.<sup>2</sup>

$S_g$  *includes*  $S$  with respect to a set of boundary variables if the state space described by  $S$  for the boundary variables is a subset of the space described by  $S_g$ . The following requirements must be met.

- $S_g$  must be a time-interval state or  $S$  must be a

<sup>1</sup>These are the boundary variables for the sub-model currently being simulated. They are within-partition variables from the perspective of the upstream models which were used to generate the guide trees.

<sup>2</sup>The guide trees provide a behavioral description for a number of variables besides the current boundary variables. Branches may exist which do not reflect changes in these boundary variables. Thus, the closest ancestor that differs from  $S_g$  in the value of at least one current boundary variable is the predecessor of  $S_g$  from the perspective of these variables.

time-point state. A time-interval state within the current simulation cannot be matched against a time-point state in the guide tree.

- For each boundary variable  $b$  guided by  $S_g$ , the region of the state space described by  $qval(b, S)$  must be equal to or a subset of the region described by  $qval(b, S_g)$ .  $Qval(b, s)$  is the qualitative value of variable  $b$  in state  $s$ .

A new state is marked inconsistent if a valid matching state cannot be identified within each guide tree. This algorithm ensures that every behavior for a boundary variable within the current behavior tree matches a behavior in the appropriate guide tree.

This technique for restricting the behavior of a set of variables based upon a predefined behavioral description has been generalized to allow a modeler to control the behavior of any exogenous variable within a QSIM simulation.

### 4.3.2 Abstracting Boundary Variable Distinctions

A standard QSIM simulation provides a total ordering of events<sup>3</sup> for all variables contained within the model. In a partitioned simulation, each sub-model contains both within-partition and boundary variables. Since each sub-model is only required to provide a behavioral description for the within-partition variables, abstraction is used to eliminate branches caused by distinctions in the values of the boundary variables. This eliminates the temporal correlations between within-partition variables and boundary variables during the simulation of a sub-model.

The abstraction process is performed during the simulation after the successors of a state are computed. Equivalent successor states with respect to the within-partition variables are combined to form a single abstract state. There are two main steps in the creation and simulation of abstract states:

- creating an abstract state from a set of detailed (i.e. non-abstracted) successor states
- computing the successors of an abstract state.

**Creating an Abstract State** An abstract state contains a unique value for each within-partition variable. Qualitative value information for the boundary variables is maintained in the form of a disjunctive list of *sub-states*. Each sub-state contains a complete set of values for the boundary variables. A sub-state is created for each detailed state used to create an abstract state. The information in the sub-states is used to ensure that no constraining power

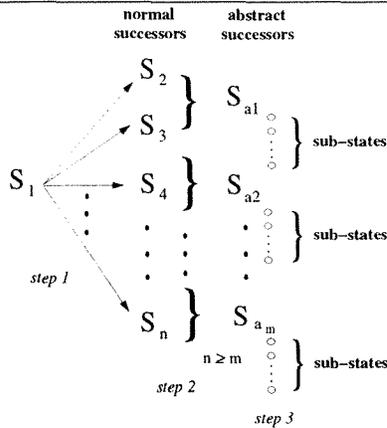


Figure 6: Eliminating Branches through Abstraction

Branches due to distinctions in the boundary variable values are eliminated by combining successors which are equivalent with respect to the within-partition variables. Abstract successor states are created as follows:

1. Create the standard QSIM successor states. The consistency of these successors is determined using the QSIM consistency filters.
2. Using only the consistent successors, create abstract successor states by combining states with equivalent values for the within-partition variables.
3. For each abstract state, create a set of sub-states containing possible qualitative values for the boundary variables. Each sub-state provides a unique set of values for the boundary variables.
4. Replace the original successors with the abstract successors within the behavior tree.

is lost when the successors of an abstract state are created. Figure 6 provides additional information about the state abstraction algorithm.

The boundary variable behavior guide described in the previous section is actually applied to sub-states. A sub-state is marked inconsistent if it cannot be matched against a state within each of the boundary variable guide trees. An abstract state is considered inconsistent if all of its sub-states are marked inconsistent. A mapping is maintained between a sub-state and the matching states within the guide trees.

### Creating Successors of an Abstract State

The abstraction process eliminates distinctions in the boundary variables which are normally used by QSIM when computing a state's successors. The algorithm used to create the successors of a state during a simulation has been modified to handle an abstract state. These modifications retain the

QSIM soundness guarantee as well as the constraining power of a standard QSIM simulation. The successors of an abstract state are computed as follows:

1. Continuity is used to determine the possible successor values for each variable within the model.
  - For within-partition variables, use the unique qualitative value provided within the abstract state to compute the set of valid successor values. This is the normal algorithm used by QSIM for all of the variables.
  - For boundary variables, apply the normal QSIM algorithm to each of the possible values included in the sub-states and then take the union of these possible successor values.
2. Use the standard QSIM algorithm to create and filter potential successor states from this set of possible values. Each state will contain a single qualitative value for each variable within the sub-model (i.e. both within-partition and boundary-variables).
3. For each successor state, perform a continuity test to ensure that the boundary variable values can be reached from at least one sub-state maintained by the abstract state.<sup>3</sup>
4. These successor states are then used to form the abstract successor states as described in the previous section.

The basic QSIM algorithm generates all possible behaviors of the modeled system. Step 1 in this algorithm ensures that this guarantee is retained. The set of possible successor values for each variable is the union of the possible successor values of the individual non-abstracted successor states. Thus, all of the successor states which would have been computed by the standard QSIM algorithm for the non-abstracted states are included in the set of successors for the abstract state.

This abstraction technique also retains all of the constraining power of the standard QSIM algorithm. No additional behaviors are generated due to the elimination of distinctions in the boundary variables via abstraction. Figure 7 shows how each successor of

<sup>3</sup>Since the union of the possible successor values for the boundary variables is used (in step 1), it is possible that these successor values may combine in ways that would not have been possible with the non-abstracted states. This test ensures that a possible successor value that is valid for only one of the sub-states does not combine with a value from a different sub-state.

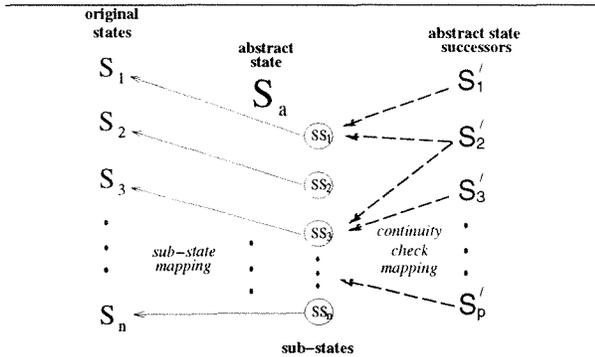


Figure 7: Relationship between an abstract state's successors and the original states.

Original states  $S_1$  through  $S_n$  were combined to form the abstract successor  $S_a$ . A sub-state ( $ss_1$  through  $ss_n$ ) was formed for each of these original states.  $S'_1$  through  $S'_p$  are the successors of the abstract state.

- A one-to-one correspondence (the sub-state mapping) exists between the sub-states attached to an abstract state and the original successors states used to create the abstract state.
- A continuity check (the third step in the algorithm) ensures that each successor of an abstract state can be reached from at least one of the sub-states. One abstract state successor can be a successor for multiple sub-states. This mapping information is not retained during the simulation.

Each abstract state successor can be mapped back to the original states which it would have succeeded if the abstraction were not performed. Thus, no additional successors are generated due to the abstraction process.

an abstract state would have followed from at least one of the original non-abstracted states.

Problems may be encountered when applying extensions of the QSIM algorithm to an abstract state. Some constraining power may be lost due to the inability to apply certain constraints such as the energy filter or some of the quantitative reasoning techniques to the boundary variables. In most cases, this information would have been incorporated into the upstream sub-model that originally determined the behavior of the boundary variable. Otherwise, problems such as this can be addressed by the initial variable partitioning.

### 4.3.3 Simulation Control and Concurrent Simulation

Simulation of a causally downstream component requires information from the upstream sub-model about the behavior of the boundary variables. Once

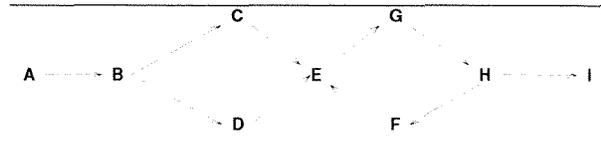


Figure 8: Idealization of Model Decomposition Applied to a Large Model

Model decomposition in conjunction with other abstraction techniques will allow larger models to be simulated producing comprehensible results. These models will be divided into a number of interrelated components. This graph shows an idealization of how the decomposition of a large model would be simulated.

- The nodes of the graph represent components (i.e., a set of variables and constraints) within the model while the arcs are causally directed relationships between components.
- Model decomposition begins simulation with the causally upstream components and then moves through the graph. In this example, component simulation would be performed in the following order:
  - component A
  - component B
  - components C and D (but they do not have to be simulated concurrently)
  - components E, F, G, and H are all simulated concurrently since they form a feedback loop
  - component I

the variables are partitioned, the causal analysis can be used to create a component graph. The nodes in this graph are the sub-models while the links show the direction of causal influence relating these nodes. Some components may be acausally related. The sub-models are simulated in the order dictated by this graph starting with the causally upstream components. Figure 8 shows an idealization of this process for a larger model.

If a feedback loop exists between components (as in figure 3) or if components are acausally related, it is not possible to sequentially order their simulation since they require boundary variable information from each other. In this case, these simulations must be performed concurrently. Since the complete behavior of the boundary variables is not available during the simulation of a sub-model, a truth maintenance system is used to record information about any assumptions which are made.

A concurrent simulation alternates among extending the simulation of each of the related components. A *dependency* is recorded when a state is marked

inconsistent due to the boundary variable behavior guide and the behavior of the boundary variable is incomplete. The dependency records information about the condition required to satisfy the consistency check. As the behavior of the related component is extended, the dependencies are checked to determine if any of the conditions are satisfied by the additional behavioral information. If a dependency is satisfied, then the state which had previously been marked inconsistent is reinserted in the behavior tree and simulated. A dependency is satisfied when a state is created in a related tree fulfilling the conditions which originally caused the dependency to be created (i.e. providing the specified values for the boundary variables).

The dependencies are checked to see if any violations have occurred after each sub-model has been extended one time step. Dependencies are cross checked against each other to ensure that a deadlock condition does not occur (i.e. two states in different components are marked inconsistent because they are each waiting for the other to occur.) Some of the details of the concurrent simulation algorithm are still being developed to ensure that the desired generality is provided.

## 5 Alternative Abstraction Techniques

Model decomposition provides a general framework for performing abstraction during the simulation of a large model. The decomposition of a model is independent of the simulation. Furthermore, the behavior of each component is reasoned about separately from the interactions between components. This architecture allows other decomposition methods to be applied to provide alternative abstraction techniques.

Iwasaki and Bhandari [1988] build upon Simon's [1961] techniques for variable aggregation in dynamic structures. They provide a formal analysis of how variables within a system of equations can be partitioned and aggregated based upon a quantitative analysis of the equations and their roots. Some of these techniques could be extended to address the partitioning and aggregation of variables in a qualitative model. This type of analysis could be used to extend the partitioning techniques currently applied.

Raiman and Williams [1992] describe a method of identifying dominant behaviors which are used to divide the state space of a model into regions in which different behaviors dominate. Order of magnitude reasoning is used to simplify the analysis of

each region. A similar technique could be used when partitioning the variables to simplify the constraint network and minimize the interactions between components. A separate simulation of the model would be performed for each of the regions identified.

Kuipers' [1987] time-scale abstraction groups variables based upon the relative speed of the mechanisms controlling their values. Variables controlled by slower mechanisms are considered constant when simulating the faster mechanisms, while the faster mechanisms are viewed as instantaneous with respect to the slower mechanisms. Time scale information could be used in partitioning the variables to automate this abstraction technique.

## 6 Conclusions and Future Work

State-based qualitative reasoning techniques require a total temporal ordering of events within the behavioral description. This can lead to irrelevant temporal distinctions which increase the complexity of the simulation and obscure the relevant behavior. Before developing heuristics to determine which temporal correlations are relevant to the current task, simulation techniques must be developed to allow for a range of temporal distinctions. Model decomposition and simulation provides an architecture which does not require many of the temporal orderings specified in a traditional qualitative simulation.

Model decomposition has been applied successfully to a number of examples. Two of these examples are discussed in this paper. The simulation techniques used are separate from the method used to decompose the model. Our objective is to ensure that the constraining power of a standard qualitative simulation is retained by these simulation techniques regardless of the variable partitioning selected. This would allow the decomposition algorithm to select a partitioning which highlights distinctions relevant to the current task. Due to the dependency information which must be maintained during a concurrent simulation, model decomposition may increase the computational complexity of a simulation if the granularity of the partitioning is too small. This information must also be taken into account when decomposing a model.

Extensions to the model decomposition and simulation technique are still required to provide the generality that is desired. In particular, this technique has yet to be applied to large, multi-component models with complicated interaction patterns. This research is currently being extended in a number of ways.

- The details of the concurrent simulation algorithm must be developed further to ensure that the variable partitioning selected does not affect the constraining power of the simulation.
- Model decomposition requires a partitioning of the variables into components. This partitioning determines the temporal correlations which are provided by the simulation. Techniques for automatically selecting the optimal partitioning for a given model and task are being investigated.
- Model decomposition provides an architecture which reasons about the behavior of each component separately from the interactions between the components. This architecture allows other decomposition methods to be applied to provide alternative abstraction techniques. In particular, we are interested in incorporating order of magnitude and time-scale information when performing the variable partitioning.
- A complexity analysis of the simulation algorithm must be performed and compared against a standard QSIM simulation to evaluate its effectiveness at reducing the computational complexity introduced by these irrelevant temporal distinctions.

## Acknowledgments

This work has taken place in the Qualitative Reasoning Group at the Artificial Intelligence Laboratory, The University of Texas at Austin. Research of the Qualitative Reasoning Group is supported in part by NSF grants IRI-8904454, IRI-9017047, and IRI-9216584, and by NASA contracts NCC 2-760 and NAG 9-665.

## References

- [Clancy and Kuipers, 1992] D. J. Clancy and B. J. Kuipers. Aggregating Behaviors and Tractable Simulation. In *AAAI Design from Physical Principles Fall Symposium Working Notes*, pp 38-43, Cambridge, MA, 1992.
- [Clancy and Kuipers, 1993] D. J. Clancy and B. J. Kuipers. Behavior Abstraction for Tractable Simulation. In *Proceedings of the Seventh International Workshop on Qualitative Reasoning about Physical Systems*, pp 57-64, 1993.
- [Coiera, 1992] E. W. Coiera. Qualitative Superposition. In *Artificial Intelligence*, 56:171-196, 1992.
- [De Kleer and Brown, 1984] J. De Kleer and J. S. Brown. A Qualitative Physics Based on Confluences. In *Artificial Intelligence* 24:7-83, 1984.
- [Forbus, 1984] K. D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85-168, 1984.
- [Fouché and Kuipers, 1991] P. Fouché and B. J. Kuipers. Towards a Unified Framework for Qualitative Simulation. In *Proceedings of the Fifth International Workshop on Qualitative Reasoning about Physical Systems*, 295-301, 1991.
- [Hayes, 1985] P. J. Hayes. The Second Naive Physics Manifesto. In J. Hobbs and B. Moore (Ed), *Formal Theories of the Commonsense World*. Ablex Publishing Corp.
- [Ironi and Stefanelli, 1993] L. Ironi and M. Stefanelli. A Framework for Building Qualitative Models of Compartmental Systems. Technical Report N. 897, Istituto Di Analisi Numerica, Pavia, Italy, 1993.
- [Iwasaki, 1988] Y. Iwasaki Causal Ordering in a Mixed Structure. In *Proceedings from the Seventh National Conference on Artificial Intelligence, AAAI-88*, pp 313-318, 1988.
- [Iwasaki and Simon, 1986] Y. Iwasaki and H. A. Simon. Causality in Device Behavior. In *Artificial Intelligence*, 19, 1986.
- [Iwasaki and Bhandari, 1988] Y. Iwasaki and I. Bhandari. Formal Basis for Commonsense Abstraction of Dynamic Systems. In *Proceedings from the Seventh National Conference on Artificial Intelligence, AAAI-88*, pp 307-312, 1988.
- [Kuipers, 1984] B. J. Kuipers. Commonsense reasoning about causality : Deriving behavior from structure. *Artificial Intelligence*, 24:169-204, 1984.
- [Kuipers, 1986] B. J. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:289-338, September 1986.
- [Kuipers, 1987] B. J. Kuipers. Abstraction by Time-Scale in Qualitative Simulation. In *Proceedings from the Sixth National Conference on Artificial Intelligence, AAAI-87*, 1987.
- [Kuipers, 1994] B. J. Kuipers. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. Cambridge, MA: MIT Press, 1994. In press.
- [Raiman and Williams, 1992] O. Raiman and B. C. Williams. Caricatures: Generating Models of Dominant Behavior. In *Proceedings of the Sixth International Workshop on Qualitative Reasoning about Physical Systems*, 1-6, 1992.
- [Rickel and Porter, 1994] J. Rickel and B. Porter. Automated Modeling for Answering Prediction Questions: Selecting the Time Scale and System Boundary. To appear in *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 1994.
- [Simon and Ando, 1961] H. A. Simon and A. Ando. Aggregation of Variables in Dynamic Systems. In *Econometrica*, Vol 29, 1961.
- [Williams, 1986] B. C. Williams. Doing time: Putting qualitative reasoning on firmer ground. In *Proceedings of the American Conference on Artificial Intelligence, AAAI-86*, pp 105-113, 1986.

# A Distance Measure for Attention Focusing and Anomaly Detection in Systems Monitoring

Richard J. Doyle

Artificial Intelligence Group  
Jet Propulsion Laboratory  
California Institute of Technology  
Pasadena, CA 91109-8099  
rdoyle@aig.jpl.nasa.gov

## Abstract

Any attempt to introduce automation into the monitoring of complex physical systems must start from a robust anomaly detection capability. This task is far from straightforward, for a single definition of what constitutes an anomaly is difficult to come by. In addition, to make the monitoring process efficient, and to avoid the potential for information overload on human operators, attention focusing must also be addressed. When an anomaly occurs, more often than not several sensors are affected, and the partially redundant information they provide can be confusing, particularly in a crisis situation where a response is needed quickly.

The focus of this paper is a new technique for attention focusing. The technique involves reasoning about the distance between two frequency distributions, and is used to detect both anomalous system parameters and “broken” causal dependencies. These two forms of information together isolate the locus of anomalous behavior in the system being monitored.

## 1 Introduction

Mission Operations personnel at NASA have the task of determining, from moment to moment, whether a space platform is exhibiting behavior which is in any way anomalous, which could disrupt the operation of the platform, and in the worst case, could represent a loss of ability to achieve mission goals. A traditional technique for assisting mission operators in space platform health analysis is the establishment of alarm thresholds for sensors, typically indexed by operating mode, which summarize which ranges of sensor values imply the existence of anomalies. Another established technique for anomaly detection is the comparison of predicted values from a simulation to actual values received in telemetry. However, experienced mission operators reason about more than alarm threshold crossings and discrepancies between predicted and actual to detect anomalies: they may ask whether a sensor is behaving differently than it has in the past, or whether a current behavior may lead to—the particular bane of operators—a rapidly developing alarm sequence.

Our approach to introducing automation into real-time systems monitoring is based on two observations: 1) mission

operators employ multiple methods for recognizing anomalies, and 2) mission operators do not and should not interpret all sensor data all of the time. We seek an approach for determining from moment to moment which of the available sensor data is most informative about the presence of anomalies occurring within a system. The work reported here extends the anomaly detection capability in Doyle’s SELMON monitoring system [4, 5] by adding an attention focusing capability.

Other model-based monitoring systems include Dvorak’s MIMIC, which performs robust discrepancy detection for continuous dynamic systems [6], and DeCoste’s DATMI, which infers system states from incomplete sensor data [3]. This work also complements other work within NASA on empirical and model-based methods for fault diagnosis of aerospace platforms [1, 7, 8, 10].

## 2 Background: The SELMON Approach

How does a human operator or a machine observing a complex physical system decide when something is going wrong? Abnormal behavior is always defined as some kind of departure from normal behavior. Unfortunately, there appears to be no single, crisp definition of “normal” behavior. In the traditional monitoring technique of limit sensing, normal behavior is predefined by nominal value ranges for sensors. A fundamental limitation of this approach is the lack of sensitivity to context. In the other traditional monitoring technique of discrepancy detection, normal behavior is obtained by simulating a model of the system being monitored. This approach, while avoiding the insensitivity to context of the limit sensing approach, has its own limitations. The approach is only as good as the system model. In addition, normal system behavior typically changes with time, and the model must continue to evolve. Given these limitations, it can be difficult to distinguish genuine anomalies from errors in the model.

Noting the limitations of the existing monitoring techniques, we have developed an approach to monitoring which is designed to make the anomaly detection process more robust, to reduce the number of undetected anomalies (false negatives). Towards this end, we introduce *multiple* anomaly models, each employing a different notion of “normal” behavior.

### 2.1 Empirical Anomaly Detection Methods

In this section, we briefly describe the empirical methods that we use to determine, from a local viewpoint, when a

sensor is reporting anomalous behavior. These measures use knowledge about each individual sensor, without knowledge of any relations among sensors.

### Surprise

An appealing way to assess whether current behavior is anomalous or not is via comparison to past behavior. This is the essence of the *surprise* measure. It is designed to highlight a sensor which behaves other than it has historically. Specifically, *surprise* uses the historical frequency distribution for the sensor in two ways: To determine the likelihood of the given current value of the sensor (*unusualness*), and to examine the relative likelihoods of different values of the sensor (*informativeness*). It is those sensors which display unlikely values when other values of the sensor are more likely which get a high *surprise* score. *Surprise* is not high if the only reason a sensor's value is unlikely is that there are many possible values for the sensor, all equally unlikely.

### Alarm

Alarm thresholds for sensors, indexed by operating mode, typically are established through an off-line analysis of system design. The notion of *alarm* in SELMON extends the usual one bit of information (the sensor is in alarm or it is not), and also reports how much of the alarm range has been traversed. Thus a sensor which has gone deep into alarm gets a higher score than one which has just crossed over the alarm threshold.

### Alarm Anticipation

The *alarm anticipation* measure in SELMON performs a simple form of trend analysis to decide whether or not a sensor is expected to be in alarm in the future. A straightforward curve fit is used to project when the sensor will next cross an alarm threshold, in either direction. A high score means the sensor will soon enter alarm or will remain there. A low score means the sensor will remain in the nominal range or emerge from alarm soon.

### Value Change

A change in the value of a sensor may be indicative of an anomaly. In order to better assess such an event, the *value change* measure in SELMON compares a given value change to historical value changes seen on that sensor. The score reported is based on the proportion of previous value changes which were less than the given value change. It is maximum when the given value change is the greatest value change seen to date on that sensor. It is minimum when no value change has occurred in that sensor.

## 2.2 Model-Based Anomaly Detection Methods

Although many anomalies can be detected by applying anomaly models to the behavior reported at individual sensors, robust monitoring also requires reasoning about interactions occurring in a system and detecting anomalies in behavior reported by several sensors.

### Deviation

The *deviation* measure is our extension of the traditional method of discrepancy detection. As in discrepancy detection, comparisons are made between predicted and actual sensor values, and differences are interpreted to be indications of

anomalies. This raw discrepancy is entered into a normalization process identical to that used for the *value change* score, and it is this representation of relative discrepancy which is reported. The *deviation* score for a sensor is minimum if there is no discrepancy and maximum if the discrepancy between predicted and actual is the greatest seen to date on that sensor.

*Deviation* only requires that a simulation be available in any form for generating sensor value predictions. However, the remaining *sensitivity* and *cascading alarms* measures require the ability to simulate and reason with a causal model of the system being monitored.

### Sensitivity and Cascading Alarms

*Sensitivity* measures the potential for a large global perturbation to develop from current state. *Cascading alarms* measures the potential for an alarm sequence to develop from current state. Both of these anomaly measures use an event-driven causal simulator [2, 9] to generate predictions about future states of the system, given current state. Current state is taken to be defined by both the current values of system parameters (not all of which may be sensed) and the pending events already resident on the simulator agenda. The measures assign scores to individual sensors according to how the system parameter corresponding to a sensor participates in, or influences, the predicted global behavior. A sensor will have its highest *sensitivity* score when behavior originating at that sensor causes all sensors causally downstream to exhibit their maximum value change to date. A sensor will have its highest *cascading alarms* score when behavior originating at that sensor causes all sensors causally downstream to go into an alarm state.

## 2.3 Previous Results

In order to assess whether SELMON increased the robustness of the anomaly detection process, we performed the following experiment: We compared SELMON performance to the performance of the traditional limit sensing technique in selecting critical sensor subsets specified by a Space Station Environmental Control and Life Support System (ECLSS) domain expert, sensors seen by that expert as useful in understanding episodes of anomalous behavior in actual historical data from ECLSS testbed operations.

The experiment asked the following specific question: How often did SELMON place a "critical" sensor in the top half of its sensor ordering based on the anomaly detection measures?

The performance of a random sensor selection algorithm would be expected to be about 50%; any particular sensor would appear in the top half of the sensor ordering about half the time. Limit sensing detected the anomalies 76.3% of the time. SELMON detected the anomalies 95.1% of the time.

These results show SELMON performing considerably better than the traditional practice of limit sensing. They lend credibility to our premise that the most effective monitoring system is one which incorporates several models of anomalous behavior. Our aim is to offer a more complete, robust set of techniques for anomaly detection to make human operators more effective, or to provide the basis for an automated monitoring capability.

The following is a specific example of the value added of SELMON. During an episode in which the ECLSS pre-heater failed, system pressure (which normally oscillates within a

known range) became stable. This “abnormally normal” behavior is not detected by traditional monitoring methods because the system pressure remains firmly in the nominal range where limit sensing fails to trigger. Furthermore, the fluctuating behavior of the sensor is not modeled; the predicted value is an averaged stable value which fails to trigger discrepancy detection. See [4, 5] for more details on these previous results in evaluating the SELMON approach.

### 3 Attention Focusing

A robust anomaly detection capability provides the core for monitoring, but only when this capability is combined with attention focusing does monitoring become both robust *and* efficient. Otherwise, the potential problems of information overload and too many false positives may defeat the utility of the monitoring system.

The attention focusing technique developed here uses two sources of information: historical data describing nominal system behavior, and causal information describing which pairs of sensors are constrained to be correlated, due to the presence of a dependency. The intuition is that the origin and extent of an anomaly can be determined if the misbehaving system parameters *and* the misbehaving causal dependencies can be determined. Such information also supports reasoning to distinguish whether sensors, system parameters or mechanisms are misbehaving due to the fact that the signature of “broken” nodes and arcs in the causal graph are distinguishable. See Figure 1.

For example, the expected signature of an anomalous sensor includes the node of the sensor itself and the immediately adjacent arcs corresponding to the causal dependencies that the sensor participates in directly. The intuition is that the actual system is behaving normally so the locus of “brokenness” is isolated to the sensor and the set of adjacent causal dependencies which attempt to reconcile the bogus value reported by the sensor.

The expected signature of an anomalous system parameter also includes nodes and arcs which are downstream in the causal graph from the node corresponding to the system parameter. The intuition here is that the misbehavior, being in the actual system, will propagate.

The expected signature of an anomalous mechanism also includes arcs and nodes causally downstream from the arc corresponding to the mechanism. Once again, the intuition is that the misbehavior is in the system itself, and it will propagate. The way to distinguish this case from the anomalous system parameter case is to examine all input arcs (assuming there are more than one) to the most causally prior node in the “broken” subgraph.

#### 3.1 Two Additional Measures

While SELMON runs, it computes incremental frequency distributions for all sensors being monitored. These frequency distributions can be saved as a method for capturing behavior from any episode of interest. Of particular interest are historical distributions which correspond to nominal system behavior.

To identify an anomalous sensor, we apply a distance measure, defined below, to the frequency distribution which represents recent behavior to the historical frequency distribution representing nominal behavior. We call the measure simply

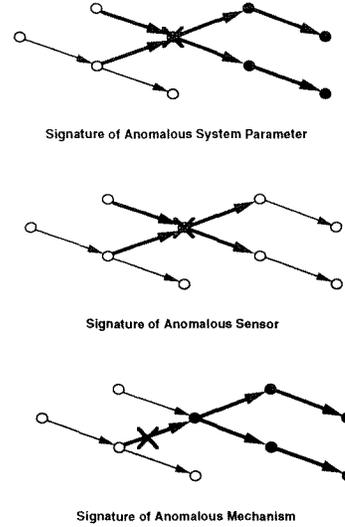


Figure 1: Anomalous System Parameters, Sensors and Mechanisms.

*distance*. To identify a “broken” causal dependency, we first apply the same distance measure to the historical frequency distributions for the cause sensor and the effect sensor. This reference distance is a weak representation of the correlation that exists between the values of the two sensors due to the causal dependency. This reference distance is then compared to the distance between the frequency distributions based on recent data of the same cause sensor and effect sensor. The difference between the reference distance and the recent distance is the measure of the “brokenness” of the causal dependency. We call this measure *causal distance*.

#### 3.2 Desired Properties of the Distance Measure

Define a distribution  $D$  as the vector  $d_i$  such that

$$\forall i, 0 \leq d_i \leq 1$$

and

$$\sum_{i=0}^{n-1} d_i = 1$$

For a sensor  $S$ , we assume that the range of values for the sensor has been partitioned into  $n$  contiguous subranges which exhaust this range. We construct a frequency distribution as a vector  $D_S$  of length  $n$ , where the value of  $d_i$  is the frequency with which  $S$  has displayed a value in the  $i$ th subrange.

If our aim was only to compare different frequency distributions of the same sensor, we could use a distance measure which required the number of partitions, or bins in the two distributions to be equal, and the range of values covered by the distributions to be the same. However, since our aim is to be able to compare the frequency distributions of different sensors, these conditions must be relaxed.

Before defining the other desired properties of the distance measure, we define two special types of frequency distribution. Let  $F$  be the random, or flat distribution where  $\forall i, d_i = \frac{1}{n}$ . Let  $S_i$  be the set of “spike” distributions where  $d_i = 1$  and  $\forall j \neq i, d_j = 0$ .

We seek a distance measure for frequency distributions with the following properties:

*Distance*

$$\forall D_1, D_2, \Delta(D_1, D_2) \geq 0$$

This property merely defines the measure as a distance measure.

*Identity*

$$\forall D, \Delta(D, D) = 0$$

*Symmetry*

$$\forall D_1, D_2, \Delta(D_1, D_2) = \Delta(D_2, D_1)$$

We do not wish to emphasize whether we are comparing recent data to historical data or vice versa, or cause data to effect data or vice versa.

*Distinctness*

$$\forall D_1, D_2, \text{if } D_1 \neq D_2, \text{ then } \Delta(D_1, D_2) > 0$$

The distance measure should distinguish distinct frequency distributions.

*Spike Distinctness*

$$\forall i \neq j, \Delta(S_i, S_j) > 0$$

We wish the set of  $S_i$  to be distinguishable.

*Spike Ordering*

$$\forall i, \Delta(S_i, S_{i+1}) < \Delta(S_i, S_{i+2})$$

The distance measure should preserve the fact that there is an ordering on the bins.

*Spike Equidistance*

$$\forall i \neq j, \Delta(S_i, S_{i+1}) = \Delta(S_j, S_{j+1})$$

There should be no difference in weighting of the spike distributions.

*Spike/Flat Equidistance*

$$\forall i \neq j, \Delta(S_i, F) = \Delta(S_j, F)$$

The difference between any spike distribution and the flat distribution is to be the same.

$$\text{Extrema } \forall D_1, D_2 \forall i, \Delta(D_1, D_2) \leq \Delta(S_i, F)$$

Any spike distribution and the flat distribution are to be considered the most different. All other distributions fall in between.

### 3.3 The Distance Measure

The distance measure is computed by projecting the two distributions into the two-dimensional space  $[f, s]$  in polar coordinates and taking the euclidian distance between the projections.

Define the "flatness" component  $f(D)$  of a distribution as follows:

$$\sum_{i=0}^{n-1} \frac{1}{2} \left| \frac{1}{n} - d_i \right|$$

This is simply the sum of the bin-by-bin differences between the given distribution and  $F$ . Note that  $0 \leq f(D) \leq 1$ . Also,  $f(S_i) \rightarrow 1$  as  $n \rightarrow \infty$ .

Define the "spikeness" component  $s(D)$  of a distribution as:

$$\sum_{i=0}^{n-1} \phi \frac{i}{n-1} d_i$$

This is simply the centroid value calculation for the distribution. The weighting factor  $\phi$  will be explained in a moment. Once again,  $0 \leq s(D) \leq 1$ .

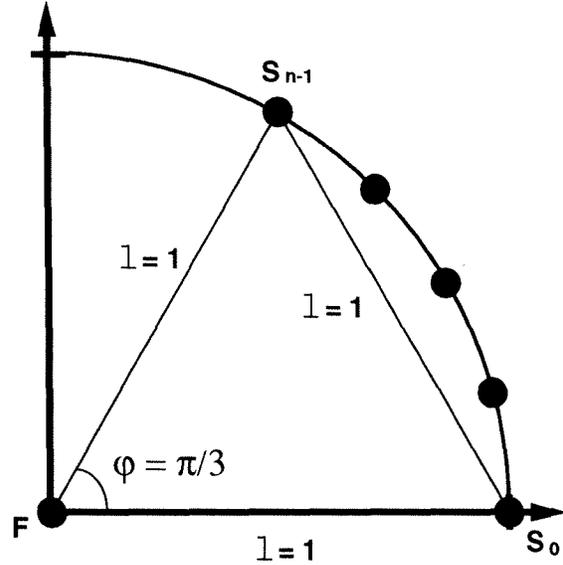


Figure 2: The function  $\Delta(D_1, D_2)$ .

Now take  $[f, s]$  to be polar coordinates  $[r, \theta]$ . This maps  $F$  to the origin and the  $S_i$  to points along an arc on the unit circle. See Figure 2.

By inspection, the *Spike Distinctness*, *Spike Ordering* and *Spike/Flat Equidistance* properties are satisfied. The *Spike Equidistance* property is satisfied because there is no unequal weighting applied in the centroid calculation. The *Distance*, *Identity* and *Symmetry* properties follow from taking the euclidian distance between the projections of the distributions. The *Extrema* property is satisfied by taking  $\phi = \frac{\pi}{3}$ . This choice of  $\phi$  guarantees that  $\Delta(S_0, S_{n-1}) = \Delta(F, S_0) = \Delta(F, S_{n-1}) = 1$  and all other distances in the region which is the range of  $\Delta$  are by inspection  $\leq 1$ .

The *Distinctness* property is not satisfied by the function  $\Delta(D_1, D_2)$ . This is not surprising because the multi-dimensional space arising from the number of bins in a distribution is collapsed to a two-dimensional space  $[f, s]$ . (Consider any two distributions  $D_1, D_2$  with the same even number of bins such that the frequencies in the first  $\frac{n}{2}$  bins and the frequencies in the second  $\frac{n}{2}$  bins both sum to 0.5 in both  $D_1$  and  $D_2$ . These two frequency sets within each distribution may be exchanged and/or permuted without violating  $\Delta(D_1, D_2) = 0$ ). Thoughts on how to address this limitation appear below.

Insensitivity to the number of bins in the two distributions and the range of values encoded in the distributions is provided by the  $[f, s]$  projection function, which abstracts away from these properties of the distributions.

We may note in passing that the distance measure described here may be easily modified to apply to continuous distributions, when theoretical models of the behavior of a system are available. The centroid calculation of the  $s$  component is easily accomplished, and the  $f$  component involves merely the integral of a difference, which may be accomplished numerically if necessary.

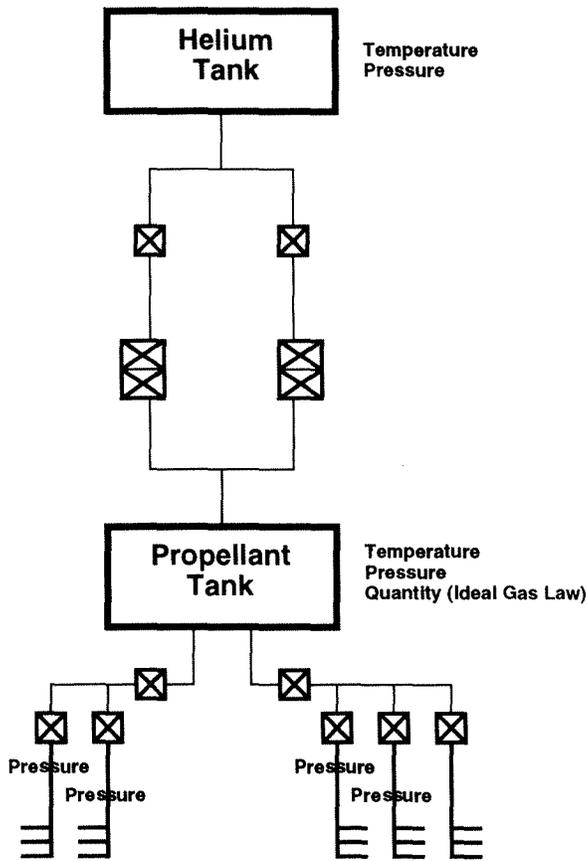


Figure 3: The Forward Reactive Control System (FRCS) of the Space Shuttle.

### 3.4 Results

In this section, we report on the results of applying the distribution distance measure to the task of focusing attention in monitoring. The distribution distance measure is used to identify misbehaving nodes (*distance*) and arcs (*causal distance*) in the causal graph of the system being monitored, or equivalently, detect and isolate the extent of anomalies in the system being monitored.

#### 3.4.1 A Space Shuttle Propulsion Subsystem

Figure 4 shows a causal graph for a portion of the Forward Reactive Control System (FRCS) of the Space Shuttle. A full causal graph for the Reactive Control System, comprising the Forward, Left and Right RCS, was developed with the domain expert.

#### 3.4.2 Attention Focusing Examples

SELMON was run on seven episodes describing nominal behavior of the FRCS. The frequency distributions collected during these runs were merged. Reference distances were computed for sensors participating in causal dependencies.

SELMON was then run on 13 different fault episodes, representing faults such as leaks, sensor failures and regulator failures. Two of these episodes will be examined here; results were similar for all episodes. In each fault episode, and for each sensor, the distribution distance measure was applied to the incremental frequency distribution collected

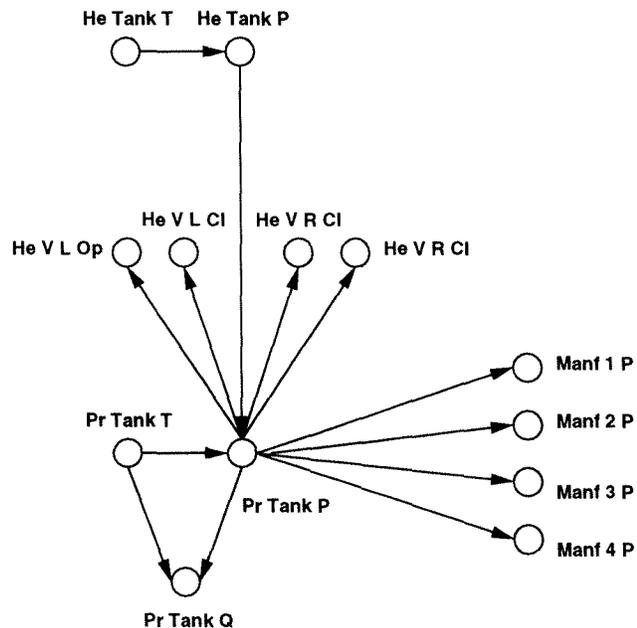


Figure 4: Causal Graph for the Forward Reactive Control System (FRCS) of the Space Shuttle.

during the episode and the historical frequency distribution from the merged nominal episodes. These distances were a measure of the “brokenness” of nodes in the causal graph; i.e., instantiations of the *distance* measure.

New distances were computed between the distributions corresponding to sensors participating in causal dependencies. The differences between the new distances and the reference distances for the dependencies were a measure of the “brokenness” of arcs in the causal graph; i.e., instantiations of the *causal distance* measure.

The first episode involves a leak affecting the first and second manifolds (jets) on the oxidizer side of the FRCS. The pressures at these two manifolds drop to vapor pressure. The dependency between these pressures and the pressure in the propellant tank is severed because the valve between the propellant tank and the manifolds is closed. Thus there are two anomalous system parameters (the manifold pressures) and two anomalous mechanisms (the agreement between the propellant and manifold pressures when the valve is open).

The *distance* and *causal distance* measures computed for nodes and arcs in the FRCS causal graph reflect this faulty behavior. See Figure 5. (To visualize how the distribution distance measure circumscribes the extent of anomalies, the coloring of nodes and the width of arcs in the figure are correlated with the magnitudes of the associated *distance* and *causal distance* scores). An explanation for the apparent helium tank temperature anomaly is not available. However, we note that this behavior was present in the data for all six leak episodes.

The second episode involves an overpressurization of the propellant tank due to a regulator failure. Onboard software automatically attempts to close the valves which isolate the helium tank from the propellant tank. One of the valves sticks and remains open.

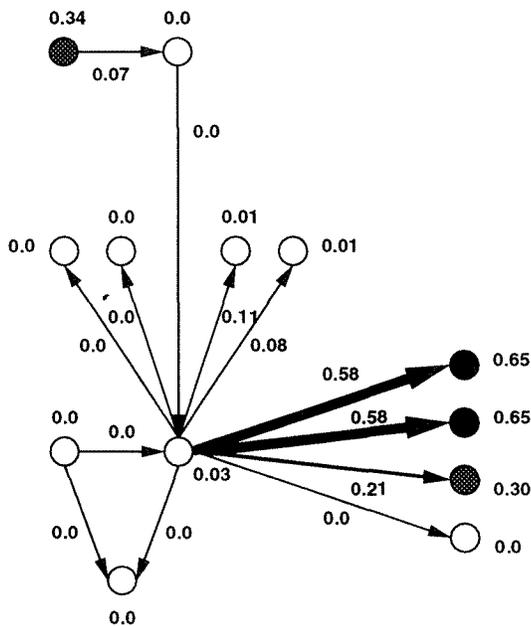


Figure 5: A leak fault.

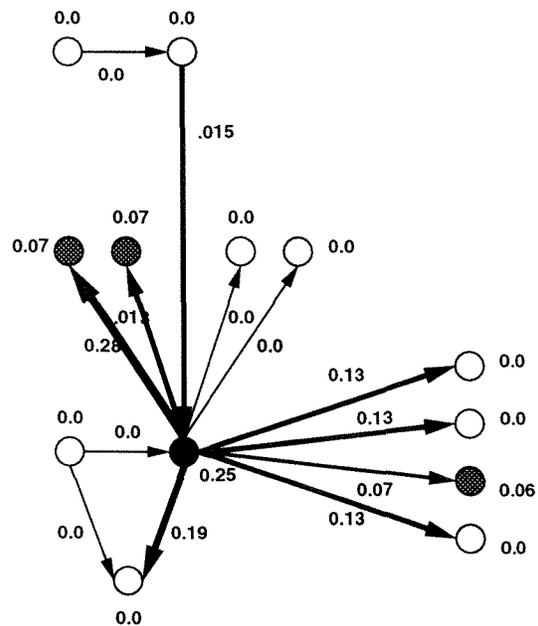


Figure 6: A pressure regulator fault.

The *distance* and *causal distance* measures isolate both the misbehaving system parameters (propellant pressure and valve status indicators) and the altered relationships between the helium and propellant tank pressures and between the propellant tank pressure and the valve status indicators. Over-pressurization of the propellant tank also alters the usual relation between propellant tank pressure and manifold pressures. See Figure 6.

## 4 Discussion

The *distance* and *causal distance* measures based on the distribution distance measure combine two concepts: 1) empirical data alone can be an effective model of behavior, and 2) the existence of a causal dependency between two parameters implies that their values are somehow correlated. The *causal distance* measure constructs a model of the correlation between two causally related parameters, capturing the general notion of constraint in an admittedly abstract manner. Nonetheless, these models of constraint arising from causality provide surprising discriminatory power for determining which causal dependencies (and corresponding system mechanisms) are misbehaving. (In the *distance* measure for detecting misbehaving system parameters, we are simply using the degenerate constraint of expected equality between historical and recent behavior.)

The approach described in this paper has usability advantages over other forms of model-based reasoning. The overhead involved in constructing the causal and behavioral model of the system is minimal. The behavioral model is derived directly from actual data; no offline modeling is required. The causal model is of the simplest form, describing only the existence of dependencies. For the Shuttle RCS, a 198-node causal graph was constructed in a single one and one half hour session between the author and the domain expert.

### 4.1 Monitoring Architecture

The attention focusing capability provided by the *distance* and *causal distance* measures can be combined with the multiple-viewpoint anomaly detection capability already developed in SELMON to construct a general monitoring architecture.

The multiple anomaly measures (including the *distance* and *causal distance* measures, which are anomaly detection measures in their own right) provide continuous anomaly detection capability. All of these measures are normalized to the range  $[0, 1]$  so their sensitivity, individually or collectively, can be fine-tuned for the behavior of particular monitored systems. Whenever a detected anomaly is announced, the extent of the anomaly is isolated by applying the results of the *distance* and *causal distance* measures to the causal graph of the system. If SELMON is supporting a human operator, the operator's attention is focused on the locus of the anomaly, rather than the potentially long and confusing list of the individual manifestations of the anomaly.

Once an anomaly is detected and circumscribed, the operator can once again use the set of anomaly detection measures to gather supplementary, multiple-viewpoint information about the detected anomaly at multiple probe points.

### 4.2 Anomaly Characterization

Most model-based reasoning work has focused on diagnosis, treating monitoring as a "front-end", with discrepancy detection usually chosen as the monitoring technique. Our work suggests modifications to this view.

Monitoring is a complex, subtle and important task in its own right. The most sophisticated diagnosis engine is of limited utility if it is unreliably invoked by a weak anomaly detection module.

The monitoring/diagnosis distinction actually defines two poles of a continuum. At one end is anomaly detection. The goal of anomaly detection is simply to determine if an anomaly

exists. General models of what constitutes an anomaly are utilized, with limited reference to explicit behavior models. Reasoning is local rather than global.

Next in the continuum is anomaly characterization. The goal here is to describe the extent of anomalous behavior. Again, the use of explicit behavior models is limited, but reasoning now encompasses a global view of the system. The anomaly detection capability of SELMON and the attention focusing capability which is the subject of this paper correspond to anomaly detection and anomaly characterization as defined here.

Next comes fault isolation. Reasoning now is refined from anomaly extent to anomaly source. Explicit behavior models may be used, but not explicit fault models.

Finally comes full-fledged fault diagnosis, which includes an explanation of how the proposed fault produced the anomalous behavior. Explicit fault models may be referenced to verify hypotheses.

In actual real-time monitoring practice, operators perform anomaly detection and characterization routinely, and fault isolation when enough information is available to support their reasoning. Fault diagnosis is typically done off-line.

## 5 Future Work

Several issues need to be examined to continue the evaluation of the attention focusing technique based on the distribution distance measure and its utility in monitoring.

We need to understand the sensitivity of the technique to how sensor value ranges are partitioned. Clearly the discriminatory power of the distribution distance measure is related to the resolution provided by the number of bins and the bin boundaries. The results reported here are encouraging for the number of FRCS sensor bins were in many cases as low as three and in no cases more than eight.

We need to understand the suitability of the technique for systems which have many modes or configurations. We would expect that the discriminatory power of the technique would be compromised if the distributions describing behaviors from different modes were merged. Thus the technique requires that historical data representing nominal behavior is separable for each mode. If there are many modes, at the very least there is a data management task. A capability for tracking mode transitions is also required. An unsupervised learning system which can enumerate system modes from historical data and enable automated classification would solve this problem nicely.

We need to understand consequences of the *Distinctness* property not being satisfied by the distribution distance measure. Some distinct distributions are not being distinguished; of more relevant concern is whether or not distributions we wish to distinguish are in fact being distinguished. The judicial introduction of additional components (e.g., the number of local maxima in a frequency distribution) to the distribution projection space  $[f, s]$  may be required to enhance discriminability.

The discriminatory power of the *causal distance* measure might be enhanced by retaining the flatness/spikeness distinction. For many linear functions, different input distributions may map to value-shifted but similarly shaped output distributions. In other words, the spikeness component may vary while the flatness component may be relatively invariant. It

may be possible to distinguish the case where misbehavior is the result of bogus values being propagated through still correctly functioning mechanisms.

It should be possible to describe the temporal (along with the causal/spatial) extent of anomalies by incrementally comparing recent sensor frequency distributions calculated from a "moving window" of constant length with static reference frequency distributions.

## 6 Towards Applications

SELMON is being applied at the NASA Johnson Space Center as a monitoring tool for Space Shuttle Operations and Space Station Operations. Current application efforts include the one for the Propulsion (PROP) flight control discipline reported on here, and one for the Thermal (EECOM) flight control discipline. EECOM wishes in particular to be able to know and reason about how actual orbiter thermal performance differs from predictions generated by an available mathematical model of orbiter thermal performance. An operational SELMON prototype, available starting with the recent Hubble Repair mission is available for evaluation by all flight control disciplines, only requiring that a list of sensors "owned" by that discipline be provided.

At the Jet Propulsion Laboratory, we are looking at the problem of onboard downlink determination for the Pluto Fast Flyby project, now in its early planning phase. The spacecraft will have limited communications bandwidth and it will not be possible to transmit all onboard-collected sensor data. Only eight hours of coverage from the Deep Space Network will be available per week. The challenge is to devise a method for constructing a suitable summary of a week's worth of sensor data guaranteed to report on any anomalies which occurred. The anomaly detection and attention focusing capabilities of SELMON may be well-matched to this task.

## 7 Summary

We have described the properties and performance of a distance measure used to identify misbehavior at sensor locations and across mechanisms in a system being monitored. The technique enables the locus of an anomaly to be determined. This attention focusing capability is combined with a previously reported anomaly detection capability in a robust, efficient and informative monitoring system, which is being applied in mission operations at NASA.

## 8 Acknowledgements

The members of the SELMON team are Len Charest, Harry Porta, Nicolas Rouquette and Jay Wyatt. Other recent members of the SELMON project include Dan Clancy, Steve Chien and Usama Fayyad. Harry Porta provided valuable mathematical and counterexample insights during the development of the distance measure. Matt Barry, Dennis DeCoste and Charles Robertson provided valuable discussion. Matt Barry also served invaluablely as domain expert for the Shuttle FRCS.

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

## References

- [1] K. H. Abbott, "Robust Operative Diagnosis as Problem Solving in a Hypothesis Space," *7th National Conference on Artificial Intelligence*, St. Paul, Minnesota, August 1988.
- [2] L. Charest, Jr., N. Rouquette, R. Doyle, C. Robertson, and J. Wyatt, "MESA: An Interactive Modeling and Simulation Environment for Intelligent Systems Automation," *Hawaii International Conference on System Sciences*, Maui, Hawaii, January 1994.
- [3] D. DeCoste, "Dynamic Across-Time Measurement Interpretation," *8th National Conference on Artificial Intelligence*, Boston, Massachusetts, August 1990.
- [4] R. J. Doyle, L. Charest, Jr., N. Rouquette, J. Wyatt, and C. Robertson, "Causal Modeling and Event-driven Simulation for Monitoring of Continuous Systems," *Computers in Aerospace 9*, American Institute of Aeronautics and Astronautics, San Diego, California, October 1993.
- [5] R. J. Doyle, S. A. Chien, U. M. Fayyad, and E. J. Wyatt, "Focused Real-time Systems Monitoring Based on Multiple Anomaly Models," *7th International Workshop on Qualitative Reasoning*, Eastsound, Washington, May 1993.
- [6] D. L. Dvorak and B. J. Kuipers, "Model-Based Monitoring of Dynamic Systems," *11th International Conference on Artificial Intelligence*, Detroit, Michigan, August 1989.
- [7] T. Hill, W. Morris, and C. Robertson, "Implementing a Real-time Reasoning System for Robust Diagnosis," *6th Annual Workshop on Space Operations Applications and Research*, Houston, Texas, August 1992.
- [8] J. Muratore, T. Heindel, T. Murphy, A. Rasmussen, and R. McFarland, "Space Shuttle Telemetry Monitoring by Expert Systems in Mission Control," in *Innovative Applications of Artificial Intelligence*, H. Schorr and A. Rappaport (eds.), AAAI Press, 1989.
- [9] N. F. Rouquette, S. A. Chien, and L. K. Charest, Jr., "Event-driven Simulation in SELMON: An Overview of EDSE," JPL Publication 92-23, August 1992.
- [10] E. A. Scarl, J. R. Jamieson, and E. New, "Deriving Fault Location and Control from a Functional Model," *3rd IEEE Symposium on Intelligent Control*, Arlington, Virginia, 1988.

## Prediction Sharing Across Time and Contexts

Oskar Dressler and Hartmut Freitag

Siemens AG, Corporate Research & Development, Software and Engineering

Otto-Hahn-Ring 6

D-81730 Munich, Germany

{dressler, freitag}@zfe.siemens.de

### Abstract

Sometimes inferences made at some specific time are valid at other times, too. In model-based diagnosis and monitoring as well as qualitative simulation inferences are often re-done although they have been performed previously. We propose a *new method for sharing predictions* done at different times, thus mutually cutting down prediction costs incurring at different times. Furthermore, we generalize the technique from 'sharing predictions across time' to 'sharing predictions across time *and* logical contexts'. Assumption-based truth maintenance is a form of sharing predictions across logical contexts. Because of the close connections to the ATMS we were able to *use* it as a means for implementation. We report empirical results on monitoring different configurations of ballast water tanks as used on offshore platforms and ships.

### 1. Introduction

Successfully deploying model-based diagnosis systems for complex technical devices ultimately requires an on-line coupling with the artifacts via sensors and actuators. In the field, instead of being manually activated when a malfunction occurs, as a first task an automatic on-line diagnosis system has to decide whether there actually is a diagnosis problem. Does the behavior deviate from the specified normal operation? Only then the diagnosis process will start. *A preceding monitoring phase is required.*

Once the faulty components have been identified, an integrated monitoring and diagnosis system may be allowed to switch back to monitoring mode interpreting the measurements coming from the sensors under the hypotheses that the identified components are broken. Monitoring and diagnosis may thus be interleaved.

Consider the application from (Dressler et al. 1993) depicted in figure 1. A collection of ballast tanks of various sizes is placed at different locations on a ship or offshore platform (the complete system comprises 40 tanks). Depending on load, wind and sea motion, water is pumped into or out of some of the tanks or the sea. This can be a rather time consuming process. For example, in our application on a crane ship, filling three tanks as shown in figure 1 can last up to 1.5 hours. At any time failures may occur potentially causing catastrophic damage. A broken pressure

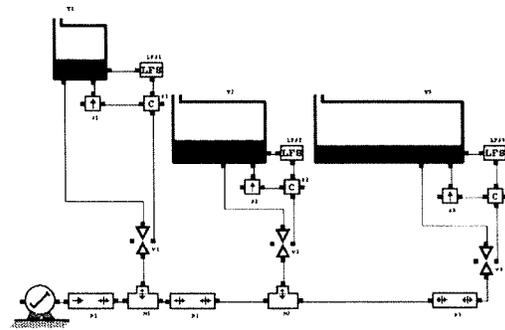


Figure 1. A ballast tank system

sensor, for instance, may enable or disable the automatic closing of a valve, thus causing an overflow or critical unbalance<sup>1</sup>.

A stream of data coming from the system gives us values for pressures, valve status, float switch status and pump activity. This data is processed in a conventional way, such that we can assume to have derivatives for these values, too.

When observed behavior is to be classified as normal or faulty, a model is an invaluable asset. It allows predicting values for system variables, hence generating expectations about behavior when data, even if incomplete, becomes available.

But the purpose of models for monitoring and diagnosis is substantially different. While the former are only needed to *detect* malfunctions, the latter must have enough detail for *localizing* malfunctioning components. Diagnosis systems like DP (Struss 1992) and Magellan (Böttcher, Dressler 1993), however, can use multiple models of different granularity during one and the same diagnosis session. Their diagnosis process starts with coarser models that are suitable for monitoring, too.

In this paper we focus on the prediction task from the viewpoint of dependency-based diagnosis. Dependencies

1. The shipwreck of the polish ferry 'Jan Heweliusz' in January 1993 is suspected to be caused by an incorrect filling of ballast tanks.

are necessary for tracing back contradictory derivations to their origins. This allows diagnosis engines such as GDE (de Kleer, Williams 1987), GDE<sup>+</sup> (Struss, Dressler 1989), Sherlock (de Kleer, Williams 1989), (de Kleer 1991) and others to *first* identify conflicting assumption sets and *then* to generate diagnoses.

We use qualitative models for both, diagnosis and monitoring:

- For consistency-based diagnosis engines they prove to be especially useful; more detailed models become obsolete, when a qualitative abstraction of them has been refuted (Struss 1992). There often is no need to explore further details.
- For monitoring only significant deviations from normal operation are of interest. Using a qualitative model for the normal mode one can capture the complete set of good behaviors instead of just a single one.

When no discrepancies between observed and expected behavior are detected, the empty diagnosis is computed meaning that every component is working correctly. With this in mind, we can view monitoring as ‘*diagnosis without discrepancies*’.

For on-line coupling prediction is necessary at the rate of incoming data. Speed is of prime interest. Allowing a fault to go undetected potentially leads to catastrophe. Ideally, the consistency check, i.e. prediction, should be carried out at the sampling rate of the sensors, say every 10 seconds.

Suppose we are monitoring the process of filling the tanks. Incoming real values are first mapped to their qualitative abstractions, and then fed into the prediction machinery. This we can afford to do every, say two minutes, depending on the cost for running the model. Using qualitative models, most of the time nothing changes in qualitative terms, since different real values are mapped to the same qualitative value. Therefore, we end up making more or less the *same* predictions every two minutes while we would like a higher sampling rate and do prediction with *new* values only.

We propose a new technique for caching and generalizing previously made predictions. It allows carrying over predictions from one time to another. An inference made at a specific time in the past “generalizes” to the same inference made at *all* possible times. There is no need to ever make an inference twice. Consequently, prediction can be much faster when relevant previous inferences have been made. Due to the use of qualitative models this happens all of the time. Intuitively, only when a monitored variable changes its qualitative value, *new* predictions have to be made. In the ballast tanks application (Dressler et al. 1993), the sampling rate during the monitoring phase went down from 2 minutes to 2 seconds !

In the next section we introduce the key concept of ‘*prediction sharing across time*’. Section 3 generalizes our re-

sults by combination with another popular concept called ‘*prediction sharing across contexts*’, commonly known as ATMS (de Kleer 1986). In section 4 delayed consequences of inferences are considered and built into the framework. Finally, in section 5, we discuss empirical results we obtained for different configurations of ballast tanks.

## 2. Prediction Sharing Across Time

### 2.1. Temporally Generic Formulae

The system description  $SD$  is temporally generic in the sense that it describes behavior independent of the specific time at which, for example, a filling process takes place. For example, the model for the normal mode of a valve looks like follows:

```
ok (valve) → valve.status = valve.cmd
ok (valve) → valve.[i1] = - valve.[i2]
ok (valve) → ( valve.status = close → valve.[i1] = valve.[i2] = 0 )
ok (valve) → ( valve.[i1] ≠ 0 → valve.status = valve.cmd = open )
```

The meaning of the variables is: *valve.cmd*: control input, *valve.status*: valve’s state output, *valve.i1*: flow into the left/top end, *valve.i2*: flow into the right/bottom end. Square brackets [.] indicate qualitative variables. The complete models can be found in (Dressler et al. 1993).

In general, inferences made from such descriptions have the form

$$\alpha_1(t) \wedge \dots \wedge \alpha_n(t) \rightarrow \beta(\Delta(t))$$

where  $\alpha_i$  and  $\beta$  are propositional atoms, temporal index  $t$  refers to some specific time and  $\Delta(t)$  denotes another temporal index. We call  $\Delta$  a *delay function*, but allow for negative delays. Therefore, we may draw conclusions about past system variable values, too.

A component  $c$  operating in mode  $m(c)$  is assumed to exhibit the same behavior at every specific time index given that the same values are fed into it. This includes that the delay function is also independent of the specific time. Assuming linear time, we can depict the situation in figure 2.

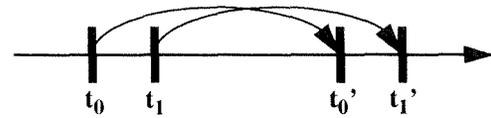


Figure 2. A time-independent delay-function

$$\forall x. [t_1 = t_0 + x \rightarrow \Delta(t_0) + x = \Delta(t_0 + x) = \Delta(t_1)] \wedge \\ [ [\alpha_1(t_0) \wedge \dots \wedge \alpha_n(t_0) \rightarrow \beta(\Delta(t_0))] \\ \rightarrow [\alpha_1(t_0 + x) \wedge \dots \wedge \alpha_n(t_0 + x) \rightarrow \beta(\Delta(t_0 + x))] ]$$

More generally, assuming arbitrary delay functions  $\Delta_1$  and  $\Delta_2$ , ‘independence of an inference of the specific time’ is expressed as:

$$\forall \Delta_1 \forall \Delta_2 \forall t. [ \Delta_2(\Delta_1(t)) = \Delta_1(\Delta_2(t)) ] \wedge \\ [ [\alpha_1(t_0) \wedge \dots \wedge \alpha_n(t_0) \rightarrow \beta(\Delta_1(t_0))] \\ \rightarrow [\alpha_1(\Delta_2(t_0)) \wedge \dots \wedge \alpha_n(\Delta_2(t_0)) \rightarrow \beta(\Delta_1(\Delta_2(t_0)))] ]$$

A formula of the form  $\alpha_1(t) \wedge \dots \wedge \alpha_n(t) \rightarrow \beta(\Delta(t))$

with propositional atoms  $\alpha_1, \dots, \alpha_n$  and  $\beta$  indexed by time  $t$  and  $\Delta(t)$  where the delay function  $\Delta$  adheres to this restriction is called *temporally generic* (or t-generic). Without loss of generality we assume the system description to be a set of t-generic formulae. Please note, that we are not committed to a specific ontology of time like time points or intervals.

For the rest of section 2 and section 3 the delay function considered is identity, i.e. no delays. In section 4 we show how delay is built into the framework.

## 2.2. Temporal Generalization of Single Instance Inferences

From the discussion so far it is clear that some inferences made at a specific time will be valid at other times, too. Thus, there is no need to re-do them when we employ an appropriate caching scheme.

We start from statements like proposition  $\phi$  holding at time  $t_i$ ,  $\phi@t_i$ , called *temporally indexed statements*.

In consistency-based diagnosis we are given a set of them, usually the observations *OBS* made at certain times and the assumptions  $\Pi$  that the corresponding components are working in a specific mode at a certain time. The task is to check the consistency of  $SD \cup OBS \cup \Pi$  where *SD* is independent of time in the sense discussed before.

In qualitative simulation we are given a set of such statements for some initial time  $t_0$ . The task there is to enumerate possible evolutions of the system from this point on. Again, we are dealing with a system description *SD* which is independent of time and observations at a specific time  $t_0$ .

When we predict a value  $\phi$  to hold at  $t_i$ ,  $\phi@t_i$ , the underlying support consists of a set of t-generic formulae  $SD' \subseteq SD$  and a set of sentences  $S$  holding at specific times  $t_1, \dots, t_n$ :  $SD' \cup S \models \phi$

Since we have restricted the delay functions to identity, all of these temporal indices are identical to  $t_i$ , i.e.  $t_i = t_1 = \dots = t_n$ . It follows immediately that the derivation of  $\phi$  can be generalized from the single time index  $t_i$  to sets of time indices.

**Definition:** The *temporal extent* of  $\alpha$ ,  $TE(\alpha)$ , denotes the set  $\{t_i \mid \alpha \text{ holds at } t_i\}$ .

The t-generic formulae in the system description hold at all times, but propositions about observed values etc. are only available at certain times.

**Definition:** A set *GS* of non-universally holding formulae is called *ground support* for  $\phi$  iff there exists  $SD' \subseteq SD$  such that  $SD' \cup GS \models \phi$ .

**Lemma:** If *GS* is a ground support for  $\phi$  then  $\bigcap_{\alpha \in GS} TE(\alpha) \subseteq TE(\phi)$

This means we can generalize a derivation of  $\phi$  at a specific time  $t_i$  to the intersection of temporal extents of  $\phi$ 's support. Whenever all the propositions in *GS* hold at some

time  $t_j \neq t_i$  we know *without re-deriving*  $\phi$  that it holds at  $t_j$ , too.

## 2.3. Symbolic Computation of Temporal Extents

For derived formulae  $\phi$  which do not occur in temporally indexed statements, like e.g.  $\phi@t_{13}$ , the temporal extent can be computed *symbolically* by considering *all* ground support sets for  $\phi$ ,  $GS(\phi)$ .

**Lemma:** Let no explicit temporal statements about  $\phi$  be available. Then

$$TE(\phi) = \bigcup_{S \in GS(\phi)} \bigcap_{\alpha \in S} TE(\alpha).$$

If explicit temporal statements about  $\phi$  are available, we have to add these times.

**Lemma:** If explicit temporal statements about  $\phi$  at times  $t_1, \dots, t_n$  are available, then

$$TE(\phi) = \bigcup_{S \in GS(\phi)} \bigcap_{\alpha \in S} TE(\alpha) \cup \{t_1, \dots, t_n\}.$$

For the propositions  $\alpha$  that may occur in the ground support of derived formulae we introduce symbols  $TE_\alpha$  to represent  $TE(\alpha)$ . These propositions  $\alpha$  are exactly the propositions for which we have temporally indexed statements. The symbols  $TE_\alpha$  are called *temporal base symbols*. In model-based diagnosis this means we are creating these symbols for the observable values and for the modes of components. In qualitative simulation the qualitative values of the initial state are treated in this way.

Using these symbols each atom is labelled with a *unique* symbolic representation of its temporal extent.

**Definition: Temporal Label**

A set of symbol sets,  $\{\{TE_{\alpha_{i1}}, \dots, TE_{\alpha_{in}}\}, \dots, \{TE_{\alpha_{m1}}, \dots, TE_{\alpha_{mk}}\}\}$ , is called *temporal label* of  $\phi$ ,  $TL(\phi)$ , iff

[Correctness]

Each set  $\{TE_{\alpha_{i1}}, \dots, TE_{\alpha_{ij}}\}$  is a ground support for  $\phi$ .

[Completeness]

If  $S$  is a ground support for  $\phi$ , then there exists

$\{TE_{\alpha_{i1}}, \dots, TE_{\alpha_{ij}}\}$  in  $TL(\phi)$  such that

$\{TE_{\alpha_{i1}}, \dots, TE_{\alpha_{ij}}\} \subseteq S$ .

[Minimality]

For no  $i$  and  $j$ ,  $i \neq j$ ,  $\{TE_{\alpha_{i1}}, \dots, TE_{\alpha_{ik}}\}$  is a subset of

$\{TE_{\alpha_{j1}}, \dots, TE_{\alpha_{jm}}\}$ .

[Consistency]

For no  $i$ ,  $\{TE_{\alpha_{i1}}, \dots, TE_{\alpha_{ik}}\}$  is a ground support for  $\perp$ .

## 2.4. Implementation

The similarities to logical labels as used in the ATMS (de Kleer 1986) are apparent and our implementation makes use of this fact. Simply defining the newly introduced symbols  $TE_\alpha$  to be assumptions (in ATMS terminology) suffices. The ATMS will then compute the temporal labels as defined. The relation to the ATMS is very close as we shall see in the next section.

### 3. Prediction Sharing Across Time and Contexts

In section 2 we have seen how from statements such as e.g.  $\alpha@t_1$  and  $\alpha@t_2$  the temporal information is factored out and handled separately from the proposition  $\alpha$ 's content: we compute  $\alpha$ 's temporal label. In systems like TCP (Williams 1986), HEART (Joubel, Raiman 1990), EEP (Guckenbiehl 1991) and TARMS (Holtzblatt et al. 1991) the above statements would be handled as two separate entities. This not only prevents these systems from sharing predictions across time as described in section 2. When these approaches are combined with assumption-based truth maintenance for the purpose of dependency-recording, they are hit by a multiplied exponential blowup: since the two statements are two separate entities, both of them have their own ATMS-label.

Our approach allows for a smooth integration with the ATMS. Actually, we have *used* the ATMS to implement it. After a brief review of the ATMS, we sketch how 'prediction sharing across time' is done. Then we extend the scheme to cover the usual prediction sharing across *logical* ATMS contexts.

#### 3.1. Prediction Sharing Across Logical Contexts

The language of the ATMS (de Kleer 1986) consists of propositional horn clauses called *justifications*

$$\alpha_1 \wedge \dots \wedge \alpha_n \rightarrow \beta.$$

A distinguished subset *ASSM* of the occurring propositional atoms *PROP* is called *assumptions*:  $ASSM \subseteq PROP$ . The set of atoms derivable from a set of assumptions (*environment*)  $E$  is called (*logical*) *context* of  $E$  and denoted by  $cxt(E)$ . All environments which allow deriving the constant  $\perp$  are considered inconsistent.

Reasoning in *multiple contexts* then can be characterized as considering all consistent contexts  $cxt(E)$  of all subsets  $E \subseteq ASSM$  of the given assumptions. All propositions are labelled with the complete set of minimal (w.r.t. set inclusion) consistent environments from which they are derivable. I.e. for a proposition  $p$  its (*logical*) *label* is defined as

$$LL(p) = \{ E \subseteq ASSM \mid E \text{ consistent} \wedge p \in cxt(E) \\ \wedge \forall E' \subset E \ p \notin cxt(E') \}$$

Justifications are used to record the inferences as performed by a problem solver, in our case a predictive engine. The label of a proposition is computed by propagating labels in the network of justifications using basic set operations. By caching inferences as justifications an inference is done once for some context and the results are shared by contexts characterized by superset environments, thus avoiding expensive re-computations.

The labels the ATMS must compute can grow big and hamper larger applications. Focusing on *interesting* contexts (Dressler, Farquhar 1990) avoids this problem while

maintaining the essential properties of assumption-based truth maintenance.

#### 3.2. The ATMS as a Mechanism for Maintaining Temporal Labels

As usual we use the ATMS for recording the inferences made by the problem solver, i.e. the predictive engine. When the antecedents  $\alpha_1, \dots, \alpha_n$  simultaneously hold at some time  $t_i$ , the predictive engine will conclude that  $\beta$  holds, too, given the t-generic formula

$$\alpha_1(t) \wedge \dots \wedge \alpha_n(t) \rightarrow \beta(t)$$

in SD. A justification  $\alpha_1 \wedge \dots \wedge \alpha_n \rightarrow \beta$  is then submitted to the ATMS, and for temporal base symbols ATMS assumptions are created. The following theorem shows that this suffices to compute temporal labels.

**Theorem:** Let *TBS* be the set of temporal base symbols,  $TBS\text{-}ASSM \subseteq ASSM$  be the subset of ATMS assumptions corresponding to temporal base symbols, and  $\Psi: TBS\text{-}ASSM \rightarrow TBS$  be the bijective mapping that associates assumptions with their corresponding symbols. Then

$$TL(\phi) = \{ \{e' \mid e \in E \wedge e' = \Psi(e)\} \mid E \in LL(\phi) \}.$$

All that remains to be done is to record temporally indexed statements. To this end, we create symbols  $EXT - TE_{\alpha_i}$  that denote the enumeration of times where  $\alpha_i$  holds. Each temporal index  $t_i$  actually occurring in an observation like  $X=15@t_i$  is treated as an assumption, too. Then justifying  $EXT - TE_{\alpha_i}$  by temporal indices at which  $\alpha_i$  holds like e.g.

$$t_{17} \rightarrow EXT - TE_{\alpha_i} \text{ and } t_{143} \rightarrow EXT - TE_{\alpha_i}$$

guarantees that the logical label of  $EXT - TE_{\alpha_i}$  enumerates the appropriate times:

$$LL(EXT - TE_{\alpha_i}) = \{ \{t_{17}\}, \{t_{143}\}, \dots \}.$$

Please note that the possibly large number of assumptions for times  $t_i$  does not cause an exponential growth of label sizes.  $LL(EXT - TE_{\alpha_i})$  grows linearly and from  $EXT - TE_{\alpha_i}$  no further propagation is possible.

Querying the system about the temporal extent of an atom  $\phi$  proceeds in two stages. First, a lookup of  $\phi$ 's temporal label is done. Then the union of intersections of the enumerated temporal extents  $EXT - TE_{\alpha_i}$  gives the answer. In a similar way queries about a specific time are processed.

**Lemma:**

$$\phi@t_i \text{ iff } \{t_i\} \in \bigcup_{tenv \in TL(\phi)} \bigcap_{TE_{\alpha_i} \in tenv} LL(EXT - TE_{\alpha_i})$$

Please, note that if temporally indexed statements about  $\phi$  are available, then  $\{TE_{\phi}\}$  is an element of  $TL(\phi)$ .

#### 3.3. Combining Prediction Sharing Across Time with Assumption-based Truth Maintenance

Logical and temporal contexts are orthogonal concepts in the sense that they ought to be combinable without restriction. For example, we might want to state that  $\alpha$  holds at  $t_{17}$  but only under (*logical*) assumptions  $A$  and  $B$ . There are

two principal entry points for this type of statements with logical context qualifications. On the level of temporally indexed statements we need to capture conditions like the one above. On the level of recorded inferences we must be able to express that

$$\alpha_1(t) \wedge \dots \wedge \alpha_n(t) \rightarrow \beta(t)$$

holds regardless of the time  $t$  as before, *but only under (logical) assumptions*, say  $A$  and  $B$ .

Temporally indexed statements can be qualified with logical context information by using justifications like

$$A \wedge B \wedge t_{17} \rightarrow EXT - TE_{\alpha_i} \text{ instead of } t_{17} \rightarrow EXT - TE_{\alpha_i}.$$

Consequently, the logical label  $LL(EXT - TE_{\alpha_i})$  is  $\{\{t_{17}, A, B\}, \dots\}$ . Each environment contains exactly one temporal index assumption while the rest of its assumptions provides the desired logical context. Note, that the usual minimization and consistency maintenance done by the ATMS takes care of redundant and inconsistent information in  $LL(EXT - TE_{\alpha_i})$ .

On the level of recorded inferences the solution is equally simple. Logical assumptions, say  $A$  and  $B$ , are added to the antecedents:

$$A \wedge B \wedge \alpha_1 \wedge \dots \wedge \alpha_n \rightarrow \beta$$

The temporal labels then are relative to the logical context.

**Definition:** *Temporal Label under Assumptions*

A set of symbol sets,  $\{\{TE_{\alpha_{i1}}, \dots, TE_{\alpha_{in}}\}, \dots, \{TE_{\alpha_{m1}}, \dots, TE_{\alpha_{mk}}\}\}$ , is called *temporal label* of  $\phi$  under logical assumptions  $\Theta$ ,  $TL(\phi, \Theta)$  iff

[Correctness]

Each set  $\{TE_{\alpha_{i1}}, \dots, TE_{\alpha_{ij}}\} \cup \Theta$  is a ground support for  $\phi$ .

[Completeness]

If  $S \cup \Theta$  is a ground support for  $\phi$  with temporal base symbols  $S$ , then there exists  $\{TE_{\alpha_{i1}}, \dots, TE_{\alpha_{ij}}\}$  in  $TL(\phi)$  such that  $\{TE_{\alpha_{i1}}, \dots, TE_{\alpha_{ij}}\} \subseteq S$ .

[Minimality]

For no  $i$  and  $j$ ,  $i \neq j$ ,  $\{TE_{\alpha_{i1}}, \dots, TE_{\alpha_{ik}}\}$  is a subset of  $\{TE_{\alpha_{j1}}, \dots, TE_{\alpha_{jm}}\}$ .

[Consistency]

For no  $i$ ,  $\{TE_{\alpha_{i1}}, \dots, TE_{\alpha_{ik}}\} \cup \Theta$  is a ground support for  $\perp$ .

Given this relative notion of temporal label the theorem from 3.2 changes, too.

**Theorem:** Let  $TBS$  be the set of temporal base symbols,  $TBS-ASSM \subseteq ASSM$  the subset of ATMS assumptions corresponding to temporal base symbols,  $\Psi: TBS-ASSM \rightarrow TBS$  the bijective mapping that associates assumptions with their corresponding symbols and  $\Theta$  a set of logical assumptions. Then

$$TL(\phi, \Theta) = \{e' \mid e \in E \cap TBS-ASSM \wedge e' = \Psi(e)\} \mid E \in LL(\phi) \wedge (E \setminus TBS-ASSM) \subseteq \Theta\}$$

The two stage approach to answering queries remains. The evaluation, however, is done relative to the logical context specified as part of the query.

**Lemma:**  $\phi @ t_i$  under assumptions  $\theta$  iff

$$\{t_i\} \cup S \in \bigcup_{tenv \in TL(\phi, \theta)} \bigcap_{TE_{\alpha_i} \in tenv} LL(EXT - TE_{\alpha_i}) \wedge S \subseteq \theta$$

## 4. Delayed Consequences

Delayed consequences are required to model a component such as a valve which receives e.g. an 'open' command and then changes to state 'open' *after some time*. Generally, in qualitative simulation (Kuipers 1986) the interstate behavior, i.e. P- and I-transitions, requires delay. An inference with delayed consequent

$$\alpha_1(t) \wedge \dots \wedge \alpha_2(t) \rightarrow \beta(\Delta(t))$$

is handled specially. No direct translation into a justification is possible. Instead, for the atom  $\beta$  in the delayed consequent a temporal base symbol  $TE_{\beta}$  is introduced and handled as before: an assumption is created and also a symbol  $EXT-TE_{\beta}$  to denote the extensional description of times and logical contexts where  $\beta$  holds. A simple demon mechanism guarantees that, whenever the conjunction of  $\alpha_i$  holds at some  $t_j$  under assumptions  $\Theta$ ,  $\{\Delta(t_j)\} \cup \Theta$  is recorded by  $EXT-TE_{\beta}$ , i.e. an assumption  $\Delta(t_j)$  and the justification

$\Theta_1 \wedge \dots \wedge \Theta_n \wedge \Delta(t_j) \rightarrow EXT-TE_{\beta}$  with  $\Theta_1, \dots, \Theta_n \in \Theta$  are created.

## 5. Empirical Results

We experimented with a variety of ballast tank configurations to provide evidence that we have actually met our goal of reducing prediction costs when relevant inferences have been made previously. In the figures below we show run time and number of necessary new predictive inferences (y-axis) as they develop over time (x-axis). As a measure for new predictive inferences we have chosen the increment of the number of justifications submitted to the system. This, however, can only be an approximation of the really necessary efforts since a single justification may cause a huge amount of label propagation. In the figures the dashed curve shows the new justifications while the solid line indicates prediction time.

The correlation between run time and number of necessary new predictions is apparent. All our experiments on different configurations of ballast tanks show the same pattern: In the beginning the prediction cost (run time) is substantial. No previous predictions have been cached and every possible derivation has to be done explicitly. Later on *when a number of variables change their qualitative value*, prediction cost increases but does not reach the initial cost. Without prediction sharing run time is in the range of the initial cost all of the time !

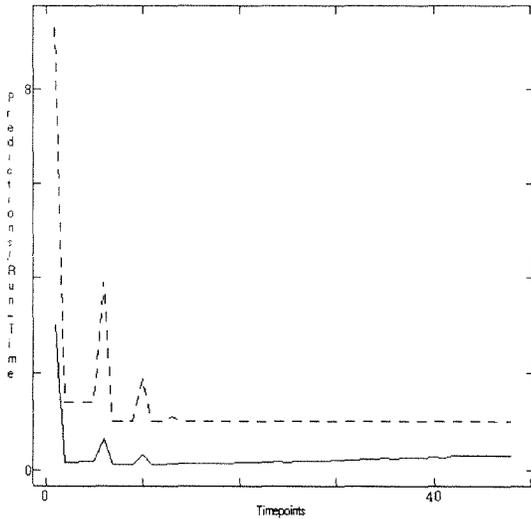


Figure 3. Monitoring the 3-Tank system from figure 1, considering 10 different test vectors occurring at most 10 times. Prediction time for first timepoint: 3.04s, average for additional timepoints: 0.21s

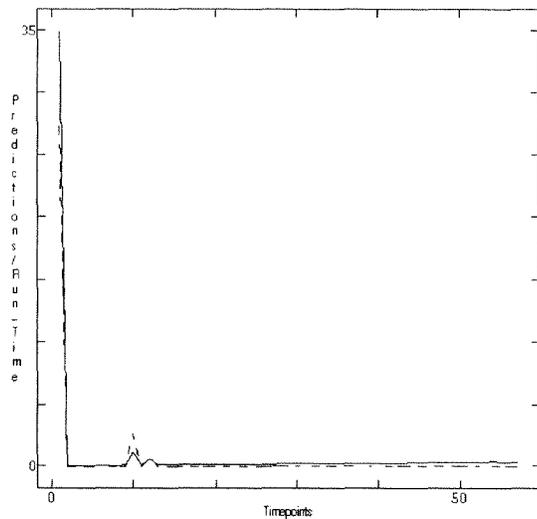


Figure 4. Monitoring a 20-Tank system, considering 10 different test vectors occurring at most 10 times. Prediction time for first timepoint: 36.3s, average for additional timepoints: 3.8s

Currently we attribute the slow, seemingly linear increase of runtime to the monotonically increasing number of justifications and assumptions. Simply handling these structures requires some time. For example, after 57 timepoints the systems maintains 12792 justifications and 405 assumptions for the 20 tanks system. This suggests that we reduce the amount of recorded past data by introducing a time window for relevant data.

**Acknowledgements.** We would like to thank Anton Beschta, Thomas Guckenbiehl, Wera Klein, Michael Montag and Peter Struss for comments on earlier drafts and discussions on the subject of the paper. This work has been supported by the BMFT, project BEHAVIOR, ITW 9001 A9.

## References

- C. Böttcher and O. Dressler. Diagnosis Process Dynamics: Holding the Diagnostic Trackhound in Leash. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann Publishers, 1993.
- J. de Kleer. An Assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.
- J. de Kleer. Focusing on Probable Diagnoses. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 842–848, Anaheim, 1991. Morgan Kaufmann Publishers.
- J. de Kleer and B. C. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32:97–130, 1987.
- J. de Kleer and B. C. Williams. Diagnosis with Behavioral Modes. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1324–1330. Morgan Kaufmann Publishers, 1989.
- O. Dressler and A. Farquhar. Putting the problem solver back in the driver's seat: Contextual control over the ATMS. In M. Reinfrank J.P. Martins, editor, *Truth Maintenance Systems*, pages 1–16. Springer LNAI 515, 1990.
- O. Dressler, C. Böttcher, M. Montag and A. Brinkop. Qualitative and Quantitative Models in a Model-Based Diagnosis System for Ballast Tank Systems. In *Proceedings of the International Conference on Fault Diagnosis (TOOLDIAG)*, pages 397–405, Toulouse, France, April 1993.
- T. Guckenbiehl. The extended episode propagator, version 2-preliminary report. Internal Report, FhG-IITB, 1991.
- L. Holtzblatt, M. Neiberg, R. Piazza, and M. Vilain. Temporal Methods: Multi-Dimensional Modeling of Sequential Circuits. In L. Console, editor, *Working Notes of the International Workshop on Principles of Diagnosis (DX)*, Technical Report RT/DI/91-10-7, pages 111–120. Dipartimento di Informatica, Università di Torino, Italy, 1991.
- C. Joubel and O. Raiman. How Time Changes Assumptions. In *9th European Conference on Artificial Intelligence (ECAI 90)*, Stockholm, 1990.
- B. Kuipers. Qualitative Simulation. *Artificial Intelligence*, 29(3):289–338, 1986.
- P. Struss. Diagnosis as a Process. In W. Hamscher, J. de Kleer, and L. Console, editors, *Readings in Model-based Diagnosis*. Morgan Kaufmann Publishers, 1992.
- P. Struss. What's in SD? Towards a Theory of Modeling for Diagnosis. In W. Hamscher, J. de Kleer, and L. Console, editors, *Readings in Model-based Diagnosis*. Morgan Kaufmann Publishers, 1992.
- P. Struss and O. Dressler. Physical Negation – Integrating Fault Models into the General Diagnostic Engine. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1318–1323. Morgan Kaufmann Publishers, 1989.
- B. C. Williams. Doing Time: Putting Qualitative Reasoning on Firmer Ground. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Philadelphia, Penn., August 1986.

# Topology-based Spatial Reasoning

Boi Faltings  
Artificial Intelligence Laboratory (LIA)  
Swiss Federal Institute of Technology (EPFL)  
IN-Ecublens, 1015 Lausanne, Switzerland  
FAX: +41-21-693-5225, e-mail: [faltings@lia.di.epfl.ch](mailto:faltings@lia.di.epfl.ch)

April 18, 1994

## Abstract

Determining the possible positions and motions of objects based on their geometry is fundamental to reasoning about the physical world, for example in robot planning or mechanical design. Existing techniques are based on the geometry of object boundaries and limited in the degrees of freedom they allow, or in the object shapes that can be considered.

In this paper, I present a technique which is based on the *topology* of objects and space, and does not require a closed-form representation of object boundaries. The technique is simpler, more efficient and more robust than techniques based on geometry. However, it is limited to objects which can be represented as the union of convex subparts.

## 1 Introduction

Spatial reasoning about possible motion and kinematics of physical objects is fundamental for reasoning about the physical world. Figure 1 shows an example of 5 different positions of a designated *moving object* in the free space left by a set of *obstacles*. The spatial reasoning task I address in this paper is to determine a complete vocabulary of all legal qualitative positions of the moving object, called *places* ([3]), and the connectivity between these positions. Using the place vocabulary, it is for example possible to classify positions *A*, *B*, *C* and *D* as belonging to different places, and to show that positions *D* and *E* are connected, but *A* and *C* are not. For example, a place vocabulary can be used to solve the *piano-*

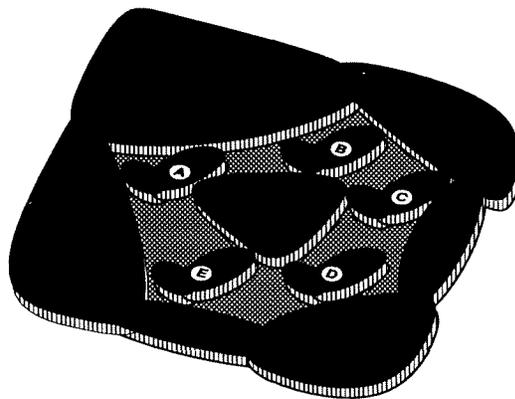


Figure 1: *Five qualitatively different positions of an object in a space of obstacles. My technique predicts the existence as well as the connectivity of the positions on the basis of the geometry of the obstacles and the moving object.*

*movers problem*, which has been extensively studied in the literature ([10, 6]).

Existing techniques for computing place vocabularies (among many others, [9, 3]) first model the object surfaces using equations, transfer these into *constraints* in a *configuration space* ([11]), and then obtain a region structure by algebraic methods. However, complexity imposes limitations on the allowable object shapes and their freedom of motion. Thus, no algorithm has been known which can compute place vocabularies for curved objects as shown in Figure 1. Furthermore, modeling object geometry using algebraic constraints poses several problems:

- fitting: shapes are observed as bitmaps and their automatic modeling with algebraic curves is not solved in a reliable manner.
- brittleness: small numerical errors can result in grossly incorrect topologies. Furthermore, errors propagate even to parts of the place vocabulary which were otherwise computed correctly.
- adaptability: solutions for similar situations cannot be reused.

An alternative to modelling objects algebraically by their boundaries is to model their topology as a set of regions. In this paper, I present a novel method for computing place vocabularies based on such an object model and principles of algebraic topology. Similarly to the work of Cui, Cohn and Rendell ([2]), reasoning is based on overlaps between regions. However, our work is different in that it takes into account the shapes and dimensions of rigid objects and applies algebraic topology to infer additional information. I will first define the method for the case of two-dimensional objects with two degrees of freedom, and then show how it generalizes to rotations. I have not yet investigated the generalization to three dimensions.

## 2 Object and constraint representation

Objects are modelled as the union of convex *parts*, and one of the objects is identified as a *moving object*. A *configuration* is a particular position and orientation of the moving object and can be defined as a

point in a *configuration space* ([11]), which is spanned by these parameters. Configuration space consists of blocked configurations where the moving object would overlap others, called *blocked space*, and its complement of legal positions, called *free space*. Each possible overlap between parts of the moving object and a fixed object defines a configuration space region (*c-region*) of illegal configurations, called an *obstacle*. Blocked space is the union of all obstacles.

In order to be able to refer to positions of the moving object within free space, we define a set of convex regions called *cavities* which completely cover the empty space left by the fixed objects. The choice of these regions is arbitrary: they can be understood as defining a quantity space of positions of the moving object. For example, the situation of Figure 1 can be represented by the regions shown in Figure 2. The possible overlaps between a part of the moving object and a cavity defines a *c-region* which we call a *bubble*. Note that in contrast to blocked space, free space is only a *subset* of the union of all bubbles.

When the moving object is placed in an arbitrary configuration  $C$ , it overlaps some parts and cavities. In configuration space, this means that the configuration falls within a certain combination of corresponding obstacles and cavities, and we call such a combination an *environment*:

**Definition 1** *An environment is a combination of c-regions, and denotes the set of configurations where the moving object exhibits at least the overlaps corresponding to them (and possibly others as well).*

The environments corresponding to two sample configurations are shown in Figure 2.

**Properties of c-regions** Because they represent configurations of overlap between convex regions, *c-regions* have the following properties which will be important for computing the topology of a combination of *c-regions*. Note that in this section, we consider only *translations*. Rotations will be discussed later in the paper.

**Theorem 1** *Every c-region formed by two convex pieces or cavities  $A$  and  $B$  is a simply connected region.*

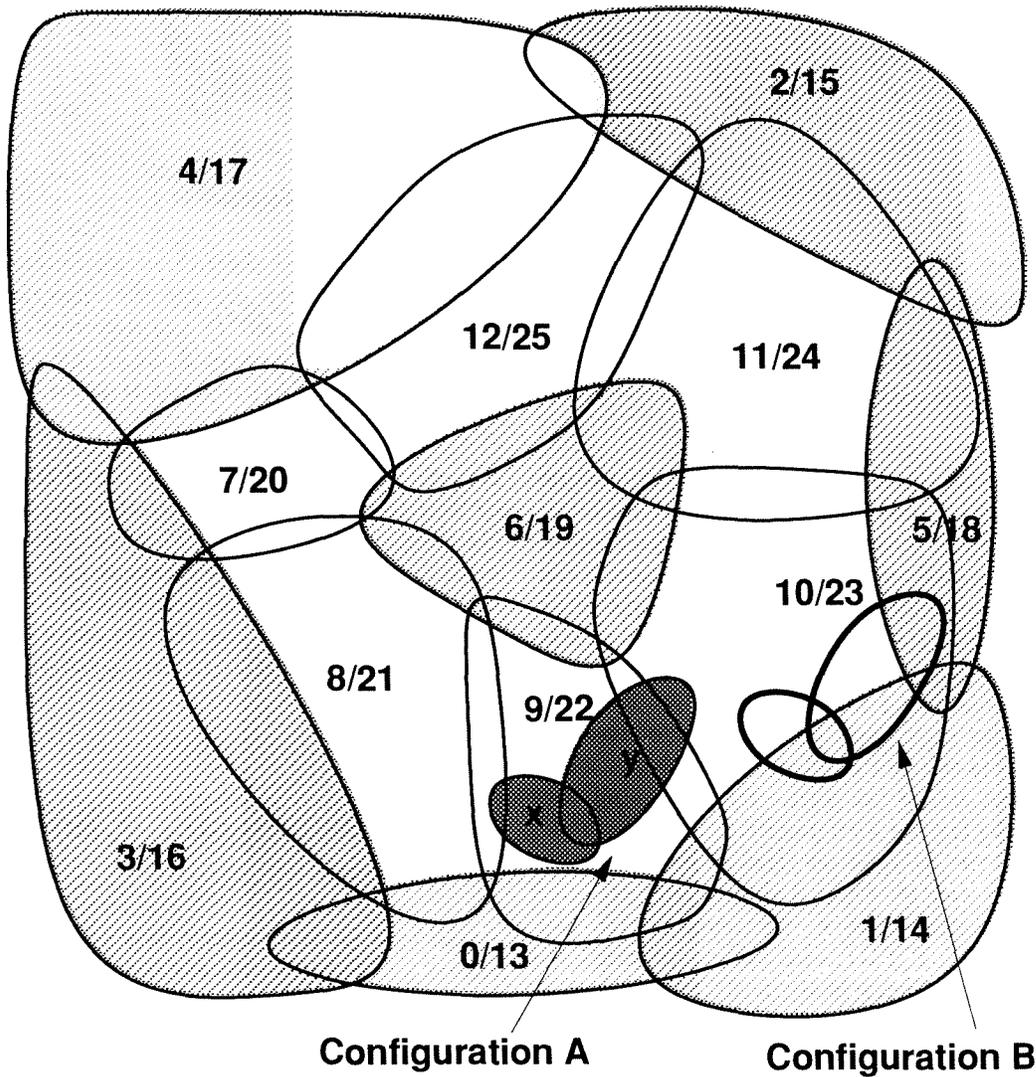


Figure 2: *Input representation of a situation. Pieces, shown in grey, and cavities, shown in white, are labelled by a combination of two numbers which number the c-regions of overlap with the two pieces  $x/y$  of the moving object, shown in grey. The shown legal configuration falls within the environment  $E_1 = \{8, 9, 22, 23\}$ . The configuration shown as an outline falls within the environment  $E_2 = \{1, 10, 14, 18, 23\}$ .  $E_1$  contains only bubbles and is thus legal, whereas  $E_2$  contains several obstacles and is illegal.*

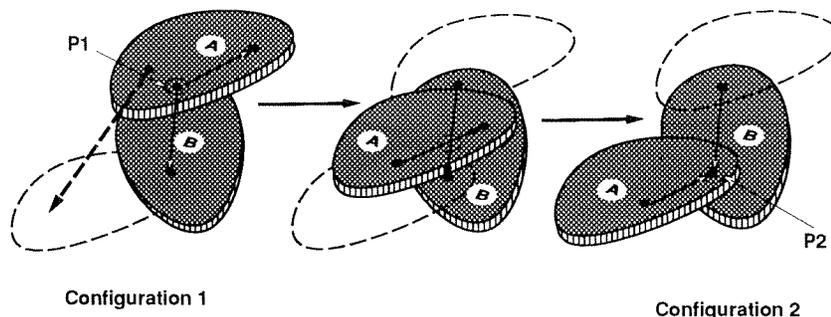


Figure 3: We identify points  $P1$  and  $P2$  as selected overlapping points in the two shown configurations of overlap between  $A$  and  $B$ , and denote their positions on  $A$  and  $B$  by subscripts. The two configurations can be transformed into each other by moving object  $A$  in a straight line as shown. In any intermediate position, there is an overlap between a point on  $A$  which falls on  $\overline{P1_A P2_A}$  and a point on  $B$  which lies on  $\overline{P1_B P2_B}$ . Consequently, the path where  $A$  moves in a straight line with respect to  $B$  lies completely within the  $c$ -region.

**Proof:** Consider an arbitrary pair of configurations 1 and 2. Figure 3 shows a proof that there always exists a path entirely within the  $c$ -region which connects the two configurations, and thus the  $c$ -region is connected. Now consider a cycle of configurations within the  $c$ -region. The cycle can be approximated as a sequence of configurations which can be transformed into each other using the transformation shown in Figure 3. On each part, the sequence of transformations of the selected point forms a polygon whose edges all fall within the part. By shortening each translation by the same proportion  $\epsilon$ , the corresponding sides of the polygon are also shortened by the same fraction and we obtain a similar polygon which is smaller by a factor of  $\epsilon$ . By repeated and continuous application of this contraction operation, the polygon and thus the cycle of transformations can be contracted into a single point. Since any closed path within it can be transformed into a single point, the  $c$ -region is simply connected.

QED

Computing the topology of a set of regions requires consideration of their intersection. For this, the following property is important:

**Theorem 2** Every intersection of  $k$   $c$ -regions  $c_1, c_2, \dots, c_k$  is a simply connected region.

**Proof:** In both configurations  $A$  and  $B$ , let there be an overlap between the  $k$  pairs of pieces which define the  $c$ -regions. The proof of Theorem 1 shows that the straight line translation between  $A$  and  $B$  maintains each of the overlaps and thus falls within the intersection of the  $k$   $c$ -regions. By the same reasoning as before, this intersection is thus a simply connected region.

QED

### 3 Environments, Cliques and Places

Recall that an environment is a combination of obstacles and bubbles. An environment  $E$  is *feasible* if there actually exists a configuration which falls *only* within  $E$ , and in particular does not intersect any other  $c$ -regions outside of  $E$ . A feasible environment thus represents a set of configurations where the moving object overlaps exactly the pieces and cavities designated by  $E$ . An environment is called *maximal* if there is no feasible environment that  $E$  is a proper subset of, and *minimal* if there is no feasible environment which is a proper subset of  $E$ . Environments which are feasible and contain only bubbles are regions of legal configurations of the moving object, and make up the *places* in the place vocabulary.

The principle underlying the method I present is

the following:

A place  $P = \{B_1, B_2, \dots, B_k\}$  is part of the place vocabulary only if

- it is an environment consisting only of bubbles, and
- removing the intersection of  $\{B_1, B_2, \dots, B_k\}$  leaves a "hole" in the configuration space.

which is true because if there is a configuration which falls only in  $P$ , removing  $P$  will remove the point from the configuration space, thus creating the hole. This principle allows using the topological notions of connectedness for computing place vocabularies.

An environment is the intersection of a set of  $c$ -regions and thus by Theorem 2 simply connected. A *place* represents a set of feasible positions of the moving object, i.e. positions where it does not overlap any obstacles. Thus, we define:

**Definition 2** *A place is a feasible environment which contains only bubbles.*

The method for computing topologies is based on a theorem which allows to decide the existence of regions where any number of parts overlap from knowledge of the existence of all regions where only  $d + 1$  parts overlap, where  $d$  is the dimensionality of the configuration space. More specifically, we define:

**Definition 3** *The  $n$ th-order region graph  $G_n(R)$  of a set of  $c$ -regions  $R$  is the hypergraph whose nodes are the  $c$ -regions in  $R$  and whose arcs are all intersections of up to  $n$   $c$ -regions in  $R$ .*

and

**Definition 4** *An  $n$ -clique of a hypergraph  $G$  is a set  $N = \{n_1, n_2, \dots, n_k\}$  of nodes such that any subset of  $n$  nodes  $\in N$  is an arc of  $G$ .*

and have the theorem:

**Theorem 3** *Let  $G$  be the  $(n+1)$ -th order region graph of a set  $R$  of  $c$ -regions in an  $n$ -dimensional configuration space. The environment  $E$  consisting of an overlap of the set of  $c$ -regions  $N$  exists if and only if  $N$  is an  $(n+1)$ -clique of  $G$ .*

Proof: The "only if" direction is obvious, since an intersection of all regions in  $N$  automatically implies an intersection of all subsets of  $N$ . The "if" direction is proven inductively in following way. Obviously the theorem is true for  $|N| = n + 1$ . Assume that it holds for some  $|N| = l \geq (n + 1)$ , and let  $N = \{n_1, n_2, \dots, n_{l+1}\}$ . Then all intersections of subsets of  $l$  regions exist, so all the Čech-cohomology groups of  $N$  up to degree  $l - 1$  are identical to those of an intersection of  $l + 1$  identical unit balls in  $\mathcal{R}^n$ . But as a consequence of the Alexander duality, all Čech-cohomology groups of degree  $\geq n$  are identically zero. Thus, Čech-cohomology of the  $N$  is entirely the same as that of an intersection of  $l + 1$  unit balls, and the intersection of all regions is non-empty.

QED

More details about Čech-cohomology can be found in textbooks on algebraic topology, for example [8] or [7]. Theorem 3 is a generalized version of Helly's theorem ([1]), which states the same relation but for convex sets only.

For two dimensional configuration spaces, Theorem 3 implies that the set of environments formed by the  $c$ -regions is given by the 3-cliques of the region graph  $G$ . The region graph can be obtained by exhaustive testing of all possible intersections between triples of obstacles and bubbles: in the example of Figure 2, the region graph contains 26 nodes and 469 hyperarcs linking sets of three nodes.

Because the region graph is a type of intersection graph, and each 3-clique corresponds to an actual region of space, their number is limited to grow only polynomially in the number of nodes. Thus, the set of *maximal* 3-cliques of  $G$  can be determined by exhaustive search without extensive complexity; in the example we find a total of 34 maximal cliques. All these maximal cliques are feasible: configurations in them cannot be part of any other regions as otherwise the clique would not be maximal. However, most are not places since they are not composed exclusively of bubbles.

**Computing feasible environments** Many feasible environments are non-maximal cliques, i.e. subsets of maximal cliques, but not all subsets of maximal cliques are feasible. In order to decide whether a given non-maximal clique is feasible, we make use of the criterion mentioned in the introduction,

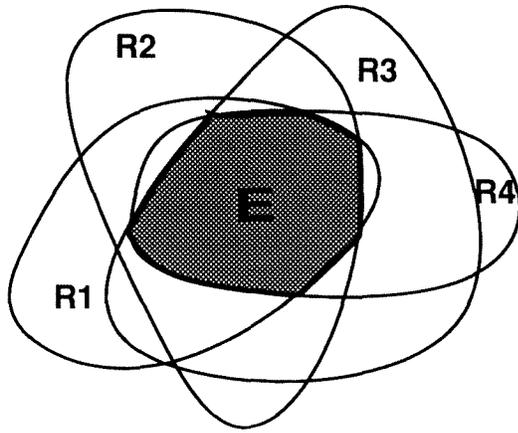


Figure 4: *An example of an environment:  $E$  is formed by the intersection of c-regions  $R1$  through  $R4$ .*

namely that an environment is feasible if removing its c-regions changes the connectivity of configuration space.

More precisely, a set of c-regions  $E$  is an environment if and only if it is a subset of a maximal clique. An environment is an intersection of c-regions and by Theorem 2, its topology is always that of a simply connected region. Figure 4 shows an example of an environment formed by the intersection of c-regions  $R1$  through  $R4$ .

We now consider the set of c-regions  $O(E)$  overlapping  $E$ , called the *overlap set of  $E$* , is the union of maximal cliques containing  $E$ , restricted to the points within  $E$ :

$$O(E) = \left( \bigcup_{s \in S(E)} s \right) \cap E$$

where  $S(E) = \{s | s \text{ is a maximal clique and } s \supset E\}$ .

In a two-dimensional configuration space, the overlap set of a set  $E$  can have the following topologies, illustrated by Figure 5:

- a) **simply connected:** c-regions  $R5$  and  $R6$  cover all configurations in the environment, consequently removal will not create a hole, and the environment is **not feasible**.

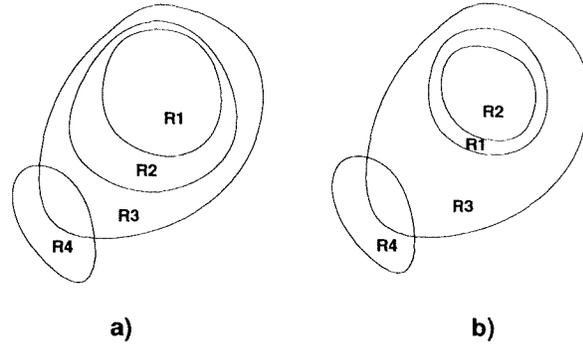


Figure 6: *Two indistinguishable situations with different environments.*

- b) **simply connected:** c-region  $R5$  covers some of the configurations in the environment, removal will create a hole, and the environment is **feasible**.
- c) **multiply connected:** c-regions  $R5$ - $R8$  form a cycle which leaves an opening when the environment is removed, and thus the environment is **feasible**.
- d) **not connected:** c-regions  $R5$  and  $R6$  are not connected, removal of the environment leaves an opening, and thus it is **feasible**.

The topology of the overlap can be computed using a decomposition into elementary spaces, as described later. The method for computing the feasible non-maximal environments is based on the following theorem:

**Theorem 4** *If the topology of the overlap set  $O(E)$  is different from that of the environment  $E$  itself,  $E$  is feasible.*

**Proof:** since  $O(E)$  has a different topology from  $E$ , and  $O(E) \subset E$ ,  $E$  must contain points which are not in  $O(E)$ . Thus, there are some points in  $E$  which are not in any other c-region, and thus  $E$  is feasible.

QED

This theorem leaves ambiguous the case where an environment and its overlap set have identical topologies, in this case simply connected. In fact, the formulation of region intersections does not contain enough information to distinguish whether or not such an environment is feasible. Consider the example shown

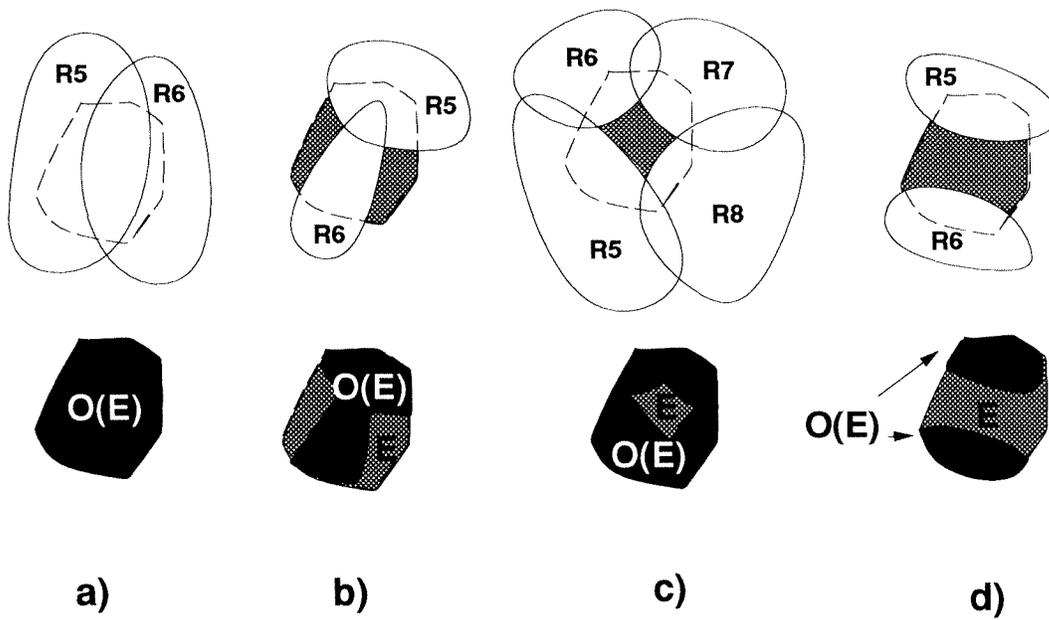


Figure 5: Let  $E$  be the environment consisting of the intersection of the 4 c-regions  $R1-R4$ , shown in grey. Depending on the topology of the c-regions overlapping  $E$ , it may (b, c and d) or may not be (a) feasible.

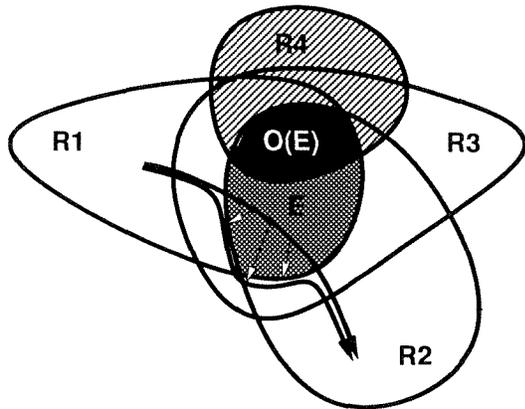


Figure 7:  $E$  is formed by the intersection of  $R1$ ,  $R2$  and  $R3$  and overlapped by  $R4$ .  $O(E)$  is simply connected, but does not completely cover  $E$ . Any path through  $E$  may then be transformed into a path through subset environments, in this case  $\{R1, R3\}$ ,  $\{R3\}$ ,  $\{R2, R3\}$ . Thus, omitting  $E$  from the place vocabulary does not cause any change in connectivity.

in Figure 6. Cases a) and b) both have the same two maximal 3-cliques:  $\{R1, R2, R3\}$  and  $\{R3, R4\}$  and are thus indistinguishable in the input information given to my algorithm. However, in case a) the environment  $\{R2, R3\}$  is feasible and the environment  $\{R1, R3\}$  is not, whereas in case b) the opposite is true. Note, however, that all environments in question are subsets of the single environment  $\{R3\}$ , which is feasible by the criterion given above (overlap set not connected). Thus, both situations are correctly modelled by the single place  $\{R3\}$ , and the ambiguous environments can be ignored.

It remains to show that ignoring these environments does not result in incorrect connectivity of the space. To do this, we show that any path through an environment whose overlap set is simply connected can be transformed into an equivalent path through one of its subsets, as shown in Figure 7. We begin by showing the following lemma:

**Lemma 1** *Let  $E$ ,  $O(E)$  be simply connected and  $E - O(E)$  consist of the components  $c_1, c_2, \dots$ . Then each  $c_i \cap Bnd(E)$ , where  $Bnd(E)$  is the boundary of  $E$ , is connected.*

**Proof:** assume there was a component intersecting  $Bnd(E)$  in two disjoint pieces. Then the boundaries of  $E - O(E)$  connecting the endpoints of the pieces delimit two disjoint pieces of  $O(E)$ , and thus  $O(E)$  is not simply connected.

QED

As a consequence, we have the following theorem:

**Theorem 5** *Let  $E$  be an environment such that both  $E$  and its overlap set  $O(E)$  are simply connected. Then any legal path through  $E - O(E)$ , the part of  $E$  not covered by  $O(E)$ , can be continuously transformed into a legal path through subsets of  $E$ .*

**Proof:** Let  $c$  be a component of  $E - O(E)$ . A path through  $c$  must cross the boundary of  $E$   $Bnd(E)$  an even number of times. Since both  $c$  and  $c \cap Bnd(E)$  are simply connected, the crossings can all be continuously contracted into a single point and the path thus removed from  $c$ .

QED

Thus, it is correct to consider as feasible only those environments which are either maximal or whose overlap sets are not simply connected, and this is the rule my algorithm uses.

**Computing the place vocabulary** Those feasible environments which do not contain any obstacles, i.e. are made up purely of bubbles, are environments in which there is no overlap between moving object and fixed pieces and make up the places in the place vocabulary. In the example given earlier, there are 81 environments which are feasible by the topology of their overlap set or by being maximal, of which 9 do not contain any obstacles and thus make up the places. They are shown in Figure 8.

Transitions between places are possible at boundaries of  $c$ -regions. In particular, a transition from  $P1$  to  $P2$  can correspond to either entering or leaving a  $c$ -region which distinguishes  $P1$  from  $P2$ . Thus,  $P1$  is adjacent to  $P2$  whenever  $P1$  is a proper subset of  $P2$ , or  $P2$  is a proper subset of  $P1$ . This generates the adjacency relations shown in Figure 8.

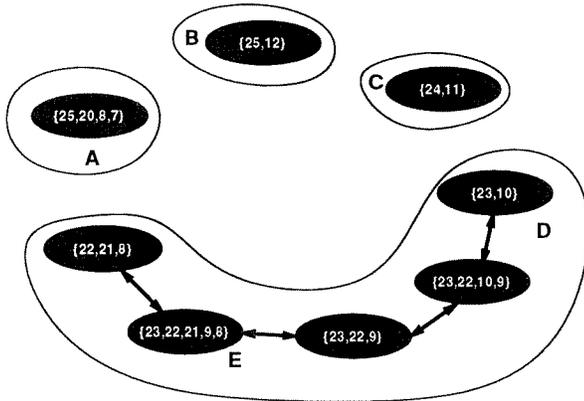


Figure 8: Place vocabulary for the example. The numbering of the regions refers to Figure 2, the letters to configurations in Figure 1.

## 4 Compositional computation of topology

For deciding whether an environment is feasible or not, it is necessary to compute the topology of its overlap set. The algorithm I use is based on constructing a decomposition of the space into subspaces of known topology. One basis for this computation is the theorem of Seifert & Van Kampen ([7]), which states that

**Theorem 6** *Two simply connected regions A and B, overlapping in a simply connected region C, form a simply connected region.*

**Proof:**  
see [7].

The other important basis is the following:

**Theorem 7** *Two simply connected regions A and B, overlapping in several disjoint regions  $C_1, C_2, \dots$ , form a multiply connected region.*

**Proof:** A path through  $A \rightarrow C_1 \rightarrow B \rightarrow C_2 \rightarrow A$  cannot be contracted into single point.

QED

The algorithm, shown as function topology in Figure 9, works by searching for a decomposition of the set of c-regions into subregions such that the connectedness can be unambiguously determined using the 2

**function** topology( $O, E$ )

1.  $C \leftarrow$  maximal cliques within  $O$ .
2. for all  $c \in C$  do
  - (a)  $e \leftarrow O \setminus c$ , if  $e$  is empty return simply-connected
  - (b)  $o \leftarrow \text{overlap}(c, e, E)$ , if  $o$  is empty return not-connected.
  - (c)  $tr \leftarrow \text{topology}(e, E)$ ,  $to \leftarrow \text{topology}(o, E)$
  - (d) if  $tr =$  simply-connected and  $to =$  simply-connected return simply-connected
  - (e) if  $tr =$  simply-connected and  $to =$  not-connected return multiply-connected
  - (f) if  $tr =$  not-connected and  $to =$  simply-connected return not-connected
3. if no result has been found: return multiply-connected

**function** overlap( $X, Y, E$ )

return a list of all c-regions  $x \in X, y \in Y$  such that  $\{x, y\} \cup E$  is a clique.

Figure 9: Algorithm for computing the topology of the overlap set  $O$  of an environment  $E$ .

theorems above. When no such decomposition exists, the result must be multiply connected, as this is the only case where the decomposition can fail. The complexity of this procedure is significantly reduced by considering cliques of c-regions - which are known to be simply connected - as the elementary units of decomposition. In order to accurately determine the topology of the part which overlaps the initial environment  $E$ , it is important that the algorithm must only consider those connections which fall within  $E$ . This is achieved by the function overlap, which only returns the overlapping c-regions within  $E$ . Note that all operations can be implemented by subset tests on the maximal cliques of the region graph.

## 5 Rotations

When the moving object is allowed to rotate, four important differences appear:

- the topology of c-regions includes the doubly connected rotation group  $S^1$ , and thus c-regions are doubly connected.
- intersections of c-regions can be multiply connected or consist of multiple subregions.
- The three-dimensional configuration space admits  $S^2$  as another subgroup.
- Theorem 3 must now be applied to the 4-th order region graph (rather than 3-rd order).

These differences imply a number of complications to the algorithm, which however do not change its principal properties. The details of the modifications are more involved and beyond the scope of this paper. Implementation of the technique for the case of rotations is currently under way.

## 6 Implementation

I have implemented the techniques for two-dimensional objects. The input to the program is given in the form of three collections of convex bitmaps, representing the parts of the fixed objects, the moving object, and the cavities. A preprocessor uses these bitmaps to compute the region graph for the configuration space by exhaustively searching for simultaneous overlaps. This is by far the slowest operation since it performs iterative approximations on bitmaps. The preprocessor defines the set of obstacles and bubbles, and a graphical interface permits visualizing sample configurations in each.

Because arcs in the region graph represent intersections between regions which are present in a set of configurations, there is a high probability of finding them by random generation. Computation of the region graph can be made much more efficient by first generating a number of random configurations, and noting the part overlaps they exhibit. Only simultaneous overlaps which have not been ruled out and which have not been found by this procedure need to be searched for explicitly.

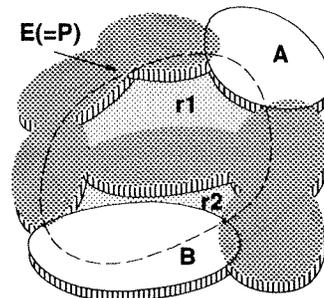


Figure 10: *Environment E (and consequently place P) models two disjoint regions, thus leading to incorrect connectivity between A and B.*

The implementation shows that it is indeed possible to perform spatial reasoning on the basis of bitmaps only. In practical applications, computations could first be carried out on polygonal approximations of objects, and representations of the precise shapes need only be used in case of ambiguities.

## 7 Discussion

**Resolution limits of cavities** The resolution of the free space representation is limited by the cavities covering it. In particular, a single environment  $E$  of bubbles, and thus a single place  $P$ , may cover several disjoint regions  $r_1, r_2, \dots$  of free space. The main problem caused by this phenomenon is that the connectivity of free space computed by the program may be incorrect by predicting a path from  $A \leftrightarrow P \leftrightarrow B$  when the existing connections are in fact  $A \leftrightarrow r_1$  and  $B \leftrightarrow r_2$  with no connection between  $r_1$  and  $r_2$  (Figure 10).

Insufficient resolution can be detected in the topological computation by the fact the the overlap set of  $E$  will be more than doubly connected, i.e. contain several "holes". This can be detected by modifying the topology computation algorithm described earlier so that it can return a different default solution if the best decomposition found is one which implies several holes. This problem can only be solved by increasing the resolution of the cavities whenever it is

insufficient. In our current implementation, we only signal the problem; increasing the resolution is up to the user.

**Using prior knowledge** If a place vocabulary has already been computed for a scenario involving some of the same objects, the corresponding parts of the region graph can be reused to solve a new problem. For example:

- when an obstacle is moved between two scenarios, the new region graph can be found by only recomputing those arcs which involved the moved obstacle. Only parts of the place vocabulary which depend on the newly created cliques need to be recomputed.
- the region graph for a situation could be composed of prototypes. A pattern recognition procedure could serve to identify the right prototypes to model each combination of obstacles.

The possibility of using prior knowledge makes it possible to envisage algorithms for computing place vocabularies for complex situations in real time.

**Extension to qualitative kinematics** Besides predicting the possible regions and their connectivity, it is often important to reason about the kinematics of contacts between surfaces. To do this, it would be useful to know the contact relationships which can be reached from a given place without passing through any other one. The region-based object representation is not detailed enough to represent contacts themselves, but only combinations of object pieces which could come into contact; these are represented as obstacles. For each place  $P$ , the overlap set of its underlying environment gives directly the set of obstacles which are adjacent to it, and might have to be considered for kinematic analysis.

**Three dimensions** The work presented in this paper is restricted to two dimensions, primarily to simplify graphical representation. For the case of pure translation, Theorems 2 and 3 generalize without modification since they only refer to the convexity of sets. I have not investigated the problem of three-dimensional rotations yet.

**Robustness** An important problem with all geometrical computation ([4]), and especially computation involving configuration spaces, is that small errors in the numerical computations can cause grossly erroneous results. For example, an algorithm which computes configuration spaces by tracing out its boundaries will give an entirely incorrect topology even if only one connection of the boundaries is computed incorrectly.

The topology-based computation is very robust against such errors. When a numerical error results in predicting a single non-existent overlap, this normally creates an additional clique containing only that overlap. Since the clique will be very small, it will most likely not be a superset of any interesting environment, and thus have no influence on the place vocabulary. When an overlap is missing, this can result in a clique being broken into two smaller subsets. In the case where this overlap is indeed the only one ruling out a place, this will cause a spurious place to appear, but not affect the rest of the place vocabulary.

**Complexity** Since each maximal clique models a feasible region of configuration space, its number does not grow more than  $O(n^d)$ , where  $n$  is the number of c-regions and  $d$  is the dimensionality of the configuration space. Since in a search algorithm all leaf nodes are maximal cliques with one more c-region added, the complexity of finding them is no greater than  $O(n^{d+1})$ .

The second important part of the algorithm is to compute the topology of region overlaps. This involves a search which is in fact of exponential complexity, but the number of c-regions which can overlap any environment is limited by the fact that the moving object cannot overlap parts which are farther apart than its size would allow. Thus, as the number of c-regions gets large, the complexity of determining the overlap topology of an environment is about constant. The number of overlap region computations is bounded by the number of environments which are examined. There are not more environments than feasible regions of configuration space, and thus their number again does not grow by more than  $O(n^d)$ .

Thus, the total complexity of the algorithm can be estimated to be about  $O(n^{d+1})$ . In practice, the computation is very fast: for the example of Figure 1, it

takes less than 1 second to compute the place vocabulary on the basis of the maximal cliques.

## 8 Conclusions

The novelty of the spatial reasoning technique presented in this paper lies in two aspects: using a region-based object representation, and using topological rather than geometrical properties for computing a place vocabulary.

Because the method is based on topology, it completely avoids the problems associated with algebraic surface models:

- object representations as unions of convex parts have long been postulated in vision research ([5]), and there are reliable methods for computing them.
- the methods are robust: inaccuracies in the possible overlaps have only local influence on the place vocabulary. Furthermore, since any overlap of  $c$ -regions always consists of a *set* of points, the numerical analysis required to find them is simpler than in cases where precise configurations must be detected.
- information about simultaneous overlaps obtained from previously solved subproblems can be directly reused in other contexts. Furthermore, it is possible to use *abstractions*: groups of parts can be approximated by their convex hull until a more precise representation is needed.

However, the technique requires that objects can be represented as unions of convex parts and thus cannot deal with concave surfaces. Furthermore, it requires a method for deciding the existence of configurations where certain combinations of small sets of parts and cavities can simultaneously overlap.

Further development of the methods for three-dimensional problems, rotations and multiple moving objects are currently under way. I expect it to yield much simpler methods for spatial reasoning than were possible with geometric boundary-based methods.

## References

- [1] V. Chvátal: "Linear Programming," W. H. Freeman, 1983
- [2] Z. Cui, A.G. Cohn, D.A. Randell: "Qualitative Simulation Based on a Logical Formalism of Space and Time," *Proceedings of the 10th National Conference of the AAAI*, San Jose, July 1992
- [3] Boi Faltings: "Qualitative Kinematics in Mechanisms," *Artificial Intelligence* 44(1), June 1990.
- [4] C.M. Hoffman: *Geometric and Solid Modelling: An Introduction*, Morgan-Kaufman Publishers, 1989
- [5] D.D. Hoffman, W.A. Richards: "Parts of Recognition," *Cognition* 18, 1985
- [6] Y.K. Hwang, N. Ahuja: "Gross Motion Planning - A Survey," *ACM Computing Surveys* 24(3), 1992
- [7] William S. Massey: "A Basic Course in Algebraic Topology," *Graduate Texts in Mathematics*, Springer, 1991
- [8] E. Spanier: "Algebraic Topology", Mc. Graw Hill, 1966
- [9] J. T. Schwartz, M. Sharir: "On the Piano Movers Problem. II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds," *Advances in Applied Mathematics*, 4, 1983
- [10] J.T. Schwartz, C.K. Yap: *Advances in Robotics, Vol. 1: Algorithmic and Geometric Aspects of Robotics*, Erlbaum, Hillsdale, N.J., 1987
- [11] T. Lozano-Perez, M. Wesley: "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Comm. of the ACM*, 22, 1979,

# A Semi-Quantitative Physics Compiler

**Adam Farquhar**

Knowledge Systems Laboratory  
Stanford University  
Palo Alto, CA, USA  
Ph: +001 415-723-9770  
Fax: +001 415-725-5850  
Adam.Farquhar@ksl.stanford.edu

**Giorgio Brajnik**

Dipartimento di Matematica e Informatica  
Università di Udine  
Udine, Italy  
Ph: +39 (432) 272210  
Fax: +39 (432) 510755  
giorgio@dimi.uniud.it

## Abstract

Incomplete information is present in many engineering domains, hindering traditional and non-traditional simulation techniques. This paper describes SQPC (semi-quantitative physics compiler), an implemented approach to modelling and simulation that can predict the behavior of incompletely specified systems, such as those that arise in the water control domain. SQPC is the first system that unifies compositional modeling techniques with semi-quantitative representations. We describe SQPC's foundations, QSIM and QPC, and how it extends them. We demonstrate SQPC using an example from the water supply domain.

## 1 Introduction

Consider the problem of water supply control. A lake has a dam with floodgates that can be opened or closed to regulate the water flow through power generating turbines, the water level (stage) of the lake, and the downstream flow. The goal of a controller is to provide adequate reservoir capacity for power generation, consumption, industrial use, and recreation, as well as downstream flow. In exceptional circumstances, the controller must also work to minimize or avoid flooding both above and below the dam. This task is both difficult and vitally important to the residents of surrounding areas. The work of controllers could be substantially eased by sound automatic modeling and simulation tools.

There are several forms of incomplete information that appear in this domain. The precise shape and capacity of lakes or reservoirs is rarely known; the outflow from opening a dam's floodgates is only crudely measured; empirical data on the level/flow-rate curve for rivers becomes less and less accurate when flood conditions approach; few quantities are measured (e.g. flow rates of minor tributaries are not measured at all); the amount of runoff to be expected from a given rainfall depends on difficult to measure surface characteristics such as saturation; the amount of rainfall that actually falls on a lake and surrounding areas is difficult to predict and is imprecisely measured. Nonetheless, both mathematical analysis and observations do provide rough bounds on the quantities involved. Often,

rough accurate bounds suffice to select appropriate actions.

This domain is challenging for existing approaches to modeling and simulation. Pure qualitative reasoning techniques (Forbus 1984; Kuipers 1986) do not exploit the partial information available and consequently provide insufficiently strong predictions. Traditional numeric methods require much more precise information than is available, forcing modelers to make assumptions which may invalidate results and which may be difficult to evaluate.

Fortunately, recent advances in *semi-quantitative* simulation techniques provide a method for predicting the behavior of such systems. This work extends the purely qualitative representation (Kuipers 1986) with means for representing semi-quantitative information (Berleant & Kuipers 1988; Kuipers & Berleant 1992; Kay & Kuipers 1993). In this work, semi-quantitative information is represented in two forms: bounds on variable values and functional bounds (envelopes) on otherwise unspecified monotonic functions. This is exactly the kind of information that is available in the water supply and many engineering domains.

Several systems (Forbus & Falkenhainer 1990; Iwasaki & Low 1991; Amador, Finkelstein, & Weld 1993) have been developed that use compositional modelling techniques and exploit qualitative models to provide explanations of numeric simulations. They are unable to represent or use semi-quantitative information. In order to provide a numeric simulation, they all require complete precise initial conditions and algebraic equations.

This paper describes SQPC (semi-quantitative physics compiler), an implemented approach to modelling and simulation that uses semi-quantitative knowledge. SQPC extends QPC (qualitative physics compiler) (Crawford, Farquhar, & Kuipers 1990; Farquhar 1993; 1994) to exploit the recent advances in semi-quantitative simulation. SQPC is the first compositional modeling system to employ semi-quantitative representation and simulation.

The input to SQPC is a *domain theory* and a *scenario*. The domain theory is composed primarily of

*model fragment* definitions that describe both the conditions under which physical phenomena are active, and their consequences. The scenario specifies objects that are known to be of interest, some initial conditions, and some relations that hold throughout the scenario. Both the domain theory and scenario may include bounds on numeric values and monotonic functions. From this, SQPC generates a set of behavioral descriptions, guaranteed to cover any system trajectory consistent with the scenario and domain theory. A behavior may pass through a number of distinct operating regions, each of which is characterized by a distinct mathematical model.

## 2 Foundations

### 2.1 Semi-quantitative simulation

SQPC is built on top of the QSIM qualitative simulator (Kuipers 1986; 1993). The input to QSIM is a *qualitative differential equation* (QDE) which specifies: (i) a set of variables (continuously differentiable functions of time); (ii) a *quantity space* for each of these variables, specified in terms of a totally ordered set of symbolic *landmark* values; (iii) a set of constraints expressing (algebraic, differential or monotonic) relationships between variables. A QDE is an abstract description of a, perhaps infinite, set of ordinary differential equations. The output of QSIM is a set of behaviors. Each behavior is a sequence of states, where a state is a mapping of variables to qualitative values. A qualitative value represents the magnitude of the variable, which is either equal to a landmark or in the open interval specified by adjacent landmarks, and its direction of change (the sign of its time derivative: *dec*, *std*, *inc*). Each state describes either a time point or an open temporal interval.

In the semi-quantitative framework employed by SQPC, the basic qualitative representation is augmented by use of Semi-Quantitative Differential Equations (SQDE) (Berleant & Kuipers 1988; Kuipers & Berleant 1992; Kay & Kuipers 1993). Each landmark may be bounded with a precise numeric upper and lower bound. Each monotonic function constraint may be bounded with a precise functional upper and lower bound. A monotonic function constraint represents an element of an infinite set of real valued functions. Its general form is  $((M s_1 \dots s_n) x_1 \dots x_n y)$  where each  $s_i$  is a sign and  $x_i, y$  are variables. Such a constraint denotes the set

$$\mathcal{F}_M = \left\{ f \left| \begin{array}{l} f \text{ is a continuous} \\ \text{differentiable function} \\ f: \mathbb{R}^n \rightarrow \mathbb{R} \\ \text{and } \forall i: \text{sign}(\frac{\partial f}{\partial x_i}) = s_i. \end{array} \right. \right\}.$$

An envelope for such a constraint is a pair of functions  $F = \langle \underline{f}, \bar{f} \rangle$  that restricts the set being characterized by the constraint. That is, an envelope  $\langle \underline{f}, \bar{f} \rangle$  for the above mentioned constraint characterizes the set

$$\mathcal{F}_F = \left\{ f \left| \begin{array}{l} f \in \mathcal{F}_M \text{ and } \forall t \in T: \\ \underline{f}(x_1(t), \dots, x_n(t)) \leq f(x_1(t), \dots, x_n(t)), \\ \bar{f}(x_1(t), \dots, x_n(t)) \geq f(x_1(t), \dots, x_n(t)) \end{array} \right. \right\}.$$

The semi-quantitative simulators augment behavior with the numeric bounds. They are also able to use the semi-quantitative information to rule out qualitatively possible behaviors. The first semi-quantitative techniques (Berleant & Kuipers 1988) propagated the bounds throughout each time-point state, and then used the mean-value theorem to constrain the values across time. The later dynamic envelope techniques (Kay & Kuipers 1993) construct extremal equations for the derivative of each state variable. These extremal equations are then explicitly integrated to provide bounds on variable values across time intervals. Neither technique strictly dominates the other. As a result, the bounds provided by the two methods may be intersected, yielding sometimes stronger predictions than either alone.

### 2.2 Qualitative Physics Compiler

SQPC is an extension of QPC, whose modeling language builds on Qualitative Process theory (Forbus 1984). The input to QPC is a *domain theory* and *scenario* specified in the QPC modeling language. A domain theory consists of a set of quantified definitions, called *model fragments*, each of which describes some aspect of the domain, such as physical laws (e.g. mass conservation), processes (e.g. liquid flows), devices (e.g. pumps), and objects (e.g. containers). Each definition applies whenever there exists a set of participants for whom the stated conditions are satisfied. The specific system or situation being modeled is partially described by the scenario definition, which lists a set of objects that are of interest, some of the initial conditions, relations that hold throughout the scenario, and boundary conditions.

Influences are compositional relations between variables that are particularly convenient for asserting fragments of information that can be composed into constraints. Three kinds of influences are supported. An *indirect influence* such as  $(Y Q^+ X)$  means that in the absence of countervailing influences, an increase in  $X$  causes an increase in  $Y$  and that  $Y$  is functionally determined by the set of influencing variables. More precisely  $(Y Q^+ X)$  means that there exists  $f$ , a continuous function, and a set of variables  $\{x_i\}_{1 \leq i \leq n}$  such that  $Y = f(X, x_1, \dots, x_n)$  and  $\frac{\partial f}{\partial X} > 0$ . In the case of  $(Y Q^- X)$  then  $\frac{\partial f}{\partial X} < 0$ . The *algebraic influences*  $Q_{add}$  and  $Q_{sub}$  provide the constraint  $\frac{\partial f}{\partial X} = 1$  and  $\frac{\partial f}{\partial X} = -1$  respectively. Finally, a *direct influence* such as  $(Y I^+ X)$  expresses that a positive  $X$  tends to increase  $Y$ . This is equivalent to an algebraic influence on the derivative of the influenced variable (*i.e.*,  $(Y I^+ X) \equiv Y' = \frac{dY}{dt}$  and  $(Y' Q_{add} X)$ ).

QPC employs a hybrid architecture in which the model building portion is separated from the simulator. This architecture allowed SQPC to exploit semi-quantitative information without changing the overall QPC algorithm.

The input to QPC (Farquhar 1994) is a domain theory describing physical phenomena and a specification of the system to be modeled, called the scenario. The domain theory and scenario induce a set of logical axioms. QPC uses this database of logical axioms to infer the set of model fragment instances that apply during the time covered by the database (called the *active* model fragments). Inferences performed by QPC include those concerning structural relationships between objects declared in the scenario, and those aiming at computing the transitive closure of order relationships between quantities. A database with a complete set of model fragment instances defines an initial value problem which is given to the simulator in terms of equations and initial conditions. If any of the predicted behaviors cross the boundary conditions the process is repeated: a new database is constructed to describe the system as it crosses the boundaries of the current model, another complete set of active model fragments is determined, and another simulation takes place.

The output of QPC is a directed rooted graph, whose nodes are either databases or qualitative states. The root of the graph is the initial database, and a possible edge in the graph may:

- link a database to a refined database (obtained by adding more facts, either derived through inference rules or assumed by QPC when ambiguous situations are to be solved);
- link a complete database to a state (which is one of the possible initial states for the only model derivable from the database);
- link a state to a successor state (this link is computed by QSIM);
- link a state to a database (the last state of a behavior which crossed the operating region; the database describes the situation just after the transition occurred).

Each path from the root to a leaf describes one possible temporal evolution of the system being modeled. Each model in path identifies a distinct operating region of the system.

### 3 SQPC extends QPC

SQPC extends the modeling language, the underlying representation and the inference methods employed. This section describes these extensions.

#### 3.1 Modeling language

SQPC extends the QPC modeling language by adding numeric bounds on magnitudes, dimensional informa-

tion, bounding envelopes on monotonic functions, and functions specified by tables.

**Numeric values.** SQPC represents numeric and qualitative magnitudes in a single framework. Both represent specific real numbers, which might be known only with uncertainty. Numeric magnitudes constrain such a number to lie within a numeric range. Note two aspects that complicate reasoning on numeric magnitudes. First, two comparable magnitudes constrained by the same range are, in general, not equal (*i.e.*,  $Range(m) = [a\ b]$  and  $Range(n) = [a\ b]$  do not entail that  $m = n$  unless  $a = b$ ). Secondly, range constraints on magnitudes may change during the analysis (range refinement). This may happen as an effect of the semi-quantitative simulation performed by QSIM. A model might entail  $Range(m) = [a\ b]$ , while a subsequent model in the behavior graph computed by SQPC might entail  $Range(m) = [a'\ b']$  where  $[a'\ b'] \subseteq [a\ b]$ . That is, as the analysis proceeds, SQPC may tighten the bounds on the numeric range of a magnitude.

**Dimensional information.** Variables and (symbolic or numeric) magnitudes are partitioned into dimensions. SQPC defines the seven International System dimensions as well as a *null-dimension*, which is provided to represent "pure number" quantities such as the efficiency of a turbine. A domain theory may also introduce derived dimensions specified by a list of dimension names with integer exponents (for example, **power-dimension** is  $ml^2/t^3$ ).

Explicit representation of dimensions enables SQPC to:

1. perform dimensional analysis and verify that equations and order relations are well formed. Dimensional errors are common when writing equations and can be easily detected;
2. constrain inference about order relations. It is senseless to compare quantities that do not have the same dimension, and a reasoning mechanism not exploiting any dimensional information can produce incorrect inferences such as  $x < 5 \wedge 10 < V \vdash x < V$  where a position ( $x$ ) is being compared to a volume ( $V$ ).

**Bounding envelopes.** An *envelope schema* is defined by a form similar to that for defining model fragments. It states a set of conditions under which a specific form of monotonic function over a tuple of variables is bounded by a functional envelope. The envelope is specified by a pair of functions. Instantiated envelope schemas are used to enrich a model with suitable envelopes. Since instantiation is automatically performed, envelopes are installed in models as needed, provided an appropriate monotonic constraint has already been included in the model.

**Tabular functions.** Tabular functions provide an important practical extension to the modeling lan-

guage. A large portion of empirically collected knowledge about time-varying systems is represented and summarized in tabular form. The SQPC language permits numeric functions (used to specify envelopes) to be defined by data in a multi-dimensional table. SQPC assumes that these tables are coarse descriptions of the continuous reasonable functions that satisfy monotonic constraints. Currently SQPC provides two mechanisms for interpolating tabular data: *step-wise* functions, providing piecewise constant upper and lower bounds, or *piecewise linear* functions, providing tighter, but possibly less accurate, interpolations. In this way it is possible to define two envelope schemas from the same underlying tabular data. One envelope schema will use a linear interpolation method in a region where this approximation is known to introduce no significant error; in other regions a safer, but less precise, envelope schema using the more conservative interpolation method based on stepwise bounding functions, will be used. Of course the set of interpolation methods being used for computing tabular functions is open ended. The current version of SQPC provides the two mentioned above.

### 3.2 Reasoning

To accommodate the representational extensions described above there are several extensions that need to be made to the reasoning mechanism based on the underlying QPC architecture.

Dimensional information allows the inferences that compute order relations between variables and magnitudes to be focused. SQPC never compares two quantities with incompatible dimensions. The compatibility test is simplified by reducing all dimensions to a canonical form, represented by a vector of exponents (each position in the vector corresponds to a basic dimension).

Numeric bounds on magnitudes require a change in the computation of order relations. Except for the simple cases, in which the bounding ranges do not overlap, SQPC leaves the computation of numeric order relations to the semi-quantitative QSIM extensions. QSIM does a good job of propagating the bounds through the constraints in the SQDE. Recreating this in the SQPC knowledge base would be unnecessary, redundant, and inefficient.

SQPC needs to determine which envelopes to include in the SQDE for each model. This is non-trivial because there are several ways to describe a monotonic relationship among a set of quantities. Because each envelope that can be included is likely to strengthen the predictions, it is important to include all of the applicable ones. For instance, suppose that the model contains the constraint  $(M (+ -) X Y Z)$  but there is an envelope defined for the constraint  $(M (+ +) Y Z X)$ . These two constraints are analytically equivalent, so the second constraint and its envelope should be included in the SQDE. This enables ranges for  $X$  to be

computed given ranges for  $Y$  and  $Z$ , whereas an envelope for the former constraint enables computing the range for  $Z$  from the ranges of the other two variables.

SQPC adds any constraint and envelope into the SQDE that is a permutation of a constraint in the SQDE. Notice that SQPC includes constraints in models after resolving influences (*i.e.*, after assuming a closed world and having determined the complete set of influencing and influenced variables). Then SQPC looks for possible envelopes (and possibly equivalent constraints) to be added to the model. This strategy makes it possible for the designer of the domain model and scenario to specify the envelopes, or envelope schemas, on the basis of the available data, independently from how influences will get resolved. In those cases where SQPC will construct models where some monotonic constraint does not have any envelope, SQPC will still be able to produce an accurate prediction, though with reduced precision.

In order to produce qualitative simulations of large systems, QPC generated attainable envisionments. Though leading to tractable simulations, this produces less detailed descriptions of the dynamics of the modeled system, for landmarks automatically generated by QSIM are crucial for identifying new critical points (and attaching numeric ranges to them). SQPC, unless appropriately instructed, does not produce an envisionment, but a more standard tree-based simulation, enabling also landmark generation. This choice, while providing more detailed results, has the drawback of requiring other methods for controlling the combinatorial explosion inherent in qualitative simulation. We are currently designing SQPC extensions based on known methods for reducing the number of spurious behaviors: *higher order derivatives* (Kuipers *et al.* 1991), *energy functions* (Fouché & Kuipers 1990), *phase-space criteria* (Lee & Kuipers 1993), and *abstraction techniques* (Clancy & Kuipers 1993).

## 4 An Example

We demonstrate SQPC on a problem from the domain of water supply control. We consider a portion of the system of lakes and rivers to be found in the scenic hill country surrounding Austin, Texas. The Colorado river flows into Lake Travis. The Mansfield Dam on Lake Travis produces hydroelectric power, controls the level of the lake, and the flow into the downstream leg of the Colorado.

The problem is to evaluate a "what if" scenario (figure 1). We are given an initial level for Lake Travis (a typical value between 690.2 and 690.3 feet) and a rough projected inflow from the Colorado river (between 791 and 950 cfs). The task is to determine what happens to the lake level and evaluate how long the hydroelectric plant can deliver power at the requested rate of 10 Mw.

In addition to the numeric bounds variables in the scenario, there are several other sources of semi-

```

(defscenario Travis-1-turbine
  "Turbine from controlled to uncontrolled regime."
  :entities ...
  :landmarks
  ((top-of-dam :variables ((stage trAVIS)) :range 714))
  :relations
  ((= (flow-rate colorado-up) (791 950)) ; cfs
   (= (top mansfield) top-of-dam)
   (is-open turb))
  :initial-conditions ((= (power turb) 10) ; Mw
                      (= (stage trAVIS) (690.2 690.3))) ; ft
  :envelopes
  ((stage-capacity
   :constraint (M+ (stage trAVIS) (capacity trAVIS))
   :upper-envelope (hi (lake-travis-stage-capacity
                       '(st cap) (list (stage trAVIS))))
   :lower-envelope (lo (lake-travis-stage-capacity
                       '(st cap) (list (stage trAVIS))))
   :upper-inverse (lo (lake-travis-stage-capacity
                      '(cap st) (list (capacity trAVIS))))
   :lower-inverse (hi (lake-travis-stage-capacity
                      '(cap st) (list (capacity trAVIS))))))

```

Figure 1: Part of the scenario description. The initial conditions specify the desired power output of the turbine and the stage (level) of Lake Travis. The flow-rate of the upstream leg of the Colorado is set for the duration of the scenario. The `:envelopes` clause specifies the state-capacity table for Lake Travis. Any model in the scenario which includes the  $M^+$  between the stage and capacity of lake trAVIS will include this envelope.

Head (ft)	Power (Mw)	Discharge-rate (cfs)
120	8	1,054
120	9	1,150
...	...	...
120	20	2,675
125	8	1,026
...	...	...
145	20	1,940
150	8	884
...	...	...
150	30	2,936

Table 1: A portion of the table describing turbine behavior. E.g. given a head of 120ft and a power setting of 8Mw, the discharge rate is expected to be 1054cfs.

quantitative information in this problem. An envelope schema (see figure 2) establishes bounds on the relation between the *head* of water above a turbine, the desired *power* output, and the *discharge rate* of water downstream. This envelope schema applies whenever the conditions and constraint portions of the envelope form are satisfied.

Lake Travis is a unique object. The relation between the *stage* (level) and *capacity* (volume) for Lake Travis is provided by an envelope specified in the scenario definition.

All of the semi-quantitative information in this do-

main theory is specified in the form of tables.<sup>1</sup> The tables reflect both observations and engineering estimates about the relationships between important variables. Table 1 shows some of the data extracted from the table describing quantitatively the behavior of the turbines in Mansfield Dam.

Solving this problem is made slightly more complex because of the behavior of the turbines. The turbines are controlled by a servo-mechanism designed to generate the desired amount of power regardless of the hydraulic pressure, which is determined by the *head* at the turbine. This is possible as long as there is sufficient *head*. When the *head* drops below the minimum threshold for a given power output, then less power is released. The domain theory captures this accurately (see figure 4). The domain theory also includes model fragments for conservation laws (e.g. of mass and energy), basic hydraulic principles (e.g. flow is proportional to head), and so on.

Figure 3 shows the SQPC output for this scenario. Under the specified conditions, the desired power level can be maintained until time T1, at least 45 days ( $3.94 * 10^6$  seconds) after the start time. After T1, there will be insufficient hydraulic pressure to provide the full power output, the discharge rate from the turbine will decrease until it reaches equilibrium with the inflow at a rate between 791 and 950 cfs, and the lake

<sup>1</sup>The Lower Colorado River Authority (LCRA) has contributed actual tables of empirical data to the Qualitative Reasoning Group of the University of Texas for evaluation.

---

```

(defenvelope MANSFIELD-TURBINE-DR-E
  :comment "HEAD x POWER --> DISCHARGE-RATE of a Mansfield turbine."
  :participants ((dam :type dams)
                (lake :type lakes
                      :conditions ((connects dam lake river)))
                (river :type rivers)
                (t :type mansfield-turbines
                  :conditions ((has-valve dam t))))
  :constraint ((M - +) (head t) (power t) (discharge-rate t))
  :upper-envelope
    (hi (lake-travis-turbine-discharge-rate
        '(hd pw dr) (list (head t) (power t))))
  :lower-envelope
    (lo (lake-travis-turbine-discharge-rate
        '(hd pw dr) (list (head t) (power t))))

```

---

Figure 2: An envelope that is applicable to any turbine of the kind found on Mansfield dam. It bounds the function that determines the discharge rate of the turbine given the head above it and the desired power output. The two functional bounds are computed by extracting the lowest and highest points from the range returned by a Lisp function (`lake-travis-turbine-discharge-rate`) defined on the basis of the "turbine rating table" shown in the text.

---

level will stabilize between 568' and 688'. Notice that at **T1** the lake system is entering a new operating region because the turbine is no longer servo-controlled (*i.e.*, the model fragment `NORMAL-TURBINE-MF` is no longer active). Notice also that all the variables are continuous across the transition, but some of their derivatives are not (*e.g.*, the derivative of `discharge-rate`, shown in terms of *qdir* of `discharge-rate`).

These predictions are strong enough to be useful to a system controller, even though the problem statement is very imprecise: the flow rate was very coarse; there are no semi-quantitative bounds for the relationship between *power* and *head* in the low-head situation after **t1**; the table relating *stage* and *capacity* becomes very coarse below 600'.

More precise information in the domain theory or scenario will result in more precise predictions. This is the strength of the semi-quantitative inference methods. The precision of the predictions is monotonic with the precision of the model and initial conditions.

We illustrate this by first strengthening the initial conditions of the scenario and then by strengthening the domain theory. If the upper bound on the inflow rate is reduced from 950 cfs to 800 cfs, then the upper bound on **T1**, the time that power generation drops below the desired rate, is reduced to 76 days, a 58% improvement. The domain theory can be strengthened by tightening the envelopes by using a linear interpolation for the stage-capacity curve instead of a step function. This tightens the range for **T1** to 50-58 days, an improvement of 89% from the original. Increased precision in the input or model leads to increased precision in the output.

#### 4.1 Implementation status

SQPC is fully implemented in Lucid Common Lisp as an extension to QPC, which in turn uses the Algernon knowledge representation system (Crawford 1991) and QSIM. We are currently experimenting with SQPC in the water flow control domain; SQPC has been run on a dozen examples comparable to the one shown in this paper. The runtime for this example is around 4 minutes on Sun Sparc4/75. The bulk of this time is spent computing order relations with interpreted rules. Using standard rule compilation techniques or a special purpose inequality reasoner will result in a substantial (orders of magnitude) speedup.

### 5 Related work

In recent years, several research efforts have worked towards the development of self-explanatory simulators that construct numerical simulations and use a qualitative representation to help explain the results. Unlike SQPC, they do not use semi-quantitative information. Their predictions are either precise numeric ones, or purely qualitative.

SIMGEN (Forbus & Falkenhainer 1990) computes a total envisionment of the scenario and then, for each envisionment state<sup>2</sup> it builds a numerical simulator, monitors the simulation and, at the end of the analysis, interprets numerical results in terms of the envisionment graph. SIMGEN requires precise and complete numerical equations, initial and boundary conditions for the simulation. SIMGEN must be capable of building a numerical model for *each* envisionment state touched during the simulation; to this end it must be supplied with a library of numeric procedures for *every*

---

<sup>2</sup>corresponding to an operating region.

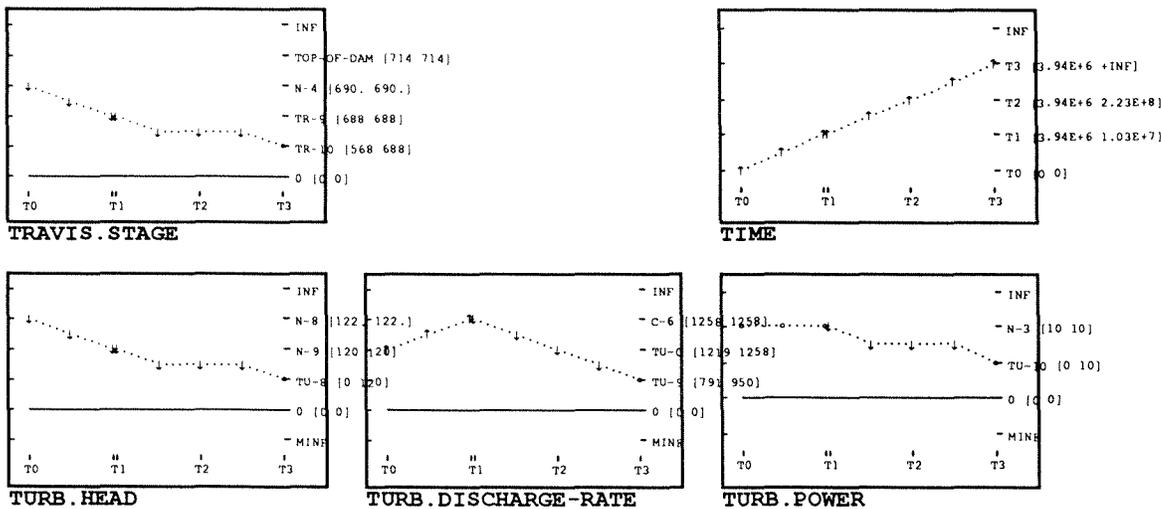


Figure 3: Behavior plot for several variables in the scenario. The power output by the turbine after T1 is below the desired level.

possible combination of influences. SIMGEN is incapable of performing a simulation when a qualitative relation is quantitatively underspecified or when precise knowledge unavailable for any initial conditions.

DME (the Device Modeling Environment) (Iwasaki & Low 1991) is an incremental compositional modeling system capable of generating self-explanatory simulations. DME can work in two exclusive modes: qualitative or numeric. In the former case DME constructs qualitative states, and uses QSIM to generate successors; in the latter case, DME builds numerical models for simulation. In both modes, crossing an operating region triggers remodelling. DME is highly interactive and provides sophisticated explanation capabilities (Gautier & Gruber 1993; Gruber & Gautier 1993). DME requires precise numerical equations, initial and boundary conditions. Therefore, DME does not integrate qualitative and quantitative information in prediction.

Pika (Amador, Finkelstein, & Weld 1993) builds a numerical model for each operating region of the system as soon as this is needed. Pika monitors the numerical simulation and, at the end of the analysis, is capable of engaging in a simple question/answering dialogue. Pika requires precise equations, complete initial conditions (unlike the other systems), and complete specification of boundary conditions (in particular inequalities are not allowed). Compared to SQPC Pika performs limited inferences: no structural inferences are possible (this limits the expressive power of the modeling language) and influences are limited to indirect and algebraic ones: no provision is made for handling more general monotonic influences.

## 6 Conclusion

We have presented SQPC, the first system to unify compositional modeling techniques with semi-quantitative simulation. This is crucial for automatically building models of systems whose dynamics cross several operating regions. SQPC automatically constructs semi-quantitative models and produces useful predictions with imprecise knowledge. We have shown how QPC was extended to accomplish this. We argued that semi-quantitative knowledge is crucial to many applied engineering domains the one chosen for demonstrating SQPC, water supply control.

The current version of SQPC relies primarily on the Q2 static envelope (Berleant & Kuipers 1988) method which is based on the mean-value theorem. The NSIM (Kay & Kuipers 1993) dynamic envelope method, which performs explicit numeric integration of extremal bounding equations, often provides tighter bounds. Reliance on Q2 is due partly to software development schedules and partly to the extensive use of tabular information in the water supply control domain. Because NSIM performs explicit numeric integration, it requires smooth continuous functional bounds. This requires an extension to our current tabular data tools to generate smooth monotonic functional bounds for the tables. We are currently developing this extension.

The ability to use semi-quantitative information in a compositional modeler is tremendously exciting. We look forward to extending existing domain theories, such as the chemical engineering theory constructed by Catino (Catino 1993) to include semi-quantitative information and exploring the construction of substantial engineering quality models that are tractable due to greater available precision.

---

```

(defmodelfragment NORMAL-TURBINE-MF
:participants ((t :type turbines)
              (dam :type dams :conditions ((has-valve dam t)
                                           (is-open t)))
              (lake :type lakes :conditions ((connects dam lake river)))
              (river :type rivers))
:operating-conditions ((> (stage lake) (base t))
                      (<= (min-head t) (head t))
                      (<= (head t) (max-head t)))
:consequences ((I- (capacity lake) (discharge-rate t))
              (Q- (discharge-rate t) (head t))
              (Q+0 (discharge-rate t) (power t))
              (Q-add (flow-rate river) (discharge-rate t))
              (Q+0 (min-head t)(power t))
              (Q+0 (max-head t)(power t))))

```

---

Figure 4: This model fragment describes normally operating turbines. An instance exists if there is an open turbine on a dam. It is active when the head above the turbine is between the variables `min-head` and `max-head` and the lake level is above the base of the turbine. Consequences specify equations that hold when an instance is active. The qualitative influence `I-` specifies that a positive `discharge-rate` decreases lake `capacity`; influences `Q-` and `Q+0` partially specify monotonic functions on the `discharge-rate` of the turbine (which is causally influenced by both `head` and `power`). The `Q-add` specifies that the `discharge-rate` sums into the river's `flow-rate`. The final influences specify that both `min-head` and `max-head` are affected by the desired `power` output.

---

Coupled with compositional modeling, the semi-quantitative techniques have the promise of achieving one of the major goals of qualitative reasoning: to make strong predictions about behavior, given the strongest model available.

### Acknowledgements

Adam's work has taken place in part at the Qualitative Reasoning Group at the Artificial Intelligence Laboratory, The University of Texas at Austin, which is supported in part by NSF grants IRI-8904454, IRI-9017047, and IRI-9216584, and by NASA contracts NCC 2-760 and NAG 9-665. It has continued at the Stanford Knowledge Systems Laboratory, which is sponsored by the Advanced Research Projects Agency, ARPA Order 8607, monitored by NASA Ames Research Center under grant NAG 2-581; and by NASA Ames Research Center under grant NCC 2-537.

Giorgio started working on the research reported in this paper while he was visiting the UT Qualitative Reasoning Group during 1992. He would like to thank all the members of the group, and especially prof. Ben Kuipers, Bert Kay and Adam.

### References

- Amador, F.; Finkelstein, A.; and Weld, D. 1993. Real-time self-explanatory simulation. In *Proc. of the Eleventh National Conference on Artificial Intelligence*. AAAI Press/MIT Press.
- Berleant, D., and Kuipers, B. 1988. Using incomplete quantitative knowledge in qualitative reasoning. In *Proc. of the Sixth National Conference on Artificial Intelligence*, 324-329.

Catino, C. 1993. *Automated Modeling of Chemical Plants with Application to Hazard and Operability Studies*. Ph.D. Dissertation, Department of Chemical Engineering, University of Pennsylvania, Philadelphia, PA.

Clancy, D., and Kuipers, B. 1993. Behavior abstraction for tractable simulation. In *Proc. of the Seventh International Workshop on Qualitative Reasoning about Physical Systems*, 57-64.

Crawford, J.; Farquhar, A.; and Kuipers, B. 1990. QPC: a compiler from physical models into qualitative differential equations. In *Proc. of the Eight National Conference on Artificial Intelligence*. AAAI Press/The MIT Press.

Crawford, J. 1991. Algernon — a tractable system for knowledge representation. *SIGART Bulletin* 2(3).

Farquhar, A. 1993. *Automated Modeling of Physical Systems in the Presence of Incomplete Knowledge*. Ph.D. Dissertation, Department of Computer Sciences, the University of Texas at Austin. Available as technical report UT-AI-93-207.

Farquhar, A. 1994. A qualitative physics compiler. In *Proc. of the Twelfth National Conference on Artificial Intelligence*.

Forbus, K., and Falkenhainer, B. 1990. Self-explanatory simulations: an integration of qualitative and quantitative knowledge. In *Proc. of the Eight National Conference on Artificial Intelligence*, 380-387. AAAI Press/The MIT Press.

Forbus, K. 1984. Qualitative process theory. *Artificial Intelligence* 24:85-168.

Fouché, P., and Kuipers, B. 1990. An assessment of current qualitative simulation techniques. In *Proc. of Fourth International Workshop on Qualitative Reasoning about Physical Systems*, 195-205.

Gautier, P., and Gruber, T. 1993. Generating explanations of device behavior using compositional modeling and causal ordering. In *Proc. of the Eleventh National Conference on Artificial Intelligence*. AAAI Press/MIT Press.

Gruber, T., and Gautier, P. 1993. Machine-generated explanations of engineering models: a compositional modeling approach. In *Proc. International Joint Conference on Artificial Intelligence*.

Iwasaki, Y., and Low, C. M. 1991. Model generation and simulation of device behavior with continuous and discrete changes. Technical Report KSL 91-69, Knowledge Systems Laboratory — Stanford University.

Kay, H., and Kuipers, B. 1993. Numerical behavior envelopes for qualitative models. In *Proc. of the Eleventh National Conference on Artificial Intelligence*. AAAI Press/MIT Press.

Kuipers, B., and Berleant, D. 1992. Combined qualitative and numerical simulation with Q3. In Faltings, B., and Struss, P., eds., *Recent advances in qualitative physics*. MIT Press. 3-16.

Kuipers, B.; Chiu, C.; Molle, D. D.; and Throop, D. 1991. Higher-order derivative constraint in qualitative simulation. *Artificial Intelligence* 51:343-379.

Kuipers, B. 1986. Qualitative simulation. *Artificial Intelligence* 29:289-338.

Kuipers, B. 1993. Reasoning with qualitative models. *Artificial Intelligence* 59:125-132.

Lee, W. W., and Kuipers, B. 1993. A qualitative method to construct phase portraits. In *Proc. of the Eleventh National Conference on Artificial Intelligence*. AAAI Press/MIT Press.

# Physical Model Generation in Thermal Engineering Problems described by Partial Differential Equations

**Donal P. Finn**  
Hitachi Dublin Laboratory  
Trinity College  
University of Dublin  
Dublin 2, Ireland  
dfinn@hdl.ie

**Padraig Cunningham**  
Dept. of Computer Science  
Trinity College  
University of Dublin  
Dublin 2, Ireland  
Padraig.Cunningham@cs.tcd.ie

**Abstract** Model generation has emerged as a key task in engineering design and analysis. AI research in this area has focused on model based reasoning emphasising qualitative models in attempting to automate this process. In this paper, we propose that this work on the use of model based reasoning in model generation would benefit from the inclusion of case-based reasoning (CBR) techniques. We argue that the use of cases constrains the reasoning process as cases reflect known good routes in the solution space. Cases also have the advantage of facilitating the integration of heat transfer exemplars, approximations, formulae and correlations. In addition, much of human competence in this area is based on reusing solutions to previously solved problems and CBR emulates this. In the paper, we advance these arguments based on our experience with CoBRA, a CBR system for physical model generation for the domain of heat transfer described by partial differential equations.

## 1 INTRODUCTION

Model generation has been recognised to be one of the significant research challenges of the qualitative reasoning community [22, 23, 24]. In recent years, research work has focused on different aspects of model generation including; modelling of engineering systems using compositional modelling [7,12], behavioural modelling of engineering phenomena using model abstraction switching [1], modelling across multiple ontologies using meta-modelling techniques [13], simplification of design analysis models using first principles reasoning [14], differential equation modelling using order of magnitude reasoning [25]. Although these projects have been motivated by different goals and adopt different artificial intelligence approaches, a number of general points can be made. Firstly, most work has interpreted model generation as a 'model switching' task between an initial complex model and some simpler but unspecified model. Consequently, this perspective has led to the development of model generation systems that have been based on traversing a vast solution space of engineering knowledge using model-based reasoning techniques. Secondly, few of the research efforts appear to have been explicitly grounded on a cognitive understanding of how engineers in practise actually carry out modelling. This, we believe, has resulted in the overlooking of a large body of experiential engineering know-how and techniques. Thirdly, most of the research efforts have aspired towards automated modelling environments which aim to replicate the skills and expertise of engineers. This, we argue has resulted in the focusing on modelling tasks

that are often simplistic and therefore unrelated to modelling of real world engineering problems. Finally, it is noted that for some work, there appears to have been little effort in understanding the real needs of engineers from model generation tools and to apply these findings to the research efforts; this has resulted in the development of applications that are often of little practical use to the engineering profession. It is worth noting that these comments are not unique to this paper, in so far that they have been noted by other researchers commenting on the direction of research in the qualitative reasoning community [22,23,24].

In this work, we focus on the task of physical model generation associated with the analysis of engineering problems described by partial differential equations (PDEs). PDEs are nowadays analysed using numerical simulation techniques such as the finite element method. Prior to simulation engineers must create simplified spatial, phenomenological and temporal models of real world engineering problems to facilitate efficient computation. Thus, in this context, physical model generation can be regarded as one of the preliminary stages of numerical PDE analysis [9]. It has been acknowledged by both engineering [2, 8] and numerical analysis researchers [3, 18] that these preliminary modelling tasks form a crucial part of the overall simulation process and they call for increased research efforts in the development of knowledge based model generation tools. Although, there has been considerable work from the qualitative reasoning community in model generation, there has been little effort explicitly directed towards physical model generation in numerical simulation of PDEs.

In this paper, we present a novel approach to physical modelling in heat transfer analysis which aims to address many of the issues raised in the first paragraph including: What is the nature of modelling in PDE analysis? How do engineers carry out modelling and how does this influence our approach? What do engineers require from modelling systems? What type of tools assist engineers best with the model generation task? Our examination of these questions has led us to view model generation as an iterative design task that uses both experiential and model-based knowledge. Consequently we have developed a physical modelling system called CoBRA which exploits both model-based and case-based reasoning techniques within derivational analogy framework. We argue that this approach has a number of advantages over other work including; cognitive plausibility, computational tractability, ease of

knowledge acquisition and a more pragmatic engineering approach to model generation. Finally, we believe that it addresses some of concerns raised by researchers from the qualitative reasoning community about the need to firstly, focus more clearly on significant engineering problems, and secondly, to tackle these problems in a manner that is beneficial to the engineering community [22].

The paper is laid out as follows: Section 2 discusses, firstly the issues associated with the physical modelling of heat transfer problems described by PDEs, and secondly our understanding of how engineers carry out physical model generation. Section 3 describes our approach and introduces CoBRA, a system for carrying out physical modelling in heat transfer analysis. Section 4 examines other related work and deals with some of the wider implications of our approach. Section 5 concludes the paper.

## 2 MODEL GENERATION

In this section we discuss the issues associated with the physical modelling of the heat transfer PDEs and outline our understanding and approach to model generation for this problem domain.

### 2.1 Physical modelling in PDE analysis

Convection heat transfer problems can be defined as physical systems where heat transfer occurs between a solid body and a surrounding fluid medium, each at a different temperature. Numerical analysis of convection problems is usually carried out in a number of stages (see Figure 1) which have been identified as follows [3, 18]:

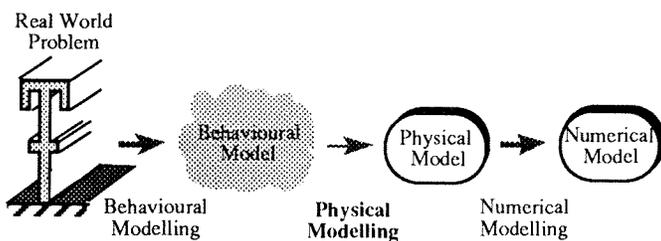


Figure 1 Physical Model Generation

- **Behavioural Analysis** This is the first task in most numerical engineering problems and it involves reasoning about the physical system with the objective of obtaining a behavioural understanding of the underlying phenomena. In this work, we assume that the engineer has already obtained a behavioural understanding of the physical system.

- **Physical Modelling** This phase involves applying idealisations and simplifications to spatial, phenomenological and temporal aspects of the physical

system with the objective of abstracting a mathematical model. This is the focus of the current work.

- **Numerical Simulation** This phase involves creating a numerical model and simulating using numerical techniques such as the finite element method.

- **Visualisation** This stage involves post processing and visualising of the numerical data produced by the simulation process.

Except for simple problems, it is neither feasible nor desirable to analyse all aspects of a physical system. This is because most real world problems contain complexities that render numerical simulation difficult and redundancies that are unnecessary to analyse. In practise, engineers simplify complexities thereby facilitating more efficient computation and ignore redundancies without loss to the integrity of the physical system. In physical model generation the major challenges to the engineer are: identifying the various complexities and redundancies in a physical system, applying appropriate modelling strategies to simplify or reduce these features and assessing the suitability of the resulting model. We consider physical modelling to consist of a number of subtasks including, spatial, phenomenological and temporal modelling.

Spatial modelling focuses on geometric features of the problem domain and involves applying modelling strategies such as: taking a two dimensional idealisation of a three dimensional physical system, finding geometric symmetries or carrying out feature modelling. Figure 2 illustrates feature modelling, and strategies can involve either replacing an existing complex feature with a simpler feature, removing the feature and substituting it with an equivalent boundary condition or removing the feature completely without any compensatory measures.

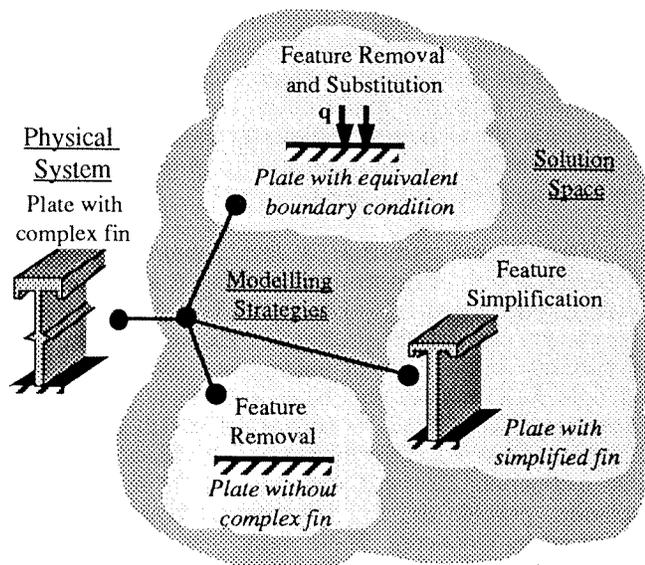


Figure 2 Feature modelling strategies

Phenomenological modelling deals with the construction of a PDE model that describes the thermal heat transfer process. Considering the full thermal PDE, it consists of three equations based on the physical laws of conservation of mass, momentum and energy. Each equation is in turn composed of terms, where each term describes a particular sub-phenomenon. In many heat transfer problems it is not necessary to model all these sub-phenomenon and therefore terms can be either simplified or even be ignored completely.

Temporal modelling involves choosing an appropriate transient or steady state model.

## 2.2 Our approach to physical modelling in PDE analysis

The central argument presented in this paper is that for physical modelling in finite element analysis the existing approach of using model-based reasoning should be augmented with case-based reasoning techniques. This argument is based on two assertions:

- This modelling task is based on a weak domain theory.
- When modelling engineers refer to exemplars and previously solved problems.

This first assertion requires some elaboration because, at first glance, heat transfer analysis is not normally considered to be a weak theory domain. This apparent contradiction exists because there is a strong theory for much of the interaction in heat transfer. The behavioural description that is the input to this modelling process is well understood as is the numerical simulation process (see Figure 1). However, the actual physical modelling process is not. The task of generating a physical model from a behavioural model is an abductive process and competence is based on experience rather than on any comprehensive theory that might be found in an engineering text. Instead, modelling skills and strategies are experiential in nature and are acquired by engineers through experience and practise. In conclusion, the models themselves are based on a strong domain theory but the process of producing and simplifying these models is not.

Our second assertion is less contentious; our experience with engineering modelling is that human experts refer extensively to heat transfer exemplars and previously modelled problems. Exemplars occur extensively in the form of fundamental scenarios that include heat transfer to plates, cylinders, fins, etc. Associated with these exemplars are a rich body of approximations and correlations which facilitate analysis and evaluation. Exemplars in the form of modelling episodes provide the basis for model generation as practised by engineers. These modelling episodes are used as building blocks for designing models for use in simulation. Engineers model by remembering these scenarios and then reason and modify them to fit the current context. These modifications usually involve

'first-principles' reasoning based around approximations and correlations associated with the exemplar. This anecdotal evidence is backed up by research in the related area of engineering design. While there has little work on the integration of CBR in engineering modelling there has been much work on using CBR in design. Arguments that human designers refer to past problem solving episodes are presented in [11, 20, 21].

Summarising then, we argue that the QR research on modelling would benefit from the integration of CBR techniques because that is the way engineers do it. In addition, we argue that the fact that modelling is based on a weak domain theory signals that a CBR approach will be fruitful.

## 3 PHYSICAL MODEL GENERATION IN CoBRA

In this section, we discuss the AI methodology adopted in this work, outline the conceptual architecture of our system and describe the CoBRA system for physical model generation.

### 3.1 AI methodology

CBR is an AI methodology that serves the basic intuition that humans reuse solutions to previously solved problems during problem solving. The most obvious advantage of this approach is that competent systems can be developed based on shallow domain models, thus requiring little knowledge engineering. However, it is generally accepted that CBR systems for design require reasonably deep domain models and much work has been done in this area [4, 11, 15, 16]. CBR systems incorporating deep domain models still have advantages over systems based on first-principles reasoning. The case organisation helps focus the knowledge acquisition process and the cases encode known good routes through the solution space and thus constraining the solution search process [6].

One of the key issues in CBR is the manner in which the cases are adapted. The standard approach is to transform the solution of the old case to meet the specification for the new case. In some circumstances the interdependencies in the solution components are too complex for this to be practical. In this case generative adaptation (derivational analogy) can be used. This involves reworking the steps in the solution generation process in the context of the new problem specification. This is the strategy adopted in CoBRA.

### 3.2 Conceptual architecture

An environment that facilitates interactive modelling between the user and the modelling system was considered to be the most suitable conceptual architecture for tackling physical model generation in heat transfer analysis. This decision was prompted by both design and pragmatic considerations. Design considerations arise from the knowledge that in finite element analysis the majority of users are expected to be

non-naive participants who will have a certain degree of understanding of physical modelling. Thus, this class of user is expected to be familiar with the various modelling tasks (outlined in Section 2.1) but will require expert advice in selecting modelling strategies and applying these strategies or actions in a particular problem context. For the task of physical modelling in finite element analysis, automated modelling or 'black box' approaches do not serve this user group well. Pragmatic considerations were prompted by the realisation of the significant implementational difficulties that were likely to be encountered if the entire modelling process was to be automated. Taking an automated approach would lead to additional formidable technical challenges (for example, feature recognition) which would have detracted from the core issue of examining whether case based reasoning techniques with derivational analogy can be applied to this domain.

Considering now how these ideas are incorporated with the CoBRA modelling system, we summarise our conceptual approach by the following points:

- Modelling is carried out in distinct stages which include phenomenological, spatial and temporal modelling.
- Within any modelling stage, modelling decisions are taken in a piecewise fashion by examining each modelling issue in turn. In this way a physical model is designed in a step by step manner.
- Case based reasoning with model-based generative adaptation forms the core AI approach.
- A case consists of a description of the modelling problem, a modelling solution and a derivational trace.
- Derivational traces consist of a model based reasoning trace by which a modelling solution was reached. They also act as a validation mechanism and explanation facility of the case solution.

Because the user is involved in the model creation process, physical modelling is considered to be a design task in itself. This perspective allows physical modelling to be viewed as an interpretation of the propose-critique-modify design model as proposed by Chandrasekaran [27] for design. In this case, Chandrasekaran's approach is adapted for designing physical models, where, the user examines and *proposes* the sequence of modelling events, case based reasoning tools retrieve similar modelling scenarios with solutions, derivational analogy techniques adapt the solutions and also act as a *critiquing* mechanism in the context of the adapted solution and finally *modification* is carried out by the user applying the modelling action to the physical system.

### 3.3 Case Descriptions in CoBRA

In CoBRA, a target case consists of a frame based representation of the physical system. Frames are created on the basis of geometric information obtained through an AutoCAD interface. In a target frame, representation is organised according to the different modelling stages,

spatial, phenomenological and temporal. This representation is built up component by component by the user, in this case (see Figure 3), a base fin (longitudinal rectangular fin), a secondary appendage (longitudinal rectangular fin) and the additional minor features associated with the appendages (cavities, features, etc.). In spatial modelling, this involves choosing from the user interface, the appropriate qualitative descriptors to define each feature. Feature indices are based on qualitative engineering terms that are used by engineers to describe and distinguish fundamental modelling scenarios. In addition to the feature indices, problem parameters such as geometric data are included in the target cases. However this information is not used as indices, but is included for use in the derivational traces.

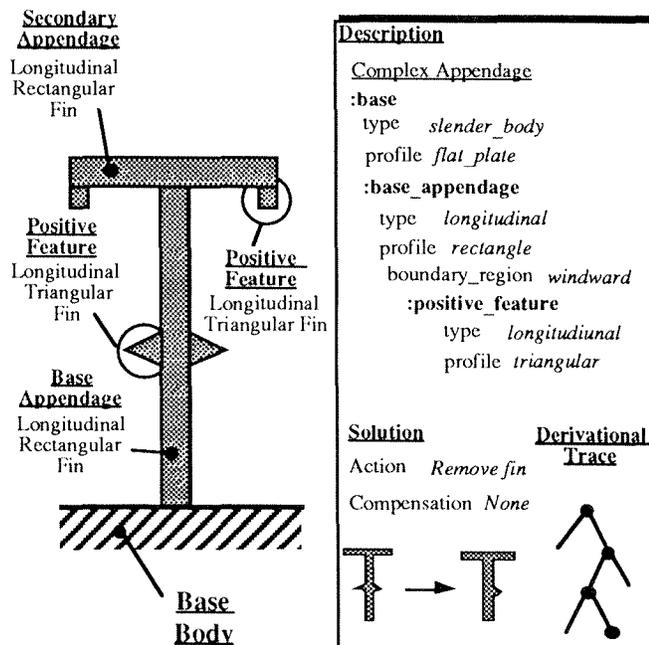


Figure 3 A case in the CoBRA system

A base case consists of a representation of the real world physical system, the solution in the form of a simplified model, and a reasoning trace of the justifications for the transformations in going from the real world problem to the simplified model. Cases are constructed at the level of fundamental modelling scenarios and this determines directly the type of indices used as well as the contents of the derivational traces. Figure 3 illustrates a typical convection heat transfer problem that can be tackled by the modelling system. The physical system is a finned heat exchanger that dissipates heat to the surrounding ambient air, such exchangers are often used as heat sinks in electronic devices. The heat exchanger consists of a base appendage with an associated secondary appendage. Each appendage has additional minor features. The task being addressed by CoBRA in this

example is to produce a simplified model of the physical system. The frame definition on the right of Figure 3 illustrates the problem description, the problem solution and the derivational trace that provided this solution. Indices are based on qualitative engineering terms that are sufficient to distinguish the fundamental modelling scenarios. A target case contains only the problem description; this is the specification of the physical system. Modelling progresses by firstly examining the role of the minor features, and then the importance of the secondary and complex appendage.

Case retrieval is implemented in a two stage process, matching (or base filtering) and mapping. Matching is carried out using an activation network which is made up of activation units which correspond to the indices of the base cases. A feature vector is created for each target case which contains the relevant indices of the problem. The feature vector is the basis by which the activation units are initialised and on completion, each case in the case base contains a value of how many indices it shares with the target case. The mapping stage is concerned with establishing the correspondences between the base cases and the target cases. Mapping based on establishing the full set of matching features between the target and base cases is the criteria used for retrieving cases.

### 3.4 Generative Adaptation using Model based Reasoning

Derivational traces are exploited in this domain, because, although the target and base cases may map qualitatively, small differences between physical parameters such as spatial or medium data can lead to significantly different solutions. Such differences cannot be expected to be captured in the initial qualitative classification of the problem, furthermore, to index all episodes based on both descriptive and parametric indices would result in an intractably large case base. In CoBRA, a derivational trace links the start and goal state of a case *and describes the basis of the modelling solution*. Each reasoning trace has two main components; a *decision part* and a resulting *action part* (after [5]). The decision part contains:

- Alternative modelling strategies considered and rejected
- Assumptions and justifications for the decisions taken.
- Heat transfer approximations and correlations to allow evaluation of a particular modelling strategy.
- Heat transfer domain knowledge describing dependencies of later decisions on earlier ones.

The action part holds the steps taken as a result of the reasoning trace of the decision part. A typical action is, "Remove the feature which faces into the flow". A sample reasoning trace is shown in Figure 4. Each node in the reasoning trace represents a decision point in the model simplification process. In this example, the solution in the base case was derived in two ordered stages. The first stage examines the influence of the

feature on the flow field and consists of Goals 1 and 2. This involves determining whether the feature is actually fully contained within a turbulent boundary layer, and if so, whether the influence of the feature on the flow field is deemed negligible. Goal 3 examines the contribution of the feature to overall heat transfer. In the base case, the heat transfer contribution of the feature was of the order of 4% of total heat transfer well within the 5% constraint, so therefore the fin was removed. In the target case, this contribution was of the order of 3.5% thereby permitting the feature to be removed.

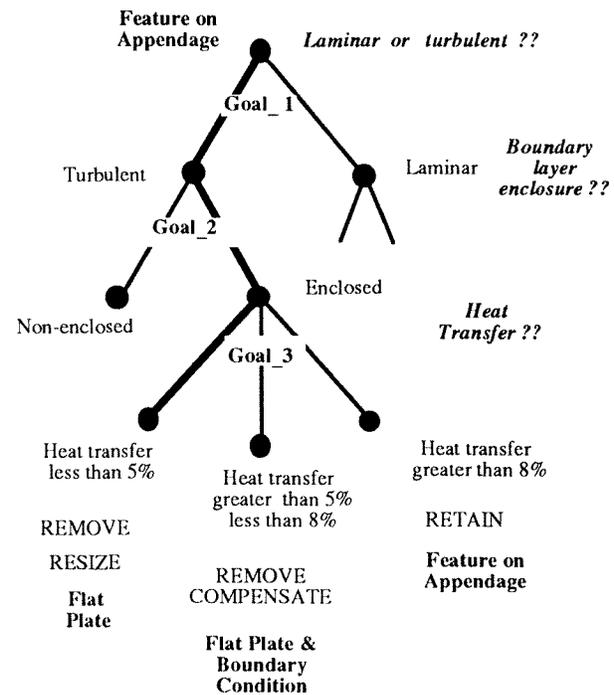


Figure 4 A model-based derivational trace

Derivational traces are based on fundamental modelling scenarios and are created at this level of complexity. Thus the contents of the derivational trace are determined by the context of each scenario in question and this can vary from exemplar to exemplar. Because the derivational traces are created at according to each fundamental modelling scenario, the issue of trade-off between the sufficiency of the indices and the complexity of the derivational traces is essentially pre-determined by the engineering nature and content of the exemplars.

Currently, the derivational traces are constructed and organised on a case by case basis, in other words, a generalised approach based on a common vocabulary and structure has not been used. Because of the varying nature and complexity of the derivational traces, it is likely that such a generalised approach would be

organised at a planning rather than at a modelling level. This is an issue that will be investigated in future work.

#### 4.0 COMPARISON WITH RELATED WORK

In this section we briefly review related work and in this context, compare our approach to model generation.

Addanki [1] describes an automated modelling approach using a methodology called the "graph of models". The basic idea is that system behaviour can be represented by a series of interlinked models which exist at different levels of abstraction. Modelling progresses by automatic selection and changing of analysis models on the basis of assumption satisfaction and model accuracy.

Iwasaki [12] describes a system called Device Modelling Environment that formulates a behavioural model of a device, simulates its behaviour and interprets the results. An input description of the device topological structure is given and a compositional modelling approach formulates the appropriate mathematical model.

Yip [25] describes a system for simplifying the Navier Stokes fluid equations using order of magnitude reasoning within a qualitative reasoning framework. The conceptual approach adopted is rather similar to the way an engineering academic would engage in deriving simplified models. PDE models produced by the system are mathematically complete, but may in some cases have no physical meaning. This modelling task is similar to the phenomenological modelling stage described in Section 2.1

Ling [14] discusses a system for generating sets of PDEs for designing thermal systems described by either algebraic equations, ordinary differential equations and PDEs. Order of magnitude and dimensional analysis techniques are used to heuristically derive a mathematical model. Currently they have implemented their approach for conduction heat transfer problems.

Shephard et al. [17] discuss the various modelling decisions that must be considered when specifying a mathematical model for numerical analysis. They describe an approach based on a rule based expert system for the domain of stress analysis in aircraft structures. Attention is focused on the use of different idealised behavioural models at different levels of abstraction.

Falkenhainer and Forbus [7] describe an approach based on compositional modelling. By using explicit modelling assumptions, domain knowledge can be decomposed into semi-independent fragments, each describing various components of the physical system.

In our work, we deal with physical model generation associated with engineering problems described by PDEs; to date only the work of Yip [25]

and Ling [14] have dealt with this class of problem. Our approach however has some significant differences.

Firstly, rather than deriving models from first principles, we use cases which are based on tried and tested episodes. One advantage is that, in practise for finite element analysis, engineers do not normally derive physical models from first principles (as described by Yip [25]). Instead, our observations have been, that they choose between known good models and then 'tweak' these models to satisfy the problem at hand [9]. Cases with model-based generative adaptation support this approach to modelling more readily. Another advantage is that, cases encode known good routes through weak domain solution spaces thereby avoiding extensive backtracking often associated with model-based approaches [6].

Secondly, we argue that by using case based reasoning techniques, we can capture a body of experiential engineering skills and know-how, that is otherwise difficult to represent by model-based techniques. Our studies of modelling have indicated that engineers make extensive recourse to this type of knowledge when carrying out physical modelling in numerical analysis [9].

Thirdly, from a knowledge engineering perspective, the use of derivational traces means that the knowledge acquisition process is carried out in the context of episodes. This we found provided no special difficulties for our domain expert, which is in contrast to experiences for elicitation of generalised knowledge associated with model based approaches [26].

Fourthly, we argue that this approach meets more closely the needs of engineering practitioners in a number of ways. For instance, compared to the work of Iwasaki [12] which aims to develop a complete modelling and simulation environment, we believe that the emergence of modelling tools that can be integrated between existing CAD and numerical packages will serve engineering needs most usefully [2,3,8,18]. In addition, we believe that such tools should aim to empower engineering analysts, and therefore, it is likely that interactive modelling support systems as advocated in this paper will achieve this aim more readily [9].

#### 5.0 CONCLUSIONS

In this paper we presented an approach to physical model generation that adopts both case based and model based reasoning. This approach has been based on the assertion that physical modelling generation is a poorly understood process and is often carried out using a combination of episodic and first principles reasoning. This argument is backed up by our belief, not only that physical modelling is based on a weak domain theory but also that engineers make extensive use of previous modelling episodes and experiential knowledge when modelling. Furthermore we argue that for physical modelling in PDE analysis, interactive modelling tools

that operate between CAD and numerical analysis systems are likely to be most useful for engineers in physical modelling tasks. Currently the case base consists of about twenty cases and future work will focus on expanding the number of cases so as to increase the coverage of the system. In addition we intend to examine the issue of creating a common vocabulary and structure for more efficient and transparent implementation and representation of the derivational traces.

## REFERENCES

- [1] Addanki, S. Cremonini, R and Penberthy, J.S. 'Graph of Models', *Artificial Intelligence*, **51(1-3)**, 145-177, (1991).
- [2] Amaral S. 'Report on DAPRA-sponsored Workshop on Design. Technical Report LCST-TR-160, LCSR, Rutgers University New Brunswick, 08903, (1991).
- [3] Bathe, K.J., Lee, N.S. and Bucalem M.L. 1990, 'On the Use of Hierarchical Models in Engineering Analysis' *Computer Methods in Applied Mech. and Engineering*, **82(1-3)**, 5-26. (1990).
- [4] Bhansali S., Harandi M., 'Synthesis of UNIX programs using Derivational Analogy', *Machine Learning*, **10**, 7-55, (1993).
- [5] Carbonell J.G., 'Derivational Analogy: A theory of reconstructive problem solving and expertise acquisition', in *Machine Learning Vol. II*, R.S. Michalski, J.G. Carbonell, T.M. Mitchell eds., 371-392, (1986).
- [6] Cunningham, P., Finn, D., Slattery, S. 'Knowledge Engineering Requirements in Derivational Analogy' in *Proc. of the First European Workshop on Case-Based Reasoning*, Kaiserslautern, Germany, November. In print (1994).
- [7] Falkenhainer, B. and Forbus, K.D., 'Compositional Modelling: finding the right model for the job', *Artificial Intelligence*, **51(1-3)**, 95-143. (1991).
- [8] Finger, S. and Dixon J.R., 'A Review of Research in Mechanical Engineering Design. Part II' *Research in Engineering Design*, **1**, 121-137, (1989).
- [9] Finn, D.P., 'Preliminary Stages of Engineering Analysis and Modelling: Workshop Summary, 2nd International Conf. on Artificial Intelligence in Design (AID '92).' *Artificial Intelligence in Engineering Design Manufacturing and Analysis*, (AI EDAM Special Issue: Guest-Editor: D. Finn), **7(4)**, 231-237, (1993).
- [10] Finn, D., Slattery, S., Cunningham, P. 'Modelling of Engineering Thermal Problems - An Implementation using CBR with Derivational Analogy' in *Proceedings of the First European Workshop on Case-Based Reasoning*, Kaiserslautern, Germany, November, 114-119, (1993).
- [11] Goel A. K., Kolodner J. L., Pearce M., Billington R., Zimring C., 'Towards a Case-Based Tool for Aiding Conceptual Design Problem Solving', In *Proceedings of DARPA Case-Based Reasoning Workshop 1991*, Washington, D.C., San Mateo, California: Morgan Kaufmann, , (1991).
- [12] Iwasaki, Y., Low C.M. 'Model Generation and Simulation of Device Behaviour with Continuous and Discrete Changes.' *Intelligent Systems Engineering*. **1(2)**, (1993).
- [13] Kirayama, T. , Tomiyama, T. 'Reasoning about Models across Multiple Ontologies' *Proceedings of the Seventh Int. Workshop on Qualitative Reasoning about Physical Systems*. University of Washington, pp. 124-131, (1993).
- [14] Ling, S. , Steinberg, L. 'Approximation Operators in Distributed Modelling.' *Proceedings of the Seventh International Workshop on Qualitative Reasoning about Physical Systems*. Dept. of Computer Science, University of Washington, pp. 138-144, (1993)
- [15] Mostow J., 'Design by Derivational Analogy: Issues in the automated replay of design plans', *Artificial Intelligence*, **40**, pp119-184, (1989).
- [16] Smyth B., Cunningham P., 'Déjà Vu: A Hierarchical Case-Based Reasoning System for Software Design', in *Proceedings of 10th. European Conference on Artificial Intelligence*, Vienna, Austria, 587-589, (1992).
- [17] Shephard, M., Bachmann, P., Georges, M. and Korngold, E., 'Framework for Reliable Generation and Control of Analysis Idealisations' *Computer Methods in Applied Mechanics and Engineering*, **82(1-3)**, 257-280. (1990).
- [18] Tworzydło, W. and Oden, J.. 1990. 'Towards an Automated Environment in Computational Mechanics' *Comp. Methods in App. Mechanics & Engineering*, **104(1-3)**, 87-143, (1993).
- [19] Veloso M., Carbonell J.G., 'Learning by analogical replay in PRODIGY: first results' *European Working Session on Learning*, Y. Kodratoff, ed., pp375-389, Porto, Portugal, Springer Verlag, (1991).
- [20] Visser W., 'Planning in routine design: some counter intuitive data from empirical studies' In J.S. Gero & F. Sudweeks (Eds.) *Workshop on Artificial Intelligence in Design at Twelfth International Joint Conference on Artificial Intelligence*. University of Sydney, Australia, (1991).
- [21] Visser W., 'Reuse of designs: Desperately seeking an interdisciplinary cognitive approach' In W. Visser (Ed.) *Workshop on Reuse of Designs at Thirteenth International Joint Conference on Artificial Intelligence*. (1993).
- [22] Weld, D.S. and DeKleer, J., 'Multiple Ontologies and Automated Modelling' *Readings in Qualitative Reasoning about Physical Systems*, 481-483, Morgan Kaufmann, (1990)
- [23] Weld, D.S 'Introduction to Forum. Taking Issue/Forum: Prolegomena to any Future Qualitative Physics' *Comp. Intelligence*, **(8)2**, pp.175-186, (1992).

- [24] Williams and deKleer 'Qualitative Reasoning about Physical Systems: a Return to Roots', *Artificial Intelligence*, **51(1-3)**, 1-9., (1991).
- [25] Yip., K. 'Model simplification by asymptotic order of magnitude reasoning' In: D. Weld (ed): *Working Papers Qualitative Reasoning '93 (QR '93)*. Seattle: University of Washington, 266-272. (1994)
- [26] Janetzko, D., Straube, G. 'Case-based Reasoning and Model based Knowledge Acquisition.' *Complementary Knowledge Engineering and Cognition. Lecture Notes in Artificial Intelligence*. Springer Verlag, 99-114 (1991).
- [27] Chandrasekaran, B. 'Design Problem Solving: A Task Analysis' *AI Magazine*, **11(4)**, 59 - 71, (1990).

# Polynomial-time compilation of Self-Explanatory Simulators

Kenneth D. Forbus  
Qualitative Reasoning Group, The Institute for the Learning Sciences  
Northwestern University  
1890 Maple Avenue, Evanston, IL, 60201, USA

Brian Falkenhainer  
Xerox Modeling Research & Technology  
1350 Jefferson Rd, Henrietta, NY, USA

**Abstract:** Self-explanatory simulators have many potential applications, including supporting engineering activities, intelligent tutoring systems, and computer-based training systems. Yet compilation methods have been too slow for large-scale systems, interpreter-based strategies are restricted to running on large computers with expensive commercial software, and neither technology has been shown to scale to very large systems. This paper describes an algorithm for compiling self-explanatory simulators that operates in polynomial time. It is capable of constructing self-explanatory simulators with thousands of parameters. This algorithm is fully implemented, and we show empirical evidence that suggests that its performance is quadratic in the size of the system being analyzed. We also analyze the tradeoffs between compilers and interpreters for self-explanatory simulation in terms of application-imposed constraints, and discuss plans for applications.

## 1. Introduction

Self-explanatory simulators [1, 2, 3, 4] integrate qualitative and quantitative knowledge to produce both detailed descriptions of the behavior of a system and causal explanations of how that behavior comes about. They have many potential applications, such as in intelligent tutoring systems and learning environments [5, 6] and in supporting the design process [7, 8]. Realizing this potential requires both developing systems that can operate efficiently on substantial models and understanding the tradeoffs involved in the automatic construction of self-explanatory simulators. This paper makes two contributions towards these goals. First, we describe a polynomial-time method for compiling self-explanatory simulators, and show that it operates successfully and quickly on models larger than most industrial applications require. Second, we analyze how the quality of the simulator produced trades off against the time taken to construct it, and consider how these tradeoffs affect potential applications. Throughout, the discussion is limited to initial-value simulations of lumped-element (ordinary differential-algebraic) systems.

Section 2 reviews the basic idea of self-explanatory simulators and the relevant literature. Section 3 describes our new polynomial-time compilation technique, including both theoretical and empirical complexity analyses. Section 4 identifies tradeoffs in constructing self-explanatory simulators in light of task requirements. Section 5 summarizes and outlines our plans for future work.

## 2. Self-explanatory simulation: The basics

Traditional numerical simulators generate predictions of behavior via numerical computation using quantitative models of physical phenomena. Most simulators are written by hand, although an increasing number are generated by domain-specific toolkits (e.g., SPICE for electronics). The modeling decisions, such as what phenomena are important to consider, how does the phenomena work, how can it be modeled quantitatively, and how can the quantitative model be implemented for efficient computer solution, are mostly made by hand. Domain-specific toolkits provide fairly good solutions to the last two problems, and by what libraries they do or do not include, can simplify the first problem. However, the choices of how to translate the physical description into the conceptual entities supported by the toolkit, and which quantitative model to use from the library provided by the toolkit to model an entity, are still made by hand. Moreover, no existing toolkit provides the intuitive explanations used by scientists and engineers to describe how a system works. These intuitive, qualitative descriptions serve several important purposes in building and working with simulations. First, they guide the formulation of quantitative models by identifying what aspects of the physical situation are relevant. Second, qualitative descriptions are used to check the results of a simulation, to ensure that it "makes sense." Thus two advantages of self-explanatory simulation are *increased automation* and *better explanations* [1].

Self-explanatory simulators harness the formalisms of qualitative physics to automate the process of creating simulators. Given an initial physical description, a qualitative analysis of the situation reveals what conceptual entities are relevant to the task at hand, and identifies what causal factors affect a parameter under different circumstances. This information is then used, in concert with quantitative information in the domain theory, to construct appropriate numerical programs for simulating the system. By incorporating explicit representations of conceptual entities (such as physical processes) in the simulator, causal explanations can be given for the simulated behavior. Moreover, the qualitative representations provide some of the same opportunities for automatic "reality checks" that an expert would apply in evaluating a simulation. For instance, a simulation of a fluid system which reported a negative amount

- 1 Create a scenario model via instantiation of model fragments from the domain theory.
- 2 Analyze the scenario model to define the appropriate notion of state for the simulator:
  - 2.1 Extract physical and conceptual entities.
  - 2.2 Define boolean parameters (i.e., logical variables corresponding to conditions such as valves being open or closed, or physical processes being active).
  - 2.3 Define numerical parameters & relevant comparisons among them.
  - 2.4 Extract influences to create causal ordering
- 3 Write simulator code
  - 3.1 Simplify boolean parameters
  - 3.2 Compute update order for numerical parameters using the influence graph and for boolean parameters using logical dependencies between them.
  - 3.3 Write code to evolve state descriptions (*evolver*).
  - 3.4 Write code to detect state transitions (*transition finder*)
  - 3.5 Write code to detect inconsistencies (*nogood checker*)
  - 3.6 Write structured explanation system

Figure 1: The SIMGEN Mk3 Algorithm

of liquid in a container is not producing realistic results. The ability to detect such conditions is called *self-monitoring*.

The first systems to generate self-explanatory simulators were compilers. That is, the creation of a simulator was done off-line, with the goal of producing code whose execution would asymptotically approach the speed of traditional numerical simulators while providing services they did not (i.e., explanations and increased self-monitoring). This goal was met, but only at the cost of high compilation times. For example, SIMGEN Mk1 [1] used envisioning for its qualitative analysis procedure. This provided powerful explanation capabilities (including answering counterfactual questions via comparing the simulated behavior to alternatives in the environment) and a high degree of self-monitoring, because the numerical state of the simulator could be checked against a complete qualitative state. Unfortunately, envisioning, like all non-resource limited forms of qualitative simulation, is exponential in the size of the system being analyzed. A closer analysis of what simulation authors do led to the substitution of a simpler qualitative analysis system: Qualitative simulation is simply not necessary for simulation construction. A person writing a numerical simulator never explicitly identifies (with the possible exception of defining initial conditions) even a single global qualitative state of the system being simulated. SIMGEN Mk2 [2] used a qualitative analysis procedure that avoided the obviously exponential steps in qualitative reasoning that are unnecessary given quantitative information, such as branching on all ways to resolve an ambiguous influence on a variable or all possible combinations of state transitions. This compiler could construct simulators of systems larger than any envisioning-based system ever could (i.e., involving dozens to hundreds of parameters), such as a twenty stage distillation column [9]. The tradeoff is that some explanatory capabilities (i.e., efficient counterfactual reasoning) and self-monitoring capabilities (i.e., the guarantee that every numerical simulator state satisfied some legal qualitative state) were lost. However, as Section 3 explains, even this compiler was still subject to combinatorial explosions.

An alternative to compiling self-explanatory simulators is to build simulators that act as *interpreters*, i.e., that interleave model-building, model translation into executable code, and

code execution[3,4]. In PIKA [4], for example, Mathematica is used in conjunction with a causal ordering algorithm to produce a decomposition of a set of equations into independent and dependent parameters, along with an order of computation to update dependent parameters. Every state transition which changes the set of applicable model fragments reinvokes Mathematica and causal ordering to produce a new simulator for the new state. (An incremental constraint system was proposed to minimize this cost.) Part of the motivation for such systems was the perceived slowness of compiler techniques: By only building models for behaviors that are known to be relevant, presumably the entire time from formulation of the problem to solution would be reduced, even though any particular execution of a simulator might be slower due to the need to perform reasoning during simulation. Such systems are not themselves immune from combinatorial explosions (see Section 4), but on some examples can exhibit impressive performance.

It should be noted that some of the specific performance claims made for PIKA are problematic, e.g., in [4] it is claimed that PIKA is "5,000 times faster than SIMGEN Mk2." There are several minor problems with this claim, such as the fact that the performance difference is ten times less on the other example for which data is available about both systems (e.g., boiling water), and it is not clear what the relative performance difference between the different computers used is. However, the worst problem is that the domain theories used by each system are substantially different: PIKA uses just two model fragments, with choices for quantitative models "hard wired" into these fragments. By contrast, in keeping with the goal of increased automation, the domain models used in SIMGEN Mk2 were basically the same as used in other qualitative reasoning systems, with quantitative information added in a modular fashion. In the SIMGEN Mk2 quantitative models, for instance, quantitative parameters such as fluid and thermal conductances and container sizes were explicit variables that could be set by the simulation user at run time. In PIKA such information was hard-wired into the model fragments in the form of example-specific numerical constants, which is not very realistic.

### 3.SIMGEN Mk3: A polynomial-time compiler for self-explanatory simulators

Previous work has shown that the advantages of self-explanatory simulation can be achieved in several ways. To better understand the tradeoffs between these methods, we wanted to figure out how fast self-explanatory simulation compilers could be. If such systems were inherently exponential, then the range of applications for them would be strictly limited. If, on the other hand, self-explanatory simulators could be compiled in polynomial time (preferably low-order, of course), then with enough software engineering such compilers could be used in a broad range of applications.

We have succeeded in developing a polynomial-time algorithm for compiling self-explanatory simulators. This has required some important simplifications, which reduce some of the advantages of self-explanatory simulators. However, the ability to quickly generate simulators for systems containing thousands of parameters suggests that these simplifications are worthwhile.

The rest of this section describes our algorithm. As we explain the algorithm we analyze its complexity. Since each step contains so many subprocesses, we only show that each step is polynomial, instead of attempting to theoretically derive a concrete bound. We evaluate its performance, and show empirical data that suggests that its performance is quadratic in the size of the input description.

#### 3.1 The SIMGEN Mk3 Algorithm

The structure of the algorithm is shown in Figure 1. Its overall structure is similar to SIMGEN Mk2 [2], so in the rest of this section we focus mainly on the tradeoffs made to achieve polynomial time performance.

##### 3.1.1 Creation of the scenario model

We assume domain theories are written in a compositional modeling language, using Qualitative Process theory [10] to provide their qualitative aspects. A key tradeoff is how much qualitative reasoning should be performed. More qualitative reasoning provides more constraints on the system's behavior, which can be exploited to generate more compact code and better self-monitoring, but at the price of more inference. In fact, the cost of qualitative reasoning was by far the dominant cost in SIMGEN Mk1 and Mk2. Interpreters like PIKA [4] and DME [3] appear to do no qualitative reasoning beyond instantiating model fragments corresponding to equations, which we suspect is the main factor responsible for their efficiency. Consequently, we minimized the amount of qualitative reasoning in this compiler to see the consequences of that design decision. Specifically, we use a qualitative reasoner to instantiate model fragments and draw certain trivial conclusions (i.e., if  $A > B$  then  $\neg A = B$ ). No transitivity inferences are made: In earlier compilers, these inferences were the dominant cost in qualitative reasoning, both because the number of such conclusions rises combinatorially with the size of the systems and the ATMS label update algorithm tended to go exponential in that subsystem. No attempt is made to

resolve influences, nor to compute possible state transition conditions (i.e., limit analysis). The consequences of this decision are made clear in subsequent sections.

We use TGIZMO, a publically available QP implementation [11] for our qualitative reasoner. We modified it in two ways. First, the pattern-directed rules that implement many important QP operations were simplified, to strip out aspects of reasoning not needed by the compiler (e.g., transitivity reasoning over ordinal relations). Second, the modeling language implementation was modified so that logical antecedents for specific kinds of facts (e.g., whether or not a physical process is active) are explicitly asserted in the database as well as being represented via clauses in TGIZMO's LTMS [12]. This information is needed by the compiler in order to write code for updating the truth values of dynamic booleans, i.e., those statements whose truth values may change during simulation. In previous compilers this information was gleaned from the statement's ATMS label, but we chose to use an LTMS [12] instead to avoid the potential exponential growth of labels [13].

The only combinatorial explosions that occurred in previous compilers occurred in the qualitative analysis phase, so the complexity of this step is crucial. Its time complexity is a function of the cost of instantiating model fragments and the cost of drawing conclusions with them. The cost of instantiation can be decomposed into two factors: The cost of pattern matching, and the size of the description produced by the operation of the system's rules. The cost of pattern-matching is polynomial in the number of antecedents [11]. We assume that both the scenario model and the domain theory are finite, and that the number of new entities introduced by the domain theory for any scenario model is a polynomial function of the size of the scenario model. We further assume that at worst the number of clauses instantiated about any particular statement in the scenario model is bounded by a polynomial. (It is easy to construct domain theories which violate these assumptions if one tries [14], but in practice these assumptions are always satisfied.) Since the work of instantiation is the product of the number of instantiations and the work to perform each, the instantiation process is polynomial-time. Furthermore, the dependency network so created is polynomial in size, as a function of the size of the domain theory and scenario model. This means that the cost of inference remains polynomial in these factors, since we use an LTMS, for which the cost of inference is worst-case linear in the size of the dependency network [11]. We thus conclude that the time complexity of this step is polynomial.

##### 3.1.2 Constructing the simulator's state

In this step the results of the qualitative analysis are harvested to create specifications for what information comprises a state of the simulator. The numerical parameters of the state include the quantities mentioned in the qualitative model. The statements for which boolean parameters are introduced are the existence of individuals, the existence of quantities, activation status of processes and views, and any statements mentioned in the antecedents of these EXIST and ACTIVE statements (except for ordinal relations, which are handled separately). Each boolean parameter has an associated *antecedents* statement, a necessary and sufficient condition for the truth of the corresponding statement.

Any truth values known in the scenario model are presumed to hold universally over any use of the simulator. If for example a fluid path is assumed to be aligned, every behavior of the simulator will be generated assuming this fact. (The compiler is clever enough to not generate simulator parameters for such statements, although they are still woven into the explanation system appropriately.) Every truth value that is not known is treated as something that must be ascertained at runtime. This technique allows the compiler to produce tighter code by exploiting constraints of the domain and any hints from the user. A simple symbolic evaluation procedure is used to test such constraints. This symbolic evaluator is used for such tasks as ascertaining what statements are universal and simplifying antecedents statements to produce tighter code.

Most of the work in this step consists of fetching information from the TGIZMO database and constructing corresponding internal datastructures in the compiler, which is obviously polynomial time. The only other potentially expensive part of this computation is the symbolic evaluation procedure. This procedure is simply a recursive analysis of propositional statements, checking the (preexisting) LTMS labels of ground terms at the leaves, and so it too is polynomial.

By comparison, when an ATMS is used such simplifications are automatically performed by the label update mechanism, which can take exponential time. The extra information available in ATMS labels was used for several optimizations in SIMGEN Mk2, including merging tests for ordinal relations that could be proven to be equivalent.

### 3.1.3 Writing the simulator code

The ATMS provided a direct connection between a fact and the assumptions underlying it, irregardless of the structure of the dependency network between them. While the use of explicit antecedents is much cheaper, there is a certain inelegance (and runtime inefficiency) in not being able to collapse long chains of inference. (It seems unfair to penalize domain modelers who use compositional modeling appropriately, i.e., by decomposing knowledge into small fragments which are then woven together inferentially in model formulation.) The symbolic evaluator mentioned above addresses part of this problem. The other technique we use (in step 3.1) is to divide the boolean parameters into equivalence classes with canonical members, according to logical dependency. That is, if a boolean parameter *A* depends only on *B*, and *B* in turn depends only on *C*, and *C* either has an empty antecedent or an antecedent with more than one ground term, then *A*, *B*, and *C* would be in the same equivalence class, and *C* would be its canonical member. Each such equivalence class is represented by a single boolean parameter (although each original statement is still part of the explanation system to preserve clarity). Since dividing a set into equivalence classes is polynomial time, this step is also.

In regard to Step 3.2, the *state space assumption*, common in engineering and satisfied by QP models [15], guarantees we can always divide the set of parameters into dependent and independent parts, with the independent parameters being those which are directly influenced (or uninfluenced) and with the dependent parameters computed from them. To gain a similar guarantee for the boolean parameters we must assume that the domain theory is *condition grounded* [14], which again is reasonable for all the domain theories we have seen in practice.

An independent boolean parameter mentions no other boolean parameters in its antecedents. It could be universally true or false, it could be something whose truth value is ascertained at runtime, either as a consequence of the simulation user's assumptions (e.g., the state of a valve) or ordinal relations (e.g., the existence of a contained liquid when there is a non-zero amount of water in the container). A boolean parameter that is not independent is *dependent*.

The update order is found for both numerical and boolean dependent parameters by sorting them according to their maximum distance from the independent parameters, using the graph of influences in the numerical case and the antecedent relations in the boolean case. This is clearly a polynomial-time process. (For comparison, the computation of a boolean update order was unnecessary when an ATMS was used, because the labels could be processed to find an appropriate set of antecedents. Aside from the cost of label updating, the ATMS method could end up producing less efficient code, by not taking advantage of the caching offered via intermediate parameters to eliminate redundant tests.

The overall structure of the code produced by the compiler in steps 3.3 through 3.5 is the same as that produced by SIMGEN Mk2. In evolvers, the effects of direct influences are calculated first to estimate derivatives, the dependent numerical parameters are then updated, followed by the boolean parameters.<sup>1</sup> In transition finders, the limit points of the system are tested to see if any state transitions have occurred, and rollback signals are generated to allow the simulator to modulate its step size to ensure that state transition points are included in the simulated behavior. In nogood checkers, logical constraints are tested against the state of a simulator to warn if the numerical model has diverged from what is qualitatively legal. We summarize the important changes in how they are generated that are caused by restricted inferencing.

The main impact of restricted inferencing in generating evolvers is in the selection of quantitative models for updating dependent numerical parameters. For instance, a domain theory might have two quantitative models for how the level of liquid in a container depends on its amount, one for cylindrical containers and one for rectangular containers.<sup>2</sup> If the compiler knows the shape of the container it can install the appropriate model, otherwise it must write a runtime conditional and provide both models in the simulator. In SIMGEN Mk2 influence resolution was performed to see what combinations of qualitative proportionalities might co-occur, so that appropriate quantitative models could be constructed for each combination. For efficiency this compiler eschews influence resolution, using instead the assumption that the domain modeler has supplied, for each class of dependent parameter, an appropriate set of quantitative models (specified in a manner similar to [1]). The antecedents for these models are used to construct a runtime conditional, ensuring that each model is executed as appropriate. Here compile-time error checking has been sacrificed to efficiency: previous compilers would detect when a quantitative model was not available for a logically possible condition.

<sup>1</sup> The default output mode uses Euler integration, since that is often the method of choice for training simulators and minimizes runtime costs, so that we can produce code which runs well on very small machines. However, we have decomposed evolvers into subroutines that can be used with more complex integration methods when needed.

<sup>2</sup> One of our domain theories in fact includes these models.

Example	SIMGEN Mk3	SIMGEN Mk2
Two containers	6.42 seconds	22.8 seconds
Boiling water	5.32 seconds	25.2 seconds
Spring/Block	1.85 seconds	6.4 seconds
3x3 grid of containers	54 seconds	16286 seconds

Table 1: Compilation times for SIMGEN Mk3 vs Mk2 on standard test examples (IBM RS/6000 Model 530, 128MB RAM, Lucid Common Lisp 4.01)

In generating transition finders, restricted inference can result in "dead code," i.e., runtime tests that are moot because they will never occur. Given how cheap inequality tests are, this is not a serious drawback, and many of them are avoided by using the symbolic evaluation procedure to exploit any information that is available about a comparison. Generating nogood checkers is also greatly simplified: Previous versions filtered the ATMS nogood database to find contradictory combinations of assumptions that, if detected in the runtime system, indicated that something is amiss. Empirically, those the filter conditions had to be quite strong, since most of the nogoods would never arise, given the definition of qualitative state in terms of known numerical parameters. SIMGEN Mk3 simply uses the symbolic evaluation procedure to see what ordinal relations are known to be impossible and test for those. In principle this could result in reduced self-monitoring, but in practice this appears to be negligible.

Each of these computations involves simple polynomial-time operations (see [2] and Section 3.1.2) over structures whose size is polynomial in the initial scenario description, so they are polynomial time as well.

### 3.1.4 Writing the explanation system

The final step in generating a self-explanatory simulator is creating a runtime system that will provide the same causal explanations that were available in the original qualitative analysis, as well as links to the quantitative models used. For this purpose we use a *structured explanation system* that concisely summarizes the qualitative and quantitative analyses. Such systems provide an abstraction layer between a reasoning system and an interface that allows each to be optimized independently. Every conceptual entity, every boolean parameter, and every numerical parameter has an associated element in the explanation system, as well as every influence and every mathematical model. These explanation elements have associated procedures that enable them to evaluate whether or not they hold at any state of the simulation, so that a user can get explanations either while the simulator is operating or as a post-mortem. These explanations are more detailed than those generated by equation-based systems such as PIKA, since they can respond both in qualitative and quantitative terms.

Generating a structured explanation system requires a case analysis of the elements used in earlier steps of the compiler construction. This analysis selects the appropriate class of explanation element and creates the necessary pointers between it and other such elements to provide coherent causal

explanations. This translation procedure is linear time, in the size of the results of the qualitative analysis and the number of quantitative models used by the system. Importantly, restricted inferencing has little effect on the quality of the explanation system: Models are still completely instantiated, so full ontological and causal information remains available.

### 3.2 Empirical Results

SIMGEN Mk3 is fully implemented, and has been tested successfully on the suite of examples described in [2]. In all cases it is substantially faster than SIMGEN Mk2, as Table 1 shows.

The simulators it produces, like those of SIMGEN Mk2, operate at basically the speed of a traditional numerical simulator, with the only extra runtime overhead being the maintenance of a concise history [1] for explanation generation. Currently the compiler's output is Common Lisp, and even with this performance handicap, the simulators it produces run quite well on even small machines (i.e., Macintosh Powerbooks).

To demonstrate that SIMGEN Mk3's performance is in fact polynomial time, we generated a set of test examples similar to those used in [2]. That is, a scenario description of size  $n$  consists of an  $n$  by  $n$  grid of containers, connected in Manhattan fashion by fluid paths. We generated a sequence of scenario descriptions, with  $n$  ranging from 2 to 10. (The reason we chose 10 as an upper bound is that the simulator which results contains just over 2,400 parameters, which is roughly three times the size of the STEAMER engine room mathematical model [16]) Extending the domain theory in [11], contained liquids include mass, volume, level, pressure, internal energy, and temperature as dynamical parameters, as well as other static parameters (e.g., boiling temperature, specific heat, density, etc.). Containers can be either cylindrical or rectangular, with appropriate numerical dimensions in each case. The liquid flow process affects both mass and internal energy. We then ran the compiler to produce simulators for each scenario, to see how its performance scaled. The results are shown in Table 2. In an  $n \times n$  grid scenario, there are  $n^2$  containers and  $2[n^2 - n]$  fluid paths, so the numbers of parts in these examples ranges from 8 to 280. The count for quantities includes both static and dynamic parameters, and the count for booleans includes both conditions controllable by the user (e.g., the state of valves) and qualitative state parameters, such as whether or not a particular physical process is occurring. The proposition count is the number of statements in the simulator's explanation system.

Grid Size	# parts	# quantities	# booleans	# propositions	Compile time (seconds)
2	8	83	24	456	6
3	21	198	63	1131	19
4	40	363	120	2108	49
5	65	578	195	3387	105
6	96	843	288	4968	202
7	133	958	399	6851	356
8	176	1523	528	9036	586
9	225	1938	675	11523	927
10	280	2403	840	14312	1429

**Table 2: Results of SIMGEN Mk3 on  $n \times n$  Manhattan grid**  
(IBM RS/6000 Model 350, 64MB RAM, Lucid Common Lisp 4.01)

The theoretical analysis in previous sections suggests that the compile time should be polynomial in the number of parts in the system. A least-squares analysis indicates that this is correct: A quadratic model ( $0.017P^2 + 0.399P + 4.586$ , where  $P$  is the number of containers and paths) fits this data nicely, with  $X^2 = 0.03$ . Additional evidence for quadratic performance is found in Table 3, which shows the compiler's performance on examples constructed out of chains of containers. A chain of length  $N$  has  $2N-1$  parts, i.e.,  $N$  containers and  $N-1$  fluid paths. A least-squares analysis indicates again that a quadratic model ( $0.018P^2 + 0.554P + 0.228$ , where  $P$  is the number of containers and paths) fits this data well, with  $X^2 = 0.004$ .

Additional tests are in progress. For example, we plan to translate Sgouros' distillation theory [9] into the simpler format used by SIMGEN Mk3 to measure the performance improvement on it. Since this example was larger than the  $3 \times 3$  container grid, and yet was compiled by SIMGEN Mk2 in less time than that example (1.5 hours versus four hours), we expect substantial speedup on this problem as well.

#### 4. Tradeoffs in self-explanatory simulators

Different applications entail different tradeoffs: In some cases potential users have powerful workstations and can afford the best commercial software (e.g., many engineering organizations), and in some cases potential users have only hand-me-down computers and publically available software (e.g., most US schools). Here we examine the tradeoffs in self-explanatory simulation methods with respect to potential applications.

Broadly speaking, the computations associated with self-explanatory simulations can be divided into three types: (1) *model instantiation*, in which the first-order domain theory is applied to the ground scenario description, (2) *model translation*, in which the equations associated with a state are identified, analyzed, and converted into an executable form, and (3) *model execution*, i.e., using numeric integration to derive descriptions of behavior from a given set of initial values. The choice of compiler versus interpreter is mainly a choice of how to apportion these computations, and the tradeoffs are analogous to those of programming language interpreters and

compilers. Interpreters are more suited for highly interactive circumstances, where a substantial fraction of effort is spent changing models compared to running them. Scientists and engineers formulating and testing models of new phenomena and highly interactive, exploratory simulation environments for education are two such applications. Compilers are more suitable for circumstances where the additional cost of compilation is offset by repeated use of the model, or when the environment for model execution cannot support the resources required by the development environment. Engineering analysis and design, where a small number of models are used many times (e.g., in numerical optimization of system properties), and most educational software and training simulators, where maximum performance must be squeezed out of available hardware, are applications where compilers have the edge.

The cost of model generation is dominated by the expressiveness of the representation language for models and the amount of simulator optimization that is performed. In SIMGEN Mk3, the order of computation is specified as an inherent part of the domain theory due to the causal ordering imposed by qualitative process theory influences. Thus, no algebraic manipulation is required at model generation time. Other systems allow a domain theory to contain equations in an arbitrary form. Thus, the equations must be sorted (using a causal ordering algorithm [7]) and symbolically reformulated to match that sort. This technique provides the ease of using arbitrarily-ordered arithmetic expressions, but can lead to exponential behavior for some classes of equations.

Chain Length	# quantities	# booleans	# propositions	Compile Time (sec)
2	38	9	194	2.05
3	58	15	305	3.43
4	78	21	416	5.02
5	98	27	527	6.66
6	118	33	638	8.72
7	138	39	749	10.5
8	158	45	860	12.5
9	178	51	971	14.8
10	198	57	1082	17.8
11	218	63	1193	19.9
12	238	69	1304	22.6
13	258	75	1415	25.6
14	278	81	1526	28.4
15	298	87	1637	31.9
16	318	93	1748	35.1

Table 3: SIMGEN Mk 3 data, linear chain of containers

(IBM RS/6000, 64MB RAM, Lucid Common Lisp 4.01)

Another way in which the representation language for models affects potential applications is in the kinds of explanations that can be generated. Domain theories that explicitly represent conceptual entities as well as equations can provide better explanations than those which do not. While in a few domains (e.g., electronics) expert causal intuitions are not strongly directional, in many domains (e.g., fluids, mechanics, thermodynamics, chemistry, etc.) expert causal intuitions are strongly directed [17], and there is no a priori guarantee that the causal accounts produced by causal ordering will match expert intuitions [18]. Using equation-based models reduces the overhead of uncovering and formalizing expert intuitions, but at the cost of reducing explanation quality. Using explicit qualitative representations provides an additional layer of explanations, but at the cost of increased domain theory development time. Interestingly, TGIZMO accounts for less than 15% of SIMGEN Mk3's time, so the penalty for using rich, compositional domain theories appears to be quite small. How these design choices fare in real applications is, of course, an empirical question, and characteristics of task environments often prove surprising. For instance, in [4] it is suggested that PIKA "...isn't quite fast enough to drive a truly interactive simulation [for an embedded multimedia system]" This assumes that the user requires instant feedback on arbitrary model changes. We doubt that this assumption is correct in practice; for example, precompiling a set of simulations for common variations of particular examples would probably cover the majority of interactions with a user community, and our experience with other educational software suggests that users who wanted to try something novel wouldn't mind waiting a minute or two for their simulation. On the other hand, our working assumption that self-explanatory simulation via interpreters is too resource-intensive for most educational applications could be proven wrong by the combination of advances in computer technology coupled with domain-specific algebraic manipulation systems. Interestingly, even our current implementation of SIMGEN Mk3 can, running on a PowerBook, compile new simulators for small systems reasonably quickly. Both kinds of systems may end up on students' desks and in their homes in the near future.

## 5. Discussion

Previous work on self-explanatory simulation has produced systems that can handle medium-sized systems (e.g., a few dozen to a few hundred parameters). In this paper we describe a new algorithm for compiling self-explanatory simulators that extends the range of the technology to systems involving thousands of parameters. We have shown, both theoretically and empirically, that self-explanatory simulators can be compiled in polynomial time, as a function of the size of the input description and the domain theory. This advance was made possible by the observation that minimizing inference could substantially improve performance [4]. These gains are not without costs: SIMGEN Mk3 does less self-monitoring and less compile-time error detection than previous versions. Algebraic manipulation is neither performed at compile time nor at run time, for example, and the simulators produced can contain code that will never actually be executed. On the other hand, no explanatory capability is lost over SIMGEN Mk2, and the ability to run rapidly on small examples, and to scale up to very large systems, outweighs these drawbacks for most applications.

One open question concerns the possibility of recovering most, if not all, of the self-monitoring and error checking of previous compilers by the judicious use of hints. Many programming language compilers accept advice from programmers, in the form of declarations. Qualitative representations can be viewed as declarations, providing advice to self-explanatory simulators at the level of physics and mathematics rather than code. Most qualitative reasoning systems infer as much as possible from limited information, such as inferring that a particular flow rate must always be positive. It would be interesting to see how well domain-specific and example-specific hints could replace the functionality provided by inference in earlier compilers.

At this point, we believe self-explanatory simulators are ready for applications. We believe that the major remaining hurdles are building domain theories plus software engineering. The only way to prove this is to attempt some applications. Consequently, we are building a *virtual laboratory* for engineering thermodynamics, containing the kinds of components used in building power plants, refrigerators, and

heat pumps, using a domain theory developed in collaboration with an expert in thermodynamics. We are also building a shell to support the construction of training simulators, such as a self-explanatory simulator for a shipboard propulsion plant, to finally fulfill one of the early goals of qualitative physics [19].

## 6. Acknowledgements

This research was supported by grants from NASA Langley Research Center and from the Office of Naval Research. We thank Franz Amador for supplying us with a sample PIKA domain theory.

## 7. Bibliography

- 1 Forbus, K. and Falkenhainer, B. Self-explanatory simulations: An integration of qualitative and quantitative knowledge, *Proceedings of AAAI-90*.
- 2 Forbus, K. and Falkenhainer, B. Self-Explanatory Simulations: Scaling up to large models, *Proceedings of AAAI-92*.
- 3 Iwasaki, Y. & Low, C. Model generation and simulation of device behavior with continuous and discrete changes. *Intelligent Systems Engineering*, 1(2), 1993.
- 4 Amador, F., Finkelstein, A. and Weld, D. Real-time self-explanatory simulation. *Proceedings of AAAI-93*.
- 5 Forbus, K. Towards Tutor Compilers: Self-explanatory simulations as an enabling technology, *Proceedings of the Third International Conference on the Learning Sciences*, August, 1991.
- 6 Neville, D., Notkin, D., Salesin, D., Salisbury, M., Sherman, J., Sun, Y., Weld, D. and Winkenbach, G. Electronic 'How Things Work' Articles: A Preliminary Report. *IEEE Transactions on Knowledge and Data Engineering*, August 1993.
- 7 Gautier, P. and Gruber, T. Generating explanations of device behavior using compositional modeling and causal ordering. *Proceedings of AAAI-93*.
- 8 Forbus, K. Self-Explanatory Simulators: Making computers partners in the modeling process. In Carrete, N. P. & Singh, M.G. (Eds.), *Qualitative Reasoning and Decision Technologies*, CIMNE, Barcelona, Spain, 1993.
- 9 Sgouros, N. Integrating qualitative and numerical models in binary distillation column design, *Proceedings of the 1992 AAAI Fall Symposium on Design of Physical Systems*, October, 1992.
- 10 Forbus, K. Qualitative Process theory. *Artificial Intelligence*, 24, 1984
- 11 Forbus, K. and de Kleer, J. *Building Problem Solvers*, MIT Press, 1993.
- 12 McAllester, D. An outlook on truth maintenance. MIT AI Lab memo AIM-551, 1980.
- 13 DeCoste, D. and Collins, J. CATMS: An ATMS which avoids label explosions. *Proceedings of AAAI91*.
- 14 Forbus, K. Pushing the edge of the (QP) envelope. In *Recent Progress in Qualitative Physics*, Faltings, B. and Struss, P. (Eds.), MIT Press, 1992.
- 15 Woods, E. The Hybrid Phenomena theory. In *Proceedings of IJCAI-91*, Sydney, Australia.
- 16 Roberts, B. and Forbus, K. The STEAMER mathematical simulation. BBN Technical Report No. 4625, 1981.
- 17 Forbus, K. and Gentner, D. Causal reasoning about quantities. *Proceedings of the Eighth annual conference of the Cognitive Science Society*, Amherst, Mass., August, 1986
- 18 Skorstad, G. Finding stable causal interpretations of equations. In Faltings, B. and Struss, P. (Eds.), *Recent advances in qualitative physics*, MIT Press, 1992.
- 19 Hollan, J., Hutchins, E., & Weitzman, L. STEAMER: An interactive inspectable simulation-based training system. *AI Magazine*, 5(2), 15-27.

# Using qualitative physics to build articulate software for thermodynamics education

**Kenneth D. Forbus**

Qualitative Reasoning Group  
The Institute for the Learning Sciences  
Northwestern University  
1890 Maple Avenue, Evanston, IL, 60201 USA  
forbus@ils.nwu.edu

**Peter B. Whalley**

Department of Engineering Science  
Oxford University  
Parks Road, Oxford, OX13PJ, UK  
whalley@vax.ox.ac.uk

## Abstract

One of the original motivations for research in qualitative physics was the development of intelligent tutoring systems and learning environments for physical domains and complex systems. This paper demonstrates how a synergistic combination of qualitative physics and other AI techniques can be used to create an intelligent learning environment for students learning to analyze and design *thermodynamic cycles*. Pedagogically this problem is important because thermodynamic cycles express the key properties of systems which interconvert work and heat, such as power plants, propulsion systems, refrigerators, and heat pumps, and the study of thermodynamic cycles occupies a major portion of an engineering student's training in thermodynamics. This paper describes CyclePad, a fully implemented learning environment which captures a substantial fraction of a thermodynamics textbook's knowledge and is designed to scaffold students who are learning the principles of such cycles. We analyze the combination of ideas that made CyclePad possible, comment on some lessons learned about the utility of various techniques, and describe our plans for classroom experimentation.

## 1. Introduction

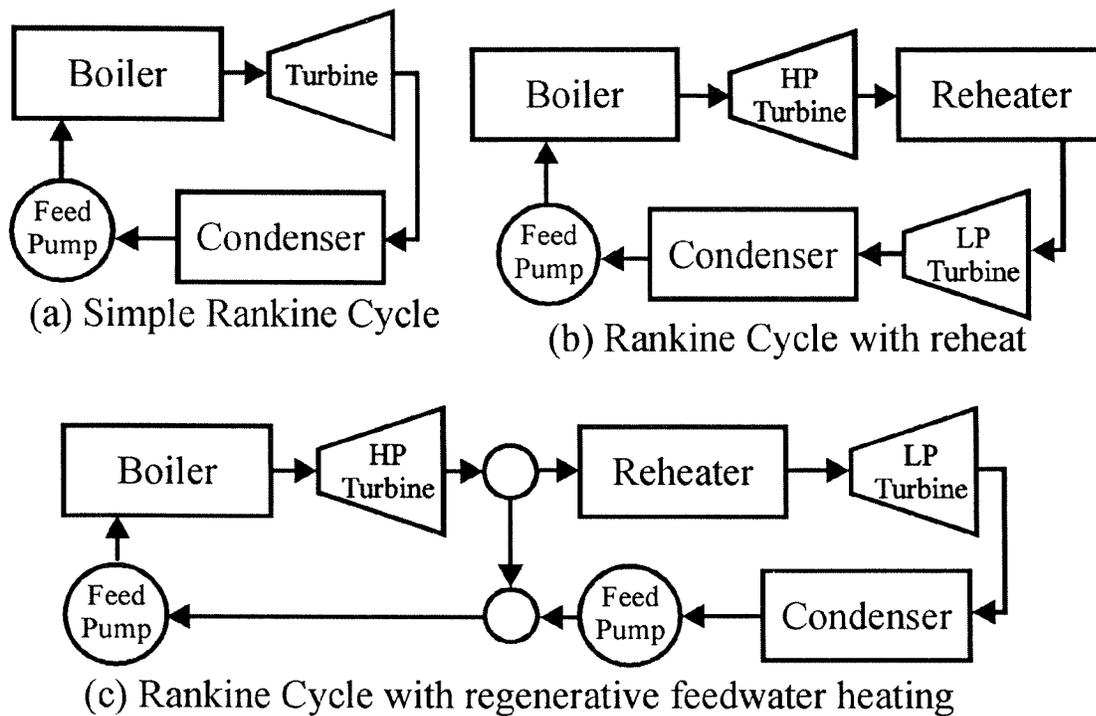
One of the central motivations for research into qualitative physics has been its potential for the construction of intelligent tutoring systems and learning environments. By providing computational accounts of human reasoning about the physical world, ranging from what the person on the street knows to the extensive expertise of scientists and engineers, qualitative physics should provide representation languages and reasoning techniques that can be applied to helping people make the transition from novice to expert reasoning about physical systems. Indeed, some of the earliest work in the field was directly aimed at instructional problems (e.g., [1,2]). Over the last decade there have been several important efforts aimed at using qualitative physics to help teach diagnosis, troubleshooting, and operation of complex physical systems (e.g., [3,4,5,6]), but little effort has been focused on using qualitative physics in classroom settings, to help undergraduates learn principles of a domain (a rare exception is [7]).

In this paper we describe a system, called CyclePad, that has been built to help engineering undergraduates appreciate and therefore learn important principles of thermodynamics. CyclePad provides a conceptual CAD environment where students can design and analyze power plants, refrigerators, and other thermodynamic cycles. It relies on a synergistic combination of existing AI techniques: compositional modeling to represent and reason with modeling assumptions, qualitative representations to express the intuitive knowledge of physics needed to detect impossible designs, truth-maintenance to provide the basis for explanations, and constraint reasoning and propagation to provide efficient mathematical reasoning. It incorporates a substantial fraction of the knowledge in a typical engineering thermodynamics textbook [8], and has been tested on over two dozen examples of problems involving steady-state, steady flow systems where numerical answers or single-parameter sensitivity analyses are required.

Section 2 describes the pedagogical problems that motivated the design of CyclePad, including a brief overview of what thermodynamic cycles are and how they work. Section 3 demonstrates CyclePad's operation from a user's perspective. How CyclePad works is the subject of Section 4, with Section 5 summarizing the lessons we have learned so far in building the system. Section 6 outlines our plans for future work.

## 2. The task: Teaching the design of thermodynamic cycles

A thermodynamic cycle is a system within which a working fluid (or fluids) undergoes a series of transformations in order to process energy. Every power plant and every engine is a thermodynamic cycle. Refrigerators and heat pumps are also examples of thermodynamic cycles. Thermodynamic cycles play much the same role for engineering thermodynamics as electronic circuits do for electrical engineering: A small library of parts (in this case, compressors, turbines, pumps, heat exchangers, and so forth) are combined into networks, thus potentially generating an unlimited set of designs for any given problem. (Practically, cycles range



**Figure 1: Sequence of conceptual designs for a power plant**

from four components, in the simplest cases, to networks consisting of dozens of components.) One source of the complexity of cycle analysis stems from the complex nature of liquids and gases: Subtle interactions between their properties must be harnessed in order to improve designs. Cycle analysis answers questions such as the overall efficiency of a system, how much heat or work is consumed or produced, and what operating parameters (e.g., temperatures and pressures) are required of its components. An important activity in designing cycles (or indeed in many engineering design problems) is performing sensitivity analyses, to understand how choices for properties of the components and operating points of a cycle affect its global properties.

To illustrate, consider the sequence of power plant designs in Figure 1. Figure 1(a) shows a simple Rankine cycle, which pumps a working fluid (as liquid water) into a boiler to produce steam. In the turbine the high-pressure steam expands, thus performing work. Heat is extracted from the steam in the condenser so that the working fluid is again water. Finally, this water is pumped into the boiler (which requires the pump to absorb work) thus beginning the whole cycle again. These processes happen continuously in steady flow, in every part of the system. As it happens, this cycle is not very efficient. Raising the temperature of the boiler increases efficiency, but due to material properties of the components, there are upper bounds on operating

pressures and temperatures. Figure 1(b) shows a more efficient design, which uses a second turbine to extract more energy from the steam. The purpose of the reheater is to ensure that the steam does not become “wet”, i.e., to begin to condense, because water droplets moving at high speed may damage a turbine. The extra energy required to reheat the steam is more than balanced by the additional work gained from the second turbine. One can do even better, however. Figure 1(c) shows a *regenerative feedwater cycle* where some of the steam from the outlet of the high-pressure turbine is routed back to the water feeding the boiler. The boiler then is adding heat to water that is starting at a higher temperature, which increases efficiency.

The analysis and design of thermodynamic cycles is the major task which drives engineering thermodynamics, aside from applications to chemistry. In thermodynamics education for engineers, cycle analysis and design generally appears towards the end of their first semester, or is even delayed to a second course, since understanding cycles requires a broad and deep understanding of the fundamentals of thermodynamics. Even the most introductory engineering thermodynamics textbooks tend to devote several chapters to cycle analysis, and in more advanced books the fraction devoted to cycles rises sharply. Indeed, some textbooks focus exclusively on cycle analysis (e.g., [9]). Aside from their intrinsic interest, the conceptual design of

thermodynamic cycles provides a highly motivating context for students to learn fundamental principles more deeply than they would otherwise.

```

Q: Why is Nu_thermal(CYCLE) = 46%?
A: Nu_thermal(CYCLE) = 46%
   was found by using
   Nu_thermal(CYCLE) =
     net_work(CYCLE)/Q_in(CYCLE)
   on
     Q_in(CYCLE) = 4069.555 kW
     net_work(CYCLE) = 1872.219 kW

Q: Why is net_work(CYCLE) = 1872.219 kW?
A: net_work(CYCLE) = 1872.219 kW
   was found by using
   net_work(CYCLE) =
     work_in(CYCLE)+work_out(CYCLE)
   on
     work_in(CYCLE) = -15.68446 kW
     work_out(CYCLE) = 1887.903 kW

Q: Why is work_out(CYCLE) = 1887.903 kW?
A: work_out(CYCLE) = 1887.903 kW
   was found by using
   work_out(CYCLE) =
     work(TURBINE2)+work(TURBINE1)
   on
     work(TURBINE2) = 1372.131 kW
     work(TURBINE1) = 515.7723 kW

Q: Why does work_out(CYCLE) =
   work(TURBINE2)+work(TURBINE1) hold?
A: work_out(CYCLE) =
   work(TURBINE2)+work(TURBINE1)
   because we assumed:
   CYCLE is a thermodynamic cycle
   the work-flows-out of CYCLE consist
   of {work(TURBINE2),work(TURBINE1)}

Q: How has TURBINE1 works isentropically
   been used?
A: TURBINE1 works isentropically
   was used to derive
   TURBINE1 works adiabatically
   T(S3) = Tout_i(TURBINE1)
   s(S2) = s(S3)

```

**Figure 2: A CyclePad hypertext dialog**

A variety of problems arise when teaching students how to design and analyze thermodynamic cycles:<sup>1</sup> (1) Students tend to get bogged down in the mechanics of solving equations and carrying out routine calculations. This leads them to avoid exploring multiple design alternatives and to avoid carrying out trade-off studies (e.g., seeing how efficiency varies as a function of turbine efficiency versus how it varies as a function of

boiler outlet temperature). Yet without making such comparative studies, many opportunities for learning are lost. (2) Students often have trouble thinking about what modeling assumptions they need to make, such as assuming that a heater operates isobarically, leading them to get stuck when analyzing a design. (3) Students typically don't challenge their choices of parameters to see if their design is physically possible (e.g., that their design does not require a pump that produces rather than consumes work).

CyclePad was designed specifically to help students learn engineering thermodynamics by providing an intelligent learning environment that handles routine calculations, facilitates sensitivity analyses, helps students keep track of modeling assumptions, and detects physically impossible designs.

### 3. Overview of CyclePad

CyclePad can be viewed as a CAD system for the conceptual design of thermodynamic cycles, although it provides substantially more explanation capabilities than existing CAD software. CyclePad performs steady state analyses of steady-flow thermodynamic cycles. The restriction to steady-state is standard for this kind of analysis, since issues of how to start up and shut down the plant, or how easy it will be to monitor, maintain, or troubleshoot are issues of concern only after the basic design has been shown to be sound with respect to the goals for it (e.g., amount of work produced, efficiency, etc.). The restriction to steady-flow systems means that CyclePad cannot currently be used to analyze internal combustion engines, such as Otto or Diesel cycles. Although we plan to extend CyclePad to analyze such systems, steady flow cycles constitute the majority of the cycle-related material taught to engineering students. (For example, in [8] four out of five chapters on cycles concern steady flow cycles, in [9] it is 9 out of 10 chapters, and [10] focuses only on steady-flow systems.)

When a user starts up CyclePad, they find a menu of component types (e.g., turbine, compressor, pump, heater, cooler, heat exchanger, throttle, splitter, mixer) that can be used in their design. Components are connected together by *stuffs*, which represent the properties of the working fluid at that point in the system. (Stuffs serve the same role as nodes in electronic circuits.) The interface helps the user put together a design by highlighting what parts remain unconnected and providing simple critiques of the structure. Once the structural description of the cycle is finished (e.g., there are no dangling connections or stuffs), CyclePad allows the user to enter an analysis mode, where the particular properties of the system, such as the choice of working

<sup>1</sup> These observations are based on the experience of the second author, who teaches engineering thermodynamics to undergraduates.

fluid, the values of specific numerical parameters, and modeling assumptions can be entered.

CyclePad accepts information incrementally, deriving from each user assumption as many consequences as it can. At any point questions can be asked, by clicking on a displayed item to obtain the set of questions (or commands) that make sense for it. In addition to numerical parameters and structural information, all modeling assumptions made about a component are displayed with it, and clicking on a component shows the modeling assumptions that can legitimately be made about that component, given what is known about the system so far. The questions and answers are displayed in English, and include links back into the explanation system, thus providing an incrementally generated hypertext. Figure 2 illustrates.

In addition to numerical assumptions, selecting a component provides commands for making or retracting modeling assumptions concerning that component. For example, clicking on a new turbine yields a menu of commands which offers the options of assuming the turbine is adiabatic or isentropic. Such modeling assumptions can introduce new constraints which may help carry an analysis further and new parameters (e.g., the efficiency of the turbine) that must be set.

When CyclePad uncovers a contradiction, it changes the interface to provide tools to resolve the problem by presenting the source of the contradiction (e.g., an impossible fact becoming believed, or conflicting values for a numerical parameter) and the set of assumptions underlying that contradiction. The hypertext dialog facilities can be used with this display to figure out which assumption(s) are dubious and change them accordingly.

We have tested CyclePad on over two dozen examples to date, ranging from simple ideal gas problems to the analysis of a combined gas turbine/steam Rankine cycle system. We believe that the current version of CyclePad can solve all of the problems in [8] concerning steady-state analyses of steady-flow cycles that require numerical answers or sensitivity analyses involving a single parameter. (We are continuing to test it on new examples, drawn from other textbooks as well.) CyclePad is very efficient. The combined gas turbine/steam Rankine cycle is the most complex system in [8], consisting of ten components. Good students take between 20 minutes and one hour to solve this problem. CyclePad does somewhat more work in analyzing this problem than a good student would, instantiating 219 equations involving 362 parameters, whereas a solution can be found using only 52 equations. However, CyclePad is still faster, taking just over two minutes on a workstation, versus just over ten minutes on a PowerBook 165c. We believe that the combination of the speed at which CyclePad carries out the routine

calculations, its explanation facilities, and its consistency-checking facilities, will make it a valuable tool for students learning thermodynamics.

#### 4. How CyclePad works

The overall structure of CyclePad was inspired in part by EL [11], an experimental system for DC and AC analysis of analog electronic circuits. EL was one of the first systems to use constraint propagation and dependency networks to organize its reasoning, and introduced the idea of dependency-directed backtracking. In this section we see how CyclePad exploits the advances made by the field since EL, by examining each of the AI ideas that contributes to CyclePad's operation, and the reasons for these particular design choices.

##### 4.1 The role of compositional modeling

Compositional modeling [12, 13, 14] provides formal representation and reasoning techniques for formulating and reasoning about models. Knowledge about a domain is organized as collections of *model fragments*, organized by modeling assumptions and the ontology of the domain. Formulating a model for a specific problem consists of instantiating fragments from the domain theory, taking into account the kinds of tasks the model is to be used for.

As noted previously, steady-state analyses are required for the conceptual design of thermodynamic cycles. By restricting ourselves to steady-flow systems, it is also the case that the process structure (i.e., the collection of physical processes acting in each component) is fixed for all time. These restrictions allow us to organize the domain theory around the components which comprise a cycle and the properties of the working fluid at particular locations (i.e., the connections between components).

The modeling language used in CyclePad is similar to other implementations of compositional modeling. For example, Figure 3 shows part of CyclePad's model of a heater. CyclePad's knowledge base consists of 29 conceptual entities, 5 physical processes, 9 assumption classes, 98 equations, 40 pattern-directed rules and 41 background facts about thermodynamics.

Modeling assumptions are organized into *assumption classes* [15, 13]. Assumption classes are always associated with particular classes of components. The relevance of one assumption class can depend on the particular choices made for another assumption class. For example, it only makes sense to consider whether a compressor is isentropic if it is already known (or assumed) to be adiabatic.

```

(defEntity (Abstract-hx ?self ?in
           ?out)
  (thermodynamic-stuff ?in)
  (thermodynamic-stuff ?out)
  (total-fluid-flow ?in ?out)
  (== (mass-flow ?in)
      (mass-flow ?out))
  (parameter (mass-flow ?self))
  (parameter (Q ?self))
  (parameter (spec-Q ?self))
  (heat-source (heat-source ?self))
  ((parts :cycle) has-member ?self)
  (?self part-names (in out))
  (?self IN ?in) (?in IN-OF ?self)
  ?self out ?out) (?out out-of
?self))

(defAssumptionClass
  ((abstract-Hx ?hx ?in ?out))
  (isobaric ?hx)
  (:not (isobaric ?hx)))

(defEntity (Heater ?self ?in ?out)
  (abstract-Hx ?self ?in ?out)
  (?self instance-of heater)
  (heat-flow (heat-source ?self)
             (heat-source ?self)
             ?in ?out)
  ((heat-flows-in :cycle)
   has-member (Q ?self))
  (> (Q ?self) 0.0))

(defEquation Hx-law
  ((Abstract-Hx ?hx ?in ?out))
  (:= (spec-h ?out)
      (+ (spec-h ?in) (spec-Q
?hx))))

(defEquation spec-Q-definition
  ((Abstract-Hx ?hx ?in ?out))
  (:= (spec-Q ?hx)
      (/ (Q ?hx) (mass-flow ?hx))))

```

Figure 3: A sample of CyclePad's knowledge base

#### 4.2 The role of constraint reasoning and propagation

A design is not finished until numerical values have been chosen for its parameters. This is one reason why the overwhelming majority of thermodynamics textbook problems require numerical answers.<sup>2</sup> This fact, plus the relative simplicity of the equations involved, has meant that constraint propagation has sufficed for CyclePad.

<sup>2</sup>In a typical textbook we surveyed, 90% of the exercises required numerical answers.

In compiling CyclePad's knowledge base, equations are automatically converted into antecedent constraint rules that propose values for the  $n$ th variable in an equation whenever the other  $n-1$  variables are known. Redundant equations are introduced when needed to overcome simultaneities. This automatic translation simplifies development. Equations in their original form are still represented in the knowledge base, however, and are used in two ways. First, they are part of the dependency structure for any results calculated via constraint propagation, for explanatory accuracy. Second, they can be inspected via the query system, so that students can find out what equations mention a specific parameter, and what equations might be used to calculate a desired value.

Property tables comprise a critical source of information for CyclePad. Property tables are woven into the constraint propagator via pattern-directed rules, operating under the same protocol as the rules compiled for equations. Due to the inherent loss of accuracy in interpolation, it is important, unlike equations, to *avoid* using tables in every logically possible fashion. Given a superheated vapour, for instance, knowing the pressure and temperature suffice to determine everything else (e.g., the specific enthalpy, specific entropy, etc.). If one redundantly computes from, say the specific enthalpy and specific entropy what the pressure and temperature will be, it is very likely that the newly estimated values will trigger a contradiction, given the accumulated inaccuracies in the interpolation process. Consequently, an important design choice in implementing tables is selecting which directions of access are likely to prove most productive for the kinds of analyses being made.

#### 4.3 The role of qualitative physics

In CyclePad qualitative physics provides the medium for representing constraints on what is physically possible. Occurrences of physical processes inside components are explicitly represented. Each process occurrence includes ordinal constraints that are tested against numerical values by CyclePad's constraint propagation mechanism. Figure 4 shows a sample of what CyclePad knows about physical processes.

```

(defProcessEpisode (fluid-flow ?in ?out)
  (same-substance ?in ?out))

(defProcessEpisode (total-fluid-flow
  ?in ?out)
  (fluid-flow ?in ?out)
  (== (mass-flow ?in) (mass-flow ?out)))

(defProcessEpisode
  (heat-flow ?src-start ?src-end
    ?dst-start ?dst-end)
  (> (T ?src-start) (T ?dst-start))
  (:not (< (T ?src-start) (T ?dst-end)))
  (:not (> (T ?dst-end) (T ?src-end))))

(defProcessEpisode (compression
  ?in ?out ?worker)
  (> (P ?out) (P ?in))
  (< (spec-shaft-work ?worker) 0))

(defProcessEpisode (expansion
  ?in ?out ?receiver)
  (< (P ?out) (P ?in))
  (> (spec-shaft-work ?receiver) 0))

```

Figure 4: Physical process knowledge in CyclePad

#### 4.4 The role of truth maintenance

We used an LTMS [16] in CyclePad because it offered the best tradeoff between inferential power and economy. (In fact, CyclePad's inference engine is the LTRE system from [17]) We ruled out a JTMS because Horn clauses are too clumsy for many of CyclePad's inferential needs, including biconditionals (used in definitional consequences of modeling assumptions, e.g., a compressor is operating isentropically exactly when its isentropic efficiency is 1.0) and TAXONOMY constraints [18] (used in implementing assumption classes). The ability of an ATMS to provide rapid switching between very different contexts was not required: While frequent additions and retractions of assumptions are made in carrying out an analysis, typically these changes are a small fraction of the working set of assumptions in force.

A critical role for the LTMS dependency network is as an input for explanation generation. Explanations in CyclePad are in terms of *structured explanations*, an abstract layer between the reasoning system and the interface that casts the consequences of the inference system in terms relevant to the user. This includes summarization (e.g., [19]), as in removing any reference to implementation-dependent information such as the constraint propagation mechanism from an argument. It also includes making explicit implicit dependencies, such as the variables whose values must be known before the constraint rule implementing a particular equation will fire when explaining what assumptions might lead to more progress.

## 5. Lessons learned from developing CyclePad

CyclePad represents one of the first attempts to apply ideas developed by the qualitative physics community to a real application. While CyclePad has not yet been fielded, we believe that we have already learned several generally useful lessons in building it.

### 5.1 Compositional modeling scales up

Previous uses of compositional modeling have either focused on large but purely qualitative domain theories, or small quantitative theories. CyclePad demonstrates that the ideas of compositional modeling can be used to organize a substantial body of quantitative and qualitative knowledge so that it can be used effectively.

Automatic model formulation, which typically has been the focus of previous compositional modeling work, is less relevant for this application. Nevertheless, the mechanisms of assumption classes and logical constraints between modeling assumptions provide a valuable service in helping the user organize an analysis. In fact, one of the skills being taught in using CyclePad is model formulation. A boiler, for instance, is typically approximated as a heater for the purposes of cycle analysis. A flash chamber is modeled as a splitter whose working fluid is saturated and with particular assumptions about the dryness of the outlets. A multi-stage turbine is modeled as a sequence of turbines and splitters. CyclePad helps users analyze models, so they can figure out if their choice of idealization makes sense, but currently CyclePad does not provide direct assistance with formulating an idealized model from an informal specification.

### 5.2 Regarding constraint reasoning

In this task numerical constraint propagation suffices. There are however natural extensions of CyclePad's analytic abilities for which algebraic manipulation would be useful. For instance, some insights about how a cycle works are best captured via equations.<sup>3</sup> We plan to extend CyclePad to derive such equations on demand. Our experience with the constraint rules compiler in CyclePad, and other work on thermodynamics problem solving [20], suggests that relatively simple algebraic capabilities will suffice for this extension.

We draw two additional conclusions regarding constraint manipulation. First, the commercial world has developed many powerful symbolic algebra packages, such as Mathematica, Maple, and Macsyma, which in some cases are excellent off-the-shelf solutions to

<sup>3</sup> For example, figuring out that for a gas turbine cycle the maximum specific work output is achieved when the pressure ratio is the square root of its maximum possible value [8].

particular problems. However, we suspect that many educational applications will be like CyclePad: Simple algebraic facilities are all that is required, and thus the complexity (and expense) of integrating commercial symbolic algebra packages can be avoided. Second, we found that special-purpose constraint languages (e.g., [21]) were too restrictive for our purposes. Given the need to reason about modeling assumptions and the need to integrate information from property tables, it was much easier to implement a simple constraint propagator inside a pattern-directed inference system than it was to interface a special-purpose constraint manipulator. Aside from applications where scaling up to extremely large system descriptions (e.g., VLSI CAD) is a key requirement, it is hard to see any situation where using such languages makes sense.

### 5.3 Regarding qualitative physics

The combination of steady-state analysis, the restriction to steady-flow systems, and the use of idealized components dramatically simplified the representation of physical processes, since the occurrence of particular physical processes could simply be stipulated inside a component.

CyclePad's focus on quantitative analysis also means that the major inferential role for qualitative physics is ruling out physically impossible designs. We believe similar simplifications will hold in many other applications, since well-designed artifacts explicitly represent the important physical changes in terms of the kinds of components and connections that comprise a schematic, and many science and engineering educational applications involve quantitative knowledge heavily.

On the other hand, certain extensions to CyclePad's capabilities will require substantially more qualitative representations and reasoning. For instance, CyclePad currently does not try to explain how components work, nor does it provide assistance for understanding the physical rationale underlying design changes. To formalize such arguments will take richer qualitative representations, as well as the ability to reason with property diagrams (e.g. [22]). Fortunately, the automatic instantiation of physical process descriptions from a domain theory is an inexpensive and well-understood operation.

### 5.4 Regarding explanation generation and TMSs

The use of a structured explanation system as an abstraction layer between interface and reasoning system was extremely helpful in developing CyclePad, since it allowed us to optimize each independently. We also found, as suggested by [23], that sophisticated natural

language generation techniques were inappropriate for this task. The ability to automatically generate hypertext in response to a user's questions obviates the need for discourse planning, and the fixed nature of the task means that issues such as selecting the appropriate level of detail in an explanation can be postponed. Hypertext allows users to select how much they want to know about a topic, and since the hypertext is only generated on demand, many navigation problems common in fixed hypertexts are avoided.

ATMS technology [24] has been widely used in qualitative reasoning systems because of its ability to rapidly switch between alternate interpretations. As noted previously, this ability is unnecessary in CyclePad, and we suspect that this will be true for most educational applications.

## 6. Discussion

CyclePad demonstrates that qualitative physics has advanced enough to support new applications of AI to educational problems. Compositional modeling provides representational tools and techniques that can be used to encode a substantial body of knowledge about engineering thermodynamics, with constraint propagation providing analytic capabilities and qualitative representations providing the intuition needed to detect student blunders. Automatically generated hypertext explanations enable the user to explore the consequences of his or her assumptions, and figure out what modeling assumptions are needed to make further progress.

To date, CyclePad has only been tested with graduate student volunteers. We will be testing it with undergraduate engineering students both at Oxford and at Northwestern this academic year. Feedback will be gathered via a combination of electronic mail and interviews, which we will use to further improve the system. Our goal is to have CyclePad continuously available to undergraduates, so that their needs will help guide subsequent development.

Several extensions to CyclePad are in progress. First, we will extend it to handle non-steady flow cycles, such as Otto and Diesel cycles. Second, we will add some algebraic capabilities, so that CyclePad can help students derive algebraic expressions that capture important tradeoffs in specific systems.

We view CyclePad as part of a *virtual laboratory* for exploring thermodynamic cycles. A virtual laboratory is a software environment consisting of a set of parts, corresponding to physical parts or important abstractions in the domains of interest, tools for assembling collections of these parts into designs, and facilities for analyzing and testing designs. By working in this

software environment, students can “build” their designs and try them out without expense or danger. In simpler domains some commercial software exists that can be viewed as virtual laboratories (e.g., Interactive Physics for simple dynamics and Electronics Workbench). A novel contribution of qualitative physics is the ability to generate explanations. For educational applications, explanation generation is vital, to help students see what aspects of a situation are important and to tie what they are observing back to fundamental principles. One of our next steps is to extend CyclePad’s explanation facilities, by adding *coaches* [25, 26, 27] to help students, both to guide them through the analysis process (including the representation of real devices in terms of ideal components) and to suggest improvements to a student’s design.

## 7. Acknowledgments

This work was supported by a grant from the Science and Engineering Research Council in the UK and by grants from the Office of Naval Research and NASA Langley Research Center in the U.S.. We thank Yusuf Pisan and John Everett for many enlightening bug reports and suggestions.

## 8. References

- 1 Forbus, K. & Stevens, A. Using Qualitative Simulation to Generate Explanations. *Proceedings of the Cognitive Science Society*, August 1981.
- 2 Brown, J.S., Burton, R. & de Kleer, J. Pedagogical, natural language, and knowledge engineering techniques in SOPHIE I, II, and III. In Sleeman, D. and Brown, J.S. (Eds.), *Intelligent Tutoring Systems*, Academic Press, 1982.
- 3 White, B. & Frederiksen, J. Causal model progressions as a foundation for intelligent learning environments. *Artificial Intelligence*, **42**, 99-157.
- 4 Massey, L., de Bruin, J. and Roberts, B. A Training System for System Maintenance. In Psotka, J. Massey, L., and Mutter, S. *Intelligent Tutoring Systems: Lessons Learned*. Erlbaum, 1988.
- 5 Govindaraj, T. Qualitative approximation methodology for modeling and simulation of large dynamic systems: Applications to a marine steam power plant. *IEEE transactions on systems, man, and cybernetics*, vol SMC-17, no. 6, November/December 1987.
- 6 Masahiro Inui, et al. Development of a model-based intelligent training system for plant operations. *Proceedings of International Conference on ARCE*, Tokyo, pp 89-94, 1990.
- 7 Roschelle, J. Collaborative Conceptual Change: Jointly acting social and cognitive processes. *Proceedings of CogSci-93*.
- 8 Whalley, P. *Basic Engineering Thermodynamics*, Oxford University Press, 1992.
- 9 Haywood, R. W. *Analysis of Engineering Cycles: Power, Refrigerating and Gas liquefaction Plant*, Pergamon Press, 1985.
- 10 El-Wakil, M. *Powerplant Technology*, McGraw-Hill, 1984.
- 11 Stallman, R.M. and Sussman, G.J. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis, *Artificial Intelligence* **9** (1977), 135--196.
- 12 Falkenhainer, B., and Forbus, K. Setting up large-scale qualitative models. *Proceedings of AAAI-88*, August, 1988.
- 13 Falkenhainer, B. and Forbus, K. Compositional Modeling: Finding the Right Model for the Job. *Artificial Intelligence*, **51**, (1-3), October, 1991.
- 14 Nayak, P. Automated modeling of physical systems. Ph.D. dissertation, Computer Science Department, Stanford University, 1992.
- 15 Addanki, S., Cremonini, R., & Penberthy, J.S., Reasoning about assumptions in graphs of models. *Proceedings of IJCAI-89*, 1989.
- 16 McAllester, D. An outlook on truth maintenance. MIT AI Lab memo AIM-551, 1980.
- 17 Forbus, K. and de Kleer, J. *Building Problem Solvers*, MIT Press, 1993.
- 18 Hayes, P. Naive Physics 1: Ontology for Liquids. In Hobbs, J. and Moore, R. (Eds.) *Formal Theories of the Commonsense World*, Ablex, Norwood, NJ, 1985.
- 19 Gruber, T. & Gautier, P. Machine-generated explanations of engineering models: A compositional modeling approach. *Proceedings of IJCAI-93*.
- 20 Skorstad, G. and Forbus, K. Qualitative and quantitative reasoning about thermodynamics, *Proceedings of the Cognitive Science Society*, August, 1989.
- 21 Steele, G. and Sussman, G.J. CONSTRAINTS: A language for expressing almost-hierarchical descriptions, *Artificial Intelligence*, **14** (1980):1--39.
- 22 Pisan, Y. Visual reasoning about physical properties via graphs, *submitted for publication*, 1994.
- 23 Reiter, E. & Mellish, C. Optimizing the costs and benefits of natural language generation, *Proceedings of IJCAI-93*, 1993.
- 24 de Kleer, J. An assumption-based truth maintenance system. *Artificial Intelligence*, **28**(1986): 127--162.
- 25 Burton, R. & Brown, J.S. An investigation of computer coaching for informal learning activities. In Sleeman, D. and Brown, J.S. (Eds.), *Intelligent Tutoring Systems*, Academic Press, 1982
- 26 Lesgold, A., Eggan, G., Katz, S. and Rao, G. Possibilities for Assessment Using Computer-Based Apprenticeship Environments. In Regian, J. & Shute, V. *Cognitive Approaches to Automated Instruction*. Erlbaum, 1992.
- 27 Schank, R.C., & Nohan, M.Y. Empowering the student: New Perspectives on the Design of Teaching Systems. *The Journal of the Learning Sciences*, **1**(7-35), 1991.

# Integrating Qualitative Simulation for Numerical Data Fusion Methods

Yang Gao and Hugh F. Durrant-Whyte

Department of Engineering Science

University of Oxford

Oxford OX1 3PJ, UK

yang@robots.oxford.ac.uk and hugh@robots.oxford.ac.uk

## Abstract

*Work described in this paper is aimed at developing a monitoring and fault detection system for the process plant. In particular this work concentrates on how to combine qualitative reasoning techniques with conventional numerical sensor fusion techniques. The advantages and disadvantages of using pure numerical or pure qualitative technique alone are first demonstrated. A method of developing a combined semi-quantitative model is then introduced. A number of algorithms are designed to analyse semi-quantitative predictions information. Finally, the semi-quantitative model is used to perform the system monitoring and change detection. The application of the algorithms developed is demonstrated by a number of examples throughout the paper, based on real data.*

## 1 Introduction

Work described in this paper is aimed at developing a monitoring and fault detection system for a process plant. In particular this work concentrates on how to combine qualitative reasoning techniques with conventional numerical sensor fusion techniques.

Qualitative physics provides an account of behaviour in the physical world. The vision of qualitative physics, in conjunction with conventional physics, is to provide a much broader formal account of behaviour, an account rich enough to enable intelligent systems to reason effectively about the real world. Qualitative physics predicts and explains behaviour in qualitative terms. In contrast with qualitative techniques, numerical data fusion methods described physical system in more accurate numerical terms. Most sensor based systems employ a large variety of sensors to obtain information. How the information obtained from different sensing devices is combined to form a description of the system is the sensor fusion problem. A number

of sensor fusion methods have been developed, ranging from simple least-squares fitting algorithms to complex statistical inference methods. These algorithms provide statistical descriptions of system behaviour based on statistical uncertainties involved. However, qualitative reasoning methods provide features which are difficult to capture using pure numerical methods. It functions with incomplete knowledge, a qualitative analysis does not require a detailed quantitative model or complete data on the system which may be difficult to obtain; A qualitative model may represent a class of numerical models, because different numerical models can corresponding to the same abstract qualitative model; this is in contrast to the complete problem specific type of a numerical model. Qualitative analysis provides direct, often causal explanations between different variables.

If the information available about a system is purely qualitative, then the qualitative methods will be used; when precise numerical information is available, then a number of numerical methods can be utilised. However, in many situations, where numerical information is substantial but stochastically imprecise, the predictions made are more confident than using the purely qualitative ones. The qualitative predictions on the other hand provide causal explanations. In such cases, using a combined method may take advantage of both techniques and avoid the weakness of using one alone. The work described in this paper is an example of this case.

In this paper, we focus on general physics of a system. The idea behind integrating the two techniques is also to get some qualitative knowledge of the process and sensor physics and incorporate them into the numerical data fusion processes. The application of the algorithms developed is demonstrated by a number of examples throughout the paper, based on a real system, process plant.

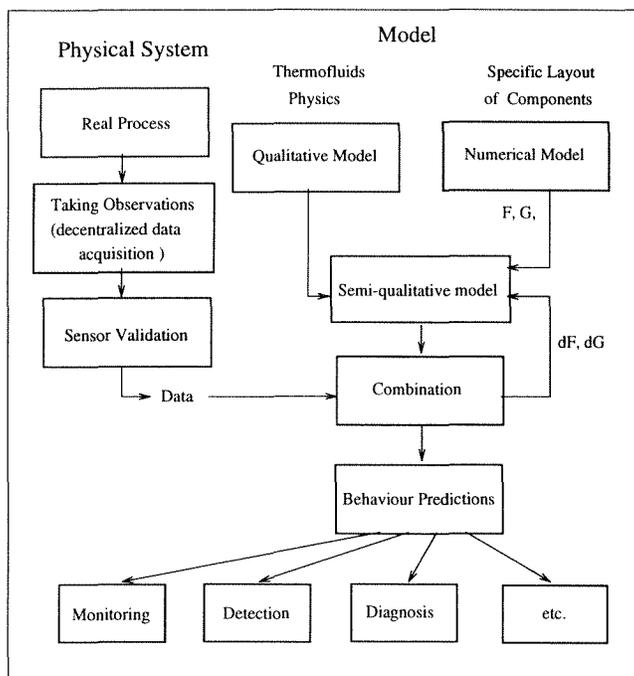


Figure 1: Outline of System

The structure of this system is outlined in Figure 1. We first introduce a method of building a qualitative physical model to simulate the system behaviours. We then develop the system model using a classical state-space description, and explain problems associated with parameter uncertainty. Finally we discuss the common theme that lies between them and the model generated using the combined knowledge. The system state estimate and behaviour prediction generated by this combined model, which we term it as semi-quantitative model, are analysed for the purpose of system monitoring and fault detection.

## 2 System Modelling

The process plant dynamic system is loosely modelled around a conventional nuclear power plant. It incorporates four pumps, three boilers, two heat exchangers and fifty six computer controlled valves. The sensing level provides large quantity of redundant sensor information collected from more than two hundred sensors (thermocouples, flow meters and pressure sensors) which are distributed at appropriate locations in the plant. Details of the process plant system and sensing system can be found in [11].

We consider the simple case for the process plant system. The circuit incorporates one boiler, one pump and one heat-exchanger. For reasons of simplicity,

we assume all the valves along the circuit are in the “open” positions and that there is no energy loss in pipes and pumps. Only temperatures from the input and output of the boiler are to be measured and predicted. We start by switching on all three components. When the temperature of the output of the boiler reaches a certain point, the second pump which is in parallel with the first one will be switched on. What happens next?

To answer this question, we have modelled the system both qualitatively and numerically.

### 2.1 Qualitative Model

The qualitative model of the example discussed above is developed based on the principle of energy conservation in thermodynamics:

$$Q_{in} = C_b \dot{m} (T_{out} - T_{in}) \quad (1)$$

$$Q_{out} = U A \left( \frac{T_{in} + T_{out}}{2} - T_{air} \right) \quad (2)$$

$$Q_{out} = C_f \dot{m} (T_{out} - T_{in}) \quad (3)$$

$$Q_{in} - Q_{out} = Q_{net} \quad (4)$$

$$d\dot{T}_{out}/dt = Q_{net} \quad (5)$$

where  $Q_{in}$  and  $Q_{out}$  are the heating input and heating output,  $T_{in}$  and  $T_{out}$  are the temperature input and output from the boiler,  $T_{air}$  is the plant room temperature,  $\dot{m}$  is the mass flow,  $C_b$  and  $C_f$  are the specific heat for boiler and heat exchanger,  $U$  is the heat transfer coefficient for convection loss, and  $A$  is the heat transfer area. In qualitative modelling, these system parameters are defined using qualitative ranges and landmark values as follows: the heat input is a value  $Q_{in*}$  which lies in the range  $(0, \infty)$ , mass flow has two values when working with one pump or two pumps respectively: denoted as  $mf$  and  $2mf$ . The rest of the parameters are similarly defined. One particular landmark value is setup for the temperature output from the boiler,  $T_{hot}$ . This landmark is set only for the purpose of deciding the time point at which to switch on the second pump, that is when  $T_{out}$  is higher than that of  $T_{hot}$  and increasing, then the second pump is switched on.

The above model provides a qualitative description of the structure of the mechanism and an initial qualitative state without knowing the exact values of the landmarks. The qualitative simulation package (QSIM) is used here to provide automatic generation of the qualitative behaviours of the state variable. The behaviour predictions generated by the above qualitative model are shown in Figure2 and Figure 3. The

Plotting behaviors of parameter  $T_{in}$  for (structure): (CIRCUIT)+(+pp).  
Simulation from 1 complete initialization.  
A total of 4 behaviors.

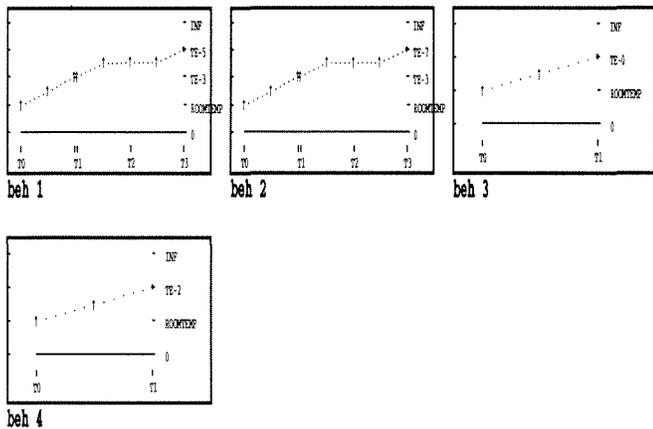


Figure 2: QSIM plot of temperature input of one to two pumps running.

corresponding changes in mass flow are shown in Figure 4.

## 2.2 Discussion

From Figure 2, Figure 3 and Figure 4 we can see that qualitative model predictions include all the possible behaviours that can be deduced from the given qualitative information. Behaviour 1 indicates the case that when the second pump is switched on,  $T_{out}$  drops back to room temperature then increases again; behaviour 2 indicates  $T_{out}$  drops to a temperature point which is between room temperature and  $Thot$ . Behaviour 3 and 4 show the cases which the state transition conditions ( $T_{out} \gg Thot$  and  $T_{out}$  is increasing) are not met, so  $T_{out}$  is either steady at  $Thot$  (beh4), or below  $Thot$  (beh3), the second pump is not switched on at all, see beh3 and beh4 in Figure 4.

Qualitative predictions may also contain spurious behaviours that do not reflect any real situation. For example, behaviour 1 shown above,  $T_{out}$  can not drop back to room temperature when  $T_{in}$  is higher than room temperature. In a real case we can always guarantee that the transition conditions are satisfied, so only behaviour 2 is the correct answer, although beh3 and beh4 are logically correct.

## 2.3 Numerical Modelling Using State-Space Description

The system we consider is assumed to be a first order system following [12], where the physical differen-

Plotting behaviors of parameter  $T_{out}$  for (structure): (CIRCUIT)+(+pp).  
Simulation from 1 complete initialization.  
A total of 4 behaviors.

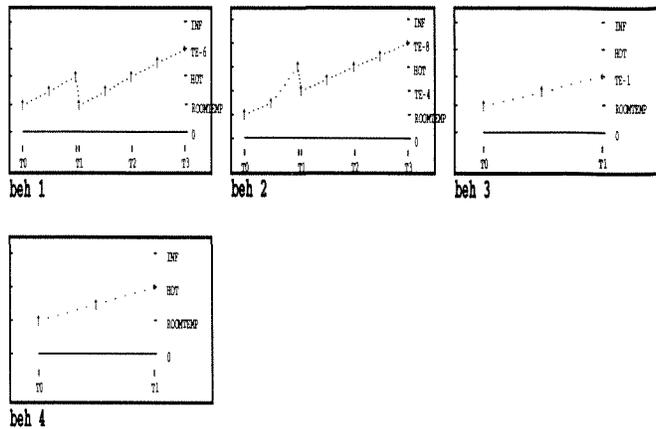


Figure 3: QSIM plot of temperature output of one to two pumps running.

tial equation of steady flow processes can be described as a linear lumped-parameter differential equation. A recursive least-square parameter estimator is used to process the sensor data and find the parameters for the state transition matrix and the input gain matrix.

Using a lumped-parameter system, the discretised state transition equation is of the following form:

$$\mathbf{x}(k+1) = \mathbf{F}\mathbf{x}(k) + \mathbf{G}\mathbf{u}(k) + \mathbf{v}(k), \quad (6)$$

where  $\mathbf{x}$  is the vector of temperature (input and output from the boiler in our example),  $\mathbf{F}$  is the state transition matrix,  $\mathbf{G}$  is the input gain matrix,  $\mathbf{u}$  is the input vector (heat inputs) and  $\mathbf{v}(k) \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}(k))$  is the process noise.

Equation (6) can also be written as the following:

$$\begin{aligned} & \begin{bmatrix} T_{out}(k+1) \\ T_{in}(k+1) \end{bmatrix} \\ &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} T_{out}(k) \\ T_{in}(k) \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} [T_{air}(k)] + v(k) \end{aligned} \quad (7)$$

In the system we are considering, all the temperatures are measurable; the determination of the parameters for this model is from experimental data obtained from the process plant. Real measurement data of the temperature input and output from the boiler is shown in Figure 5(a). The corresponding changes in flow cross the boiler are shown in Figure 5(b).

The system model is obtained using least squares

Plotting behaviors of parameter MASS for (structure): (CIRCUIT) (beh1-4).  
Simulation from 1 complete initialization.  
A total of 4 behaviors.

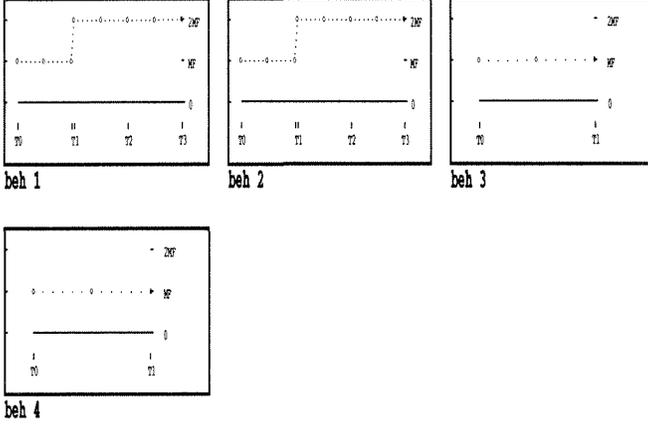


Figure 4: QSIM plot of mass flow changes with one to two pumps running.

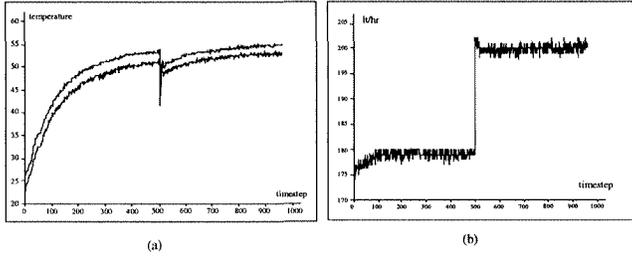


Figure 5: (a) Temperature data with one to two pumps running; (b) Flow changes with one to two pumps running.

regression through the following equations:

$$T_{out}(k+1) = a_{11}T_{out}(k) + a_{12}T_{in}(k) + c_1 \quad (8)$$

$$T_{in}(k+1) = a_{21}T_{out}(k) + a_{22}T_{in}(k) + c_2 \quad (9)$$

For the one-pump case:

$$F_{1p} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} 0.596973 & 0.380452 \\ 0.243474 & 0.75596 \end{bmatrix} \quad (10)$$

and

$$G_{1p} = \begin{bmatrix} -0.493984 \\ 1.11595 \end{bmatrix} \quad (11)$$

Considering the two-pump case:

$$F_{2p} = \begin{bmatrix} 0.532277 & 0.433116 \\ 0.188163 & 0.809515 \end{bmatrix} \quad (12)$$

and

$$G_{2p} = \begin{bmatrix} -0.794684 \\ 1.12695 \end{bmatrix} \quad (13)$$

The process noise covariance is also estimated based on experimental data.

$$Q = \begin{bmatrix} \tilde{T}_{out}\tilde{T}_{out} & \tilde{T}_{in}\tilde{T}_{out} \\ \tilde{T}_{out}\tilde{T}_{in} & \tilde{T}_{in}\tilde{T}_{in} \end{bmatrix} \quad (14)$$

$$= \begin{bmatrix} 0.1 & 0.00064219 \\ 0.00064219 & 0.02 \end{bmatrix}, \quad (15)$$

where

$$\tilde{T}(k) = \mathbf{T}(k) - \hat{T}(k). \quad (16)$$

## 2.4 Parameter Uncertainty

The advantages of using a numerical model is that it finds a exact solutions, providing the model is correct. However, constructing an accurate numerical model for “real-world” domains is usually a difficult, error-prone and time-consuming process. Moreover, due to model inaccuracy, erroneous results may be generated, which may fail to cover all the ranges that need to be considered.

Here we emphasis parameter uncertainty or model error. We define a system error model as follows

$$\mathbf{F} = \bar{\mathbf{F}} + \Delta\mathbf{F} \quad (17)$$

where  $\bar{\mathbf{F}}$  is the “real” (accurate) model of the system,  $\Delta\mathbf{F}$  is the model error and  $\mathbf{F}$  is the model incorporating the error. Using two different models ( $\mathbf{F}$  and  $\bar{\mathbf{F}}$ ), the two predictions based on the same estimates  $\hat{\mathbf{x}}(k|k)$  are:

$$\hat{\mathbf{x}}(k+1|k) = \mathbf{F}\hat{\mathbf{x}}(k|k) + \mathbf{G}\mathbf{u}(k) \quad (18)$$

and

$$\tilde{\mathbf{x}}(k+1|k) = \bar{\mathbf{F}}\hat{\mathbf{x}}(k|k) + \mathbf{G}\mathbf{u}(k). \quad (19)$$

From equation (18) – equation (19) we have:

$$\hat{\mathbf{x}}(k+1|k) - \tilde{\mathbf{x}}(k+1|k) \quad (20)$$

$$= (\mathbf{F} - \bar{\mathbf{F}})\hat{\mathbf{x}}(k|k) \quad (21)$$

$$= \Delta\mathbf{F}\hat{\mathbf{x}}(k|k). \quad (22)$$

We term the model prediction error as

$$\delta\hat{\mathbf{x}}(k+1|k) = \Delta\mathbf{F}\hat{\mathbf{x}}(k|k). \quad (23)$$

Thus

$$\begin{aligned} & (\delta\hat{\mathbf{x}}(k+1|k))^T(\delta\hat{\mathbf{x}}(k+1|k)) \\ &= (\Delta\mathbf{F}\hat{\mathbf{x}}(k|k))^T(\Delta\mathbf{F}\hat{\mathbf{x}}(k|k)) \\ &= (\hat{\mathbf{x}}(k|k))^T(\Delta\mathbf{F})^T(\Delta\mathbf{F})(\hat{\mathbf{x}}(k|k)) \end{aligned} \quad (24)$$

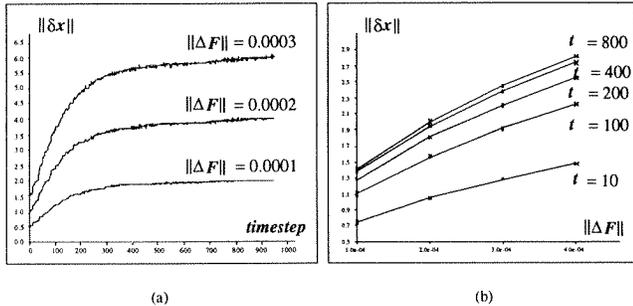


Figure 6: The relation between the model error and prediction error.

Or

$$\sum_j \delta x_j^2 = \sum_j x_j^2 \sum_{ij} \Delta \mathbf{F}_{ij}^2 \quad (25)$$

Here  $\delta x_j$  denotes the element of  $\delta \hat{x}$ ,  $x_j$  is the element of  $\hat{x}$  and  $\Delta \mathbf{F}_{ij}$  is the element of  $\Delta \mathbf{F}$ .

Alternatively, we can rewrite equation (25) as the norm of the vectors:

$$\|\delta x\|^2 = \|\hat{x}\|^2 \|\Delta \mathbf{F}\|^2 \quad (26)$$

Equation (26) shows the relationship between the model error and prediction error. Figure 6(a) shows a plot of  $\|\delta x\|$ s against timestep when given a number of fixed  $\|\Delta \mathbf{F}\|$ s. (Here  $\|\hat{x}\|$ s are the temperature estimates from an experiment using one boiler start-up process.) Figure 6(b) shows how  $\|\delta x\|$  changes against  $\|\Delta \mathbf{F}\|$  when we consider different time points. Using purely numerical modelling techniques, it can be seen that model errors can have substantial effects on system predictions.

### 3 Integration: Semi-quantitative Model Building

Several recent papers have focussed on the relationship between qualitative simulation and numerical simulation. One of the notable approaches is the idea of landmark refinement (Q2)[1]. This aims to employ quantitative information to refine qualitative landmarks; to make better predictions about model trajectories than are possible with pure qualitative simulation, and to prune a tree of qualitative behaviours as it is generated. Dvorak's MIMIC [5, 4] used Q2 for his dynamic system monitoring and diagnosis. However, Q2 suffers from suboptimal and often rather poor quantitative inference. Step size refinement (Q3 [2]) has been developed which is meant to improve Q2's quantitative results by producing progressively

smaller step sizes. Another distinctive method is QP theory based approach which uses qualitative model to predict and explain the behaviours generated by a numerical model ([9] and SIMGEN [10]). Causal ordering was used on the qualitative models to guide explanations. The explanations also incorporated numerical information produced by the numerical simulation module. The limitation of this approach is that it requires a comprehensive domain model and a total environment. A number of other papers considered the abstraction of interval valued information into qualitative simulation. The general problem of measurement interpretation can be split into two cases, figuring out what is happening in a system at a particular time (taking one look) [6, 7, 8] and describing what is happening over a span of time DATMI [3].

#### 3.1 Generation of the Semi-quantitative Model

From our analysis of numerical and qualitative modelling, we can see the following points: using system identification methods to obtain a numerical model is relatively easy given the sensing system, therefore, the numerical state space model formulation is used to describe the system. However models constructed in this way give little physical inside, as the parameters of the model have no direct physical meaning, they are used only as tool to provide a description of the system's overall behaviour. While on the other hand, the qualitative model derived from the fundamental balance equations provides better causal explanation and more intuitive insight into system behaviour. Thus, qualitative knowledge is incorporated as a guide to the numerical prediction process and also to provide an indication of the necessity or otherwise of using more complex data fusion techniques. Therefore, our approach is to develop a semi-quantitative or semi-qualitative model of the system which takes advantage of both numerical and qualitative method.

Using the state transition form described in Equation (6), the state transition model matrix and input model are described in the following form:

$$F = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \quad (27)$$

and

$$G = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \quad (28)$$

Each element of the state transition matrix,  $F$ , or input matrix,  $G$ , is a purely numerical value. The dimension  $n$  is decided by the number of the parameters. The corresponding semi-quantitative models follow the same formulation but each corresponding element of the matrices is an interval value  $[a, b]$  ( $a \ll b$ ) which can be one of the following cases:

- $a = b$ , which is  $[a, a]$  when it is a fully specified value, the interval endpoints are identical.
- $a \neq b$ , where both  $a$  and  $b$  are numerical values, thus  $[a, b]$  is a real range value, where the true value falls into this range.
- $a \neq b$ , where either  $a$  or  $b$  or both are qualitative values, eg.  $(-\infty, b]$ ,  $(0, \infty)$  or  $(-\infty, \infty)$ .

The semi-quantitative model of the system is initially generated by taking measurements during a number of runs of the system under different conditions, including different durations, and different situations, eg. different initial states (temperature values in our application). The original system design parameters can also be a source of information when making decision on the initial range of the parameters.

The principle of finding this initial interval matrix is independent of the application system. For a particular physical system with which we are concerned, we have a sensing network as described earlier. Thus we are able to take measurements from time to time. The measurements made can be used to relate the quantitative model to the qualitative model.

Having a number of experimental runs, we can obtain a group of corresponding numerical models  $F_1, F_2, \dots, F_m$  (all  $n \times n$  matrices) and  $G_1, G_2, \dots, G_m$  (all  $n \times 1$  matrices.) using the method discussed in Section 2.3. The new semi-quantitative state space model  $F_{interval}$  is then generated in the following way. Each element of the matrix is an interval value which takes the minimum and maximum values from the corresponding element of  $F_i$  and  $G_i$ , that is:

$$a_{ij} = \min\{a_{ij_{F_1}}, a_{ij_{F_2}}, \dots, a_{ij_{F_m}}\} \quad (29)$$

and

$$b_{ij} = \max\{b_{ij_{F_1}}, b_{ij_{F_2}}, \dots, b_{ij_{F_m}}\}. \quad (30)$$

Input matrix:

$$c_i = \min\{c_{i_{F_1}}, c_{i_{F_2}}, \dots, c_{i_{F_m}}\} \quad (31)$$

and

$$d_i = \max\{d_{i_{F_1}}, d_{i_{F_2}}, \dots, d_{i_{F_m}}\}. \quad (32)$$

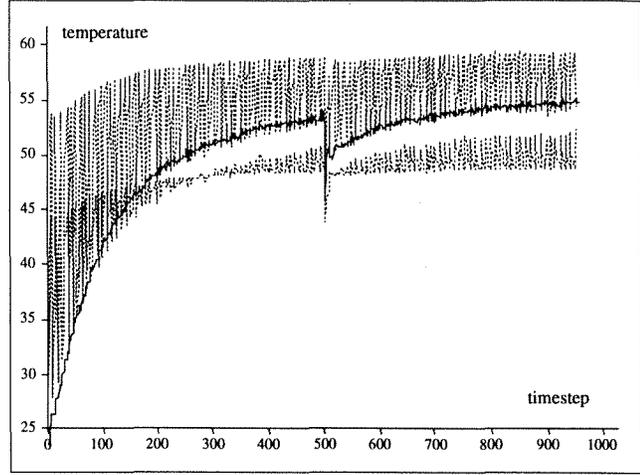


Figure 7: Dotted lines: boiler output temperature prediction ranges; solid line: real measurements. Interpolation time step  $n = 10$ .

We term the range valued matrix an *interval matrix* (following [13]). The semi-quantitative form of the system model and input model are denoted as:

$$[F_{interval}] = \begin{bmatrix} [a_{11}, b_{11}] & [a_{12}, b_{12}] & \dots & [a_{1n}, b_{1n}] \\ [a_{21}, b_{21}] & [a_{22}, b_{22}] & \dots & [a_{2n}, b_{2n}] \\ \vdots & \vdots & \ddots & \vdots \\ [a_{n1}, b_{n1}] & [a_{n2}, b_{n2}] & \dots & [a_{nn}, b_{nn}] \end{bmatrix}, \quad (33)$$

$$[G_{interval}] = \begin{bmatrix} [c_1, d_1] \\ [c_2, d_2] \\ \vdots \\ [c_n, d_n] \end{bmatrix} \quad (34)$$

When two variables in the state vector are not related, the corresponding element in the matrix is zero. When there is at least one element in the original interval matrix which is a purely qualitative value, like  $(0, \infty)$ ,  $(-\infty, \infty)$ , the qualitative simulation process is used to predict the system behaviours initially until a real valued interval is found to replace the qualitative interval.

### 3.2 The Prediction

Having found the semi-quantitative models of the system,  $F_{interval}$  and  $G_{interval}$ , the predictions can be made based on the interval matrices. The interval analysis techniques developed by Moore [13, 14] form the basis for interval calculation. The individual

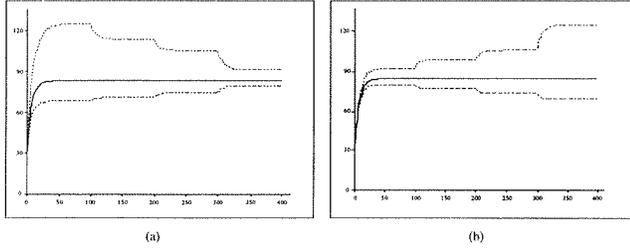


Figure 8: (a) Semi-quantitative model predictions (dotted lines) approach the unique numerical model prediction (solid line) when reducing the range information of the model. (b) The predictions (dotted lines) made by the semi-quantitative model have excessive widths when the elements of the model have excessive range values, they still include the pure numerical prediction (solid line), but approach the qualitative prediction.

arithmetic operations  $\{+, -, *, /\}$  are defined by:

$$\begin{aligned} [a, b] + [c, d] &= \{x + y : x \in [a, b], y \in [c, d]\} \\ &= [a + c, b + d] \end{aligned} \quad (35)$$

$$\begin{aligned} [a, b] - [c, d] &= \{x - y : x \in [a, b], y \in [c, d]\} \\ &= [a - d, b - c] \end{aligned} \quad (36)$$

$$\begin{aligned} [a, b][c, d] &= \{xy : x \in [a, b], y \in [c, d]\} \\ &= [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)], \end{aligned} \quad (37)$$

$$\begin{aligned} [a, b]/[c, d] &= \{x/y : x \in [a, b], y \in [c, d]\} \\ &= [a, b][1/d, 1/c], \text{ for } 0 \notin [c, d] \end{aligned} \quad (38)$$

However, the propagation of interval matrices may result in the excessive widths for the range information. Therefore, for a given system, the prediction made based on this semi-quantitative model will be a subset of the prediction that the pure qualitative model has made and it will include the exact behaviour as predicted by an accurate model. When the range information in the interval matrices reduces (which in general is the case when the model error  $\Delta F \in (0, \inf)$  decreases) the prediction, will reduce its range and approach the unique real solution. This is demonstrated in Figure 8(a). Figure 8(b) shows the reverse case. Here when widening the constraints on the range information ( $\Delta F$  increases), the semi-quantitative model predictions approach the purely qualitative one.

### 3.3 Measurement Interpolation

To avoid problems with divergent ranges, a measurement interpolation method was developed. An algorithm is designed to keep the predictions convergent within certain ranges involving the use of measurements to “refine” state estimation or interpolating new measurements to replace the estimate.

As  $F$  and  $G$  are interval matrices, after one prediction cycle, the prediction vector is an interval vector, shown in the following equation:

$$[\hat{\mathbf{x}}(k+1 | k)] = [\mathbf{F}]\hat{\mathbf{x}}(k | k) + [\mathbf{G}]\mathbf{u}(k) \quad (39)$$

Further cycles may further widen the intervals. (With the square brackets denoting the interval matrices or vectors). To avoid an unbounded increase of the interval width, we take the intersection of the measurements with the state estimate at every  $n$  time steps:

$$\begin{aligned} \text{When } k &= l \cdot n; \\ \text{where } n &= \text{predefined number of time steps,} \\ \text{and } l &= 0, 1, \dots, l \cdot n \leq \text{current time step.} \end{aligned} \quad (40)$$

$$\hat{\mathbf{x}}(k | k)_{new} |_{n=} = \hat{\mathbf{x}}(k | k) \cap \mathbf{z}(k) |_{n} \quad (41)$$

The importance of using the measurements to refine the estimate is that the observation values are always within reasonable small ranges. (That is:  $[obs - \Delta, obs + \Delta]$ , where  $\Delta$  is the measurement precision.) Thus by taking the intersection of both, the interval width of the next step’s prediction can be bound within small ranges.

The inherent principle idea works for the following cases. Assume:

$$\hat{\mathbf{x}}(k | k) \in [l_i, l_j] \quad \text{and} \quad \mathbf{z}(k) \in [l_p, l_q]. \quad (42)$$

when

$$\begin{aligned} \text{either } l_p &< l_j, \text{ and } l_q > l_i; \\ \text{or } l_i &< l_p < l_j \end{aligned}$$

Then the left hand side of equation (41) is non-zero. However when  $l_q < l_i$ , or  $l_p > l_j$ , equation (41) is zero, the above algorithm does not work in a straight-forward way. This is the case that the state estimated is inconsistent with the measurement. This represents one of the following cases:

- The system model needs to be modified so that when the system behaves normally, the estimate should be consistent with the real measurements;

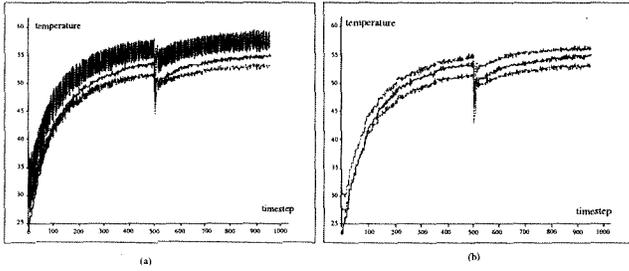


Figure 9: (a) Interpolation time step  $n = 3$ , (b) Interpolation time step  $n = 2$ .

- The conflict shows that the system is in an abnormal state; further analysis techniques will be used;
- The abnormal state model is to be augmented based on both the new measurements and qualitative knowledge about the physical systems.

Here we explain the algorithm used through some examples. We consider the same case discussed in Section 2 that transits the state from one pump to two pumps. Figure 7 shows the predictions made by the semi-quantitative model with every 10 time steps ( $n = 10$ ) of the measurement interpolations. Figure 9(a) shows results with  $n = 3$ . And finally Figure 9(b) shows the refinement time step with  $n = 2$ .

## 4 System Monitoring and Fault Detection

The interval estimates and predictions are used to perform system monitoring and fault detection. As interval data provides us with a different type of information from conventional single valued data, different algorithms are also designed here to deal with interval values.

### 4.1 Monitoring: Qualitative Interpretations

To monitor the physics of the system and to understand it using common sense knowledge, we abstract range information to sign information. This is one way of interpreting the qualitative information from semi-quantitative information. The interpretation criteria are described below.

Suppose interval values change from  $A$  to  $B$ ,  $A \in [l_m, l_n]$ ,  $B \in [l_\alpha, l_\beta]$ . They can be interpreted into

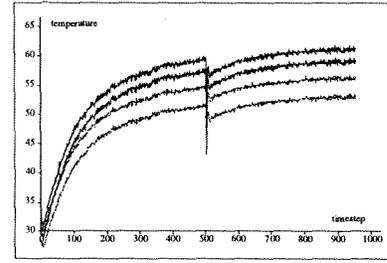


Figure 10: Example of two range valued temperatures; Solid lines:  $[T2_{lo}, T2_{hi}]$ , dotted lines:  $[T1_{lo}, T1_{hi}]$ .

t.step	qual.int	t.step	qual.int	t.step	qual.int
4	std-dec	9	std-inc	14	std-inc
19	std-inc	24	std-inc	29	std-inc
34	std-inc	39	std-inc	44	std-inc
49	std-inc	54	std-inc	59	std-inc
64	std-inc	...	...	489	std-dec
494	std-inc	499	std-dec	504	dec
509	inc	514	std-dec	519	std-inc
524	std-dec	529	std-inc	534	std-mag
539	std-dec	544	std-inc	549	std-inc
554	std-inc	...	...	919	std-dec
924	std-inc	929	std-inc	934	std-dec
939	std-inc	944	std-inc	949	std-inc

Figure 11: Qualitative Interpretation

three different types of qualitative changes defined as follows:

$$\begin{aligned}
 inc & \text{ (increase;)} & ifl_\alpha \geq l_n \\
 dec & \text{ (decrease;)} & ifl_\beta \geq l_n \\
 std & \text{ (steady;)} & otherwise.
 \end{aligned} \tag{43}$$

The level of detail that the interpretation involves depends on how close the two interval values  $A$  and  $B$  are chosen. Figure 10 gives two range valued temperature sequences. Figure 11 shows the qualitative interpretation based on  $B(timestep) - A(timestep) = 5$ . However, the level of detail may be changed over the interpretation of one sequence. Either at some crucial points (eg. state transition) or when there are some significant changes of the qualitative states (eg. from *inc* to *dec*), higher level of details can be introduced.

By continuing this qualitative interpretation of the range information over time, a qualitative understanding of the physical system can be obtained incrementally. Domain-specific knowledge about the state and the transition possibilities can be used to suggest the interpretation which is most likely. When no consis-

tent interpretation exists, faulty hypotheses may be generated.

## 4.2 Fault Detection

The interval estimates and predictions are used to perform system monitoring and fault detection. As interval data provides us with a different type of information from conventional single valued data, different algorithms are also designed here to deal with interval values. Before giving detailed explanation of the algorithms, an assumption is introduced first. That is the systems considered do not have any unknown controlled state changes/transitions. Any state changes are regarded as faults. As a consequence, there will be “abrupt changes” in the sensor readings. At this stage, only temperature readings are considered. Pressure and flow readings will be taken into account in the diagnosis process.

Two methods are designed to analyse the interval sequences and detect abrupt changes.

### 4.2.1 Smoothing method

Assume that we have a sequence of prediction vectors; they represent interval information, denoted as:

$$([\hat{\mathbf{z}}(k)] \mid k = 0, 1, \dots, n)$$

$$\text{where } \hat{\mathbf{z}}(k) \in [\hat{\mathbf{z}}_{lo}(i), \hat{\mathbf{z}}_{hi}(i)], \quad \text{and } i \in (0, n).$$

By choosing a time width, which we term as the window size  $\tau$ , we can analyse the changes of higher bound and lower bound of this prediction sequence in the following way:

$$\frac{\hat{\mathbf{z}}_{hi}(t + \tau) - \hat{\mathbf{z}}_{hi}(t)}{\tau} = \Delta \mathbf{Z}_{hi} \quad (44)$$

$$\frac{\hat{\mathbf{z}}_{lo}(t + \tau) - \hat{\mathbf{z}}_{lo}(t)}{\tau} = \Delta \mathbf{Z}_{lo} \quad (45)$$

As an example, given the two sequence of temperature predictions in Figure 10, by running of the “smoothing” program to  $[T_{1lo}, T_{1hi}]$ , the corresponding  $\Delta \mathbf{Z}_{hi}$  and  $\Delta \mathbf{Z}_{lo}$  obtained are shown in Figure 12. From Figure 12 we can see that: this process has smoothed small changes, but retains the large changes. Therefore, by checking  $\max(\Delta \mathbf{Z}_{hi})$  and  $\max(\Delta \mathbf{Z}_{lo})$ , the abrupt change points or failure points can be detected. This change detection process is done from timestep zero to the current time step. When  $\tau \rightarrow 0$ , then the above equations become:

$$\frac{d\hat{\mathbf{Z}}_{hi}}{dt} = \Delta \mathbf{Z}'_{hi} \quad \text{and} \quad \frac{d\hat{\mathbf{Z}}_{lo}}{dt} = \Delta \mathbf{Z}'_{lo}, \quad (46)$$

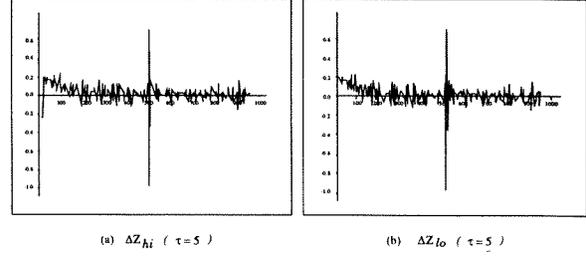


Figure 12: Higher bound and lower bound of the range value analysis.

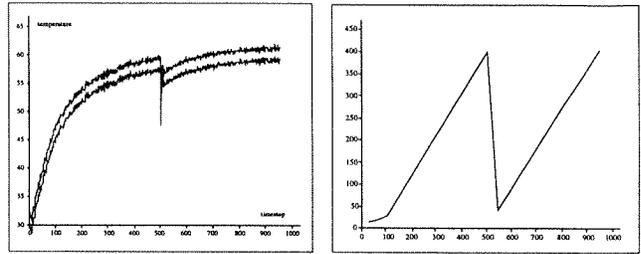


Figure 13: Interval changes detection example 1

which show the changing rate of the range information. The window size  $\tau$  here can be adjusted according to specific application needs.

### 4.2.2 Non-smoothing method

Another way of looking at changes of interval information quantitatively is the non-smoothing method. Rather than considering the higher bound or lower bound of the interval values separately, here each interval value as a whole will be compared with the previous time step value. The algorithm continuously carries out the following process, from the current time step scanning backwards to the first time step:

$$\begin{aligned} \text{If } & \hat{\mathbf{Z}}_{lo}(k) \geq \hat{\mathbf{Z}}_{hi}(k-1) \\ \text{Then } & k-1 \rightarrow k, \text{ and } k-2 \rightarrow k-1 \\ \text{Else } & k \rightarrow k \text{ and } k-2 \rightarrow k-1 \end{aligned} \quad (47)$$

Figure 13 shows an example of the execution of this algorithm on the sequences in Figure 10. The algorithm will pick up any abrupt changes along the time sequence. Figure 14 is another demonstration of the algorithm.

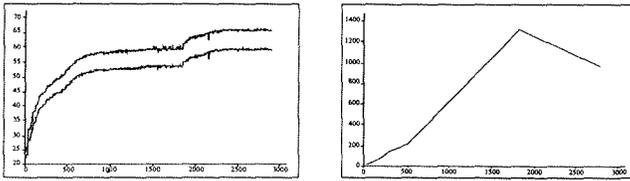


Figure 14: Interval changes detection example 2

## 5 Conclusions and Future Work

This paper describes the work done towards integrating qualitative reasoning into numerical data fusion. Presented here is a methodology for generating the combined model, and analysing the interval information for the purpose of system monitoring and fault detection. Various algorithms developed have been implemented on a real system, process plant. Effort has been made in integrating the results from the engineering community and from the qualitative community. Further work along this line will be carried out, emphasizing in particular the use of qualitative reasoning techniques to explain various phenomena.

Future work also includes using the integrated technique to perform the diagnosis task. Although a semi-quantitative model can be generated, it describes only standard behaviour. Therefore, it is impossible to use it to simulate unforeseen situations. However, considering the process plant system we have been studying, we may have a fixed number of qualitative model based physically on the principle of violation of continuity or energy equations. Then we should be able to deduce, for example, from the successful violation of the energy constraint or mass flow constraint, that there exists a particular type of fault in a particular location.

### Acknowledgements

Here we gratefully acknowledge Prof. B. Kuipers and his students D. Clancy and B. Kay for their providing the QSIM software and support for its use. This work is supported in part by SERC/IED/1199.

### References

[1] D. Berleant and B. Kuipers. Qualitative and quantitative simulation: Bridging the gap. Technical Report AI90-158, U. of Texas Dept. Comp. Sci., 1990.

- [2] D. Berleant and B. Kuipers. Qualitative-numerical simulation with q3. In B. Faltings and P. Struss, editors, *Recent Advances in Qualitative Physics*. The MIT Press, 1992.
- [3] D. DeCoste. Dynamic across-time measurement interpretation. *Artificial Intelligence Journal*, 51(1-3):273-342, 1991.
- [4] D. Dvorak. *Monitoring and Diagnosis of Continuous Dynamic Systems Using Qualitative Simulation*. PhD thesis, University of Texas AI Lab., 1992.
- [5] D. Dvorak and B. Kuipers. Model-based monitoring of dynamic systems. In *Int. Joint Conf. Artificial Intelligence*, pages 1238-1243, 1989.
- [6] K. Forbus. Measurement interpretation in qualitative process theory. In *Int. Joint Conf. Artificial Intelligence*, volume 1, pages 315-320, 1983.
- [7] K. Forbus. Interpreting measurements of physical systems. In *National Conference on Artificial Intelligence*, pages 113-117, 1986.
- [8] K. Forbus. Interpreting observation of physical systems. *IEEE Trans. Systems Man and Cybernetics*, SMC-17(3):350-359, 1987.
- [9] K. Forbus and B. Falkenhainer. Self-explanatory simulations: Integrating qualitative and quantitative knowledge. In B. Faltings and P. Struss, editors, *Recent Advances in Qualitative Physics*, pages 49-66. MIT Press, 1992.
- [10] K. D. Forbus. The qualitative process engine. In D.S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann Publishers, INC, 1990.
- [11] Y. Gao and H. Durrant-Whyte. A transputer-based sensing network for process plant monitoring. In *Third Int. Conf. Transputers and Applications*. IOS Press, 1991.
- [12] K.A.Loparo, M.R.Buchner, and K.S.Vasudeva. Leak detection in an experimental heat exchanger process: A multiple model approach. *IEEE Trans. Automatic Control*, 36(2):167-177, February 1991.
- [13] R. E. Moore. *Interval Analysis*. Prentice-Hall, Inc., 1966.
- [14] R. E. Moore. *Methods and Applications of Interval Analysis: SIAM Studies in Applied Mathematics*. SIAM, 1979.

# Compositional Modeling for Complex Spatial Reasoning Tasks\*

Kyungsook Han and Andrew Gelsey

kshan@cs.rutgers.edu gelsey@cs.rutgers.edu

Department of Computer Science

Rutgers University

New Brunswick, NJ 08903

U. S. A.

## Abstract

Reasoning about a complex physical system generally requires the creation and execution of a model of the system, the creation of which in turn depends on the types of knowledge available for the physical system and their representation. Such a model is normally created by the person studying the system. Despite the considerable time and effort spent, a hand-crafted model is often error-prone. Modifying a hand-crafted model to solve a similar problem about other physical systems is also difficult, and may take more time than building a new model for the systems. We describe a method which uses first principles to automatically create models and simulators for spatially complex motions. This method solves several problems with existing AI modeling work on motion by: (1) explicit handling of vector quantities and frames of reference; (2) simultaneous handling of multiple equations (algebraic or differential, linear or nonlinear); and (3) declarative, algorithm-neutral representation of physics knowledge. The method has been implemented in a working program called ORACLE and tested in the domains of mechanical devices and sailboats. Experimental results show that ORACLE is capable of generating correct models of several different types of physical systems if enough domain knowledge is available.

## Introduction

Spatial reasoning problems are considered in a variety of areas, but different areas have different spatial reasoning tasks. In computer vision, for example, recognition of familiar objects can be the main spatial reasoning task. Our focus in this paper is on modeling physical systems for reasoning about spatially complex motion of the systems. By spatially complex motion we mean the motion of multiple moving objects in arbitrary configurations in three dimension. Motion in three dimension is much more difficult to understand

and deal with than the motion in one or two dimension because modeling such a motion involves reasoning in vector space.

There are several works on modeling motion, but spatially complex motions did not get much attention. Some qualitative physics approaches have been used to model motion. But they focus on developing representations for physical systems and reasoning about the systems within the representations, and their capability of reasoning about motion is limited to simple motions only. Consider, for example, a spring with one end attached to a fixed point and the other end attached to a block. What happens if the block is pulled *and* rotated from its equilibrium position and released, as illustrated in Figure 1a? This spring-block system is different from a linear harmonic oscillator, which is a common textbook example often used in qualitative physics research. It is well known that the harmonic oscillator has one degree of freedom, i.e., displacement of the block from its equilibrium position, and its motion is oscillatory on a straight line. However, predicting the behavior of the spring-block system in Figure 1a is not as simple as the harmonic oscillator. Many qualitative physics approaches which can model the harmonic oscillator (Forbus 1984; Kuipers 1986; Struss 1988; Weld 1988; Williams 1986) cannot handle this spring-block system. The primary reason for this difficulty can be attributed to the fact that qualitative values are not adequate for spatially complex problems and that they lack the ability to reason explicitly about vector quantities and moving frames of reference. Some research in spatial reasoning or commercial mechanics simulators have powerful algorithms to model motion, but they incorporate knowledge of physical phenomena directly into algorithms and difficult to reuse or extend them to solve similar problems about other physical systems.

The work of this paper is motivated by two goals. The first goal is to automate the model formulation and simulation process for complex motion. The second goal is to make the modeling process as general as possible so that common domain theories can be shared and reused instead of being duplicated. As we

\*This research was partially supported by the Advanced Research Projects Agency (ARPA) and National Aeronautics and Space Administration under NASA grant NAG2-645 and by the National Science Foundation through grant CCR-9209793.

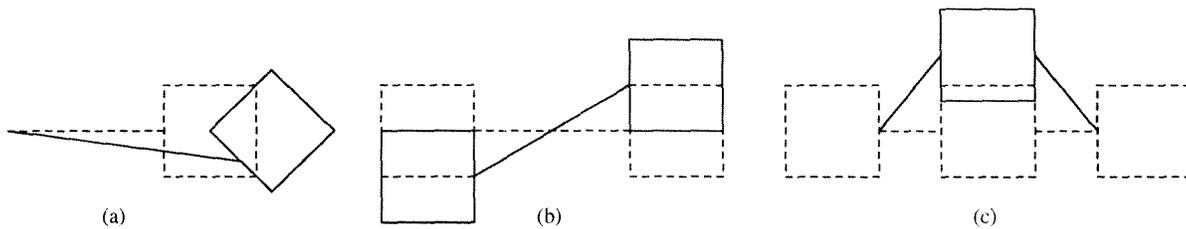


Figure 1: (a) 1 block attached to a spring. The block is pulled and rotated from its equilibrium position and released. (b) 2 blocks connected by 1 spring. Both blocks are pulled in opposite directions and released. (c) 3 blocks connected by 2 springs. The middle block is pulled directly to the side and released.

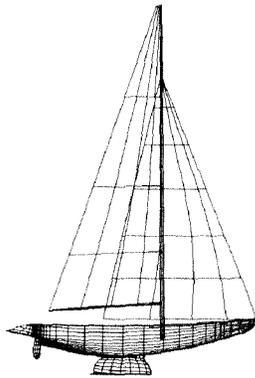


Figure 2: *Stars & Stripes*, winner of the 1987 America's Cup competition.

will show later, we handle explicitly vector quantities and reference frames, and construct a model from a set of model fragments. The model fragments represent the fundamental physics knowledge in a declarative and algorithm-independent way, and are reused in building models of different types of physical systems.

The remainder of this paper discusses a framework for automatically creating and simulating behavioral models of moving physical systems, and illustrates the framework using the single spring-block system in Figure 1a as a running example. We also discuss the epistemological adequacy of the framework for broader class of physical systems, and address related issues such as: Can the modeling system of the single spring-block system be used to predict the behavior of more general spring-block systems such as Figure 1b and Figure 1c, or different types of physical systems such as a sailboat shown in Figure 2? Do we need a separate modeling system for every physical system? Or the same modeling system just with more knowledge will suffice to model them?

## Framework for Model Building and Simulation

In this section we describe the framework of our modeling system called ORACLE, implemented in the mathematical manipulation language Maple (Char *et al.*

1991).

### Ontology and Representation

The principal elements of ORACLE's ontology are entities, phenomena, model fragments, and models, each represented in a frame (Minsky 1975). An *entity* is a physical object which either constitutes a physical system by itself (i.e., primitive object) or is a part of a physical system (i.e., composite object). The properties of an entity are expressed as variables in equations. The block entity, for example, has properties such as position and velocity. An entity is represented in a frame with slots for the properties. Facets allowed in a slot are value, form, range, if\_needed, and if\_added. The value facet is initially set to null but will be assigned a vector, scalar, string, set, or any other expression as it becomes known. The form facet distinguishes the slot type (e.g., scalar or vector) and is consulted when the system creates a new Maple variable name during the problem solving process. For example, if the system is asked to compute the position of a block b1, a set of new variables {b1x(t), b1y(t), b1z(t)} will be created for the position vector and used in equations. The range facet specifies a valid range of the property value if it is known. The if\_needed facet or if\_added facet holds the procedure call, invoked when a slot value is needed or added. The if\_needed procedure of the velocity slot in the example below says that velocity is derivable from position.

```
block=[AKO=rigid_body,
       position(t)=[value=null, form=[x(t),y(t),z(t)]],
       velocity(t)=[value=null, form=[u(t),v(t),w(t)],
                  if_needed=[derive_velocity, position(t)]],
       .... (other slots not shown) .... ]
```

A *phenomenon* is a process which changes one or more properties of an entity in a physical system. Force from a spring, for example, is a phenomenon which can change the position and/or orientation of an entity which is attached to the spring.

A *model fragment* is a characterization of a physical phenomenon by a set of entities, variables, assumptions, and equations. There may be more than one model fragments for a single phenomenon, each with different assumption or approximation. The equations of a model fragment are applicable when the corre-

spending phenomenon occurs. The spring force, for example, exerted on an object attached to end2 of a linear spring with linear damping is represented as follows (syntax slightly modified for readability):

```
Springforce2=[phenomenon='spring force at end2',
  entities=[s=linear_damped_spring],
  variables=[k=s[force_const],
    b=s[damping_coeff],
    e1(t)=s[end1(t)], e2(t)=s[end2(t)],
    l=s[rest_length], f(t)=s[force2(t)],
  equations=[f(t)=-k*(||e2(t)-e1(t)||-l)*
    (e2(t)-e1(t))/||e2(t)-e1(t)||-
    b*diff((||e2(t)-e1(t)||-l)*(e2(t)-e1(t))/
    ||e2(t)-e1(t)||,t)]]
```

It says that  $s$  is a linear damped spring,  $k$  is a force constant of the spring,  $b$  is a damping coefficient,  $e_1(t)$  and  $e_2(t)$  are the position vectors of end1 and end2,  $l$  is the rest length, and  $f(t)$  is the spring force at end2.  $\|e_2(t) - e_1(t)\|$  is the vector norm representing the length of the spring at time  $t$ ,  $\|e_2(t) - e_1(t)\| - l$  is the signed length change from the rest length, and  $(e_2(t) - e_1(t))/\|e_2(t) - e_1(t)\|$  is a unit vector with direction from end1 to end2.

A *model* is a composition of model fragments applicable to a physical system in a particular situation. *Simulation* is the execution of a model.

The motion of an entity at any instant can be described by a set of ordinary differential equations in the twelve components of four vectors: position, orientation, velocity, and angular velocity.<sup>1</sup> The differential equations are usually nonlinear and do not have a solution in closed form, so they must be solved by numeric integration. For a moving entity, ORACLE constructs a model with the twelve components of the four vectors (position, orientation, velocity, and angular velocity) as *state variables*, which take numeric values during simulation.

The state variables of each subpart of an entity are initially defined in the local reference frame, which is assumed to be fixed to the entity. Then each subpart defined in its local reference frame is translated and rotated by having its reference frame redefined in a common inertial reference frame. The system chooses the common inertial reference frame from local reference frames which are not accelerated. If there is no such reference frame (i.e., all the local reference frames are noninertial), it introduces a new inertial reference frame. If there are several inertial reference frames, the choice is arbitrary.

<sup>1</sup>The degrees of freedom of a moving entity are six instead of twelve because the velocity function and the angular velocity function are derivable by differentiating the position and orientation functions, respectively. The motion of a physical system with  $n$  subparts can be characterized by maximum  $12n$  state variables with  $6n$  degrees of freedom.

## Algorithm

ORACLE takes as input a description of a physical system in terms of the entities of the system, any constraints to be satisfied, and the properties of the entities (i.e., variables) whose values are going to be computed by modeling and simulation. As output, it produces a model of the motion of the system and the variable values obtained by solving the model. The algorithm of ORACLE consists of three phases: (1) problem analysis, (2) model creation, and (3) model execution. In the first phase, ORACLE represents each entity of a problem statement in a frame by copying a class frame and filling in slots for property values specified in the

---

### Algorithm 1 ORACLE's top-level algorithm

**Problem Analysis** Analyze a problem statement.

1. Analyze entities, and create frames of the entities and a set INIT of initial-value conditions.
2. For each constraint, determine its type and represent them in equations.
3. Analyze variables and generate a set DRVD of differential equations.

**Model Creation** Search for relevant model fragments and compose a behavioral model with them.

1. For each entity  $E$  of the problem statement
  - For each model fragment  $MF$  indexed by the "mf" slot of  $E$ 
    - If  $MF$  has not been instantiated for  $E$ 
AND every variable of  $MF$  either
corresponds to an entity property or variable
of the input or can be derived from them
AND the assumption (if any) of  $MF$  does
not violate any entity property or constraint
of the input
      - Put  $MF$  in a list MFS.
2. model  $M = \text{DRVD}$
3. #equations = #equations( $M$ )
4. retry: For each model fragment  $MF$  in MFS
  - (a) Instantiate  $MF$  for the problem.
  - (b)  $M = M \cup \{MF\}$
  - (c) #equations = #equations( $M$ )
  - (d) If #equations = #variables, do **model execution**.
5. Print the dead-end situation, and quit.

**Model Execution** Solve the model  $M$  either analytically or by numeric simulation.

1. Determine the types of equations of the model and solve them with INIT for the variables.
  2. If a valid solution is obtained, print the model and solutions, and quit.
  3. If a valid solution is not obtained, retract the most recent  $MF$  from the model and go to retry.
-

problem statement. It also transforms vector quantities expressed in the local reference frames into those in the inertial reference frame, formulates initial conditions, and executes if\_added procedures in the slots. A model fragment specifying forces on a component of a composite object is instantiated by if\_added procedures in the this phase. After constraints are analyzed, variables are examined to determine if their values are already known in their slot values or derivable from other variables. In the second phase, additional model fragments which have not been instantiated are retrieved and a model is constructed from them. In the final phase, the constructed model is solved for the problem. If ORACLE runs out of potentially relevant model fragments before it finds a valid solution, it prints the situation, asks more information, and quits. The top-level algorithm of ORACLE is outlined in Algorithm 1.

### Example

We illustrate how ORACLE works with the spring-block system of Figure 1a. Suppose the following problem description is given as an input. There is no particular constraint in this problem and the system is asked to compute the four vector variables (twelve variables in component form) of the block.

```

entities=[b1=[block, mass=1,
  principal_moments_of_inertia=[1/6,1/6,1/6],
  position(0)=[3,0,0],
  orientation(0)=[Pi/4,Pi/2,0],
  velocity(0)=[0,0,0],
  ang_velocity(0)=[0,0,0]],
s1=[spring, force_const=10,
  damping_coeff=1/10, rest_length=3/2,
  end1(t)=[0,0,0], end2(t)=b1[-1/2,0,0]],
sb=[composite_object, parts={b1,s1}]];
constraints=[ ];
variables=[b1[position(t)], b1[orientation(t)],
  b1[velocity(t)], b1[ang_velocity(t)]];

```

For each entity b1, s1, and sb, a frame is created and the given properties of the entities are recorded in their slot values. The if\_added procedure in the end2 slot of s1 computes the spring force acting on b1 using a model fragment Springforce2 and records the value in the force slot of b1. The position of end2 in the inertial reference frame is computed from a translation and a rotation of the local reference frame of b1. The initial conditions of the block are also formulated. Here are the Maple variable names ORACLE assigns for this example.

```

INIT =
{b1x(0)=3, b1y(0)=0, b1z(0)=0,
 b1phi(0)=0, b1theta(0)=Pi/2, b1psi(0)=0,
 b1u(0)=0, b1v(0)=0, b1w(0)=0,
 b1omega1(0)=0, b1omega2(0)=0, b1omega3(0)=0}

```

None of the four state variables of b1 can be assigned a value simply by looking at slot values of b1, but the velocity and the angular velocity functions can be derived by differentiating the position and the

orientation functions, respectively, according to their if\_needed facets. The system generates trivial differential equations for the velocity and angular velocity by the procedures attached to the if\_needed facets.

```

DRVD =
{b1u(t)=diff(b1x(t),t),
 b1v(t)=diff(b1y(t),t),
 b1w(t)=diff(b1z(t),t),
 b1omega1(t)=diff(b1theta(t),t)*cos(b1phi(t))+
  diff(b1psi(t),t)*sin(b1theta(t))*sin(b1phi(t)),
 b1omega2(t)=diff(b1theta(t),t)*sin(b1phi(t))-
  diff(b1psi(t),t)*sin(b1theta(t))*cos(b1phi(t)),
 b1omega3(t)=diff(b1phi(t),t)+
  diff(b1psi(t),t)*cos(b1theta(t))}

```

Now ORACLE focuses on finding equations for the position and the orientation. The equations for them cannot be derived from other variables since they are basic variables, so ORACLE looks for relevant model fragments. It examines model fragments, indexed by the mf slot of the block. ORACLE decides that Newton2 and Euler are potentially relevant because the entities (solid and rigid body, respectively) of the model fragments are superclass of a block and the equations of the model fragments contain at least one variable of the problem. Model fragments of Newton2 and Euler are as follows.

```

Newton2=[phenomenon='Newton's second law
  of motion',
  entities=[r=solid],
  variables=[f(t)=r[net_force(t)],
  p(t)=r[momentum(t)]],
  assumptions=[ ],
  equations=[f(t)=diff(p(t), t)]]

Euler=[phenomenon='time-dependency of
  ang_velocity',
  entities=[b=rigid_body],
  variables=[Omega(t)=b[ang_velocity(t)],
  M(t)=b[ang_momentum(t)],
  T(t)=b[net_torque(t)]],
  assumptions=[ ],
  equations=[add(diff(M(t), t),
  crossprod(Omega(t),
  M(t)))=T(t)]

```

The entity and variable names of the model fragments are instantiated as those of the problem and they are substituted in the equations of the model fragments. The angular momentum is derived from principal moments of inertia and angular velocity by the if\_needed procedure in the ang\_momentum slot. Likewise, the net torque is derived from force and position vector of the point at which the force acts.

$$M(t) = \begin{pmatrix} I_1 \Omega_1(t) \\ I_2 \Omega_2(t) \\ I_3 \Omega_3(t) \end{pmatrix}, \quad T(t) = \sum r \times f(t)$$

The principal moments of inertia ( $I_1, I_2, I_3$ ) and the position vector ( $r$ ) of the spring-attached point can be assigned from the information of the problem description. The angular velocity is one of the state variables

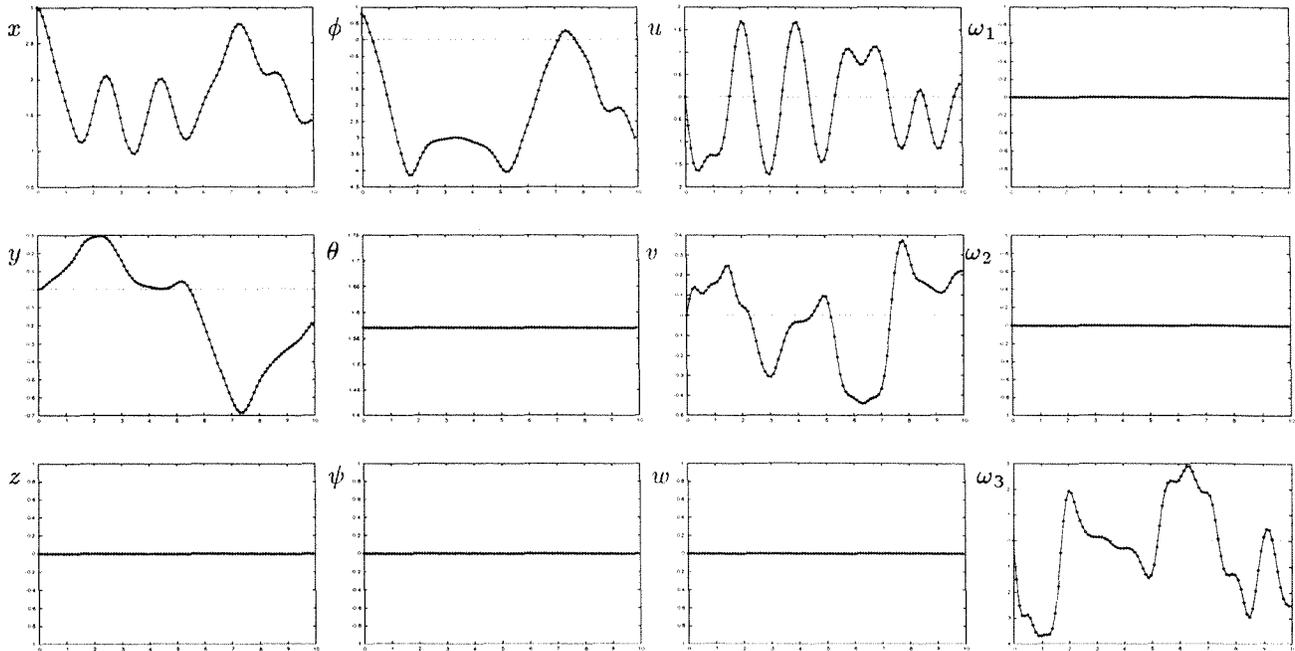


Figure 3: Plots of the 12 state variables of the block b1 as functions of time. The first column shows the position vector, the second column the orientation vector, the third column the velocity, and the last column the angular velocity.

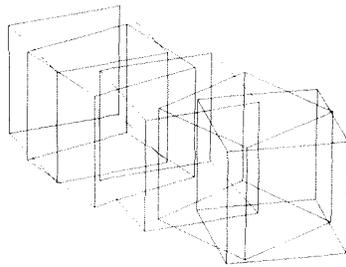


Figure 4: Motion of the block b1. Spring not shown.

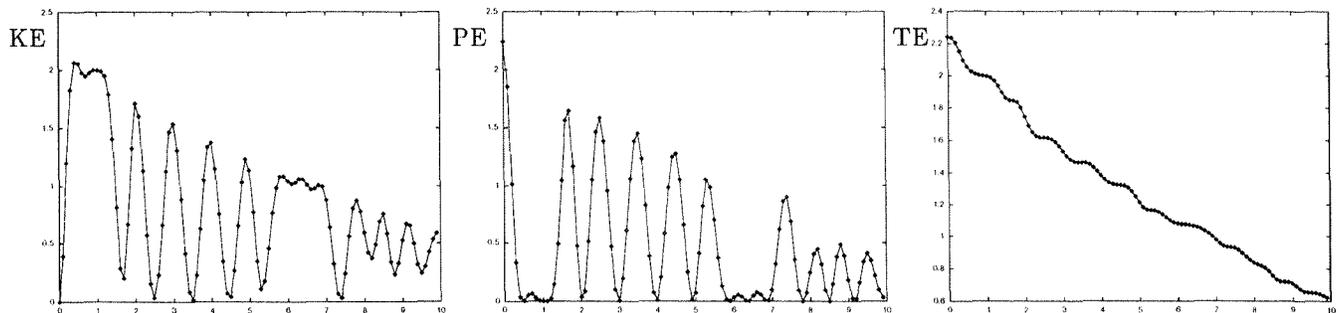


Figure 5: The kinetic, potential, and total energy of the single spring-block system as functions of time during the simulation.

asked by the problem, and its functions are derived in DRVD. The value of force  $f(t)$ , which has computed using a model fragment Springforce2 (shown earlier in section ), is available in the force slot of b1, and substituted in the equations of Newton2 and Euler.

The system has now total 12 equations in component forms (6 from the model fragments and 6 from DRVD) plus 12 initial conditions for 12 unknowns. Several of the differential equations are nonlinear, and when ORACLE attempts to solve the model analytically, it does not find a solution in closed form. ORACLE then solves the differential equations by numeric simulation. ORACLE displays the simulation result by showing the state variables as functions of time using gnuplot (Figure 3). Animation of the moving block is shown (Figure 4) using PADL-2 solid modeling system (Hartquist 1983). Figure 4 contains several animation scenes superimposed. Note that the motion of the block is much more complex than that of the linear harmonic oscillator. The kinetic energy, potential energy, and total energy of the system are also displayed as part of validation criteria of the results (Figure 5). The total energy in Figure 5 decreases over time due to the nonzero damping coefficient of the spring of the problem statement.

## Extension to Broader Class of Physical Systems

### Multiple Spring-block Systems

The previous section showed how ORACLE predicts the behavior of the single spring-block system. Can the modeling system of the single spring-block system be used to predict the behavior of the multiple spring-block systems such as Figure 1b and Figure 1c? The answer is "yes". The multiple spring-block systems have additional entities and phenomena, but they are simply the multiple occurrences of the same types as the single spring-block system. Having already enough knowledge represented in general form to handle the single spring-block system, ORACLE can handle the multiple spring-block systems with no change. The way it solves the problem is the same. It computes the positions of ends of each spring in the inertial reference frame by transforming the local reference frame of its associated block, and derives differential equations for the velocity and angular velocity of the blocks from the procedures attached to the if\_needed facets. It then instantiates model fragments of Spring force, Newton2 and Euler, and composes a model. Notice that model fragment sharing occurs within the models because each of those model fragments is instantiated more than once for different entities. The result of the execution of the models indicate that although none of the blocks are initially rotated, both blocks of Figure 1b and the end blocks of Figure 1c rotate as well as translate due to spring forces which are not parallel to the radius vectors of the points to which the springs are attached. If the spring damping is ignored (i.e., damp-

ing\_coeff = 0), the middle block of Figure 1c shows translational motion only, but it shows both translational and rotational motions if the spring damping is considered (damping\_coeff  $\neq$  0). Figure 6 shows part of animation scenes for the case when the spring damping is considered. In fact ORACLE is able to handle multiple rigid bodies connected by springs in arbitrary positions and orientations because the way of identifying relevant model fragments and composing them is not restricted by the number of entities or their connections.

### Sailboat in Fluid

A sailboat is a composite object whose driving force comes from the differential motion of air over water. Before we model the sailboat, we can ask the same question as before. Can we use the modeling system of the spring-block systems to predict the behavior of a sailboat in fluids? The answer is "yes", provided that the modeling system has enough domain knowledge to handle the problem. We do not need to build a different modeling system. A modeling system with the same algorithm and the same model fragments plus additional model fragments and entities can predict the behavior of the sailboat.

New classes of entities added to the knowledge base are fluids (water and air) and lifting surfaces (hull and sail). The sailboat, water, and air entity have their own reference frames, which move as their entities move. New phenomena include hydrodynamic and aerodynamic forces, each with two components (lift and drag), and skin friction. A single model fragment is used to represent both hydrodynamic and aerodynamic frictional drag forces, and later instantiated for them. Likewise, a single model is used to represent both hydrodynamic and aerodynamic lift forces.

```
FDrag=[phenomenon='frictional drag force on
an object in fluid',
entities=[s=physical_object, f=fluid],
variables=[FD=s[fdrag(t)],
v=s[rel_fluid_speed(t)],
fd=s[rel_fluid_direction(t)],
Pa=s[parasitic_area],
rho=f[density]],
assumptions=[ ],
equations=[FD=1/2*Pa*rho*v^2*fd]]
```

```
Lift=[phenomenon='lift and lift induced force on
an object in fluid',
entities=[s=physical_object, f=fluid],
variables=[LF=s[lift(t)],
L=s[lift_magnitude(t)],
v=s[rel_fluid_speed(t)],
fd=s[rel_fluid_direction(t)],
pd=s[perpendicular_rel_fluid_dir(t)],
Ca=s[effective_capture_area],
rho=f[density]],
assumptions=[s[rel_fluid_speed(t)] > 0],
equations=[LF=L*pd+L^2/(2*Ca*rho*v^2)*fd]]
```

For the hull, the rel\_fluid\_speed is the speed of the boat relative to water. For the sail, the rel\_fluid\_speed

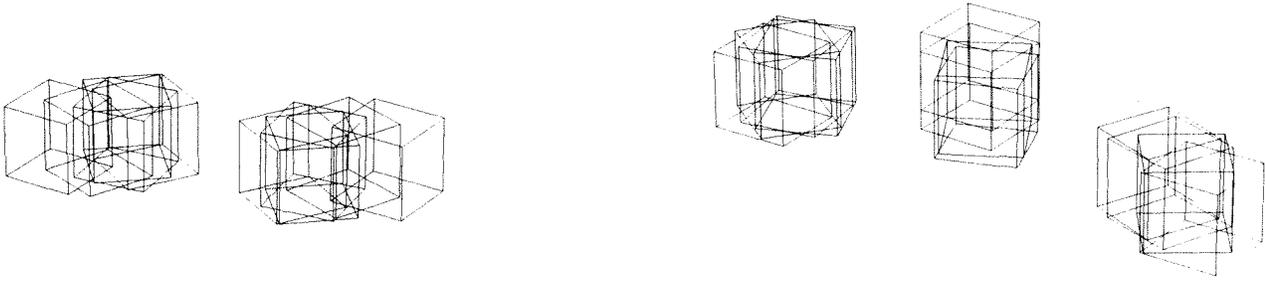


Figure 6: Motion of the multiple spring-block systems in Figure 1b and Figure 1c. Springs not shown.

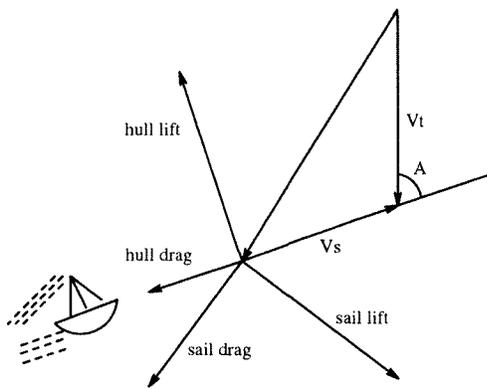


Figure 7: Directions of force components on a sailboat, adapted from (Letcher, 1976)

is the speed of the boat relative to air. The magnitude and direction of the hydrodynamic forces acting on the hull depend on the `rel_fluid_speed` of the hull, and the aerodynamic forces depend on the `rel_fluid_speed` of the sail. `rel_fluid_direction` is an angle between the direction of fluid and the direction of an object, represented in angle. Drag forces acting on the hull have components opposite to the direction of its `rel_fluid_speed`. Lift forces on the hull are perpendicular to the direction of its `rel_fluid_speed`. Similarly, drag forces on the sail have components opposite to the direction of its `rel_fluid_speed`, and lift forces on the sail are perpendicular to the direction of its `rel_fluid_speed`. Lift forces on the hull and sail of a sailboat are horizontal, not vertical.

The directions of the force components are summarized in Figure 7. In the figure, water is assumed to be at rest (i.e., speed = 0) and  $V_t$ ,  $V_s$ , and  $A$  denote the wind speed, sailboat speed, and course angle from the wind direction, respectively.

Notice that the model fragment `Lift` has a nonempty assumption slot, saying that the relative fluid speed must be positive. When both the fluid and the object are at rest or the fluid has the same speed as the object in the same direction, the relative fluid speed becomes zero and the equations of the model fragment

cannot be defined due to zero denominator. Thus we have another model fragment of lift with a different assumption slot; it says that lift force is zero when the relative fluid speed is zero.

```
Lift_at_zero_speed=[
  phenomenon='lift and lift induced
              force on an object in fluid',
  entities=[s=physical_object, f=fluid],
  variables=[L=s[lift(t)],
            sv=s[rel_fluid_speed(t)]],
  assumptions=[s[rel_fluid_speed(t)] = 0],
  equations=[L[1]=0,
            L[2]=0,
            L[3]=0]]
```

During problem solving, ORACLE automatically chooses between the two model fragments of lift by checking their assumptions. At present, the kinds of assumptions ORACLE can process are confined to the algebraic properties of entities, such as numeric ranges or arithmetic expressions.

After the entities and model fragments are added, ORACLE can solve several types of problems on a sailboat, but we will focus on one type of problem in this section. Suppose that a sailboat is heading in the angle of 49 degrees from the direction of wind at uniform speed 16.9 ft/sec and that water is at rest. The system is asked to compute the sailboat speed which will balance all the forces involved.

ORACLE first infers all the forces on the sailboat from the forces acting on its components, hull and sail. It instantiates the model fragments `FDrag` and `Lift` for each of them and records the summation of them in the `net_force` slot of the sailboat.

$$F = \sum_{i \in \{\text{hull, sail}\}} (F\text{Drag}_i + \text{Lift}_i)$$

It then searches for a model fragment which relates forces with speed, and finds the model fragment of `Newton2`. It substitutes the equations of the forces in the equation of `Newton2`,  $F(t) = d(p(t))/dt$ . Since the problem states that all the forces are balanced, the net force on the sailboat must be zero, implying the momentum  $p(t)$  is constant. The right hand side of the equation becomes zero from the constant momentum, resulting in an algebraic equation. However, the

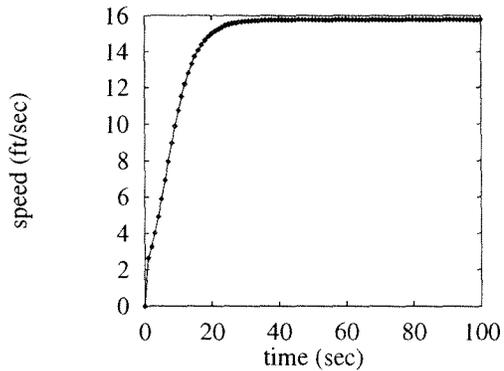


Figure 8: The sailboat speed as a function of time.

problem is under constrained in the sense that total number of equations in component form is 2 ( $F_x=0$ ,  $F_y=0$ ,  $F_z$  becomes a trivial equation  $0=0$ ) but the total number of unknowns in the equations is 3 (boat speed, sail lift magnitude, and hull lift magnitude). ORACLE prints the situation, asking for further information. The user provides an additional equation,  $\partial(F_x)/\partial(\text{sail\_lift\_magnitude}) = 0$ , by making a simplifying assumption that the sail is controlled as to maximize the sailboat force in the direction of boat heading. The equations are solved algebraically, producing a solution, boat speed = 15.7 ft/sec. The solution is checked against the range facet of the speed slot of the boat, which says that value of the boat speed must be in the range of  $[0 \dots \text{infinity}]$ . Since the solution of 15.8 is included in the range, it is returned to the user as a valid solution. However, if there are several solutions, only those within the values specified by the range facet (if any) are selected as valid solutions. When the information in the range facet is not sufficient to choose a correct solution from multiple solutions, additional or alternative procedure for filtering correct solutions is to use the result of the numeric simulation of its corresponding differential model, as we will show below. A correct solution of an algebraic model describing a behavior in a steady state must agree with the numeric simulation result of its corresponding differential model. This procedure of validating the solution of an algebraic model against the numeric simulation of a differential model has not been automated in the current implementation of ORACLE, and the user should try both models to compare their results.

The previous example showed how ORACLE composes a model to compute the sailboat speed in the equilibrium state of forces. If we are interested not only in such a speed but also in how the boat arrives at the speed, starting from zero speed, the boat speed must be computed as a function of time. Relevant model fragments are retrieved and instantiated in a similar way. In this case, however, the net force on the sailboat is not necessarily zero all the time because the boat ac-

celerates until it reaches the equilibrium state of forces. Therefore, the right hand side of the equation of Newton2 does not become zero, but stays as  $d(p(t))/dt$ . Since  $p(t) = d(m \cdot v(t))/dt$ , ORACLE solves the differential equation,  $F(t) = d(m \cdot v(t))/dt$  for  $v(t)$  by numeric simulation. A plot of the simulation result in Figure 8 shows that the sailboat ultimately accelerates to the same speed as the one predicted by the algebraic method, thus confirming the algebraic solution. Also notice that the model fragment Newton2 used for modeling the spring-block systems is reused for modeling the sailboat and that model fragments Lift and FDrag are shared by hull and sail.

### Issues in Scaling Up

There are several problems raised in scaling up ORACLE not only to broaden the types of physical systems to be modeled but also to model a physical system with a large number of components and phenomena involved.

First, the size of a model generated by the current version of ORACLE can restrict scaling up either because of practical limitations of solving a huge model or because solving a huge model takes too much time to be useful. Table 1 shows the sizes of the models and the times for formulating and solving the models for the examples shown in this paper. The model sizes of spring-block systems with different configurations from the examples in Table 1 are about the same as those of the spring-block systems with the same number of blocks and springs in Table 1, that is, the model sizes of spring-block systems are independent of their configurations. As we can see in Table 1, the size of a model is not directly proportional to the number of model fragments instantiated. Rather it is proportional to the number of variables specified in the problem statement, or to the number of unknowns in the equations of the model ( $\#\text{unknowns}$  includes  $\#\text{variables}$  and newly generated variables in the equations).

As described earlier, a model contains a set of variables, a set of assumptions, a set of names of model fragments, and a list of equations. The equations of a model are composition of the equations of the model fragments instantiated to construct the model. Although the equations of most model fragments exist in a very short, simple form *before* instantiation, they may become very long and complex *after* they are instantiated for the particular entities and physical phenomena in the problem. This explains the large variations in the sizes of models with similar number of model fragments; the big size of a model is attributed to the long equations of instantiated model fragments. Even for a same model fragment, the equations after instantiation can be very different in their sizes depending on for which variables they are to be solved and which variables of the equations are known.

Figure 9 shows the growth rate of the model size and the growth rate of the time for modeling and simula-

example	#variables	#unknowns	#model fragments	#lines of model
	model fragments (#instantiation)			
	model_gen time	model_sol time	display_save time	total time
algebraic model of boat	1 FDrag (2), Lift (2), Newton2 (1) 4.900	3 4.616	5 4.917	30 14.333
differential model of boat	1 Newton2 (1), FDrag (2), Lift (2), Lift_at_zero_speed (1) 4.700	3 18.700	6 4.716	31 28.116
one block system	12 Newton2 (1), Euler (1), Springforce2 (1) 14.583	12 19.800	3 14.833	206 49.216
two block system	24 Newton2 (2), Euler (2), Springforce1 (1), Springforce2 (1) 108.533	24 118.283	6 110.433	2010 337.249
three block system	36 Newton2 (3), Euler (3), Springforce1 (2), Springforce2 (2) 339.350	36 264.616	10 344.767	5330 948.733

Table 1: Size of a model and time for formulating and solving it for each of the ORACLE examples. #variables is the number of variables in component form, specified in the problem statement; #unknowns includes #variables and newly generated variables in the equations; #model fragments is the total number of model fragments instantiated for the model; #lines of model is a count of lines of the model; model\_gen time is CPU time for generating a model; model\_sol time is CPU time for solving the equations of a model; display\_save time is CPU time for displaying the result of solving the model and saving all the results; the units of the CPU times are in seconds.

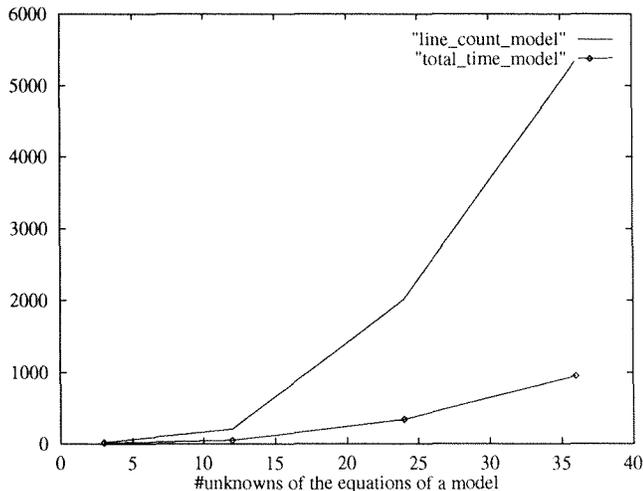


Figure 9: The size of a model and the time for formulating and executing the model. For the count of lines and times of a model with 3 unknowns, the average values of the 2 models with 3 unknowns (algebraic model of boat, differential model of boat) are used. The units of the model sizes are in the line counts of models and the units of the times are in seconds.

tion as a function of the number of unknowns of the a model. The size of a model is measured in terms of the number of lines of the model. When the number of unknowns increases from 3 to 12, the model size increases from 30.05 lines to 206 lines, which is almost 7 times. This is because the spring-block example involves much more complex computation which results in long equations. It is also notable that when the number of unknowns increases from 12 to 24, the model size increases from 206 lines to 2010 lines, about 10 times, and that when the number of unknowns increases from 24 to 36, the model size increases from 2010 lines to 5330 lines. Given limited data, the growth rate of a model is roughly quadratic in the number of unknowns. The total time for generating a model and solving it is also proportional to the number of unknowns of the model, but the growth rate of time is not as fast as that of the model size, as shown in Figure 9.

Some of the large models have long equations which cannot be simplified further in their nature unless we decide to produce approximate models instead. However, some models may be simplified without losing accuracy of their predictions by doing additional processing on the equations instead of applying Maple-builtin simplification functions. Reformulating the model generation and solving process of ORACLE is another direction to consider in order to efficiently construct and solve a large model. Restricting ORACLE to a certain type of physical system is another way to scale up the modeling system to handle a complex physical system with a large number of components and

phenomena involved. For example, if we want a special purpose modeling system for spring-block systems only, we can make the modeling system generate a single model that works for any number of blocks. As the number of blocks increases, the model would need more data storage but the model itself does not become larger. This kind of approach to scaling up a modeling system has advantage of being able to model and simulate a complex physical system without the size problem of a generated model, but has disadvantage of losing the breadth of physical systems covered by the modeling system.

Second, selecting relevant model fragments would become a more important issue in scaling up. Including additional properties of an object, such as electrical or thermal properties, in the description of an entity as slots does not cause a problem in selecting model fragments because when ORACLE determines the relevance of a model fragment to a given problem it checks whether each variable of the model fragment is mentioned as an entity property or variable of the problem statement or derivable from them. The difficulty is in selecting a model fragment among multiple model fragments with same or similar variables but with different assumptions. In the current implementation, model fragments are indexed by the mf slots of entities, with more frequently relevant ones first, and therefore the search process is sensitive to the order of the model fragments. We may want a more efficient method for organizing model fragments which will facilitate identifying and retrieving relevant model fragments.

The third problem in scaling up is related to the size of a generated model. As a model gets bigger, it would become more difficult to understand the model or to validate its solution. Having ORACLE provide an explanation of its solution, or checking the solution against that of an approximate model or experimental data will help understand or validate the model.

## Related Work

Falkenhainer and Forbus (1991) describe a form of compositional modeling where a device model is automatically formulated by composing a set of relevant model fragments which are initially obtained by matching the terms of a query to a domain theory and then elaborated later. There are several differences between our work and theirs. In ORACLE we distinguish model fragments from entities; model fragments are used for describing physical phenomena and entities for objects (both composite and primitive); model fragments are indexed by the "mf" slot of an entity. In Falkenhainer and Forbus' approach, model fragments are used for describing all the phenomena, objects, and devices, and are organized into mutually exclusive sets called assumption classes. When the class condition holds, one and only one of the assumptions associated with the class must hold and the model fragment containing that assumption must be included in a model. Once

the model fragments with appropriate assumptions are selected, the process of instantiating the model fragments and assembling them is straightforward. While a composite object in ORACLE can consist of any heterogeneous parts, a unique minimal covering of parts taken from a single part-of hierarchy is required to exist in their approach to generate a simplest possible model. They do not have a capability of handling detailed structural relations among parts and choosing appropriate reference frames for parts, and therefore cannot handle complex motions such as motion of multiple objects. For behavior generation Falkenhainer and Forbus use either qualitative simulation by QPE (Forbus 1990) or quantitative simulation by numeric simulation, whereas we use numeric simulation or analytic method.

In (Nayak 1992), Nayak describes a method to construct a device model by selecting an appropriate model for each component of the device using structural, behavioral, and expected behavioral constraints. In his system, a model is formulated by composing a set of model fragments, as in ours. However, the uses of the models produced by the two systems are different. While ORACLE constructs a model to predict motions of physical systems, his system builds a model to explain causal relations between parameters of a device. Another difference is that he uses order of magnitude reasoning for behavior generation while we use numeric simulation. His order of magnitude reasoning method is restricted to generating the behavior at a fixed point in time, but we can predict the behavior changing with time as well as the behavior at a fixed point in time.

The SIGMA system developed at NASA Ames Research Center (Keller and Rimón 1992) is a tool which aids a scientist-user in building a model. After the interaction with the user, it produces a model specified in data flow graph and executes the model to compute an unknown quantity. Like ORACLE, SIGMA organizes and represents domain knowledge in frame structures. However, it is a user-assistant system rather than an autonomous model-building system, and has several restrictions in constructing and executing a model, which ORACLE does not have. For example, multiple quantities cannot be computed at the same time because it cannot solve more than one equations simultaneously, and the types of equations are restricted to algebraic equations or first-order ordinary differential equations; model fragments cannot be put together in an arbitrary order due to the strict backchaining control strategy of its model building process. It converts the input values into a common, consistent set of scientific units, but does not have a provision to transform a vector quantity measured in one reference frame to another, which is necessary in dealing with moving objects.

The MSG system developed by Ling *et al.* (1993) generates mathematical models for analyzing heat transfer behavior. The approach of the MSG system to building a model is similar to that of ORACLE in the

sense that it is compositional. However, there are several differences. While ORACLE focuses on modeling physical systems involving motion, MSG models physical systems involving heat flow. Therefore, the domain knowledge the two systems use are different. A second difference is that ORACLE represents the knowledge explicitly in general, declarative form, but much of the knowledge that the MSG system uses is embedded in the system as part of its algorithm. A third difference is that, while ORACLE generates a model and then solves the equations of the model to predict the behavior of a given physical system, MSG presently does not solve the equations of its generated model.

Another relevant line of work concerns model selection (Addanki *et al.* 1991; Weld 1992), or model simplification (Yip 1993; Falkenhainer 1993), rather than model generation.

Yet another related works concern simulation generation instead of model generation. The SIMLAB system (Palmer and Cremer 1991) produces a simulator from a user-provided physics model. Given a mathematical model of a physical phenomenon and instructions for solving the resulting equations, SIMLAB transforms the model into an executable simulation code to analyze the phenomenon. However, the user still has the burden of creating the mathematical model. The program built by Berkooz *et al.* (1992) is similar to SIMLAB. It is basically a compiler for translating differential equations expressed in mathematical and programming constructs into an executable code. The SINAPSE system (Kant 1992) also automatically transforms a given model into a program in desired language, though again the human user must create the input model.

A number of mechanical device simulators are commercially available, such as ADAMS (Dawson 1985), and DADS (Haug 1989). These programs, like most simulators, incorporate physics knowledge such as Newton's laws of motion directly into algorithms rather than representing them explicitly. The simulators include powerful algorithms for forming and solving the equations of motions for a wide variety of mechanisms, but lack the flexibility that ORACLE has to explicitly instantiate general model fragments in particular situations.

Previous AI research in spatial reasoning about mechanical devices (Faltings 1987; Gelsey 1989; 1994; Joskowicz and Sacks 1991) has devoted considerable attention to reasoning about contacts between solid bodies, a problem ORACLE does not presently address. Like the commercial simulators, these programs incorporate knowledge of physical phenomena directly into algorithms rather than attempting to explicitly instantiate general model fragments in particular situations, as ORACLE does.

### Future Work

There are several directions in which the work described in this paper can be extended. In the cur-

rent implementation of ORACLE, there are only certain classes of entities and model fragments available in the knowledge base. Adding more model fragments and entities would expand the types of physical systems covered by ORACLE. It would also be a valuable test for the extensibility of the system.

Problems which do not involve new physical phenomena, but require model fragments with different assumptions or representations will involve minor changes. For example, the two model fragments for spring forces presently assume a linear spring with linear damping. To model nonlinear springs, we need to add a new model fragment with different equations and assumptions. The types of springs should be considered when an if-added procedure chooses between the two model fragments of spring forces.

Some spatial reasoning problems can be solved by qualitative interpretation of the quantitative models produced by ORACLE. For example, qualitative description of motions (such as translational, rotational, oscillatory, or tumbling) can be easily obtained by postprocessing the simulation results of the models. Coverage of space of a moving object, any regularity of the coverage over time (such as monotonically decreasing coverage of a damped spring), or possible contact/collision with other moving objects (intersection of the coverages during same time intervals) can also be produced by postprocessing the simulation results.

When something goes wrong during problem solving (e.g., a model cannot be solved due to fewer equations than unknowns), ORACLE currently prints the dead-end situation, asks for further information, and quits. The user has to figure out the cause of the problem and rerun the program with new information. In order for ORACLE to suggest possible directions to fix the problem, it must have a capability of reasoning about equations and unknowns.

### Conclusion

The model of general motion in three dimension is difficult to formulate by hand but important because of its relevance to many practical applications, including computer graphics, robotics, and design. We have made important progress in automating the model-building process for physical systems with multiple moving objects in arbitrary configurations by developing a new method which uses basic domain knowledge. The method has been implemented in a working program called ORACLE and tested in the domains of mechanical devices and sailboats. Given a description of a problem involving a moving physical system, ORACLE automatically identifies relevant model fragments, instantiates them for the particular entities and physical phenomena in the problem, composes the instantiated fragments to form a model, and simulates the model to solve the problem. Knowledge of physical phenomena is represented with model fragments which can be shared and reused by many models. Most of

the knowledge is just the same fundamental equations that appear in any standard mechanics textbook, with their implied semantics of vectors and frames of references. Starting with the most basic, simple concepts in the domain of mechanics, ORACLE can still generate a powerful model for complex motions. This is a new method which solves several problems with existing AI modeling work on motion by: (1) explicit handling of vector quantities and frames of reference; (2) simultaneous handling of multiple equations (algebraic or differential, linear or nonlinear); and (3) declarative, algorithm-neutral representation of physics knowledge.

As discussed earlier, there are many programs developed for reasoning about motion of mechanical devices. However, the programs do not solve problems in a general method from basic physics principles, but rely on specific methods specialized for certain classes of problems. Letcher (1976), for example, uses a numerical procedure adapted from Newton-Raphson iteration to find the optimum sailboat velocity with the maximum component in the wind direction, or the sailboat velocity which will balance all the forces. ORACLE solves the same sailboat problems without requiring special-purpose problem solving methods, as it does for other mechanical devices.

## References

- S. Addanki, R. Cremonini, and J. S. Penberty. Graphs of Models. *Artificial Intelligence*, 51:145-177, 1991.
- G. Berkooz, P. Chew, J. Cremer, R. Palmer, and R. Zippel. Generating spectral method solvers for partial differential equations. Technical Report 92-1308, Cornell University, 1992.
- B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *Maple V Language Reference Manual*. Springer-Verlag, New York, 1991.
- G. Dawson. The Dynamic Duo: Dram and Adams. *Computers in Mechanical Engineering*, March 1985.
- B. Falkenhainer and K. D. Forbus. Compositional modeling: finding the right model for the job. *Artificial Intelligence*, 51:95-143, 1991.
- B. Falkenhainer. Ideal physical systems. In *Proc. of the 11th National Conference on Artificial Intelligence*, pages 600-605, 1993.
- B. Faltings. *Qualitative Place Vocabularies For Mechanisms in Configuration Space*. PhD thesis, University of Illinois at Urbana-Champaign, July 1987.
- K. D. Forbus. Qualitative Process Theory. *Artificial Intelligence*, 24:85-168, 1984.
- K. D. Forbus. The Qualitative Process Engine. In D. S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 220 - 235. Morgan Kaufmann, 1990.
- A. Gelsey. Automated Physical Modeling. In *Proc. of the 11th International Joint Conference on Artificial Intelligence*, pages 1225-1230, 1989.
- A. Gelsey. Automated reasoning about machines. *Artificial Intelligence*, 1994. to appear.
- G. Hartquist. Public PADL-2. *IEEE Computer Graphics and Applications*, pages 30-31, October 1983.
- E. J. Haug. *Computer Aided Kinematics and Dynamics of Mechanical Systems, Volume 1: Basic Methods*. Allyn and Bacon, Boston, etc., 1989.
- L. Joskowicz and E. P. Sacks. Computational Kinematics. *Artificial Intelligence*, 51:381 - 416, 1991.
- E. Kant. Code synthesis for mathematical modeling. In *Working Notes of AAAI Fall Symposium on Intelligent Scientific Computation*, pages 54-59, 1992.
- R. M. Keller and M. Rimón. A Knowledge-based Software Development Environment for Scientific Model-Building. In *Proc. of the 7th Knowledge-Based Software Engineering Conference*, 1992.
- B. Kuipers. Qualitative Simulation. *Artificial Intelligence*, 29:289-388, 1986.
- J. S. Letcher, Jr. Optimum windward performance of sailing craft. *Journal of Hydronautics*, 10:140-144, 1976.
- S. R. Ling, L. Steinberg, and Y. Jaluria. MSG: A Computer System for Automated Modeling of Heat Transfer. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 7(4):287-300, 1993.
- M. Minsky. A Framework for Representing Knowledge. In *The Psychology of Computer Vision*, pages 211 - 277. McGraw-Hill, New York, 1975.
- P. P. Nayak. *Automated Modeling of Physical Systems*. PhD thesis, Stanford University, 1992. STAN-CS-92-1443.
- R. S. Palmer and J. F. Cremer. SIMLAB: Automatically creating physical systems. Technical Report 91-1246, Cornell University, 1991.
- P. Struss. Global Filters for Qualitative Behaviors. In *Proc. of the 7th National Conference on Artificial Intelligence*, pages 275 - 279, 1988.
- D. S. Weld. Comparative Analysis. *Artificial Intelligence*, 36:333-374, 1988.
- D. S. Weld. Reasoning about model accuracy. *Artificial Intelligence*, 56:255-300, 1992.
- B. C. Williams. Doing Time: Putting Qualitative Reasoning on Firmer Ground. In *Proc. of the 5th National Conference on Artificial Intelligence*, pages 105 - 112, 1986.
- K. M. Yip. Model Simplification by Asymptotic Order of Magnitude Reasoning. In *Proc. of the 11th National Conference on Artificial Intelligence*, pages 634-640, 1993.

# A Theoretical Analysis of Thought Experiments

David L. Hibler

Department of Computer Science

Christopher Newport University

Newport News, VA 23606.

tele: (804)-594-7065

e-mail: dhibler@pcs.cnu.edu

## Abstract

The thought experiment methodology is a recently developed technique for qualitative reasoning. Thought experiments involve four main steps: (i) simplification of the original problem, (ii) solution of the simplified problem, (iii) conjecture the result of the original problem based on the solution of the simplified version, and finally (iv) verification that the conjecture is correct. This methodology has been implemented and has been applied successfully to problems involving static electricity, the motion of charged pendulums, fluid flow, and thermodynamic cycles. The purpose of this paper is to analyze this technique using probabilistic models

## Introduction

Imaginary, simplified situations are often analyzed by human problem solvers in order to understand the principles behind more realistic situations. In physics, this technique is sometimes referred to as a thought experiment [Prigogine & Stengers 1984].

As an example of a thought experiment, consider how to solve the following problem.

*A certain amount of charge is placed on a conductor. What ultimately happens to the charge? Where does it go?*

There are sophisticated ways to solve this problem, but a beginning physics student can often determine the solution in the following way.

1. We know that even a small amount of charge in the macroscopic sense involves a huge number of particles. Despite this, we **simplify** the problem to two charged particles.
2. Using the fact that like charges repel and other elementary knowledge, we **solve** the problem for two charged particles. We conclude that the particles separate until they reach the surface of the conductor. If charge cannot leave the conductor (no sparks) the answer to the simplified problem is that the charge moves to the surface and stays there.

3. We **conjecture** that the solution to the simplified problem is essentially the same as the solution to the original problem.
4. The previous step produces a result but we are, perhaps, uncomfortable with it. Did we oversimplify? Just to check, we ask ourselves if it would make any difference if we had three particles or four. Solving these related problems we discover that the result was the same. With this amount of **verification** we feel that the result very probably applies even to a large number of particles.

The above example is an informal thought experiment. It involved the following four steps: *simplification*, *solution*, *conjecture*, and *verification*. Since the verification method employed in this case was not rigorous it is termed *heuristic* verification.

We have formalized this method and use it for qualitative physics problem solving. It has been implemented in Prolog in a system called TEPS [Hibler 1992; Hibler & Biswas 1992b; Hibler & Biswas 1989]. TEPS stands for *Thought Experiment Problem Solver*.

In this paper we analyze the operation of a thought experiment problem solver. We will construct some elementary probabilistic models of the thought experiment problem solving process and determine some of its characteristics.

First, we describe the thought experiment methodology in more detail.

Next, we consider a thought experiment for which there is one right answer and  $n$  possible wrong answers. The thought experiment is verified using heuristic verification, which means that a total of  $s$  different simulations agree. We analyze the probability,  $P[C | A]$ , that the answer is correct given that all  $s$  simulations agree. This might be termed the validity of the heuristic verification. This analysis allows us to provide a condition for heuristic verification to work, to discuss the expected behavior of heuristic verification, and to consider the effect of fine versus coarse descriptions.

Finally, we analyze thought experiments in terms of the expected time,  $T_E$ , for achieving a verified result. The important quantity in this analysis is the effec-

tiveness,  $E_i$ , of each individual thought experiment. A theorem is proved which indicates the order in which thought experiments should be performed based on effectiveness. Estimation of effectiveness is discussed.

## Thought Experiments

The purpose of this section is to describe the steps in the thought experiment methodology in a brief, straightforward way. The thought experiment methodology has been described in a more formal, mathematical manner elsewhere [Hibler & Biswas 1989].

The thought experiment problem solver (TEPS) which we have implemented uses a qualitative reasoning system based on Forbus Qualitative Process Theory [Forbus 1984; Hibler & Biswas 1992a]; however, thought experiment methodology is very flexible. The purpose of this paper is to analyze this methodology and not any particular implementation. We will mention some other possibilities for implementation as we discuss the steps of the thought experiment process.

## Simplification

The first step in a thought experiment involves simplification. Simplifications are problem transformation rules provided to the problem solver by the system designer. Some simplifications rules are domain specific; others are domain independent. Practical simplification strategies and a partial catalog of useful simplifications are presented in [Hibler 1992].

As an example of a simplification, the thought experiment mentioned in the introduction used *Population Reduction*. The Population Reduction simplification maps any problem specification with multiple identical objects to a problem specification with only two objects. This state is simpler because there are fewer variables involved, and it is computationally much easier to determine the time evolution of the behavior of the system.

Another example of a simplification is the *Simple Stereotype* method. Simple stereotypes are easily built into modules describing objects or processes. They represent simpler versions of the object or process. Other examples of simplifications which have been implemented in TEPS include *Monte Carlo*, *Combined Change*, *Variable Blocking* and *Superposition*. Monte Carlo simplification is based on randomly sampling the state graph. Combined Change samples the state graph based on the way variables change. Variable blocking is a method for ignoring variables thought to be irrelevant. Superposition combines results from separate subproblems but has been implemented only for special cases. Some other possible simplifications which have not been implemented are discussed in [Hibler 1992].

Many abstraction techniques developed by others could be considered as simplifications in our sense. If heuristic verification is used with these techniques they might be applied in cases where the validity

of the abstraction is questionable. Examples include Ontological Perspective [Falkenhainer & Forbus 1990], Structural Consolidation [Weld & Addanki; Falkenhainer & Forbus 1990], Temporal Abstraction [Kuipers 1987], and Aggregation [Weld 1986]. The Exaggeration method of Weld [Weld 1988] can definitely be considered a simplification in our sense.

## Solution

The next step in a thought experiment involves "solving" the simplified model. This means that the problem solver must contain a reasoning engine which takes a problem specification and reasons about it to produce some "results".

**The TEPS Reasoning Engine** TEPS takes as input a specification of a qualitative state of a physical system. A qualitative state is a collection of qualitative values, one for each of the variables pertaining to the system. Qualitative values consist either of special, qualitatively significant landmark values or of intervals between two adjacent landmark values [Forbus 1984].

The dynamical behavior of the system is described by direct or indirect influences which can cause changes of qualitative state. They correspond roughly to qualitative versions of differential equations and qualitative versions of functional relationships.

Given a problem specification in TEPS we can simulate the time evolution of the system by generating a graph of qualitative states which can be reached from the original state. This is known as a reachable environment.

**Other Reasoning Engines** Other reasoning engines could be used as the basis for a thought experiment problem solver. Examples include not only other qualitative reasoning systems such as the of Kuipers [Kuipers, 1986] or de Kleer and Brown [de Kleer & Brown 1984], but also numerical simulation systems.

For a numerical reasoning system the input would consist of the initial state and the dynamical equations. The result would consist of the entire numerical simulation of the system's trajectory in phase space.

**Description of Results** The thought experiment problem solver is designed to answer specific equations about a physical system given an initial state for that system. We are thus not concerned about the output of the reasoning system directly because it usually does not constitute an answer to a question. What does constitute a possible answer is specified by some description function,  $D$ .  $D$  is sometimes called a description basis. We will assume that any query only has a finite number of possible answers.  $D$  can be thought of as a function which classifies the results produced by the reasoning engine into one of a finite number of categories, one for each possible answer. Thus even if our reasoning engine uses numerical simulation the description of the results is qualitative.

In the example involving charge placed in a conductor the description function takes the output of the problem solver and categorizes it by giving a list of regions which contain a nonzero amount of charge in the final state.

TEPS contains a library of description functions; however, a description function can also be input for a particular problem.

### Conjecture

A conjecture is a guess about the description of the result of the original problem based on the solution obtained on the simplified version of the problem. We will assume in this paper that the conjecture is always that the description function classifies the result of the simplified problem the same way that it would have classified the result of the original problem. In other words, the same description is produced.

### Verification

Verification can be rigorous or heuristic. It could even be empirical. Rigorous verification requires establishing a formal proof that the conjecture is true. This is usually difficult. Empirical verification consists of comparing the predictions with what actually occurs in the real world. Often, this approach is not practical. The most common type of verification is heuristic. With this type of verification other simplifications are tried, and the resulting conjectures are compared with the original conjecture. If they agree, we accept the conjecture as a reasonable belief.

### The Validity of Thought Experiments

The first concern of our analysis is the validity of thought experiments. Do they, in fact, give the correct answer? Our certainty in any particular case will depend on the verification method used. If exact verification is available then we can be sure whether the result is correct. More has been said about exact verification elsewhere [Hibler 1992]. Many uses of thought experiments involve heuristic verification. Heuristic verification seems intuitively reasonable; however, we need to clarify some of the ideas behind it. In order to do this, we explore a probabilistic model of heuristic verification, and determine the probability that a thought experiment is correct given that it is heuristically verified. From this we determine the actual requirement for heuristic verification to be useful. We also examine the effect of independent versus dependent confirmation and of fine versus coarse descriptions on validity.

### Probabilistic Models

To create a simple model of the thought experiment process we make several assumptions.

A probabilistic model requires the definition of a problem space. Let  $\mathbf{S}$  be the set of all problems which the problem solver with its particular library of processes will accept. We will assume that we sample

the problems using some fixed probability distribution. The probability distribution we use is left implicit and is not specified in the notation. The key assumption is that the probability distribution is fixed. Based on this sampling, we can discuss the relative frequency with which events which are functions of specific problems occur and associate probabilities with these events. In a like manner we can define probabilities on any subset of  $\mathbf{S}$ .

Consider any simplification method together with its associated conjecture and verification methods. When these methods are applied to a randomly selected problem from the problem space we obtain a thought experiment which we can characterize probabilistically.

A key assumption states that solving a problem consists of performing separate, independent thought experiments until one is verified. This *independent thought experiment assumption* implies that we are ignoring the use of complementary models and inheritance in this analysis. Since the use of additional information by inheritance in general helps the performance of the problem solver it is safe to say that the simple analysis provides a conservative estimate of complexity of the problem solving task, and, in most cases, the actual results will be better.

### Correctness Probability

Assume that the thought experiment process uses  $s$  simulations whose results are compared. Our description of the results classifies those results into  $n + 1$  possible categories. One of those categories is correct. In other words, if we had applied the classification to a full simulation of the original problem the result would have been in that category. The other  $n$  categories are wrong. We use small letters  $c$  and  $w_1 \dots w_n$  to denote the correct answer and the  $n$  wrong answers. Let  $C$  denote the event that all  $s$  simulations have the correct answer. Let  $A$  denote the event that all  $s$  simulations have the same answer. Let  $W$  denote the event that all  $s$  simulations have a wrong answer. We indicate intersections of the above events by writing the letters together.

The thought experiment is heuristically verified if all  $s$  simulations have the same answer. Thus the probability of interest is  $P[C | A]$ , the probability that the answer is correct given that all simulations agree. We know that

$$P[C | A] = P[C]/P[A]$$

If the simulations agree they must obviously agree and be correct or agree and be wrong. These possibilities are disjoint. Thus  $P[A] = P[AC] + P[AW]$ ; but  $P[AC] = P[C]$  so we can rewrite  $P[C | A]$  as

$$P[C|A] = 1/(1 + R) \quad (1)$$

where

$$R = P[AW]/P[C]. \quad (2)$$

The smaller  $R$  is, the better the accuracy of our thought experiment; the larger  $R$  is, the worse the accuracy.

To obtain more insight we analyze the components of  $R$ . Let  $a$  be any answer, either the correct answer  $c$  or any of the  $w_1 \dots w_n$  wrong answers. The probability of getting answer  $a$  in all  $s$  simulations is

$$P_1[a | (0)a] P_2[a | (1)a] P_3[a | (2)a] \dots P_s[a | (s-1)a]$$

The notation  $P_i[a | (j)a]$  denotes the probability that we obtain result  $a$  on simulation  $i$ , having had  $(j)$   $a$ 's on all the previous  $j$  simulations. We call this the probability for a confirmation of answer  $a$ .  $P[a | (0)a]$  is just  $P[a]$ . Using this formula we obtain

$$P[C] = P_1[c | (0)c] \dots P_s[c | (s-1)c] \quad (3)$$

$$\begin{aligned} P[AW] &= P_1[w_1 | (0)w_1] \dots P_s[w_1 | (s-1)w_1] \quad (4) \\ &+ P_1[w_2 | (0)w_2] \dots P_s[w_2 | (s-1)w_2] \\ &\vdots \\ &+ P_1[w_n | (0)w_n] \dots P_s[w_n | (s-1)w_n]. \end{aligned}$$

Using equations 3 and 4 we can rewrite equation 2 as a sum of products of ratios of probabilities. It is convenient to write 2 as

$$R = r_1^s + r_2^s + \dots + r_n^s \quad (5)$$

where  $r_k$  is the geometric mean of  $P_i[w_k | (i-1)w_k]/P_i[c | (i-1)c]$  over the different simulations, i.e.

$$r_k = ((P_1[w_k | (0)w_k]/P_1[c | (0)c]) \dots (P_s[w_k | (s-1)w_k]/P_s[c | (s-1)c]))^{1/s} \quad (6)$$

We must be careful to note that  $r_k$  depends on  $s$ .

Next, let us assume  $r \geq r_k$  so there exists an upper bound on the  $r_k$  ratios for all  $s$ . In that case  $R \leq nr^s$  so equation 1 becomes

$$P[C | A] \geq 1/(1 + nr^s) \quad (7)$$

This equation is sufficiently important that we reiterate what the quantities mean in words. A thought experiment is performed for which there is 1 right answer and  $n$  possible wrong answers. The thought experiment is verified using heuristic verification, which means that a total of  $s$  different simulations agree.  $P[C | A]$  is the probability that the answer is correct given that all  $s$  simulations agree.  $r$  is a type of bound on the probability of obtaining any single wrong answer versus the probability of obtaining the right answer.

This model demonstrates certain basic points about thought experiments which we discuss below.

### Heuristic Verification Requirement

The conditions for equation 7 to hold are so important that we express them as the heuristic verification requirement:

*Heuristic Verification Requirement:*

A sufficient requirement for the heuristic verification method to work is that on the average over all simulations the probability for a confirmation of the correct answer be greater than the probability for a confirmation of any single wrong answer by some fixed amount.

(The average mentioned is the geometric mean.)

If we assume that the heuristic verification requirement is satisfied then equation 7 holds and  $r < 1$ . This implies that the probability that the thought experiment is correct is bounded from below by a monotonically increasing function of the number of successful verifications. In fact, given enough verifications  $P[C | A]$  will be arbitrarily close to 1.

On the other hand if even one of the geometric means in equation 5 is bounded from below by a quantity greater than one, then for large enough  $s$ , additional verifications make  $P[C | A]$  worse and not better. If some of the  $r_k$  are one and any others have an upper bound less than one then  $P[C | A]$  eventually stabilizes at a value beyond which no improvement is possible with additional verifications.

### Dependent Versus Independent Confirmation

The confirmation probabilities  $P_i[a | (i-1)a]$  will reduce to  $P_i[a]$  if each simulation,  $i$ , used is independent of the previous  $(i-1)$  simulations. This is often not the case, however. The simulations used in heuristic verification are often simulations for models produced by less extreme versions of the same simplification method. The production of an answer,  $a$ , using one version might be correlated with the production of  $a$  by another version. This is why the confirming  $P_i$  must be expressed in terms of conditional probabilities.

If heuristic verification uses less extreme versions of the same simplification method to verify answers then how might the conditional probabilities for the right and wrong answers behave? First, it is very plausible to believe that  $P_i[c | (i-1)c]$  is very close to 1 if  $i > 1$ . In this case the preceding  $(i-1)$  simulations have produced correct answers. The  $i$ th simulation involves a more realistic (less simplified) model than the preceding ones and it is based on the same type of simplification which produced correct results in these cases. Thus it would be expected to produce a correct answer. Second,  $P_i[w_k | (i-1)w_k]$  should eventually decline as  $i$  increases simply because the models become more realistic as  $i$  increases. Unfortunately the correlation produced by using the same method might make this decline slow.

### Fine Versus Coarse Descriptions

A last consideration concerns the question of whether a coarse or a fine description is more reliable. The explicit factor of  $n$  in equation 7 suggests that the smaller the number of alternatives in our description

of results the larger  $P[C | A]$  is. This assumption is somewhat dangerous because  $r$  will depend on  $n$  also. For example, assume  $n = 3$  and each simulation has the same probabilities:  $P[c] = 3/9, P[w_1] = P[w_2] = P[w_3] = 2/9$ . If we halve the number of categories by combining  $c$  with  $w_1$  and  $w_2$  with  $w_3$  we obtain  $P[c'] = 5/9, P[w'_1] = 4/9$ , and  $n = 1$ . In the first case,  $r = P[w_i]/P[c] = 2/3$ ; in the second case  $r = P[w'_i]/P[c'] = 4/5$ . Any raising of  $r$  dominates the lowering of  $n$  if  $s$  is high enough. On the other hand, if  $s$  is small enough then coarser categories are better. For example, if  $s = 1$  then  $P[C | A]$  is just  $P[c]$  and when categories are combined it is always true that  $P[c'] \geq P[c]$  so coarser categories are better.

### Efficiency of Thought Experiments

The next issue after correctness of thought experiments is their efficiency compared to a direct solution of the problem. We analyze thought experiments in terms of the expected time,  $T_E$ , for achieving a verified result. From this we prove a theorem showing how to achieve maximum efficiency. We then derive upper bounds on the probability that a thought experiment fails to be verified and on the average time required for a thought experiment. We next describe how to estimate efficiency parameters and give a heuristic rule for maximizing efficiency. Finally, we discuss practical efficiency issues.

### Time Requirements

Let  $T_n$  be the total time required for the  $n$ th thought experiment. This time includes the time required for simplification, conjecture, and most importantly, the time required for verification. Verification may require a considerable amount of time since it usually involves solving at least one other simplified model. Let  $P_n$  be the probability that the  $n$ th thought experiment fails. We assume these probabilities are independent. Thus we assume thought experiments are not only functionally, but also statistically independent. With these assumptions, the expected time,  $T_E$ , required for a successful thought experiment is

$$T_E = T_1 + T_2P_1 + T_3P_1P_2 + T_4P_1P_2P_3 + \dots \quad (8)$$

Thus,  $T_E$  depends on the whole sequence of thought experiments which might be performed.

The probability,  $P_F$ , that the problem solver fails to obtain any verified solution is just

$$P_F = P_1P_2 \dots P_n, \quad (9)$$

where  $n$  is the number of possible thought experiments which may be performed.

Let us briefly consider the meaning of equations 8 and 9. If there were an infinite series of possible thought experiments  $n$  would approach infinity. If each of the  $P_i$  were bounded from above by a number less than 1 then  $P_F$  would approach zero. In this case, we would expect the problem solver to always obtain

a verified solution. Since we do not actually have an infinite series the problem solver may fail.  $T_E$  represents the average time taken to either obtain a verified solution or to fail.

### Thought Experiment Ordering

Our first application of equation 8 for  $T_E$  is to prove the thought experiment ordering theorem.

Let the time required to solve the problem by direct qualitative simulation be  $S_o$ ; we define the effectiveness,  $E_i$ , of a thought experiment to be  $E_i = (S_o/T_i)(1 - P_i)$ . The significance of this definition will be seen later.

#### Thought Experiment Ordering Theorem

*A sequence of independent thought experiments will have a minimum expected time for achieving a verified result if the sequence is performed in order of effectiveness from most effective to least effective.*

Proof:

Consider thought experiment  $i$  and  $i + 1$  which are adjacent in the sequence for  $T_E$  used in equation 8. If we interchange the order in which these experiments are performed the only terms in the series which change are the terms involving  $T_i$  and  $T_{i+1}$ .

$$T_E(\text{original}) = T_i(P_1 \dots P_{i-1}) + T_{i+1}(P_1 \dots P_{i-1})P_i + R$$

$$T_E(\text{exchanged}) = T_{i+1}(P_1 \dots P_{i-1}) + T_i(P_1 \dots P_{i-1})P_{i+1} + R$$

By algebraic manipulation we discover that  $T_E(\text{exchanged}) < T_E(\text{original})$  if and only if  $(1/T_{i+1})(1 - P_{i+1}) > (1/T_i)(1 - P_i)$ . Multiplying by  $S_o$  we obtain the following exchange lemma:

*Exchange lemma:  $T_E(\text{exchanged}) < T_E(\text{original})$  if and only if  $E_{i+1} > E_i$ .*

We prove the ordering theorem by contradiction. If an optimal sequence were not in the order given by the theorem then by the exchange lemma we could improve the sequence. This would contradict the assumption that the sequence was optimal. *Q.E.D.*

### Upper Bounds

**Probability of Failure** First, let us assume that we employ only simplification methods which are useful. If we have a finite number of simplifications, we have a finite number of  $P_i$  representing probability of failure. One of these  $P_i$  has a maximum value; call it  $P$ . Any simplification method which produces thought experiments which always tend to fail verification ( $P_i = 1$ ) is dropped from a problem solver as useless. Therefore,  $P$  is an upper bound on the probability of failure of any simplification and this upper bound is less than one. We call this the *usefulness assumption*, equations 10, 11.

$$P_i \leq P \quad (10)$$

$$P < 1 \quad (11)$$

The usefulness assumption provides an upper bound on the probability of failure,  $P_F$ , of a thought experiment given in equation 9.

$$P_F \leq P^n \quad (12)$$

As pointed out earlier, assumption of 10 and 11 guarantees that as the number of thought experiments increases, the probability of failure,  $P_F$ , may be made as small as we please.

**Average Time** Next, let us determine an upper bound for  $T_E$ . This will depend on a bound for the  $T_i$  as well as a bound for the  $P_i$ . We assume that for all  $i$ ,

$$T_i \leq T. \quad (13)$$

There might be some thought experiments which could go on forever without reaching a conclusion. This would be due to the fact that the qualitative simulation did not terminate. In practice any thought experiment which goes on too long can be terminated and verification considered to have failed. Thus assumption 13 is, in fact, acceptable.

Given equations 10, 13 and the fact that all quantities are positive, an upper bound for  $T_E$  is  $T_E \leq T + TP + TP^2 + \dots$ . This is not an infinite series, but it is approximated by an infinite series if many different simplifications are possible. Furthermore, since all terms are positive the infinite series is certainly an upper bound. We have an ordinary geometric series and since  $P < 1$  it converges. Thus

$$T_E \leq T/(1 - P). \quad (14)$$

The improvement ratio over straight simulation is  $S_o/T_E$ . Using 14 we have a lower bound on this of  $S_o/T_E \geq (S_o/T)(1 - P)$ . This bound is just an effectiveness calculated using  $T$ , and  $P$ ; thus, it provides us with an interpretation of the effectiveness of a thought experiment. The significance of the effectiveness,  $E_i$ , of thought experiment  $i$  is that if every thought experiment,  $k$ , is no worse than the given experiment,  $i$ , ( $S_o/T_k \geq S_o/T_i$ , and  $P_k \leq P_i$ ) then the thought experiment problem solver is faster than straight simulation by a factor of at least  $E_i$ .

### Estimation of Parameters

Can we obtain estimates for parameters such as  $P_i$  and  $T_i$  which characterize thought experiments? This is one of the most important issues for practical applications. One approach is to simply assume various values for these parameters for the sake of theoretical analysis. Another approach is to attempt to make empirical estimates for them even if they are crude.

In order to make empirical estimates, we must distinguish between specific thought experiments which

are attempts to solve unique problems and the simplification methods which are used in the thought experiments. Any specific thought experiment either succeeds or it doesn't. If we repeat it we always get the same result. A simplification method on the other hand, may produce a verifiable result if used in one thought experiment but not in another. If we have enough experience with a thought experiment problem solver we can collect rough statistics to indicate the frequency with which a given simplification method is useful in giving verifiable results. We can also estimate the time improvement ratio  $S_o/T_i$  for thought experiments using a given simplification. The estimates are crude because the success of a simplification method may depend on the type of problem, i.e., the characteristics of the individual problem space. If the problem solver has not encountered a similar type of problem before, the frequency based estimate may not be very reliable. Another reason for the estimates being crude is that previously encountered problems may not constitute a random or sufficiently large sample of the problem space.

Next, let us make some estimates of  $T_i$  in terms of more basic quantities. A thought experiment involves a simplification and a generalization step. The simplification step involves finding a simplified version of the problem and performing a qualitative simulation on that simplified version. Finding a simplified version takes a constant amount of time which is small compared to the time required to perform the qualitative simulation. The generalization step involves making a conjecture and verifying that conjecture. Making a conjecture takes a constant amount of time which is small compared to the time required for a qualitative simulation. Ignoring these small quantities  $T_i = S_i + V_i$ .  $S_i$  is the time required for qualitative simulation, and  $V_i$  is the time required for verification.

To make our estimate for  $T_i$  more useful we must make some rough estimates concerning verification. Verification usually involves performing at least one additional qualitative simulation and comparing the results to the previous simulation. Making the simplification and comparing the results would take a small amount of time compared with  $S_i$ . Our estimate for  $T_i$  becomes  $T_i = nS_i'$  where  $n$  is the number of qualitative simulations performed and  $S_i'$  is the average time taken for each. A reasonable estimate for  $n$  is 2. The results of the original simulation are checked by comparing with one additional simulation. In some cases  $n$  could be 1. This would mean that the result would be compared with the results from previous failed thought experiments for the same problem. It would also make analysis more difficult as the thought experiments would no longer be independent. Whether a thought experiment succeeded or failed would depend on the ordering of thought experiments. In some cases the simplification method might specify an  $n$  greater than 2 but these are probably rare. Considering both

these effects an estimate of 2 seems reasonable. We will assume  $T_i = S'_i$ ; the analysis would not be greatly different if 3 or some other small number were chosen.

The effectiveness of a thought experiment becomes  $E_i = (S_o/S'_i)(1 - P_i)/2$ .  $S_o/S'_i$  can be considered the average amount of simplification achieved by the simplification method used in the  $i$ th thought experiment. It represents an average time improvement factor in using the two simplified simulations in the thought experiment versus simulating the original problem. This parameter is important because we can often rank simplification methods by  $(S_o/S'_i)$ . Examination of two simplification methods will often indicate which is more extreme and should therefore yield a larger value of  $(S_o/S'_i)$ . On the other hand, knowledge required to estimate  $P_i$  may be more difficult to obtain. In this situation, the thought experiment ordering theorem suggests the following heuristic ordering rule:

#### Heuristic Ordering Rule:

*If independent simplification methods can be ranked by degree of simplification, but no information is available about verification probabilities then a thought experiment problem solver should try the simplification methods in order of degree of simplification from strongest to weakest.*

If  $S$  is an upper bound on the time any of the qualitative simulations might take then we can estimate equation 14 by:

$$T_E < 2S/(1 - P). \quad (15)$$

We want the thought experiment method to take less time than solving the original problem directly by qualitative simulation. If qualitative simulation of the original problem takes time  $S_o$ , we require  $S_o/T_E > 1$ . This means that we would like our upper bound to be such that  $(S_o/S)(1 - P)/2 > 1$ ; perhaps much greater than 1. Since  $S$  is an upper bound on the  $S_i$ ,  $S_o/S$  is a lower bound on the amount of simplification used in any individual thought experiment.

#### Practicality

Are thought experiments practical? Unless absolute verification is available even a successful thought experiment provides only a reasonable conclusion and not a certain one. Thus, this method will be used only when direct simulation is not feasible, usually because it would take too long. This is, in fact, the case with most real world problems. In order for the thought experiment problem solver to be worthwhile we must be able to find simplifications which have an effectiveness which is greater than one, preferably much greater than one. In order to achieve this we need an extreme degree of simplification with reasonable probability of verification. For example, if probability of verification is at least  $1/5$  then  $P$  is  $4/5$  and we need a degree of simplification  $(S_o/S) > 10$ . Experience with TEPS is too limited to take any estimates very seriously;

this is a possibility for future research. With TEPS so far, the values for  $P$  are normally less than one half. It seems likely that simplifications would be dropped from a problem solver if they fail verification in the great majority of cases.

It might be argued that we need exhaustive statistical testing to determine that the effectiveness for a given simplification is adequately high to be of use. This is not true if the degree of simplification provided by the simplification method is high enough. In that case, demonstration of even a few successful verifications makes it reasonable to believe that  $E_i$  is large enough. For example, if  $(S_o/S_i) > 100$  then  $E_i > 1$  if  $(1 - P_i) > 0.02$ ; if  $(S_o/S_i) > 1000$  then  $E_i > 1$  if  $(1 - P_i) > .002$ ; if  $(S_o/S_i) > 1,000,000$  then  $E_i > 1$  if  $(1 - P_i) > .000002$ . Since  $(1 - P_i)$  represents probability of successful verification even a very few successful examples indicate that the effectiveness is high enough if the degree of simplification is really large.

#### Conclusions

We have provided a preliminary theoretical framework for the thought experiment methodology. This framework combined with the empirical experience with TEPS suggests that this is a viable technique. It does not replace conventional methods of qualitative reasoning but rather augments them. Further development of this technique seems desirable.

There are many possibilities for future development. Analysis of known simplification techniques and development of new ones is an important area of research. Theoretical analyses of simplifications should be attempted if possible and statistics on practical effects should be obtained. On a practical side, it would be useful to explore thought experiment problem solvers as a basis for tutoring systems. The simplified model automatically generated by the problem solver could be a basis for helping students understand the original problem.

#### References

- Falkenhainer B. and Forbus K. D. 1990. Compositional modeling of physical systems. In *4th International Workshop on Qualitative Physics*.
- Forbus K. D. 1984. Qualitative process theory. *Artificial Intelligence*, 24:85-168.
- Hibler D. L. 1992. *The Thought Experiment Method: A New Approach to Qualitative Reasoning*. PhD thesis, The University of South Carolina.
- Hibler D. L. and Biswas G. 1992a. Teps: applying the thought experiment methodology to qualitative problem solving. In B. Faltings and P. Struss, editors, *Recent Advances in Qualitative Physics*, pages 345-360, MIT Press.
- Hibler D. L. and Biswas G. 1992b. Thought experiments as a framework for multi-level reasoning. In *Working Papers of the Sixth International Workshop*

*on Qualitative Reasoning about Physical Systems*, August.

Hibler D. L. and Biswas G. 1989. The thought experiment approach to qualitative physics. In *IJCAI-89*, pages 1279-1284.

de Kleer J. and Brown J. S. 1984. A qualitative physics based on confluences. *Artificial Intelligence*, 24:7-83.

Kuipers B. 1987. Abstraction by time-scale in qualitative simulation. In *Proceedings of AAAI-87*, pages 621-625.

Kuipers B. 1986. Qualitative simulation. *Artificial Intelligence*, 29:289-388, 1986.

Prigogine I. and Stengers I. 1984. *Order Out of Chaos*. Bantam Books.

Weld D. S. 1988. Exaggeration. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 291-295, Minneapolis, MN, August.

Weld D. S. 1986. The use of aggregation in causal simulation. *Artificial Intelligence*.

Weld D. S. and Addanki S. 1990. Task-driven model abstraction. In *Fourth International Workshop on Qualitative Physics*.

# QCMF: a tool for generating qualitative models from compartmental structures

L. Ironi<sup>(1)</sup> and M. Stefanelli<sup>(2)</sup>

<sup>(1)</sup> Istituto di Analisi Numerica del C.N.R., Pavia, Italy

<sup>(2)</sup> Dipartimento di Informatica e Sistemistica dell'Università di Pavia, Pavia, Italy  
liliana@supers1.ian.pv.cnr.it

## Abstract

This paper describes a framework, called QCMF (Qualitative Compartmental Modeling Framework), which assists the user in formulating models of a physical system and in analyzing their behaviors through the simulation of the effects of a variety of different perturbations of the system. QCMF has adopted the compartmental theory as modeling ontology: a system is represented as a finite set of interacting compartments. The user enters, through an iconic language and menus, the compartmental structure of a physical system and defines the kinds of functional relationships describing the interactions between compartments. Then, QCMF automatically generates a behavior model of the system. Such a model consists of a set of ordinary differential equations, which are currently qualitatively expressed, and is directly coded into the language (QSIM) which is interpreted by the simulation algorithm. The system behavior can be obtained by simulating the model starting from an initial state which describes the perturbations acting on the system. The code defining the initial state is automatically built by QCMF as well. Finally, explanations of the predicted behavior are also automatically generated. Moreover, QCMF is capable to build and to maintain a library of models for a given system.

**Keywords:** Mathematical modeling, qualitative modeling, qualitative simulation, compartmental theory, graphical interface.

## Introduction

System dynamics modeling is a powerful tool for reasoning about physical systems. Model-based reasoning goes through different stages: first, model formulation and generation, then prediction and explanation of the system behavior. Model formulation, that is the definition of the relations between model variables, requires the expert's domain knowledge as well as a large

body of domain independent knowledge. This includes mathematical concepts and fundamental principles of physics. Moreover, in order to be computationally tractable, the formulated model needs to be manipulated so that it can be represented through the language required by the adopted simulation algorithm. This part of the modeling process, called model generation, is strictly domain independent and has been proved to represent a heavy and time-consuming activity. In fact, although today available simulation languages are more user-friendly than standard programming languages, they still require precise syntax and semantics. A good knowledge of the simulation algorithm is also necessary for interpreting the simulation results. Hence, the need derives for building tools that have the necessary methodological knowledge for assisting the user in the model formulation phase, and for automatically executing the generation, simulation and explanation steps.

This paper describes a framework, called QCMF (Qualitative Compartmental Modeling Framework), for assisting the user in the process of model building (i.e., model formulation and generation) and simulation. The chosen modeling ontology is the compartmental one: a system is viewed as a finite set of *compartments* which interact by exchanging or transforming material. Flows through the system are represented as transfers from one compartment to another. Compartmental modeling can be properly exploited for the formal description of the dynamics of chemical reactions and material transfer processes. Although its most common application remains bio-medical science, its use has spread to other domains such as ecology, epidemiology, hydrology as well as chemical engineering.

QCMF includes knowledge of the theory of compartmental systems (Atkins, 1974), while the domain ontology, that is the decomposition of a specific system into compartments and the network of interactions between these, is entered by the user through an iconic language. A representation of the compartmental topology in a given system is called *structure model*, whereas the set of representations of the behaviors of each compartment, namely a set of Ordinary Differen-

\*A more complete version of this paper will appear in *Computer Methods and Programs in Biomedicine*

tial Equations (ODE) based on the mass balance law, defines a *behavior model* of the given structure. Within QCMF, a *system model* is represented by a behavior model associated with an initial state which describes the perturbations acting on a structure model.

Although our long-term goal aims at building a framework capable to execute the tasks of generating and simulating both quantitative and qualitative models, this paper deals only with qualitative models. Nevertheless, a quantitative formulation can be easily derived from the structure model by symbolically manipulating the acquired knowledge in order to simplify the process of writing the set of ODEs, and by mapping the behavior model into its corresponding numerical one according to the selected numerical simulation algorithm. Similar facilities are now provided by the most recent release of SAAM (SAAM, 1993), which could supplement QCMF with quantitative compartmental modelling tools.

QCMF embeds all the needed methodological knowledge, and therefore allows the user to go through the different modeling phases very easily:

1. A structure model is entered through a user-friendly graphical interface;
2. The behavior model is formulated by performing an automatic analysis of the given structure, and by acquiring information about functional relationships between variables;
3. The behavior model is automatically generated in the QSIM code (Kuipers, 1986);
4. The initial state corresponding to either single or multiple perturbations acting on the system is automatically generated in the QSIM code;
5. The system model is simulated in order to predict the effects caused by the considered perturbations;
6. A causal explanation of the simulated results, in addition to the graphical representation given by QSIM, is automatically generated.

The choice of QSIM among the several proposed approaches (Bobrow, 1984; Weld and De Kleer, 1990) to qualitative modeling is due to the fact that it seems to offer the most suitable formalism to represent the dynamics of processes whose behavior can be quantitatively described by ODEs (Kuipers and Kassirer, 1985; Kuipers, 1989).

Let us observe that each system can be described by several models, each of them providing a description at a different level of detail. This means that, according to the available knowledge and the purpose of model building, a system can be decomposed in different ways and different assumptions on the functional relationships between variables can be explored. Hence the need for building a library of models for a given system, where different entities composing models are represented through frames, and the whole set of frames is organized into a hierarchical structure.

The problem of automatically selecting within the library the appropriate model for a specific goal is crucial in model-based reasoning (Addanki *et al.*, 1991; Falkenhaier and Forbus, 1991; Iwasaki, 1992; Nayak *et al.*, 1991; Weld, 1990), and will be considered in the future work.

QCMF, which has a practical value as a stand-alone system, could also be viewed as an acquisition tool of knowledge represented through system models. The availability of a library of models allows a Knowledge-Based System (KBS) to reason exploiting deep knowledge, so enhancing its problem solving ability.

This paper is organized as follows: after a brief introduction to compartmental theory, the next sections deal with an overview of QCMF. The user-QCMF interaction in model formulation, generation and simulation phases is illustrated through an example taken from the medical domain. Finally, the major strengths and limits of this work are discussed.

## Compartmental modeling

A compartment is fundamentally an idealized store of a substance, homogeneously distributed: if a substance is present in a physical system in several forms or locations and passes from one form or location to another, then each form or location constitutes a separate compartment for that substance. It should be noted that different compartments can coexist in the same location of a system and that the compartments do not always correspond to physically identifiable components.

Such modeling technique is applicable when the study concerns the flow of some substance through the system. In such models, the flow through the system is represented as transfers from one compartment to another. The *structure model*, that is the system's decomposition and the network of interactions between compartments, is usually displayed by means of a compartmental diagram. Decomposing a system into compartments leads directly to its *behavior*, namely a set of equations based on the mass balance law. The state variables of the system, which are denoted by  $x_i(t)$ , represent the concentration or amount of substance in the  $i$ -th compartment which exchanges matter with other compartments or the external environment at time  $t$ . The rate of change of  $x_i$  is obtained by the equation:

$$\dot{x}_i = f_{i0} + \sum_{\substack{j=1 \\ j \neq i}}^n f_{ij}(x_j) - \sum_{\substack{j=1 \\ j \neq i}}^n f_{ji}(x_i) - f_{0i}(x_i) \quad (1)$$

where  $\dot{x}_i$  denotes the time derivative of  $x_i$ ;  $f_{ij}$  and  $f_{ji}$  denote flow variables, that is the rates of mass transfer into the  $i$ -th compartment from the  $j$ -th compartment and into the  $j$ -th compartment from the  $i$ -th compartment, respectively. (The compartment 0 denotes the external environment). In general, the flow of material depends on the quantity or concentration

of material in the source compartment and may also be controlled by the quantity or concentration in some other compartments, that is:

$$f_{ij} = f_{ij}(x_j; x_l, x_m, \dots)$$

where  $x_j$ ,  $x_l$ ,  $x_m, \dots$  denote the state variable of the source compartment and the variables which control  $f_{ij}$ , respectively. The dependency of a flow variable on state variables can be either linear or non-linear. Both state and flow variables are always non-negative.

In order to formulate the behavior model, the functional dependencies of each flow  $f_{ij}$  on  $x_j$  in (1) must be expressed. The nature of these functional dependencies will vary according to the system under investigation and will be suggested either by observing the real system behavior or by the available knowledge.

### Overview of QCMF

QCMF is a tool which integrates the steps necessary to reason about a system behavior: model building, model simulation, and results explanation. The interest for automating such processes has recently arisen, and a number of methods has been proposed by several researchers (e.g. (Addanki *et al.*, 1991; Falkenhaier and Forbus, 1991; Iwasaki, 1992; Nayak *et al.*, 1991; Weld, 1990; Capelo *et al.*, 1993; Crawford *et al.*, 1992; Low and Iwasaki, 1992)).

Fig. 1 gives an overview of QCMF's modules. It is menu-driven and its major components are the GENERATE, SIMULATE, EXPLAIN and BROWSE modules.

The GENERATE module helps (1) the user in entering both the compartmental structure and the assumptions about the functional relationships between variables; (2) then, it automatically analyzes the structure in order to (3) write down the model equations into the QSIM language.

The so generated qualitative model is given as input to the SIMULATE module. This module (1) manages the information entered by the user about the perturbations acting on the system; (2) it updates, if needed, the behavior model; (3) it automatically generates the initial state in the QSIM language; (4) it predicts, by simulating the model, the possible behaviors of the system which are presented to the user in a graphical form. The analysis of such an outcome can suggest a model revision of either the structure and behavior model or the initial state.

The user is allowed to request a causal explanation of the predicted behaviors. The EXPLAIN module has access to all the information about the system model and to the predicted behavior to be explained in order to generate a causal chain describing the system transition from one state to another. To this end, it exploits: (1) the structure model as a causal graph where the effects of a change propagate following the paths in the model topology, (2) the behavior model to detect the 'primary' causes of the change, and (3) knowledge of the QSIM algorithm, namely the transition rules, to

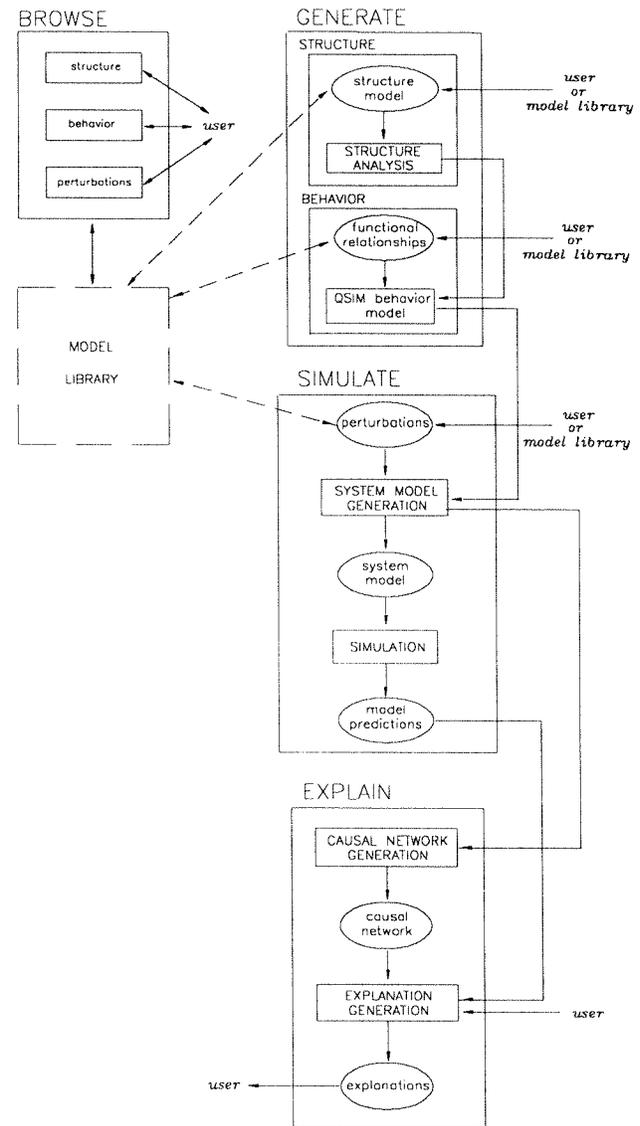


Figure 1: Overview of QCMF's modules. Information entered by the user are explicitly indicated by arrows. Rectangles indicate tasks automatically executed by the system

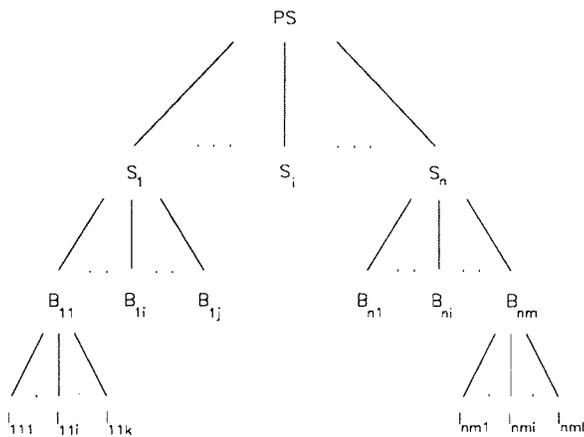


Figure 2: The hierarchical structure of the library of models

identify which changes occur from one system state to its successor.

Moreover, the GENERATE and SIMULATE modules organize and maintain a library of models which can be efficiently retrieved and used by a KBS for a given goal. Each model  $M$  can be identified with the tuple  $(PS, S, B, I)$ , where  $PS$  denotes a physical system which has been modeled by specifying for each considered structure model ( $S$ ) different behavior models ( $B$ ), each one requiring the definition of the considered perturbations ( $I$ ). All these entities are represented by frames which are hierarchically organized in a tree structure as shown in Fig. 2. Any branch of the tree represents a different model  $M$  of  $PS$ . It may represent either a different level of abstraction (i.e., different structure model), or a different theory (i.e., different behavior model), or different working conditions (i.e., different perturbation). Thus, each piece of knowledge entered by the user during her interaction with QCMF fills a slot of a suitable frame of the model components hierarchy shown in Fig. 2. This view makes clear which different models have been built for representing the available knowledge. This organization is useful on one side to QCMF to correctly access the model library, and on the other side to the user to easily examine the assumptions made in building the models. The latter of these operations is performed by the BROWSE module, which provides facilities to visualize to the user all the information stored in the library.

### Model formulation and generation

Two different sub-modules, which can be selected through a sub-menu associated with the GENERATE button, deal separately with the structure and behavior model.

### Structure model

The structure model consists of information about the decomposition of the system into compartments, and about the network of interactions between compartments and external environment.

A user-friendly graphical interface allows the user to enter the definition of the structure model and/or to edit a previously defined structure. Each object of the structure, i.e. compartments, flows, controls, must be defined by the user according to the available knowledge. Each object is represented through an icon, and is entered by the user by clicking the corresponding button. According to the selected object, further information must be entered by the user. Namely, it is required to specify:

- for a compartment, the name of the substance it contains, and the position on the input/output graphical window;
- for a flow, the connections between compartments and the exchanges with the environment, i.e. which compartment the flow is leaving from to reach another compartment or the environment, and which compartment the flow is entering from another compartment or from the environment;
- for a control, the controlling variable and the controlled flow.

Each compartment is automatically labeled with a number, and the flows are automatically labeled taking into account their direction. This facilitates the model formulation process since the direction of each flow can be easily identified, as well as the model variables. Thus, writing the equations in the behavior model generation phase is straightforward. All input operations are guided: they mostly occur through either menu and/or the selection of compartments.

The components of the structure are recorded as values of slots of the *structure frame* that the user can modify at any time during the generation phase. The values associated with the compartment, flow and control attributes/slots are respectively the list of the compartment number labels with their respective dubbed names, the list of the flows, and the list of the controlling compartment number labels with their respective controlled flows. Moreover, information about the system which the structure is related to and the list of its associated behavior models have to be recorded into slots of the structure frame in order to suitably locate the structure within the library of models. Therefore, the structure frame is described by five slots, namely physical-system, behaviors, compartments, flows and controls.

In order to draw a clear and nice view of the compartmental diagram, functions looking for the minimal connection path of two compartments, and avoiding the overlapping of objects have been implemented. Graphical editing facilities are available to the user in order to modify a structure: compartments, flows

and controls can be deleted. The system updates the structure frame and all related lists and the compartmental diagram, that is the compartments and flows are relabeled. The structure model can be stored, and then retrieved, through the frame and lists containing graphical information. However, the saving operation is performed only after an automatic debugging of the introduced structure ensures that it is an acceptable compartmental structure, i.e. all compartments are in some way connected among themselves.

## Behavior model

The formulation of the behavior model of a given structure requires to specify the variables, the functional relationships between variables, and then to write down the differential equation (1) for each compartment.

The identification of all the relevant variables is automatically performed by QCMF by exploiting the information represented in the structure: the number of compartments allows us to identify the state variables  $x_i$ , and their time derivative variables  $\dot{x}_i$ ; the flow slot permits the identification of the flow variables  $f_{ij}$ , and for each of them to state if it is a leaving or entering flow, and consequently its dependency on the variable  $x_j$ . The possible dependency of  $f_{ij}$  on other variables can be derived by reading the content of the control slot. In the case of controlled flows, the multi-valued functional dependencies between flow and state variables are expressed as sum or product of single-valued functions. These functions, which actually are auxiliary variables, are automatically dubbed and introduced into the list of variables. Then, the initial quantity spaces  $L_i$  must be specified for each variable. First, the sets  $L_i$  associated with the flow and state variables are instantiated to (0 INF) for the non-negativity property of these variables, whereas the sets  $L_i$  associated with the variables  $\dot{x}_i$  are instantiated to (MINF 0 INF), where MINF and INF in the QSIM notation mean  $-\infty$  and  $+\infty$ , respectively. As soon as further information about the system is provided by the user, the sets  $L_i$  are updated by inserting new landmarks values. The order relation of the new landmarks with respect to the previously defined ones is stated by the user, through menus, on the basis of her knowledge.

In addition to update the quantity space associated with each variable, and to translate into the QSIM language the relationships between variables, QCMF derives automatically for each compartment the differential equation which describes the dynamics of the contained substance. First, for each compartment, all entering and leaving flows must be identified so that the corresponding mass balance equation (1) can be written. Moreover, the analysis of the compartmental diagram allows us to identify particular features of the structure, and to write further equations describing global properties of the system (for example, the mass conservation law for closed structure). The translation into the QSIM qualitative constraints is straightfor-

ward in most cases. However, since QSIM constraint operators are relations and not functions, the syntax does not allow nesting of constraints to create complex expressions. In such cases, QCMF introduces a variable, which has no physical meaning, to represent intermediate results. In the following, the variables  $x_i$ ,  $f_{ij}$ ,  $\dot{x}_i$  are denoted by Xi, Fij, DXi, respectively, when they are instantiated by QCMF. The kind of relationship between variables is suggested by the specific domain knowledge: therefore, it is entered by the user who is helped in this operation by a menu. The allowed kinds of relationships allow the user to describe qualitatively a wide set of functional dependencies between variables in terms of their regions of monotonicity.

Information about the behavior model is stored into a frame, which is represented by five slots, namely physical-system, structure, perturbations, flows and controlled-flows. The first three slots store all the information needed for identifying, in the hierarchical tree structure of the model library, the branch which the behavior model belongs to, and its possible associated branches. Their values correspond, respectively, to the names of the physical system and the structure which the behavior is related to, and to the list of its associated perturbations. The flows and controlled-flows slots allow us to characterize a behavior model of an underlying structure, that is they give information about the assumptions made on the functional relationships between flows and state variables. The values given to these slots are the list of flows, or auxiliary variables defining flows, with their respective information about the assumed type of relationship, and the list of controlled flows with their associated kind of algebraic operator which allows us to define the multi-valued functional dependency of the flow on the state variables.

## Model simulation

The main task of the SIMULATE module deals with the automatic generation of the initial state of the perturbed system directly in the QSIM code, i.e. the automatic definition for each variable of its qualitative value (*qual*) and its direction of change (*qdir*). QCMF tries to build, starting from the values of the perturbed variables, the initial state of the subset of system variables which guarantees the generation of a single complete system state. QCMF builds the initial state of the variables  $x_i$  and  $\dot{x}_i$ . Knowing the state of such variables is sufficient when the flow variables are not controlled, otherwise the user may be asked to give further information, if any, about the state of the controlled flows which, being the result of algebraic operations between single valued functions, could be not univocally determined. Such a partial initial state together with the behavior model are given as input to QSIM, which determines its consistent completion by propagating the state values of the specified variables through the constraints. The basic steps in the design

and implementation of the algorithm generating the initial state are the following:

1. the user enters, through menus, the qualitative value of one or more perturbed variables;
2. QCMF automatically identifies, by analyzing the behavior model, the variables which are linked to the perturbed ones by functional relationships, and then indirectly interested by the perturbation, and determines their *quals*;
3. the sign of  $\dot{x}_i$  is calculated by exploiting the *qual* of the variables  $f_{ij}$  appearing in the equation which defines  $\dot{x}_i$ . This allows us to determine the *qdir* of  $x_i$ , and of all the flow variables linked by functional relationships to  $x_i$ . Multiple perturbations are allowed. Whenever, due to multiple perturbations, there is ambiguity in defining the sign of  $\dot{x}_i$ , the user is asked to state, according to the available knowledge, which is the prevailing perturbation.
4. the *qdirs* of the variables  $x_i$  are determined by considering the *qdirs* of the related flow variables. When a *qdir* can not be univocally determined, the user is asked to select one on the basis of her knowledge.

Once the initial state has been defined, the user can run a simulation by selecting the related option in the SIMULATE menu or simply save the generated code. Another task performed by the SAVE option is the updating of the behavior frame and its associated perturbation frame within the library of models. The *perturbation frame* is described by four slots, three of them identify the physical system, the structure and the behavior models the system perturbations refer to, and the last one records the perturbed variables and their associated perturbation.

The user is allowed to consider two different kinds of scenario in defining the initial state. In one case, the effects of perturbing a reference steady state of the system can be simulated; in the other one, a tracer kinetics can be analyzed. Although there is no conceptual difference in building the initial state, two sub-modules, which are selected through a sub-menu of the SIMULATE button, deal separately with the different scenarios. The main difference between the two sub-modules concerns the interaction between the user and QCMF.

### Perturbation of the steady state

System models should describe how one or multiple disturbances on a system cause the transition from its normal state to an abnormal one. Then, such models are obtained by perturbing the normal state supposed to be steady, i.e. by changing the qualitative state of some variables, and possibly by introducing either structural or behavioral changes in the model describing the physical process during the GENERATION phase. On the other hand, it is equally interesting the simulation of the time evolution of the system

from an abnormal steady state to a normal one, for example, in a medical context, caused by a therapy.

In both cases, the initial state is defined by both the reference steady state and the perturbed one. Therefore, the first task executed by this sub-module is the definition of the steady value of each variable. This operation is performed automatically by QCMF whenever the quantity space of the variable defined in the behavior model is  $(0 \infty)$ ; otherwise the user is asked to specify, through a menu, in which interval the steady value of a variable lies. New landmarks representing the steady values are added to their respective quantity spaces in the behavior model by exploiting the functional dependencies between variables, and new corresponding values are added to the corresponding constraints. Then, the user must choose, always through a menu, which variables are perturbed, whether their value is greater or lower than the steady one, and possibly the ordinal relation with the other landmarks. The options offered in the menus represent either the variables that can be perturbed, i.e. state variables, constant flows and fractional transfer rate coefficients which define linear flows, or the intervals where the steady and the perturbed values can lie. As the system is initially supposed to be in equilibrium, this means that the *qual* of  $\dot{x}_i$  must be equal to zero, that is the flows entering a compartment perfectly balance the flows leaving the compartment. QCMF checks the consistency of the specified steady values with the structure model.

### Tracer kinetics

Compartmental modeling plays an important role in describing both the distribution and metabolism of either tracer labeled compounds or drugs. An ideal tracer experiment consists in the application of a test signal having negligible mass and behaving metabolically exactly as the original substance to which is added as a marker. A number of useful properties emerge (Carson *et al.*, 1983) from such hypotheses. First, the tracer exhibits linear dynamics if the system is in a constant steady state and irrespective of whether or not the functional dependencies defining the intercompartmental flows are linear. Furthermore, these linear dynamics are time invariant. Qualitative models of tracer kinetics can easily allow the user to investigate the consequences of removing the hypothesis of "small-signal perturbation" by considering non linear functional dependencies.

In this scenario, the generation of the initial state could be seen as a particular case of the perturbation of a steady state where the steady value of each variable is assumed to be equal to zero (this means absence of tracer in all compartments), and there is a single perturbation which corresponds to the introduction of a tracer labeled compound into one compartment. However a specific sub-module has been implemented mainly to manage in a suitable way the dialogue with

the user. Within this context, the variables represent the concentration and transfer of the tracer substance.

The user must choose, through menus, both the compartment where the tracer is introduced and the kind of input signal for the tracer. The input signal can be either an impulse (injection experiment) or a step signal (infusion experiment). QCMF verifies if the structure model is consistent with the chosen input signal, that is the absence or presence of a flow from the external world to the compartment, respectively.

### Behavior explanation

We consider explanations as presentation of information that offer a meaningful interpretation of simulation results, which describes, in a language that is comprehensible by the user, how and why a behavior occurs. Therefore, the core of the EXPLAIN module, which has been fully designed, but still under development, deals with a method for generating a causal interpretation of behaviors; more precisely, given a succession of qualitative states describing a simulated behavior, QCMF identifies the cause-effect links between variables in successive states. One of the basic distinctions between the proposed different approaches to the generation of causal explanations of the system behaviors (De Kleer and Brown, 1986; Iwasaki and Simon, 1986a; Iwasaki and Simon, 1986b; Gautier and Gruber, 1993) is whether the relations or constraints used in the model are directional or not. In the former case, a causal account is simple to be obtained because the values are propagated through the directional constraints supplied by the model builder. Alternatively, bidirectional constraints require causal ordering techniques because they lack an explicit representation of causality. The context in which the EXPLAIN module generates a causal account is limited to compartmental modeling methodology in which relationships are directional. This allows us, given a bidirectional QSIM model, to interpret the system's behavior by combining simulation results and the knowledge of the directional relationships expressed through the compartmental diagram. In the transition from a state ( $QS_i$ ) to its successor ( $QS_{i+1}$ ) one or more variables will change qualitative state. This transition from state  $QS_i$  to  $QS_{i+1}$  will be necessarily due to the *qdirs* in state  $QS_i$  describing the evolution occurring in the system at time  $t_i$ . This principle is the basis of the algorithm underlying the EXPLAIN module because it allows QCMF to causally concatenate the changes from a state to its successor: the algorithm when comparing two successive states will single out those changes that are a direct consequence of the directions of change in  $QS_i$  and will propagate the effects of these changes through the directional constraints to account for the remaining changes. Applied to a behavior chosen by the user, this procedure is repeated beginning from the initial state to the whole sequence of qualitative states describing the behavior. The causal account we are

aiming at arises from chaining the separate accounts obtained for the successive transitions. The user will be allowed to ask for causal explanations of either a specific behavior of the system or the differences the whole set of possible system behaviors exhibits. The presentation of explanations are provided at different level of details: first, the relevant events are presented, and then, on the user request, more and more detailed causal explanations for any change in the system variables are produced.

### An example of application of QCMF to the medical domain

The lack of methodological knowledge necessary for building and simulating models has been a serious deterrent for physicians, much more than for expert in other domains, from using system models in spite of their great utility for reasoning about pathophysiological systems. As shown in (Ramoni *et al.*, 1992), medical reasoning may be broken down into two different phases: first, initial information is exploited to select candidate problem's solutions (*hypothesis selection phase*), and then these solutions are used as starting conditions to forecast expected consequences that should derive if adopted as final solutions (*hypotheses testing phase*). Diseases represent the solutions of diagnostic reasoning, i.e. the search for the best explanation of the current situation of a patient, while treatments represent the solutions of a therapy planning problem, i.e. the search for the best action to ameliorate a patient's conditions. Structure and behavior models provide an essential knowledge representation formalism for exploiting genuine pathophysiological theories, when available, in the testing phase of medical reasoning (Ironi *et al.*, 1990): the consequences of a selected solution can be obtained by simulating a model of the considered pathophysiological process. A disease can be modeled by changing the values of some model variables, and possibly either the functional relationships between variables or the structure model with respect to the one describing the physiological behavior. Thus, the model simulation provides the new steady state, which is abnormal, as well as the transient to achieve it. In case of therapy planning, also the action of the hypothesized treatment needs to be modeled: for example a drug's pharmacokinetics is essential to describe how it affects the system's behavior. Thus, a KBS should contain different structure and behavior models which could be used whenever the solution's hypotheses they represent are selected as candidate solutions. In the medical context, QCMF can be viewed as a powerful tool for the acquisition of pathophysiological knowledge.

Let us consider, for example, the Glucose-Insulin Regulatory system (G-I-R) in order to describe QCMF at work. After the selection of the system, the user is then allowed to enter one of the model-reasoning phases. The structure model shown in Fig. 3 corre-

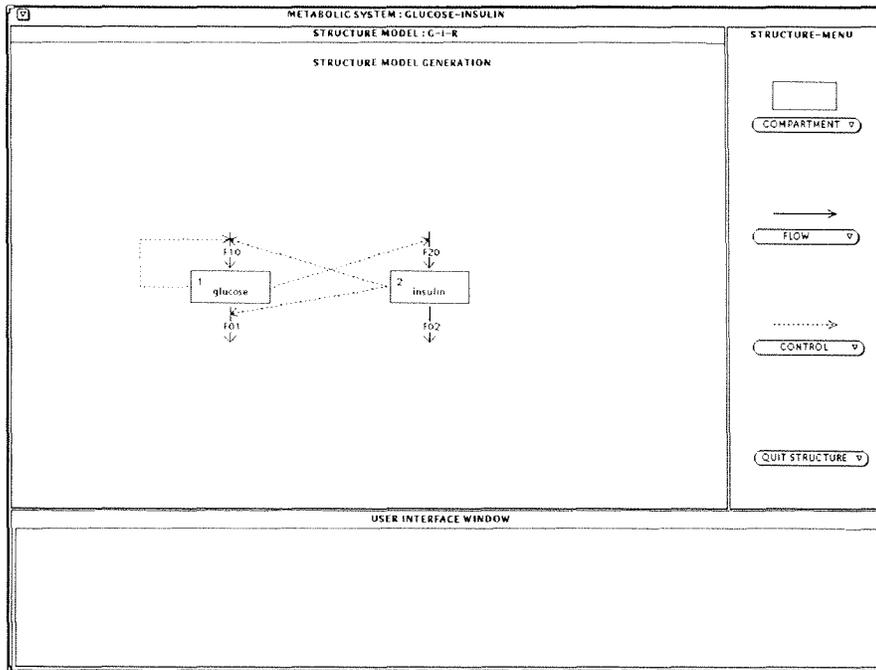


Figure 3: A structure model of the glucose-insulin regulatory system

sponds to the simplest decomposition for G-I-R, and includes two compartments: glucose and insulin.

The glucose is described by a single compartment representing extracellular fluids. The processes that have been explicitly considered are the liver glucose production  $f_{10}$  and the glucose elimination  $f_{01}$ . The liver glucose production depends on both extracellular glucose and plasma insulin; while the glucose elimination is due to renal excretion, as well as to the insulin-dependent glucose utilization by muscles and adipose tissues and the insulin independent glucose utilization by the central nervous system and red blood cells. The insulin subsystem is also described by a single compartment representing plasma insulin. The insulin production derives from the pancreatic response to glucose stimulation. There is no exchange of flow between the two compartments, but they interact to control the glucose production and elimination and the insulin production.

At first, the automatic analysis of the structure in Fig. 3 identifies the following variables:  $X_1$ ,  $X_2$ ,  $DX_1$ ,  $DX_2$ ,  $F_{10}$ ,  $F_{01}$ ,  $F_{20}$ ,  $F_{02}$ . The information stored in the structure frame (Fig. 4) allows QCMF to identify which flow and state variables are related to. The flow  $F_{01}$  depends on  $X_1$  and on the controlling variable  $X_2$ ,  $F_{10}$  depends on both  $X_1$  and  $X_2$ , while  $F_{20}$  and  $F_{02}$  are respectively function of  $X_1$  (controlling variable), and  $X_2$ .

As far as the liver glucose production ( $F_{10}$ ) is concerned, it is reasonable to assume that  $F_{10}(X_1, X_2)$  is expressed by summing two single valued functions of

**name:** *G-I-R*

**pathophysiological-system:** *Glucose-Insulin*

**behaviors:** *normal beh-1 beh-2*

**compartments:** *1-glucose 2-insulin*

**flows:** *F10 F01 F20 F02*

**controls:** *1-F10 2-F10 2-F01 1-F20*

Figure 4: The frame representing the structure model of the Glucose Insulin Regulation system shown in Fig. 3. Slot names are indicated in bold while slot values in italic

$X_1$  and  $X_2$ , whose dubbed names  $X_1F_{10}$  and  $X_2F_{10}$  are automatically built by joining the names of the controlling variable and the controlled flow. Both  $X_1F_{10}$  and  $X_2F_{10}$  can reasonably be assumed to behave as a monotonic decreasing function of  $X_1$  in the interval  $(0 X_1^*)$  and of  $X_2$  in the interval  $(0 X_2^*)$ , respectively, and constant elsewhere. Analogously, the function  $F_{01}(X_1, X_2)$ , describing the glucose elimination, can be defined by the sum of  $X_1F_{01}$  and  $X_2F_{01}$ , where  $X_1F_{01}(X_1)$  can be assumed to be monotonic increasing, and  $X_2F_{01}(X_2)$  monotonic increasing in the interval  $(0 X_2^{**})$ , and constant elsewhere. The production of insulin ( $F_{20}$ ) can be reasonably assumed to increase monotonically with the concentration of glucose ( $X_1$ ) reaching a saturation value ( $F_{20}^{**}$ ) for

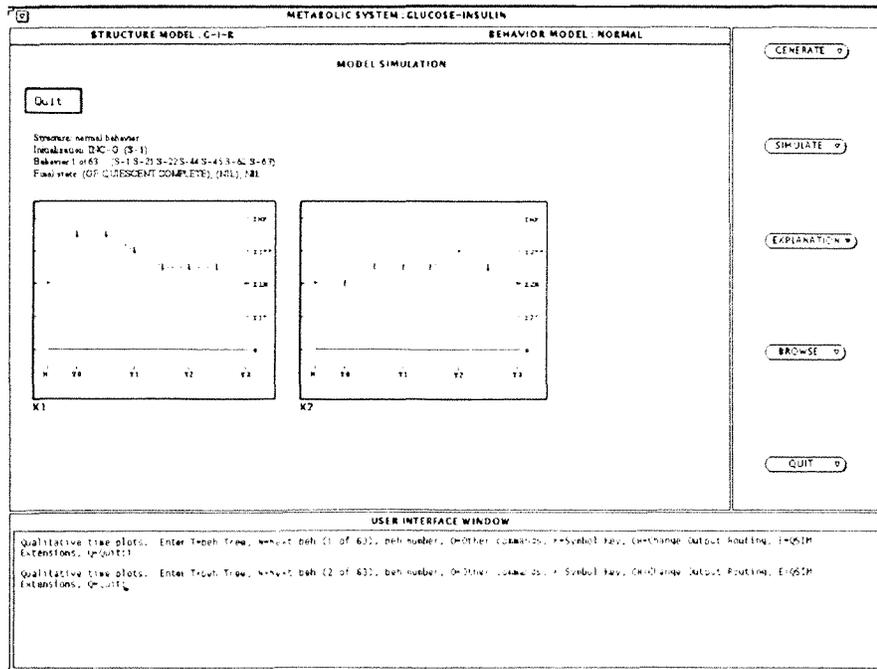


Figure 5: A possible behavior of the G-I-R system in response to an increased glucose concentration with respect to the normal value, denoted in correspondence to  $N$

$X1=X1^{**}$ . Such a choice is supported by available experimental knowledge. All the above mentioned landmarks are automatically generated and included in the quantity spaces of their respective variables. Let us notice that a landmark can also be removed from its quantity space when new information allows us to narrow either the domain or the codomain of the corresponding variable. For example, the quantity space of  $F20$ , firstly instantiated to  $(0 \infty)$ , is restricted to  $(0 F20^{**})$  as  $F20^{**}$  represents the maximum value  $F20$  can take on. The insulin elimination  $F02$  can be assumed to be a monotonic increasing function of  $X2$ . As far as the liver glucose production ( $F10$ ) is concerned, it is reasonable to assume that  $F10(X1, X2)$  is expressed by summing two single valued functions of  $X1$  and  $X2$ , whose dubbed names  $X1F10$  and  $X2F10$  are automatically built by joining the names of the controlling variable and the controlled flow. Both  $X1F10$  and  $X2F10$  can reasonably be assumed to behave as a monotonic decreasing function of  $X1$  in the interval  $(0 X1^*)$  and of  $X2$  in the interval  $(0 X2^*)$ , respectively, and constant elsewhere. Analogously, the function  $F01(X1, X2)$ , describing the glucose elimination, can be defined by the sum of  $X1F01$  and  $X2F01$ , where  $X1F01(X1)$  can be assumed to be monotonic increasing, and  $X2F01(X2)$  monotonic increasing in the interval  $(0 X2^{**})$ , and constant elsewhere. Of course, other kinds of functional dependencies between variables could be hypothesized, and, therefore, the assumed behavior model is not the only possible one for the structure in Fig. 3.

Let us now suppose that the perturbed condition of G-I-R consists of an increased glucose concentration with respect to the normal value (reference value). The initial state generated by QCMF results in a LISP function describing both the normal state and the perturbed one. By the analysis of the behavior model, the variables identified as possibly interested by the perturbation of  $X1$  are  $F20$ ,  $X1F10$ ,  $X1F01$ , and consequently  $F10$  and  $F01$ . As the normal value of  $X1$  is supposed to lie in the interval  $(X1^*, X1^{**})$ , as well as  $X2$  does in  $(X2^*, X2^{**})$ ,  $F20$ ,  $X1F01$  and  $F01$  take on a value greater than the normal one, whereas  $X1F10$ , and then  $F10$ , do not change since the perturbed value of  $X1$  still lies in the saturation interval. Therefore, it is  $DX1 < 0$  ( $DX1 = F10 - F01$ ), and  $DX2 > 0$  ( $DX2 = F20 - F02$ ). This means that  $qdir(X1)$  is decreasing and  $qdir(X2)$  increasing, and consequently, by the analysis of the constraints, it is  $qdir(DX1)$  increasing and  $qdir(DX2)$  decreasing.

In response to the assumed perturbation, two classes of behaviors have been obtained: transient courses showing damped oscillations or not. An example of behavior taken from the latter class is shown in Fig. 5. Let us notice that all the behaviors generated by the simulation are featured by a final steady state where all the variables assume again the normal value. This is the expected result as the considered model represents the physiological behavior of G-I-R. The simulated behaviors show all the possible ways the system in a perturbed condition reacts to bring itself back to the

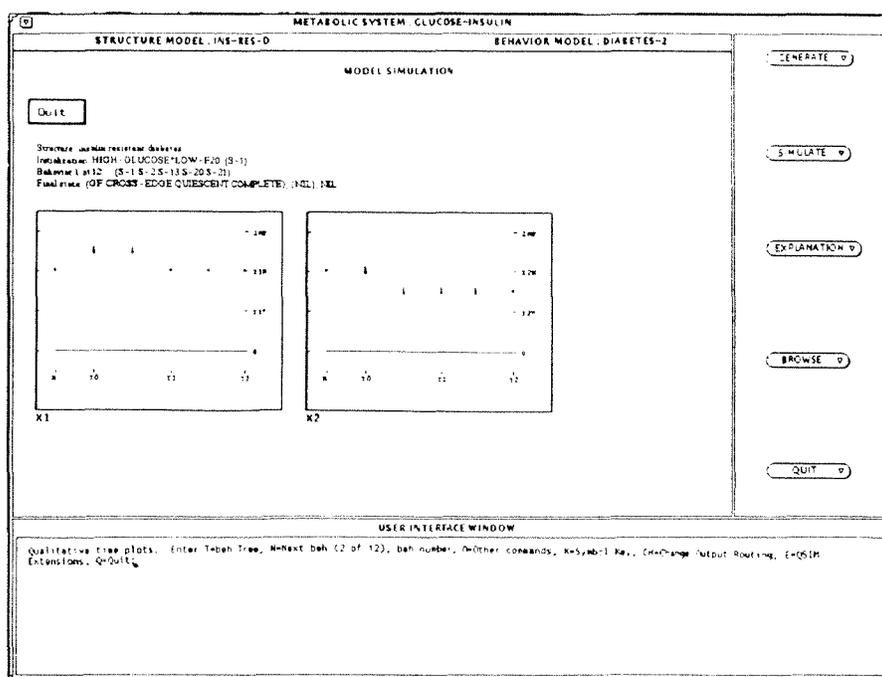


Figure 6: A typical behavior of the type II diabetes model in response to an increased value of the glucose concentration with respect to the normal one

normality. The behaviors grouped into the two classes mainly differ for the different temporal orders in which different events occur. More interesting scenarios describing G-I-R abnormal behaviors, such as type I and type II diabetes, cannot be obtained by simply perturbing the variables of the previous defined model, but structural and behavioral changes are needed. In type I diabetes there is almost no insulin secretion from pancreatic cells, that is the insulin production is not controlled any more by the glucose concentration. In type II diabetes, in addition to a reduced production of insulin, the glucose uptake by muscles and adipose tissues does not depend on insulin. As far as the behavior models are concerned, in the first case the variable F20 does not appear any more in the list of system variables and constraints. In the second case the variable F01 can be supposed monotonically increasing with X1, and F20 a constant function whose value is lower than the normal one.

Fig. 6 shows a typical response of the type II diabetes model in response to an increased value of the glucose concentration with respect to the normal one. As expected, as the glucose elimination is insulin resistant, the glucose concentration can reach again the normal value, whereas the insulin, for its reduced production, decreases and achieves a steady value lower than the normal one. Nevertheless, another class of predicted behaviors, where the glucose concentration achieves a steady value lower than the starting value but still higher than the normal one, is equally accept-

able from a pathophysiological point of view.

## Conclusion

Compartmental modeling is a well-known modeling technique for the representation of the behavior of complex systems, and its use by researchers in physics, biology and medicine dates back to the 1920's. In the compartmental approach, the behavior of a complex system is represented through an idealized decomposition of the system into interacting compartments, whose behavior is basically governed by the mass balance law. The compartments do not necessarily correspond to actual distinguishable physical components unlike the device-centered approach in which components have a direct correspondence with the different physical parts an artifact is made up of.

This paper describes the architecture of QCMF, a computer-based framework which integrates facilities for generating a model of a compartmental system, simulating its behavior and providing a causal explanation of the obtained behavior.

In other works (Addanki *et al.*, 1991; Falkenhaier and Forbus, 1991; Low and Iwasaki, 1992) dealing with qualitative model construction, the modeling problem is approached by selecting, within a large predefined model library, model fragments, and possibly by composing them. In particular, the selection is performed according to either a set of assumptions or a user's query about the system's behavior. In these works, the model formulation procedure is based on QPT (Forbus,

1984). QCMF, which is based on QSIM, allows the modeller to specify in a graphical way, the compartmental structure of any new system, and acquires the needed knowledge to generate a system model. Then, a new model is stored into a library from which it can be easily retrieved. The basic model components, i.e. compartments, are not worth to be stored separately since they are not physiologically meaningful outside the context where they are placed, i.e. the overall model. This represents the major difference between the two approaches: in the former one components can be fully defined independently from their combination into a specific model, in the latter one models can be built instantiating few conceptual entities, i.e. compartments, flows, control signals, and so on.

In this paper the main emphasis has been given to the description of QCMF's ability in model formulation, generation and simulation. These facilities are now fully operational. The main problems we faced deal with the simulation process. QCMF inherits from QSIM all of its strengths and limitations (Fouchè and Kuipers, 1992; Kuipers, 1987; Kuipers and Bearlant, 1988; Kuipers *et al.*, 1991; Lee and Kuipers, 1988; Shen and Leitch, 1991; Struss, 1988): it is capable to predict all possible system behaviors but can lead to an intractable branching of the behavioral tree. In order to control the simulation, QCMF automatically invokes filters for reducing the proliferation of behaviors, namely the Higher Order Derivative (HOD) constraint (Kuipers *et al.*, 1991) or, whenever this is not applicable, the chatter box elimination technique (Clancy and Kuipers, 1993). Whenever it is possible, QCMF also exploits filters expressing the mass conservation law by introducing into the behavior model the suitable variables and constraints. Another way to further reduce the behavior tree consists in performing an attainable envisionment, i.e. no new landmarks are introduced. As new landmarks are related to a specific behavior, this makes easier the comparison of behaviors allowing us to aggregate the similar ones. The aggregation of behaviors aims at taking the significant distinctions for the user needs out of the behavior tree. At a first instance, the user could be interested only in the distinctions either in a subset of variables or in the final equilibrium state. For example, in a therapeutic context, knowing if a system in an abnormal steady state, when perturbed by a therapy, is capable to achieve a normal state is the essential information. In fact, if the set of produced alternatives did not contain the expected behaviors, the perturbed model should be revised.

As regards its possible use, many issues can be considered. First, QCMF can work as a stand-alone system resulting in a powerful didactic tool for reasoning about the pathophysiological behaviors of a system. Then, QCMF can be fully integrated within larger knowledge-based systems that use different formalisms. In fact, it may generate knowledge sources that can be

properly exploited in the deductive inference of medical reasoning in the execution of both diagnostic and therapeutic tasks.

## Acknowledgements

We especially thank B. Kuipers who provided the Common LISP implementation of QSIM. We are also indebted to A. Cattaneo who implemented part of QCMF in his Master Thesis, to D. Clancy for providing his last extensions to QSIM, and to S. Tentoni for technical support. This work is part of the AIM project A2034, a General Architecture for Medical Expert Systems II (GAMES II), which is supported by the European Community.

## References

- Addanki, S.R.; Cremonini, R.; and Penberthy, J.S. 1991. Graphs of models. *Artificial Intelligence* 51:145-177.
- Atkins, G.L. 1974. *Multicompartmental Models in Biological Systems*. Chapman and Hall, London.
- Bobrow, D.G.(ed) 1984. *Qualitative Reasoning about Physical Systems*. North Holland, Amsterdam.
- Capelo, A.C.; Ironi, L.; and Tentoni, S. 1993. A model-based system for the classification and analysis of materials. *Intelligent Systems Engineering* 2(3):145-158.
- Carson, E.R.; Cobelli, C.; and Finkenstien, L. 1983. *The Mathematical Modeling of Metabolic and Endocrine Systems*. Wiley, New York.
- Clancy, D.J. and Kuipers, B.J. 1993. Behavior abstraction for tractable simulation. In *Proc. 7th International Workshop on Qualitative Reasoning*, Seattle. 57-64.
- Crawford, J.; Farquhar, A.; and Kuipers, B.J. 1992. QPC: A compiler from physical models into qualitative differential equations. In B. Faltings, P. Struss, editor 1992, *Recent Advances in Qualitative Physics*. MIT Press. 17-32.
- De Kleer, J. and Brown, J.S. 1986. Theories of causal ordering. *Artificial Intelligence* 29:33-61.
- Falkenhaier, B. and Forbus, K.D. 1991. Compositional modeling: finding the right model for the job. *Artificial Intelligence* 51:95-143.
- Forbus, K.D. 1984. Qualitative process theory. *Artificial Intelligence* 24:85-168.
- Fouchè, P. and Kuipers, B.J. 1992. Reasoning about energy in qualitative simulation. *IEEE Transactions on Systems, Man and Cybernetics* 22:47-63.
- Gautier, P.O. and Gruber, T.R. 1993. Generating explanations of device behavior using compositional modeling and causal ordering. In *Proc. 7th International Workshop on Qualitative Reasoning*, Seattle. 89-97.

- Ironi, L.; Stefanelli, M.; and Lanzola, G. 1990. Qualitative models in medical diagnosis. *Artificial Intelligence in Medicine* 2:85-101.
- Iwasaki, Y. and Simon, H.A. 1986a. Causality in device behavior. *Artificial Intelligence* 29:3-32.
- Iwasaki, Y. and Simon, H.A. 1986b. Theories of causal ordering: reply to de Kleer and brown. *Artificial Intelligence* 29:63-72.
- Iwasaki, Y. 1992. Reasoning with multiple abstraction models. In Faltings, B. and Struss, P., editors 1992, *Recent Advances in Qualitative Physics*. MIT Press. 67-82.
- Kuipers, B.J. and Bearlant, D. 1988. Using incomplete quantitative knowledge in qualitative reasoning. In *Proc. AAAI-88*, Los Altos. Morgan Kaufmann. 324-329.
- Kuipers, B.J. and Kassirer, J.P. 1985. Qualitative simulation in medical physiology: a progress report. Technical Report MIT/LCS/TM-280, MIT.
- Kuipers, B.J.; Chiu, C.; Dalle Molle, D.T.; and Throop, D.R. 1991. Higher-order derivative constraints in qualitative simulation. *Artificial Intelligence* 51:343-380.
- Kuipers, B.J. 1986. Qualitative simulation. *Artificial Intelligence* 29:280-338.
- Kuipers, B.J. 1987. Abstraction by time-scale in qualitative simulation. In *Proc. AAAI-87*, Los Altos. Morgan Kaufmann. 621-625.
- Kuipers, B.J. 1989. Qualitative reasoning with causal models in diagnosis of complex systems. In Widman, L.E.; Loparo, K.A.; and Nielsen, N.R., editors 1989, *Artificial Intelligence, Simulation and Modeling*. Wiley, New York. 257-274.
- Lee, W.W. and Kuipers, B.J. 1988. Non-intersection of trajectories in qualitative phase space: a global constraint for qualitative simulation. In *Proc. AAAI-88*, Los Altos. Morgan Kaufmann. 286-290.
- Low, C.M. and Iwasaki, Y. 1992. Device modelling environment: an interactive environment for modelling device behaviour. *Intelligent Systems Engineering* 1(2):115-145.
- Nayak, P.; Joskowicz, L.; and Addanki, S. 1991. Automated model selection using context-dependent behaviors. In *Proc. 5th International Workshop on Qualitative Reasoning*, Austin. Austin Univ. 10-24.
- Ramoni, M.; Stefanelli, M.; Magnani, L.; and Barosi, G. 1992. An epistemological framework for medical knowledge based systems. *IEEE Transactions on Systems Man and Cybernetics* 22:1361-1374.
- SAAM, 1993. User's guide manual. Resource Facility for Kinetics Analysis - Univ. Washington.
- Shen, Q. and Leitch, R.R. 1991. Combining qualitative simulation and fuzzy sets. In Faltings, and Struss, P., editors 1991, *Recent Advances in Qualitative Physics*. MIT Press. 83-100.
- Struss, P. 1988. Global filters for qualitative behaviors. In *Proc. AAAI-88*, Los Altos. Morgan Kaufmann. 275-279.
- Weld, D.S. and De Kleer, J.(eds.) 1990. *Readings in Qualitative Reasoning About Physical Systems*. Morgan Kaufmann, Los Altos.
- Weld, D.S. 1990. Approximation reformulations. In *Proc. AAAI-90*, Los Altos. Morgan Kaufmann. 407-412.

**Integration of Real-Time System Prototyping  
with Qualitative and Quantitative Reasoning-Based Parameter Tuning Methods**

Kiyoshi Itoh

Laboratory of Information and Systems Engineering, Faculty of Science and Technology,  
Sophia University, Kioi-cho 7-1, Chiyoda-ku, Tokyo 102, Japan  
itohkiyo@hoffman.cc.sophia.ac.jp, tel.+81-3-3238-4143, fax.+81-3-3238-3885

<KEYWORDS>integration paradigm, prototyping, parameter tuning, qualitative reasoning, real-time systems, synchronized queueing network

<ABSTRACT>There are a number of parameter tuning plans for improving a real-time system prototype performance. This paper describes the integration of real-time system prototyping with qualitative and quantitative methods by Bottleneck Diagnosis / Improvement Expert Systems for Synchronized queueing network (BDES-S and BIES-S). By the qualitative reasoning, BDES-S diagnoses or identifies bottlenecks and their sources, and generates qualitative improvement plan. BIES-S quantitatively estimates the effects of the improvement for bottleneck. BDES-S and BIES-S assume a real-time transaction oriented concurrent software system (TCSS) as a synchronized queueing network (SQN).

#### 1. Effective Performance Improvement for Real-Time System Prototype

Software prototyping is a software development method by which a prototype is constructed in a rapid and iterated fashion on the basis of careful verifications and evaluations of the prototype in order to clarify the requirements for the target software system to be developed or assure its feasibility. Prototyping cycle consists of prototype construction, execution and evaluation steps in order to satisfy the purpose of the software prototyping. The purpose of software prototyping in requirements analysis is to clarify, correct or modify ambiguous requirements of users or customers by reviewing the behavior of the prototype by repeating the cycle. The purpose in software design is to assure the feasibility of the target system by determining the implementation mechanism, algorithms or data structures to be adopted in the target system.

Real-time system prototyping methodology has been recognized to be very important as mentioned in [Luq88]. We developed a software prototyping environment called TransObj (TRANSACTION and OBJECT oriented prototyping environment) for designing real-time Transaction oriented Concurrent Software Systems (TCSS) ([ITO84, 92abc]). TCSS is a system in which one or more concurrent functional objects (abbreviated to FObjs) process a lot of transactions (abbreviated to Xacts) simultaneously in real-time. Examples of TCSSs include online reservation systems, database inquiry system, etc.. The Xact is an entity which generates on the external, is processed by the TCSS, and transmits the response to the external. FObj is an entity which is a processing component of the TCSS. One or more FObjs exist and run concurrently in one TCSS. The FObjs process the Xacts cooperatively. The TCSS has a synchronized queueing network (SQN) structure which consists of connected, Xact-driven FObjs. As a model for representing the behavior of complex TCSS, we devised the novel model, i.e., a synchronized queueing network (SQN) ([ITO91],[SHI91]). SQN is different from ordinary queueing networks (QN) ([KLE75], [GLE80]) in that SQN provides synchronized server objects.

TCSS applications require strict performance requirements. For example, a TCSS requires much more precise timing than a business application system. The response time is defined as the elapsed time between the generation of the Xact on the external and the output of the required response to the external. If the response time exceeds the permissible limit, malfunctions would occur. If the actual performance is discovered to be out of the permissible range after installation, labor will inevitably be spent in maintenance activities, such as reconfigurations and modifications. It is very important to predict the target TCSS performance precisely during the design phase, in order to find the optimal TCSS configurations.

The TransObj's methodology is called "Stepwise Prototyping Method (SPM)" by which a designer can identify and design concurrent FObjs clearly. TransObj aims at providing an integrated set of prototyping tools. In addition, TransObj has the ability for qualitative reasoning based parameter tuning for object networks in [ITO89bc, 90ab, 91ab]. For TCSS to be designed, any bottlenecks should be detected rapidly. Appropriate performance improvement plans for the bottlenecks should be generated. In particular, the reduction of prototyping cost depends on the ability to produce improvement plans. The prototyping evaluation step should accomplish appropriate parameter tuning to reduce prototyping cost. In performance design, there are a large number of parameter tuning plans for the bottleneck of a TCSS prototype.

For non-experts, in order to select appropriate plans, We developed two knowledge-based expert systems, BDES (Bottleneck Diagnosis Expert System) and BIES (Bottleneck Improvement Expert System). BDES qualitatively diagnoses or identifies bottlenecks and their sources, and generates qualitative improvement plan. BIES quantitatively estimates the effects of the improvement for bottleneck and their sources on the whole queueing network [ITO89b, 90]. BDES and BIES are based on "qualitative and quantitative reasonings," respectively. The application range of BDES and BIES is beyond the range of analytical methods which are based on "queueing theory" because it is not easy for analytical methods to produce distinct improvement plans for bottlenecks. As further extension, we have developed BDES-S and BIES-S [ITO92c]. BDES-S and BIES-S assume a real-time TCSS as a synchronized queueing network (SQN). In SQN, an object is called a server. BDES-S and BIES-S call it a server object. Xacts are processed by TCSS, i.e., Xacts are flowing on a SQN.

To determine a tuning plan, it is effective to arrange experts' knowledge. The knowledge can be well formulated within a framework of qualitative expression. At the first step for a parameter tuning, a qualitative reasoning method determines more than one tuning plans, each indicating the increases or decreases of parameters valid for tuning. All that such a tuning plan indicates is the judgment of increase or decrease of parameters; it does not indicate the specific amounts of increases or decreases of parameters. At the second step for the parameter tuning, a quantitative tuning is performed based on a quantitative reasoning method for each qualitative tuning plan proposed at the first step.

Qualitative reasoning is usually used to provides a model to express typical qualitative behaviors of some economic systems, electric circuits, and etc. ([APT86], [BOB85], [DEK85]). Our approach, i.e., applying a qualitative reasoning method to the QN, is quite a new approach ([ITO89ab,90,91a], [SAW90]).

## 2. SQN and Bottleneck

The open-type QN is dealt with in which all Xacts to be processed arrive / go out from / to the external. We assume that only 1 type of Xacts exists in the QN. A Synchronized QN (SQN) contains not only ordinary FCFS (First Come First Served) server objects in ordinary QN but also 3 types of synchronized server objects. Performance Parameters are as follows: LAMBDA: average arrival rate of Xacts for a server object,  $\mu$  : average servicing rate of a server object for Xacts, T : average throughput of Xacts by a server object, RHO: average utilization rate of a server object, Q: average queue length of Xacts in front of a server object, and R: branching probability of Xacts at a branching point.

A SQN is composed of the following 4 types of server objects. These 4 types of server objects can be used to represent various aspects of the behavior of actual systems.

(1)Normal Server object (1st entry of Table 1): A Xact arrived enters into the server object By the FCFS rule and leaves the server object after a finite time period.

(2)SPLIT Server object (2nd entry of Table 1): A Xact arrived enters into the server object By the FCFS rule and produces a clone after a finite time period. The 2 Xacts leave the server object through separate output paths simultaneously. Thus, the 2 outputs always have the same quantity.

(3)MERGE Server object (3rd entry of Table 1): Xacts arrive at 2 different queues through 2 different

paths. By the FCFS rule, 2 Xacts, from the 1st and 2nd queues, enter into the server object simultaneously and are merged into 1 Xact after a finite time period. If there is no Xact in another queue, the Xact waits for another Xact to arrive at another queue. Xact B in 3-b-2) and 3-c) of Table 1 is waiting for Xact A.

(4)MATCH Server object (4th entry of Table 1): Xacts arrive at 2 different queues through 2 different paths. By the FCFS rule, 2 Xacts, from the 1st and 2nd queues, enter into the server object simultaneously and, after a finite time interval, each Xact leaves the server object through separate output paths simultaneously. Thus, the 2 outputs always have the same quantity.

In an ordinary QN, a bottleneck exists at the server object whose RHO is very close to 1. The queue at the bottleneck server object may grow into an infinite length. If such a bottleneck server object exists, the QN is in an unstable (overloaded) state.

An expert's heuristics is that RHO of 0.7 is a bottleneck landmark (BL). If  $RHO \geq BL$ , experts judges, from the failure-to-safety aspect, that the possibility of bottleneck exists. Q of 1 is another bottleneck landmark (QBL). If  $Q \geq QBL$ , there is a risk that excessive Xacts will arrive, causing a further increase in the RHO. The excessive Q might be gradually reduced later, but experts judges, from the failure-to-safety aspect, that the possibility of bottleneck exists.

Failure-to-safety aspect means, in narrow sense, that system has the mechanism by which it fails with not hazardous state but safety state. In broad sense, it means that hazardous conditions are avoided in advance if the possibility of hazardous state exists. In the SQN, the bottleneck is considered as one of hazardous states. In M/M/1 (Poisson arrival / exponential service) type of server object, an equation,  $Q = RHO^2 / (1 - RHO)$ , holds at the stable state. Our objective QN may consist of non-M/M/1 type server objects, may be in the unstable state, or may not be measured in long time period. Therefore, distinct bottleneck landmarks, both  $RHO \geq 0.7$  and  $Q \geq 1$ , are provided on the failure-to-safety aspect.

On MERGE or MATCH server object, if the 2 queues have different LAMBDAs, the quantity processed by the server object is equal to the smaller LAMBDA. The difference of the 2 LAMBDAs will be accumulated as the system is running. From the failure-to-safety aspect, it can be judged that MERGE and MATCH server objects with 2 different LAMBDAs have the possibility of bottlenecks.

### 3. QLBE and QL-BIE Adopted in BDES-S

In the qualitative reasoning notation, [RHO] and [Q] are qualitative values evaluated with BL and QBL to be the origin, respectively.

[RHO]= +: Possibility of bottleneck exists. This means  $RHO > BL$ . From failure- to-safety aspect, we assume that  $dRHO = +$ . [Q]=+: Possibility of bottleneck exists. This means  $Q > QBL$ . We assume that  $dQ = +$ .

Qualitative expressions describe the behavior of each server object. These expressions are called qualitative behavior expressions (QLBE) for each server object. The expressions used in the case of bottleneck possibility are specifically called qualitative bottleneck improvement expressions (QL-BIE).

Table 2 shows QLBE and QL-BIE for each server object. The 1st item [RHO] = - indicates QLBE and the 2nd item [RHO] = + indicates QL-BIE. The 1st and 2nd items of Table 2(a) correspond to 1-a) and 1-b), respectively, of the 1st entry of table 1. For example, the 1st item of Table 2(a) indicates that RHO and t follows an increase and decrease of LAMBDA when [RHO] = - to a normal server object, because equation " $RHO = LAMBDA / MU$ " holds as shown in 1-a) of Table 1. The 2nd item of Table 2(a) indicates that increasing RHO ( $dRHO = +$ ) is inhibited. In the same way, the 1st row in the 3rd item of Table 2(c) indicates that a decrease of LAMBDA-B decreases Q-B but has no effect on RHO, T and Q-A when [RHO] = -, [Q-A] = - and [Q-B]=+ to a MERGE server object.

In order to change LAMBDA of a server object, t from a server object placed in its upstream should be changed. An increase of t of a server object may increase LAMBDA of a server object placed in its downstream. Therefore, expressions for all server objects should be resolved simultaneously.

Fig.1 shows an example of a SQN referred to as "SQN5" which represents a real-time system prototype. The performance is measured by a simulation package. The total number of expressions for SQN5 is about 350. With additional information on the linkage information between server objects, solving these expressions simultaneously based on the qualitative behavior reasoning method is not efficient because of an explosion of the number of states.

#### 4. Substructure-Based QL-BIE adopted in BDES-S

Instead of solving all the expressions, we formulate expressions representing the correlations among internal performance parameters for each substructure, to reduce the overall number of expressions.

The following heuristics can be used to identify bottlenecks: 1)The server object whose  $RHO \geq 0.7$  may be the bottleneck. 2)The server object whose  $RHO < 0.7$  but  $Q \geq 1$  may be the bottleneck. 3)The MERGE / MATCH server object in which one LAMBDA is greater than another LAMBDA (i.e., one  $q$  is greater than another  $q$ ) in the 2 input paths may be the bottleneck.

Table 3 lists 14 knowledges for bottleneck improvement using substructures and their internal parameter values. To improving bottleneck at tandem server object s2, knowledge 3 tells us that  $t$  in at the just upstream server object be reduced. Also knowledge 2 indicates that, when increasing MU, we must reduce excessive RHO downstream.

The knowledges in Table 3 and QL-BIEs reveal that, if we focus our attention to substructures of the QN, the number of performance parameters is considerably reduced from the case where we discuss all server object parameters. For example, if server object s3 that receives throughput from a loop has the possibility of bottleneck, we can improve it by reducing only the amount of arrival at the loop, instead of reducing the performance parameters within the loop (see knowledge 7 in Table 3). We can use only the following qualitative improvement expressions to improve the bottleneck:  $dRHO_3 = -$   $dT_{in} = -$

For MERGE and MATCH server objects before transformation, LAMBDA of the normal server objects after transformation may be increased to prevent queues from growing too large. To increase LAMBDA of a server object,  $t$  from a server object that is placed in the upstream of the server object should be increased. Thus, an increase of LAMBDA results in increases of RHOs of server objects which are in its upstream, and can cause bottlenecks of these server objects. To improve these bottlenecks, expressions for these conditions should be solved using the knowledge from 10 to 14 in Table 3.

In 2 input paths of MERGE / MATCH server object, if one LAMBDA is greater than another LAMBDA (i.e., one  $q$  is greater than another  $q$ ), there exist 4 types of bottleneck improvement, i.e.,

- the large LAMBDA may be decreased,                      - the small LAMBDA may be increased,
- both may be decreased in different rates, and       - both may be increased in different rates.

#### 5. Example of Qualitative Parameter Tuning by BDES-S

BDES-S can enumerate qualitative improvement plans using heuristics as shown in Table 3. For bottleneck object s19 (MERGE server object) in SQN5 as shown in Fig.1, Fig.2 shows a qualitative diagnosis example by BDES-S. Item (1) enumerates bottleneck server objects by heuristics on initial diagnosis. Item (2) shows the structural diagnosis on bottleneck server object s19. Item (3) shows the rough sketch on improvement for the bottleneck s19. Item (4) shows qualitative improvement plans for decreasing the large LAMBDA. There are 4 alternative plans. In Item (5), some combination method shows that "\*" and "+" represent "and" and "or", respectively. There are 4 plans, i.e., 1, 2, 3\*4, 3\*5 for increasing small LAMBDA. Items (6) and (7) show that both upper and lower parts of dashed line should be required for decreasing or increasing both LAMBDAs.

#### 6. Quantitative Parameter Tuning by BIES-S

The automatic production of quantitative equations is based on the heuristics about so-called flow balance meaning that, if the bottlenecks can be improved, the input quantity is equal to the output quantity at any part of the SQN. For example, LAMBDA for the server object is equal to its T, and LAMBDA for the loop is equal to its T, if there is no bottleneck server object. By the flow balance, RHO is forced to decrease to 0.7. Only if the bottleneck server object can be improved: new MU = original LAMBDA / 0.7 or new LAMBDA = original MU \* 0.7. For a server object whose LAMBDA may be increased after tuning: new MU = new LAMBDA / 0.7. For SQN with NORMAL type server objects and SPLIT type server objects, quantitative parameter tuning can be performed using the above equations. For MERGE type server objects and MATCH type server objects with unbalanced LAMBDA's or which may get unbalanced after tuning, new LAMBDA-B = new LAMBDA-B, new MU = new LAMBDA-A / 0.7 or new LAMBDA-B / 0.7

Table 4 shows the example of quantitative parameter tuning for S19.

## 7. Concluding Remarks

BDES-S and BIES-S have been implemented in Prolog on a PC.

## References

- [APT86]Apte, C. et al. Using qualitative reasoning to understand financial arithmetic, Proc.AAAI, (1986).
- [BOB85]Bobrow,D.G.,et al.:Qualitative reasoning about physical systems, MIT Press, (1985).
- [DEK85]De Kleer, J.:How circuits work, in [BOB85].
- [GEL80]Gelenbe,E. et al.: Analysis and Synthesis of Computer Systems, Academic Press,(1980).
- [ITO84]Itoh,K. et al.: Software Design Process: Chrysalis Stage under the Control of Designers, J.Inf.Process., Vol.7, No.1, pp.5-15 (Mar. 1984).
- [ITO89a]Itoh,K. et al.: Tools for Prototyping for Developing Software, Journal of IPSJ, Vol.30, No.4, pp.387-395, (Apr. 1989), in Japanese.
- [ITO89b]Itoh, K., et al.: Knowledge-based parameter tuning for queueing network type system - A new application of qualitative reasoning,IFIP CAPE'89, pp.209- 216, (Oct.1989).
- [ITO90]Itoh,K., et al.: Role of Qualitative and Quantitative Reasoning in Diagnosis and Improvement for Queueing Network Bottleneck, InfoJapan'90,Vol.2, pp.171-178, (Oct.1990).
- [ITO91a]Itoh,K. et al. : Application of Qualitative Reasoning to Parameter Tuning, Journal of IPSJ, Vol.32, No.2, pp.171-178, (Feb., 1991), in Japanese.
- [ITO91b]Itoh,K. et al. : Qualitative Reasoning Based Parameter Tuning on Bottleneck of Synchronized Queueing Network, IEEE Compsac '91, pp.307-314, (Sep. 1991)
- [ITO92a]Itoh,K., et al.: TransObj: Software prototyping environment for real- time transaction-based software system applications, International Journal of SE and KE, Vol.2, No.1, pp.5-30, (Mar. 1992).
- [ITO92b]Itoh,K.: Systematic integration of qualitative and quantitative parameter tuning methods for improving real-time system prototypes, 2nd IEEE ICSI, pp.54-65, (June, 1992).
- [ITO92c]Itoh,K. et al.: An integrated method for parameter tuning on synchronized queueing network bottlenecks by qualitative and quantitative reasoning, IEICE Trans.Information and Systems, Vol.E75D, pp.635-647, (Sep. 1992).
- [KLE75]Kleinrock,L.: Queueing Systems, John Wiley & Sons, Inc., (1975).
- [LUQ88]Luqi et al.: Rapidly Prototyping Real-Time Systems, IEEE Software, pp.25-36, (Sep., 1988).
- [SAW90]Sawamura,J., Itoh,K. et al.: A Method for Diagnosis and Improvement on Bottleneck of Queueing Network by Qualitative and Quantitative Reasoning, Trans. JSAI, Vol.5, No.1, pp.92-105, (Jan.,1990), in Japanese.
- [SHI91]Shida,K., Itoh,K. et al. : A Method for Qualitative Parameter Tuning of Synchronized Queueing Network Bottleneck, Trans.JSAI, Vol.6, No.6, pp.891-903, (Nov., 1991).

Table 2 QLBE and QL-BIE for each type of servers.

	$d\rho$	$dt$	$d\lambda$	$d\mu$
$[\rho] = -$	$\pm$	$\pm$	$\pm$	
	$\mp$	0		$\pm$
$[\rho] = +$	-	0	-	
	-	+		+

	$d\rho$	$dt_A$	$dt_B$	$d\lambda$	$d\mu$
$[\rho] = -$	$\pm$	$\pm$	$\pm$	$\pm$	
	$\mp$	0	0		$\pm$
$[\rho] = +$	-	0	0	-	
	-	+	+		+

	$d\rho$	$dt$	$dq_A$	$dq_B$	$d\lambda_A$	$d\lambda_B$	$d\mu$
$[\rho] = -$	$\pm$	$\pm$	0	0	$\pm$	$\pm$	
$[q_A] = -$	0	0	+	0	+	0	
$[q_B] = -$	-	-	0	+	-	0	
	$\mp$	0	0	0			$\pm$
$[q_A] = +$	-	0	-	-	-	-	
$[q_B] = +$	-	+	-	-	-	-	+
*	-	+	-	-	-	-	+
$[\rho] = -$	0	0	0	-	-	-	
$[q_A] = -$	-	-	0	-	-	-	
$[q_B] = -$	+	+	0	-	+	-	
$[q_A] = +$	+	+	0	-	+	+	

	$d\rho$	$dt_A$	$dt_B$	$dq_A$	$dq_B$	$d\lambda_A$	$d\lambda_B$	$d\mu$
$[\rho] = -$	$\pm$	$\pm$	$\pm$	0	0	$\pm$	$\pm$	
$[q_A] = -$	0	0	0	+	0	+	0	
$[q_B] = -$	-	-	-	0	+	-	0	
	$\mp$	0	0	0	0			$\pm$
$[q_A] = +$	-	0	0	-	-	-	-	
$[q_B] = +$	-	+	+	-	-	-	-	+
*	-	+	+	-	-	-	-	+
$[\rho] = -$	0	0	0	0	-	-	-	
$[q_A] = -$	-	-	-	0	-	-	-	
$[q_B] = -$	+	+	+	0	-	+	-	
$[q_A] = +$	+	+	+	0	-	+	+	

\* denotes the expression to be used in case of  $q_A < q_B$

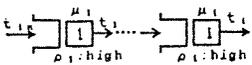
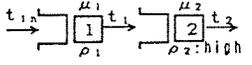
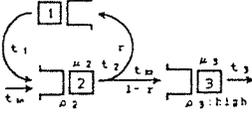
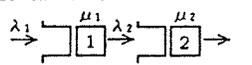
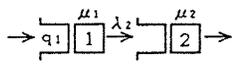
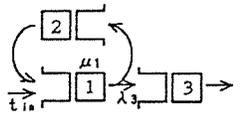
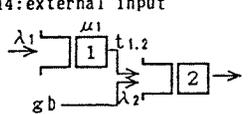
Table 1 Four types of servers and their performance parameters.

<p>1) normal server</p> <p>1-a) case of <math>\lambda &lt; \mu</math></p>	<p>1-b) case of <math>\lambda \geq \mu</math></p>
<p>2) split server</p> <p>2-a) case of <math>\lambda &lt; \mu</math></p>	<p>2-b) case of <math>\lambda \geq \mu</math></p>
<p>3) merge server</p> <p>3-a) case of <math>\lambda_A = \lambda_B &lt; \mu</math></p>	<p>3-b) case of <math>\lambda_A \geq \mu</math> and <math>\lambda_B \geq \mu</math></p> <p>3-b-1) case of <math>\lambda_A = \lambda_B \geq \mu</math></p>
<p>3-b-2) case of <math>\mu \leq \lambda_A &lt; \lambda_B</math></p>	<p>3-c) case of <math>\lambda_A &lt; \lambda_B</math> and <math>\lambda_A &lt; \mu</math></p>
<p>4) match server</p> <p>4-a) case of <math>\lambda_A = \lambda_B &lt; \mu</math></p>	<p>4-b) case of <math>\lambda_A \geq \mu</math> and <math>\lambda_B \geq \mu</math></p> <p>4-b-1) case of <math>\lambda_A = \lambda_B \geq \mu</math></p>
<p>4-b-2) case of <math>\mu \leq \lambda_A &lt; \lambda_B</math></p>	<p>4-c) case of <math>\lambda_A &lt; \lambda_B</math> and <math>\lambda_A &lt; \mu</math></p>

Table 4 Qualitative improvement plans and quantitative improvement equations & plans.

server	plan name plan type	qualitative improvement plan	quantitative improvement equation	quantitative improvement plan
s19	S19LD1 large $\lambda \rightarrow$	d r7,13 = -	$X = r7,13 (0.0 < X < 0.8)$ $0.096 \times X = 0.054$	r7,13 0.8 $\rightarrow$ 0.563
	S19LD4 large $\lambda \rightarrow$	d ga = -	$X = ga (0.0 < X < 0.166)$ $X \times 0.7 \times 0.8 \times 0.8$ $= X \times 0.7 \times 0.2 + X \times 0.3$	ga 0.166 $\rightarrow$ 0.0
	S19S11 small $\lambda \rightarrow$	d r3,8 = +	$X = r3,8 (0.2 < X < 1.0)$ $0.117 \times (1-X) \times 0.8$ $= 0.117 \times X + 0.033$	r3,8 0.2 $\rightarrow$ 0.2877
	S19S13 small $\lambda \rightarrow$	d ga = + d r2,5 = +	$X = ga (0.166 < X < 1.0)$ $Y = r2,5 (0.2 < Y < 1.0)$ (s9) $X \times 0.3 = 0.167 \times Y$ (s19) $X \times 0.7 \times 0.8 \times 0.8$ $= X \times 0.7 \times 0.2 + X \times 0.3$	ga 0.166 $\rightarrow$ 0.0 r2,5 0.2 $\rightarrow$ 0.0
	S19B01 both $\lambda \rightarrow$	d r3,8 = - d r7,13 = -	$X = r3,8 (0.0 < X < 0.2)$ $Y = r7,13 (0.0 < Y < 0.8)$ $0.117 \times (1-X) \times Y$ $= 0.117 \times X + 0.033$	r3,8 0.2 $\rightarrow$ 0.15 r7,13 0.8 $\rightarrow$ 0.504
	S19B02 both $\lambda \rightarrow$	d r3,8 = - d r1,3 = -	$X = r3,8 (0.0 < X < 0.2)$ $Y = r1,3 (0.0 < Y < 0.7)$ $0.166 \times Y \times (1-X) \times 0.8$ $= 0.166 \times Y \times X + 0.033$	r3,8 0.2 $\rightarrow$ 0.1 r1,3 0.7 $\rightarrow$ 0.3206
	S19B06 both $\lambda \rightarrow$	d ga = - d r2,5 = - d r7,13 = - d $\mu 22 = +$	$X = ga (0.0 < X < 0.166)$ $Y = r2,5 (0.0 < Y < 0.2)$ $Z = r7,13 (0.0 < Z < 0.8)$ $W = \mu 22 (0.1 < W < 1.0)$ (s9) $X \times 0.3 = 0.167 \times Y$ (s19) $X \times 0.7 \times 0.8 \times Z$ $= X \times 0.7 \times 0.2 + X \times 0.3$ (s22) $0.167 \times (1-Y) \times 0.4 \times Z / W = 0.7$	ga 0.166 $\rightarrow$ 0.10 r2,5 0.2 $\rightarrow$ 0.1796 r7,13 0.8 $\rightarrow$ 0.7857 $\mu 22$ 0.1 $\rightarrow$ 0.15657
	S19B11 both $\lambda \rightarrow$	d r3,8 = + d r7,13 = +	$X = r3,8 (0.2 < X < 1.0)$ $Y = r7,13 (0.8 < Y < 1.0)$ $0.117 \times (1-X) \times Y = 0.117 \times X + 0.033$	r3,8 0.2 $\rightarrow$ 0.32 r7,13 0.8 $\rightarrow$ 0.885
	S19B16 both $\lambda \rightarrow$	d r1,3 = + d r3,8 = +	$X = r1,3 (0.7 < X < 1.0)$ $Y = r3,8 (0.2 < Y < 1.0)$ (s9) $0.166 \times (1-X) = 0.033$ (s19) $0.166 \times X \times (1-Y) \times 0.8$ $= 0.166 \times X \times Y + 0.033$	r1,3 0.7 $\rightarrow$ 0.8012 r3,8 0.2 $\rightarrow$ 0.3066

Table 3 Substructure-based knowledge and QL-BIE adopted by BDES-S.

substructure	knowledge	QL-BIE
1 The servers whose " $\rho$ "s $\geq 0.7$ or whose " $q$ "s $> 1.0$ are may be bottleneck.		
2 	On improving a bottleneck server by increasing its " $\mu_1$ ", decrease " $\rho_1$ "s of its downstream servers whose " $\rho_1$ "s $\geq 0.7$ or whose " $q_1$ "s $> 1.0$ .	$d\rho_1 = -$ $\leftarrow d\mu_1 = +$ $d\rho_1 = -$ $\leftarrow d\rho_1 = -$
3: tandem 	Decrease $t_{1s}$ for decreasing $\rho_2$ .	$d\rho_2 = -$ $\leftarrow dt_{1s} = -$
7: loop 	Decrease $t_{1s}$ for decreasing $t_{2s}$ (output of a loop).	$d\rho_3 = -$ $\leftarrow dt_{1s} = -$
10: tandem-1 	Increase $\lambda_1$ for increasing $\lambda_2$ . (max: $\lambda_1 = \mu_1 * BL$ )	$d\lambda_2 = +$ $\leftarrow d\lambda_1 = +$
10: tandem-2 	When $q_1 \geq QL$ , increase $\mu_1$ for increasing $\lambda_2$ .	$d\lambda_2 = +$ $\leftarrow d\mu_1 = +$
13: loop 	Increase $t_{1s}$ (input of a loop) for increasing $\lambda_3$ (output of loop).	$d\lambda_3 = +$ $\leftarrow dt_{1s} = +$
14: external input 	a) When $g_b$ is modifiable, increase $g_b$ for increasing $\lambda_2$ . b) When $g_b$ is not modifiable, increase $t_{1,2}$ for increasing $\lambda_2$ .	$d\lambda_2 = +$ $\leftarrow dg_b = +$ $d\lambda_2 = +$ $\leftarrow dt_{1,2} = +$

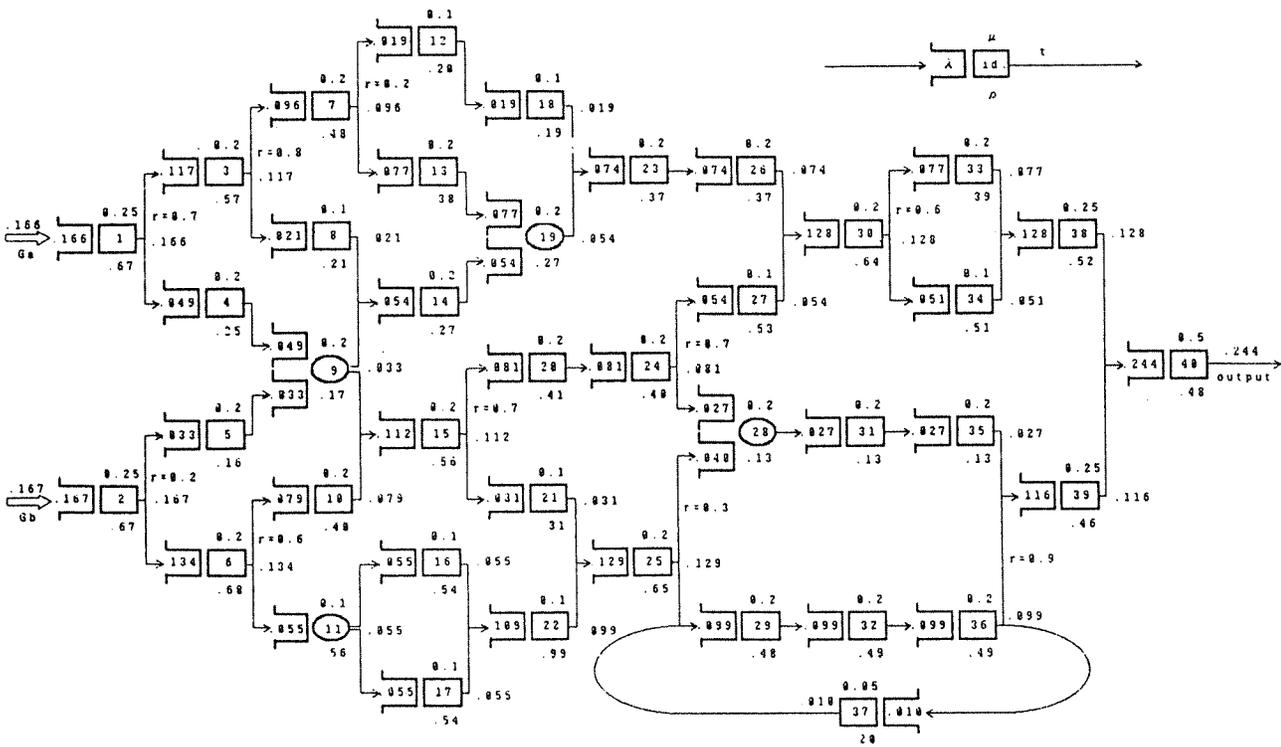


Fig. 1 Measurement for SQN5 (simulation termination time: 10,000).

```

Please input the example number: 5.
Bottleneck diagnosis for synchronized queueing
network** sqn5 ** starts.
Please input Bottleneck landmark: 0.7.
server with maximum rho (server, rho, q) (1)
s22 0.99 44.14
server whose rho >= 0.7
s22 0.99 44.14
Merge / Match server with unbalanced lambda s
(server, rho, q1, q2)
s19 0.27 111.48 0.02
s28 0.14 65.82 0.00
s9 0.17 71.21 0.07
Please input the server name to be diagnosed
:s19. (2)
s19 is merge-type-server.
s19 is judged to have 2 different lambda s.
lambda from server s13 is larger than lambda from
server s14.
Therefore, q13 is large.
There is no downstream server whose rho(q) is
large.
There is no downstream synchronized server
whose 2 lambda s are different.
For improving the bottleneck server s19 (3)
  ① decrease both lambda (3a)
  ② increase both lambda (3b)
  ③ decrease larger lambda (3c)
  ④ increase smaller lambda (3d)
**** Diagnosis process is omitted ****

qualitative improvement plans for s19 of
synchronized queueing network** sqn5
(4) decrease larger lambda
1 decrease rs7s13
2 decrease rs3s7
3 decrease rs1s3
4 decrease g[ga]
(5) increase smaller lambda
1 increase rs3s8
2 increase g[ga]
3 increase g[ga]
4 increase rs2s5
5 increase g[gb]
decrease rho[s22]
*option (3)*(4+5)
(6) decrease both lambda
1 decrease rs3s8
2 decrease g[ga]
3 decrease g[ga]
4 decrease rs2s5
decrease rho[s22]
5 decrease g[gb]
*option (3)*(4+5)
(7) increase both lambda
1 increase rs7s13
2 increase rs3s7
3 increase rs1s3
4 increase g[ga]
1 increase rs3s8
2 increase g[ga]
3 increase g[ga]
4 increase rs2s5
5 increase g[gb]
decrease rho[s22]
*option (3)*(4+5)
  
```

Fig. 2 Parameter tuning on bottleneck server s19 by BDES-S.

---

# Reasoning in Logic about Continuous Systems

---

**Benjamin J. Kuipers**  
Computer Science Department  
University of Texas at Austin  
Austin, TX 78712  
kuipers@cs.utexas.edu

**Benjamin Shults**  
Department of Mathematics  
University of Texas at Austin  
Austin, TX 78712  
bshults@math.utexas.edu

## Abstract

An intelligent agent, reasoning symbolically in a continuous world, needs to infer properties of the behaviors of continuous systems. A qualitative simulator, such as QSIM, constructs a set of possible behaviors consistent with a qualitative differential equation (QDE) and initial state. This set of behaviors is expressed as a finite tree of qualitative state descriptions. In the case of QSIM, this set is guaranteed to contain the “actual” behavior under certain circumstances. We call this property the “soundness” of QSIM. The behavior tree can then be interpreted as a model for statements in a branching-time temporal logic such as Expressive Behavior Tree Logic (EBTL), which we introduce. Because QSIM is sound, validity of an EBTL proposition (**necessarily**  $p$ ) implies the corresponding theorem about the dynamical system described by the QDE. Therefore, at least for universals, statements in temporal logic about continuous systems can be proved by qualitative simulation. This allows a hybrid reasoning system to prove such common-sense statements as “what goes up (in a constant gravitational field) must come down”, or to do such expert reasoning about dynamical systems as proving the stability of a non-linear, heterogeneous controller.

## 1 INTRODUCTION

The world is infinite and continuous. A logical proof is finite and discrete. Nonetheless we want, and reasonably expect, to use logic to draw reliable conclusions about continuous behavior in the world.

A qualitative differential equation (QDE) is a symbolic description expressing a state of incomplete knowledge of the continuous world, and is thus an abstraction of an infinite set of ordinary differential equations. Qual-

itative simulation, using an algorithm such as QSIM [Kuipers, 86], predicts the set of possible behaviors consistent with a QDE and an initial state.

The QSIM algorithm generates a tree of qualitative states representing a branching-time description of the possible behaviors of the system being described. Qualitative simulation can be viewed as proving a theorem of a very specialized form:

$$QSIM \vdash QDE \wedge QState(t_0) \rightarrow or(QBeh_1, \dots, QBeh_n)$$

where  $QDE$  is a qualitative differential equation,  $QState(t_0)$  is a qualitative description of an initial state, and each  $QBeh_i$  is a sequence of qualitative states. The QSIM Guaranteed Coverage Theorem states that this prediction describes all possible behaviors of all ordinary differential equations which are consistent with the given qualitative differential equation and initial state [Kuipers, 86]. The set of predictions may, however, include spurious predictions, those not corresponding to any real solution.

Building on the basic qualitative simulation algorithm, a variety of methods have been developed for filtering out additional classes of spurious behaviors, obtaining tractable predictions from a wider range of models while retaining the QSIM coverage guarantee. These methods include deeper types of mathematical analysis, application of partial quantitative information, appeal to carefully chosen additional assumptions, and change of the qualitative level of description [Kuipers, 93b].

Since the qualitative model and behavior tree are expressible in logic, we can show that a logical statement  $\Phi$  follows from the model by showing that it follows from the behavior tree. We do this by showing that the behavior tree can serve as a logical model for  $\Phi$ .

Since the qualitative behavior tree is a branching-time description of temporal sequences, the appropriate language for such statements  $\Phi$  is some form of modal temporal logic [Emerson, 90]. Temporal logic augments propositional logic with operators for temporal relations on time-varying truth-values, such as *sometimes*, *always*, *eventually*, and *until*. Modal logic adds

operators for relations among truth-values in alternate possible worlds (i.e., alternate behaviors), such as *necessarily* and *possibly*.

We introduce Expressive Behavior Tree Logic (EBTL) as a tool for expressing statements about QSIM behavior trees, and hence about the continuous systems they describe. EBTL is a branching time temporal logic closely related to CTL and CTL\* [Emerson, 90]. We describe an algorithm for checking the validity of an EBTL statement against a given QSIM behavior tree.

Based on the QSIM Guaranteed Coverage Theorem, we prove that for any EBTL statement  $\Phi$  which is *universal* in a sense defined below, if  $\Phi$  is true for the qualitative behavior tree predicted by QSIM, then the corresponding theorem holds for any ordinary differential equation consistent with the QDE that generated the QSIM behavior tree.

There are a number of applications of model-based reasoning that can profit from reliable inference over the set of all possible behaviors of a continuous system. Since applications – such as monitoring, diagnosis, and design – must often cope with conditions of incomplete knowledge, the ability to reason with all possible behaviors of a system described by a qualitative model is particularly valuable. A discussion of potential applications is provided in [Kuipers, 93a], and a specific application to the validation of heterogeneous controllers is provided in [Kuipers & Åström, 94] and briefly at the end of this paper.

## 2 BTL AND EBTL

Behavior Tree Logic (BTL) is a branching-time temporal logic. The theory of branching-time temporal logics is described in [Emerson, 90]. BTL is intended to be an extension and customization of Computational Tree Logic (CTL) to work with QSIM behavior trees. We are more interested in its more expressive extension, Expressive Behavior Tree Logic (EBTL), which is similar to CTL\* [Emerson, 90]. Customization is necessary because CTL only applies to infinite temporal structures. A QSIM behavior tree is finite although it may be considered to represent an infinite tree. (In this paper, when we say “a QSIM behavior tree” we are referring to the actual output of the QSIM algorithm after finite time. Therefore, although the structure may grow without bound if QSIM were allowed to run indefinitely without memory constraints, a QSIM behavior tree in our discussion is necessarily finite. However, our theorems are applied to the often infinite trees represented by these finite structures.) Therefore, we have modified the logic so that it is applicable to finite QSIM behavior trees. Our definitions are only slight modifications (or complexifications) of Emerson’s definitions of CTL and CTL\*.

## 2.1 TERMINOLOGY AND NOTATION

In this section we define the structures related to the theory of Expressive Behavior Tree Logic. QSIM behavior trees are distinguished motivational examples of these structures but EBTL is applicable to a general class of behavior trees. A QSIM behavior tree [Kuipers, 86] can easily be compared to a temporal structure in the sense defined in [Emerson, 90]. This motivates the following general definition.

**Definition 1** *In (E)BTL, a behavior tree  $M$  is an ordered triple  $\langle S, R, L \rangle$  where*

*$S$  is a set of states,*

*$R$  is a binary relation on  $S$ , and*

*$L$  is a labeling which maps each state  $s$  to an interpretation of all atomic proposition symbols in  $s$ .*

It is useful to view a behavior tree as a directed graph with node-set  $S$  and arc-set  $R$ . Without loss of generality, we can assume that a behavior tree is a *tree* (thus the name), i.e. an acyclic directed graph in which each node has at most one predecessor and there is exactly one root. The root is the only node with the property that it has no predecessor and every node is accessible from it.

It may be helpful for the reader to beware of confusing the structures associated with the logic EBTL (of which QSIM behavior trees are examples) with QSIM structures. The logic EBTL may be applied to structures other than QSIM behavior trees. When we describe the application of EBTL to QSIM trees, many details such as the unwinding method for handling cycle pointers and the labeling of states will be made more explicit. We will try to make it clear when we are referring to the QSIM structures.

We let  $\Lambda(x)$  denote the length of a finite ordered set  $x$ . A *behavior*  $x = \langle s_0, s_1, s_2, \dots \rangle$  in a behavior tree  $M$  is any path in the behavior tree which either terminates at a state with no  $R$ -successors or is infinite. In case  $x$  is of infinite length, we say  $\Lambda(x) = \infty$ . By a *path*  $x = \langle s_0, s_1, s_2, \dots \rangle$  we mean that for all  $0 \leq i < \Lambda(x) - 1$ ,  $\langle s_i, s_{i+1} \rangle \in R$ . If  $\Lambda(x) = \infty$  then by  $i \leq \Lambda(x) - 1$  we mean  $i$  is any nonnegative integer. Notice that the last state in a finite behavior  $x = \langle s_0, s_1, s_2, \dots \rangle$  is  $s_{\Lambda(x)-1}$ . In this paper we do not require  $s_0$  to be the root of the behavior tree as is customary when referring to QSIM behaviors.

For simplicity we sometimes write  $x \in M$  to mean that  $x$  is a behavior in  $M$ . We say the behavior  $x = \langle s_0, s_1, s_2, \dots \rangle$  *starts at the state*  $s_0$ , and that  $s_0$  *is the first state of*  $x$ . We will say that a behavior  $x' \in M'$  *extends* a behavior  $x = \langle s_0, s_1, \dots, s_n \rangle$  in  $M$  if the first  $n+1$  states in  $x'$  are  $\langle s_0, s_1, \dots, s_n \rangle$ . When we speak of one tree  $M$  being a subset of another tree  $M'$  if every behavior in  $M$  extends some behavior in  $M'$ . We call

a behavior *rooted* if it starts at the root of its tree.

We now describe the behavior quantifiers and the basic temporal operators on propositions. We prefer to give the reader a rough description before the formal syntax and semantics are defined. Suppose some state  $s_0$  and behavior  $x$  starting at  $s_0$  are given. The two behavior quantifiers are

- (necessarily  $p$ ), which is true if  $p$  is true of every behavior starting with  $s_0$ , and
- (possibly  $p$ ), which is true if  $p$  is true of some behavior starting at  $s_0$ .

The elementary temporal operators are (**next**  $p$ ) and (**until**  $p$   $q$ ).

- (**next**  $p$ ) is true of the behavior  $x$  if  $p$  is true of the behavior obtained from  $x$  by deleting its first state, and
- (**until**  $p$   $q$ ) is true of  $x$  if  $q$  is true of some state in  $x$  and  $p$  is true of every state preceding the first state in which  $q$  is true. We may also call this relation **strong-until**, to distinguish it from **weak-until** to be defined below.

Let it be stressed that these descriptions are only given in order to give the reader a rough idea. The exact meaning of these operators comes from the formal definition of the syntax and semantics of the logic which are in subsequent sections. We use the following abbreviations:

- (**eventually**  $p$ )  $\equiv$   
(**strong-until** true  $p$ )
- (**always**  $p$ )  $\equiv$   
(not (**eventually** (not  $p$ )))
- (**strict-precedes**  $p$   $q$ )  $\equiv$   
(and (not  $q$ )  
(**strong-until** (not (**next**  $q$ ))  $p$ ))
- (**weak-precedes**  $p$   $q$ )  $\equiv$   
(**eventually** (and  $p$  (**next** (**eventually**  $q$ ))))
- (**strong-precedes**  $p$   $q$ )  $\equiv$   
(and (**strict-precedes**  $p$   $q$ ) (**eventually**  $q$ ))
- (**weak-until**  $p$   $q$ )  $\equiv$   
(or (**strong-until**  $p$   $q$ ) (**always**  $p$ ))
- (**infinitely-often**  $p$ )  $\equiv$   
(**always** (**eventually**  $p$ ))
- (**almost-everywhere**  $p$ )  $\equiv$   
(**eventually** (**always**  $p$ ))

These last two expressions seem to presume an infinite tree. The problem of reasoning about the infinite tree represented by a finite QSIM behavior tree is discussed later.

The statement (**strict-precedes**  $p$   $q$ ) is true of a behavior if  $p$  is true in some state in the behavior and  $q$  is not true in any state previous to the first state in which  $p$  is true. The statement (**weak-precedes**  $p$   $q$ ) is true of a behavior if  $q$  is true in some state in the behavior following some state in which  $p$  is true. The statement (**strong-precedes**  $p$   $q$ ) is true of a behavior if  $q$  is true in some state in the behavior and  $p$  is true in some state previous to the first state in which  $q$  is true.

An expression in BTL is formed by an application of a behavior quantifier to a single one of the usual temporal operators: **always**, **strong-until**, **weak-until**, **next**, or **eventually**. EBTL is much more expressive because it allows boolean combinations and nestings of the behavior quantifiers and the usual temporal operators. Thus every statement in BTL is also a statement in EBTL, but "infinitely often" and "for all but finitely many" and other interesting statements can only be expressed in EBTL.

Our BTL is closely related to Emerson's CTL and our EBTL is closely related to Emerson's CTL\*. The most noticeable difference is that BTL and EBTL are applicable to finite trees as well as infinite trees. Because (E)BTL is applicable to finite trees, the temporal operator **next** may seem ambiguous. This is so because some states do not have a successor. Therefore, we must distinguish between what is called **strong-next** and **weak-next**. The statement (**strong-next**  $p$ ) is true of a behavior if the behavior has a second state and  $p$  is true of that state. The statement (**weak-next**  $p$ ) is true of a behavior if the behavior has no next state or if the behavior has a second state and  $p$  is true of it. In our discussion, we consider **next** alone to mean **weak-next**. However, the language includes both terms and the user of our program may use both.

In the following two subsections we give the formal definitions of BTL and EBTL.

## 2.2 SYNTAX

The formal definitions of the syntax for the temporal operators and behavior quantifiers informally described above are given below. These definitions follow the treatment of CTL(\*) in [Emerson, 90]. The definition of the syntax includes three state-formula generators, followed by one behavior-formula generator in the case of BTL, but followed by three behavior-formula generators in the case of EBTL. A state formula is a formula which is true or false of a state and a behavior formula is a formula which is true or false of a behavior. State formulae in both BTL and EBTL are generated by rules (S1-S3) below. The behavior formulae in BTL are generated by the rule (B0) below. The behavior formulae in EBTL are generated by rules (B1-B3) below.

**Definition 2** The syntax of EBTL is defined as follows.

- (S1) Each atomic proposition  $P$  is a state formula,
- (S2) if  $p, q$  are state formulæ then so are  $(\text{and } p \ q)$  and  $(\text{not } p)$ ,
- (S3) if  $p$  is a behavior formula then  $(\text{possibly } p)$  and  $(\text{necessarily } p)$  are state formulæ,
- (B0) if  $p, q$  are state formulæ then  $(\text{next } p)$ ,  $(\text{strong-next } p)$ ,  $(\text{strong-until } p \ q)$ ,  $(\text{always } p)$ ,  $(\text{weak-until } p \ q)$  and  $(\text{eventually } p)$  are behavior formulæ.
- (B1) each state formula is also a behavior formula,
- (B2) if  $p, q$  are behavior formulæ then so are  $(\text{and } p \ q)$  and  $(\text{not } p)$ ,
- (B3) if  $p, q$  are behavior formulæ then so are  $(\text{next } p)$ ,  $(\text{strong-next } p)$ ,  $(\text{strong-until } p \ q)$ ,  $(\text{always } p)$ ,  $(\text{weak-until } p \ q)$  and  $(\text{eventually } p)$ .

There are several things to notice here. First notice that (B0) is subsumed by (B1) and (B3). Therefore every expression in BTL is in EBTL. Also notice that the following formula is well-formed in both BTL and EBTL:  $(\text{strong-until } (\text{possibly } (\text{next } p)) (\text{necessarily } (\text{next } p)))$ . However, EBTL is strictly more expressive because, for example,  $(\text{necessarily } (\text{precedes } p \ q))$  and  $(\text{possibly } (\text{not } (\text{weak-until } p \ q)))$  are expressible in EBTL but not in BTL. We also allow the standard boolean abbreviations for or and implies.

### 2.3 SEMANTICS

The following notation is needed before the semantics of our logic can be defined. Given a behavior  $x = \langle s_0, s_1, s_2, \dots \rangle$ , for  $1 \leq i \leq \Lambda(x) - 1$  we let  $x^i$  denote the behavior  $\langle s_i, s_{i+1}, s_{i+2}, \dots \rangle$ , which is the subbehavior of  $x$  starting at  $s_i$ . I.e. it is the behavior obtained from  $x$  by deleting from  $x$  the first  $i$  states.

Notice that if  $\Lambda(x)$  is finite, then  $x^i$  is not defined for  $i > \Lambda(x) - 1$  and that  $\Lambda(x^i) = \Lambda(x) - i$ .

Now we are ready to give the semantics for the language. We write  $M, s_0 \models \Phi$  (respectively  $M, x \models \Phi$ ) to mean that state formula  $\Phi$  (respectively behavior formula  $\Phi$ ) is true in the behavior tree  $M$  at the state  $s_0$  (respectively of the behavior  $x$ ). Each item below gives the interpretation of the corresponding item in the syntax above.

**Definition 3** If  $s_0$  is a state in  $M$  and  $x = \langle s_0, s_1, \dots \rangle$  is a behavior in  $M$  starting at  $s_0$ , then we inductively define  $\models$  as follows:

- (S1)  $M, s_0 \models P$  if and only if  $P$  is true in  $L(s_0)$ ,
- (S2)  $M, s_0 \models (\text{and } p \ q)$  if and only if  $M, s_0 \models p$  and  $M, s_0 \models q$ ,  
 $M, s_0 \models (\text{not } p)$  if and only if it is not the case that  $M, s_0 \models p$ ,
- (S3)  $M, s_0 \models (\text{possibly } p)$  if and only if there is a behavior  $y$  in  $M$  starting at  $s_0$ , such that  $M, y \models p$ ,  
 $M, s_0 \models (\text{necessarily } p)$  if and only if for every behavior  $y$  in  $M$  starting at  $s_0$ ,  $M, y \models p$ .
- (B1)  $M, x \models p$  if and only if  $M, s_0 \models p$ ,
- (B2)  $M, x \models (\text{and } p \ q)$  if and only if  $M, x \models p$  and  $M, x \models q$ ,  
 $M, x \models (\text{not } p)$  if and only if it is not the case that  $M, x \models p$ ,
- (B3)  $M, x \models (\text{strong-until } p \ q)$  if and only if there is a nonnegative integer  $i \leq \Lambda(x) - 1$ , such that  $M, x^i \models q$  and for every nonnegative integer  $j < i$ ,  $M, x^j \not\models p$ ,  
 $M, x \models (\text{next } p)$  if and only if  $\Lambda(x) = 1$  or  $M, x^1 \models p$ ,  
 $M, x \models (\text{strong-next } p)$  if and only if  $\Lambda(x) > 1$  and  $M, x^1 \models p$ ,  
 $M, x \models (\text{weak-until } p \ q)$  if and only if for every nonnegative integer  $j \leq \Lambda(x) - 1$ , if for every nonnegative integer  $k \leq j$  we have  $M, x^k \models (\text{not } q)$ , then  $M, x^j \models p$ ,  
 $M, x \models (\text{always } p)$  if and only if for every nonnegative integer  $j \leq \Lambda(x) - 1$ ,  $M, x^j \models p$ ,  
 $M, x \models (\text{eventually } p)$  if and only if there is a nonnegative integer  $j \leq \Lambda(x) - 1$ , such that  $M, x^j \models p$ ,

The semantics of BTL formulæ are the same as those given above with (B3) giving the semantics of the formulæ given in (B0) of the definition of the syntax.

Now that the semantics are defined, the reader will notice that there are two definitions of the following operators: **weak-until**, **always** and **eventually**. We have given semantic definitions for these operators and we have also defined them as abbreviations of expressions involving **strong-until** and **next**. The proofs of the equivalence of these definitions are omitted because they are straight-forward but tedious manipulations of quantifiers, negation symbols, and boolean operators.

## 3 QSIM AND THE IMPLEMENTATION OF THE LOGIC

Here we consider how the logic is implemented and applied to QSIM. First, we define the relations that make finite QSIM behavior trees into possibly infinite

trees. Second, we show exactly how QSIM behavior trees and the trees they represent are used as logical models for EBTL statements. Finally, we discuss the implementation of the program which checks the truth of statements in EBTL against a QSIM behavior tree. We call the program TL for “temporal logic”.

### 3.1 QSIM AS A MODEL FOR (E)BTL

Qualitative simulation with QSIM produces a tree of *qualitative states*, linked by *successor* and *transition* relations.<sup>1</sup> A *QSIM behavior* is a path in the behavior tree, terminating at a leaf of the tree, but not necessarily starting at the root state. (This differs from normal usage.) Each state describes the *qualitative value* of each *variable* appearing in the QDE model. The qualitative value of a variable  $v$  over a state  $s$  is of the form  $\langle qmag, qdir \rangle$ , where  $qmag$  describes the magnitude of  $v$  as equal to a landmark value or in an open interval defined by two landmarks, and  $qdir$  is the sign of the derivative  $v'$  of  $v$ . By considering the qualitative values of the variables at  $s$ , and the constraints in the QDE, QSIM is able to derive a number of properties of the state, including quiescence, stability, cycles, etc. Please see [Kuipers, 86, 94] for more detailed information on QSIM.

A QSIM behavior tree is made a logical model for statements in EBTL in the following way. A *QSIM behavior tree*  $M$  is an ordered triple  $\langle S, R, L \rangle$  where the set  $S$  of states is the set of states in the output of the QSIM algorithm, the set  $R$  is the union of the QSIM successor and transition relations, and the interpretation  $L(s)$  is as follows.

For the sake of brevity, we consider only the atomic propositions associated with any QSIM state  $s$  which are of one of the following forms:

- (**status tag**) where  $tag$  is an element of  $\{\text{quiescent, stable, unstable, transition, cycle}\}$ . Such a proposition is true exactly when  $tag$  is a member of the QSIM structure  $s.status$  associated with the state  $s$ .
- (**qval  $v$  ( $qmag$   $qdir$ )**) where  $v$  is a variable of the state  $s$ ,  $qmag$  is a landmark or open interval defined by a pair of landmarks in the quantity space associated with  $v$ , and  $qdir$  is one of  $\{\text{inc, std, dec, ign}\}$ . Such a proposition is true exactly when the value of  $v$  in the state  $s$  matches the description ( $qmag$   $qdir$ ).

The expressiveness of the application of EBTL to QSIM could easily be increased without adding to the complexity by adding expressiveness to this proposi-

<sup>1</sup>There is also a *completion* relation not discussed here, that holds between an incomplete state description and a complete one consistent with it. Handling this relation is a straight-forward extension of the methods discussed here.

tional part of the language. In particular, we could allow propositional formulæ other than the two given above. For example, we could add the ability to compare the values of two variables, or to consider quantitative information about variable values.

By a *QSIM behavior* we mean a path in a QSIM behavior tree, not necessarily starting at the root state, such that the last node in the path has no  $R$ -successor. We call the state at which a behavior starts the *first state* of the behavior. In this paper, when we say “a QSIM behavior tree” we are referring to the finite output of the QSIM algorithm. That is to say, given a qualitative differential equation and allowed a finite amount of time to run, QSIM will return a finite tree. The finiteness of QSIM trees may seem to be a terrible limitation. For example, expressions such as “for all but finitely many” and “infinitely often” would apparently never be sensibly satisfied by a QSIM behavior tree. However, a QSIM behavior tree may *represent* an infinite behavior tree.

### 3.2 THE TREE REPRESENTED BY A QSIM BEHAVIOR TREE

QSIM has two ways of presenting a behavior over an infinite time-interval with a finite sequence of qualitative states. First, a fixed-point of a behavior is represented by a state with status **quiescent**. Second, repeated patterns in a behavior can be described by cycles. A *cycle state* in a QSIM behavior is one that matches a previously-generated state elsewhere in the behavior tree, so its successors are already represented by the successors of the previously-generated state. The user may select the state-matching criterion, and whether cycles must lie within a single behavior or may cross among behaviors. With respect to the tree  $\widehat{M}$  represented by a QSIM behavior tree  $M$ , the expressions (**infinitely-often  $p$** ) and (**almost-everywhere  $p$** ) have exactly the desired meaning. The solution to the problem of reasoning about the infinite tree in finite time is discussed later.

**Definition 4** *The ordered pair  $\langle s_i, s_j \rangle$  of states is in a status-bound relation if either of the following two conditions holds:*

- (1) *The proposition (**status quiescent**) is true of  $s_i$ , and  $s_i = s_j$  or*
- (2) *the proposition (**status cycle**) is true of  $s_i$ , and  $s_j$  is a successor of the previous state  $s'$  in the tree such that  $s' = s_i$ .*

If the ordered pair  $\langle s_1, s_2 \rangle$  is an element of the set of status-bound relations, then we say that  $\langle s_1, s_2 \rangle$  is a *cycle relation* if  $s_1 \neq s_2$ .

**Definition 5 (Represented Tree)** *The possibly infinite tree  $\widehat{M} = \langle \widehat{S}, \widehat{R}, \widehat{L} \rangle$ , represented by a QSIM behavior tree  $M = \langle S, R, L \rangle$ , is the tree which results by*

adding the status-bound relations to the set  $R$ . The set  $\widehat{R}$  is the union of  $R$  with the set of status-bound relations. The set  $\widehat{S}$  is the union of  $S$  and the new states which are generated first as second elements of status-bound relations and then by the unwinding process. (Cf. [Emerson, 90] for a precise definition of unwinding.) We will call the new states copies of the state in  $S$  to which they correspond. Each new state inherits the interpretation  $L(s)$  of its proposition symbols from the state of which it is a copy.

### 3.3 CLOSED BEHAVIOR TREES

In the best case, every behavior in the tree returned by QSIM terminates with a quiescent or cycle state. We will call such a tree *closed*. There are cases, however, in which QSIM does not return a closed tree regardless of how long it is allowed to run. In cases where QSIM returns a tree which is not closed, the Guaranteed Coverage Theorem does not necessarily apply. If the behavior tree  $M$  is not closed then it is possible that the actual behavior of the system is not represented in  $\widehat{M}$ .

Using the normal QSIM simulation style, creating new landmarks for critical values and applying a strong cycle-match criterion (all variables have identical landmark values), certain systems such as the damped spring have infinite behavior trees. In such cases, the QSIM algorithm cannot produce a closed behavior tree in finite time. However, by applying the *envisionment* simulation style (no new landmarks and weak cycle-match criterion), every qualitative model has a finite behavior tree. [Kuipers, 94] discusses this and a variety of methods for obtaining tractable behavior trees.

### 3.4 THE PARTIAL EXTENSION OF A BEHAVIOR TREE

Given an EBTL statement  $\Phi$  to check against a QSIM behavior tree  $M$ , we can define a *partial extension*  $\overline{M}(\Phi)$  of  $M$ ,

$$M \subseteq \overline{M}(\Phi) \subseteq \widehat{M}$$

that is finite (where  $\widehat{M}$  might not be) and enough larger than  $M$  to make the truth value of  $\widehat{M}, s_0 \models \Phi$  be the same as that of  $\overline{M}(\Phi), s_0 \models \Phi$ .

In section 3.6 we will prove that if the truth checker is given a QSIM behavior tree  $M$  and a statement  $\Phi$  in EBTL, then it returns the truth value of  $\Phi$  regarding the generally larger tree  $\widehat{M}$  represented by  $M$ . As we will see, this tree can be infinite and complex. We think of a statement in EBTL as a question which the user is asking about the given behavior tree. The user expects the program to respond with the truth value of  $\widehat{M}, s_0 \models \Phi$ , where  $\widehat{M}$  is the possibly infinite tree represented by the QSIM tree  $M$  and  $s_0$  is the root of

the tree. The program TL accomplishes this by constructing the partial extension of the given QSIM tree and checking the truth of the given expression on this larger yet still finite tree. The proof is accomplished by showing that the partial extension of the tree is large enough to decide the question  $\Phi$ .

The reader may find it helpful to examine Figure 1 for a motivation of the following definitions. The partial extension  $\overline{M}(\Phi)$  of  $M$  depends also on  $\Phi$ . It is constructed from  $M$  and  $\Phi$  by expanding  $M$  according to the structure of nestings of **until** and **next** statements in  $\Phi$ .

Let us now give the needed definitions. Recall that a QSIM behavior tree  $M$  is necessarily finite. The *until extent*,  $x(\text{until})$ , of a behavior  $x$  in  $M$  is a set of possibly truncated behaviors in  $\widehat{M}$  which extend  $x$ . The addition of these longer behaviors enlarge  $M$  exactly enough to answer properly any propositional **until** statement.

**Definition 6** *The until extent  $x(\text{until})$  of a finite behavior  $x = \langle s_0, s_1, \dots, s_n \rangle$  is the singleton set containing  $x$  unless (**status cycle**) is true of  $s_n$  in which case  $x(\text{until})$  is the set of paths  $x'$  in  $\widehat{M}$  extending  $x$  but truncated at the first state  $s \in x'$  at which the following property is satisfied:*

The **Until Property**:  $s$  is not in  $x$  and either (**status quiescent**) is true of  $s$  or (**status cycle**) is true of  $s$  and  $s$  is a copy of some previous state in  $x'$ .

It is important to understand that the until extent of a behavior  $x$  in a finite QSIM tree  $M$  is a finite set of finite behaviors. To see this we need to recall two facts. First, QSIM behavior trees are finitely branching. Second, cycle states occur only at the terminal states of a QSIM behavior tree, thus there are only finitely many cycle states in a finite QSIM tree. If some  $x' \in x(\text{until})$  were infinite, it would have to pass through infinitely many cycle states. Thus it would have to pass through one of them more than once, contradicting the Until Property. Since each behavior in  $x(\text{until})$  is finite, and  $M$  is finitely branching,  $x(\text{until})$  must be finite.

The *next extent*,  $x(\text{next})$ , of a behavior  $x$  in  $M$  is the set of possibly truncated behaviors in  $\widehat{M}$  which are sufficiently extended to answer a propositional **next** question.

**Definition 7** *The next extent  $x(\text{next})$  of a finite behavior  $x = \langle s_0, s_1, \dots, s_n \rangle$  is the set of paths  $x'$  in  $\widehat{M}$  extending  $x$  but truncated at the first state  $s \in x'$  satisfying one of the following properties: (**status cycle**) is true of  $s_n$  and  $s$  satisfies the Until Property or (**status quiescent**) is true at  $s$  and  $s$  is not in  $x$ .*

A similar argument as the one given above shows that the next extent of a finite behavior in a finite behavior tree is a finite set of finite behaviors.

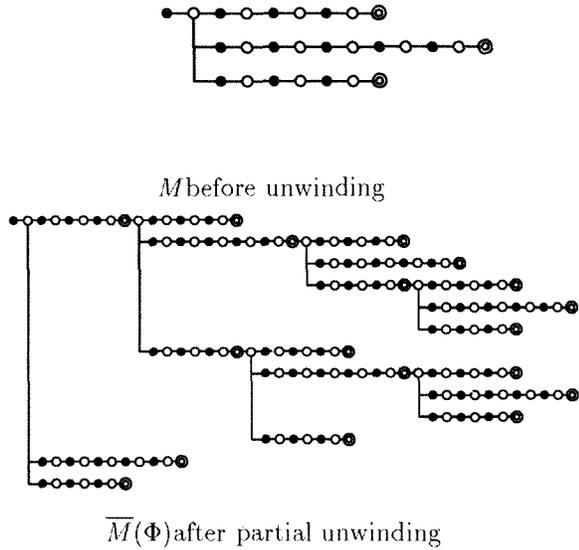


Figure 1: Partial unwinding for  $\Phi = (\text{until } p \ q)$ , along first behavior only.

Each cycle state is expanded, stopping each branch at the second occurrence of a given cycle state. The double circles represent cycle states.

**Definition 8** We define the partial extension  $\overline{M}(\Phi)$  generated by a tree  $M$  and an EBTL expression  $\Phi$  recursively as follows:

- If  $\Phi$  is a proposition then  $\overline{M}(\Phi) = M$ ,
- if  $\Phi$  is  $(\text{and } p \ q)$  then  $\overline{M}(\Phi)$  is the union of  $\overline{M}(p)$  and  $\overline{M}(q)$ .
- if  $\Phi$  is  $(\text{not } p)$ ,  $(\text{possibly } p)$  or  $(\text{necessarily } p)$ , then  $\overline{M}(\Phi) = \overline{M}(p)$ ,
- if  $\Phi$  is  $(\text{strong-until } p \ q)$  then  $\overline{M}(\Phi)$  is the union over each behavior  $x \in \overline{M}(p) \cup \overline{M}(q)$  of  $x(\text{until})$ , or
- if  $\Phi$  is  $(\text{next } p)$ , or  $(\text{strong-next } p)$ , then  $\overline{M}(\Phi)$  is the union over each behavior  $x \in \overline{M}(p)$  of  $x(\text{next})$ .

Notice that the paths in the until and next extents of a behavior in  $M$  are generally not behaviors in  $M$  or  $\widehat{M}$ . They may, however, be behaviors in  $\overline{M}(\Phi)$  for some  $\Phi$ .

Our implemented prover TL, given as inputs a QSIM tree  $M$  and an EBTL expression  $\Phi$ , returns **true** if and only if  $\widehat{M}, s \models \Phi$  where  $s$  is the root state of  $M$ . We will prove that it is enough for the truth checker to examine  $\overline{M}(\Phi)$ , which in fact is what TL does. To be specific, we have defined the partial extension of a behavior tree generated by a QSIM behavior tree  $M$  and an EBTL statement  $\Phi$  to be at least as large as the largest tree generated by TL in the process of checking

$\Phi$  on  $M$ . It, however, should be clear that this tree must be finite. This is true because each statement in EBTL is finite and each QSIM behavior tree is finite.

We will prove that given an EBTL expression  $\Phi$  and a QSIM behavior tree  $M$ , TL correctly returns the truth or falsity of  $\Phi$  in the behavior tree  $\widehat{M}$  represented by  $M$  in finite time.

We need a way of distinguishing, for a given subexpression  $\Phi_0$  of  $\Phi$ , whether  $\Phi_0$  is in the scope of a necessarily or of a possibly quantifier. The definition in the next subsection fulfills this need.

### 3.5 IMMEDIATE SCOPE

We recursively define an occurrence of  $p$  being in the *immediate scope* of a behavior quantifier as follows:

**Definition 9 (Immediate Scope)**

The occurrence of  $p$  in  $(\text{possibly } p)$  is in the immediate scope of **possibly**.

If  $(\text{and } p \ q)$ ,  $(\text{not } p)$ ,  $(\text{strong-until } p \ q)$ ,  $(\text{next } p)$ ,  $(\text{strong-next } p)$ ,  $(\text{always } p)$ , or  $(\text{eventually } p)$  occurs in the immediate scope of **possibly** then these occurrences of  $p$  and  $q$  are said to occur in the immediate scope of **possibly**.

The occurrence of any EBTL expression which can not be shown to be in the immediate scope of **possibly** by the above conditions is in the immediate scope of **necessarily**.

Other temporal operators are treated as abbreviations of expressions involving the operators mentioned above.

Consider the following example of an EBTL expression:

$(\text{and } (\text{possibly } (\text{strong-until } p \ (\text{necessarily } q))))$   
 $(\text{next } (\text{possibly } r))$

The **and** and its arguments are in the immediate scope of **necessarily** as is the occurrence of  $q$ . Also  $(\text{possibly } r)$  is in the immediate scope of **necessarily**. However, the **strong-until** statement and its arguments are in the immediate scope of **possibly**.

### 3.6 CORRECTNESS OF THE IMPLEMENTATION

Now we come to the promised proof. Because TL examines the partial extension  $\overline{M}(\Phi)$  of  $M$ , what we really need to prove is the following:

**Theorem 1** *If  $M$  is a QSIM behavior tree with root  $s$  and  $\Phi$  is an ECTL state expression, then*

$$\widehat{M}, s \models \Phi \Leftrightarrow \overline{M}(\Phi), s \models \Phi.$$

The proof goes by induction on the structure of  $\Phi$ . If  $\Phi$  is a proposition, then the theorems are obvious.

Also it is clear how to handle the booleans, i.e. we simply pass the proof on to their arguments.

If  $\Phi$  is of the form (**necessarily**  $p$ ) then we must really prove the following:  $\widehat{M}, x \models p$  for all behaviors  $x \in \widehat{M}$  if and only if  $\overline{M}(p), x' \models p$  for all behaviors  $x' \in \overline{M}(p)$ . If  $\Phi$  is of the form (**possibly**  $p$ ) then we must prove the following:  $\widehat{M}, x \models p$  for some behavior  $x \in \widehat{M}$  if and only if  $\overline{M}(p), x' \models p$  for some behavior  $x' \in \overline{M}(p)$ .

So the interesting parts of the proof will be when  $\Phi$  begins with a temporal operator within the immediate scope of either **necessarily** or **possibly**. Theorem 2 takes care of the case when a **strong-until** statement occurs in the immediate scope of **necessarily**.

**Theorem 2** *If  $\Phi$  is of the form (**strong-until**  $p$   $q$ ), then  $\widehat{M}, x \models \Phi$  for all rooted behaviors  $x \in \widehat{M}$  if and only if  $\overline{M}(\Phi), x' \models \Phi$  for every rooted behavior  $x' \in \overline{M}(\Phi)$ .*

**Proof:** ( $\Rightarrow$ ) Suppose  $\widehat{M}, x \models \Phi$  for every behavior  $x \in \widehat{M}$  starting at the root. Let  $x'$  be a behavior in  $\overline{M}(\Phi)$  starting at the root. Suppose for the sake of contradiction that for every nonnegative integer  $k \leq \Lambda(x') - 1$ , if  $\overline{M}(\Phi), x'^k \models q$ , then there is a number  $l < k$  such that  $\overline{M}(\Phi), x'^l \not\models p$ . There is a behavior  $x$  in  $\widehat{M}$  which extends  $x'$ . The existence of this behavior in  $\widehat{M}$  contradicts our hypothesis.

( $\Leftarrow$ ) Now suppose that for every behavior  $x' \in \overline{M}(\Phi)$  starting at the root

$$\overline{M}(\Phi), x' \models \Phi \quad (1)$$

Suppose for refutation that for every behavior  $x$  in  $\widehat{M}$  starting at the root, for all  $i \leq \Lambda(x) - 1$  if  $\widehat{M}, x^i \models q$ , then there is a nonnegative integer  $j < i$  such that  $\widehat{M}, x^j \not\models p$ . There are two cases to consider.

First we suppose there is a rooted behavior  $x \in \widehat{M}$  such that for all  $i \leq \Lambda(x) - 1$ ,  $\widehat{M}, x^i \not\models q$ . Let  $x'$  be a behavior in  $\overline{M}(\Phi)$  which  $x$  extends. (I.e. cut off  $x$  at its first state which satisfies the Until Property.) The existence of this  $x' \in \overline{M}(\Phi)$  contradicts our hypothesis (1).

Now suppose that there is a rooted behavior  $x \in \widehat{M}$  such that for every  $i \leq \Lambda(x) - 1$  such that  $\widehat{M}, x^i \models q$  there is a  $j < i$  such that  $\widehat{M}, x^j \not\models p$  and that such an

$i$  exists. Choose  $i$  to be the smallest number such that  $\widehat{M}, x^i \models q$ . Let  $j < i$  be such that  $\widehat{M}, x^j \not\models p$ . Let  $y$  denote the path from the root to the first state,  $s_i$  in  $x^i$ . Suppose there is a state  $s$  in  $y$  which satisfies the Until Property. If such a state exists then the path  $\langle s_0, s_1, \dots, s \rangle$  is a behavior in  $\overline{M}(\Phi)$ . The existence of this behavior contradicts the hypothesis (1). If there is no state in  $y$  satisfying the Until Property, then there is some behavior  $x'$  in  $\overline{M}(\Phi)$  which extends the path  $y$ . This behavior once again contradicts our hypothesis (1).

This completes the proof.

Theorem 3 takes care of the case when a **strong-until** statement occurs in the immediate scope of **possibly**.

**Theorem 3** *If  $\Phi$  is of the form (**strong-until**  $p$   $q$ ), then there is a rooted behavior  $x \in \widehat{M}$  such that  $\widehat{M}, x \models \Phi$  if and only if there is a rooted behavior  $x' \in \overline{M}(\Phi)$  such that  $\overline{M}(\Phi), x' \models \Phi$ .*

**Proof:** ( $\Rightarrow$ ) First, suppose there is a rooted behavior, call it  $x$ , in

$\widehat{M}$  such that  $\widehat{M}, x \models \Phi$ . There is a path  $x' \in \overline{M}(\Phi)$  which  $x$  extends. Suppose for contradiction that for all nonnegative integers  $k \leq \Lambda(x') - 1$  if  $\overline{M}(\Phi), x'^k \models q$  then there is a nonnegative integer  $l < k$  such that  $\overline{M}(\Phi), x'^l \not\models p$ .

If  $\overline{M}(\Phi), x'^k \models q$  for some  $k \leq \Lambda(x') - 1$  and  $\overline{M}(\Phi), x'^l \not\models p$  for some  $l < k$  then the same is true for  $x \in \widehat{M}$  which is a contradiction.

If there is no  $k \leq \Lambda(x') - 1$  such that  $\overline{M}(\Phi), x'^k \models q$ , then it must be the case that for every state  $t \in x', \overline{M}(\Phi), t \models p$ . Thus far, we have assumed nothing about the behavior  $x \in \widehat{M}$  except that  $\widehat{M}, x \models \Phi$ . We know that for any such behavior there is a smallest number  $i(x)$  such that  $\widehat{M}, x^{i(x)} \models q$ . Let  $i$  be the smallest of all of the  $i(x)$  ranging over behaviors  $x$  for which  $\widehat{M}, x \models \Phi$  and let  $x$  now denote the behavior corresponding to  $i$ . If there is no state,  $s$ , in the path  $y = \langle s_0, s_1, \dots, s_i \rangle$  satisfying the Until Property, then there is a behavior  $x' \in \overline{M}(\Phi)$  which extends  $y$ . In this case we are done because  $\overline{M}(\Phi), x' \models \Phi$ .

If there is a state in  $y$  which satisfies the Until Property, then we let  $s$  denote the first such state. Either (**status quiescent**) or (**status cycle**) must be true at  $s$  (or  $s$  has no successors). In the former case we are done because the path  $\langle s_0, s_1, \dots, s \rangle$  is a behavior in  $\overline{M}(\Phi)$ . In the latter case, we delete from  $y$  the state of which  $s$  is a copy and the states between it and  $s$ . What remains of the path  $y$  is again a truncated path in  $\widehat{M}$ , but a shorter path, and  $\widehat{M}, z \models \Phi$  for any path  $z \in \widehat{M}$  which extends  $y$ . But this contradicts our assumption that the state  $s_i \in x$  was the nearest

state to the root satisfying these conditions.

( $\Leftarrow$ ) Now suppose there is a behavior  $x' \in \overline{M}(\Phi)$  such that  $\overline{M}(\Phi), x' \models \Phi$ . There is a behavior  $x \in \widehat{M}$  which extends  $x'$ . Thus we are done.

This completes the proof.

Similar theorems follow for **next** expressions and the other temporal operators can be treated as abbreviations of these two.

Therefore, it is enough for the truth checker to examine only  $\overline{M}(\Phi)$  when trying to check  $\widehat{M}, s_0 \models \Phi$ . Since the until (or next) extent of a behavior in a QSIM tree starting at any state is finite and each EBTL expression is finite, the truth checker will terminate with the correct answer.

## 4 THE MAIN THEOREM

Our main theorem states that, under appropriate hypotheses, the answer that TL gives to an EBTL statement concerning a QSIM behavior tree will be true of the solution to any differential equation consistent with the qualitative differential equation which produced the QSIM behavior tree.

Before we state our main theorem we need some notation, definitions and a lemma. We define the parity of a position in an EBTL expression as follows:

### Definition 10

- The first operator in any EBTL expression given to TL is in a position of parity 0.
- If (**not**  $p$ ) occurs in a position of parity  $n \in \{0, 1\}$ , then  $p$  is in a position of parity  $n + 1 \pmod{2}$ .
- If ( $O$   $p$ ) or ( $O$   $p$   $q$ ) occurs in a position of parity  $n \in \{0, 1\}$  and  $O$  is some temporal, boolean, or modal operator other than **not**, then  $p$  and  $q$  occur in positions of parity  $n$ .

Recall that (**implies**  $p$   $q$ ) is an abbreviation of (**not** (**and**  $p$  (**not**  $q$ ))) so if (**implies**  $p$   $q$ ) occurs in a position of parity  $n \in \{0, 1\}$  then  $p$  is in a position of parity  $n + 1 \pmod{2}$  and  $q$  is in a position of parity  $n$ . This follows the use of "positive" and "negative" position in [Wang, 60].

**Definition 11** An EBTL expression  $\Phi$  is said to be universal if every occurrence of the behavior quantifier **possibly** is in a position of parity 1 and every occurrence of the behavior quantifier **necessarily** is in a position of parity 0.

With a little thought, the reader will see that if a formula is universal, then the truth checker should examine the entire tree in order to establish the truth of the formula. This is the motivation for the definition.

If  $\Phi$  is a universal formula in EBTL, then  $\Phi'$  denotes the linear-time behavior formula obtained from  $\Phi$  by deleting all occurrences of the behavior quantifiers. For example, if

$$\Phi = (\text{necessarily} \\ \text{strong-until } p \text{ (necessarily } q)),$$

then

$$\Phi' = (\text{strong-until } p \text{ } q).$$

We are now ready to proceed to the details of our main theorem. We say a real-valued function,  $u$ , satisfies a given QSIM qualitative behavior description if the qualitative description of the function matches the given qualitative behavior. The following theorem is proved in [Kuipers, 86].

**Theorem 4 (Guaranteed Coverage)** Let  $F = 0$  be an ordinary differential equation with solution  $u$ , a real valued function. Let  $C$  be a QDE with which  $F = 0$  is consistent. Let  $M$  be the QSIM behavior tree generated by the QSIM algorithm applied to  $C$ . If  $M$  is closed, then  $u$  satisfies some QSIM behavior  $x = \langle s_0, s_1, s_2, \dots \rangle$  in  $\widehat{M}$  where  $s_0$  is the root state of  $M$ .

**Theorem 5 (The Main Theorem)** Let  $\Phi$  be a universal state formula in EBTL. Let  $u$  and  $M$  be as in the hypotheses of the Guaranteed Coverage Theorem. Let  $s_0$  be the root state of  $M$ . If  $\overline{M}(\Phi), s_0 \models \Phi$ , then  $\Phi'$  is true of the qualitative description of  $u$ .

**Proof:** Suppose  $\overline{M}(\Phi), s_0 \models \Phi$ , as in the hypotheses of the theorem. By definition,  $\Phi'$  is a behavior formula. For simplicity, let us start by replacing every occurrence of the temporal operators **weak-until**, **always**, **precedes**, **strong-precedes**, **infinitely-often**, **almost-everywhere**, and **eventually** with expressions involving only the temporal operator **strong-until** and **next**. (Since  $M$  is closed, it makes no difference whether we consider **next** to be strong or weak.) This is made possible by the abbreviations on page 3. So now  $\Phi'$  is a behavior formula whose only temporal operators are **next** and **strong-until**. By the Guaranteed Coverage Theorem and the fact that  $\Phi$  is universal, it is enough to show that  $\Phi'$  is true of every behavior in  $\widehat{M}$  starting at  $s_0$ . So, by the results in section 3.6, we need only to show that  $\overline{M}(\Phi'), x \models \Phi'$  for every behavior  $x$  in  $\overline{M}(\Phi')$  starting at  $s_0$ . So let  $x$  be a behavior in  $\overline{M}(\Phi)$  starting at  $s_0$ . We will induct on the complexity of  $\Phi'$ . Unless otherwise noted, references to (S1-S3, B1-B3) refer to the definition of the semantics.

If  $\Phi'$  is an atomic proposition, then  $\Phi'$  is a state formula by (S1) of the definition of the syntax of EBTL. Since  $\Phi'$  is a propositional state formula,  $\Phi = \Phi'$  (Cf.

(S3) of the definition of the syntax of EBTL). Therefore  $\overline{M}(\Phi'), x \models \Phi'$  by hypothesis and we are done.

Suppose  $\Phi'$  is of the form (**and**  $p$   $q$ ). Then we reduce to the case of showing  $\overline{M}(p), x \models p$  and  $\overline{M}(q), x \models q$ .

Suppose  $\Phi'$  is of the form (**not**  $p$ ). Then we reduce to the case of showing that it is not the case that  $\overline{M}(p), x \models p$ .

Suppose  $\Phi'$  is of the form (**strong-until**  $p$   $q$ ). We reduce to showing that for some nonnegative integer  $j \leq \Lambda(x) - 1$  and for all nonnegative integers  $k \leq j$ ,  $\overline{M}(q), x^j \models q$  and  $\overline{M}(p), x^k \models p$ .

Suppose  $\Phi'$  is of the form (**next**  $p$ ) where  $p$  is a behavior formula. It must be the case that  $\Lambda(x) > 1$ , otherwise  $M, x \models \Phi$  could not have been true (Cf. (B3)). Thus we reduce to proving  $\overline{M}(p), x^1 \models p$ .

In each case we have reduced  $\Phi'$  to a more simple expression. The obvious induction argument on the complexity of  $\Phi'$  finishes the proof.

## 5 APPLICATIONS OF EBTL AND QSIM

EBTL may be useful any time QSIM is used. QSIM has been used to simulate controllers, human organs and disease, abstract and real physical systems, electrical circuits, population dynamics, chemical reactions, etc.

### 5.1 PROVING PROPERTIES OF CONTROLLERS

Kuipers & Åström [1994] have used TL and QSIM to prove properties of heterogeneous control laws. A heterogeneous controller is a nonlinear controller created by the composition of local control laws appropriate to different operating regions. Such a controller can be created in the presence of incomplete knowledge of the structure of the system, the boundaries of the operating regions, or even the control action to take. A heterogeneous control law can be analyzed, even in the presence of incomplete knowledge, by representing it as a qualitative differential equation and using qualitative simulation to predict the set of possible behaviors of the system. By expressing the desired guarantee as a statement in EBTL, the validity of the guarantee can be automatically checked against the set of possible behaviors. Kuipers & Åström [1994] demonstrate the design of heterogeneous controllers, and prove certain useful properties, first for a simple level controller for a water tank, and second for a highly nonlinear chemical reactor.

It should be noted that [Moon, et. al., 92] used CTL to prove a guarantee for a discrete-time control system. EBTL and QSIM make it possible to apply temporal

logic to continuous-time control systems, and indeed to dynamical systems in general.

The program TL is equally easily applied to the behavior trees output by QSIM extensions such as NSIM and Q2, which use quantitative bounding information and produce quantitative bounds on the predictions. For these applications a slight extension of the propositional part of the language is helpful. We add the ability to include numerical information in the state propositions. This added expressiveness does not add to the complexity of the algorithm.

The program can be and has been used on terminals which do not support the graphics needed to see QSIM trees. In these circumstances, the user can learn everything he may need to know about a QSIM tree by evaluating a few carefully chosen EBTL statements.

### 5.2 TL AS A DEBUGGING TOOL FOR QSIM MODELS

Because QSIM is not complete, a QSIM tree may contain behaviors which do not correspond to real behaviors. Therefore, the truth of an EBTL statement (e.g. one beginning with the quantifier **possibly**), does not imply the truth of the corresponding statement in an actual behavior. This apparent limitation, however, can be and has been used as a debugging tool. For example, if the QSIM user knows that a certain sequence of events cannot occur in a real behavior, he can use TL to find out if that sequence of events occurs in any of the behaviors in the QSIM tree. The implemented program TL allows EBTL formulæ to have side effects. Therefore, it can be used to print out the undesirable behaviors or states which satisfy a certain EBTL formula. In the actual TL code, there are features which make this process very easy.

### 5.3 EXAMPLES

We demonstrate the use of TL to ask and answer questions about two simple models: the undamped oscillator, whose behavior tree is rooted in the initial state SS; and the damped oscillator, whose behavior tree is rooted in the state DS.

**Undamped Oscillator** The simple spring conserves energy, so all behaviors are cycles, as shown by the behavior tree in figure 1. The three behaviors differ according to whether the amplitude of the oscillations passes a predefined landmark value. The queries shown demonstrate that the simple spring never becomes quiescent, always reaches a cycle state, and necessarily has an infinite sequence of events crossing  $x = 0$  in opposite directions.

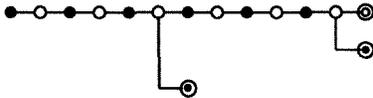
```
(TL SS '(necessarily
          (always (not (status quiescent))))))
=> T
```

```
(TL SS '(necessarily (eventually (status cycle))))
=> T

(TL SS '(necessarily
  (and (infinitely-often (qval x (0 inc)))
        (infinitely-often (qval x (0 dec)))))
=> T

(TL SS '(necessarily
  (infinitely-often
    (precedes (qval x (0 dec))
              (qval x (0 inc)))))
=> T
```

**Damped Oscillator** The damped spring loses energy. The first behavior is a cycle representing a decreasing oscillation. The second two are partial cycles followed by “nodal” convergence to quiescent states at the origin (indicated by circled dots in the behavior tree). This finite behavior tree represents an infinite family of behaviors, oscillating a finite number of half-cycles around the origin before “nodal” convergence. Each of the universal questions asked about the simple spring behavior is false of the damped spring, but the corresponding existential statements are true.



```
(TL DS '(possibly (always (not (status quiescent)))))
=> T

(TL DS '(possibly (eventually (status cycle))))
=> T

(TL DS '(possibly (eventually (status quiescent))))
=> T

(TL DS '(possibly
  (and (infinitely-often (qval x (0 inc)))
        (infinitely-often (qval x (0 dec)))))
=> T

(TL DS '(possibly
  (infinitely-often
    (precedes (qval x (0 dec))
              (qval x (0 inc)))))
=> T
```

## 6 FUTURE DIRECTIONS

QSIM and EBTL can be combined to help in the design of a QDE. One possibility is to allow EBTL formulæ as part of the input to the QSIM program. In this case, QSIM would only generate those behaviors which are models for the EBTL formulæ. I.e. QSIM would test the satisfiability of the conjunction of the EBTL formulæ. This would allow a qualitative model to be described jointly by a QDE and an EBTL description of its behavior.

The limiting case, with an EBTL specification of the desired behavior and no QDE, raises an intriguing possibility. QSIM would predict all behaviors consistent with continuity and the EBTL specifications. A recently-developed program called MISQ takes as input a set of qualitative behaviors and produces the minimal QDE capable of producing that behavior [Richards, et al, 92]. This would be useful, for example, to a controller designer who knows that he wants certain qualitative events to occur, not to occur, or to occur infinitely often. By providing this specification in the form of EBTL formulæ, this combination of EBTL, QSIM, and MISQ might be able to design the appropriate QDE model.

Work is currently being done with the goal of automatically generating natural explanations of the structures associated with QSIM. This requires the detection of certain common features in physical systems, e.g. negative feed-back loops, oscillation, etc. While EBTL is useful for many parts of this process, more expressiveness is clearly required.

In particular, it will be important to compare (not just quantify over) behaviors and states, and to compare and quantify over variables in the QDE. In some cases this can be done in EBTL, though awkwardly. It would not be enough to build EBTL on a first-order logic instead of a propositional logic, since quantification to compare behaviors, states or variables must be scoped *outside* of the modal and temporal operators. This would undoubtedly have a substantial impact on complexity.

## 7 MISCELLANY

### 7.1 COMPUTATIONAL COMPLEXITY

Checking the validity of statements in BTL is polynomial, and EBTL is exponential, in the size of the statement. However, since the statements are typically not enormous, the more important constraint is that validity checking is linear in the size of the behavior tree.

### 7.2 CODE

The code for QSIM is available via anonymous ftp at [cs.utexas.edu](http://cs.utexas.edu) in the directory `ftp/pub/qsim`. The up-to-date version of TL will be included with the release of QSIM by KR'94.

### 7.3 RELATED WORK

Related work has been done in applying temporal logics to various models. Some of the logics developed have been able to express more quantitative *time* information. Since QSIM does not express information about the “real” length of time intervals, these lan-

languages are not practicable in our situation. We specifically mention for example [Jahanian, 88]. In this paper, real time systems are modeled in the Modechart language. Statements in Real Time Logic can be checked against a Modechart model. Real Time Logic is undecidable in general but certain classes of statements are shown to be decidable. These languages are suited for time-critical systems. However, if all that is important is the *order* of events, then languages such as CTL\* are sufficiently expressive. In [Moon, 92], statements in CTL are checked against state transition graphs generated from programmable logic controller ladder diagrams. The specific application in [Moon, 92] is to chemical process control. Possibly the most work has been done in applications of temporal logics to computer processes such as parallel computing. [Emerson, 90] and [Lichtenstein, 84] are examples of such work. [Collins, 89] took an early step in the application of temporal logic to QSIM.

#### 7.4 HISTORY

In 1989, Kuipers began discussing the application of branching-time temporal logic to QSIM with David W. Franke and E. Allen Emerson. In 1990, Kuipers wrote the code on which TL is based. In 1992-93, Shults added the finite unwinding of cycle states and discovered the new theorems presented in this paper.

## 8 CONCLUSION

This paper has presented a method using modal and temporal logic to prove properties of the behavior of a continuous physical system. If the user can describe a physical system in terms of a set of qualitative constraints, then by using QSIM and TL, he or she can prove theorems about the behavior of any real system consistent with those constraints. We therefore provide a meaningful and sound interpretation for the phrase, "proof by simulation."

We expect that this link between logic-based and simulation-based inference methods will support a variety of hybrid reasoning techniques that could be of substantial value.

#### Acknowledgements

We would like to thank Daniel Clancy, Sowmya Ramachandran, Rich Mallory, Jeff Rickel and Robert Schrag for fruitful discussions.

The work of Benjamin Kuipers and the Qualitative Reasoning Group at the Artificial Intelligence Laboratory, The University of Texas at Austin is supported in part by NSF grants IRI-8904454, IRI-9017047, and IRI-9216584, and by NASA contracts NCC 2-760 and NAG 9-665.

The body of this paper was previously presented to a

different audience as [Kuipers & Shults, 1994].

#### References

- Tim Collins. A Temporal Logic for QSIM. unpublished term paper. 1989.
- E. Allen Emerson. 1990. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, (J. van Leeuwen, ed.), Elsevier Science Pub. B. V./MIT Press, 1990, pp. 995-1072.
- David W. Franke. 1991. Deriving and using descriptions of purpose. *IEEE Expert*, April 1991, pp. 41-47.
- Farnam Jahanian and Douglas A Stewart, A Method for Verifying Properties of Modechart Specifications. *Proceedings of the Real-time Systems Symposium*. Huntsville, AL December 1988.
- Benjamin J. Kuipers. 1986. Qualitative simulation. *Artificial Intelligence* **29**: 289 - 338.
- Benjamin J. Kuipers. Reasoning with qualitative models. 1993. *Artificial Intelligence* **59**: 125-132.
- Benjamin J. Kuipers. Qualitative simulation: then and now. 1993. *Artificial Intelligence* **59**: 133-140.
- Benjamin J. Kuipers. 1994. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. Cambridge, MA: MIT Press, in press.
- B. J. Kuipers and K. Åström. 1994. The composition and validation of heterogeneous control laws. *Automatica* **30**(2): 233-249.
- B. J. Kuipers and B. Shults. 1994. Reasoning in logic about continuous systems. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR-94)*, Morgan Kaufmann, San Mateo, CA.
- O. Lichtenstein & A. Pnueli. 1984. Checking that finite state concurrent programs satisfy their linear specifications. *Twelfth Annual ACM Symposium on Principles of Programming Languages*, pp. 97-107.
- I. Moon, G. J. Powers, J. R. Burch & E. M. Clarke. 1992. Automatic verification of sequential control systems using temporal logic. *AICHe Journal* **38**(1): 67-75.
- Bradley L. Richards, Ina Kraan and Benjamin J. Kuipers. 1992. Automatic abduction of qualitative models. *Proceedings of the National Conference on Artificial Intelligence (AAAI-92)*, AAAI/MIT Press, 1992.
- Hao Wang. 1960. *Toward Mechanical Mathematics*. reprinted in *Automation of Reasoning I*. ed. Jörg Siekmann and Graham Wrightson. Springer-Verlag 1983 pp. 244-264.

# Context-Dependent Causal Explanations

Maria Lee\* and Paul Compton†

\* CSIRO Division of Information Technology, Locked Bag 17, North Ryde, NSW 2113,  
Australia, E-mail: lee@syd.dit.csiro.au

† School of Computer Science and Engineering, University of New South Wales, PO Box 1,  
Kensington, NSW 2033, Australia, E-mail: compton@cs.unsw.oz.au

## Abstract

We outline a way of generating causal explanations from mathematical models. This approach is derived from the causal ordering theory of Iwasaki and Simon (Iwasaki and Simon, 1986a, 1986b; Iwasaki, 1988). Rather than produce a single causality by propagating causality from variables whose values are determined from outside the model, we allow causality to be propagated from variables within the model which are little influenced from within the model. This allows us to deal with wider range of models including systems with feedback, however, multiple causal explanations may then result. However, with propagation from the "most exogenous variables" a comparatively small number of explanation are produced which include those of interest to domain experts. We have applied this approach to large models including an environment greenhouse effect model. We suggest that the cost of a range of "plausible" models is small compared to the advantages of dealing with a wider range of model types.

## INTRODUCTION

Computer simulation of complex systems based on mathematical models has long been an area of interest. With the emergence of high performance computers, simulation has come to play an increasingly important role and current models are very large. The area now even has a name, Computational Science. Simulation is applied to engineering design, scientific development and forecasting. For example environmental modelling is an increasingly critical activity. As knowledge based systems continue to expand in scope and application and their knowledge sources continue to diversify, proper linking of KBS and large mathematical models will become increasing important (Kowalik, 1986).

The link between mathematical models and KBS is problematic because these models are mathematical. Causal reasoning is a core method of reasoning about how

physical systems work (Iwasaki, 1988). However, modern physics pays little attention to causality, and mathematics does not attempt to formalise it. One of the limits of the mathematical models is that they provide no explicit knowledge of how to perform analysis or to interpret results (Kunz *et al*, 1989). When we examine a single simulation output we cannot necessarily understand the factors involved. We have to perturb parameters or examine a range of behaviours or have an intimate knowledge of the behaviour of such mathematical equations. However, even a simple mathematical model can have very complex dynamic behaviour (May, 1976).

The interpretation of an equation or a diagram is highly context-dependent. Low-level graphical elements or abstract symbols do not have the precise meanings that words have in natural language. The symbols of  $x$  and  $y$  in  $x = y$  take on different meanings depending on the problem under consideration.

Researchers have worked on constructing causal explanations from mathematical equations. Forbus (Forbus, 1984) suggests that the causal reasoning of an equation should be fixed *a priori*. Iwasaki and Simon (Iwasaki and Simon, 1986, 1986a) assign a causal ordering to variables given only the equations and a list of which variables are *exogenous*. That is, the initial value is influenced from outside the system. Fixing the causal order a priori limits the behaviours generated, since different causal explanations are often possible.

According to Pearl and Verma (Pearl and Verma, 1991), the task of causal modelling can also be viewed as an identification game played by scientists against Nature. The notion of causality is context-dependent, which allows humans to decide on the

structure of the models and consequently process them in a different way.

## CAUSAL ORDERING THEORY

Causal Ordering (Iwasaki and Simon, 1986a, 1986b) is a technique for assigning an ordering to variables given only a set of equations and a list of which variables are exogenous. An exogenous variable is a variable that is influenced from outside the system directly and produces a change to other variables. That is it is a variable whose initial value is fixed by the user. Their approach is based on the theory of causal ordering first presented by Simon in 1952 (Simon, 1952).

The theory of causal ordering defines causal ordering as an asymmetric relation among variables in a set of simultaneous equations.

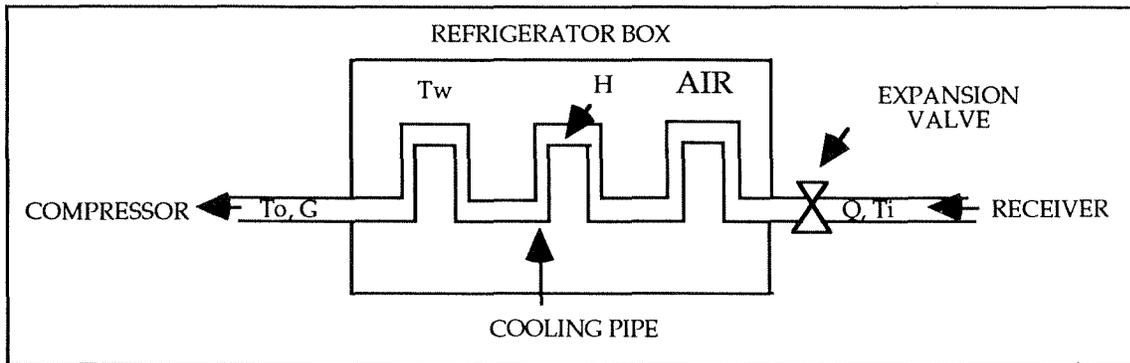


Figure 1. An Evaporator example

direction of causality based on the circumstantial knowledge.

We propose a set of heuristics to transform equations to a suitable form to produce reasonable causal explanations. Our method produces a number of explanations. We use some simple heuristics to produce likely explanations from the way people seem to normally construct models. We further use information from the user and other sources to decide which particular models are of interest. In essence we find out the same sort of causality as the method of Iwasaki and Simon (Iwasaki and Simon, 1986a, 1986b). However in order to find out causality for a wide range of models, we have to use different assumptions about the

Establishing a causal ordering involves finding subsets of variables whose values can be computed independently of the remaining variables and then using those values to reduce the structure to a smaller set of equations containing only the remaining variables. We illustrate the causal ordering procedure by applying it to the evaporator example shown in Figure 1 (adapted from (Iwasaki, 1986a)).

The system is modelled by the equations of Figure 2a. The equations have the following interpretation (the constants are  $c_i$ 's): (1) The rate of heat gained by the refrigerant,  $H$ , is proportional to the temperature difference.  $T_c$  is the condensing temperature and  $T_w$  is the temperature in the chamber. (2) The sum

of the heat absorbed,  $H$ , and the energy of the incoming fluid is the energy of the outgoing fluid, where  $G$  is the ratio of vapour to total mass in the outgoing refrigerant,  $T_i$  and  $T_o$  are the temperatures of incoming and outgoing refrigerant. (3) The condensing temperature of the refrigerant is a monotonically increasing function ( $f$ ) of the pressure,  $P$ . (4) The output temperature of the refrigerant is equal to the condensing temperature in the refrigerator chamber.

The causal ordering procedure assigns causal dependencies between variables by propagation through *self-contained* equations (Figure 2a-2d). Self-contained equations are a system of  $n$  equations with exactly  $n$  unknowns. Each matrix element is either blank or marked as a "1". A mark in

shows how causation propagates to other variables. By substituting the value for all the occurrences of variables, a new self-contained structure is obtained, until there are no more self-contained subsets. Figure 2b - 2c show the derived structures of higher orders. The variable in the minimal complete subset of the matrix is circled in the self-contained matrix. The final causal structure is shown in figure 2e.

In order for causal ordering to produce a "correct" causal structure, each equation must be a *structural equation*, i.e. it represents a conceptually distinct *mechanism* in the system. The term "mechanism" identifies physical processes as a kind of law and each equation is assigned to one mechanism. Iwasaki states that unfortunately there is no simple way to identify that an equation is structural

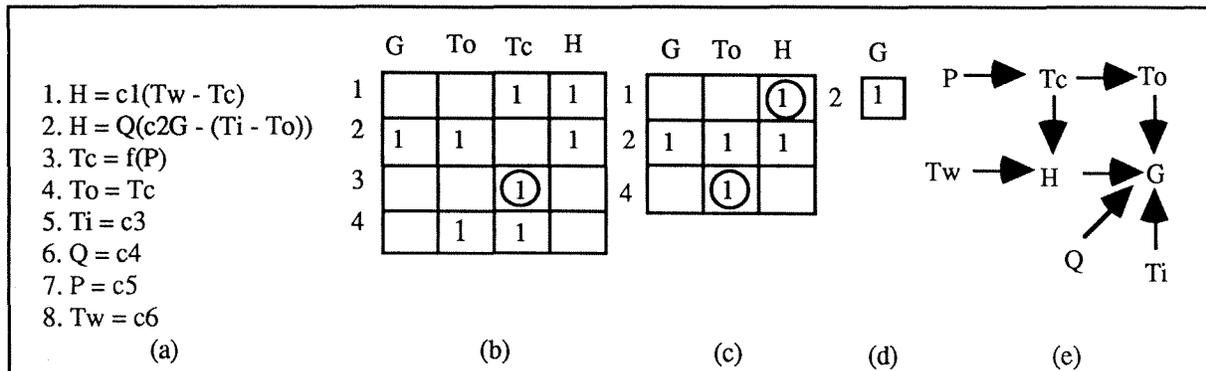


Figure 2. Equations for evaporator and the derived structure of causal ordering

row  $i$  column  $j$  means that variable  $x_j$  appears in equation  $i$ . Each row can have one or more marks. In order to make the system self-contained, the causal ordering needs four additional assumptions, in the form of additional equations. The four additional equations are:

$$T_i = c_3, \quad (5)$$

$$Q = c_4, \quad (6)$$

$$P = c_5, \quad (7)$$

$$T_w = c_6. \quad (8)$$

Each additional equation defines an exogenous variable, which provides a causal input to the phenomenon and is external to the evaporator. The causal ordering is derived from these exogenous variables and

(Iwasaki, 1986a, 1986b). Causal ordering assumes that equations used in the model are structural equations, and does not provide a method for transforming equations to structural equations. The causal ordering theory requires a self-contained structure to describe a system. To make a system self-contained and to assign exogenous variables relies upon an expert's experience and general knowledge of the model (Top and Akkermans, 1991). de Kleer and Brown (de Kleer and Brown, 1986) point out that the method of causal ordering specifies the same ordering for all behaviours. This is problematic as many systems have multiple modes of functioning, each characterised by its own distinct causal interaction. Iwasaki and Simon (Iwasaki and Simon, 1993) declare that the causal ordering theory is not

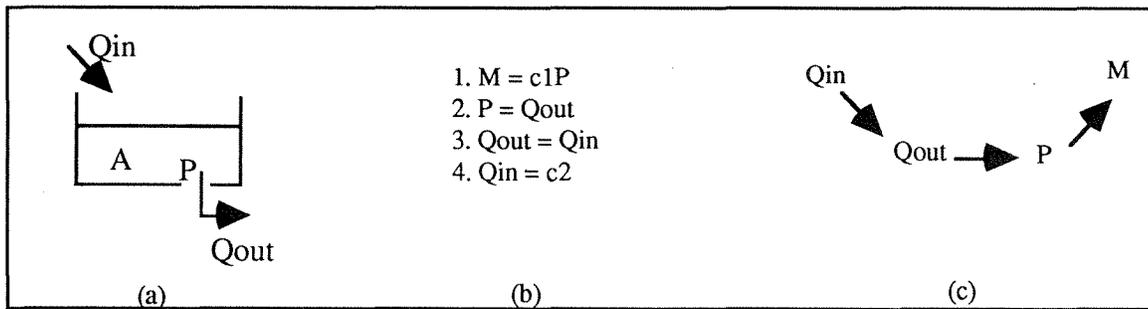
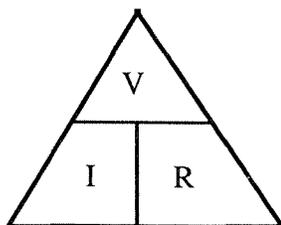


Figure 3. A steady State Bathtub Example

sufficiently developed to interpret all possible causal directions. In particular causal ordering cannot deal with feedback. The following section explains the problem of the causal ordering.

### CAUSAL STABILITY AND CAUSAL CONSISTENCY

Electrical engineers use Ohm's law, and Kirchhoff's voltage and current laws to describe the fundamental relations among voltage, current and resistance in a circuit. The laws are presented as algebraic equations, which can be manipulated (e.g.  $V = IR$ ,  $I = V/R$ ,  $R = V/I$ ). The equation  $V = IR$  represents electrical conduction in a resistor, where a voltage  $V$  volts produces a current of  $I$  amps through a resistor  $R$  ohms. Depending on the context, the equation can be causally explained as either the voltage  $V$  is causally dependent on current  $I$  and resistor  $R$  or current  $I$  on voltage  $V$  and resistance  $R$ . The third alternative, resistor  $R$  is causally dependent on voltage  $V$  and current  $I$ , does not make sense (Nayak, 1992). There is no set way of looking at it, engineers must be able to think about it in all possible ways, but only some of which make sense. Sometimes people use the following triangle to help remember the three forms of the formula:



No one form of this equation is used more than the others.

White and Frederiksen (White and Frederiksen, 1990) state that the problem-solving process that students are taught does not necessarily facilitate an understanding of the physical system under study. Hence their view that qualitative theories are not consistent concerning basic causal relations between voltage, current, and resistance. They argue that our mental models should be consistent in the assumed direction of causality among resistance, voltage, and current in a circuit example. However many mental models have no mapping to the physical world - hence the mental models won't have the same sort of causality (in contrast to many biological models where the causality comes first (Feldman and Compton, 1989)). We state intuitively that adding water ( $Q_{in}$ ) to a bathtub (Figure 3a), increases the mass ( $M$ ) of water and increases the pressure ( $P$ ), which in turn increases the output flow rate ( $Q_{out}$ ). However, there are a lot of things in the physical world which are NOT intuitive - and are often counter-intuitive. The causal structure of the steady state bathtub may seem counter-intuitive (figure 3c). It shows that the output flow rate directly depends on the input flow rate, the pressure depends on the output flow rate, and the mass of water depends on the pressure. The idea of current leading voltage is another counter-intuitive example.

Skorstad (Skorstad, 1992) states that one of the limitations of the causal ordering theory of Iwasaki and Simon (Iwasaki and Simon, 1986, 1986a) is *context sensitivity*. He argued that the causal dependencies produced by the causal ordering theory may change depending on the context or scenario in which the underlying physical system

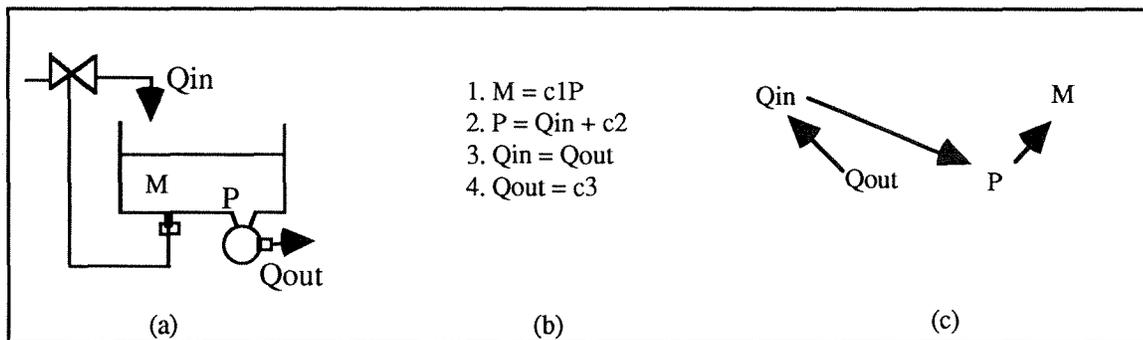


Figure 4. An example of Bathtub drain attached to a pump.

operates. Causal ordering uses a set of exogenous variables to place a system in different situations, which may change the interactions between the system and its environment. Thus, this restricts the qualitative modeller by providing a fixed causal interpretation.

Skorstad (Skorstad, 1992) defines the meaning of *causal stability* "A set of algebraic equations at a particular modelling viewpoint is causally stable if and only if its causal ordering is invariant with respect to its scenario space. Such a set of equations is unidirectional with respect to the modelling viewpoint". A modelling viewpoint means that the modeller makes decisions about ontology, perspective, and assumptions when conceptualising the phenomenon. A scenario space is a set of possible situations which are consistent with the equations.

Figure 4 shows a bathtub in a steady state where a bathtub drain is attached to a pump and the input stream is attached to a control valve (adapted from (Skorstad, 1992)). Skorstad notes that the context sensitive equations in the bathtub example are:

$$Q_{in} = Q_{out}, \quad (\text{equation 3 of figure 4b})$$

$$Q_{out} = Q_{in}. \quad (\text{equation 3 of figure 3b})$$

The causal dependency of the above equations changes depending on the circumstances. In the simple scenario of figure 3a, the output flow rate  $Q_{out}$  is causally dependent on the input flow rate  $Q_{in}$ . However, in figure 4a the output flow rate  $Q_{out}$  has become exogenous. The input stream is attached to a control valve, thus the input flow rate is no longer independent of the system and cannot be treated as exogenous. Skorstad argued that if an

equation is unidirectional with respect to the modelling viewpoint, then the equation is causal stable equation, such as:

$$P = c_1M. \quad (\text{equation 1 of figure 1b and 2b})$$

de Kleer and Brown (de Kleer and Brown, 1986) state that ambiguity is the single key advantage in qualitative causal analysis. It is not necessary to have a unique solution in the  $n$  independent equations with  $n$  unknowns. In fact, unique solutions occur only rarely. Thus each solution potentially reflects a different global functioning with a distinct causality. Most systems are indeterminate. Therefore qualitative causal reasoning should be able to interpret all its possible behaviours. This is crucial for using the model to explain a physical system's operation.

## CONTEXT-DEPENDENT CAUSAL EXPLANATIONS

In this section, we propose an approach to overcome some of the limitations of the causal ordering theory. We use equations that are a finite set of simultaneous equations and are from a mathematical model that describes the dynamic behaviour of the system. We identify variable dependencies first, then restructure equations to be asymmetric causal equation. We use the term "asymmetric causal equation" instead of "structural equation", because a structural equation should express the "real" causality; our equations are structural-like, but express "reasonable causality" (Lee *et al*, 1992a, 1992b). Causality can then be explicitly represented in asymmetric causal equations. If the dependency of a variable can not be fully

specified, then multiple plausible causal behaviours are generated.

An asymmetric causal equation is an equation which can be understood as containing independent and dependent variables. This asymmetry manifests itself in that variables on the LHS are dependent on the variables on the RHS. Hence:

- The output variable appears on the LHS of the equation, it is the variable whose behaviour is of interest.
- The input or independent variables appear on the RHS of the equation in the model.
- Dependent variable occurring once and only once on the LHS of the equation in the model.
- For a differential equation, the

important variables will be used frequently in the model, while other variables are added to fill in the gaps. In contrast to the causal ordering theory by propagating from exogenous, we deal with the causal influences from these "lesser" variables or "least caused" variables first. Once the greatest number of least occurring variable are chosen. The equation is manipulated so that the least occurring variables are on the RHS in the model. The remaining of the equations are then manipulated to give an asymmetric causal form. The first equation is now considered fixed and the process repeated for the rest of the equations etc.

The identification of appropriate causal explanations from equations is highly dependent on the problem under consideration. As the need for second-generation expert systems are to express

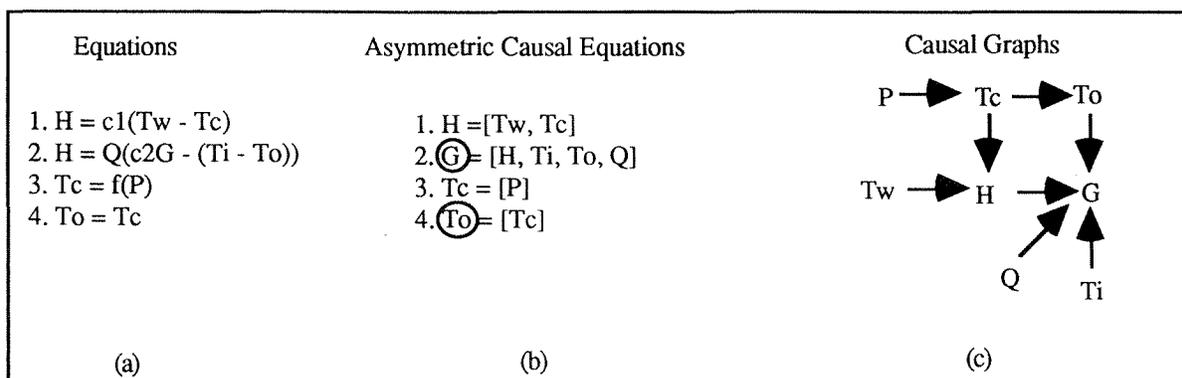


Figure 5. Equations for evaporator and the derived structure of our method.

derivative is on the LHS of the equation, with only one derivative in each equation.

To reduce the number of causal alternatives in feedback system loops, an important heuristic is to reorganise equations by propagating causality from the parameters which are "the least caused". Those variables are independent variables, which are set by the user. We look rather for the "most" independent variable. We start from the equation where there are the greatest number of such variables. We hypothesise that this has something to do with parsimony in scientific explanation. Model builders often want to discover (create) the smallest number of entities and causal connections to explain the behaviour of a system. More

how the things work; how different mechanisms interact; and to *explain* evidence in terms of structure and behaviour (Kuipers and Williams, 1988). In order to explain the evaporator internal behaviour of the total mass of the outgoing refrigerant, G, and the outgoing temperature, To, our system propagate variables within the equations, which come from an evaporator mathematical model (figure 5a), and generates the asymmetric causal equations (figure 5b) of the evaporator model. In figure 5b, circles show the variables of interests. Once the system constructs this asymmetric causal structure for the model, the causal graph is from the RHS of variables direct to the LHS of variables (figure 5c).

In order to express the different phenomena interact at pressure within chamber, P, and the total mass of outgoing refrigerant, G, multiple causal asymmetric causal equations are generated (figure 6b). Based on the multiple asymmetric causal equations, the equations could produce all plausible causal directions. In this evaporator example, the context sensitive equations are:

$$T_o = T_c, \quad (\text{equation 4 of figure 6.2a})$$

$$T_c = T_o. \quad (\text{equation 4 of figure 6.2b})$$

Its causal dependency varies depending on its situational context. In figure 6.2a, the condensing temperature is shown to be causally dependent on the temperature of the outgoing refrigerant. However, in figure

We impose the restriction that a caused or dependent variable must appear only once on the LHS of the equation. That is, each equation expresses all the causal influences on a particular parameter. Thus once the dependent variable is determined, the causal relationships within the equation can be fully specified. In a feedback system, sometimes it is difficult to identify the dependency of variables, then we apply our causal heuristic. Since the causal ordering limits the equations on a self-contained structure, i.e  $n$  equations with  $n$  unknowns, this restricts the causal ordering theory in finding a unique solution for the model. Thus if the self-contained structure could not maintain during the causality constructing process, the implementation of causal

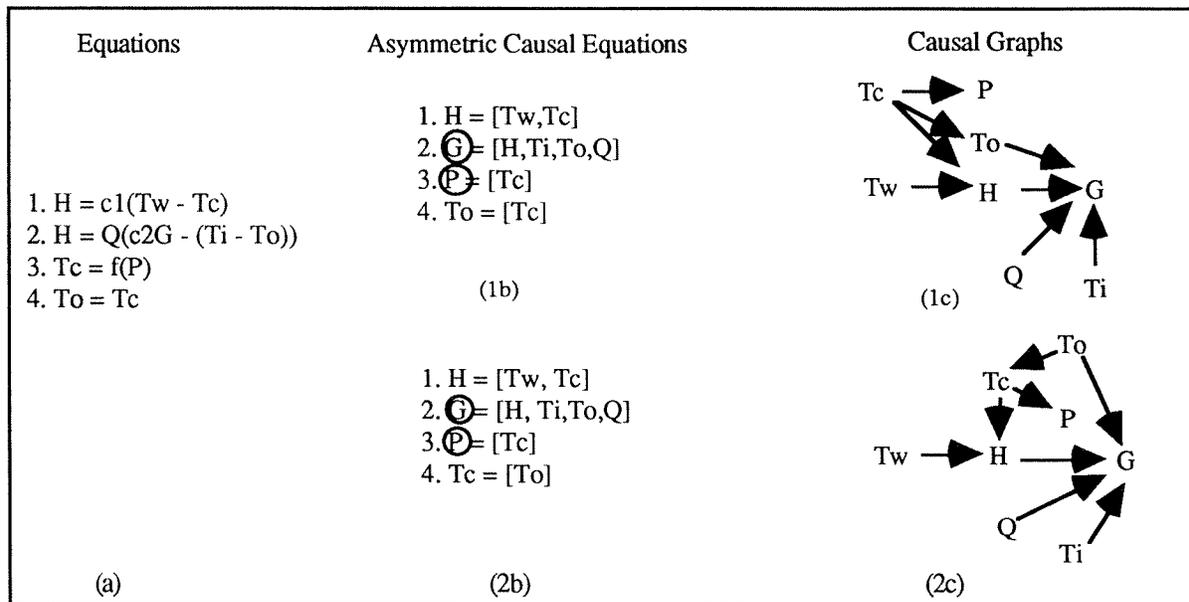


Figure 6. An example of generating multiple causal asymmetric causal equations.

6.2b the temperature of the outgoing refrigerant is causally dependent on the condensing temperature. According to the causal stability definition by Skorstad (Skorstad, 1992), the equation of the evaporator:

$$H = [T_w, T_c] \quad (\text{equation 1 of figure 5b, 6.2a and 6.2b})$$

is causally stable or unidirectional, which holds in any scenario that might be encountered.

ordering eventually ceased. If there is no minimal complete subset within equations, there is no substitution between variables in the causal ordering theory. The self-contained structure is not only limited multiple possible causalities but also restricted causal behaviour generation in feedback loop systems.

Further, our method is able to be used for model revision. It has the capabilities of allowing the users to make their hypotheses, we then backtrack through the proposed causal graph to construct a new set of

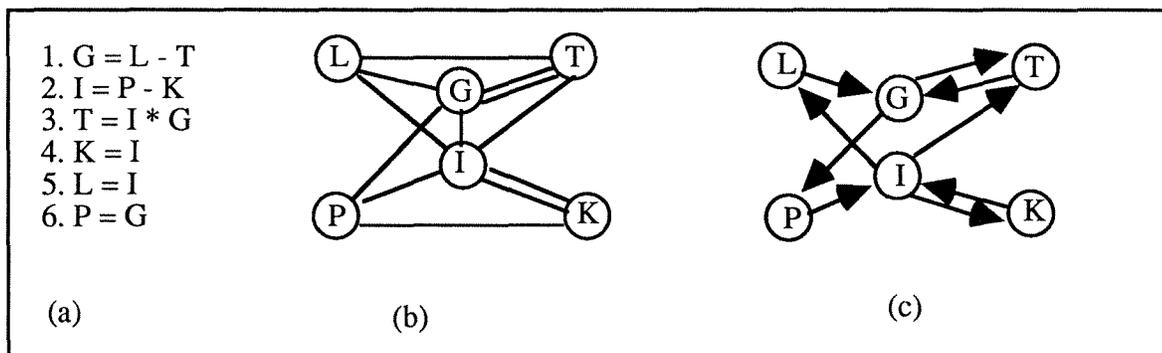


Figure 7. Equations with it's undirected and a minimal directed spanning

asymmetric causal equations. Thus the user can check the consistency of the hypothesis.

### The Algorithm

We present the basic algorithm that formulates a set of equations as asymmetric causal equations in an efficient way. The algorithm delivers all possible causal models if the set of equations together with the causality information are consistent, otherwise it generates appropriate error messages and stops. The interpretation of the causality information is up to the user. Unlike the causal ordering approach, this algorithm can be used in a system with feedback (Lee *et al* , 1992a).

```

procedure genAsyCausalEqu
  % MultiSol is a stack of storing all plausible
  equation models
  % FoundEqu is the equations with great number of
  least occurring variables
  TotalVar := empty; InputDevices :=
  empty; MultiSol := empty; FoundEqu := empty;
  PUSH all the variables occurring in the set of
  equations INTO TotalVar
  while TotalVar is not empty do
    search (least-occurrence-variables) and put
    into InputDevices
    for all InputDevices do
      search (greatest-number-of-least-
      occurring-variables-in-a-equ) and
      put into FoundEqu
      if no more FoundEqu then stop
      for all equations in the FoundEqu do
        set the inputDevices on
        RHS and pop theinputDevices
        if there are more then one
        emaining variable on the
        equation then
          save all the possible situation
          on MultiSol
  
```

```

else put it to LHS and push the
variable to Inputdevices
if (this conflicts with causality
information) then
  backtrack to next solution
else "model inconsistent" stop
  
```

```

end
end
end
  
```

### Complexity

In order to determine the order of magnitude of the time-complexity of the algorithm, we represent a set of abstract equations (figure 7(a)) together with the set of variables in an undirected spanning tree (figure 7(b)). The vertex represents the variable and the edges are the connection between vertices. In order to detect the edges, each vertex of the undirected graph needs to be visited and the edges incident upon each visited vertex needs to be directed. Visiting each vertex of a graph is equivalent to obtaining its spanning tree. A spanning tree is any tree consisting solely of edges in a graph  $G$  and including all vertices of  $G$ . Thus the complexity of our algorithm is comparable with the Kruskal algorithm (Horowitz and Sahni, 1976) for obtaining the minimal spanning tree, a spanning tree with minimum cost. Figure 7(c) show the minimal directed spanning tree of the model. Although our algorithm does not look for a lowest-cost edge, we search for the greatest number of least occurring variables for consistency checking. Hence the complexity of our algorithm seems to be  $O(n \log n)$  where  $n$  is the number of edges. However, if the users provide more causal information, its complexity will be significantly reduced.

## DYNAMIC STRUCTURE

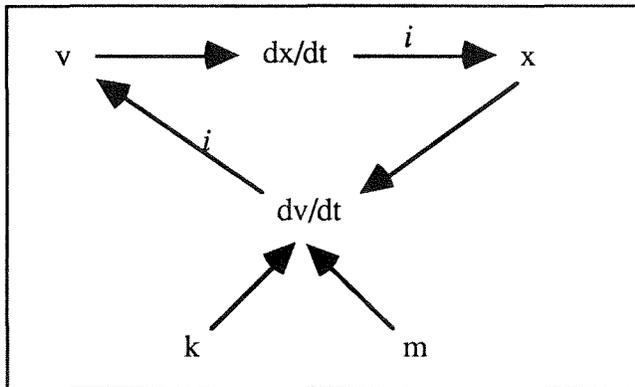


Figure 8. The causal graph of the spring-mass system.

Our approach can be used to represent systems of equations that are of a high order. We reduce higher order system to first order as first order changes with time are reasoned about causality. It seems that normal reasoning about causality is first order. People reason primarily about effects over time.

Consider the following differential equations of a spring-mass system:

$$dx^2/dt^2 + (k/m) * x = 0 \quad (1)$$

where

x = displacement,  
k = Hooke's spring constant,  
and m = mass.

Solving for  $dx^2/dt^2$  yields:

$$dx^2/dt^2 = -(k/m) * x$$

Velocity (v) is the first derivative of displacement (i.e.  $v = dx/dt$ ). We can replace the second order differential,  $dx^2/dt^2$ , with a first order differential in terms of v:

$$dv/dt = -(k/m) * x$$

Now there are two first order differential equations:

$$dx/dt = v \quad (1)$$

$$dv/dt = -(k/m) * x \quad (2)$$

The causal relations are explicitly represented on the above equations such that the variables on the LHS are dependent on the variables on the RHS. The illustration below represents the causal diagram corresponding to the spring-mass system.

In figure 8, an integration link, which is an edge connecting a derivative of a variable to the variable itself, is marked by *i*. The rate of change of v is determined by x, k and m.

## DISCUSSION

Iwasaki and Simon in the retrospective on "Causality in device behaviour" (Iwasaki and Simon, 1993) state that the causal ordering theory was not sufficiently developed in that equations are not able to interpret in all possible causal direction; that the theory does not show how to formulate equation models; that the theory defined causality sometimes are not consistent with the underlying perceived causal direction in dynamic systems.

This paper is aimed at addressing the first point that our proposed theory could interpret all plausible situations that are consistent with the equations, subject to the underlying modelling viewpoint. Our method is not only to identify equations, which are context sensitive depending on the circumstances but also uncover equations, which are causally stable or unidirectional. The causally stable algebraic equations could provide the qualitative model builder with a unique qualitative component which holds in any scenario that might be encountered.

Model formulation is a difficult problem in qualitative physics (Forbus, 1984; Falkenhainer and Forbus, 1991; Weld, 1990). In order to successfully produce causal relations that reflect our intuitive perception in the causal ordering theory, the equations must be structural. Structural equations represent conceptually distinct mechanisms in the system being modelled. However, deciding which equations are structural in a given situation is an essential problem of model formulation. Iwasaki

(Iwasaki 1988; Iwasaki and Simon, 1993) points out that there is no simple way to identify that an equation is structural. However, our proposed method seems to have identified a structural equation and to assign a direct relationship to physical components of the equation by using independent and dependent variables. The method parses the defined independent and dependent variables within equations and reconstructs the equations to be asymmetric causal equations. We use the term "asymmetric causal equation" instead of "structural equation", because a structural equation should express the "real" causality; our equation are structural-like, but express "reasonable causality" (Lee *et al*, 1992a, 1992b). Causality then can be explicitly represented in the asymmetric causal equations. We will explore this issue more and address it later as we haven't discovered from where a mathematical model comes from and from a cognitive point of view, how a model builder constructs a mathematical equations.

In terms of the causal relations in dynamic systems, our proposed method and the causal ordering theory require the differential equations in the model to be in a *canonical form*, where the derivative is on the LHS of the equation and with only one derivative in each equation (Iwasaki, 1988; Lee *et al*, 1992a). The direction of causality is from the variables on the RHS to the derivate on the LHS. However, sometimes in the dynamic physical systems where a change in a quantity is perceived as the cause of some other quantity, such as the Faraday's law of induction,  $E = -d\Phi_B/dt$ , where a change in magnetic flux ( $d\Phi_B/dt$ ) produces electromagnetic force (E), but not vice versa (Iwasaki and Simon, 1993). Also, in order to represent a higher order differential equation, our approach reduces higher order to first order. We do so because most higher order equations are derived from first order equations and causality seems to be explicitly represented in the first order equations. However, in some dynamic systems, such as an equation to describe the bend of a beam:

$$\partial^4 V / \partial X^4 - \partial^4 U / \partial Y^4 = 0,$$

it is difficult to reduce higher order equations to first order equations.

We have successfully tested our proposed method on more than 20 mathematical models, which include all of those in the relevant literature and some greenhouse effect models, i.e. a very large Global Energy Model (Edmonds and Reilly, 1983) with 41 equations and 73 variables. In all cases the method discovers the "correct" causality. However, equations in mathematical models do not model any original causality. We need to further investigate what class of equations (if any) have implicit causality and what classes of equations do not. Our heuristic does manage to recapture the original causality where relevant, or at least a reasonable causality where the equation was not based on an initial causal model. We attempt to understand on what classes of mathematical models this heuristic works or in what way causality is implicit in these models. Also, we need to assign causal effects (+ or - signs) in the causal directions. If, for example, air temperature increases as solar radiation increases, then the causal link between the two is positive (+) or proportional. Conversely, if the level of water in a lake decreases as solar radiation increases, the causal link between the two is negative (-).

## CONCLUSION

Decision support systems may require the use of existing complex mathematical models. It is desirable to reduce the equations of such a model to an explanatory causal form to support decision making. We have shown that fixing causal dependencies in a specific context is extremely limiting to behaviour generation. An asymmetric causal explanation approach has been proposed to generate causal knowledge in context. We have shown that it is possible to support the model builder's problem solving by generating all the plausible causal directions in a physical system. Our approach has overcome some limitations of the causal ordering theory in a feedback system. We have presented the algorithm that formulates equations as asymmetric causal equations.

This approach also applies to temporal knowledge in a dynamic system.

## REFERENCES

Edmonds, J. and Reilly, J. (1983). A Long-Term Global Energy-Economic Model of Carbon Dioxide Release from Fossil Fuel Use, in *Energy Economics*, April, 1983.

Falkenhainer, B. and Forbus, K.D. (1991). Compositional modelling: finding the right model for the job, *Artificial Intelligence* 51:95-143.

Feldman, B., Compton, P., and Smythe, G. (1989). Hypothesis testing: an appropriate task for knowledge-based systems, in *Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada.

Forbus, K. (1984). Qualitative Process Theory, *Artificial Intelligence*, 24:85-168.

Horowitz, E. and Sahni, S. (1976). *Fundamentals of Data Structures*, Computer and Software Engineering Series.

Iwasaki, Y. and Simon, H. (1986a). Causality in Device Behaviour, *Artificial Intelligence* 29: 1-33.

Iwasaki, Y. and Simon, H. (1986b). Theories of Causal Ordering: Reply to de Kleer and Brown, *Artificial Intelligence* 29: 73-117.

Iwasaki, Y. (1988). Model Based Reasoning of Device Behaviour with Causal Ordering, a PhD Thesis from Carnegie Mellon University.

Iwasaki, Y. and Simon, H. (1993). Retrospective on "Causality in device behaviour", *Artificial Intelligence* 59: 141-146.

Kowalik, J.S. (1986). Coupling Symbolic and Numerical Computing in Expert Systems, edited by Kowalik, J.S., North-Holland, Elsevier Science Publishers, B.V.

Kuipers, B. and Williams, B. (1988). AAAI-88 Qualitative Simulation and Causal Models Tutorial program.

Kunz, J., Stelzner, M. and William, M. (1989). From Classic Expert Systems to Models: Introduction to a Methodology for Building Model-Based Systems, in *Topics in Expert System Design*, Elsevier Science Publishers.

Lee, M., Compton, P. and Jansen, B. (1992a). Modelling with Context-Dependent Causality, in the Proceedings of JKAW'92, pp357-369.

Lee, M., Compton, P. and Jansen, B. (1992b). Causal Explanation From Mathematical Models, in the Proceedings of AI'92, pp38-43.

May, R. (1976). Simple Mathematical Models with very Complicated Dynamics, in *Nature* vol 261.

Nayak, P. (1992). Causal Approximations, in the proceedings of AAAI-92, pp 703-709.

Pearl, J. and Verma, T.S. (1991). A Theory of Inferred Causation, in Allen, J.A. et al (Eds) *Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann.

Simon, H. (1952). On the Definition of the Causal Relation, *Journal of Philosophy* 49:517-528.

Skorstad, G. (1992). Finding Stable Causal Interpretations of Equations, in *Recent advances in Qualitative Physics, Faltings and Struss (Eds)*, MIT Press, pp399-413.

Top, J. and Akkermans, H. (1991). Computational and Physical Causality, in the proceedings of IJCAI-91, pp1171-1176.

White, B. and Frederiksen, J. (1990). Causal Model Progressions as a Foundation for Intelligent Learning Environments, *Artificial Intelligence* 42: pp99-157.

Weld, D.S. (1990). Approximation reformulations, in proceedings AAAI-90, pp407-412.

# Qualitative Reasoning with Spatially Distributed Parameters

Monika Lundell \*

Artificial Intelligence Laboratory  
Computer Science Department  
Swiss Federal Institute of Technology  
IN-Ecublens, 1015 Lausanne, Switzerland  
E-mail: lundell@lia.di.epfl.ch

## Abstract

This paper presents work on modelling the qualitative behaviour of physical systems of spatially distributed parameters. The distribution of each parameter is given as a set of observation points. A metric diagram is constructed by defining a connectivity structure on the point set. The metric diagram is used to construct a topological map that represents the distribution of the parameter as a pattern of contiguous regions. The regions represent values in the parameter's quantity space, which is a discretization of its value domain. Topological combinations of parameter distributions are used to infer the distributions of non-observed parameters according to models of parameter correspondences, e.g. qualitative versions of equations. The spatial evolution of the system is inferred by matching the scenario's parameter patterns against modelled patterns of physical processes. The approach is suitable for modelling common-sense reasoning in the natural sciences, e.g. meteorology, agriculture, climate studies and natural resource management.

## 1 Introduction

Qualitative models of physical systems often focus on describing how various parameters will evolve in *time*. The work described in this paper is also concerned with how parameters evolve in *space*. We present work on modelling the qualitative behaviour of physical systems of interacting spatially distributed parameters.

A *distributed* parameter is one that takes on different values at different points in space as well as in time. Many parameters describing the physical world can be

modelled as distributed, e.g. temperature, colour, vegetation type, etc.

The human part of modern weather prediction is a good example of the kind of common-sense reasoning about spatially distributed physical systems we want to model. The role of the meteorologist is to analyze, understand and, if possible, predict the behaviour of the spatially distributed parameters of the atmosphere. The tools are a mixture of quantitative and qualitative methods. We will briefly outline some important steps:

- **Data collection:** Some key parameters in the atmosphere, e.g. temperature, air pressure and rain fall, are regularly and simultaneously measured at a number of observing stations.
- **Objective analysis:** The collected data is fed into a central computer where a numerical model is used to calculate a prediction for a large geographical region, in general for the next 24–72 hours. The numerical model uses some key physical laws in the form of differential equations, but contains many simplifications in order to make it tractable.
- **Subjective analysis:** The collected and predicted data is plotted on separate weather maps that are analyzed by hand by the meteorologist.
- **Prediction:** The meteorologist makes a prediction based on both the subjective and objective analyses. Most predictions concern short time periods and limited geographical regions that are not specifically catered for in the objective analysis, e.g. the area around an airport in the next hour.

From this description, we see that the computer-supported number-crunching of the objective analysis is only one part of the weather prediction process. We are interested in modelling the common-sense reasoning that takes place in the last two phases, i.e. subjective analysis and prediction.

Good weather predictions are based on a thorough understanding of the on-going physical processes in

---

\*This research is supported by the Swiss National Science Foundation, project no. 5003-034269.

the atmosphere. The subjective analysis is time-consuming but necessary in order to build a mental model of these processes. This model is called "the inner weather picture" in [Perby, 1988], where the mental processes underlying weather prediction are discussed in more detail.

The subjective analysis starts with a study of the spatial distribution of each observed parameter. The observed values are indicated as points on a weather map. The meteorologist analyzes one parameter at a time by indicating regions of similar values on the map, e.g. isobars, isotherms, regions of precipitation, fog, cloudiness, etc. The analyzed map is used as a means of communication between meteorologists, and enables them to detect significant patterns of regions that indicate which underlying physical processes are at work. This process-based understanding creates an expectation of how the situation will develop, which is compared with the prediction of the objective analysis. The final prediction is based on the meteorologist's total understanding of the situation, which has been created from various sources: knowledge of physics, previous experience, collected data, objective prediction.

This kind of reasoning is interesting to artificial intelligence research as it involves at least four different research areas:

- **Model-based reasoning:** The reasoning is based on underlying models of physical phenomena.
- **Spatial reasoning:** The location where a parameter is observed is as important as the measured value.
- **Qualitative reasoning:** Due to the sparseness of observed data, assumptions and simplifications of a qualitative nature are necessary.
- **Diagrammatic reasoning:** Diagrams are extensively used to understand complex situations and to communicate this understanding.

We believe that the working methods of meteorologists are representative of many other scientific areas where physical systems of spatially distributed parameters are studied. Some examples are natural resource management, agriculture, ecological modelling, ocean studies, etc. We propose to model this reasoning process as follows:

- **Interpretation phase:** Building a scenario of the situation through analysis of the spatial distribution of individual parameters given as sets of observation points and a model of the physical properties of each parameter.
- **Simulation phase:** Simulation of the evolution of the situation through application of physical models to the scenario. The simulation phase consists of a static and a dynamic part:
  - **Static inference:** Inference of non-observed parameters through combinations of observed parameter values.

- **Dynamic inference:** Inference of the spatial evolution of parameters in terms of modifications to their spatial distributions.

The rest of this paper describes each of the above phases in turn, followed by a discussion where we put this work into perspective by comparing it to other approaches. We also discuss the utility of this approach and outline the current state of research and directions for future work.

## 2 Interpretation Phase

The goal of the interpretation phase is to build a scenario of the situation that can be used to detect which physical processes are causing the situation and simulate their evolution. In accordance with our study of meteorological practices, we propose to model the distributed parameters individually and use combinations of distributions to reason about the evolution of the system.

A physical system of spatially distributed parameters occupies a region of space, where each point can be assigned a value for each parameter. The values of each parameter are distributed in a specific pattern within the region. A *qualitative* description of this pattern is obtained by a double discretization: on the value domain of the parameter and on the space it describes. The value domain, e.g. the set of real numbers  $\mathcal{R}$ , is discretized into qualitative categories, e.g. intervals. An analogous spatial discretization is carried out on the points in the described space by grouping neighbouring points with equal values into larger spatial units, i.e. *regions*. The spatial distribution of the parameter is described qualitatively as a patchwork-like pattern of contiguous regions.

Figure 1 illustrates an example of the kind of physical system we want to model with this method.

The illustrated physical system is a cross-section of a part of the Earth-Atmosphere system, which can be described by different physical parameters, e.g. temperature, relative humidity, etc. In this scenario, the parameter *temperature* divides the space of the physical system into a pattern of three regions, corresponding to a discretization of the value domain  $\mathcal{R}$  into the symbolic values  $\{cool\ warm\ hot\}$ . The parameter *relative humidity*, on the other hand, divides the same space into a different pattern of only two regions, corresponding to its proper value domain discretization:  $\{dry\ humid\}$ .

The initial information on the distribution of a parameter is quantitative/metric and limited to the coordinates of the observation points and the values that have been observed at those points. Inferring the rest of the distribution from this sparse data requires a number of assumptions, which means that the resulting description will be qualitative in nature.

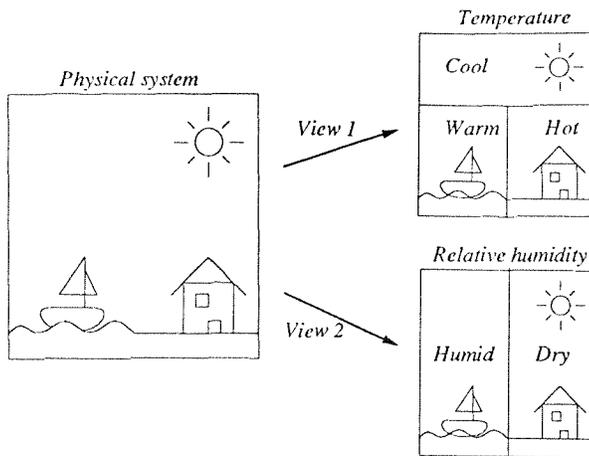


Figure 1: A physical system of spatially distributed parameters.

In order to distinguish between known quantitative/metric data and approximated/simplified qualitative data, we divide the description of the distribution of a parameter into two parts: a metric diagram and a place vocabulary. This division was proposed as a general model for qualitative spatial reasoning in [Forbus *et al.*, 1987]:

- The *metric diagram* describes the metric and quantitative properties of the world to be reasoned about. It is used for those queries that cannot be answered by purely qualitative reasoning.
- The *place vocabulary* describes the same world in qualitative terms.

Since the metric diagram and the place vocabulary describe the same world, although in different ways, they should be compatible. This is accomplished by using the quantitative information in the metric diagram to calculate the qualitative representation of the place vocabulary.

In this approach, the metric diagram consists of the set of observation points and a suitable connectivity structure. The place vocabulary is a topological map of the regions derived from the metric diagram. The construction of these two structures is supported by a model of the specific physical properties of the parameter. In the following sections, we will describe the construction and purpose of each of these components.

## 2.1 Metric Diagram: Connected Point Set

The goal of the analysis of the observation point set is to describe the spatial distribution of the parameter as a pattern of contiguous regions. This is accomplished by comparing the values observed at neighbouring points. If the same value is observed at two neighbouring points, then they can be considered qualitatively equal and grouped into a larger spatial unit,

i.e. a region. If the observed values are not the same, then the two neighbouring points lie on the boundaries of two different regions.

Space is a continuous medium and consists of an infinite number of points. Between two points, there will thus always be an intermediate point, making the concept of *neighbour* very relative. Two points are only neighbours with respect to some level of approximation where all intermediate points are disregarded.

In the case of observation point sets, it is not always obvious which points are neighbours, since they can be spread out in an irregular pattern. Figure 2 shows a set of observation points for the parameter *temperature*. The observed values have been categorized into the qualitative values {cool warm hot}. The observation points can be in two or three dimensions, depending on which physical system is being modelled.

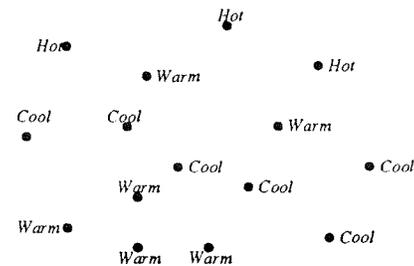


Figure 2: A set of observation points for the parameter *temperature*.

In order to know which points are neighbours and can be compared, a connectivity structure must be defined on the observation point set, i.e. a graph structure that indicates neighbourhood relations. A triangulation provides a natural connectivity structure for this kind of point set. A point set can be triangulated in many different ways. For the purpose of comparing observed values at neighbouring points, a triangulation that minimizes the distance between connected points is the most suitable. In [Preparata and Shamos, 1985], several algorithms are given for constructing various connectivity structures on point sets.

Figure 3 shows a triangulation of the point set in figure 2, where two points are neighbours only when the straight line connecting them does not intersect any shorter line connecting two points.

The metric diagram of our representation is the observation point set and a chosen connectivity structure. It is used to construct the place vocabulary, as described in the next section.

## 2.2 Place Vocabulary: Topological Map

The place vocabulary describes the qualitative, non-metric properties of the metric diagram. Whereas the metric diagram describes the spatial distribution of the parameter as a network of observation points, the place

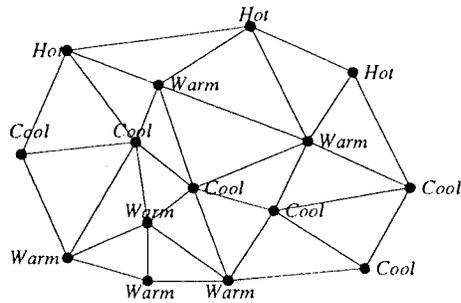


Figure 3: Metric diagram: triangulation of the point set in figure 2

vocabulary will describe the same distribution as a pattern of contiguous regions.

The place vocabulary is constructed by comparing neighbouring points in the metric diagram, according to the chosen connectivity structure, and grouping points with equal values into larger regions. Figure 4 shows how to detect regions in the metric diagram in figure 3.

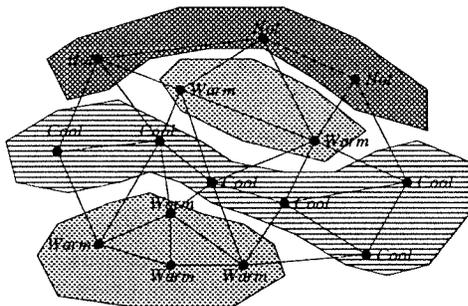


Figure 4: Regions detected in the metric diagram in figure 3.

The connectivity structure of the detected regions can be represented as a *topological map*. A topological map is an abstract illustration of the neighbourhood relations between regions, and does not convey any information on size or shape. Figure 5 shows the topological map of the regions in figure 4.

This particular topological map indicates the regions that can be detected by a straightforward analysis of the metric diagram. The next section describes how the topological map can be refined through the use of a model of the parameter's physical properties.

### 2.3 Parameter Model

During the interpretation phase, when representations of individual parameter distributions are being constructed, a model of the physical properties of a parameter enhances the information in the metric diagram and can lead to the inference of additional regions in the topological map or to a refinement of it.

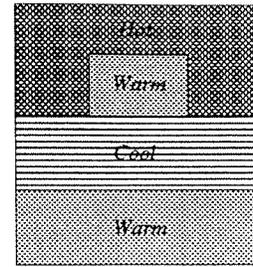


Figure 5: Place vocabulary: topological map of the regions in figure 4.

A parameter is defined by its name, unit and value domain, which can be finite or infinite. By dividing the value domain into different *quantity spaces*, i.e. sets of qualitative values, the distribution of the parameter can be described at varying levels of detail. Examples of quantity spaces are sets of intervals or symbolic values. The quantity spaces define alternative views of the value domain. The parameter model provides information on how to map between different quantity spaces and the value domain.

The value domain of a parameter is modelled as being either spatially *ordered* or *unordered*. This modelling choice depends on which properties of the physical system one wants to convey.

Spatially ordered value domains indicate that the spatial transition from one value to another must follow the order given in the quantity space and that there can be no discontinuities. This is a convenient property since it enables us to infer more information from the metric diagram than has actually been observed.

The topological map in figure 5 indicates that the two value regions *cool* and *hot* are neighbours. If the value domain of the parameter, in this case *temperature*, is defined as spatially ordered with the quantity space {*cool warm hot*}, then we can infer the existence of an intermediate *warm* region, although this value has not been observed. Figure 6 shows the resulting topological map.

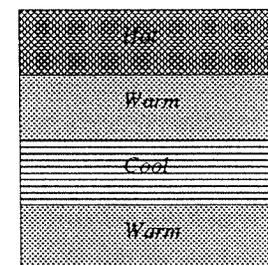


Figure 6: A refined version of the topological map in figure 5.

Parameters with spatially unordered value domains are equally common and have the property that any

two values in the quantity space can correspond to neighbouring regions in the topological map. One example is the parameter *weather-type* which is an important observation in meteorology. An example of a quantity space for this parameter is {*rain thunder cloudy fog fine*}. Any two regions can be neighbours, thus it is not possible to infer any other regions than those detected in the metric diagram.

The metric diagrams and topological maps constructed for the distribution of each parameter make up a scenario describing the situation in a conceptual way. This scenario will be used during the simulation phase.

### 3 Simulation Phase

During the simulation phase, the constructed scenario is used to reason about the spatial properties and evolution of the physical system.

The scenario is a conceptual model of the situation, where the spatial distribution of each observed parameter is described by a metric diagram and a topological map. By modelling physical processes as topological maps and matching these against the topological patterns in the scenario, alternative descriptions of the physical system can be inferred and its evolution simulated.

The drawn inferences can be characterized as either *static* or *dynamic* as follows:

- **Static inference:** The scenario is used to infer the distributions of non-observed parameters through combinations of observed parameter values. This inference is static since it leads to alternative views of the physical system in the form of new parameter distributions, but existing parameter distributions are not modified.
- **Dynamic inference:** The scenario is used to infer the spatial evolution of existing parameter distributions, either observed or inferred. This inference is dynamic since it will modify the representation of existing parameter distributions. This may trigger further static or dynamic inferences.

The following sections will describe how to model static and dynamic inference respectively.

#### 3.1 Static Inference

Reasoning about physical systems often means *combining* values of parameters in order to infer the value of some other parameter. A combination model describes which parameters are involved and how to calculate the result as a function of the parameters' values.

The combination model can be an equation or some other relevant computable function of several parameters. It can be expressed either as a qualitative version of an equation, e.g. using interval arithmetic, or as a matrix of value correspondences. Several versions of a

combination model are possible to cater for all possible combinations of quantity spaces for the same value domain, i.e. different levels of granularity.

In the case of spatially distributed parameters, the values to combine must coincide in space as well as in time. As an example, consider the meteorological form of the equation of state:

$$P = \rho RT$$

$P$  is pressure,  $\rho$  is density,  $R$  is the specific gas constant and  $T$  is temperature.  $P$ ,  $T$  and  $\rho$  are distributed parameters with spatially ordered value domains, that can be discretized into intervals or symbolic values.  $R$ , the gas constant, also has a spatial distribution in the sense that it is applicable at all points where the specific gas has a distribution.  $P$  and  $T$  are readily observable parameters, whereas direct observation of  $\rho$  requires quite complicated equipment. It is thus convenient to infer the distribution of  $\rho$  from the given equation and the observed parameter distributions.

$P$  and  $T$  are alternative views of the same region in space. By superimposing the spatial distributions of  $P$  and  $T$ , a new description of the same space emerges as a pattern of regions where the values of both  $P$  and  $T$  are constant. This pattern is the spatial distribution of  $\rho$ . The value of  $\rho$  in each region is calculated by applying the equation, or a qualitative version of it, to the values of  $P$  and  $T$  in these regions.

In the following sections, we will describe how to construct the spatial combination of two parameter distributions. We will also discuss how to handle the spatial ambiguity that may arise due to sparse data.

#### 3.1.1 Combined Topological Maps

In order to infer the distribution of a parameter expressed as a function of other parameters, we must know which value regions intersect in space. For this purpose, a *combined topological map* is constructed for the involved parameters.

A topological map describes the connectivity structure, i.e. neighbourhood relations, of the regions *within* a single parameter distribution. Analogously, a combined topological map describes the topological relations *between* parameter distributions, i.e. where different regions intersect in space. The term for this topological relation is *overlap*. Two regions overlap if they have at least one point in common.

The metric diagram does not allow any inference of the exact shape of the different value regions. It is thus impossible to say exactly where and how two regions overlap. What can be inferred is whether two regions are *certain* to overlap, whether they *may* overlap or whether they are *certain not* to overlap. This amounts to finding out whether two regions have at least one point in common, do not have a point in common or it cannot be decided if they have a point in common.

The combined topological map is constructed by combining the metric diagrams of the parameters. In

doing this, we want to infer which values could have been observed for the second parameter at the observation points of the first parameter, and vice versa.

In many practical applications, the parameters will have been observed at the same points, i.e. their metric diagrams will be spatially equal, only the observed values will be different. E.g. in the case of meteorology, most parameters are observed at the same observing stations. If two observation points are identical, then we know that the two regions, one for each parameter, that were inferred by means of that observation point are certain to overlap, since they have at least that point in common.

However, in the general case, two parameters need not have identical metric diagrams. By adding the observation points of the second parameter to the metric diagram of the first, we see that each new point falls within exactly one triangle in the metric diagram, as defined by the chosen connectivity structure. The values observed at the points connected by the triangle are the values that could have been observed at the newly inserted point.

Figure 7 shows an example of this situation. A point has been added to the metric diagram of the parameter *weather-type*, which has the spatially unordered quantity space {rain thunder cloudy fog fine}. The inserted point falls within a triangle connecting three observation points, where the values *rain*, *fine* and *thunder* have been observed. According to our assumption that value transitions take place between neighbouring points according to the chosen connectivity structure, exactly one of these values must have been observed at the inserted point.

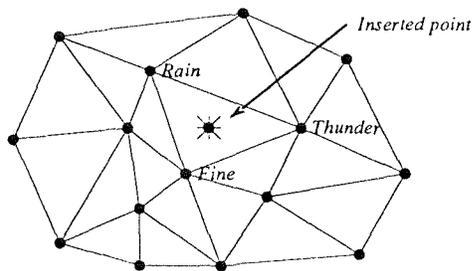


Figure 7: Inference of values that could have been observed at an inserted point.

This is an ambiguous situation with one, two or three alternatives, depending on how many different values have been observed at the three points connected by the triangle. The metric diagram does not allow us to decide which of the three regions the inserted point belongs to. However, it does allow us to decide whether a spatial intersection between regions in two different distributions is possible or not.

Figure 8 shows the different situations that can arise, assuming that the inserted point belongs to the metric diagram of the parameter *temperature* and the ob-

served value is *cool*. The situations correspond to the following rules:

- **Certain overlap:** If two regions have at least one observation point in common then they are *certain* to overlap.
- **Possible overlap:** If two regions do not have any observation point in common, but some point falls within a triangle that has led to the inference of the region in the other distribution, then the two regions *may* overlap.
- **No overlap:** If the above rules do not apply, then the two regions are certain to be disjoint, i.e. they do *not* overlap.

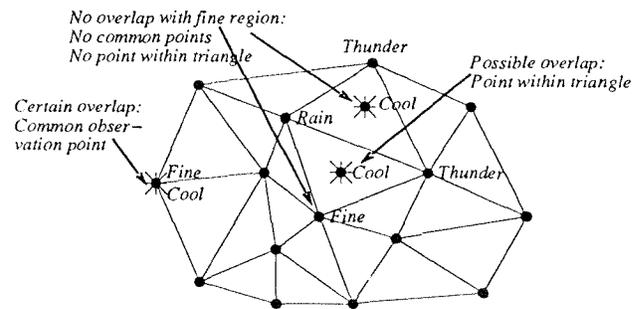


Figure 8: Inference of overlapping regions: the three situations deciding whether two regions are *certain* to overlap, *may* overlap or are certain *not* to overlap.

The combined topological map is constructed by comparing each pair of regions in the two topological maps according to the rules mentioned above. In case of ambiguity, the result is a *set* of combined topological maps, each indicating a possible overlap situation.

Figure 9 shows an example of two topological maps, for the parameters *temperature* and *weather-type*, and one possible combined topological map given the overlap structure indicated in table 1.

In the next section we discuss how to reduce the occurrence of spatially ambiguous situations.

Temperature	Weather-type	One possible combination
Hot	Thunder	Thunder/Cool
Warm	Cloudy   Rain	Cloudy/Cool   Rain/Cool
Cool	Fine	Cloudy/Warm   Rain/Warm
		Fine/Hot

Figure 9: Topological maps for the parameters *temperature* and *weather-type* and one possible combined topological map given the overlap structure in table 1.

	Cool	Warm	Hot
Fine	Disjoint	Disjoint	Certain
Cloudy	Certain	Possible	Disjoint
Rain	Certain	Certain	Possible
Thunder	Certain	Certain	Possible

Table 1: Overlap structure for the topological maps in figure 9.

### 3.1.2 Controlling Spatial Ambiguity

In the previous section, we saw that combinations of topological maps sometimes contain ambiguous regions, where it cannot be decided whether two regions in the original distributions overlap or not. This ambiguity is due to the sparseness of data in the metric diagram. Human experts, e.g. meteorologists, use domain knowledge to disambiguate in this kind of situation. The following methods can be used to handle spatially ambiguous situations:

- **Treat the ambiguous region locally:** The ambiguity only concerns a pair of regions and is thus *local*. It does not influence the rest of the combined topological map, provided all other regions can be combined without ambiguity. Reasoning can thus continue unambiguously for a large part of the space of the physical system. The ambiguous region can be treated locally, either by indicating its value as unknown or by branching into multiple representations of that region.
- **Use proximity information to solve the ambiguity:** In some applications, proximity information can be used to disambiguate. A plausible model is to let an inserted point belong to the region of the observation point it is closest to. Figure 10 shows an example of this situation.

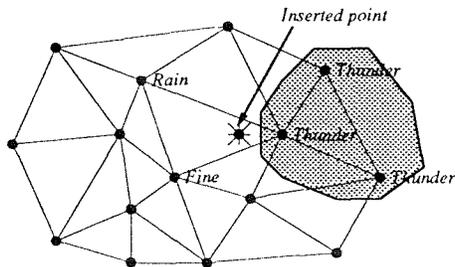


Figure 10: Disambiguation through proximity: the inserted point is inferred to belong to the shaded region since it lies closest to that observation point.

- **Use hierarchical parameter models to avoid unnecessary ambiguity:** Many parameters are physically relevant only in conjunction with some other parameter. By including this domain knowledge in the model, many potentially ambiguous sit-

uations can be avoided. Figure 11 shows an example of such a case. Again, the example is taken from climate modelling and illustrates a part of the Earth-Atmosphere system. The system is described by two parameters: *atmospheric-layer* and *soil-type*. The parameter *atmospheric-layer* divides space into regions according to the simplified quantity space {*stratosphere troposphere ground*}. The parameter *soil-type*, with the quantity space {*sand clay peat*}, can only describe points within regions described by the value *ground* for the parameter *atmospheric-layer*. Regions in the topological map of the parameter *soil-type* can thus only overlap with regions characterized as *ground* in the topological map of the parameter *atmospheric-layer*, and no further combinations need be considered in the construction of the combined topological map.

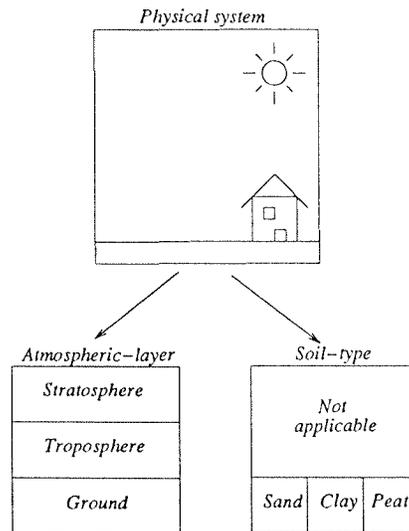


Figure 11: Hierarchical parameter models reduce the occurrence of spatial ambiguity.

### 3.2 Dynamic Inference

Dynamic inference differs from static inference in that it modifies the distributions of existing parameters instead of inferring new parameter distributions. In physics, dynamic evolution is usually modelled by differential equations. In qualitative physics, the temporal evolution of a parameter is usually modelled as transitions between subsequent landmark values in the quantity space of the parameter's value domain.

Analogously, in qualitative spatial simulation, the transitions will reflect significant changes to the spatial distributions of the parameters, or more precisely changes to their topological maps. Significant changes can take place either *within* a distribution, by rearranging the neighbourhood structure of the regions, or *between* distributions, in which case the overlap struc-

ture between regions is modified.

Physical processes are modelled as topological patterns of parameter regions that are matched against the scenario constructed during the interpretation phase. The spatial evolution of the system is given as a sequence of subsequent topological modifications to the parameter distributions.

Spatially distributed parameters often evolve through flow processes. We will outline a model of radiative flow from the sun through the layers of the atmosphere. Figure 12 illustrates the situation, which is, again, a part of the Earth-Atmosphere system, this time described by the parameters *emissivity*, *transmissivity* and *irradiation*.

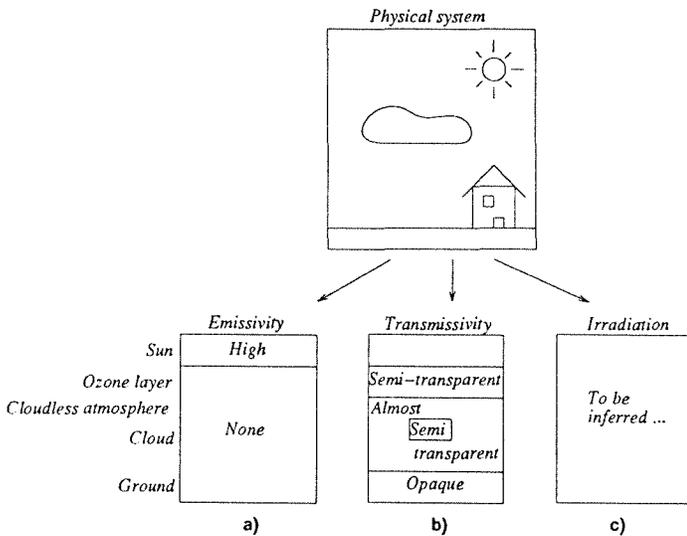


Figure 12: Topological maps for some key parameters in a model of radiative flow.

The parameter *emissivity* describes sources of short-wave radiation. Figure 12a shows the topological map of the parameter *emissivity* in this situation, using the quantity space {*high none*}. There is only one region of high emissivity, namely the sun.

The parameter *transmissivity* indicates how much of the radiation received by a region will be transmitted to more distant regions. The radiation that is not transmitted is either reflected or absorbed, which will increase the temperature of the region. However, those processes are not modelled in this example, which focuses on radiative flow. The topological map of the transmissivity regions in this situation is given in figure 12b. Its quantity space in this example is {*transparent almost-transparent semi-transparent almost-opaque opaque*}. There are four different transmissivity regions in the scenario, corresponding to the *semi-transparent* ozone layer, which filters a lot of the radiation coming into the atmosphere, the *almost-transparent* cloudless atmosphere, a *semi-transparent* cloud, and finally the *opaque* ground, which absorbs

or reflects all radiation it receives and transmits none. The transmissivity of the sun is not relevant to this model, so we leave the corresponding region unspecified.

The parameter *irradiation* indicates regions that receive radiation. The initial distribution of this parameter in figure 12c indicates no irradiated regions. The model will describe the spatial evolution of the distribution of this parameter. The final distribution will indicate which regions in space receive more radiation than others.

In this model, we want to reason about how some regions are shadowed by others, and thus receive less radiation. In order to do this, it is necessary to include the notion of *flow direction* in the model. *Direction* is a spatially distributed parameter that divides the space of the physical system into qualitative vector fields with respect to some region. Figure 13a shows the distribution of the parameter *direction* with respect to the sun, i.e. the *high* emissivity region in figure 12a. Figure 13b shows another distribution of *direction*, this time with respect to the ozone layer, i.e. the upper *semi-transparent* transmissivity region in figure 12b. Finally, figure 13c shows the distribution of *direction* with respect to the cloud, i.e. the smaller *semi-transparent* transmissivity region in figure 12b. The value domain has been discretized into the categories {*inside beneath above left right*} which are convenient to this model.

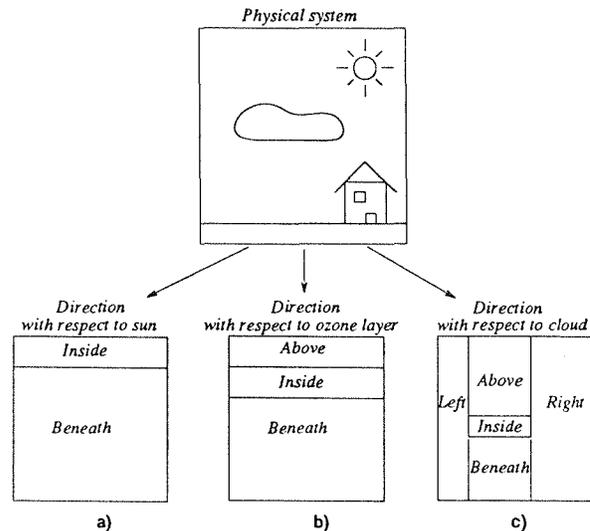


Figure 13: Topological map of the metric parameter *direction* with respect to different regions.

The simulation of radiative flow proceeds in the following steps:

- Regions of high emissivity match the pattern required for the physical process *radiative-flow*. There is only one region that matches this description in

the topological map of the parameter *emissivity* (figure 12a), namely the sun. The region of *high* emissivity becomes the source region of this instance of the flow process.

- Once a source of radiative flow has been found, the regions that it will flow into must be detected. Flow follows a spatial order, so the receiving regions will be neighbours of the source region. The source region has only one neighbour in its topological map (figure 12a), which is indicated by the value *none*. This becomes the sink region for the flow.
- Once there is a source region and a sink region, the direction of the flow can be determined. Figure 13a shows the distribution of the parameter *direction* with respect to the source region, i.e. the sun. The overlap structure between this topological map and that of the parameter *emissivity* indicates that the sink region is totally contained within the *beneath* region. This value becomes the direction of the flow.
- The overlap structure between the topological maps *emissivity* and *transmissivity* indicates that the space occupied by the designated sink region contains several different regions of transmissivity. The flow will proceed gradually through these regions.
- The first step is the region of *semi-transparent* transmissivity that lies closest to the source, i.e. the ozone layer. That region will receive all the radiation sent from the emitting region, i.e. 100%. This is indicated in the model as a modification to the topological map of the parameter *irradiation*. A region that corresponds to the ozone layer is introduced into the *irradiation* distribution and given the value 100%, see figure 14a. For the sake of this example, we will not bother with defining a quantity space for the parameter *irradiation*, but simply indicate the irradiation with approximative percentages.
- The flow will pass through the irradiated region according to the inferred flow direction. However, only a part of the received radiation is transmitted, as some of it is absorbed or reflected. Consultation of the overlap structure between the parameters *irradiation* and *transmissivity* indicates how much radiation will be transmitted. In this case, the irradiated region corresponds to a region of *semi-transparent* transmissivity, so we presume that 50% of the radiation passes through it. In the real model, a suitable qualitative equation would be used.
- The flow from the current irradiated region, i.e. the ozone layer, proceeds into neighbouring regions of constant transmissivity. The flow parameter indicates that these regions must also lie *beneath* the initially irradiated region. This is the case for the cloudless atmosphere, indicated by *almost-transparent* in figure 12b.
- However, the flow model also requires that the receiving region have no holes. The topological map

in figure 12b indicates that the *almost-transparent* cloudless atmosphere contains a region of lower transmissivity, namely a cloud. The correct region to irradiate is constructed by removing the cloud and the area *beneath* it, according to the flow parameter, from the cloudless atmosphere. The resulting irradiated region is shown in figure 14b. Its value is indicated as 50%, reflecting that some of the radiation was absorbed by the preceding region in the flow.

- The radiation continues to flow through the atmosphere, reaching the cloud, the shadowed region beneath the cloud and the ground. The final distribution of the parameter *irradiation* is shown in figure 14c.

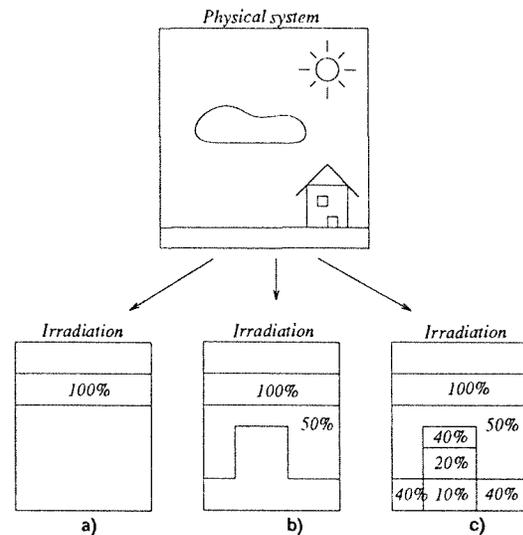


Figure 14: Steps in the inference of the irradiation distribution.

In this model, the existence of a cloud resulted in non-uniform radiation of the ground. The received radiation will be absorbed and transformed to heat according to the distribution of e.g. the parameter *heat-capacity*, thus creating a pattern for the parameter *temperature*. Differences in temperature often trigger other physical processes, e.g. sea breezes, cloud formation, plant growth, etc. These are just a few examples of physical processes that can be modelled with spatially distributed parameters.

## 4 Discussion

This paper has described work on modelling the qualitative behaviour of physical systems of spatially distributed parameters. We will put this work into perspective by comparing it to some other approaches.

In qualitative physics research, physical systems are often modelled as sets of spatially discrete objects.

The parameters are seen as attributes that describe the objects they are associated with. The model describes how the objects interact through their parameters. Object-oriented models are appropriate for many applications. see e.g. the thermodynamics model in [Collins and Forbus, 1991].

However, in applications like meteorology, it is not evident which objects the model should be built on. The system is more appropriately described by the spatial distribution of the individual parameters. The traditional approach in physics is to model a physical system as a set of differential equations, each describing how the value of a parameter varies as a function of some spatial axis. By combining different dimensions as needed, a full description of the parameter is obtained. Work on using differential equations in spatial qualitative reasoning has been presented e.g. in [Throop, 1989] and [Nishida, 1993].

A third approach, based on topology, is given in [Cui *et al.*, 1992] where the process of phagocytosis, i.e. amoeba ingesting food, is modelled as a sequence of topological relations between discrete regions in space.

Modelling a physical system in terms of objects is attractive, since it gives the model a tangible touch. It is appropriate both for device- and process-oriented qualitative simulation and the envisioned situations can easily be illustrated diagrammatically. However, these models fail to convey the notion of continuous spatial distribution, which is readily modelled by differential equations. Models based on explicit differential equations are, however, not as readily understood by non-experts and are also not available for all kinds of physical systems.

The work presented in this paper can be seen as a combination of these two modelling approaches, where the spatial distributions of parameters are divided into patterns of discrete regions that can be manipulated as objects.

The utility of qualitative models of spatially distributed physical processes is manifold. Such models would provide a reasoning component for Geographic Information Systems (GIS) and programs for scientific visualization. They would provide a means of communication between professionals, e.g. meteorologists, by making it easier to hand over analyses of spatial situations to the next person on the shift. Their utility for pedagogical purposes is obvious. In fact, most of the examples in this paper have been taken from textbooks on meteorology and climate modelling, which, although their main purpose is usually to convey a quantitative understanding of the atmosphere, often devote a substantial part of each chapter to qualitative and diagrammatic descriptions of atmospheric processes.

Work on this approach continues actively. The next step will be to refine and implement the methods described in this paper. Models of basic atmospheric processes, such as radiation, conduction, convection and

advection, are being developed, and will be integrated in a model of a fairly complex atmospheric process: the life-cycle of a sea breeze. We are also investigating applications in agriculture and natural resource management.

## 5 Acknowledgments

I would like to thank the Swiss National Science Foundation for supporting this research under SPP Project No 5003-034269. Mr. Jacques Ambühl at the Swiss Meteorological Institute has kindly introduced me to practical weather prediction and pointed me to relevant literature. I would also like to thank Professor Boi Faltings and Dr. Dean Allemang for useful comments.

## References

- Collins, John and Forbus, Ken 1991. Building qualitative models of thermodynamic processes. Technical report, Beckman Institute, University of Illinois.
- Cui, Z.; Cohn, A. G.; and Randell, D. A. 1992. Qualitative simulation based on a logical formalism of space and time. In *AAAI*. 679-684.
- Forbus, Ken; Nielsen, Paul; and Faltings, Boi 1987. Qualitative kinematics: A framework. In *IJCAI*. 430-436.
- Nishida, Toyooki 1993. Generating quasi-symbolic representation of three-dimensional flow. In *Seventh International Workshop on Qualitative Reasoning about Physical Systems*.
- Perby, Maja-Lisa 1988. Computerization and skill in local weather forecasting. In *Knowledge, Skill and Artificial Intelligence*. Springer-Verlag. 39-52.
- Preparata, Franco and Shamos, Michael 1985. *Computational Geometry: An Introduction*. Springer-Verlag.
- Throop, David R. 1989. Spatial unification: Qualitative spatial reasoning about steady state mechanisms, an overview of current work. In *Third International Workshop on Qualitative Reasoning about Physical Systems*.

# QUALITATIVE BEHAVIOR HYPOTHESIS FROM DEVICE DIAGRAMS

N. Hari Narayanan\* and Masaki Suwa and Hiroshi Motoda

Advanced Research Laboratory, Hitachi Ltd.  
Hatoyama, Hiki-gun, Saitama Pref. 350-03, Japan.  
{narayan,suwa,motoda}@harl.hitachi.co.jp

## Abstract

This paper introduces a “qualitative” problem solving task that humans are adept at, but one which has not received much attention within the qualitative physics community. This is the task of predicting the operation of a simple mechanical device, in terms of spatial behaviors of its components, from a labeled schematic diagram of the device showing the spatial configuration of its components and a given initial condition. Using the example of a pressure gauge we define this task, present a cognitive strategy for solving such problems, and describe the architecture of a corresponding computer model.

## Introduction

We often use spatial information implicit in diagrams to make inferences. The task of *qualitative behavior hypothesis from device diagrams* is a case in point: given the labeled schematic diagram of a device that shows the spatial configuration of its components and an initial condition or behavior, predict the operation of the device by hypothesizing the behaviors of its components.

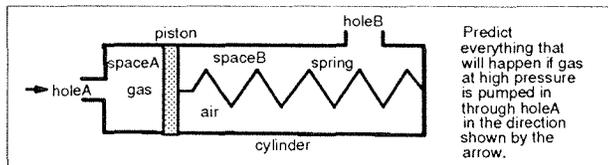


Figure 1: A Behavior Hypothesis Problem: Pressure Gauge

Consider someone examining the cross-sectional diagram of a device, such as the pressure gauge shown in fig. 1, and reasoning about its operation. This requires that he or she reason about spatial processes occurring

\*Current address: Knowledge Systems Laboratory, Stanford University, 701 Welch Rd., Palo Alto, CA 94304. narayan@ksl.stanford.edu.

inside the device. Information used in this type of reasoning is of two kinds: *visual* and *conceptual*. Visual information is obtained from the diagram, and includes spatial configurations and shapes of the device and its components. Conceptual information comes from the domain knowledge of the reasoner, and includes predictive knowledge used for making hypotheses about the device's operation.

In such reasoning situations diagrams clearly serve as compact representations of spatial information. However, this is only part of the story of the role diagrams play in this task. Diagrams also facilitate the indexing of relevant problem solving knowledge. Furthermore, diagrams support mental visualizations of spatial behaviors of device components during the course of reasoning. It has been shown that such mental visualizations guide human reasoning along the direction of causality as perceived from the diagram (Hegarty 1992).

This cognitive capability for “qualitative” visual reasoning from diagrams has not hitherto received much attention in the qualitative physics literature<sup>1</sup>. The automation of visual reasoning can be of benefit in a variety of domains and applications: in automating expert reasoning using phase diagrams (Yip 1991), in developing instructional or demonstration systems whose operation is easily explainable and understandable (Tessler, Iwasaki & Law 1993), and in developing systems whose reasoning spans multiple ontologies or models (Fishwick et. al. 1994, Kiriya & Tomiyama 1993), to cite a few examples.

This paper describes an approach to automating visual reasoning about devices from diagrams. The reasoning task is defined first. Then hypotheses about the pressure gauge in fig. 1 that human subjects generated in an experimental study are presented. Following this, we develop a cognitive process model for this task and

<sup>1</sup>With the exception of Funt's early work (Funt 1980) and the more recent REDRAW system (Tessler, Iwasaki & Law 1993). The work of Förbus and colleagues (Förbus, Nielsen & Faltings 1987) on using a metric diagram for spatial reasoning addressed a different capability than the one being considered here.

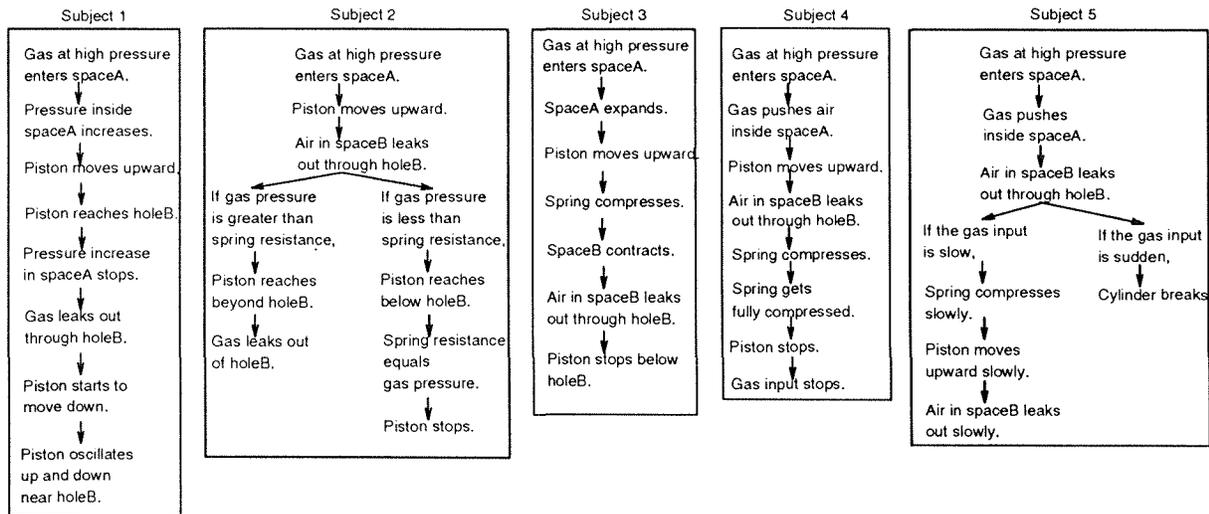


Figure 2: Subjects' Hypotheses about the Pressure Gauge

enumerate behavior hypothesis steps for the pressure gauge according to this model. Finally, the architecture and control algorithm of a computer system designed for this task are described.

## Hypothesizing Behaviors from Device Diagrams

We conducted a set of protocol analysis (Ericsson & Simon 1993) experiments with five subjects solving six qualitative behavior hypothesis problems each. Verbal data (concurrent verbal reports) and gestural data were collected during the course of problem solving and analyzed. Details of these experiments can be found in (Narayanan, Suwa & Motoda 1994). What is of relevance here are the behavior hypotheses generated by the five subjects for the problem shown in fig. 1.

Fig. 2 shows these hypotheses, with the arcs indicating the order in which the hypotheses were generated.

The main goal of these experiments was to characterize how visual information from the diagram and conceptual information (prior knowledge) interact and influence the direction of reasoning during problem solving. Though the solutions that the subjects provided were not always complete and contained inaccuracies, the analyses we carried out - both of the task and the data collected - indicated that the diagram played two important roles during problem solving.

- It facilitated the indexing and recall of both factual knowledge regarding components and inferential knowledge using which the reasoner generated new hypotheses.
- It supported visualizations of hypothesized spatial behaviors of components, which in turn enabled the reasoner to detect effects of these behaviors.

Based on task and data analyses (Narayanan, Suwa & Motoda 1993), we developed a cognitive process model of problem solving in this task. It is shown in fig. 3. It explicates the visual reasoning strategy employed in solving qualitative behavior hypothesis problems. Notice that reasoning proceeds in cycles. At first, short term memory contains only the given initial condition. So reasoning starts with a component and its behavior mentioned in the initial condition. In later cycles, a component and its behavior to focus on are selected from among the hypotheses in short term memory. The diagram facilitates the indexing and recall of relevant knowledge in two different ways: (i) attending to a component may cue some relevant factual information about it which is either recalled from long term memory or retrieved from the diagram, and (ii) configurational and shape information about components from the diagram together with prior knowledge about components and behaviors allow the indexing and recall of inferential knowledge. New hypotheses are generated in three ways: (i) by deliberating about effects of non-spatial behaviors, (ii) by observing the diagram to locate connected/contacting components and deliberating about how these will be affected by spatial behaviors, or (iii) by mentally visualizing spatial behaviors, detecting interactions among components that result, and deliberating about effects of these interactions. In each of these cases, the application of the recalled inferential knowledge creates new hypotheses in short term memory.

Now let us reconsider the problem in fig. 1 and enumerate steps of reasoning according to this process model to generate one solution to the problem.

1. Consider the given initial condition.
2. Observe from the diagram that holeA opens to

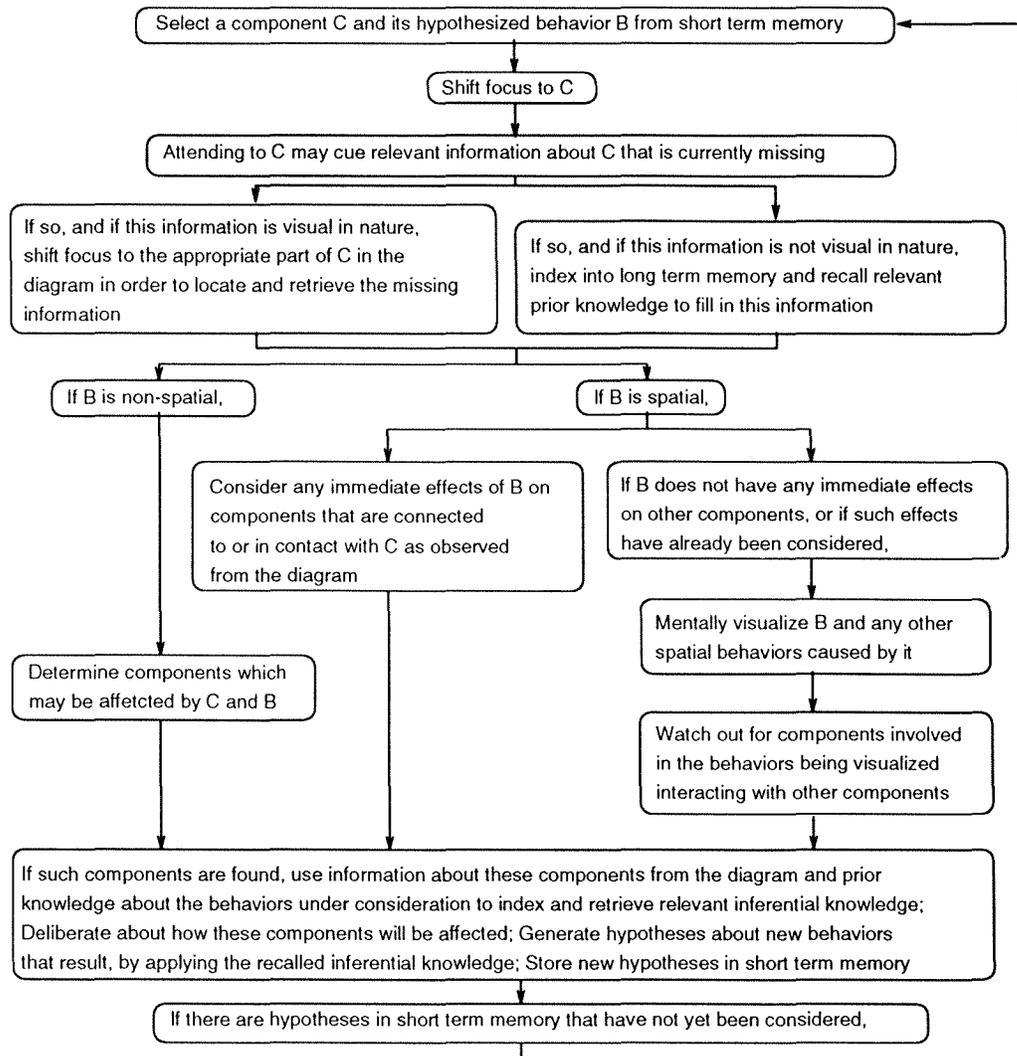


Figure 3: A Cognitive Process Model of Behavior Hypothesis from Device Diagrams

- spaceA.
3. Infer that the pressurized gas will enter spaceA.
4. Observe from the diagram that spaceA is a closed cavity.
5. Recall the inferential knowledge that if a pressurized gas is contained in a cavity, it will exert a force in the normal direction on walls of the cavity.
6. Observe from the diagram that the cylinder and piston form walls surrounding spaceA.
7. Infer that a force in the normal direction will be exerted on the piston and cylinder by the pressurized gas.
8. Recall the inferential knowledge that force can induce motion in a movable component.
9. Recall the factual knowledge that the piston is movable in a piston-cylinder assembly.
10. Observe the piston in the diagram and infer that it is free to move up or down.
11. Infer that the piston will move upward.
12. Observe from the diagram that the piston is connected to a spring and is in contact with air in spaceB; consider each in turn.
13. Recall the inferential knowledge that if a component is connected to another, and the former starts moving in one direction, it will exert a force on the latter in the same direction.
14. Infer that when the piston starts moving upward, it will exert an upward force on the spring.
15. Observe from the diagram that the other end of the

spring is connected to the cylinder.

16. Recall the inferential knowledge that a force applied on a spring will either compress it or expand it depending on the direction of the force.
17. Infer that the spring will compress.
18. Consider the air inside spaceB.
19. Observe from the diagram that spaceB is an open cavity with holeB.
20. Recall the inferential knowledge that if gas inside an open cavity is pushed, it will escape through the cavity's openings.
21. Infer that air in spaceB will exit through holeB.
22. Now that all immediate effects of the hypothesized piston motion have been considered, visualize its upward motion and the spring's compression.
23. Recall the inferential knowledge that as a spring gets compressed or expanded, it will exert an increasing force in the opposite direction.
24. Infer that the spring will exert a force on the piston which, at some point, will equal the force exerted by the pressurized gas on the piston.
25. Observe from the diagram that this may happen before or after the piston reaches holeB; consider each case.
26. In the former case, infer that the piston will stop somewhere below holeB.
27. Infer that the spring compression will cease.

A similar enumeration can be done for the other case, generating the following behavior hypotheses: the piston will reach holeB and allow the pressurized gas to escape; this will decrease the force the gas is exerting on the piston, making it move downwards until a new equilibrium between the gas and spring forces is achieved; this will prevent the gas from escaping, increase its pressure, and the piston will start moving upward; this cycle will then repeat.

## Architecture of a Visual Reasoning System

In this section we describe an architecture for a visual reasoning system designed to solve qualitative behavior hypothesis problems from diagrammatic representations by emulating the cognitive processes outlined previously. It has five main elements: a graphical user interface, a knowledge base, a rule base, a working memory, and an inference engine (see fig. 4). The knowledge base contains two kinds of representations: one which stores descriptive knowledge about the components of a physical device using knowledge structures such as frames that organize knowledge around each component type, and another that stores knowledge

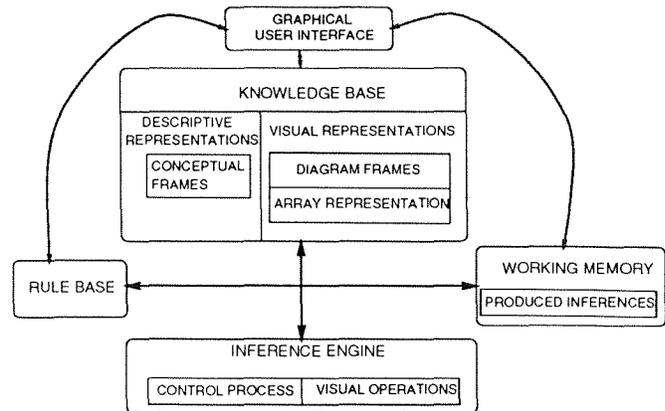


Figure 4: An Architecture for Visual Reasoning

about the shape and geometry of components and devices in a spatially distributed fashion. When a problem to be solved is given by the user, the user interface stores descriptive and spatial parts of the problem specification in the descriptive and visual representations in the knowledge base. Then inference is initiated by the inference engine. It generates new inferences by accessing and manipulating information from both kinds of representations in the knowledge base in accordance with rules selected from the rule base. The rule base contains inference rules with an if-part and a then-part. The if-part of a rule describes conditions regarding properties of components which may be verified by accessing the descriptive representations, and conditions regarding the shape, geometry and configuration of components which may be verified by accessing and manipulating information stored in the visual representations. The then-part of a rule contains new inferences that may be asserted in the working memory if conditions in the if-part are satisfied. The generated inferences/hypotheses are stored in the working memory.

The descriptive representations in the knowledge base contain conceptual information and the visual representations contain spatial information. Conceptual information includes both general knowledge about types of components in the domain and particular knowledge about the components of the device in the input problem. This information is stored in knowledge structures called "conceptual frames" that organize such information around component types. The spatial information consists solely of the shape, geometry, and spatial configuration of the components of the device in the input problem, information that a diagram of the device typically captures. This information is represented in two ways: one by "diagram frames" which are data structures similar to frames or records storing spatial information, and the second by means of filling in appropriate labels in appropriate el-

ements of a two-dimensional array. This array may be seen as directly representing space, spatial configurations of components, their shape and geometry. Fig. 5

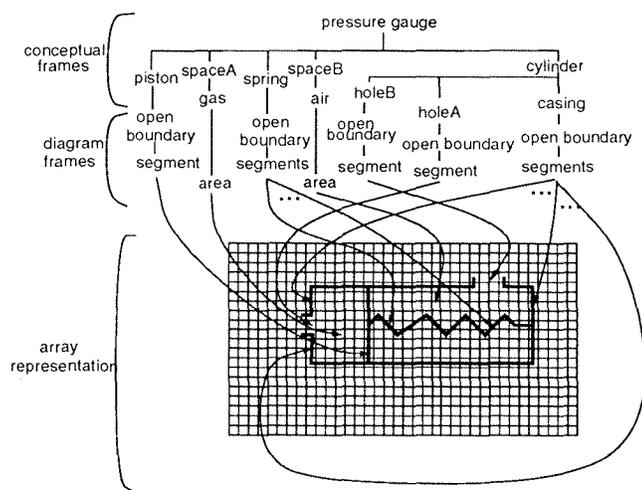


Figure 5: Problem Representation

shows how a device is represented using descriptive and visual representations. It also shows how conceptual frames, diagram frames, and the array representation are related.

A behavior hypothesis problem can be provided to the system by specifying, via the graphical user interface, the components of the device to be reasoned about, any relevant conceptual knowledge about these components, the device diagram, and an initial condition involving behaviors of the device's components. The user interface program takes this specification, and

1. stores conceptual information about the device's components in descriptive representations in the knowledge base;
2. uses given information about component types to link this knowledge with general knowledge about various types of components that already exists in the knowledge base;
3. takes the specification of the device diagram and converts it into a form suitable for representing as visual representations in the knowledge base;
4. represents this above information using both diagram frames and array representation; and
5. takes the initial conditions and stores these in the working memory in a last-in-first-out queue (henceforth referred to as LIFO-Q).

Thus, from the input specifications the user interface program generates an internal representation of the form shown in fig. 5.

Each inference rule in the rule base has three parts: an antecedent or if-part containing conditions that re-

fer to descriptive information and/or visual information, a consequent or a then-part containing new inferences that the system hypothesizes to hold if the conditions in the if-part have been verified, and side-effects, which are procedures that manipulate both descriptive and visual representations in the knowledge base and which get activated when the corresponding rule is fired. Fig. 6 shows a sample inference rule. The rules contained in the rule base can be classified into the following four categories: rules whose if- and then-parts refer only to conceptual frames; rules whose if- and then- parts refer only to diagram frames and the array representation; rules whose if- and then- parts refer to conceptual frames, diagram frames, and the array representation; and rules for carrying out inferences about inequalities and numbers.

The working memory is the computational equivalent of short term memory, except that it is not subject to capacity limitations of human short term memory. The LIFO-Q of inferences/hypotheses is maintained in the working memory. It also contains all new information generated during the course of problem solving.

The reasoning steps carried out by the inference engine fall into the following seven classes:

1. *Diagram Observation (DO)*: Access the diagram frames and/or the array representation to find and retrieve spatial information;
2. *Factual Retrieval (R)*: Retrieval of general knowledge from descriptive representations.
3. *Inference Rule Retrieval (IR)*: Indexing and retrieval of relevant rules from the rule base.
4. *Conceptual Inference (C)*: Making an inference based only on conceptual information from descriptive representations in the knowledge base.
5. *Visual Inference (VI)*: Making an inference based only on spatial information from the visual representations in the knowledge base.
6. *Hybrid Inference (HI)*: Making an inference based on both conceptual information and spatial information.
7. *Visualization (V)*: The operation of simulating a spatial behavior by incrementally modifying the visual representation of the device diagram.

It should be evident that these operations use three different kinds of information: conceptual information from the descriptive representations, spatial information from the visual representations, and associative information in the form of inference rules from the rule base. Similarly, these operations produce inferences that may be classified as conceptual, visual or hybrid.

In order to facilitate accessing and manipulating the visual representations, a set of "visual operations" are made available to the inference engine. Visual operations are procedures for accessing and manipulating both types (diagram frames and the array representation) of visual representations. These are of four kinds:

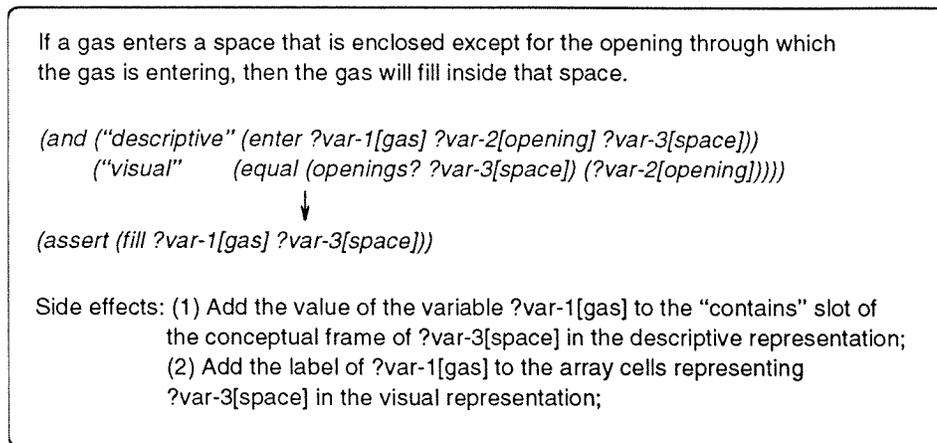


Figure 6: An Inference Rule with Side Effects

basic operations, indexing operations, scanning operations and visualization operations.

*Basic Operations:*

*Read(x,y)* returns labels *l* of the array element with index  $(x, y)$ ; *Write(x,y,l)* marks the array element with index  $(x, y)$  using labels *l*; Two additional operations, *erase(x,y)* and *test(x,y)*, can be defined in terms of the previous two as:  $erase(x,y) = write(x,y,\phi)$ ;  $test(x,y) = false$  if  $read(x,y)$  returns  $\phi$ , true otherwise.

*Indexing Operations:*

Indexing operations generate indices or addresses of array elements. At least four such operations are required.

*Directional indexing:* given an index  $(x, y)$  and a direction<sup>2</sup>, generate the sequence of indices of cells which fall in that direction from  $(x, y)$ .

*Boundary indexing:* given an index  $(x, y)$  and a symbol *s*, generate a sequence of indices of cells, each of which is adjacent to the previous one and contains *s* in its label.

*Neighborhood indexing:* given an index  $(x, y)$ , generate the sequence of indices of its neighborhood cells. The exact definition of neighborhood may vary.

*Fill indexing:* given an index  $(x, y)$ , generate a sequence of indices of cells such that these gradually cover the area surrounding  $(x, y)$ .

Combining indexing routines with basic operations creates procedures that can be used to build the scanning and visualization operations described below.

*Scanning Operations:*

Scanning operations use indexing operations to generate indices of array elements and basic operations to test those elements for various conditions. At least three different kinds of scanning operations are re-

quired.

*Directional Scanning:* Given a starting point in the array, a direction, and one or more conditions, test all array elements from the starting point that fall along the given direction for the given conditions.

*Boundary Following:* Given a starting point on a boundary and one or more conditions, follow the boundary from the starting point and test the boundary elements for the given conditions.

*Sweeping:* Given a line in the array, a direction, and one or more conditions, test all array elements that would be covered if the line were to be moved in the given direction, for the given conditions.

*Visualization Operations:*

Visualization operations manipulate components in the array representation. At least the following four are required: *Move(component, direction)*, *Rotate(component, direction)*, *Delete(component)* and *Copy(component)*.

Figs. 7 and 8 together show the control process underlying the inference engine's operation. This process was developed from the model in fig. 3 by replacing mental representations and mental operations by corresponding knowledge representations and operations on those representations. For example, the storage and selection at the beginning of each cycle of a hypothesis (involving a component and its behavior) from short term memory is implemented using the LIFO-Q data structure in the working memory. The inferential knowledge that is indexed and recalled from long term memory during deliberation is represented as productions with antecedent conditions and consequents. Similarly, the computational process that corresponds to mental visualization is a simulation of spatial behaviors by applying visual operations on the array-based visual representation of the diagram and scanning for component interactions in the cells of the array. The immediate effects on connected/contacting

<sup>2</sup>Sixteen discrete directions are defined on the array, with each differing from the next by 22.5 degrees; this is an arbitrary choice.

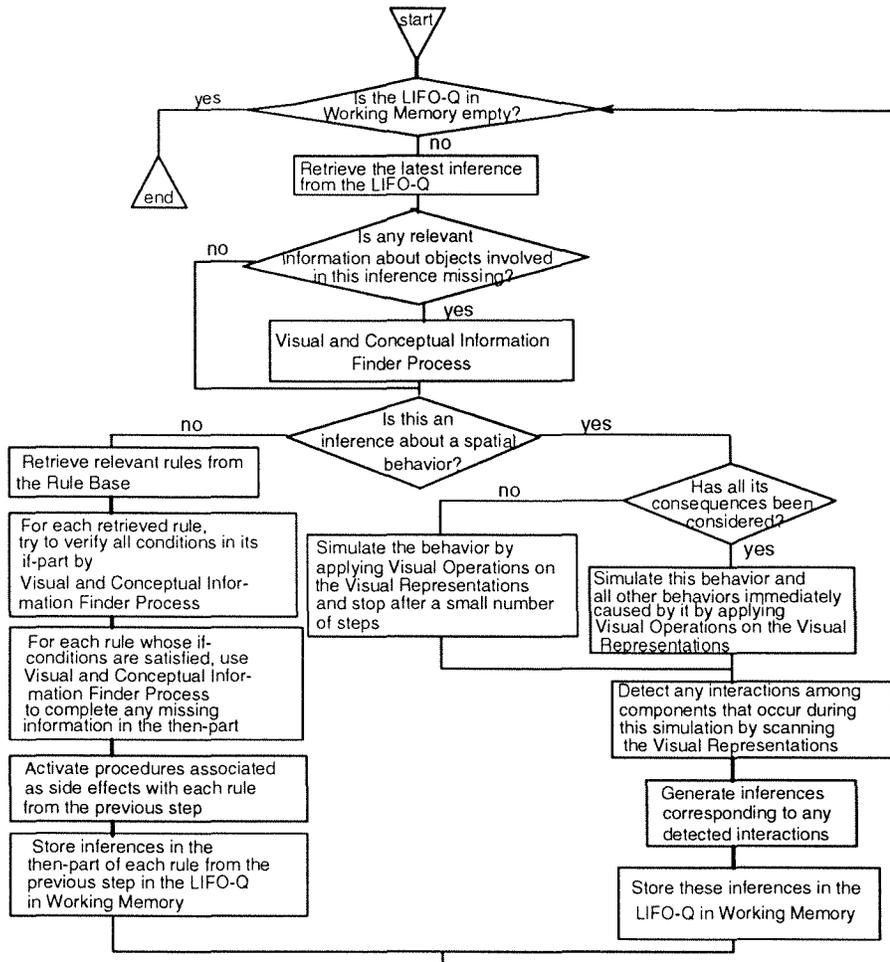


Figure 7: Control Algorithm

components of a spatial behavior can also be detected by a similar computational process: simulating the behavior for a small number of steps and scanning for interactions.

This inference method is a mixture of rule-based reasoning and diagram-based reasoning. The left pathway in the flowchart in Fig. 7 describes rule-based reasoning and the right pathway describes diagram-based reasoning. At the start, reasoning begins in the forward reasoning mode, acting on the first element in the LIFO-Q. Reasoning changes to a diagram-based mode when either (i) the current inference is a hypothesis regarding the spatial behavior of a component and its immediate consequences on other components need to be determined, or (ii) it is a hypothesis regarding a spatial behavior whose immediate consequences have already been determined. In the case of (i) a simulation of the behavior for a few steps is carried out. This will detect any immediate consequences of that spatial behavior on any nearby components. These detected

effects are stored in the LIFO-Q, and forward reasoning will resume at the beginning of the next cycle. In the case of (ii), we have a behavior whose immediate consequences (which might be other behaviors by affected components nearby) have already been determined. So the next step will be to simulate this behavior and its consequences using the visual representations. In this case, the simulation will not be terminated after a few steps. Instead, it will proceed until a spatial interaction between components is detected. Once an interaction is detected, its effects are determined, and stored in the LIFO-Q. Forward reasoning will resume in the next cycle. This is an overview of the control process.

## Conclusion

This paper presented a study of visual reasoning from diagrams in qualitative behavior hypothesis tasks. We began with a definition of the task, following which results from experimental studies were discussed. Anal-

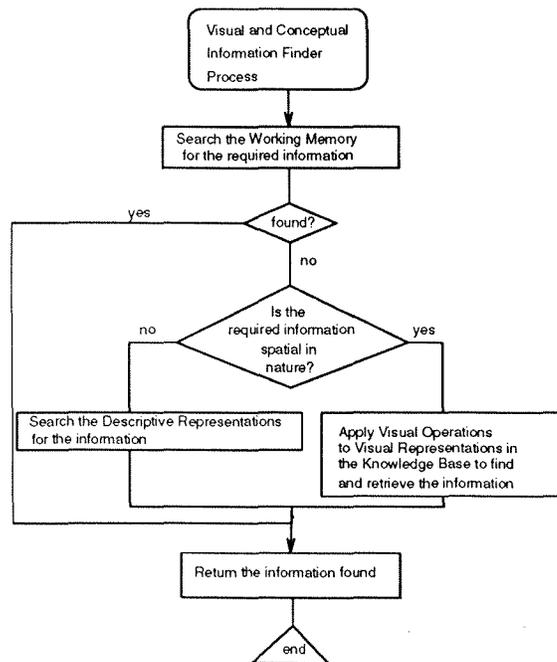


Figure 8: Control Algorithm - Contd.

yses of this task and a close examination of hypotheses generated by human subjects allowed us to formulate a cognitive process model of problem solving in this task. Using this model as a basis we showed how an example problem could be solved. Then we described a control process and the computational architecture of a visual reasoning system designed to solve behavior hypothesis problems using diagrammatic representations. Elements of this architecture were explained. Implementation of a prototype of this system is currently in progress.

**Acknowledgments** We thank Dr. Toyoaki Nishida for providing software for part of the user interface module of the architecture.

## References

- Forbus, K. D., Nielsen, P., and Faltings, B. (1987). Qualitative kinematics: a framework. *Proc. 10<sup>th</sup> International Joint Conference on Artificial Intelligence*, 430-436.
- Funt, B. V. (1980). Problem-solving with diagrammatic representations. *Artificial Intelligence*, 13: 201-230.
- Hegarty, M. (1992). Mental animation: inferring motion from static displays of mechanical systems. *Jnl. of Experimental Psychology: Learning, Memory, and Cognition*, 18(5): 1084-1102.
- Ericsson, K. A. and Simon, H. A. (1993). *Protocol Analysis: Verbal Reports as Data (Revised Edition)*, MIT Press, Cambridge, MA.
- Fishwick, P., Narayanan, N. H., Sticklen, J. and Bonarini A. (1994). A multimodel approach to reasoning and simulation. *IEEE Transactions on Systems, Man and Cybernetics*, in press.
- Kiriyama, T. and Tomiyama, T. (1993). Reasoning about models across multiple ontologies. *Proc. Int'l Workshop on Qualitative Reasoning about Physical Systems*, 124-131.
- Narayanan, N. H., Suwa, M., and Motoda, H. (1993). Behavior hypothesis from schematic diagrams: a hybrid approach. *Proc. IJCAI-93 Workshop on Principles of Hybrid Representation and Reasoning*, 50-61.
- Narayanan, N. H., Suwa, M., and Motoda, H. (1994). A study of diagrammatic reasoning from verbal and gestural data. to appear in *The 16<sup>th</sup> Annual Conference of the Cognitive Science Society*.
- Tessler, S., Iwasaki, Y., and Law, K. (1993). Qualitative structural analysis using diagrammatic reasoning. *Proc. Int'l Workshop on Qualitative Reasoning about Physical Systems*.
- Yip, K. M. (1991). Understanding complex dynamics by visual and symbolic reasoning. *Artificial Intelligence*, 51: 179-221.

# Visual Reasoning With Graphs

Yusuf Pisan

Qualitative Reasoning Group, The Institute for the Learning Sciences  
Northwestern University, 1890 Maple Avenue, Evanston, IL 60201, USA  
e-mail: y-pisan@nwu.edu

## Abstract

Understanding diagrams is an important part of human cognition. Computer programs need to understand and reason using diagrams to communicate effectively with people. This paper explains how line graphs can be interpreted in a domain independent manner. We present a computer program called SKETCHY that reasons about physical phenomena visually by using line graphs. SKETCHY can interpret graphs to recover functional relationships, answer comparative analysis questions and generate qualitative descriptions using geometric models.

## 1 Introduction

People use diagrams to solve problems, to give explanations, to summarize information and to represent spatial relations. Computer programs that can represent, interpret and reason with diagrams will have a great impact on education, cognitive science and artificial intelligence. By making spatial relationships explicit, diagrams can reduce the amount of search and inference required[12]. Diagrams serve both as devices to aid in visualization of the situation[14] and as short-term fast access memory devices for holding information[7]. Although we do not yet have computer programs that can do general diagrammatic reasoning, diagrams have been successfully integrated with computer programs in a number of areas: to explain complex mechanical and dynamic systems[5,2,6,8], to solve geometry and physics problems[7,13], to constrain the search space in problem solving[9] and to understand the role of visual reasoning in problem solving[15].

Diagram understanding requires being able to identify objects, determine the relevant features for a particular problem and map the graphical features to the domain. A graph is a specialized form of diagrammatic representation that does not involve object recognition. Different graph formats emphasize different relationships between variables[10]. For instance, pie graphs are used to show relative percentages, bar graphs and step graphs to show relative amounts, scatter plots to show trends in data and line graphs to show continuous changes. In this paper, we only consider line graphs.

We are interested in interpreting line graphs because of their extensive use in

thermodynamics, physics, economics and other fields. For example, an intelligent tutoring system that reasons with line graphs can use the graph as a shared medium for communication. The system can explain concepts by constructing graphical explanations. The user can also represent her understanding of the concepts graphically and the computer program can check the correctness of her understanding. The user and the computer can both make modifications to the line graph, therefore, providing an additional communication channel between the user and the program. Similar advantages would occur for engineering analysis and knowledge acquisition.

We present a computer program called SKETCHY that reasons about physical phenomena visually by using graphs. SKETCHY provides an interactive drawing environment, answers questions about graphs and interprets user modifications to the graph. SKETCHY has generated interpretations for all the graphs in a college level thermodynamics textbook[17], as well as a number of thermodynamic graphs from [18] and economics graphs from [1]. SKETCHY's interpretations are similar to graph descriptions found in textbooks. To our knowledge, SKETCHY is the first computer program that interprets graphs in a domain independent manner.

The next section (Section 2) gives examples from SKETCHY. Section 3 discusses the theory underlying SKETCHY. Section 4 explains the algorithms and design choices made in building SKETCHY. Finally, Section 5 discusses possible extensions to SKETCHY.

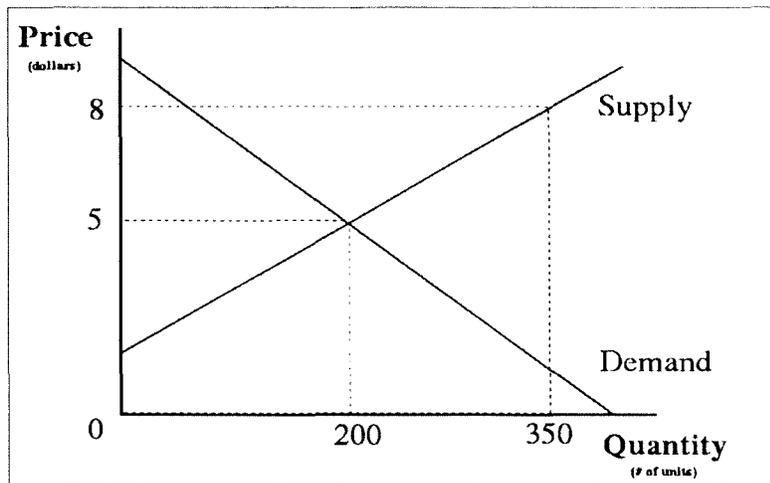


Figure 1: Supply and Demand graph for cassette tapes (Ekeleund & Tollison, p. 84)

## 2 Examples from SKETCHY

The input to SKETCHY is a graph drawn and labeled by the user. The supply-demand graph, shown in figure 1, is a typical graph found in many introductory economics textbooks. Supply-demand graphs are used to show three important relationships: how price effects the supply for the product, how price effects the demand, and the market price of the product.

To draw the graph shown in figure 1 using SKETCHY, the user starts with an empty graph consisting only of two axes. The user labels the axes as price and quantity and chooses scales for the axes. The supply and demand lines are drawn by choosing two points on the screen. SKETCHY requires the user to label the lines, so they can be referred in question and answers as well as in the graph interpretation. SKETCHY does not have any other knowledge about the labels price, quantity, supply and demand. The labels are only used to indicate the graph object that is being referred to. SKETCHY produces the following interpretation for the supply-demand graph.

For edge SUPPLY:

QUANTITY and PRICE are directly proportional.

For edge DEMAND:

QUANTITY and PRICE are inversely proportional.

Because the intersection point is not labeled, SKETCHY does not include that information in

the graph description. This information can be obtained by asking questions to SKETCHY.

Q: At what point is SUPPLY equal to DEMAND?

The SUPPLY equals DEMAND at the point when QUANTITY equals 200 and PRICE equals 5.

Q: What is the PRICE for the SUPPLY line when the QUANTITY is 350?

For SUPPLY line, when QUANTITY is 350 the PRICE is 8.

Because supply-demand graph is relatively simple, SKETCHY's description is concise. A more complicated graph taken from a thermodynamics textbook is shown in figure 2. Since all substances exhibit the same qualitative behavior shown in the pressure-volume graph, understanding this graph is essential for solving many thermodynamics problems. The curved temperature lines (31°C, 40°C and 50°C) are drawn by choosing control points for the curve. Lines with discontinuities, such as 10°C and 20°C, are drawn by combining curved and straight lines. Lines 10°C to 50°C are grouped together and identified as a temperature contour by the user since their labels indicate a third dimension to the graph. The critical point is drawn by choosing a single point and labeling it. SKETCHY infers that the critical point is on line 31°C from its location. The pressure-volume graph also has three regions— liquid, liquid-and-vapour and vapour— corresponding to the state(s) the substance is in. The regions are labeled by points at their boundaries. The complete graph interpretation SKETCHY generates is:

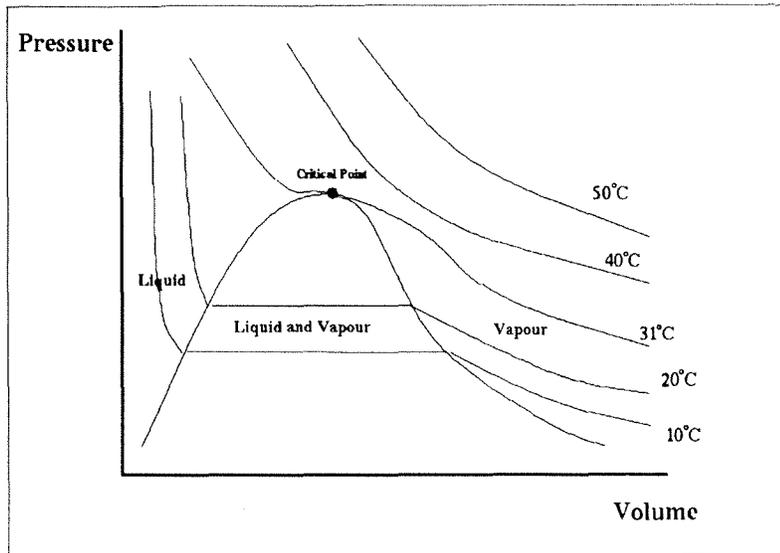


Figure 2: Compression of carbon dioxide (Whalley, p. 22)

For line 50-C:

VOLUME and PRESSURE are inversely proportional.

For line 40-C:

VOLUME and PRESSURE are inversely proportional.

For line 31-C:

VOLUME and PRESSURE are inversely proportional.

For line 20-C:

The slope of 20-C has discontinuities

*associating discontinuities with regions*

Inside region LIQUID:

VOLUME INCREASE and PRESSURE DECREASE.

Inside region LIQUID-AND-VAPOUR:

VOLUME INCREASE and PRESSURE CONSTANT.

Inside region VAPOUR:

VOLUME INCREASE and PRESSURE DECREASE.

For line 10-C:

The slope of 10-C has discontinuities.

Inside region LIQUID:

VOLUME INCREASE and PRESSURE DECREASE.

Inside region LIQUID-AND-VAPOUR

VOLUME INCREASE and PRESSURE CONSTANT.

Inside region VAPOUR:

VOLUME INCREASE and PRESSURE DECREASE.

CRITICAL-POINT is on lines (31-C)

CRITICAL-POINT is on regions (LIQUID LIQUID-AND-VAPOUR VAPOUR)

For TEMPERATURE contour:

As TEMPERATURE increases

the slopes of TEMPERATURE lines become more LINEAR.

*basis for Boyle's Law*

For a constant PRESSURE:

As VOLUME increases TEMPERATURE INCREASE.

VOLUME and TEMPERATURE are directly proportional.

For a constant VOLUME:

As PRESSURE increases TEMPERATURE INCREASE.

PRESSURE and TEMPERATURE are directly proportional.

The examples described above are interpretations of static graphs, but graphs do not have to be static. Graph designers typically superimpose graphs or show sequence of graphs to describe changes in the situation. SKETCHY demonstrates that this natural form of comparative analysis[16] can be done via visual processes on a graph.

Analyzing engineering cycles is an important task in thermodynamics. The basic cycle for a steam power plant is the Rankine cycle, shown in figure 3. A common modification to the Rankine cycle is superheating the steam in the boiler to increase the efficiency of the cycle. The net work of the cycle before modification is represented by area 1-2-3-4-1 and after modification by area 1-2-3'-4'-1.

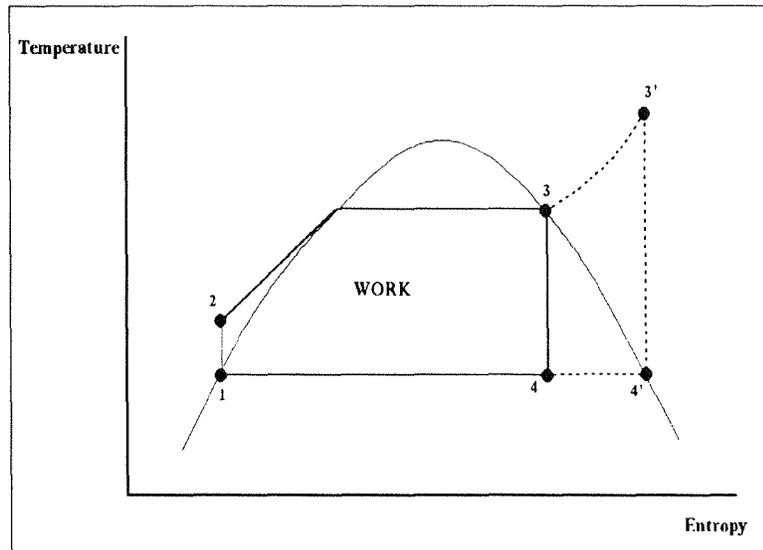


Figure 3: Effect of superheating on Rankine cycle

SKETCHY produces the following interpretation when 3 is moved to 3' and 4 to 4':

As a result of moving 3, 4:

For point 3:

The ENTROPY of 3 INCREASE.

The TEMPERATURE of 3 INCREASE.

For point 4:

The ENTROPY of 4 INCREASE.

The TEMPERATURE of 4 CONSTANT.

For region WORK:

The area covered by WORK INCREASE.

### 3 The theory underlying SKETCHY

SKETCHY demonstrates that visual reasoning via graphs is a domain independent process. Most graphs are made up of three basic elements: points, lines and regions. Although the meanings of these elements could change from one domain to another, the interactions among graph elements does not change. Just as our perceptual mechanism can easily compute the graph properties, SKETCHY derives them using geometric models to perform visual reasoning.

Qualitative results provide valuable insights to understanding of a domain. Graphs represent qualitative information effectively by presenting it in a spatial format that our perceptual mechanism can process easily. SKETCHY qualitatively describes line slopes and curvatures. A straight line represents a numerical proportionality, and a vertical or a horizontal line represents that one of the properties is constant

while the other is increasing. Smooth curves are described in terms of qualitative proportionalities[3,11]. Lines with discontinuities are described by dividing them up into qualitatively distinct regions. If the change in line slope corresponds to an intersection with another object, a relation between the intersection and the change of slope can be inferred (e.g. the existence of phase transition points).

Most of the information SKETCHY extracts from the graph can be easily represented using constructs of qualitative physics[3,11]. Directly proportional lines can be represented via qualitative proportionalities, points on the graph can be represented via correspondences. As a result, the output of SKETCHY can feed easily into qualitative reasoners.

Even when a verbal explanation is sufficient, textbooks use graphs to emphasize the verbal explanations. In these cases, although the graphs do not contain additional information, they create an additional representation in the form of an image. The image provides additional cues which makes remembering the information easier.

Our perceptual mechanisms are good at comparing the size and orientation of objects. Graphs exploit this ability to help perform comparative analysis. The modification made to increase the work output of the Rankine cycle (see Fig. 3), is an essential concept in understanding power plants. The graphical

demonstration of the modification makes a comparative argument which is a lot more lucid than a numerical argument could have been. Graphical representations take advantage of peoples ability to follow qualitative and comparative arguments by using the graph as a shared medium for communication.

Similar lines in graphs are grouped as contours by our perceptual mechanism. SKETCHY treats contour lines, identified by the user, as a single element. Any changes of slope (or curvature) among contour lines is detected in the same way changes in line slopes are detected. Because contours are generally used to represent a third dimension, SKETCHY derives the relationship between the contour and each of the variables represented on the axes.

Graphs do not have to be drawn to scale when the representing only qualitative information. A common graph convention used in the absence of scales is to assume that moving from left to right on the horizontal axis and bottom to top on the vertical axis implies an increase in the variable represented by that axis. SKETCHY uses this assumption in making qualitative interpretations when numerical scales are not specified.

To determine what people notice in graph, we have examined graph descriptions in various textbooks and identified graph properties that the authors consider important. We claim that any computer program will need to detect all of these properties to effectively understand and reason using graphs. The common graph properties are:

1. Relative orientation of points, lines and regions
2. Intersection point of lines
3. Slopes of lines
4. Changes in line slopes
5. Minimum, maximum and inflection points of lines
6. Relative size of regions

SKETCHY implements these operations and based on our sample of 65 graphs (from [17],[18] and [1]), we believe SKETCHY provides strong support that these operators are sufficient and necessary for general graph interpretation.

#### 4 Algorithms and design choices

The graph components of SKETCHY are labels, axes, points, lines, contours and regions. The

labels on the graphs provide the vocabulary for SKETCHY's interpretations. The distinction between spatial relations represented in a diagram and the interpretation of the diagram has been made in earlier work in spatial reasoning[4]. Except when there is a single component of a particular type in the graph, unlabeled components can not be referred to either by the user or by SKETCHY. The labels on the axes are required to make meaningful interpretations since axis labels represent the physical properties which provide the framework for the graph. When the axes have scales, SKETCHY includes numerical information in addition to qualitative interpretation of the graph.

Points represent discrete state information. Points can be connected to lines (whether they are on the line or not), thus move the lines when they are moved, preserving the orientation relationship between them. They also serve as boundary markers for regions and modify the shape of the region when moved. Moving a point changes the values of physical properties represented. SKETCHY interprets the modification by comparing the property values for the old and the new location.

Internally, lines are represented by an ordered list of segments (for convenience, the user can enter equations which are converted to line segments). Since segments can be of any length, using segments provides arbitrary precision for line curvatures without needing to manipulate complex equations. Like points, lines also serve as boundary markers for regions. Moving or extending the line changes the shape and the size of the region.

When the shape of the region changes, SKETCHY compares the old area to the new area in the interpretation. If the change in the region's shape results in including (or excluding) an object, this information is also included in the region. Any changes in the line slope while traversing a region is included in the graph interpretation as significant. In the pressure-volume graph, noticing that the pressure stays constant inside the liquid-vapour region and decreases outside the region is an essential observation for understanding phase changes.

SKETCHY compares the slopes of the contour lines to find any trends of changes. Lines with discontinuities, such as temperature lines 10°C and 20°C in pressure-volume graph, are approximated by curves to be able to make qualitative statements about all the contour lines.

A common intersection point of contour lines which indicates a convergence to a specific value is detected and included in SKETCHY's graph interpretation. When all contour lines exhibit similar qualitative changes in their slopes, the contour is described in terms of these qualitative changes.

Understanding a graph necessitates being able to answer questions about it. By using built in templates, SKETCHY answers questions similar to the ones in the GRE<sup>1</sup> and the SAT<sup>2</sup>. SKETCHY answers questions that involve reading values of the graph (e.g. What is the pressure when the volume is 3 and the temperature is 21?), comparing slopes, areas or locations of objects with respect to each other and describing relations of objects to each other (e.g. Is the critical point on line 20°C?).

It is especially interesting that SKETCHY's graph interpretations are so similar to explanations found in textbooks given its lack of any other understanding of the domain. The interpretations produced using object labels are meaningful and insightful for the domain. The major difference is that SKETCHY's interpretations lack the briefness of experts' explanations because SKETCHY does not differentiate between relevant and irrelevant information.

## 5 Discussion

Understanding graphs is an important part of human cognition. We have shown how visual reasoning can be used to interpret graphs in a domain independent way using geometric properties. The labels in graphs provide the vocabulary, the points and lines provide the relationships for interpreting graphs. SKETCHY produces output that can be used by qualitative reasoners and other problem solvers. The ability of SKETCHY to interpret graphs suggests that these ideas are robust.

There are a number of avenues we are exploring for extending SKETCHY. One possible extension is to extend SKETCHY's drawing environment and interpretation mechanism to other kinds of graphs, such as scatter plots bar graphs, and pie graphs. By understanding the graph properties exploited by these other graphic

representations, we hope to build a model for general graph understanding.

Another avenue we are pursuing is incorporating SKETCHY in an intelligent tutoring system where SKETCHY provides an additional communication medium between the user and the computer. The user and the computer program both can make modifications to the graph and discuss the graph relating the concepts represented with other ideas in the domain. The ideas implemented in SKETCHY can be used to construct graphs that the user will be able to interpret and reach the intended conclusions.

## 6 Acknowledgments

The research reported was supported by Office of Naval Research. I owe many thanks to my advisor, Ken Forbus, for valuable discussions, comments and encouragement. My gratitude also goes to Meryl McQueen who patiently proofread the original draft and to John Everett, Ron Ferguson and Keith Law who read subsequent drafts.

## References

1. R. B. Ekelund and R. D. Tollison. *Economics*. Little, Brown, Boston, 1986.
2. B. Faltings. Qualitative models in conceptual design: a case study. In *1st International Conference on Artificial Intelligence in Design*, 1991.
3. K. D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85-168, 1984.
4. K. D. Forbus. Qualitative reasoning about space and motion. In D. Gentner and A. L. Stevens, editors, *Mental models*, Erlbaum, Hillsdale, NJ, 1983.
5. K. D. Forbus, P. Nielsen, and B. Faltings. Qualitative spatial reasoning: the clock project. *Artificial Intelligence*, 51:417-471, 1991.
6. L. Joskowicz and E. P. Sacks. Computational kinematics. *Artificial Intelligence*, 51:381-416, 1991.
7. G. S. Novak Jr. and W. C. Bulko. Uses of diagrams in solving physics problems. In *AAAI Symposium on Reasoning with Diagrammatic Representations*, Stanford, CA, March 1992.
8. H. Kim. *Qualitative reasoning about fluids and mechanics*. PhD thesis, 1993.
9. K. R. Koedinger and J. R. Anderson. Abstract planning and perceptual chunks: elements of expertise in geometry. *Cognitive Science*, 14:511-550, 1990.

---

<sup>1</sup>Graduate Record Examination

<sup>2</sup>Scholastic Aptitude Test

10. S. M. Kosslyn. *Elements of Graph Design*. Freeman and Company, New York, NY, 1994.
11. B. J. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:289- 388, 1986.
12. J. Larkin and H. Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11:65-69, 1987.
13. T. F. McDougal and K. J. Hammond. Representing and using procedural knowledge to build geometry proofs. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 1993.
14. N. H. Narayanan and B. Chandrasekaran. A computer model of diagrammatic reasoning. In *AAAI Symposium on Reasoning with Diagrammatic Representations*, Stanford, CA, 1992.
15. S. Tessler, Y. Iwasaki, and K. Law. Qualitative structural analysis using diagrammatic reasoning. In *The Seventh International Workshop on Qualitative Reasoning about Physical Systems*, 1993.
16. D. S. Weld. *Theories of Comparative Analysis*. MIT Press, Cambridge, MA, 1990.
17. P. B. Whalley. *Basic Engineering Thermodynamics*. Oxford University Press, New York, 1992.
18. G. J. Van Wylen and R. E. Sonntag. *Fundamentals of Classical Thermodynamics*. Wiley, New York, 3rd edition, 1985.

# Learning Qualitative Models for Systems with Multiple Operating Regions\*

Sowmya Ramachandran, Raymond J. Mooney, Benjamin J. Kuipers

Department of Computer Sciences,  
University of Texas,  
Austin, Tx 78712

Email : sowmya@cs.utexas.edu

## Abstract

The problem of learning qualitative models of physical systems from observations of its behaviour has been addressed by several researchers in recent years. Most current techniques limit themselves to learning a single qualitative differential equation to model the entire system. However, many systems have several qualitative differential equations underlying them. In this paper, we present an approach to learning the models for such systems. Our technique divides the behaviours into segments, each of which can be explained by a single qualitative differential equation. The qualitative model for each segment can be generated using any of the existing techniques for learning a single model. We show the results of applying our technique to several examples and demonstrate that it is effective.

## Introduction

Qualitative reasoning is an elegant approach to studying the behaviour of a physical system without going into as much detail as in a numerical simulation. *Model building* and *model simulation* constitute the two major sub-problems of qualitative reasoning. There are several approaches to qualitative simulation, such as QSIM (Kuipers, 1986) and QPT (Forbus, 1984). Rapid advances have been made to improve the efficiency of the simulations and to fine tune them. However, the model building problem remains somewhat of an art form. Building models for a complex system requires significant knowledge of how the system works and is a time-consuming process.

Many researchers are addressing the problem of automatic model generation. One approach is to build models from existing libraries of model fragments (Forbus, 1984; deKleer and Brown, 1984;

Crawford et al., 1990; Rickel, 1992). However, these techniques still require complete knowledge of all the model fragments. Another approach is to learn the model of a physical system from observations of its behaviour. Doyle (1988); Amsterdam (1993) have proposed techniques for learning models from behaviours using existing knowledge of processes and mechanisms commonly found in physical systems. These approaches are knowledge-intensive as well.

A number of researchers have formulated techniques for generating qualitative models of physical systems from a set of qualitative behaviours using inductive techniques (Coiera, 1989; Kraan et al., 1991; Richards et al., 1992; Dzeroski and Todorovski, 1993; Bratko et al., 1991). These require little knowledge of the system being modeled. Given a set of input behaviours, these techniques generate a single *qualitative differential equation (QDE)* that is consistent with the behaviours. The models generated are represented so that they can be used by QSIM.

Many complex physical systems cannot be described by a single QDE. They are explained by different QDEs that hold under different *operating conditions* or *regions*. For example, water boiling in a closed container requires three QDEs to explain its behaviour depending on whether it is below or at its boiling and whether all of the water has evaporated. A typical behaviour shown in Figure 1 passes through several of these regions.

However, MISQ (Kraan et al., 1991; Richards et al., 1992) and the other systems cannot learn models described by multiple QDEs. Given the behaviour in Figure 2, they will incorrectly learn a single QDE that explains the entire behaviour. It is desirable to have inductive model generators that can recognise that the behaviour is best described by multiple QDEs and learn them.

This paper describes a simple technique for automatically recognising that there are multiple QDEs underlying a physical system. It assumes only the QSIM (Kuipers, 1986) formalism and is independent of the induction algorithm used to generate

\*This work has taken place in the Qualitative Reasoning Group at the Artificial Intelligence Laboratory, The University of Texas at Austin. Reason of the Qualitative Reasoning Group is supported in part by NSF grants IRI-8904454, IRI-9017047, and IRI-9216584, and by NASA contracts NCC 2-760 and NAG 9-665.

Structure: Heating liquid in a closed container.  
 Initialization: Heat water in a pot (S-1)  
 Behavior 6 of 14: (S-1 S-2 S-10 S-13 S-28 S-41 S-56 S-82 S-93 S-99 S-104).  
 Final state: (GF QUIESCENT COMPLETE), (NIL), NIL.

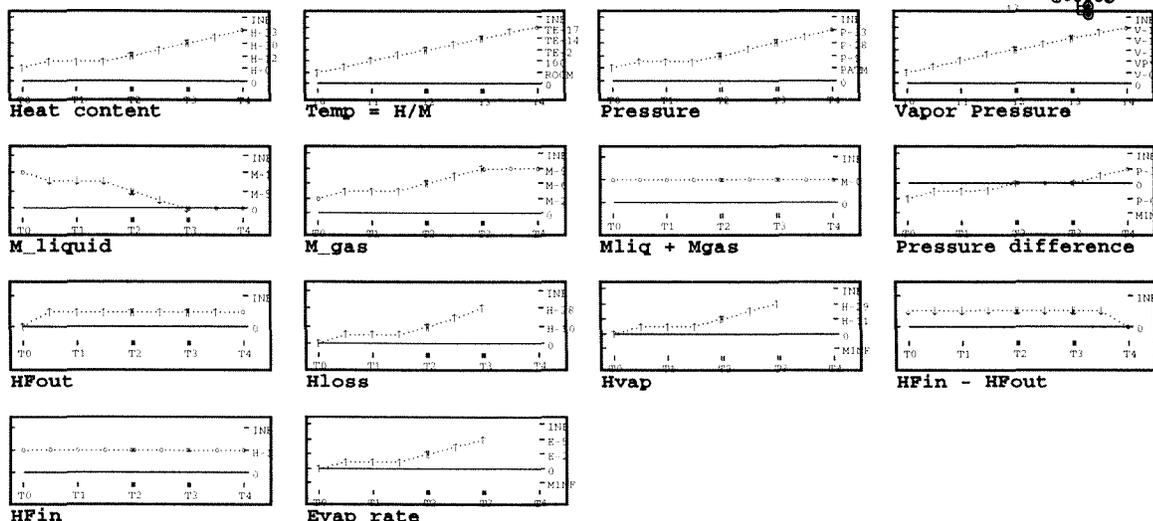


Figure 1: The Boiling-Water System: Typical Behaviour

the model. Given qualitative observations of the behaviours of a system, this technique identifies the various operating regions of the system where different QDEs hold.

We have evaluated our technique on the following physical systems: the Plant Water-balance system, the Boiling-Water system and the Divided-Tank system. Ultimately, we would like to be able to learn multiple QDEs from quantitative data as well.

In the next section, we will describe our technique in detail. Section 3 describes an experimental evaluation of this technique. In Section 4, we discuss future directions.

### Learning Models with Multiple QDEs

A QDE is valid over some operating region. The conditions, expressed in terms of the values of the variables involved, over which the QDE is valid are called the *operating conditions* of the QDE. The movement of a system from the operating region of one QDE to that of another is called a *region transition*.

Behaviours of physical systems that pass through multiple operating regions exhibit region transitions. The segment between two consecutive transitions is governed by a single QDE. Our approach

to learning models with multiple QDEs is to first break up the behaviours into such segments. Then, the system can use any of the existing induction algorithms to generate the QDE for each segment.

Thus, the problem of learning models for systems with multiple operating regions can be divided into the following sub-problems.

1. Break up the example behaviours into segments that fall within a single operating region.
2. Learn the QDE for each of the segments.
3. Identify the operating conditions for each QDE.
4. Unify the QDEs that describe the same operating region.

Although step 2 involves an induction algorithm to learn QDEs for each region, in the following subsections we describe techniques for performing steps 1, 3 and 4 that are independent of the learning algorithm used.

#### Step 1: Breaking up the behaviours into segments

To break up a behaviour into segments, it is sufficient to detect the time points where the behaviour moves from one region to another, i.e., to detect the region transitions. Our system uses the following heuristics to recognise transition points.

- **Discontinuous-Change heuristic**

One of the interpretations of a region transition is that the actual state of the mechanism undergoes a discontinuous change (Kuipers, 1994). Thus, a discontinuity in the behaviour of any variable in the system can be used to detect a region transition. A discontinuity can be any of the following kind.

1. Discontinuity in the magnitude of a variable. An example of this is a variable whose magnitude goes from being positive to being negative without going through zero.

Figure 2 shows the variable *uptake* undergoing a discontinuous change in magnitude at time point T2.

2. Discontinuity in the sign of the derivative of a variable. For instance, this happens when the derivative of a variable goes from being positive to being negative without going through zero.

For example, in Figure 2, the derivative of the variable *net inflow* undergoes a discontinuous change at time point T2.

This heuristic is justified by the fact that QSIM does not predict discontinuous behaviour unless it encounters a region transition that introduces the discontinuity. In order to learn a model that covers the given discontinuous behaviour, MISQ has to hypothesise a region transition at the point of discontinuity.

- **Non-analytic-Function heuristic**

This heuristic relies on the properties of a certain class of functions called *analytic* functions. If a function is analytic over an interval, and is constant over any open sub-interval, it must be constant over the entire interval (Kuipers, 1994). Thus, under the assumption that the actual behaviours exhibited by all the variables in the system being modeled are analytic, if a variable is observed to be constant over some interval, but not over some other interval in the same behaviour, then the two intervals must be governed by different constraints and hence different QDEs.

For example, in the behaviour shown in figure 2, the variable *turgor* exhibits non-analytic behaviour. It is non-constant over the interval (T0, T5) and is constant over the interval (T5, T6).

This heuristic is justified because, under the *analytic function assumption*, QSIM will never generate a non-analytic behaviour unless it encounters a region transition that introduces the non-analyticity.

## **Step 2: Learning the QSIM models for each operating region**

The previous section described the heuristics to break up behaviours into segments corresponding to different operating regions. The learner can now generate a single QDE to model each segment using any of the existing induction algorithms (Coiera, 1989; Kraan et al., 1991; Richards et al., 1992; Dzeroski and Todorovski, 1993; Bratko et al., 1991).

In our implementation, we have used MISQ (Kraan et al., 1991; Richards et al., 1992) to generate the QDEs. Given a set of qualitative behaviours, MISQ uses a *most specific generalisation* algorithm to generate QSIM models that are guaranteed to be consistent with the behaviours.

## **Step 3: Identifying the operating conditions for each region**

Identifying the operating condition for each QDE can be thought of as an inductive process. The behaviour segment associated with each QDE provides positive examples of the condition under which it is active. We have used a most specific conjunctive generalisation approach to induce the operating conditions from positive examples, where the operating condition induced for a region is the range of all the qualitative values observed for each variable in the behaviour segment. Table 1 shows the operating condition for the region between T0 and T1 in the behaviour shown in Figure 2.

An operating condition represents the interior of a region, whereas QSIM represents a region by its boundaries. The region of validity of each QDE is specified in terms of transition mappings. These mappings specify *transition conditions*, i.e., conditions under which the system moves out of the operating region of one QDE into that of another.

It is easy to derive the boundary conditions of each QDE from its operating conditions. We view the variables that define the boundary of a QDE as *triggers* that cause the transition out of the region. Table 4 shows some examples of triggers that cause transitions between regions. The following observations help us in identifying such triggers.

1. Since discontinuities cannot occur spontaneously, a variable that changes discontinuously cannot cause a transition.
2. A variable that is steady over a region or does not cross a landmark value cannot cause a transition.

The rest of the variables are potential triggers and the transition condition is specified as the conjunction of their boundary values.

## **Step 4: Unification of regions**

At the end of step 3, there are as many QDEs as there are behaviour segments. However, many of the segments could have the same underlying

Structure: Effect of decreasing soil moisture on plant water.  
 Initialization: Normal plant, decreasing soil moisture. (S-0)  
 Behavior 66 of 209: (S-0 S-1 S-3 S-22 S-27 S-37 S-52 S-55 S-75 S-123 S-139 S-187 S-219 S-235 S-236 S-237).  
 Final state: (TRANSITION FINAL GF COMPLETE), (TRANSITION-IDENTITY), T<INF.

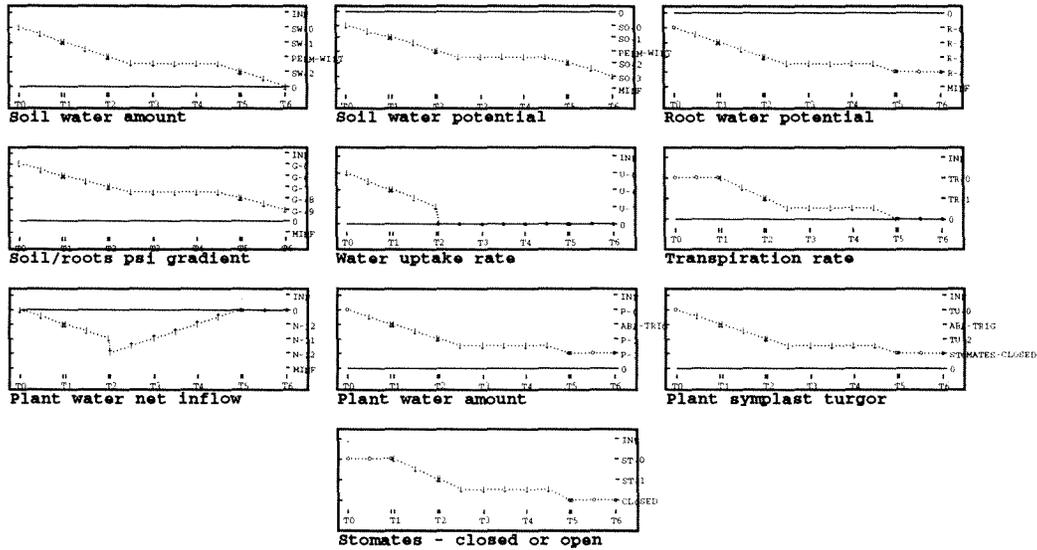


Figure 2: The Plant Water-Balance System: Input Behaviour

Variable	Qualitative interval
soil water amount	(SW-0 SW-1)
soil water potential	(SO-0 SO-1)
root water potential	(R-0 R-2)
soil/root psi gradient	(G-0 G-2)
water uptake rate	(U-0 U-6)
transpiration rate	TR-0
plant water net inflow	(0 N-12)
plant water amount	(P-0 ABA-TRIG)
plant symplast turgor	(TU-0 ABA-TRIG)
stomates	ST-0

Table 1: An Example of an operating condition

model. It is important to identify such segments and unify them.

A simple criterion to decide whether a set of regions should be unified is to check the set of constraints in the QDEs associated with the regions. In principle, two QDEs are identical if and only if their constraints sets are identical. However, since each QDE is learned inductively, two QDEs that should be identical may have different constraint sets. In practice, we have to rely on heuristics to guide region unification. We use two heuristics to identify the regions to be unified.

### 1. Identical Constraints heuristic

If two regions have QDEs with identical sets of constraints, then they are unified. The operating condition for the unified region is the disjunction of the operating conditions of the individual regions, if they are *disjoint*. Otherwise, the two operating regions are combined by combining the qualitative intervals for each variable across the regions unified. Two operating conditions are *disjoint* if and only if there is at least one variable with non-overlapping qualitative values across the two regions.

### 2. Identical Operating Conditions heuristic

Two regions are unified if they have identical operating conditions. The set of constraints defining the unified QDE is the intersection of the sets of constraints in the regions unified.

Our system applies the two heuristics repeatedly until no more regions can be unified.

## Experimental Evaluation

We have implemented this technique in a system called MISQ-RT. We have used MISQ-RT to generate multiple QDE models for the following physical systems: the Plant Water-balance system, the Boiling-Water system and the Divided-Tank system.

We will first outline our experimental methodology and then describe the results for each of the three systems.

### Methodology

We designed our experiments to test the effectiveness of our approach and our heuristics. We considered the heuristics for the identification of region transitions to have been effective if they could successfully detect all the region transitions. We evaluated the heuristics for region unification by comparing the regions generated by MISQ-RT with a model of the same physical system generated by an expert. We expected the heuristics to unify exactly those regions deemed identical by the expert. The specification of the conditions for region transitions were also evaluated by comparing the generated models with the expert model.

In each of the experiments, we generated behaviours by using QSIM to simulate the expert model. We picked a few of these behaviours as input for MISQ-RT, making sure that the selected behaviours traced different trajectories through the regions and exhibited different region transitions. The input to MISQ-RT also included totally ordered quantity spaces and dimensions for each variable.

The output from MISQ-RT was the generated model in a format that could be used by QSIM. The specification for each QDE included the constraints and the transition mappings.

QSIM indicates region transitions in the behaviours that it generates. This made it easy to check if MISQ-RT has identified all the region transitions in a behaviour.

Since the behaviours were generated by QSIM, there was a correspondence between the behaviours and the QDEs in the expert model that generated them. Thus, we could establish a correspondence between the regions generated by MISQ-RT and the expert QDEs. This was crucial in evaluating the heuristics for region unification. We expected MISQ-RT to unify only those regions that correspond to the same expert QDE.

The following subsections present the results of our experiments, followed by a discussion of the results.

## Results

**Water Boiling in a Closed Container** This experiment modeled the scenario of water being brought to boil in a closed container. The model defined by the expert had *three* QDEs, *Heating*, *Boiling* and *Gas-only*. The QDE *Heating* is active when the water is being heated up to its boiling point. *Boiling* is active when the water has reached its boiling point. *Gas-only* is active when all the water has evaporated.

Figure 1 shows one of the four inputs to MISQ-RT. The other input behaviours showed all the water evaporating before it reached its boiling point. MISQ-RT identified *three* regions as well.

Before region unification, MISQ-RT generated as many QDEs as the number of behaviour segments between transitions. In this case, it generated *nine* QDEs. Table 6 shows the correspondence between these QDEs and those defined by the expert. Table 7 shows the correspondence between the generated QDE and the expert QDE after unification. MISQ-RT identified exactly those regions corresponding to the same QDE in the expert model.

Table 4 shows the transitions for each of the QDE as defined by the expert. Table 5 shows the transitions learned by MISQ-RT for each QDE. These tables show that MISQ-RT was successful in identifying all the variables that cause transitions out of each QDE. The transition condition for the

Generated QDE	Expert QDE
r106498	Boiling
r106493	Heating
r106487	Heating
r106481	Heating
r106464	Heating
r106486	Gas-only
r106479	Gas-only
r106492	Gas-only
r106503	Gas-only

Table 2: The Boiling-Water System: Region correspondence before unification

Unified QDE	Component QDE	Expert QDE
q106521	r106498	Boiling
q106522	r106493 r106487 r106481 r106464	Heating
q106523	r106486 r106479 r106492 r106503	Gas-only

Table 3: The Boiling-Water System: Region correspondence after unification

QDE *q106521* seems overly-specific. However, it is merely redundant since the two conditions would occur simultaneously. This brings up the question of the interpretation of the landmark value M-9 for the variable *mgas*. This is discussed in the section 4.

**Plant Water-balance system** In this experiment, we modeled a plant balancing the amount of water in its system, as the level of water in surrounding soil decreases. The model defined four QDEs, *healthy-stomates-closed-uptake*, *water-stress-uptake*, *healthy-stomates-closed-no-uptake*, *water-stress-no-uptake*. The main factors that determine the region that is active at any point are (1) whether the plant is healthy, and (2) whether there is any uptake of water from the soil. When the plant is *healthy*, i.e., when the concentration of water in its system is above a threshold, the size of stomatal-opening is constant. When the plant is *water-stressed*, i.e., not healthy, the stomates start closing. When the concentration of water in the soil falls below a certain level, there is no uptake of water. When the *stomates* are closed, the size of the stomatal-opening is constant. Thus, both the *healthy* and the *stomates-closed* condition result in the same QDEs. This example is similar to the one described in (Rickel and Porter, 1992).

Figure 2 shows one of the two input behaviours to MISQ-RT. MISQ-RT identified *six* QDEs, whereas the expert model had only *four*.

Table 6 shows the correspondence between the generated QDEs and the expert QDEs. Table 7 shows the correspondence after unification.

MISQ-RT unified only those QDEs that corresponded to the same QDE in the expert model. However, it did not unify all such QDEs. It could

not identify that the QDE for the condition of a plant being *healthy* with the QDE for the condition when the *stomates* are *closed*. In region *q105961*, the plant is *healthy* but the *stomates* are open. Whereas in region *q105963*, the plant is *unhealthy* but the *stomates* are closed. These two conditions are explained by the same QDE in the expert model. However, this would affect neither the correctness of the model nor its generality.

Table 8 shows the transitions for each of the four QDEs defined by the expert. Table 9 shows the transitions for the QDEs generated by MISQ-RT. The transition conditions for the generated QDEs are more specific than for the expert QDEs. MISQ-RT did not propose transitions that did not occur in the input behaviours. For example, for the QDE corresponding to the region *water-stress-uptake*, it did not generate a transition when the variable *turgor* increases beyond the value *aba-trig*. This is because it did not encounter any behaviour with such a transition. We are investigating techniques for proposing transitions that did not occur in the input behaviours. This can be done by examining the generated regions pairwise for variables that could cause transitions between them.

**The Divided-Tank system** Figure 3 shows the Divided-Tank system. It has a tank with a partition in the middle. There is an inflow into region A and a drain in each of the regions A and B. This example is from (Soderman and Stromberg, 1991).

Figure 4 shows the behaviour tree and one of the predicted behaviours when the tank is filled from empty. QSIM predicted 7 behaviours, all of which were included in the input to MISQ-RT.

The expert model defined three QDEs, *FillA*,

From QDE	Transition Condition			To QDE
	Variable	Qmag	Qdir	
Heating	pdiff	0	inc	Boiling
	Mliq	0	dec	Gas-only
Boiling	Mliq	0	dec	Gas-only
Gas-only				No transition

Table 4: The Boiling-Water System: Transition table for the expert model

From QDE	Transition Condition			To QDE
	Variable	Qmag	Qdir	
q106522	pdiff	0	inc	q106521
	Mliq Mgas	0 M-9	dec NIL	q106523
q106521	Mliq Mgas	0 M-9	dec NIL	q106523
				No transition

Table 5: The Boiling-Water System: Transition table for the generated model

Generated QDE	Expert QDE
r105904	water-stress-uptake
r105918	water-stress-no-uptake
r105924	healthy-stomates-closed-uptake
r105899	healthy-stomates-closed-uptake
r105929	healthy-stomates-closed-no-uptake
r105934	water-stress-no-uptake
r105939	healthy-stomates-closed-no-uptake
r105923	healthy-stomates-closed-no-uptake

Table 6: The Plant Water-Balance System: Region correspondence before unification

Unified QDE	Component QDEs	Expert QDE
q105958	r105904	water-stress-uptake
q105959	r105918	water-stress-no-uptake
q105960	r105924 r105899	healthy-stomates-closed-uptake
q105961	r105929	healthy-stomates-closed-no-uptake
q105962	r105934	water-stress-no-uptake
q105963	r105939 r105923	healthy-stomates-closed-no-uptake

Table 7: The Plant Water-Balance System: Region correspondence after unification

From QDE	Transition Condition			To QDE
	Variable	Qmag	Qdir	
healthy-stomates-closed-uptake	turgor	aba-trig	dec	water-stress-uptake
	soil-psi	perm-wilt	dec	healthy-stomates-closed-no-uptake
water-stress-uptake	turgor	aba-trig	inc	healthy-stomates-closed-uptake
	stomates	closed	nil	healthy-stomates-closed-uptake
	soil-psi	perm-wilt	dec	water-stress-no-uptake
healthy-stomates-closed-no-uptake	turgor	aba-trig	dec	water-stress-no-uptake
	soil-psi	perm-wilt	inc	healthy-stomates-closed-uptake
water-stress-no-uptake	turgor	aba-trig	inc	healthy-stomates-closed-no-uptake
	stomates	closed	nil	healthy-stomates-closed-no-uptake
	soil-psi	perm-wilt	inc	water-stress-uptake

Table 8: The Plant Water-Balance System: Transition table for the expert model

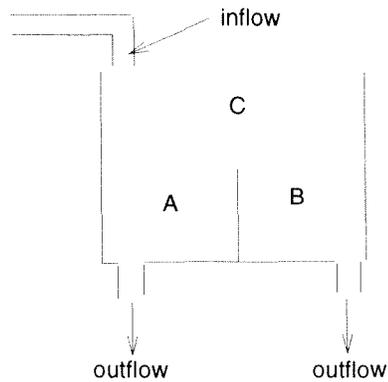


Figure 3: The Divided Tank system

From QDE	Transition Condition			To QDE
	Variable	Qmag	Qdir	
q105960	pwater	aba-trig	dec	q10598
	turgor	aba-trig	dec	
q105960	swater	perm-wilt	dec	q105961
	soil-psi	perm-wilt	dec	
	uptake	0	dec	
	netflow	n-9	dec	
q105958	swater	perm-wilt	dec	q105959
	soil-psi	perm-wilt	dec	
	uptake	0	dec	
q105961	pwater	aba-trig	dec	q105962
	turgor	aba-trig	dec	
q105963				No Transitions
q105959	root-psi	R-4	dec	q105963
	pwater	P-3	dec	
	transp	0	dec	
	turgor	stomates-closed	dec	
	netflow	0	inc	
	v	stomates	closed	
q105962	root-psi	r-4	dec	q105963
	pwater	p-3	dec	
	transp	0	dec	
	turgor	stomates-closed	dec	
	netflow	0	inc	
	stomates-closed	closed	dec	

Table 9: The Plant Water-Balance System: Transition table for the generated model

Structure: Two cascaded tanks.  
 Initialization: Fill from empty (S-0)  
 Behavior 2 of 7: (S-0 S-1 S-2 S-6 S-8 S-9 S-12 S-13 S-15).  
 Final state: (GF QUIESCENT COMPLETE), (NIL), NIL.

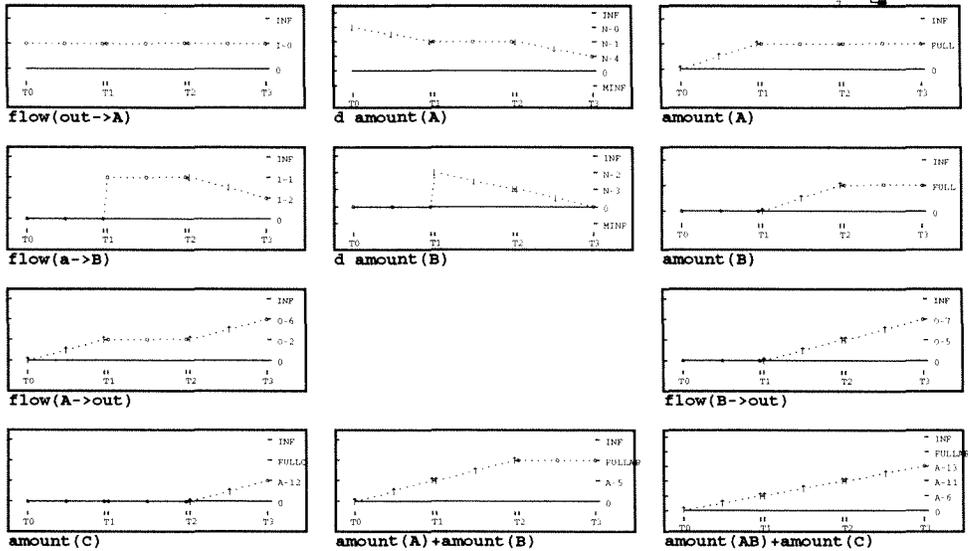
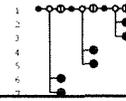


Figure 4: The Divided-Tank System: Input Behaviour

*FillB* and *Fill-both*. *FillA* is active when region A is being filled. *FillB* is active when region A is full and region B is getting filled. *Fill-both* is active when both A and B are full and region C is getting filled.

The model generated by MISQ-RT defined the same number of QDEs as the expert model. It unified exactly those QDEs that corresponded to the same QDE in the expert model. It identified all the variables responsible for causing transitions out of a QDE. Some of the transition conditions were overly-specific.

## Discussion

The outcome of the experiments showed that the heuristics for identifying region transitions in qualitative behaviours were effective. In all of the experiments, MISQ-RT identified all the region transitions. These heuristics are independent of the induction algorithm used to learn the QDEs.

The heuristics for recognising and unifying identical regions were effective in unifying a large proportion of the QDEs deemed identical by the expert. They never made the mistake of unifying regions that the expert did not consider identical.

Sometimes, the technique generated overly specific transition conditions. This could cause QSIM to miss some transitions when it uses the generated model for simulation. We are investigating techniques for avoiding this through the use of negative examples of transitions conditions, i.e., situations where a proposed transition condition did not lead to a transition.

Since we used MISQ as our induction module, the QDEs generated for each region was guaranteed to be consistent with the behaviour segments for each region.

## Future Work

### Evaluation on more complex systems

The techniques we have proposed here rely on heuristics. These heuristics have to be validated by extensive experiments. We would like to perform experiments on systems more complex than those we have studied so far. One such system is the Reaction Control System (RCS) of the space shuttle (Kay, 1992).

Although, it is not desirable to learn a single QDE to explain the behaviour of a system with multiple operating regions, generating too many QDEs would adversely affect the generality of the model. The number of regions generated should be of an order less than the number of behaviour segments in the input. We would like to study the number of behaviours generated by MISQ-RT in relation to the number of behaviour segments in the input. We would also like to evaluate the generality of the models generated by our technique by using QSIM to simulate them and see how successful they are in predicting behaviours previously unseen.

### Identifying Region Transitions from Quantitative Data

Our current implementation of the technique requires qualitative behaviours as input. However, to

be useful in modeling real systems like the RCS, the system should be able to handle quantitative data as well.

The heuristics for identifying region transitions from behaviours can be applied to quantitative data as well. Non-analytic behaviour and discontinuous changes will have to be detected from quantitative data. So far we have not investigated techniques for doing this. This task is further complicated in the presence of noise. We plan to address this issue in the near future.

### Matching Landmarks across Behaviours

The experiments have shown the *identical operating conditions heuristic* to be quite useful in unifying regions. However, the criterion for matching the operating conditions for regions is purely syntactic. Two operating conditions are considered identical if the qualitative intervals for each of the variables are identical across the two regions. Two qualitative intervals are considered identical if they are bounded by the same landmark value.

When the inputs to the learner come from different sources, landmarks may not match syntactically, even if they stand for the same event. Consider the situation where MISQ-RT is modeling a bathtub and it receives behaviours from two bathtubs, A and B. Let *FullA* and *FullB* be the landmarks for the event when A and B are full to capacity, respectively. As MISQ-RT processes the quantitative data, it has to recognise that *FullA* and *FullB* are qualitatively the same landmark values, though they have different quantitative values. If they are not recognised to be the same, the operating conditions for the QDEs proposed for the two bathtubs will not match.

There are certain landmarks like *zero* that are special and can be matched easily across behaviours. A possible approach to this problem would be to use these special landmarks and the qualitative trends in the behaviour to match the other landmarks. This is an interesting problem that could arise in other applications of Qualitative Reasoning and is worth pursuing.

### Related Work

Many techniques have been proposed for learning models of physical systems from observations of their behaviour (Coiera, 1989; Kraan et al., 1991; Richards et al., 1992; Dzeroski and Todorovski, 1993; Bratko et al., 1991). All of these, however, can only generate a single QDE model.

Falkenhainer (1990) describes a technique for building models for systems by analogy with other systems. This approach requires knowledge in the form of a library of processes. This is also true of (Rickel, 1992; Rickel and Porter, 1992) who describe a method for automatically building models from process libraries. Since they work at the

process level, they are not concerned with region transitions directly.

The machine discovery system ABACUS (Falkenhainer and Michalski, 1986) learns the mathematical equations describing a set of numerical data. It can discover multiple equations that apply under different conditions. Although their technique does learn qualitative relations between variables, its main focus is on learning quantitative laws. The quantitative laws help in recognising and learning the various operating regions of the system. MISQ-RT, on the other hand, uses qualitative heuristics to identify the different operating regions. Thus, it can be used with quantitative as well as qualitative data.

Soderman and Stromberg (1991) have proposed a technique for learning models of systems that abruptly change between linear modes of operation. They address similar issues such as identifying "jump" in behaviours, finding correspondences between various segments of the behaviour and finding conditions under which each mode is active. They use system identification techniques to detect region transitions and to fit a model for each segment. However, they have to specify the model structure in advance. Our approach does not make any assumptions about the model structure. They also require knowledge in the form of bond graphs. The models they fit are quantitative models. Our approach works on qualitative behaviours and can be used in situations where quantitative observations are not available.

Nordhausen and Langley (1993) have proposed a technique for empirically discovering the laws that govern scientific phenomena. Their system discovers both qualitative and quantitative laws. It can also identify and model the different operating regions of the system. However, the system does not break up the observations into different segments automatically. This information has to be given as input to the system. The transition conditions have also to be specified with the input.

### Conclusion

In this paper, we have proposed a method for learning models with multiple QDEs from qualitative observations of their behaviour. We have proposed heuristics to detect region transitions and for identifying corresponding regions. We have also suggested a technique for identifying the operating conditions of each QDE. The experiments reported here indicate that our approach is effective in identifying region transitions and learning models with multiple QDEs.

### Acknowledgment

We thank James Lester and Ranan Rajagopalan for comments on earlier versions of the paper. We

would also like to thank Bob Schrag for helping with the figures.

## References

- Amsterdam, J. (1993). Automated qualitative modeling of dynamic physical systems. Technical Report 1412, MIT Artificial Intelligence Laboratory.
- Bratko, I., Muggleton, S., and Varsek, A. (1991). Learning qualitative models of dynamic systems. In *Proceedings of the Eighth International Workshop on Machine Learning*.
- Coiera, E. (1989). Generating qualitative models from example behaviours. Technical Report 8901, University of New South Wales.
- Crawford, J., Farquhar, A., and Kuipers, B. (1990). QPC: A compiler from physical models into qualitative differential equations. In *Proceedings of the Eighth National Conference on Artificial Intelligence*.
- deKleer, J., and Brown, J. (1984). A qualitative physics based on confluences. *Artificial Intelligence*, 24:7-83.
- Doyle, R. (1988). *Hypothesizing Device Mechanisms: Opening Up the Black Box*. PhD thesis, Massachusetts Institute of Technology.
- Dzeroski, S., and Todorovski, L. (1993). Discovering dynamics: From inductive logic programming to machine discovery. In *Knowledge Discovery in Databases Workshop*.
- Falkenhainer, B. (1990). A unified approach to explanation and theory formation. In Shrager, J., and Langley, P., editors, *Computational Models of Scientific Discovery and Theory Formation*, 157-196. Morgan Kaufmann.
- Falkenhainer, B. C., and Michalski, R. (1986). Integrating quantitative and qualitative discovery: The ABACUS system. *Machine Learning*, 1:367-401.
- Forbus, K. (1984). Qualitative process theory. *Artificial Intelligence*, 24.
- Kay, H. (1992). A qualitative model of the space shuttle reaction control system. Technical Report AI92-188, The University of Texas at Austin.
- Kraan, I. C., Richards, B. L., and Kuipers, B. J. (1991). Automatic abduction of qualitative models. In *Proceedings of the Fifth International Workshop on Qualitative Reasoning about Physical Systems*, 295-301.
- Kuipers, B. (1986). Qualitative simulation. *Artificial Intelligence*, 29:289-338.
- Kuipers, B. (1994). *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. MIT Press. In Press.
- Nordhausen, B., and Langley, P. (1993). An integrated framework for empirical discovery. *Machine Learning*, 12:17-47.
- Richards, B. L., Kraan, I., and Kuipers, B. J. (1992). Automatic abduction of qualitative models. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 723-728.
- Rickel, J. (1992). Automated modeling for answering prediction questions: Exploiting interaction paths. Technical Report AI92-178, University of Texas at Austin.
- Rickel, J., and Porter, B. (1992). Automated modeling for answering prediction questions: Exploiting interaction paths. In *Proceedings of the Sixth International Workshop on Qualitative Reasoning about Physical Systems*, 82-95. Heriot-Watt University. Edinburgh, Scotland.
- Soderman, U., and Stromberg, J.-E. (1991). Combining qualitative and quantitative knowledge to generate models of physical systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, 1158-1163.

# An Investigation on Domain Ontology to Represent Functional Models

Munehiko SASAJIMA<sup>†</sup>, Yoshinobu KITAMURA<sup>†</sup>, Mitsuru IKEDA<sup>†</sup>  
Shinji YOSHIKAWA<sup>‡</sup>, Akira ENDOU<sup>‡</sup> and Riichiro MIZOGUCHI<sup>†</sup>

<sup>†</sup>The Institute of Scientific and Industrial  
Research, Osaka University,  
8-1, Mihogaoka, Ibaraki, 567 Japan  
sasajima,kita,ikeda,miz@ei.sanken.osaka-u.ac.jp

<sup>‡</sup>Power Reactor and Nuclear Fuel  
Development Corporation,  
4002 Narita-Cho, Oarai-Machi,  
Ibaraki Pref, 311-13 Japan

## Abstract

Although a lot of researchers have pointed out the significance of functional representation, the general relations between function and behavior is not fully understood yet. We consider the knowledge of each component in a system as consisting of two elements. One is a necessary and sufficient concept for simulation of the component which we call behavior. The other is the interpretation of the behavior under a desirable state which the component is expected to achieve, which we call function. By classification of a primitives necessary for the interpretation of the behavior in various domains, which we call "domain ontology", we can capture and represent the function by selection and combination of the primitives. This paper proposes the primitives we identified and the method to use them for representing function. Also we investigate the relation between function and behavior based on the primitives. As the primitives can represent concepts at various levels of abstraction, they will contribute to those tasks which rely on the simulations on the model of the target object, such as diagnosis, design, explanation, and so on.

## Introduction

Model-based simulation has been utilized for solving various problems, such as diagnosis (Kitamura *et al.* 1994) (Hirai *et al.* 1991) (Sembugamoorthy & Chandrasekaran 1986) (Abu-Hanna, Benjamins, & Jansweijer 1991), design (Vescovi *et al.* 1993) (Iwasaki *et al.* 1993), explanation (Swartout, Paris, & Moore 1991) (Gruber & Gautier 1993) and so on. In order to promote such model-based problem solving, the concepts of behavior and function have to be understood in depth, since they provide us with a firm foundation of the methodology.

The following two reasons support the significance of the research about the concept of function.

1) Investigation on the concept of function contributes

to fault diagnosis. The task deals with the concept of trouble of a component which can be regarded as a loss of its function. Capturing the concept of trouble and hence function is necessary to achieve efficient diagnosis.

2) Investigation of function also contributes to explanation of those tasks performed by expert systems. Using the concept of function, an expert system can reason how a component's behavior is significant in the system. Furthermore, the system can explain the system's behavior in terms of those concepts familiar to the users at an appropriate level of abstraction, which helps them understand the explanation.

Another issue to discuss is clear understanding of the difference between behavior and function. When we interpret behavior of a component, we employ certain viewpoints from which more than one interpretation is made for a behavior of a component. Necessity of a specific output of the component is an example of the viewpoint. For example, suppose a component whose behavior is to divide an input saline solution into pure salt and a saline solution. The behavior is interpreted as producing salt in the system which requires salt. In the system which requires fresh water, however, the same behavior is interpreted as desalinization.

A lot of research has been done to date aiming at a deep understanding of both behavior and function. J.de Kleer (de Kleer 1984) defines function of a component as a combination of behavior and selection of it from the set of possible behaviors of the component. Although the information required for the selection can be one of the candidate factors necessary to interpret behaviors, it is not enough. Two different interpretations, for example, are possible for one behavior of a heat exchanger, "shift thermal energy from fluid of higher temperature side to lower temperature side". One is that "the component gets the fluid of the lower side warmer", and the other is "the component gets the fluid of the higher side colder". de Kleer's definition of function does not explain this example.

V.Sembugamoorthy and B.Chandrasekaran(Sembugamoorthy & Chandrasekaran 1986) propose a framework to represent the function of a system by combination of function and behavior of each component of the system. In their work, declaration of a desired state and the environment in which the function of the system appears can be regarded as attached information. As relations between the attached information and the concept of the function are not discussed so much, proposed framework seems to have much room to refine.

Anne M.Keuneke(Keuneke 1991) classifies function into four concepts, such as To Make,To Maintain,To Prevent,and To Control. We regard them as concepts at the top level of function hierarchy which should be refined further from various view points.

The discussion we have made thus far shows there is no satisfactory theory about behavior and function in spite of its importance. We classify the knowledge about each component of a model into three elements. The first element is required to have necessary and sufficient information enabling simulation of how the the system works, by combining all the elements in the system without referring to the other components or the whole system. We use the term "behavior" to refer to this element.

Next, we recognize intended desirable states which each component is expected to achieve. Such a state is the second element, and we use term "goal" to refer to it.

Lastly, we interpret the behavior of each component under a related goal. We use the term "function" to refer to the interpretation result (the third element). Although function of a component cannot be defined without referring to the whole system, it can be described for each component.

According to the above discussion, we come up with the following definition of function:

**function = behavior + attached information**

in which attached information is some information which make function different from behavior.

Identification of primitives to represent the attached information plays a critical role in capturing the concept of function. When the identification is accomplished, functional representation of a component is easily made by selection and combination of the primitives. Furthermore, to make the primitives domain-independent enhances re-usability of them which constitute a portion of "Domain Ontology".

One of the bottlenecks in model building is the existence of a gap between model builder's concepts about a component and behavior of the component to be represented. Identification of the primitives to represent function helps decrease the gap.

Our long term goal includes to organize the concepts of function and behavior as reusable building blocks for qualitative models which facilitates model-based problem solving. As the first step toward this goal, we

investigate the primitives for describing attached information and organize them as a portion of domain ontology. This paper proposes a new method to describe models of behavior and function of components. Also we show potential of the method by describing models on different domains at various levels of abstraction.

## Primitives to represent the function

To establish a framework to describe a functional model of components, we investigate several viewpoints to capture the behavior and the additional information to interpret it. In this section, we discuss the viewpoints and primitives to represent them in order.

## Overview of the components description

There are two ways of behavior representation. One is process-centered way which views how the substances under consideration are processed. The other is device-centered way which concentrates on input-output relations of a component. We employ the latter way and represent each component as a black box which has ports for input and output. Some components, e.g. a tank, have only ports for input and other components, e.g. a battery, have only ports for output. We call an input substance "In-Obj" and an output substance "Out-Obj". Except the cases in which clear distinction is necessary, we use the term "Compo-Obj" to refer to one of those substances treated by a component. We treat substances (such as water) and energy (such as heat) which exhibit functionality of the component as Compo-Objs. Description of them is based on object-oriented paradigm.

## Description of the behavior

The behavior of a component is represented by its Compo-Objs and relations among them. This section describes how to describe them in order.

**Description of Compo-Objs** Two types of Compo-Obj are discussed: one consists of only one kind of substances or energy referred to as basic Compo-Obj and the other consists of more than one kind of substances or energy referred to as mixed Compo-Obj.

**Basic Compo-Objs.** The following six viewpoints are employed to represent basic Compo-Obj.

- The Compo-Obj is energy or substance
- The location at which the Compo-Obj exists
- Super class of the Compo-Obj
- Parameters which represent the Compo-Obj
- Relations among the parameters
- Phase of the Compo-Obj

**Mixed Compo-Objs.** There are Compo-Objs which consist of more than one kind of Compo-Objs.

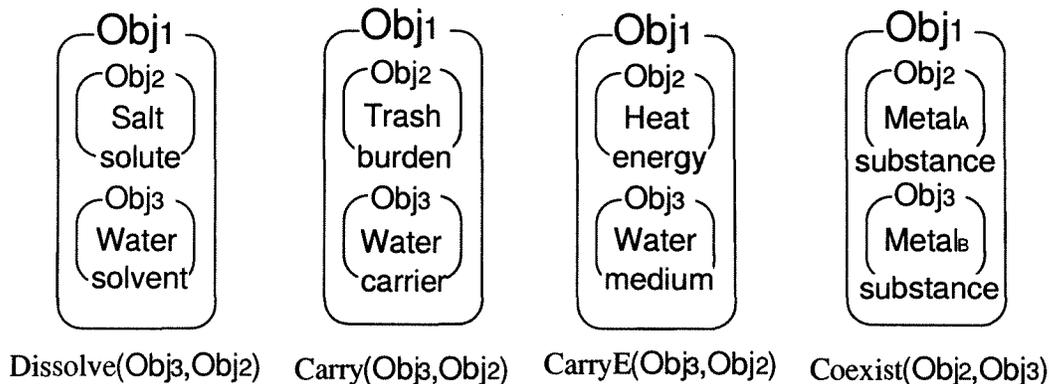


Figure 1: Mixed Compo-Objs

For example, a saline solution consists of salt and water and an alloy consists of metal<sub>A</sub> and metal<sub>B</sub>. We call such a Compo-Obj as a Mixed Compo-Obj and each of those Compo-Objs which together compose a Mixed Compo-Obj as a Compo-Obj-Composer. As an example of saline solution shown in Fig.1, we capture Mixed Compo-Obj as a Compo-Obj which includes Compo-Obj-Composers with their special relations. We employed four kinds of such relations, and represent them by predicates as follows.

#### Solute and solvent

One of the two Compo-Obj-Composers dissolves the other Composer like a saline solution.

Predicate: **Dissolve**(Obj<sub>x</sub>, Obj<sub>y</sub>)

Obj<sub>x</sub>:solvent, Obj<sub>y</sub>:solute

#### Carrier and Burden

Although not being in "dissolve" relation, one of the two Compo-Obj-Composers relies its mobility on the other Composer like trash in water flow.

Predicate: **Carry**(Obj<sub>x</sub>, Obj<sub>y</sub>)

Obj<sub>x</sub>:carrier, Obj<sub>y</sub>:burden

#### Medium and Energy

One of the two Compo-Obj-Composers carries the energy class of Composer as a medium. Example here is water and heat energy that together compose boiling water.

Predicate: **CarryE**(Obj<sub>x</sub>, Obj<sub>y</sub>)

Obj<sub>x</sub>:medium, Obj<sub>y</sub>:energy

#### Substance<sub>A</sub> and Substance<sub>B</sub>

Two Compo-Obj-Composers coexist in one Compo-Obj and give no explicit effect to each other like two kinds of metals in an alloy.

Predicate: **Coexist**(Obj<sub>x</sub>, Obj<sub>y</sub>)

Obj<sub>x</sub>:substance<sub>A</sub>, Obj<sub>y</sub>:substance<sub>B</sub>

We represent the relation between water and salt in a saline solution, for example, as

**Dissolve**(water, salt)

In this paper, we use the term Compo-Obj to refer to both a Compo-Obj consisting of one kind of substances

or energy and a Mixed Compo-Obj unless there is a necessity to distinguish them.

**Relations among parameters.** The amount of energy which a Compo-Obj possesses has a close relation to parameters representing the Compo-Obj itself.

For example, consider boiling water (water brings heat energy). The amount of heat energy is in proportion to the temperature and volume of the water which brings the heat. Explicit description of this relation allows us to reason the transition of the amount of heat energy by transition of parameters of water.

**Format for describing Compo-Objs.** According to the above discussions, we come up with the following template for Compo-Obj representation (Fig.2):

Class Name: Compo-Obj

Attributes:

Name: ID of the Compo-Obj.

ISA: Super class of the Compo-Obj: Sodium, Water, Heat, etc.

Params: Parameters to represent physical feature of the Compo-Obj.

E-Flag: If the Compo-Obj is energy then T.

Location: The location where the Compo-Obj exists: In, Out, etc.

Para-Relations: Relations among the Params of the Compo-Obj.

Phase: Phase of the Compo-Obj: Solid, Liquid, Gas.

Sub-Objs: The Compo-Obj-Composers of the Compo-Obj.

Sub-Relations: Relations among the Compo-Obj-Composers.

Figure 2: Template for describing a Compo-Obj

#### Description of relations among Compo-Objs

To represent relations among Compo-Objs in various levels of abstraction, we employ three viewpoints discussed below. Fig.3 represents an abstract model of an ideal turbine without loss of energy. With this example, this section describes the viewpoints in order.

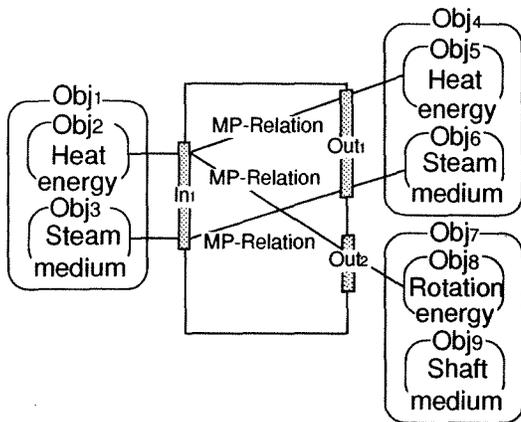
**MP-Relations: Relations among In-Objs and Out-Objs.** Some Out-Objs are made from specific

In-Objs. For example, we can recognize water produced by a desalination plant is made from saline solution. We call a set of such relations of a component as MP-Relation, and the following predicate represents them.

$MP([Out-Obj_1, \dots, Out-Obj_N], [In-Obj_1, \dots, In-Obj_M])$

In Fig.3, part of heat energy(Obj<sub>2</sub>) is converted to rotation energy(Obj<sub>8</sub>) of the shaft and rest of the heat energy is out without conversion. Thus the following predicates together represent MP-Relation of this turbine.

$MP([Obj_5, Obj_8], [Obj_2])$   
 $MP([Obj_6], [Obj_3])$



MP-Relations:

$MP([Obj_5, Obj_8], [Obj_2]), MP([Obj_6], [Obj_3])$

SameClass:

$Obj_2.ISA = Obj_5.ISA, Obj_2.ISA \neq Obj_8.ISA, Obj_3.ISA = Obj_6.ISA$

QN-Relations:

$Obj_8.Amount = k * Obj_2.Amount + C$   
 $Obj_2.Amount = Obj_5.Amount + Obj_8.Amount$   
 $Obj_6.Amount = Obj_3.Amount$   
 $Obj_6.Velocity = Obj_3.Velocity$   
 ... ..

Figure 3: Model of a turbine

**SameClass:the sameness of the class.** The behavior of the turbine in Fig.3 is viewed as to convert input energy to different kind of energy at an abstract level. This kind of relation is partially characterized from the viewpoint of the sameness of the class between the two Compo-Objs. We describe the sameness by two operators, = and ≠. In Fig.3, the following relations exist.

$Obj_2.ISA = Obj_5.ISA$   
 $Obj_2.ISA \neq Obj_8.ISA$   
 $Obj_3.ISA = Obj_6.ISA$

**QN-Relations:Quantitative relations.** Relations among the Params of Compo-Objs are represented by a set of equations, called QN-Relations.

In Fig.3, Obj<sub>8</sub>.Amount, the amount of rotation energy to be output, is in proportion to Obj<sub>2</sub>.Amount, the amount of heat energy. At the same time, the summation of the amount of output heat energy and rotation energy equals to the amount of the input heat energy according to the assumption of no loss of energy. The following two equations reflects these relations, respectively.

$Obj_8.Amount = K * Obj_2.Amount + C$

$Obj_2.Amount = Obj_5.Amount + Obj_8.Amount$

### Description of the attached information

In order to describe the function of components, we investigated the attached information and obtained the primitives from four viewpoints: (1)goal of the component (2)function type (3)focus on Compo-Objs (4)necessity of Compo-Objs. These four items are explained in this subsection.

**Goal of the component** We recognize a desirable state to achieve for each component in a system and the term “goal” refers to the concept. There are two types of such states. One is described by the combination of parameters and desirable values and represents its desirable state absolutely, that is, independently of input. The other is described by input-output relations and represents the desirable state relatively to the input of the component. Example here is the heat exchanger in Fig.4. A goal description “temperature of the output coolant(Obj<sub>10</sub>) does not exceed five hundred degree centigrade” is an example of the former type of the goal and “the heat energy of the coolant (Obj<sub>1</sub>) is decreased into one tenth by the other input coolant (Obj<sub>4</sub>)” is an example of the latter type. We represent such a desirable state by the predicate  $G(state)$ .

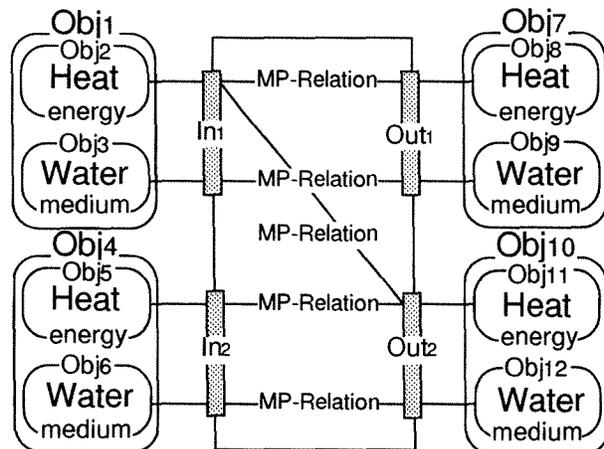


Figure 4: Heat exchanger

**Function type** Anne M.Keuneke classifies concept of the function into four types, such as To Make, To Control, To Maintain, and To Prevent(Keuneke 1991). For our representation, we give informal definition to them.

**To Make:** To set a parameter at a desirable value.

**To Control:** To shift a parameter of desirable value to another desirable one.

**To Maintain:** To keep the value of parameters desirable for certain period.

**To Prevent:** Not to make parameters to take special values which represents no good states of the system.

To Make type function is the basis of all other functions which makes parameters to be in a desirable range. Function of a cooking stove, for example, is considered as To Make temperature of a thing put on it to be more than a certain degree. If the aim of the stove is to boil a kettle of water, then the function To Make is achieved when the temperature inside the kettle exceeds one hundred degree centigrade. On the other hand, the function of an electric water heater with a sensing device which accurately makes the temperature of the water in it to be ninety five degree centigrade is To Control.

Function of a component which keeps desired state by To Control function is To Maintain.

A component whose function is To Prevent watches and controls parameters not to go undesirable states for the system. A relief valve attached to a tank To Prevent explosion of the tank watches pressure caused by substances inside the tank. The valve switches its behaviors: not to let the substances pass the valve out of the tank and to release the substances through the valve according to the pressure inside the tank.

Combining the description of the goal and the function type, we obtain an abstract explanation pattern of the function of a component, such as "function type" + "goal state" whose instance is "To Prevent Temperature of Compo-Obj<sub>N</sub> becomes ninety five degree centigrade".

Using the goal state and function type concept of trouble of a component is classified into four types:

**ToMake:** Trouble if the component does not achieve any goal.

**ToControl:** Trouble if the component in a goal does not shift to another goal.

**ToMaintain:** Trouble if the component does not keep on the achieved goal for certain period.

**ToPrevent:** Trouble if the goal parameter takes specific value which represents no good state.

**Focus on Compo-Objs** When we capture the main function of a component, we focus on a specific Compo-Obj's class. We represent such a concept by the predicate **Focus**(class of Compo-Obj). When we interpret the behavior of the component in Fig.5 as that of a resistor which lowers potential of input direct current

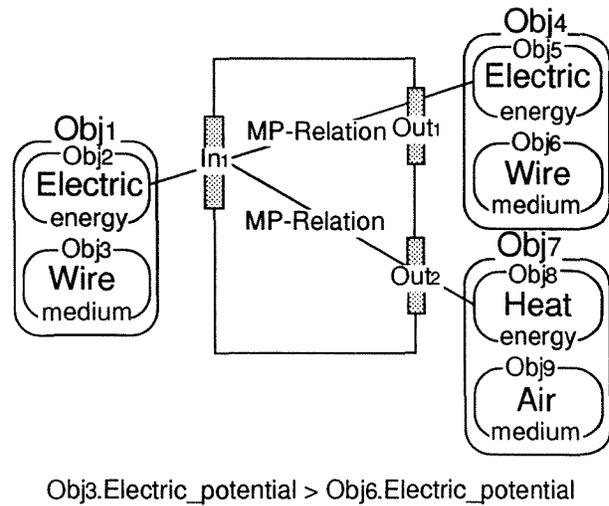


Figure 5: Resistor

electricity, focus is given to the electric energy and **Focus**(Electric energy) represents it. On the other hand, the same behavior is interpreted as that of an electric heater when we focus on the heat energy.

**Necessity of Compo-Objs** Compo-Objs which belong to the focused class often exists at different ports of a component from each other. For example, consider a behavior of the heat exchanger in Fig.4 which focuses on heat energy existing at the four ports. The heat exchanger can be interpreted not only as a heater giving the heat energy to the colder Out-Obj(Obj<sub>10</sub>), but also as a cooler taking the heat energy of the hotter In-Obj(Obj<sub>2</sub>) away.

Difference between the two interpretations is caused by difference of the necessity of each heat energy at different port, according to the goal of the component. We represent a focused class of Compo-Obj is necessary at a port by the predicate **Need**(name of the port, focused class) and not necessary by the predicate **NoNeed**(name of the port, focused class). When we interpret the behavior of the component in Fig.4 as that of heater, Obj<sub>11</sub> is necessary at the port Out<sub>2</sub>, thus **Need**(Out<sub>2</sub>, Heat) is suitable. Also when we interpret the same behavior as that of cooler, Obj<sub>8</sub> is not necessary at the port Out<sub>1</sub>, thus **NoNeed**(Out<sub>1</sub>, Heat) is suitable.

In some cases ports which do not deal with focused class of Compo-Objs play important roles. Consider Fig.5 as an abstract model of a resistor. Used in an ordinary circuit, we do not have to care the heat energy output from the resistor. Used in a precise circuit, however, the heat energy gives a harm to the circuit. Taking such port and harmful Compo-Obj's class as arguments of **NoNeed**(/2), the side effect of a component can be explicitly represented. For example, the

resister's side effect is represented by  
 $\text{NoNeed}(\text{Out}_2, \text{Heat})$ .

### Format to describe components

According to the above discussion, we propose a template for describing the behavior and the function of a component(Fig.6).

Class Name: Component  
 Attributes:  
 Ports:  
 Compo-Objs:  
 MP-Relations:  
 SameClass:  
 InherentParams:  
 QN-Relations:  
 Goal:  
 FuncType:  
 Focus:  
 Needs:

Figure 6: Template for describing a component

### Evaluation of the representation method

One of the causes of difficulty in model building is existence of a gap between conceptual level of model builder and that of a vocabulary supplied by a system for model building. A vocabulary to fill the gap is required to represent components at various levels of abstraction. This section evaluates our representation method from this viewpoint.

### Hierarchical classification of the function and the behavior

In the last section, we defined a format to represent a component captured from various viewpoints. Using these viewpoints, we came up with a hierarchical classification of the concept of the behavior and the function as shown in Fig.7.

A concept of a component which deals with one In-Obj and two Out-Objs is represented as the root node. The concept is classified into five concepts at the first level of the tree according to the condition of the composition of the In-Obj. According to the condition of composition of the two Out-Objs and their relation to the In-Obj, each node made by the first division is further classified at the second level of the tree.

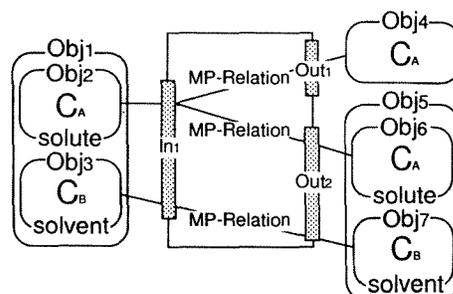
Classifications at the first and the second level are based on those viewpoints used for behavior description and is domain- and context-independent.

According to the viewpoint of focus on the class and necessity of each focused Compo-Obj, concepts represented at the second level are classified into the third level concepts. Some of the concepts represented at

the third level can be connected to those verbs we often use to represent the concept. As is discussed below, we see there still remain several viewpoints to be investigated other than we have already done. Identification of those primitives to represent function from the viewpoints will enable us to promote further classification beyond the third level.

### Representation at various levels of abstraction

Here we show some examples to evaluate the proposed representation method. The concept of a behavior "an In-Obj is a solution, one of the Out-Objs is pure solute derived from the In-Obj, and the other Out-Obj is rest of In-Obj", (represented as the node with thick circle at the second level in Fig.7) which is represented in Fig.8, is described as follows (Fig.9).



Focus(CA)

Need(Out<sub>1</sub>, CA) → Extract solute from In-Obj

NoNeed(Out<sub>2</sub>, CA) → Dilute In-Obj

Dilute In-Obj + G(Obj<sub>6</sub>.Mass=0) → Purify In-Obj

Figure 8: divide a solution

Now let CA be a class identifier to which Obj<sub>2</sub>, Obj<sub>4</sub> and Obj<sub>6</sub> belong (Fig.8). Attaching Focus(CA) and Need(Out<sub>1</sub>, CA) to the above description of the behavior changes it into the description of the function corresponding to "Extract solute from In-Obj". Replacing Need(Out<sub>1</sub>, CA) by NoNeed(Out<sub>2</sub>, CA) the last description of the function changes into different function, "Dilute In-Obj". Furthermore, attaching G(Obj<sub>6</sub>.Mass = 0) to the description of "Dilute In-Obj", the function changes into another function, "Purify In-Obj".

Attaching the following descriptions(Fig.10) to the function, "Purify In-Obj", the function of a kidney as a component of the human to maintain purifying blood by filtering waste dissolved in blood is represented.(Fig.11)

Replacing some parts of the description of the function of a kidney by the following descriptions(Fig.12), desalinization system as a component of the plant to make saline solution desalt is represented.(Fig.13)

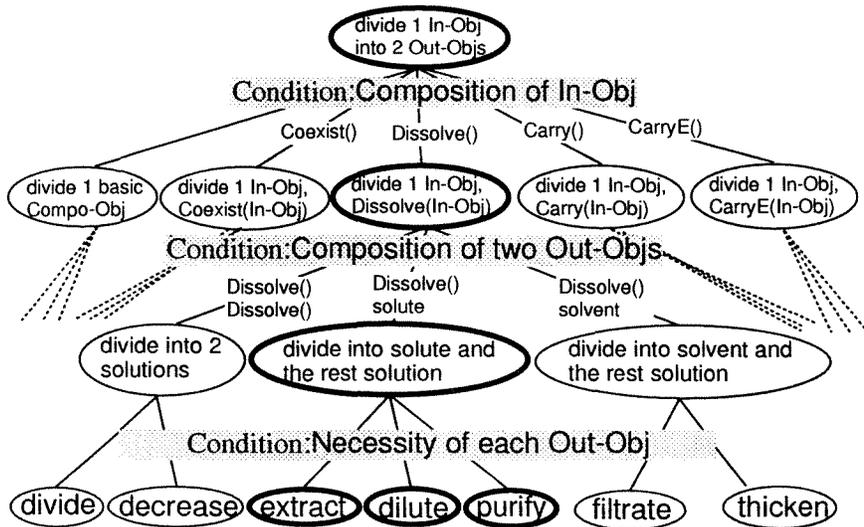


Figure 7: Classification of the behavior and function(part)

**Objects:**

**Name:**Obj<sub>1</sub>;**Location:**In<sub>1</sub>;**Phase:**Liquid;  
**Params:**[Mass,Volume,Calory,Velocity,...];  
**Sub-Objects:**Obj<sub>2</sub>,Obj<sub>3</sub>;  
**Sub-Relations:**Dissolve(Obj<sub>3</sub>,Obj<sub>2</sub>);

[**Name:**Obj<sub>2</sub>;**Location:**In<sub>1</sub>; **Phase:**Solid;  
**Params:**[Mass,Volume,Calory,Hardness...]; ]

[**Name:**Obj<sub>3</sub>;**Location:**In<sub>1</sub>;**Phase:**Liquid;  
**Params:**[Mass,Volume,Calory,Velocity,...];]

[**Name:**Obj<sub>4</sub>; **Location:**Out<sub>1</sub>; **Phase:**Solid;  
**Params:**[Mass,Volume,Calory,Hardness,...];]

[**Name:**Obj<sub>5</sub>; **Location:**Out<sub>2</sub>; **Phase:**Liquid;  
**Params:**[Mass,Volume,Calory,Velocity,...];  
**Sub-Objects:**Obj<sub>6</sub>, Obj<sub>7</sub>;  
**Sub-Relations:**Dissolve(Obj<sub>7</sub>,Obj<sub>6</sub>);]

[**Name:**Obj<sub>6</sub>; **Location:**Out<sub>2</sub>; **Phase:**Solid;  
**Params:**[Mass,Volume,Calory,Hardness,...];]

[**Name:**Obj<sub>7</sub>; **Location:**Out<sub>2</sub>; **Phase:**Liquid;  
**Params:**[Mass,Volume,Calory,Velocity,...];]

**MP-Relations:**

MP([Obj<sub>4</sub>,Obj<sub>6</sub>],[Obj<sub>2</sub>]),  
 MP([Obj<sub>7</sub>],[Obj<sub>3</sub>])

**SameClass:**

Obj<sub>4</sub>.ISA = Obj<sub>2</sub>.ISA,  
 Obj<sub>6</sub>.ISA = Obj<sub>2</sub>.ISA,  
 Obj<sub>7</sub>.ISA = Obj<sub>3</sub>.ISA

**Obj<sub>2</sub>.ISA:** Waste  
**Obj<sub>3</sub>.ISA:** Blood  
**Obj<sub>4</sub>.ISA:** Waste  
**Obj<sub>6</sub>.ISA:** Waste  
**Obj<sub>7</sub>.ISA:** Blood  
**QN-Relations:** Obj<sub>4</sub>.Amount ≤ Obj<sub>2</sub>.Amount  
 Obj<sub>6</sub>.Amount ≤ Obj<sub>2</sub>.Amount  
 Obj<sub>7</sub>.Amount = Obj<sub>3</sub>.Amount  
**Focus:** Focus(Waste)  
**FuncType:** To Maintain

Figure 10: Additional values for description of Kidney

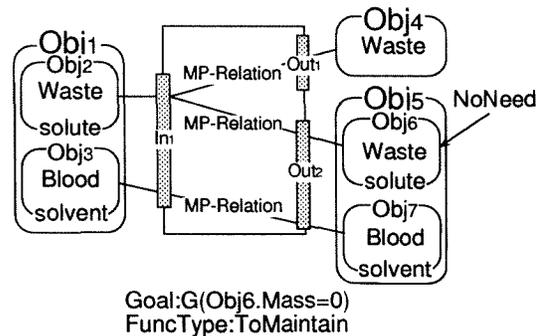


Figure 11: The function of a kidney

Figure 9: description of the concept, "divide a solution (part)"

**Obj2.ISA:** Salt  
**Obj3.ISA:** Water  
**Obj4.ISA:** Salt  
**Obj6.ISA:** Water  
**Obj7.ISA:** Salt  
**QN-Relations:** Obj4.Amount  $\leq$  Obj2.Amount  
 Obj6.Amount  $\leq$  Obj2.Amount  
 Obj7.Amount = Obj3.Amount  
**Focus:** Focus(Salt)  
**FuncType:** To Make

Figure 12: Additional values for description of a desalinization system

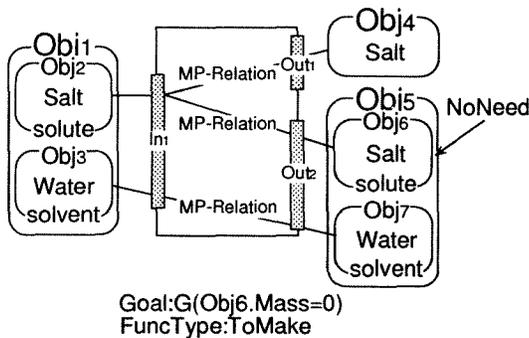


Figure 13: The function of a desalinization plant

We can describe a model of a component for simulation by attaching detailed relations among its Compo-Objs and parameters inherent to the component.

As demonstrated in this section, the representation method shows its potential to describe components of various domains at various levels of abstraction.

### Example of functional modeling(2)

Here we show another example of the representation of different interpretations of one behavior.

Figure 14 shows a model of behavior of a component which converts direct current electricity input(Obj2) to increases force(Obj5) and reaction(Obj6) of a fluid(Obj7) from another input port.

Suppose Obj7 is air. Focusing the force, represented by the predicate **Focus(Force)**, behavior of the component is interpreted as that of an electric fan which raises wind. Next, suppose Obj7 is water. Focusing the reaction from the water, the same behavior is interpreted as that of a screw gain driving force of the component itself. The predicate

**Focus(Reaction)** represents it.

### Example of functional modeling(3)

Figure 15 is a behavior model of a heat exchanger(HX). Suppose the HX is used in two ways, for cooling system

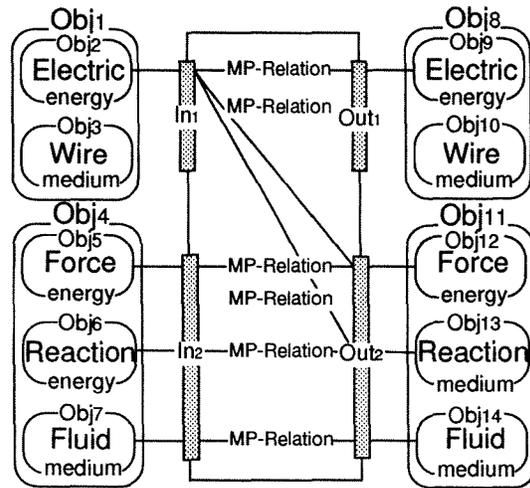


Figure 14: Behavior model of an electric fan and a screw

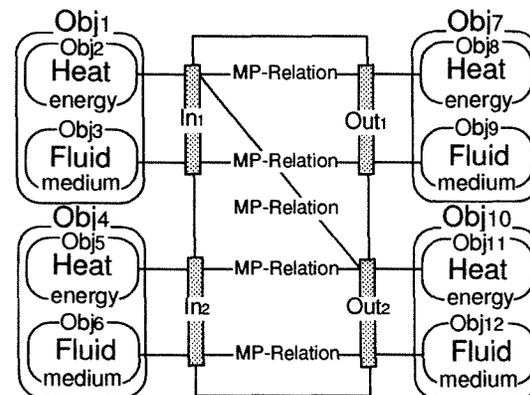


Figure 15: Behavior model of a heat exchanger

of an engine and for heating system of a room.

When the HX is used as a cooling system of an engine, its behavior is interpreted as taking away the heat of the coolant(Obj1) of engine. This interpretation is based on the idea that the heat energy of the output coolant, Obj7, is not necessary for the engine, connected to the port Out1. This interpretation is represented by the predicate,

**NoNeed(Out1,Heat Energy).**

On the other hand, when the HX is used as a heater of a room, its behavior is interpreted as heating the air of the room(Obj4). This interpretation is based on the idea that the heat energy of the output air, Obj10 is necessary for the room, connected to the port Out2. This interpretation is represented by the predicate,

**Need(Out2,Heat Energy).**

Some coolers achieve their function as air condition-

ers, for example, by maintaining the temperature of a room, and other coolers achieve their function as coolant systems which, for example, prevent overheat of an integrated circuit. Functional model of the former type of cooler can be represented by attaching the following two predicates to the functional model of the cooler.

**Goal:**G(Obj<sub>7</sub>.Temp = k)

**FuncType:**ToMaintain

In the same manner, the latter type of coolers are represented by attaching the following predicates.

**Goal:**G(Obj<sub>7</sub>.Temp < k)

**FuncType:**ToPrevent

## Discussion

In his work (de Kleer 1984), J. de Kleer proposes a method to use function of a system, which is derived from function of each component, to decrease ambiguous results of simulation. The method represents a component having more than one causal relation between input and output, where function of the component is decided by selecting one of the relations.

His work enables us to represent the function to achieve desirable state, which is also achieved by our representation method as discussed above. Furthermore, our method can explicitly represent the function which prevents system from falling into no good state and side effects which may damage the whole system.

V. Sembugamoorthy and B. Chandrasekaran (Sembugamoorthy & Chandrasekaran 1986) capture behavior as a series of states of the system and function as to achieve an intended desirable state. The function is achieved by the behavior and the function of each component whose role is defined by the function of the system. Such a framework to represent the function of large systems hierarchically is important.

In the above functional representation (Sembugamoorthy & Chandrasekaran 1986), the function is described as achieving a desirable state. Anne M. Keuneke (Keuneke 1991) classifies the concept of function into four types according to some conditions, for example, the method to achieve the function, the initial condition which raise the function, the length of the term in which effect of the function is hold, and so on. She captures function as attachment of implicit assumptions to a behavior, and employs a policy to seek for the primitives to represent the assumptions. Her policy is close to that of us in this sense. Our representation includes her classification and applies it to produce an abstract explanation of the function of a component and classification of the concepts of trouble.

Her classification of the function is employed in the work by B. Chandrasekaran, et al. (Chandrasekaran, Goel, & Iwasaki 1993) in which a functional representation framework is proposed by extending (Sembugamoorthy & Chandrasekaran 1986), which can be ap-

plied to various kinds of tasks especially design and diagnosis.

Because their representation of the function in (Chandrasekaran, Goel, & Iwasaki 1993) depends on the structure of the component of the system, description of the function of two systems differs from each other if their structure differs from each other. Thus their representation method seems to be weak in terms of re-usability of the description of function. On the other hand, our representation of the function of a component can be applied to wide range of components of the same input-output relations.

Y. Iwasaki, et al. (Vescovi *et al.* 1993) (Iwasaki *et al.* 1993) extended the framework for representing function proposed in (Sembugamoorthy & Chandrasekaran 1986) to represent the intention of designers, and proposed a framework for supporting refinement process of design through verification between behavior of a designed artifact and the intention of its designer. The function they captured is close to our concept. However, as the basic idea of functional representation is the same as (Sembugamoorthy & Chandrasekaran 1986), their method also seems to be weak in re-usability of the described components.

In order to decrease the cost for diagnosis, A. Abu-Hanna, et al. (Abu-Hanna, Benjamins, & Jansweijer 1991) propose a method to describe functional model of a system at three levels of abstraction.

We also regard abstraction and conceptualization of a component have close relations to the function of the component, since the goal of a component is always abstract one to enable interpretation of its behavior at the knowledge level. Thus we have investigated what primitives contribute to conceptualization of function. As mentioned in former sections, however, the enumeration of the primitives is not exhaustive.

M. Pegah, et al. (Pegah, Sticklen, & Bond 1993) apply the functional representation in (Sembugamoorthy & Chandrasekaran 1986) to F/A-18 aircraft fuel system, one of the large scale and complex systems, to achieve causal understanding of the system.

Our interest in application of this work goes to **KC III** (Kitamura *et al.* 1994), a model-based diagnostic shell which currently deals with a heat transportation system including negative feedbacks among its components whose explanation is difficult.

## Conclusion

This paper has proposed a new method and a vocabulary for representing components captured from the viewpoints of behavior and function. Our method does not rely on domain specific terms in representing components at a certain level of abstraction. The method enables model builders to describe a component at various levels of abstraction. Still we have several problems to be discussed about the concept of function as we discussed earlier. They are under discussion and its

result shall be reflected to refine the proposed representation method.

The potential of a functional representation method should be evaluated by not only the number of phenomena the method can describe but also the enhanced quality of a task by application of the model described by the method. Currently we aim at applying our method to the diagnostic shell **KC III** (Kitamura *et al.* 1994) in two ways. One is to support users in model building, and the other is to help users understand the process and result of a diagnosis.

W. Swartout, *et al.* (Swartout, Paris, & Moore 1991) propose a framework to build Explainable Expert Systems (EES). They regard three design aspects such as justifications of the system's action, explications of general problem-solving strategies, and descriptions of the system's terminology, as important things for producing good explanation in design activity. EES provides explanation from these aspects in a dialogue style. Another investigation by T.R. Gruber and P.O. Gautier (Gruber & Gautier 1993) shows a technique to explain the system's action and the knowledge which the system possesses.

Currently our interest goes to what information should be conveyed by the explanation for the users of **KC III**, and discussion about how can the representation method contribute to the explanation is on progress.

## References

- Abu-Hanna, A.; Benjamins, R.; and Jansweijer, W. 1991. Device understanding and modeling for diagnosis. *IEEE Expert* 26-32.
- Chandrasekaran, B.; Goel, A.; and Iwasaki, Y. 1993. Functional representation as design rationale. *COMPUTER* 48-56.
- de Kleer, J. 1984. How circuits work. *Artificial Intelligence* 24:205-280.
- Gruber, T. R., and Gautier, P. O. 1993. Machine-generated explanations of engineering models: a compositional modeling approach. In *Proceedings of the IJCAI*, 1502-1508.
- Hirai, T.; Nomura, Y.; Ikeda, M.; and Mizoguchi, R. 1991. The organization of the domain knowledge oriented to sharability - domain-specific tool based on the deep knowledge (**KC II-DST**). In *Proceedings of the 5th Annual Conference of JSAI*, 325-328. Japanese Society for Artificial Intelligence. in Japanese.
- Iwasaki, Y.; Fikes, R.; Vescovi, M.; and Chandrasekaran, B. 1993. How things are intended to work: Capturing functional knowledge in device design. In *Proceedings of the IJCAI*, 1516-1522.
- Keuneke, A. 1991. Device representation: the significance of functional knowledge. *IEEE Expert* 24:22-25.
- Kitamura, Y.; Sasajima, M.; Ikeda, M.; and Mizoguchi, R. 1994. Model building and qualitative reasoning for diagnostic shell. In *Proceedings of the '94 Japan/Korea Joint Conference on Expert Systems*, 41-46. Japanese Society for Artificial Intelligence.
- Pegah, M.; Sticklen, J.; and Bond, W. 1993. Functional representation and reasoning about the F/A-18 aircraft fuel system. *IEEE Expert* 24:65-71.
- Sembugamoorthy, V., and Chandrasekaran, B. 1986. Functional representation of devices and compilation of diagnostic problem-solving systems. In Kolodner, J., and Riesbeck, C., eds., *Experience, Memory, and Reasoning*. Hillsdale, N.J.: Lawrence Erlbaum Associates. 47-73.
- Swartout, W.; Paris, C.; and Moore, J. 1991. Design for explainable expert systems. *IEEE Expert* 58-64.
- Vescovi, M.; Iwasaki, Y.; Fikes, R.; and Chandrasekaran, B. 1993. CFRL: A language for specifying the causal functionality of engineered devices. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 626-633.

# A GENERIC HARNESS FOR THE SYSTEMATIC GENERATION OF MULTIPLE MODELS

*Q. Shen, R. R. Leitch, and A. D. Steele*

Department of Computing and Electrical Engineering  
Heriot-Watt University  
Edinburgh, Scotland

## ABSTRACT

Utilising multiple-model descriptions requires that the relationships between the various models be well-defined and can be generated systematically from a reference model. We present a generic model harness, for component-based models, that is based on a set of fundamental representational primitives that are directly related to a classification of basic model properties. This supports the customisation of the harness for a particular model and also the systematic generation of multiple models. Examples of the resulting models and their corresponding behaviours are presented for a laboratory-scale system rig.

## 1. Introduction

We are, at last, entering the meta-modelling stage in the development of problem solvers for engineering applications. More emphasis is beginning to be given to why we are adopting a given approach rather than how a particular approach is to be implemented. This implies a realisation that no one method, and hence a single model, is optimal for all potential applications. This viewpoint results in a methodological approach [7] to system specification in which the problem requirements are related to the characteristics of given solutions so that the selection of the 'best' approach for a given problem can be determined systematically. Further, there is a growing interest in problem solvers that utilise multiple models [2, 11, 12] to increase the generality and effectiveness of the application system. In which case the characteristics of the proposed solution need explicitly to be defined so that the relationship between the (multiple) models can be understood and hence the coherent use of these models be made.

In this paper, we propose a generalised model harness, based on the component-connection approach to modelling, such that various related models can be produced within the harness by varying basic model properties in a systematic way. We present the fundamental primitives of a generic

modelling language, the CBL [1], which is clearly seen as a generalisation of classical numerical simulation languages. We then define a set of primitive model properties and the operations that vary these properties. This is supported with comprehensive simulation results with reference to an experimental system-rig, clearly showing the effect of modifying the model properties and the utility of using a generic harness for developing multiple models of continuous dynamic systems.

## 2. A Generalised Approach to System Modelling

Many engineering applications require a model that explicitly represents the observable (or measurable) phenomena (variables) and the sub-systems or components that interconnect them. For instance, such component models are fundamental to many model-based diagnostic approaches in that the important variables are exactly those that determine the replaceable components and hence the level of isolation and/or identification required of the diagnostic algorithm. In which case, the modelling languages adopted should therefore be based on a component-based ontology [7], assuming that a physical system can be decomposed into a set of physically identifiable components whose combined behaviour constitutes the behaviour of the overall system. Within which, a component description is given in terms of the internal mechanisms of the component such that its stimulus-response behaviour can be simulated. Component descriptions have three basic requirements: 1) to represent various physical quantities (possibly time-dependent) that are, in principle, directly observable either by humans or mechanical sensors; 2) to represent the physical quantities that form the interconnections between at least two different components; and 3) to represent relationships (possibly time-varying and/or dynamic) between observable phenomena that influence or constrain the values of the physical quantities.

On the basis of these requirements the following primitive concepts are used in such modelling languages:

- **Structural Descriptions** describe how a component can be decomposed into parts (or sub-components) and how these parts are interconnected. Hierarchical structures are hence supported since a part itself may have a structural model.
- **Behavioural Descriptions** describe the relationships between the physical quantities related to a component which may be used to simulate the behaviour of a component. This may include different modes of the behaviour of a component, e.g., the normal and faulty behaviours in diagnosis. *Representational Primitives* are used to construct the behavioural descriptions:
  - *Variables* represent physical quantities which are in general time-dependent, but may be considered constant as part of the modelling assumptions (i.e., for systems in equilibrium).
  - *Domains* are the support sets from which variables take their values at a given time, e.g., the real numbers, the set of boolean values, or other quantity spaces [10].
  - *Terminal Variables* are the subset of the variables that can be common to other components, forming the interconnections between components.
  - *Parameters* are empirical (real-valued) coefficients between variables.
  - *Relations* represent the inter-dependencies between variables and can have various representational forms, including differential equations, algebraic constraints, and sets of if-then rules.

These primitives are based on the fundamental notion of signals and system (or equivalently data

and entity relationship) that governs all representational problems. Signals, here called variables, represent the object of the reasoning, i.e., the computation dependent part. Whereas the relationships describe the system, or subject, that determines how the computation will be done. Normally, this knowledge is assumed to be constant during computation. It can however be updated after computation as in machine learning or neural computing, for example.

## 2.1. CBL: the component based language

The potentially wide class of application areas for generalised modelling techniques require that the Component Based Language (CBL) employed within certain model-based reasoning tasks should be as general as possible. Adopting this point of view, we briefly present a description of such a language that defines a set of core concepts which is, we believe, common to all of the to the modelling of continuous dynamic systems and allows the general concepts described above to be extended by additional aspects. This allows the language to be adapted to a specified application, if necessary.

The CBL has been developed over the past ten years within a number of major European collaborative (ESPRIT) projects and used within different application tasks, including process control [7], intelligent training [5], and model-based diagnosis [4]. Unfortunately, due to the limitation on space, we cannot present the detailed syntax of the CBL in this paper. A basic outline of the syntactical structure of the language is given in figure 1 and further details can be found in [1]. Nevertheless, we hope that the way in which the CBL represents single and multiple models of physical systems will become clearer.

```
(SYSTEM      <system-name>
  {<component-instance>}
  (OBSERVABLES {<comp-name-variable-id>})
  (CONNECTIONS {<comp-name-variable-id . comp-name-variable-id>}))
```

(a) System model definition

```
(COMPONENT-CLASS <component-class-name>
  <variable-description>
  <domain-description>
  {(MODEL <model-name>
    {<behavioural-description>}))})
```

(b) Component class definition

```
(COMPONENT <component-name>
  (IS-A <component-class-name>)
  {(MODEL <model-name>
    (WITH {<parameter-name . value>}))})
```

(c) Component instance definition

Fig. 1. Outline of the CBL Syntax

## 2.2. The system rig: an illustrative example

To illustrate the representation of (multiple) models of continuous dynamic systems within the framework of the CBL we utilise a laboratory-scale system rig as a test-bed shown in figure 2.

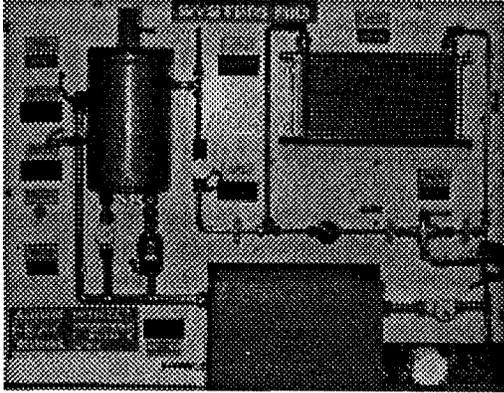


Fig. 2. The System Rig

This system is a typical representative of a wide-range class of industrial process systems and allows the behaviour of a heat exchange and extraction process to be examined experimentally. As reflected in the scanned image of this system, it consists of a number of physical components, or sub-systems, including a tank and a sump both of which store part of the fluid flowing around the system, a heater that heats the fluid in the tank, a radiator that dissipates

the thermal energy of the fluid passing through it, and a pump that drives the fluid around the rig.

To exhibit the various properties or behaviours of the different models to be developed we use a numerical model of the system as the reference model. The following is the third order numerical model of the system composed of both the flow and the thermal process loops under normal (correct) working conditions [8]:

$$h = \frac{q_i - q_o}{A}, \quad q_o = c_d \alpha \sqrt{2gh}, \quad v_2 = v_T - v_1,$$

$$\dot{T}_1 = \frac{Q + q_i e^c (T_3 - T_1)}{e^c v_1}, \quad \dot{T}_2 = \frac{q_o (T_1 - T_2)}{v_2},$$

$$T_3 = T_2 - \frac{u}{q_i e^c} (T_2 - T_a).$$

Within which, the meaning of the variables and parameters are listed in table 1. This model can be represented within the CBL as given in figure 3, where classes of components employed within the system are defined before the description of the system itself.

Variable	Variable Meaning	Parameter	Parameter Meaning
$q_i$	flow rate into tank	$A$	cross-section area of tank
$q_o$	flow rate out of tank	$\alpha$	cross-section area of tank
$h$	height of fluid in tank		output
$v_T$	total volume of fluid	$c_d$	discharge coefficient of tank
$v_1$	volume in tank		output
$v_2$	volume in sump	$e^c$	heat-density coefficient of
$Q$	total heat supply by heater		fluid in tank
$T_1$	temperature of fluid in tank	$u$	radiator heat transfer
$T_2$	temperature of fluid in sump		coefficient
$T_3$	temperature of fluid exiting radiator	$T_a$	ambient temperature

Table. 1. Explanation of Variables and Parameters

(COMPONENT-CLASS fluid-tank

(LOCAL-VARIABLES ( $f_{in}$ ,  $f_{out}$ ,  $f_h$ ,  $v$ ,  $T_{in}$ ,  $T_{out}$ ))

(RELATED-VARIABLES ( $v_{total}$ ,  $v_{other}$ ,  $Q_{total}$ ))

(DOMAIN {real} all-variables)

(MODEL correct

(BEHAVIOURAL-CONSTRAINTS

((= (deriv  $f_h$ ) (\ (-  $f_{in}$   $f_{out}$ )  $A$ )),

(=  $f_{out}$  (\*  $c_d$   $\alpha$  (sqr (\* 2  $g$   $f_h$ ))))),

(=  $v$  (-  $v_{total}$   $v_{other}$ )),

(= (deriv  $T_{cut}$ ) (\ (+  $Q_{total}$  (\*  $f_{in}$   $e^c$  (-  $T_{in}$   $T_{out}$ ))) (\*  $e^c$   $v$ ))))))

(COMPONENT-CLASS large-fluid-store

(LOCAL-VARIABLES ( $f_{in}$ ,  $f_{out}$ ,  $v$ ,  $T_{in}$ ,  $T_{out}$ ))

(RELATED-VARIABLES ( $v_{total}$ ,  $v_{other}$ ))

(DOMAIN {real} all-variables)

(MODEL correct

(BEHAVIOURAL-CONSTRAINTS

((=  $f_{out}$   $f_{in}$ ),

(=  $v$  (-  $v_{total}$   $v_{other}$ )),

(= (deriv  $T_{out}$ ) (\ (\*  $f_{in}$  (-  $T_{in}$   $T_{out}$ )  $v$ ))))))

(COMPONENT-CLASS heating-element

(LOCAL-VARIABLES  $T$ )

(DOMAIN {real} all-variables)

(MODEL correct

(BEHAVIOURAL-CONSTRAINTS ((=  $T$   $Q$ ))))

(COMPONENT-CLASS fluid-driver

(LOCAL-VARIABLES ( $f_{in}$ ,  $f_{out}$ ,  $T_{in}$ ,  $T_{out}$ ))

(DOMAIN {real} all-variables)

(MODEL correct

(BEHAVIOURAL-CONSTRAINTS ((=  $f_{out}$   $f_{in}$ ), (=  $T_{out}$   $T_{in}$ ))))

(COMPONENT-CLASS heat-radiator

(LOCAL-VARIABLES ( $T_{in}$ ,  $T_{out}$ ,  $f_{in}$ ,  $f_{out}$ ))

(DOMAIN {real} all-variables)

(MODEL correct

(BEHAVIOURAL-CONSTRAINTS

((=  $T_{out}$  (-  $T_{in}$  (\ (\*  $u$  (-  $T_{in}$   $T_a$ )) (\*  $f_{out}$   $e^c$ ))),

(=  $f_{out}$   $f_{in}$ ))))))

(SYSTEM system-rig

(COMPONENT tank

(IS-A fluid-tank)

(MODEL correct

(WITH (( $A$  .15.4  $\times 10^{-3}$ ), ( $c_d$  .0.6), ( $\alpha$  .0.17  $\times 10^{-3}$ ),

( $g$  .9.8), ( $e^c$  .4.18  $\times 10^6$ ))))

(COMPONENT sump  
 (IS-A large-fluid-store)  
 (MODEL correct)

(COMPONENT radiator  
 (IS-A heat-radiator)  
 (MODEL correct

(WITH ((u . 40), (T<sub>a</sub> . 20), (e<sup>c</sup> . 4.18 × 10<sup>6</sup>))))

(COMPONENT heater  
 (IS-A heating-element  
 (MODEL correct)))

(COMPONENT pump  
 (IS-A fluid-driver)  
 (MODEL correct))

(OBSERVABLES (heater\_Q<sub>total</sub>, tank\_f<sub>in</sub>, tank\_v, sump\_v<sub>total</sub>))

(CONNECTIONS ((tank\_f<sub>in</sub> . radiator\_f<sub>out</sub>), (tank\_f<sub>out</sub> . sump\_f<sub>in</sub>),  
 (sump\_f<sub>out</sub> . pump\_f<sub>in</sub>), (pump\_f<sub>out</sub> . radiator\_f<sub>in</sub>),  
 (heater\_Q<sub>total</sub> . tank\_T<sub>total</sub>), (tank\_v<sub>total</sub> . sump\_v<sub>total</sub>),  
 (tank\_v<sub>other</sub> . sump\_v), (sump\_v<sub>other</sub> . tank\_v),  
 (tank\_T<sub>in</sub> . radiator\_T<sub>out</sub>), (tank\_T<sub>out</sub> . sump\_T<sub>in</sub>),  
 (sump\_T<sub>out</sub> . pump\_T<sub>in</sub>), (pump\_T<sub>out</sub> . radiator\_T<sub>in</sub>))))

Fig. 3. Numerical Model of System-Rig in CBL

Based on this model the dynamic evolution of the system can be simulated using traditional numerical integration techniques in common with the CBL description. To ease the comparison later, we herein concentrate on the exhibition of the characteristics of the two essential processes running throughout the system-rig. Figure 4 presents the (numerical) simulation plot of the flow loop and that of the

temperature loop, described by the behaviour of the fluid height in the tank and the temperature of the fluid in the tank. Also shown in this figure are, (again, for the purpose of comparison to be made later) the simulation results under an assumed faulty condition where the output orifice of the tank is partially blocked.

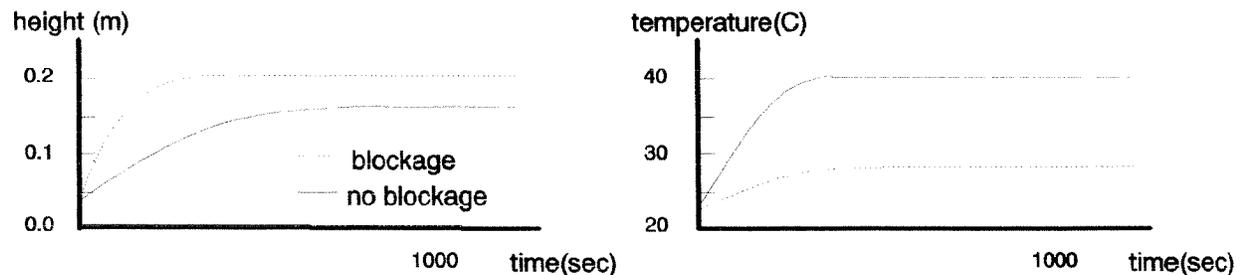


Fig. 4. Numerically Simulated Behaviours

### 3. Model Properties and Modelling Dimensions

As indicated in the introduction, we are interested in using the CBL as a harness from which different related models can be developed in a coherent and systematic manner by varying the fundamental

model properties. The first model property in the modelling process is the choice of representational *ontology* that governs knowledge representation in general and knowledge source, knowledge level, and knowledge orientation, in particular. Within this paper, we adopt the component-based ontology to

system modelling, as we are primarily interested in model-based diagnosis. Another important choice for modelling is the *scope* of the system model. It defines the physical boundary of the part of the system that is being modelled. For instance, in system engineering this property specifically determines which variables are treated as exogenous or endogenous. Having chosen the ontology and scope for the model the description of the behaviour of the system or, equivalently, the solution of a model exhibits four basic representational properties that we term *resolution*, *precision*, *accuracy*, and *uncertainty*. Resolution is a simple, but essential characteristic of system models, which denotes the number of variables used to describe the physical phenomena concerned. Precision reflects the number of distinctions supported by the description of the behaviour and the underlying semantics of such distinctions, i.e., the quantity space. Accuracy determines the closeness of the behaviour generated to that of a reference model and is clearly an important, but sometimes non-essential property for a particular task. Uncertainty describes the confidence attached to a given state or behaviour and can be used to represent the essentially subjective knowledge common in modelling real application systems.

As an example to illustrate the distinction of precision, accuracy, and uncertainty consider the case of modelling the trivial case of a single measurement, say of the temperature of the fluid in the tank of the system rig, whose true value is  $100\text{ }^{\circ}\text{C}$ . Figure 5 then shows different models of this value in terms of varying these properties. Figure 5(a) shows the 'true' value, whereas figure 5(b) shows a precise, real-valued, but inaccurate model. The accuracy of the model can be restored by reducing the precision! Figure 5(c) shows a (crisp) interval based model which describes the value as lying between  $95\text{--}105\text{ }^{\circ}\text{C}$ , which of course is correct, but less precise. In figure 5(d) a model of the same precision is used but it is now inaccurate through the process of approximation. All of the above models have assumed absolute commitment to the representation: either the true value lies inside the description or outside of it. Of course real world knowledge is not as certain as this. In which case fuzzy sets may be used to represent the inherent uncertainty. Figure 5(e) shows an uncertain, precise and accurate model, using fuzzy numbers whereas figure 5(f) shows an uncertain, less precise but still accurate model.

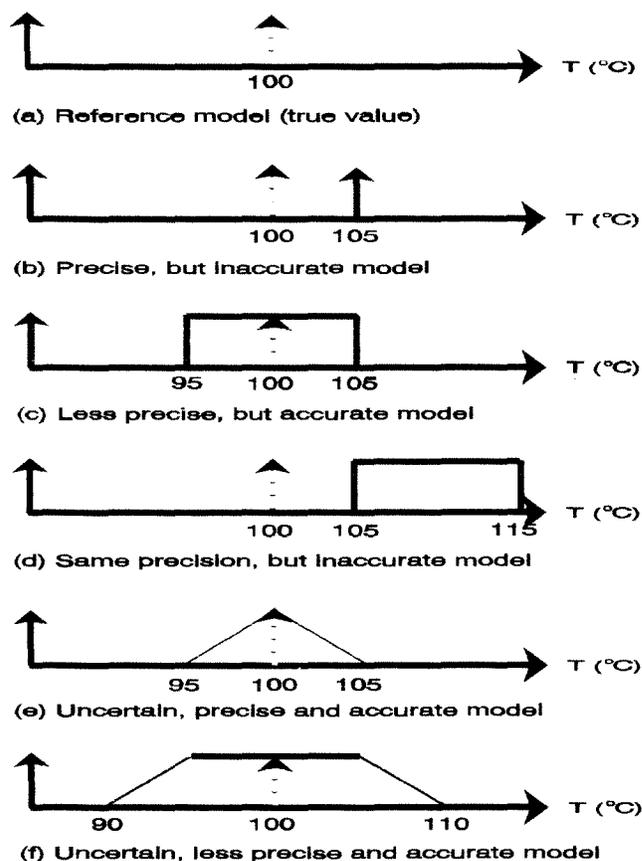


Fig. 5. Example of Model Precision, Accuracy and Uncertainty

We hope that the above discussion shows that the precision, accuracy and uncertainty are indeed distinct properties of models of system behaviour, while the scope and resolution are apparently another two basic model characteristics. Varying one of them results in a different behavioural description of the system and hence a different model is established. Following this theme, we present our view on five fundamental modelling 'dimensions' upon which to develop multiple models of a continuous dynamic system based on a common reference representation in the CBL. Application of these dimensions or model operations results in a set of related models, based on the reference model, that supports a general characterisation of multiple modelling techniques in a clear perspective. In the following, unless otherwise stated, the simulation results are obtained via the use of our Fuzzy Qualitative Simulation algorithm [9]. That is, the numerical models presented within the paper are all transformed into their corresponding (fuzzy) qualitative models. Importantly, the simulations are carried out in conjunction with a behavioural prioritiser [6] that allows the selection of the most likely behaviour out of a number of possible behaviours generated by the simulation algorithm. Also, within the results to be presented, all the lines between the qualitative states (denoted by solid circles) are given for illustrative purpose only.

```
(SYSTEM    focused-heating-tank

(COMPONENT tank
(IS-A fluid-tank)
(MODEL correct

(WITH ((A .15.4 × 10-3), (cd .0.6), (α .0.17 × 10-3),
(g .9.8), (ec .4.18 × 106))))

(COMPONENT heater
(IS-A heating-element
(MODEL correct)))

(OBSERVABLES ((heater _Qtotal, tank _fin, tank _fout),
(tank _Tin, tank _Tout, tank _v)))

(CONNECTIONS ((heater _Qtotal . tank _Ttotal))))
```

Fig. 6. Focused Model

In order to reveal the actual behaviour under an abnormal condition modifications to this model should be made. For instance, given knowledge of a partial blockage in the output orifice of the tank an adjustment of the orifice discharge coefficient  $c_d$  to a new value is made. Both the normal and abnormal

### 3.1. Focusing models

It has been pointed out that a fundamental property of a model is its scope, which denotes the part of the physical world represented by that model. Changing a model's scope by redefining this boundary between model and environment is termed the *Focus* operation. This operation is very useful in model-based applications in general and in model-based diagnosis in particular. In which case, a model reflecting a particular focus of attention may be used to first isolate a fault to a sub-system within a plant before further focusing to a suspected faulty area or an individual component in that sub-system via focus of suspicion procedures [4, 11].

Suppose that within a model-based diagnostic process, part of the system rig consisting of the tank with the embedded heater needs to be further examined for, say, fault identification. This part of the system can then be modelled with a focused model and represented by the following CBL description of the focused 'system', with detailed definition of the components unchanged. In this case, however, the flow of the fluid in and out of the tank and the corresponding temperatures are treated as the input and output of the focused system and, therefore, assumed to be observables:

conditions are simulated and the results are provided in figure 7. Compared with those shown in figure 4, clearly, qualitative simulation outcome displays the same tendency as that in numerical simulation.

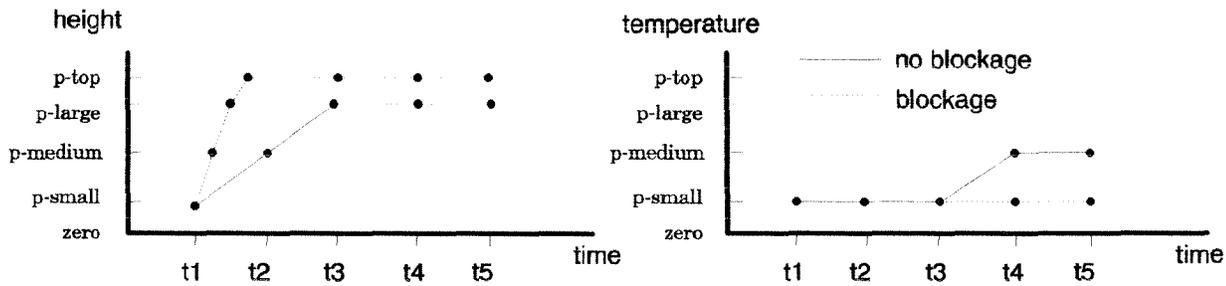


Fig. 7. Simulation Plot of Focused Models

### 3.2. Simplification of models

Different models of a unique system (within a given scope) can be obtained from the reference model by neglecting some of the internal variables, thereby affecting the *resolution* of the model. Such an operation on models we term Simplification. For example, the dynamics, or speed of response of certain variables can be assumed to be instantaneous (though known to take some finite time) with respect to the response of other variables and hence replaced by their steady state value [3]. This results in a more granular (lower resolution) model. In which case the variable may be eliminated from the simplified model.

For the system rig, when only information about the dynamics of the flow process is concerned the thermal variables can be neglected as indeed they are much slower than the dynamics of the flow loop. This leads to a simpler model with lower resolution. In terms of the CBL representation, the resulting model is presented in figure 8. Similarly, if considering the thermal process only, a simplified second-order model without flow variables can also be developed. As demonstrated in figure 9, the simplified models representing either the flow or the thermal loop only produce, again, a similar description of the evolution of the dynamics within each process to that obtained from the corresponding numerical models.

```
(COMPONENT-CLASS fluid-tank
  (LOCAL-VARIABLES ( $f_{in}$ ,  $f_{out}$ ,  $f_h$ ,  $v$ ))
  (RELATED-VARIABLES ( $v_{total}$ ,  $v_{other}$ ))
  (DOMAIN {real} all-variables)
  (MODEL correct
    (BEHAVIOURAL-CONSTRAINTS
      ((= (deriv  $f_h$ ) (\ -  $f_{in}$   $f_{out}$  ) A)),
      (=  $f_{out}$  (*  $c_d$   $\alpha$  (sqr (* 2  $g$   $f_h$ ))))),
      (=  $v$  (-  $v_{total}$   $v_{other}$ ))))))
```

Fig. 8. Simplified Model

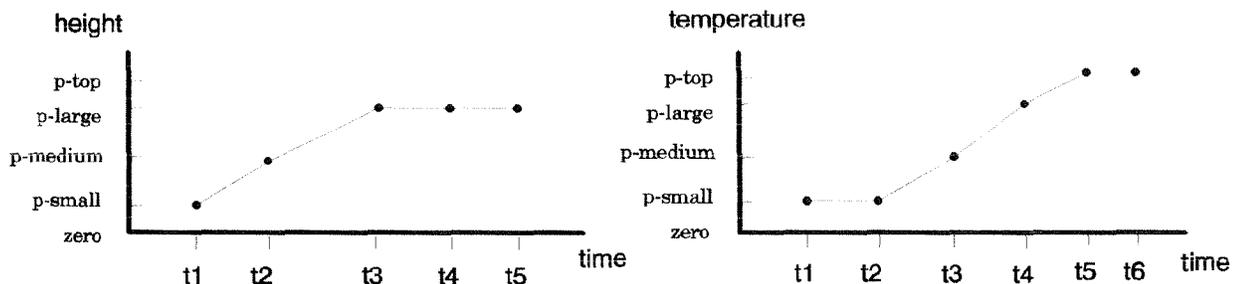


Fig. 9. Simulation Plot of Simplified Models

### 3.3. Abstraction of models

A very basic operation on models is *Abstraction* that modifies the *precision* of the underlying knowledge representation of the model in order to make less precise descriptions of the behaviour of the system. As such, this modelling dimension has been by far the most extensively studied within the Qualitative Reasoning community. An important and defining characteristic of *Abstraction* is that the resulting model is a 'faithful' transformation in that it will produce a behaviour that is consistent with an *Abstraction* operation applied to the behaviour of the reference model [12]. In other words, an *Abstraction* is a less precise but still correct description.

Various models of the system-rig with different levels of abstraction can be obtained in the CBL representation by varying the domain definition of each class of components such that, when using the

```
(COMPONENT-CLASS heating-element
 (LOCAL-VARIABLES T)
 (DOMAIN {+, -, 0} all-variables)
 (MODEL correct
 (BEHAVIOURAL-CONSTRAINTS ((= T Q))))),
```

or,

```
(COMPONENT-CLASS heating-element
 (LOCAL-VARIABLES T)
 (DOMAIN {n-top, n-large, n-medium, n-small, zero,
 p-small, p-medium, p-large, p-top} all-variables)
 (MODEL correct
 (BEHAVIOURAL-CONSTRAINTS ((= T Q))))),
```

Fig. 10. Abstracted Models

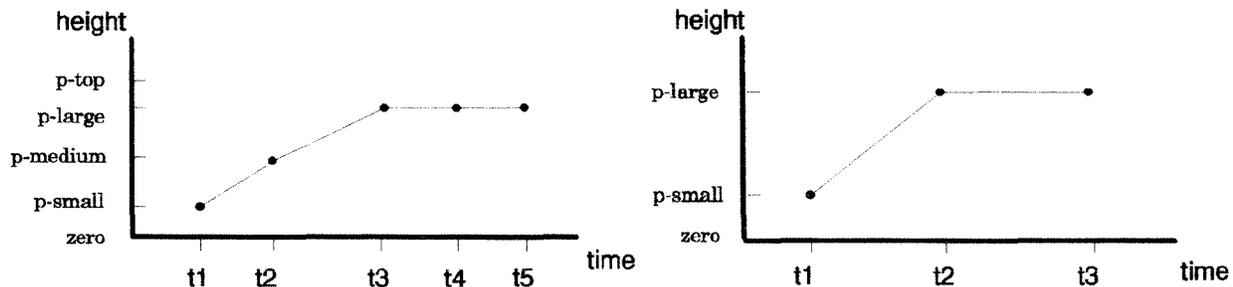


Fig. 11. Simulation Plot of Abstracted Models

### 3.4. Modifying commitment to models

It is clear that uncertainty can occur in two main ways within a model. The first is in the particular value to ascribe to a given measurement or observation. In particular, if there is a random element associated with the measurement, probability can be used to estimate the most likely

traditional three-sign space  $\{+, -, 0\}$  or a space of fuzzy qualitative values  $\{n\text{-top}, n\text{-large}, n\text{-medium}, n\text{-small}, \text{zero}, p\text{-small}, p\text{-medium}, p\text{-large}, p\text{-top}\}$ , the original domain  $\{\text{real}\}$  in the reference model is substituted by one of them. In so doing, as an example, the component class definition of heating-elements becomes one of the two given in figure 10.

We have carried out a number of simulations using various (fuzzy) quantity spaces. In particular, figure 11 presents the behaviour generated by the utilisation of the denser quantity space given above and that produced using a quantity space consisting of (fuzzy) qualitative values that collapses the definition of the underlying semantics of *p-small* and *p-medium* into *p-small*, and *p-large* and *p-top* into *p-large*.

'next' value based on historical information. However, if the description of the measurement is inherently vague then measures based on belief or fuzzy sets can be used to capture such uncertainty. The second way that uncertainty can occur, in physical system modelling, is in the relationships between the variables, i.e. in describing the physical operations themselves. In stochastic uncertainty,

Bayesian theory, or variants thereof, can be used to produce estimates of 'output' based on uncertain 'inputs' and uncertain 'operations'. Similarly, in 'fuzzy' situations possibility theory can be used to represent uncertain implications. Although uncertainty plays an important role in AI, the dimension of commitment is the least explored in Qualitative Reasoning. Most existing methods assume crisp, although abstract and possibly inaccurate models. The ability to refine uncertain measures within application systems would have important benefits for qualitative modelling applications.

As an illustration let us examine in a bit more detail the first situation that uncertainty may appear with the system-rig. For simplicity, we concentrate on the height of the fluid in the tank. Suppose that the value range of this variable in the reference model falls within  $[0, 25]$  (cm), a qualitatively precise model with full certainty may then be built upon a quantity space such as  $\{0, (0, 10), 10, (10, 25), 25\}$ . This is, of course, the same as the result of using abstraction. However, if knowledge of the important 'landmarks' (i.e., 10 or 25) is known vaguely an uncertain

quantity space like  $\{zero, p-small, p-medium, p-large, p-top\}$  should then be adopted in order to avoid potentially important difficulties in the interpretation between behaviour derived from a model and the physical observations [10]. This results in a different model from which the crisp qualitative quantity space is used. In terms of CBL, the representation of an uncertain model is similar to the one used to describe a crisp one, as illustrated earlier in figure 10. To visualise the implication that different degrees of commitment to a model has in the generation of system behaviour, we show the results attained by using two related quantity spaces that have the same precision but different distribution of uncertainty over individual qualitative values. As illustrated by the resulting behaviours, the modification of commitment degree leads to the change of the absolute time indices that indicate when a particular qualitative state appears. More importantly, perhaps, such modification results in a reduction of qualitative ambiguity. Indeed, although it is not reflected within the simulation plot (since we only present the most likely priority behaviour), the more committed model, with less uncertainty, generates less possible behaviours.

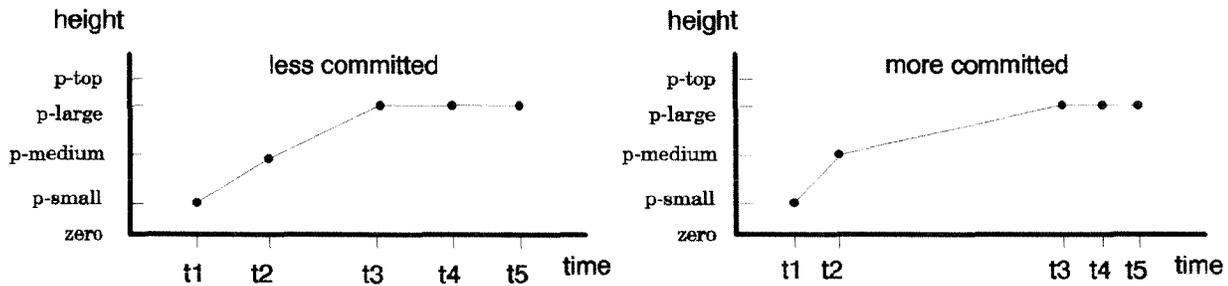


Fig. 12. Simulation Plot of Models with Different Degree of Uncertainty

### 3.5. Approximation of models

The last but not least operation of physical models is called *Approximation*. This modelling dimension corresponds to the model modifications where a known functional relationship between internal variables of the model is replaced by a simpler but less accurate function. This reduction in information results in the approximate model not necessarily maintaining the correctness of the model.

Importantly, *Approximations* need not be restricted to real-valued quantity spaces. For instance, functional relationships are represented in Fuzzy Qualitative Simulation [9] by fuzzy relations. This

allows for more or less accurate descriptions through modifying the relational matrix in much the same way as linear approximations to polynomial relationships on the real-number line. In particular for the system-rig with a given quantity space, the quadratic function between the fluid height in the tank and the flow rate out of the tank may be represented by a set of if-then rules and, further, be interpreted by a look-up table through the fuzzy compositional rule of inference as given in figure 13. As such, they are, necessarily restricted to the operating range experienced during the operation or experimentation with the process, and therefore are fundamentally approximations.

(COMPONENT-CLASS fluid-tank

(LOCAL-VARIABLES ( $f_{in}$ ,  $f_{out}$ ,  $f_h$ ,  $v$ ,  $T_{in}$ ,  $T_{out}$ ))

(RELATED-VARIABLES ( $v_{total}$ ,  $v_{other}$ ,  $Q_{total}$ ))

$$\begin{aligned}
& (\text{DOMAIN } \{n\text{-top}, n\text{-large}, n\text{-medium}, n\text{-small}, \text{zero}, \\
& \quad p\text{-small}, p\text{-medium}, p\text{-large}, p\text{-top}\} \text{ all-variables}) \\
& (\text{MODEL correct} \\
& \quad (\text{BEHAVIOURAL-CONSTRAINTS} \\
& \quad \quad ( (= (\text{deriv } f_h) (\setminus (- f_{in} f_{out}) A)), \\
& \quad \quad \quad (= (\text{deriv } T_{out}) (\setminus (+ Q_{total} (* f_{in} e^c (- T_{in} T_{out}))) (* e^c v))), \\
& \quad \quad \quad (= v (- v_{total} v_{other}))), \\
& \quad \quad ((f_{out} . f_h) (n\text{-top} . n\text{-top}), (n\text{-large} . n\text{-large}), \\
& \quad \quad \quad (n\text{-medium} . n\text{-medium}), (n\text{-small} . n\text{-small}), \\
& \quad \quad \quad (\text{zero} . \text{zero}), (p\text{-small} . p\text{-small}), \\
& \quad \quad \quad (p\text{-medium} . p\text{-medium}), (p\text{-large} . p\text{-large}), \\
& \quad \quad \quad (p\text{-top} . p\text{-top}))))))
\end{aligned}$$

Fig. 13. Approximated Model

The behaviour of the system-rig, with less accuracy than the reference model, can be generated using such an approximated model. Actually, the resulting behaviour has already been presented in sections 3.2, 3.3, and 3.4 for comparison purposes. In addition, we now present the simulated behaviour from a

further approximated model that represents a *medium* blockage at the output orifice of the tank. As illustrated in figure 14, such a model leads to the behaviour with highest priority representing an overflow of the fluid in the tank.

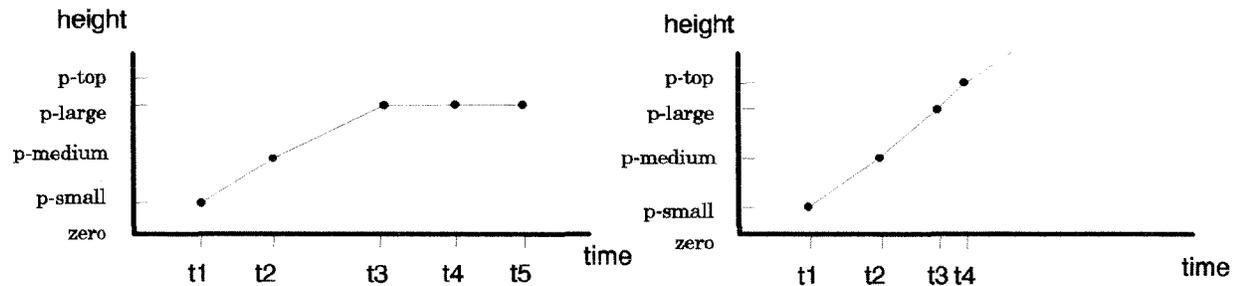


Fig. 14. Simulation Plot of Approximated Models

#### 4. Conclusion

Although many approaches to system modelling have been developed within the last decade, in general, there does not exist a consensus on the fundamental model properties. The employment of a particular modelling technique and the use of (multiple) models requires coherent definitions of such properties and a clarification of potential modification dimensions upon which different models can be built via varying certain model characteristics. We have proposed a set of five important properties of system models and their

corresponding operations and associated modelling primitives are summarised in table 2.

We have shown how these properties are related to representational primitives and that, by adopting a generic modelling harness based on these primitives, various models can be developed systematically. We believe, the resulting operations on models have a clear meaning. This allows informed decision to be made about choosing the appropriate modelling method and hence correct model(s) for a given class of problem and, also, a clearer exposition of the existing application systems that are based on the utilisation of multiple models.

Property	Scope	Resolution	Precision	Uncertainty	Accuracy
Operation	Focusing	Simplification	Abstraction	Commitment	Approximation
Primitive	Connections	Variables	Domain	Domain	Relations

Table 2. Model Properties, Operations, and Primitives

### Acknowledgements

The authors would like to acknowledge the support of our colleagues, Mike Chantler and George Coghill of the Intelligent Systems Engineering Laboratory at Heriot-Watt University and to thank them for many interesting and challenging discussions. We are also grateful to the partners in the QUIC, ITSIE, and ARTIST projects (funded by the CEC under the ESPRIT programme) for their contribution whilst taking full responsibility for the views expressed in this paper. Thanks are also extended to Chai Quek for providing the numerical simulations of section 2.2.

### References

- [1] P. Alvari, *et al*, Specification of ARTIST/I, *Deliverable Report of ESPRIT Project (P5143)*, 1991.
- [2] B. Falkenhainer and K. Forbus, Compositional modeling: finding the right model for the job, *Artificial Intelligence*, **51**, 95-144, 1991.
- [3] B. Kuipers, Abstraction by time-scale in qualitative simulation, *Proceedings of the Sixth National Conference on Artificial Intelligence*, 621-625, 1987.
- [4] R. Leitch, H. Freitag, Q. Shen, P. Struss, and G. Tornielli, ARTIST: a methodological approach to specifying model based diagnostic systems, in G. Guida and A. Stefanini (Eds), *Industrial Applications of Knowledge-Based Diagnosis*, 289-310, 1992.
- [5] R. Leitch, P. Ponnappalli, and, A. Slater, The representation of domain knowledge in intelligent training systems, *Proceedings of the First International Conference on Intelligent Systems Engineering*, 269-274, 1993.
- [6] R. Leitch and Q. Shen, Prioritising behaviours in qualitative simulation, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1523-1528, 1993.
- [7] R. Leitch and A. Stefanini, Task dependent tools for intelligent automation, *Artificial Intelligence in Engineering*, **4**, 126-143, 1989.
- [8] C. Quek, *The Application of Artificial Intelligence Techniques to the Integrated Control of Complex Dynamic Physical Systems*, Ph.D. Thesis, Heriot-Watt University, 1990.
- [9] Q. Shen and R. Leitch, Fuzzy qualitative simulation, *IEEE Transactions on Systems, Man, and Cybernetics*, **23**, 1038-1061, 1993.
- [10] Q. Shen and R. Leitch, On extending the quantity space in qualitative reasoning, *Artificial Intelligence in Engineering*, **7**, 167-173, 1992.
- [11] P. Struss, What's in SD? towards a theory of modeling for diagnosis, in: W. Hamscher, J. deKleer, and L. Console (Eds), *Readings in Model-Based Diagnosis*, 419-450, 1992.
- [12] D. Weld, Reasoning about model accuracy, *Artificial Intelligence*, **56**, 255-300, 1992.

# Model Abstraction for Testing of Physical Systems

Peter Struss

Technical University of Munich  
Computer Science Dept.  
Orleansstr. 34  
D-81667 Munich  
Germany  
struss@informatik.tu-muenchen.de

## Abstract

We present a formal theory of model-based testing and an algorithm for test generation based on it, and outline how testing is implemented by a diagnostic engine. The key to making the complex task of test generation feasible for systems with continuous domains is the use of model abstraction. Tests can be generated using manageable finite models and then mapped back to a detailed level. We state conditions for the correctness of this approach and discuss the preconditions and scope of applicability of the theory.

## 1 Introduction

Testing means shifting a system into different states by appropriate inputs in order to find observations that determine its present behavior mode. Often, the tests are designed to *confirm a particular behavior*, usually the correct or intended one, for instance in manufacturing. In diagnosis we may, in contrast, want discriminating tests which effectively and efficiently *identify the present (faulty) behavior*. This paper focuses on confirming tests. There exist theories and algorithms for test generation in particular domains. For digital circuits, for instance, a solution is feasible because, although the number of components can be large, the individual components exhibit a simple behavior and, more fundamentally, because of the Boolean domain of the variables ((Roth 1980), (Gupta & Welham 1989), (Camurati et al. 1990)). For variables with large domains or for physical systems with continuous behavior, these techniques are not applicable. In extending methods from model-based diagnosis, and exploiting our work on multiple modeling (Struss 1992), we propose a general theory that addresses the generation and application of tests in such domains.

We first discuss the problems addressed and outline the basic ideas of our approach by presenting a simple (continuous and dynamic) system, a thyristor. In section 3, we present the basic theory and an algorithm for test generation. Testing of constituents in the context of a whole device is shown to be a straightforward extension in section 4. Section 5 outlines briefly how testing is implemented by a standard model-based diagnosis engine. Finally, we discuss the achievements, preconditions, and restrictions of the approach.

Due to space limitations, we do not always treat the most general cases, and we omit proofs. Both can be found in the long version of this paper (Struss 1994).

## 2 The Intuition behind Testing

In the following, we consider a continuous dynamic system as an illustrative example (rather than a serious application). A thyristor is a semi-conductor with anode, A, cathode, C, and gate, G, that operates as a (directed) switch: it works in two states, either conducting current in a specified direction with almost zero resistance (exaggerated by the upper line of the simplified characteristic curve in Fig. 2.1a), or blocking current like a resistor with almost infinite resistance (the horizontal line). The transition from the OFF state to ON is controlled by the gate; if it receives a pulse the thyristor "fires", provided the voltage drop exceeds a threshold,  $V_{Th}$ . There is a second way to fire a thyristor (which is normally avoided, but may occur in certain circuits and situations), namely if the voltage drop exceeds the breakover voltage,  $V_{Bo}$  as is indicated by the characteristic in Fig. 2.1a. The annotation with 1 and 0 indicates the presence and absence of a gate pulse. So, for instance, for  $\Delta V > V_{Bo}$  the thyristor is ON ( $i > 0$ ), no matter whether or not it receives a gate pulse and, hence, the annotation with 0 and 1. In contrast, the section  $V_{Th} < \Delta V < V_{Bo}$ ,  $i > 0$  is annotated with 1, because a gate pulse is required for firing. This representation is based on the assumption that switching happens instantaneously (turn-On time and spreading time 0).

Now suppose we want to test a thyristor, i.e. to make sure that it behaves according to the described correct behavior. This creates several problems: voltage and current are considered to have a continuous domain. We can only gather a finite set of sample observations. But if they all agree with the desired behavior, what would then make us confident that more observations could not reveal a contradiction to this behavior? It is the fact that *there is no other possible behavior (a faulty one) that would also be consistent with the previous observations*.

What are the possible faults of a thyristor? A thyristor may be *punctured*, i.e. acting like a wire, or *blocking* like an open switch. A third kind of fault may be due to the fact that the actual *breakover voltage is less than the*

nominal one, with the result that the thyristor fires at a voltage drop well below  $V_{Bo}$  without a gate pulse. With  $V'_{Bo}$  we denote the lowest tolerable actual breakover voltage (or the highest one which is considered to characterize a faulty behavior). Fig. 2.1 shows the (idealized) characteristics of these behaviors in comparison to the correct behavior.

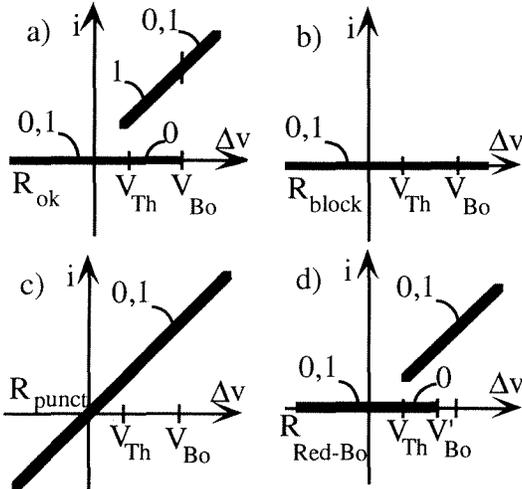


Figure 2.1 The characteristics of the behaviors of a thyristor: a) correct b) blocking c) punctured d) with a reduced breakover voltage

Considering these behaviors (and, perhaps, looking at the figures), we may get the following idea for a set of two tests: the first one with a high voltage drop (i.e. between  $V'_{Bo}$  and  $V_{Bo}$ ) without a gate pulse, and a second one with a medium or high voltage drop (i.e. between  $V_{Th}$  and  $V_{Bo}$ ) in conjunction with a gate pulse. If we obtain results that comply with the correct behavior in both cases (zero current for the former, positive current for the latter), then the thyristor must be correct, because these observations rule out all three types of faults: the first one contradicts the punctured behavior and a reduced breakover voltage, while the second one refutes the blocking mode. This simple example illustrates several fundamental ideas :

- A particular behavior is confirmed if all others can be refuted by some finite set of observations.
- We obtain such sets of tests by describing behaviors through relations among variables and by determining their distinctions (i.e. set differences).
- We may end up with less tests than the number of behaviors to be refuted (in the thyristor example two tests for an infinite number of behaviors).

Finally, the thyristor indicates a way to address the complexity problem when we have to handle large or even infinite domains:

- We may be able to perform test generation using a (qualitative) abstraction of the behavior description (e.g. with characterizations such as "high" and "medium").

In the remainder of this paper we develop these ideas into a formal theory and an algorithmic solution for test generation and testing.

### 3 Test Generation for Single Constituents

First, we present the basic definitions and results that allow the generation of tests, based on relational behavior models. For all definitions and theorems, we first paraphrase them in English before presenting the formal statement. Throughout this section, we consider one constituent (component, mechanism, process, subsystem that is) of a system that is assumed to be accessible. It has a (not necessarily finite) set of possible, mutually exclusive behaviors, BEHVS, associated with it. This set is assumed to be *exhaustive*, i.e. the constituent has exactly one behavior  $B_i \in \text{BEHVS}$ . Later, we will discuss the case of this assumption being wrong. That the constituent has one unique behavior seems to exclude the possibility of intermittent behaviors. In (Struss 94a) we show that this is not the case.

#### 3.1 The Foundation: Finding Observable Distinctions

As motivated by the example (and common in model-based reasoning systems which use constraints for modeling), we describe behavior modes by the set of value tuples that are possible under this behavior, i.e. by a relation  $R$  in some representation. Using the formalism of (Struss 1992) such a representation is determined by selecting a vector

$$\underline{v} = (v_1, \dots, v_2)$$

of local variables and their respective domains:

$$\text{DOM}(\underline{v}) = \text{DOM}(v_1) \times \text{DOM}(v_2) \times \dots \times \text{DOM}(v_k).$$

For the time being, we assume one fixed representation  $(\underline{v}, \text{DOM}(\underline{v}))$ , because this simplifies the notation and is not an essential restriction (the general case is treated in (Struss 1994)). The behavior models of the thyristor can be described in the representation

$$(\underline{v}_{Th}, \text{DOM}(\underline{v}_{Th})) = ((\Delta V, \text{gate}, i), \mathbb{R} \times \{0, 1\} \times \mathbb{R}).$$

The relations  $R \subset \text{DOM}(\underline{v}_{Th})$  from Fig. 2.1 modeling the thyristor behaviors are shown in Table 3.1. The inevitable inaccuracy in this model is reflected by the (small) numbers  $\delta$  and  $\Delta$ . By SIT we denote the set of situations physically possible under the present mode of a constituent. We define a behavior model  $M(R)$  as the claim that the relation  $R$  covers all value tuples  $\underline{v}$  may take in a situation  $s \in \text{SIT}$ :

#### Definition 3.1 (Behavior Model)

$$M(R) : \Leftrightarrow \forall \underline{v}_0 \in \text{DOM}(\underline{v}) (\exists s \in \text{SIT } \underline{v}(s) = \underline{v}_0) \Rightarrow \underline{v}_0 \in R \quad ^1$$

<sup>1</sup>  $\underline{v}(s) = \underline{v}_0$  means that  $\underline{v}$  has the value  $\underline{v}_0$  in situation  $s$  rather than equality. Because  $\underline{v}$  can take different values

	$\Delta V$	gate i	
<b>R<sub>ok</sub></b>	$(-\infty, 0]$	$\{0,1\}$	$[-\delta, 0]$
	$(0, V_{Th}]$	$\{0,1\}$	$[0, \delta]$
	$(V_{Th}, V_{Bo}]$	$\{0\}$	$[0, \delta]$
	$(V_{Th}, V_{Bo}]$	$\{1\}$	$[(res-\Delta)*\Delta V, (res+\Delta)*\Delta V]$
	$(V_{Bo}, \infty)$	$\{0,1\}$	$[(res-\Delta)*\Delta V, (res+\Delta)*\Delta V]$
<b>R<sub>block</sub></b>	$(-\infty, 0]$	$\{0,1\}$	$[-\delta, 0]$
	$(0, \infty)$	$\{0,1\}$	$[0, \delta]$
<b>R<sub>punct</sub></b>	$(-\infty, \infty)$	$\{0,1\}$	$[(res-\Delta)*\Delta V, (res+\Delta)*\Delta V]$
<b>R<sub>Red-Bo</sub></b>	$(-\infty, 0]$	$\{0,1\}$	$[-\delta, 0]$
	$(0, V_{Th}]$	$\{0,1\}$	$[0, \delta]$
	$(V_{Th}, V_{Bo}]$	$\{0\}$	$[0, \delta]$
	$(V_{Th}, \infty)$	$\{0,1\}$	$[(res-\Delta)*\Delta V, (res+\Delta)*\Delta V]$

**Table 3.1** Relations modeling thyristor behaviors. Each is the union of the lines of the table; e.g. the first line of  $R_{ok}$  is to be read  $(-\infty, 0] \times [-\delta, 0] \times \{0,1\}$ .

If  $M(R_i)$  is a model of the behavior  $B_i \in BEHVS$ , i.e.

$$B_i \Rightarrow M(R_i),$$

and if an observation (obs) contradicts the behavior model, i.e. lies outside  $R_i$ , then we can safely rule out the behavior:

$$obs \Rightarrow \neg M(R_i) \mid\text{---} obs \Rightarrow \neg B_i.$$

While this provides a way for *refuting* behaviors, we are interested in *confirming* a particular behavior.

As suggested by the example, tests are defined as sets of value tuples such that observing at least one tuple in each set in reality allows us to conclude the presence of a behavior mode. More formally: a set of value tuples  $V = \{\underline{v}_i\}$  containing at least one tuple out of each  $T_i$ ,

$$\forall T_i \exists \underline{v}_i \in V \quad \underline{v}_i \in T_i,$$

is called a *hitting set* of  $\{T_i\}$ . The fact that all the values in  $V$  are actually taken in some real situation is denoted by the sentence  $\phi_v$ :

$$\phi_v \equiv \forall \underline{v}_i \in V \exists s_i \in SIT \quad \underline{v}(s_i) = \underline{v}_i.$$

### Definition 3.2 (Test, Confirming Test Set)

A test is a non-empty relation on some representational space:  $T_i \subseteq DOM(\underline{v})$ .

A set  $\{T_i\}$  of tests is a confirming test set for a behavior  $B_0 \in BEHVS$  iff for all hitting sets  $V$  of  $\{T_i\}$ , observation of  $V$  entails  $B_0$ :

$$\phi_v \mid\text{---} B_0.$$

What assured us that the tests in section 2 actually confirm the thyristor's correct behavior? The fact that no other behavior mode would survive observations from

(from different domains, but also in the same domain), (Struss 1992) uses a special predicate  $Val$ .

both tests. In general, for each behavior  $B_j$ , different from the one to be confirmed, there must exist a test  $T_j$  lying completely outside a modeling relation of  $B_j$ . In other words, the complement of  $T_j$ ,

$$T_j^c := DOM(\underline{v}) \setminus T_j,$$

specifies a model of  $B_j$ . This is stated by Lemma 3.1.

### Lemma 3.1

$\{T_i\}$  is a confirming test set for  $B_0$  iff

$$\forall B_j \in BEHVS \quad B_j \neq B_0 \Rightarrow (\exists T_i \quad B_j \Rightarrow M(T_i^c)).$$

A test is only useful if it is observable. So, in the following, let  $OBS(\underline{v}) \subseteq VARS(\underline{v})$  be the set of observable variables in the representation  $(\underline{v}, DOM(\underline{v}))$  with the respective projection (see Fig. 3.4)

$$P_{obs} : DOM(\underline{v}) \rightarrow DOM(\underline{v}_{obs}).$$

### Definition 3.3 (Observable Test Set)

A test set  $\{T_i\}$  is observable, if all  $T_i$  are observable, i.e.  $T_i \subseteq DOM(\underline{v}_{obs})$ .

Lemma 3.1 indicates the way to generate confirming (observable) test sets for some behavior  $B_0 \in BEHVS$ : we have to find (observable) distinctions between  $B_0$  and each other mode  $B_j$ , and confirm these distinctions to be present. We can grasp them as the set differences

$$D_j := P_{obs}(R_0) \setminus P_{obs}(R_j)$$

of appropriate modeling relations of these behaviors. The number of differences  $D_j$  can be smaller than the number of behaviors to be refuted, because the modeling relations chosen may cover several behaviors (For the thyristor, for instance,  $R_{RED-BO}$  covers an infinite set of behaviors).

We even do not have to enumerate all behaviors in  $BEHVS$ , and we do not have to be able to describe them in detail; we only have to be sure that the set of relations  $\{R_j\}$  covers all possible behaviors. Table 3.2 and Fig. 3.1 show the set differences obtained for the thyristor example from the relations in Table 3.1 (in our case  $OBS(\underline{v}_{Th}) = VARS(\underline{v}_{Th})$  holds, i.e. all variables are considered observable).

	$\Delta V$	gate i	
<b>D<sub>block</sub></b>	$(V_{Th}, V_{Bo}]$	$\{1\}$	$[(res-\Delta)*\Delta V, (res+\Delta)*\Delta V]$
	$(V_{Bo}, \infty)$	$\{0,1\}$	$[(res-\Delta)*\Delta V, (res+\Delta)*\Delta V]$
<b>D<sub>punct</sub></b>	$(-\infty, 0]$	$\{0,1\}$	$[0, \delta]$
	$(0, V_{Th}]$	$\{0,1\}$	$[0, \delta]$
	$(V_{Th}, V_{Bo}]$	$\{0\}$	$[0, \delta]$
<b>D<sub>Red-Bo</sub></b>	$(V_{Bo}, V_{Bo}]$	$\{0\}$	$[0, \delta]$

**Table 3.2** The differences between the relation characterizing the correct behavior and the fault mode relations

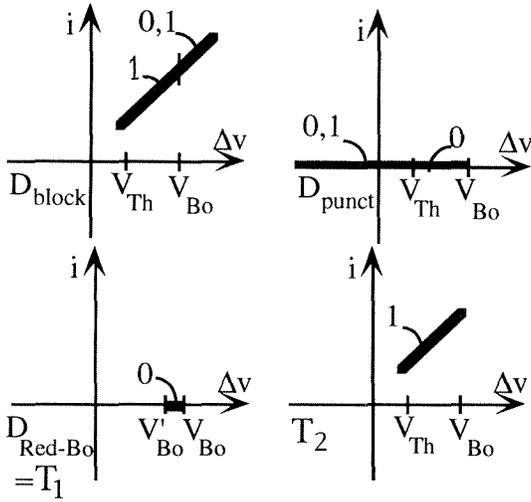


Figure 3.1 The relations of Table 3.1 and the tests of Table 3.3

Any observable test refuting  $M(R_i)$  and containing only tuples consistent with  $M(R_0)$  must be a subset of  $D_i$ . Although we could use  $\{D_i\}$  as a test set, we may further reduce the number of tests by replacing several  $D_i$  by a common subset. We call a set of sets,  $\{T_k\}$ , a *hitting set of sets* of  $\{D_i\}$ , if it contains a non-empty subset of each  $D_i$ :

$$\forall D_i \exists T_k \emptyset \neq T_k \subseteq D_i.$$

The following lemma is the basis for the generation of observable confirming test sets:

### Lemma 3.2

Let

$$\{R_i \mid R_i \subseteq \text{DOM}(\underline{v})\}$$

cover all behaviors (except  $B_0$ ):

$$\forall B_j \in \text{BEHVS} \setminus \{B_0\} \exists R_i B_j \Rightarrow M(R_i),$$

and  $R_0 \subseteq \text{DOM}(\underline{v})$  cover  $B_0$ :

$$B_0 \Rightarrow M(R_0).$$

If  $\{T_k\}$  is a hitting set of sets of

$$\{D_i\} := \{p_{\text{obs}}(R_0) \setminus p_{\text{obs}}(R_i)\},$$

then it is an observable confirming test set for  $B_0$ .

The thyristor test set is an illustration of Lemma 3.2.

Since  $D_{\text{Red-Bo}} \subseteq D_{\text{punct}}$ , the set of relations

$$\{D_{\text{block}}, D_{\text{Red-Bo}}\}$$

is a hitting set of sets of

$$\{D_{\text{block}}, D_{\text{punct}}, D_{\text{Red-Bo}}\}$$

and forms an observable confirming test set for the correct behavior.

We also obtain a necessary condition for the existence of a confirming test set: if  $B_0$  is actually a restriction of some other behavior  $B_j$ , it is impossible to find a confirming test set for  $B_0$ . An example for this case is an intermittent fault which is characterized by a relation that covers the correct behavior entirely (because sometimes

the constituent behaves correctly). This is intuitive, because even if we observe only value tuples consistent with the correct behavior, so far, we can never be sure that the future will not reveal contradictory observations (whenever the fault occurs). Note that even if  $R_0 \setminus R_i$  is non-empty,  $D_i$  may be empty, because the distinction is not observable in the given representation.

Now we have determined test sets that confirm a particular behavior, if they are observed. However, we do not want to wait for them to drop from heaven, but we would like to enforce them by an appropriate *causal input* to the system.

### 3.2 Finding Deterministic Test Inputs

We assume that the causal variables are observable, which is reasonable, because it means we know what we are doing to the constituent. So, let

$$\text{CAUSE}(\underline{v}) \subseteq \text{OBS}(\underline{v}) \subseteq \text{VARS}(\underline{v})$$

be the set of susceptible variables and

$$p_{\text{cause}} : \text{DOM}(\underline{v}) \rightarrow \text{DOM}(\underline{v}_{\text{cause}})$$

$$p'_{\text{cause}} : \text{DOM}(\underline{v}_{\text{obs}}) \rightarrow \text{DOM}(\underline{v}_{\text{cause}})$$

the respective projections into the set of input tuples (see Fig. 3.4). What we would like to have is test inputs, i.e. subsets of  $\text{DOM}(\underline{v}_{\text{cause}})$ , that are guaranteed to determine whether or not a particular behavior is present. More precisely: if we input one tuple out of each set to the constituent, the resulting value tuples of  $\underline{v}$  deterministically either confirm or refute the behavior:

#### Definition 3.4 (Test Input, Deterministic Input Set)

A test input is a non-empty relation on  $\text{DOM}(\underline{v}_{\text{cause}})$ :

$$T_i \subseteq \text{DOM}(\underline{v}_{\text{cause}}).$$

A set of test inputs  $\{T_i\}$  is deterministic for a behavior  $B_0 \in \text{BEHVS}$  iff for all sets

$$V = \{v_i\} \subseteq \text{DOM}(\underline{v})$$

whose set of causes  $\{p_{\text{cause}}(v_i)\}$  forms a hitting set of  $\{T_i\}$ , observation of  $V$  is inconsistent with  $B_0$  or it entails it:

$$\varphi_v \models \neg B_0 \text{ or } \varphi_v \models B_0$$

How can we generate deterministic input sets? Unfortunately, for a test set  $\{T_i\}$  confirming  $B_0$ , the input set  $\{p_{\text{cause}}(T_i)\}$  is not necessarily deterministic.

To illustrate this, we consider the relation  $R_{\text{neg}}$  which is a subset of  $R_{\text{ok}} \setminus R_{\text{punct}}$  (for  $\Delta V < 0$ ) and which could be used to rule out the fault "punctured" of the thyristor (Fig. 3.2).  $p_{\text{cause}}$  projects to  $(\Delta V, \text{gate})$ :

$$p_{\text{cause}}(R_{\text{neg}}) = (-\infty, -\epsilon) \times \{0\}.$$

However, if we choose a test input with  $(\Delta V, \text{gate})$  out of  $(-\infty, -\epsilon) \times \{0\}$ , a value of  $i$  might be observed such that the vector lies in the intersection of  $R_{\text{ok}}$  and  $R_{\text{punct}}$  (indicated by "x" in Fig. 3.2) and, hence, is consistent with the correct behavior but also fails to refute the fault. As a cure, we have to exclude  $p_{\text{cause}}(R_{\text{ok}} \cap R_{\text{punct}})$ , i.e. to reduce the test input for  $\Delta v$  to  $(-\infty, \epsilon)$ .

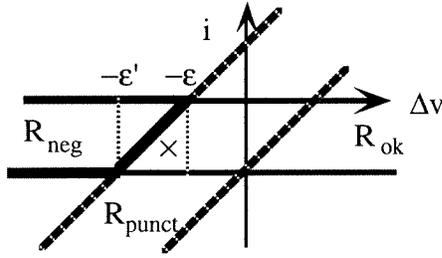


Figure 3.2  $p_{\text{cause}}(R_{\text{ok}} \setminus R_{\text{punct}})$  and  $p_{\text{cause}}(R_{\text{ok}} \cap R_{\text{punct}})$  overlap

More generally, in order to construct input sets deterministic for some  $B_0 \in \text{BEHVS}$  and leading to observable test sets, for each  $B_i \neq B_0$  we have to determine and eliminate those inputs that possibly lead to the same observations under both  $B_0$  and  $B_i$ . This is the set

$$P'_{\text{cause}}(\text{Pobs}(R_0) \cap \text{Pobs}(R_i)).$$

Hence, if we define

$$DI_i := p_{\text{cause}}(R_0) \setminus P'_{\text{cause}}(\text{Pobs}(R_0) \cap \text{Pobs}(R_i)),$$

then we are guaranteed that any input chosen from  $DI_i$  causes an observable value tuple that is inconsistent with  $M(R_i)$  or with  $M(R_0)$  (possibly with both of them). This is the idea underlying the proof of Theorem 3.3.

### Theorem 3.3

Under the conditions of Lemma 3.2, each set of test inputs  $\{TI_k\}$  that is a hitting set of sets of

$$\{DI_i\} = p_{\text{cause}}(R_0) \setminus P'_{\text{cause}}(\text{Pobs}(R_0) \cap \text{Pobs}(R_i))$$

is deterministic for  $B_0$  and

$$\{T_k\} := \{p_{\text{obs}}(R_0) \cap P'^{-1}_{\text{cause}}(TI_k)\}$$

is an observable confirming test set for  $B_0$ .

In practice, one wants to avoid test inputs that are extreme and possibly cause (or make worse) damage. For instance, we do not want to test with  $\Delta V > V_{B_0}$ , because the thyristor could be destroyed. In this case,  $DI_i$  may have to be further reduced by intersecting it with a set of admissible inputs:

$$DI_{\text{adm}} := R_{\text{adm}} \cap DI_i.$$

For the thyristor, we choose

$$R_{\text{Th adm}} = (-\infty, V_{B_0}] \times \{0, 1\} \times \mathbb{R}$$

and reduce the tests we have obtained, so far, to

$$T_1 = R_{\text{Th adm}} \cap D_{\text{block}}.$$

$$T_2 = R_{\text{Th adm}} \cap D_{\text{Red-Bo}} = D_{\text{Red-Bo}}$$

which yields the test set we proposed in section 2 (see Table 3.3 and Fig. 3.1).

	$\Delta V$	gate	$i$
<b>T1</b>	$(V_{\text{Th}}, V_{B_0}]$	{1}	$[(\text{res}-\Delta)*\Delta V, (\text{res}+\Delta)*\Delta V]$
<b>T2</b>	$(V'_{B_0}, V_{B_0}]$	{0}	$[0, \delta]$

Table 3.3 Set of two tests confirming the correct thyristor

Although in this example, the non-admissible range is related to the correct behavior, in general  $R_{\text{adm}}$  can also be chosen reflecting potential faults: for instance, if a pipe potentially has a crack, one might want to avoid high pressure even though this causes no problems for a proper pipe.

Lemma 3.2 does not prevent us from constructing observable tests that are not real, but rather an artificial result of the choice of model relations: a non-empty  $D_i = p_{\text{obs}}(R_0) \setminus p_{\text{obs}}(R_i)$  may be due to choosing  $R_0$  much larger than what is covered by the behavior, and  $D_i$  potentially contains only physically impossible values. In contrast, simply because nothing prevents us from causing inputs and observing observables, we have

### Theorem 3.4

The existence of a deterministic input set ensures the existence of an observable and controllable test set in reality.

### 3.3 A Test Generation Algorithm

Here, we outline a family of algorithms (Fig. 3.3) based on Theorem 3.3, and discuss it briefly.

```

TI-SET = NIL
FOR R IN MODEL-RELATIONS DO
(1)  $DI = R_{\text{adm}} \cap p_{\text{cause}}(R_0) \setminus P'_{\text{cause}}(\text{Pobs}(R_0) \cap \text{Pobs}(R))$ 
(2) IF  $DI = \emptyset$ 
    THEN "No (adm.) deterministic test input against" R
(3)  $DI = R_{\text{adm}} \cap p_{\text{cause}}(\text{Pobs}(R_0) \setminus \text{Pobs}(R))$ 
    IF  $DI = \emptyset$ 
    THEN "No (adm.) observable test against" R
    GOTO .NEXT
    Select  $TI \in \text{TI-SET}$  with  $DI \cap TI \neq \emptyset$ 
    IF TI exists
(4) THEN  $TI = TI \cap DI$ 
(5) ELSE Append DI to TI-SET
.NEXT
END FOR
FOR TI IN TI-SET
(6) Collect  $p_{\text{obs}}(R_0) \cap P'^{-1}_{\text{cause}}(TI)$  in T-SET

```

Figure 3.3 An algorithm for generating (preferably deterministic) test inputs  $TI$  and test sets  $T$  confirming  $B_0$

The algorithm iterates over the model relations of behaviors  $B_i \neq B_0$  and attempts to create an admissible input set that discriminates between  $R_0$  and  $R_i$  deterministically and in an observable way according to the above definition of  $DI_i$  (step 1). If this is impossible (2), it determines in (3) the admissible input set corresponding to an observable test (obtained as  $D_i$  according to Lemma 3.2) – which may fail, as well.

If there exist input sets from previous iterations with a non-empty intersection with the new  $DI$ , one of them is selected and replaced by this intersection (4). Thus, we

account for the behavior(s) corresponding to the current  $R$  without increasing the number of tests. Otherwise, the current DI is added as a new test input in itself (5). In step 6, an observable test set is constructed from the final input set according to Theorem 3.3. It is confirming  $B_0$ , if all  $R_i$  could be accounted for. The algorithm generates the two tests for the thyristor mentioned in section 2. The selection of TI for step 4 opens space for variations and heuristics. For instance, simply the first one with a non-empty intersection could be chosen, or the one with the largest intersection. The latter strategy always requires intersection with all existing input sets and assessment of the result, but may get closer to the optimum w.r.t. the number of tests generated. This algorithm produces the test set for the thyristor that is shown in Table 3.3.

If there exists a single test, the algorithm generates it in linear time. In other cases, it is quadratic w.r.t. the number of model relations (which may be less than the number of behaviors) and may fail to generate a test set of minimal cardinality. Its result, including whether or not an existing minimal cardinality test set is found, can depend on the ordering of the model relations. In many domains, it will pay off to use more elaborate and costly algorithms in order to reduce the number of tests required.

### 3.4 Making Test Generation Feasible through Model Abstraction

For physical systems with large or continuous domains and complex behavior, the question arises whether it is practically feasible to compute projections, intersections and set differences. The answer is that we do not have to. As in section 2, we want to make test generation for such domains feasible by performing it with model relations in an abstract representation (with small domains). We formalize this procedure and show its correctness. The key idea is simple: If  $M(R_i)$  is a model of  $B_i$ , i.e.

$$B_i \Rightarrow M(R_i),$$

and if  $R'_i$  is another relation (preferably in a finite domain) that specifies a weaker model, i.e.

$$M(R_i) \Rightarrow M(R'_i),$$

then refuting  $M(R'_i)$  suffices to rule out  $B_i$ . Hence, we can build test sets from such finite relations  $R'_i$ . The task is then to find conditions and a systematic way to generate models that are guaranteed to be weaker (in the logical sense specified above) by switching to a different representation  $(\underline{v}', DOM'(\underline{v}'))$  with finite domains.

In (Struss 1992), a large class of transformations between representations is characterized by conditions that are both rather weak and natural:

#### Definition 3.5 (Representational Transformation)

A surjective mapping

$$\tau: DOM(\underline{v}) \rightarrow DOM'(\underline{v}')$$

is a representational transformation iff

$$\begin{aligned} \underline{v}(s) = \underline{v}_0 &\Rightarrow \underline{v}'(s) = \tau(\underline{v}_0) \\ \underline{v}'(s) = \underline{v}'_0 &\Rightarrow \exists \underline{v}_0 \in \tau^{-1}(\underline{v}'_0) \quad \underline{v}(s) = \underline{v}_0. \end{aligned}$$

(See again Fig. 3.4). This simply means that, in the same situation, variables in the different representations have values related by  $\tau$ . Under such representational transformations, models are preserved (Struss 1992):

#### Lemma 3.5

If

$$\tau: DOM(\underline{v}) \rightarrow DOM'(\underline{v}')$$

is a representational transformation, then

$$M(R) \Rightarrow M(\tau(R)) \quad \text{and} \quad M(R') \Rightarrow M(\tau^{-1}(R')).$$

This means, if we map a model relation from some original representation into a different one under a representational transformation the image will specify a weaker model, as required. In particular, we can choose a representation with a finite domain, construct (observable) confirming test sets and (deterministic) input sets in this representation from the transformed model relations and map them back to the original detailed representation.

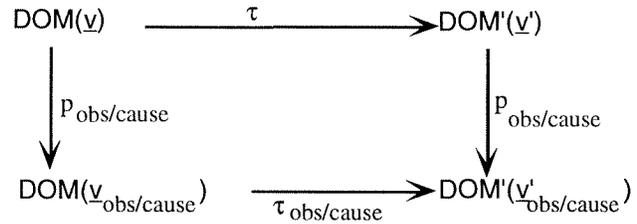


Figure 3.4 Diagram of projections and representational transformations

The following theorem states that this actually yields (deterministic) input sets and (observable) confirming test sets in the original representation, thus justifying the intuitive approach:

#### Theorem 3.6

Let

$$\tau_{obs}: DOM(\underline{v}_{obs}) \rightarrow DOM'(\underline{v}'_{obs})$$

and

$$\tau_{cause}: DOM(\underline{v}_{cause}) \rightarrow DOM'(\underline{v}'_{cause})$$

be representational transformations.

If  $\{T'_i\}$  is an observable confirming test set for  $B_0$  then so is

$$\{T_i\} := \{\tau^{-1}_{obs}(T'_i)\}.$$

If  $\{TI_i\}$  is a deterministic input set for  $B_0$ , then so is

$$\{TI_i\} := \{\tau^{-1}_{cause}(TI'_i)\}.$$

In particular, qualitative domain abstraction (mapping real numbers to a set of landmarks and the intervals between them) is a representational representation. In the thyristor example, the landmarks can be chosen as 0,  $V_{Th}$ ,  $V_{B_0}$ ,  $V_{B_0}$  for  $\Delta V$ , and 0,  $-\delta$ , and  $\delta$  for  $i$ . Ignoring the purely theoretical problem of separately treating the landmark points, this introduces quantity spaces  $Q_5$  consisting of

neg :=  $(-\infty, 0]$   
 small :=  $(0, V_{Th}]$   
 medium :=  $(V_{Th}, V_{Bo}]$   
 high :=  $(V_{Bo}, V_{Bo}]$   
 too high :=  $(V_{Bo}, \infty)$

for  $\Delta V$  and  $Q_3$  with

- :=  $(-\infty, -\delta)$   
 0 :=  $[-\delta, \delta]$   
 + :=  $(\delta, \infty)$

for  $i$ , respectively. Mapping real values for  $\Delta V$  and  $i$  to the intervals of  $Q_5$  and  $Q_3$ , respectively, they are contained in, defines the qualitative abstraction

$$\tau_q: \mathbb{R} \times \{0,1\} \times \mathbb{R} \rightarrow Q_5 \times \{0,1\} \times Q_3$$

by

$$\tau_q((\Delta V_0, gate_0, i_0)) = (q_{v0}, gate_0, q_{i0}).$$

where

$$\Delta V_0 \in q_{v0} \in Q_5 \text{ and } i_0 \in q_{i0} \in Q_3$$

Under the reasonable assumption that the holding current is greater than the leakage current:

$$(res - \Delta) * V_{Th} > \delta,$$

the representational transformation then induces model relations

$$R'_i := \tau_q(R_i) \subset Q_5 \times \{0,1\} \times Q_3$$

from the relations  $R_i$  in the real-valued representation. They are displayed in Table 3.4. (Since the two columns on the left for  $(\Delta V, gate)$  represent the causes, one can immediately see the non-determinism of the model  $M(R'_{Red-Bo})$  for (medium, 0) which is due to the fact that it covers a whole class of behaviors with actual breakover voltage anywhere between  $V_{Th}$  and  $V_{Bo}$ ).

The table also shows the respective set differences  $D'_i$  in the new representation which can easily be determined from a comparison of the  $R'_{ok}$  column with respective  $R'_i$  column. The admissible range excludes *toohigh* from  $Q_5$ :

$$R'_{adm} = Q_5 \setminus \{toohigh\} \times \{0,1\} \times Q_3,$$

and the algorithm of section 3.3 intersects  $D'_{Red-Bo}$  and

$D'_{punct}$  yielding the observable confirming test set

$$T'_1 = \{\text{medium, high}\} \times \{1\} \times \{+\},$$

$$T'_2 = \{(\text{high}, 0, 0)\}$$

and the deterministic input set

$$\{\{\text{medium, high}\} \times \{1\}, (\text{high}, 0)\}.$$

Mapping back the tests to the real domain under  $\tau_q^{-1}$  produces a test set

$$\{\tau_q^{-1}(T'_1), \tau_q^{-1}(T'_2)\}$$

$$= \{(V_{Th}, V_{Bo}) \times \{1\} \times (\delta, \infty),$$

$$(V_{Bo}, V_{Bo}) \times \{0\} \times [-\delta, \delta]\}$$

which covers the test set  $\{T_1, T_2\}$  shown in Table 3.3. Of course, the abstract representation may be too coarse to allow for the separation of particular behaviors. We can use this as a criterion for selecting representations and behavior models, for instance, as the highest level that still allows to distinguish one behavior from the others.

One may be amazed that Theorem 3.6 requires  $\tau_{obs}$  and  $\tau_{cause}$  to be representational transformations, and tempted to find a weaker sufficient condition. This is briefly discussed in the following subsection which is not essential and may be skipped.

### 3.5 Preservation of Observable Distinctions

Recall the procedure for test generation: its basis is to identify *observable distinctions* of behavior model relations. The observability of this distinction must not be destroyed under back-transformation by  $\tau^{-1}$ . In other words, if  $\underline{v}'_1$  and  $\underline{v}'_2$  can be observed as being distinct:

$$p'_{obs}(\underline{v}'_1) \neq p'_{obs}(\underline{v}'_2),$$

then the observable parts of their pre-images  $\tau^{-1}(\underline{v}'_i)$  must also be disjoint. This seems to be the weakest sufficient condition we can impose, and it is captured by the following definition (which we formulate for the causes, as well, because this is needed for the construction of deterministic input sets for analogous reasons).

$\Delta V$	gate	i						
		$R'_{ok}$	$R'_{Red-Bo}$	$R'_{block}$	$R'_{punct}$	$D'_{Red-Bo}$	$D'_{block}$	$D'_{punct}$
neg	0	0	0	0	-			0
neg	1	0	0	0	-			0
small	0	0	0	0	+			0
small	1	0	0	0	+			0
medium	0	0	0,1	0	+			0
medium	1	+	+	0	+		+	
high	0	0	+	0	+	0		0
high	1	+	+	0	+		+	
toohigh	0	+	+	0	+		+	
toohigh	1	+	+	0	+		+	

**Table 3.4** The tuples constituting the relations  $R'_i$  and the set differences  $D'_i$  in the qualitative representation. The values for the current  $i$ , in the respective column complements the pair for  $(\Delta V, gate)$  in each line.

**Definition 3.6 (Faithful w.r.t. observable (causal) distinctions)**

A representational transformation

$$\tau: DOM(v) \rightarrow DOM'(v')$$

is called faithful w.r.t. observable distinctions iff

$$\forall \underline{v}'_1, \underline{v}'_2 \in DOM'(\underline{v}') \quad (p'_{obs}(\underline{v}'_1) \neq p'_{obs}(\underline{v}'_2)) \\ \Rightarrow p_{obs}(\tau^{-1}(\underline{v}'_1)) \cap p_{obs}(\tau^{-1}(\underline{v}'_2)) = \emptyset.$$

It is called faithful w.r.t. causal distinctions if the analogous property holds for  $p_{cause}$ .

The property that the restriction of  $\tau$  to observables (causes, respectively) is also a representational transformation, which is used as a condition in the previous subsection, will be called decomposability:

**Definition 3.7 (Decomposable)**

If there exists a mapping

$$\tau_{obs}: DOM(\underline{v}_{obs}) \rightarrow DOM'(\underline{v}'_{obs})$$

such that

$$p'_{obs} \circ \tau = \tau_{obs} \circ p_{obs},$$

then  $\tau$  is called obs-decomposable.

It is called cause-decomposable if there exists a

$$\tau_{cause}: DOM(\underline{v}_{cause}) \rightarrow DOM'(\underline{v}'_{cause})$$

with

$$p'_{cause} \circ \tau = \tau_{cause} \circ p_{cause}.$$

(Here " $\circ$ " is the composition of transformations). Note that we only require that  $\tau_{obs}$  ( $\tau_{cause}$ , resp.) be a mapping. This is because any such mapping "inherits" the properties of a representational transformation from the three involved mappings:

**Lemma 3.7**

If there exists  $\tau_{obs}$  ( $\tau_{cause}$ ) with the properties of Definition 3.7, then  $\tau_{obs}$  ( $\tau_{cause}$ ) is a representational transformation.

The following lemma states that this concept is only seemingly stronger than the one of Definition 3.6.

**Lemma 3.8**

A representational transformation

$$\tau: DOM(\underline{v}) \rightarrow DOM'(\underline{v}')$$

is faithful w.r.t. observable (causal) distinctions iff  $\tau$  is obs-decomposable (cause-decomposable).

**Remark 3.9**

$\tau_{cause}$  being a representational transformation is also a **necessary** condition for the validity of backtransformation of tests in the following sense: If it is violated, we can construct behaviors and model relations such that there exist observable test sets with deterministic input sets for them in the abstract representation, but none in the stronger one. However, these constructed behaviors may be irrelevant to any real physical system, and the back-transformation of tests may work for the practical cases nevertheless.

**4 Testing Constituents in an Aggregate**

Quite often the constituent to be tested is embedded in a particular context, namely an environment consisting of other interacting constituents, and only the entire aggregate can be controlled and observed. Our approach is general enough to cover this case.

We regard the aggregate as the constituent to be tested, and observables and causes are related to this aggregate constituent. The goal is to confirm one behavior of this aggregate constituent by refuting the other behaviors out of a certain set. This set is given as the behaviors of the aggregate resulting from the different behaviors of the constituent embedded in it.

More formally, let a constituent  $C_0$  be in a particular context CTX consisting of constituents  $C_1, \dots, C_n$  with their respective variables. The aggregate is

$$C_{agg} = \{C_j\} \cup \{C_0\},$$

and representations for describing the aggregate's behavior can be obtained from the representations for single constituents by taking the union of the local variables. For the sake of simplicity, we assume that all local relations are already specified in the aggregate representation. Issues that arise if the assumption is dropped are discussed in (Struss 1994).

If  $M(R_j)$  are behavior models for constituents  $C_j$ , then

$$R_{CTX} = \bigcap R_j$$

specifies a corresponding behavior model for  $CTX = \{C_j\}$ . If  $M(R_{i0})$  are models of the behaviors  $B_i$  of  $C_0$ , then the relations

$$R_i = R_{CTX} \cap R_{i0}$$

specify models of the behaviors of

$$C_{agg} = CTX \cup \{C_0\}$$

produced by the behaviors of  $C_0$  in CTX. In applying the test generation algorithm to these relations, we can construct observable tests and deterministic test inputs for the behavior of  $C_{agg}$  that involves the particular behavior  $B_0$  of  $C_0$ .

**Corollary 4.1**

Let

$$C_{agg} = CTX \cup \{C_0\} = \{C_j\} \cup \{C_0\},$$

$$R_{CTX} = \bigcap_j R_j,$$

where the  $M(R_j)$  are behavior models of the  $C_j$ .

Let  $M(R_{i0})$  be models of the behavior modes  $B_i$  of  $C_0$ .

Define

$$R_{agg\ i} := R_{CTX} \cap R_{i0}.$$

$$D_i := p_{obs}(R_{agg\ 0}) \setminus p_{obs}(R_{agg\ i})$$

$$DI_i := p_{cause}(R_{agg\ i}) \setminus$$

$$p'_{cause}(p_{obs}(R_{agg\ 0}) \cap p_{obs}(R_{agg\ i})).$$

If  $M(R_{CTX})$  holds, then each hitting set of sets  $\{T_k\}$  of  $\{D_i\}$  is an observable confirming test set for  $B_0$  of  $C_0$ , and each hitting set of sets  $\{TI_k\}$  of  $\{DI_i\}$  is a deterministic input set for  $B_0$ .

Since  $p_{\text{cause}}$  and  $p_{\text{obs}}$  project to input sets and observables of  $C_{\text{agg}}$ , the tests are observable and controllable *through*  $C_{\text{agg}}$ . Of course, this provides a confirming test set for  $B_0$  of  $C_0$ , *only if*  $M(R_{\text{CTX}})$  holds. This corresponds, for instance, to the widespread assumption that while testing a constituent, its context works properly. However, we can also generate tests based on the assumption that the context may contain particular faults, which, for instance, have been hypothesized by a diagnosis step.

By constructing all behavior modes of  $C_{\text{agg}}$  corresponding to a single fault of any constituent, we can generate a test set confirming the correctness of all constituents under this assumption.

## 5 Realization of Testing

Now we have to implement a test system, i.e. a program that takes the test inputs and the observed responses of the device and returns whether the respective behavior has been confirmed or refuted. For this purpose, we do not have to invent a new machinery but can apply an existing diagnostic system. Tests confirming a behavior are based on refuting models of all other behaviors. Refuting behaviors through observations is also the principle of consistency-based diagnosis (de Kleer, Mackworth & Reiter 1990), and we can implement testing through one of the consistency-based diagnosis engines,  $GDE^+$  (Struss & Dressler 1989).

In more detail,  $GDE^+$  represents a constituent by the set of behavior models  $M(R_i)$ . If a complete test set  $\{T_k\}$  is observed, i.e.  $\varphi_V$  holds for some hitting set  $V$  of  $\{T_k\}$ , then we have

$$\forall T_k \exists s \in \text{SIT} \exists v_k \in T_k \quad v(s) = v_k.$$

By construction, there exists for each  $D_i := R_0 \setminus R_i$  at least one  $T_k \subseteq D_i$ . Hence, it follows

$$\forall i \neq 0 \exists s \in \text{SIT} \exists v_i \in D_i \quad v(s) = v_i,$$

which means  $GDE^+$  refutes all behaviors except  $B_0$ :

$$\forall i \neq 0 \exists s \in \text{SIT} \exists v_i \notin R_i \quad v(s) = v_i$$

$$\Rightarrow \forall i \neq 0 \neg M(R_i) \Rightarrow \forall i \neq 0 \neg B_i.$$

Then  $GDE^+$  confirms  $B_0$  by applying its "physical negation" rule (stating the completeness of the enumerated behaviors)

$$\neg B_1 \wedge \neg B_2 \wedge \dots \wedge \neg B_n \Rightarrow B_0.$$

Of course, observation of a value outside  $R_0$  lets  $GDE^+$  refute  $B_0$ . In summary,  $GDE^+$  makes the inferences required for the application of a deterministic input set.

Note that, for the purpose of testing, we can replace the constituent's model set  $\{M(R_i)\}$  by the complements of the tests,  $\{M(T_k^c)\}$ , thus potentially reducing the number and, perhaps, the complexity of models to be checked. (Again, the details are discussed in (Struss 94)).

## 6 Discussion, Future and Related Work

### 6.1 What Is Achieved?

We make the rather strong claim that the theory presented here *solves* the problem of testing physical systems. It solves it "in principle", in the same sense as model-based diagnosis is a solution to the problem of fault localization and identification. By this, we want to emphasize two aspects:

- On the positive side, it is a *general* theory covering large classes of devices, for which there exists no formal theory or systematic solution of the testing problem today. All other solutions to test generation are only variations of this principled approach, perhaps by applying heuristics, making certain assumptions, or exploiting particularities of the domain (For instance, we can show that the D-algorithm (Roth 1980) is a specialization of our algorithm for digital circuit testing).
- On the problem side, it is a solution only "in principle", because it shifts the burden to the hard task of modeling. The application to a particular domain may require substantial work and even be impossible with the knowledge available about the domain. The crucial issues are in modeling and complexity. We briefly mention some of them.

### 6.2 Application Prerequisites and Problems

*Knowledge about the Possible Faults:* Particularly people from model-based diagnosis may be sceptical about the necessity of (complete sets of) fault models for this approach. However, knowledge (or assumptions) about the possible faults is not a drawback of our system, but is *inherent to the task of testing*. In contrast to diagnosis, where we may be content with refutation of (correct) behaviors, testing aims at confirming a particular behavior, usually the correct one. This is impossible, unless we make certain assumptions about the other possible behaviors, although this may happen unconsciously and implicitly. (This is why we are talking about testing of *physical systems*, and, for instance, not about testing system designs or software.) We may deliberately exclude some faults from the set of behaviors, or, more precisely, choose model relations that do not cover some fault behaviors, e.g. because we assume they are unlikely or irrelevant. For instance, we ignored thermal triggering of the thyristor and based the model on the assumption that the turn-ON time and spreading time are negligible. Our approach has the advantage to make such assumptions explicit (and the multiple modeling framework allows us to treat them as defeasible hypotheses, see (Struss 1992)).

*Models of Behavior Modes:* We have to be able to turn our knowledge about the correct and the faulty behavior into models, relational models in our case. The

representation through relations is quite natural for broad classes of physical systems. Note that the models are *not* required to be *deterministic* (remember the model of the class of thyristor faults called "Reduced  $V_{B0}$ "). A major problem is finding appropriate models of devices with complex *dynamic* behavior. The thyristor, a dynamic device, illustrates that it can be possible to do the testing under temporal abstraction.

*Abstract and Qualitative Models:* Model abstraction is the key for the feasibility of the algorithm. But the models have to be strong enough to distinguish the behavior mode of interest from the other ones.

*Structural Complexity:* If aggregates are getting large, testing of constituents in its context may turn infeasible because of the number of fault models of the aggregate, (even though run time of several days for test generation may be insignificant compared to savings in testing time and costs). We do not expect the algorithm to handle systems with thousands of components in a flat structure. But first experiments suggest that it can produce results in a reasonable amount of time for devices which are complex enough to prohibit the completeness and/or optimality of manually generated tests. Currently, we are exploring binary-decision diagrams as a compact representation of the model relations. The core of the approach is finite constraint satisfaction. It will benefit from exploiting the specificity of the task, e.g. the device structure.

### 6.3 Perspectives

In this paper, we considered only testing with the goal of *confirming one particular mode*. Obviously, there is a generalization possible that creates tests for *identifying the present mode*. Testing for discrimination is relevant to diagnosis (Meerwijk & Preist 1992) and fits very well with a consistency-based diagnosis engine. It can be combined with probabilities of modes (and of tuples for non-deterministic models) and forms a generalization of the probe selection strategy used in (de Kleer & Williams 1987). Additionally, this approach provides a basis for a formal assessment of the (discriminating) power of representations. Testing in the context of diagnosis is the subject of another paper. Another direction is the exploitation of the strategy for design purposes: it allows to analyze whether or not and where it is (or would be) possible to detect, discriminate, and identify faults of constituents in a designed system, thus supporting design for testability and sensor placement (see (Chien, Doyle & Rouquette 1991), (Scarl 1991)).

In summary, we presented an approach to model-based test generation and testing that makes a large class of systems amenable to principled methods and well-founded algorithms. The exploitation of model abstraction is crucial to making the task practically feasible for an interesting class of technical systems, notwithstanding the fact that the general task of hypothesis testing is np-complete (McIlraith 1993). Finally, the basis of the theory is quite simple, simple enough to be powerful.

## Acknowledgements

This work has been supported in part by the Christian-Doppler-Labor of the Technical University of Vienna.

## References

- Camurati, P., Medina, D., Prinetto, P., and Sonza, M. 1990, *A Diagnostic Test Pattern Algorithm*. In: Proceedings IEEE International Test Conference, 52-58
- Chien, S., Doyle, R., and Rouquette, N. 1991, *A Model-based Reasoning Approach to Sensor Placement for Diagnosability*. In: Working Notes of the Second International Workshop on Principles of Diagnosis, Milano, 181-190
- de Kleer, J., Mackworth, A., and Reiter, R. 1990, *Characterizing Diagnoses*. In Proceedings of the AAAI 90, 324-330.
- Gupta, A., and Welham, R. 1989, *Functional Test Generation for Digital Circuits*. In: J. S. Gero (ed.), Proceedings of Artificial Intelligence in Engineering, Learning and Diagnosis, Elsevier
- McIlraith, S. 1993, *Generating Tests Using Abduction*. In Working Papers of the Fourth International Workshop on Principles of Diagnosis, Aberystwyth, 223-235.
- Meerwijk, A., and Preist, C. 1992, *Using Multiple Tests for Model-based Diagnosis*. Working Papers of the Third International Workshop on Principles of Diagnosis, Rosario
- Roth, G. P. 1980, *Computer Logic, Testing, and Verification*. Rockville: Computer Science Press.
- Scarl, E. 1991, *Analysis of Diagnosability*. In: Working Notes of the Second International Workshop on Principles of Diagnosis, Milano, 191-200
- Struss, P. 1992, *What's in SD? Towards a Theory of Modeling for Diagnosis*. In: Hamscher, W. Console, L., and de Kleer, J. eds., Readings in Model-based Diagnosis. San Mateo: Morgan Kaufmann: 419-449.
- Struss, P. 1994, *A Theory of Testing Physical Systems Based on First Principles*, Technical Report 94/63, Christian-Doppler-Labor, Technical University of Vienna.
- Struss, P. 1994a, *Multiple Models of Physical Systems - Modeling Intermittent Faults, Inaccuracy and Tests in Diagnosis*. To appear in: Annals of Mathematics and Artificial Intelligence .
- Struss, P., Dressler, O. 1989, *"Physical Negation" - Integrating Fault Models into the General Diagnostic Engine*. In Proc. 11th Int. Joint Conf. on Artificial Intelligence, Detroit, MI, 1318-1323.

# Supporting Creative Mechanical Design

Kun Sun and Boi Faltings

Artificial Intelligence Laboratory (LIA)  
Swiss Federal Institute of Technology (EPFL)  
IN - Ecublens, 1015 Lausanne, Switzerland  
Fax: +41-21-693-5225  
e-mail: sun@lia.di.epfl.ch

## Abstract

Knowledge-based CAD systems limit designers' creativity by constraining them to work with the prototypes provided by the systems' knowledge bases. We investigate knowledge-based CAD systems capable of *supporting* creative designs in the example domain of elementary mechanisms.

We present a technique based on qualitative explanations which allows a designer to extend the knowledge base by demonstrating a structure which implements a function in a creative way. Structure is defined as the geometry of the parts, and function using a general logical language based on qualitative physics. We argue that the technique can accommodate any creative design in the example domain, and we demonstrate the technique using an example of a creative design.

The use of qualitative physics as a tool for extensible knowledge-based systems points out a new and promising application area for qualitative physics.

## Introduction

We consider those knowledge-based CAD systems where designers can compose designs from a library of *prototypes* ([Gero,90]) which the CAD system "knows" how to instantiate and adapt during the design process. Such systems limit designers' creativity by constraining them to designs which can be constructed as combinations of prototypes provided by the systems knowledge base. Creative ideas often fall outside this scope and thus cannot be accommodated in such a system.

We would like to propose that a design is creative if cannot be composed exactly from the prototypes in the system's knowledge base. The CAD system *supports* creative design if it allows the designer to define novel prototypes to cover his ideas. It *is* creative if it discovers new prototypes by itself. New prototypes are created by envisioning the prototypes of knowledge base in a different environment.

The techniques we describe in this paper allow designers to extend a prototype base by providing:

- a model of the function that their creative ideas address, expressed in a general logical language based on qualitative physics.
- a geometric model of a device that implements this function.

The system envisions the qualitative behavior of the device and identifies the behavior which implements the function that the designer intends. This allows the system to explain and generalize the idea, and define a new prototype for it. The new prototype can then be instantiated in any novel device the designer might want to construct with it. Note that the example device given by the designer can be very different from the device where the creative idea will be used. For example, a behavior observed in rocks could be reused in the design of a mechanism. If the representation languages used for structure and function are sufficiently general, our technique is guaranteed to cover any creative design which designers might propose.

Creativity is generally associated with extending a space of design alternatives or variables ([Gero,92, Sargent,92]). Such extensions are always possible only if the space of possible prototypes is unbounded. This is the case only if

- (i) the set of possible functions is infinite,
- (ii) the set of structures which can implement them is infinite, and
- (iii) there is no context-free mapping between primitives of structure and function which would allow composition of any prototype from a small set of primitive ones.

One such domain is designing geometric shapes which implement kinematic functions. For kinematic function, we define a generative qualitative representation language which allows an infinite set of different expressions. We represent shapes by polygons, which allow an infinite set of shapes. Kinematic functions are generated by contacts between shapes, and there thus exists a direct mapping between elementary functions

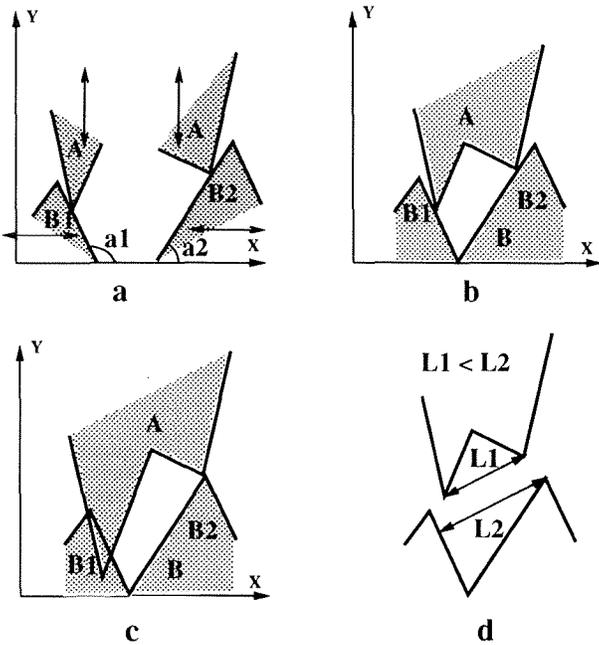


Figure 1: *Composing prototypes into a novel device.*

and structures. To satisfy requirement (iii), we must show that this mapping is not context-free, i.e. that the function of a particular structure is not independent of the context it is used in. Because of interactions through the geometry of the shapes, this is the case. As an example, consider the design of a device which prevents a block from dropping in the negative Y-direction, a function which could be represented as:

- Device (A, B)
- behavior:  $Y(A) \neq \ominus$

This function can be composed from two prototypes implemented by single contacts, shown in Figure 1 (a) and specified as follows:

- Device-1 (A, B<sub>1</sub>)
- behavior:  $Y(A) = \ominus \rightarrow X(B_1) = \ominus$
- Device-2 (A, B<sub>2</sub>)
- behavior:  $Y(A) = \ominus \rightarrow X(B_2) = \oplus$

The classical abductive reasoning says that if we know  $\beta$  and  $\alpha \rightarrow \beta$ , then the best explanation for  $\beta$  is that  $\alpha$  is true. It means that creative design can also be considered as finding the best explanation for the design goal.

We can expect the knowledge-based CAD system to be capable of proposing a composition where  $B_1 = B_2 = B$ , based on the following abductive reasoning:

- $Y(A) \neq \ominus$  if  $Y(A) = \ominus$  implies a contradiction.
- $X(B) = \ominus$  and  $X(B) = \oplus$  is a contradiction.

However, not all ways of composing the two devices actually lead to the composed function. For example, the composition of Figure 1 (b) is legal, but in Figure 1 (c) both contacts can not occur simultaneously and thus the device does not implement the desired function. This problem occurs whenever a contact can be *subsumed* by another. To satisfy the composed function, the device has to satisfy a *compositional* constraint (Figure 1 (d)) which become apparent only when the context of the device is known. But this also means that the composition of several prototypes defines a new and different prototype which was not known before. Thus, in designs which involve geometry there is no direct mapping between functions and structures, and such situations allow creative designs *even within fixed representation languages for structure and function.*

Now assume that a designer has decided to implement the support function by combining the two prototypes, a design which the CAD system does not know about. As the system's knowledge is insufficient to guarantee that the prototype composition will work as expected - the system does not know the difference between cases (b) and (c) in Figure 1 - the designer himself must draw a correct solution. By qualitative analysis of the device, our system then computes a causal *explanation* of the way it implements the given function. In particular, this explanation will show the additional constraint which has to be satisfied in order to avoid the subsumption in Figure 1 (c). This explanation is now used to automatically define a new prototype which is added to the systems knowledge base.

The system could be made creative by itself if it were provided with a mechanism for exploring possible functions and geometric structures. However, such a process must be guided by an evaluation of the interestingness of functions, known to be a very hard problem in the learning community. When a desired function is given, it is sometimes possible to use this to guide the process of exploration ([Faltings,92b]).

A more complex example of a kinematic pair is a clock escapement, shown in Figure 2. Using the theory of qualitative kinematics ([Faltings,90, Forbus *et al.*,91]), the qualitative behavior and function of any kinematic pair can be computed. Furthermore, it is possible to *invert* the computation to determine the limits up to which it is valid ([Faltings,92a]) and thus the explanation needed to define a new prototype. Our techniques concentrate on the *conceptual* design stage where creativity takes place. Subsequent *detail design* can further optimize the dimensions to accommodate non-qualitative specifications.

Our work builds on the idea of *prototypes* ([Gero,90]), but is also influenced by work on design systems using model-based abductive reasoning (for example: [Williams,90], [Neville and Weld, 92], [Sycara and Navinchandra,90]).

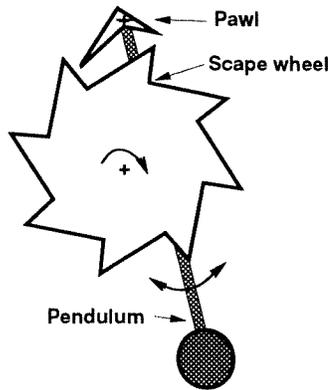


Figure 2: An escapement design produced using our system. The wheel is driven clockwise, and the pawl is attached to a pendulum which creates an oscillation with constant period.

However, existing work on model-based design has been based on the use of context-independent prototypes which do not consider the geometry of their composition. Design of kinematic chains by composing kinematic pairs has been studied in [Subramanian,93]. For certain classes of kinematic pairs, Subramanian gives an algorithm to determine their dimensions and arrangements in a chain under the assumption of uniform motion. Synthesis of kinematic pairs with considerations of geometric shapes has been studied by Joskowicz and Addanki ([Joskowicz and Addanki,88]), but they only treated the trivial cases of convex objects where no subsumptions and thus no compositional constraints can occur.

We first discuss the representation formalisms underlying our technique: a language for representing kinematic *function* based on qualitative physics, and a formalism for representing shape features. We then show how the analysis of a particular device can be generalized to define the shape features which are responsible for the its function and thus define a new prototype, and how this prototype can be used in design. Finally, we give an example of a creative design in which creative ideas developed by observing a ratchet mechanism are used to design a novel kind of forward-reverse mechanism.

### Prototypes for kinematic pair design

Prototypes for kinematic pair design consist of a qualitative function and a shape feature which implements it. Both must be expressed in a well-defined representation language so that they can be composed by the CAD system. For creative design, these languages must be generative and capable to express any possible function or structure. In our system, we use a hierarchy of representation languages, as shown in Figure 3. In this section, we present the languages we use for

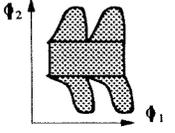
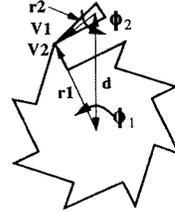
model	example	language
environment of use	$F = (+1 -1)$	qualitative vectors
functional feature	$\forall x \text{ place}(x) \rightarrow \exists y \dots$	quantified logical expressions
place vocabulary		behavior predicates
configuration space		contacts
metric diagram		vertices edges dimensions constraints

Figure 3: Representations used in our CAD system. Polygonal shapes are represented by a metric diagram consisting of vertices, edges and their dimensions. This structure defines a configuration space of the device, which shows the possible contacts between parts. Behavior predicates model qualitative features of the configuration space, and make up a place vocabulary model of the space. Finally, functions are defined as logical expressions involving the environment of  $u$  and the place vocabulary.

modeling qualitative function and shape features.

### Metric diagram representation of shape

Shapes are represented using a *metric diagram*. The metric diagram consists of a symbolic structure which defines vertices, edges and metric parameters for the positions of the vertices. In our current implementation, the metric diagram is restricted to polygons, but can be extended to include circular arcs. A metric diagram represents several objects, each of which has a well-defined degree of freedom.

Using the metric diagram, a *shape feature* (which may involve several objects) is defined by:

- a set of vertices and edges,
- the metric parameters associated with them,
- a set of constraints which must hold simultaneously for the shape feature to be present.

For example, the shape feature which corresponds to the possibility of the top of the ratchets lever being able to touch the wheel (Figure 3) can be expressed as follows:

- must exist: vertices  $v_1, v_2$
- constraints:  $|d - r_1| < r_2$

### A language for modeling qualitative kinematic function

For supporting creative design, it is crucial to be able to model any function that a designer might consider. In this section, we present a generative language capable of representing any kinematic function. *Function* is a property of behavior caused by certain external influences on the device. For example, the function of a ratchet is to block the motion of wheel in one direction when the pawl is forced downwards and to not block it in the other direction. We define a set of *behavior predicates* and a formalism for expressing external influences on mechanisms. Functions are then defined by logical expressions connecting external influences and behavior predicates. The language is similar to languages like CFRL ([Iwasaki *et al.*, 93]), but allows general logical expressions which are required to represent many mechanical functions.

### Representing qualitative kinematic behavior

Textbooks on the subject explain kinematic behavior qualitatively by sequences of *kinematic states*. Examples of kinematic states of a ratchet device are shown in Figure 4.

In qualitative physics terminology, a graph of kinematic states and transitions is called an *envisionment*. It can be computed based on a *place vocabulary*, a graph where each node represents a different combination of contact relationships, and each arc represents a potential transition between them. The envisionment is obtained by combining each node of the place vocabulary with assumed motions and keeping

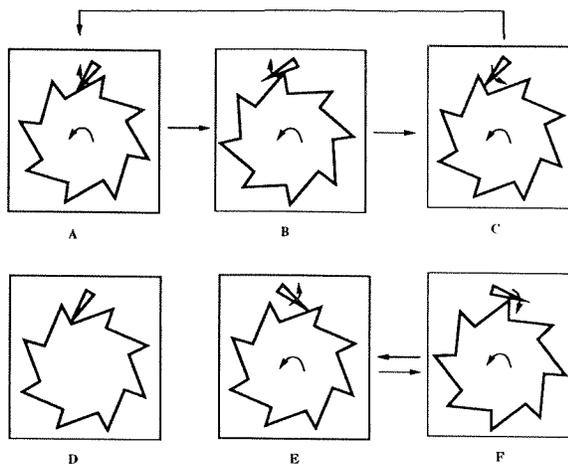


Figure 4: *Examples of kinematic states and transitions in a ratchet.*

only the states and transitions consistent with external forces and motions. We have developed and implemented complete algorithms to compute place vocabularies for arbitrary two-dimensional higher kinematic pairs in fixed-axis mechanisms. These have been used to compute envisionments for a number of mechanisms, such as a mechanical clock ([Forbus *et al.*,91]).

**Behavior predicates** We represent place vocabularies using a set of *behavior predicates* which characterize places, their features and their connectivity. For a kinematic pair, the place vocabulary defines a graph containing three types of kinematic states, corresponding to two, one and no contacts, and identified by the following behavior predicates:

- **point-place(x)**: the contacts in  $x$  hold only in a single configuration.
- **edge-place(x)**: the contacts in  $x$  hold in a one-dimensional set of configurations.
- **face-place(x)**:  $x$  is a place without any contacts and two-degrees of freedom.

For each place, the place vocabulary defines the allowed qualitative directions of motion:

- **qualitative-motion(d)**:  $d$  is a qualitative vector  $(d_0, d_1)$  whose components indicate the direction of motion of each object:  $d_i \in \{-, 0, +\}$ .
- **allowed-motion(x,d)**: motion  $d$  is possible everywhere in place  $x$ .

For each link between states, the place vocabulary defines the directions which can cause a transition:

- **transition(x,y,d)**: motion  $d$  can cause a transition from place  $x$  to  $y$ .

**Qualitative motions** The kinematic states of a mechanism are obtained by combining each place  $x$  with its maximal set of possible qualitative motions:

$$\mathcal{M}_{all}(x) = \{m \mid \text{allowed-motion}(x, m)\}.$$

In an actual behavior, only those motions  $\mathcal{M}(x)$  which in fact *caused* by an external influence actually occur. The set of transitions between states is the set:

$$\mathcal{T}_m = \{ (x, y) \mid (\exists d \in \mathcal{M}(x)) \text{ transition}(x, y, d) \}.$$

More details on envisioning mechanisms using place vocabularies can be found in [Nielsen,88].

**Representing external influence** In kinematic pairs, external influences can be either *forces*, represented by a set  $\mathcal{F}_{ass}$  of qualitative vectors, or *motions*, represented by a set  $\mathcal{M}_{ass}$ , also consisting of qualitative vectors. Since a qualitative force vector causes a qualitative motion in the direction of the same vector, the set of possible motions  $\mathcal{M}_f$  caused by external forces is then given as:

$$\mathcal{M}_f(x) = \{v \mid v \in \mathcal{F}_{ass}\}$$

The actual set of motions  $\mathcal{M}(x)$  to be considered in state  $x$  is then:

$$\mathcal{M}(x) = \mathcal{M}_{ass}(x) \cap \mathcal{M}_f \cap \mathcal{M}_{all}(x)$$

**Formulating functions** Functions are properties of behavior under certain environment. In our system, they are the assumed forces and motions. Therefore, qualitative functions can be defined as *logical conditions* on place vocabularies without first constructing the qualitative behavior. Using logical expressions on the behavior predicates which represent the place vocabulary, a set of functions can be defined as required for the application, and extended whenever required to express a new specification. For example, some functions our current prototype system uses are:

- **transmitting-place**( $x, dir_1, dir_2$ ):  
 $(\forall d = (d_1, d_2))\{d_1 = dir_1 \Rightarrow d_2 = dir_2\} \wedge \{d_2 = -dir_2 \Rightarrow d_1 = -dir_1\}$
- **blocking-place**( $x$ ):  
 $\neg(\exists d \in \mathcal{M}(x))\text{allowed-motion}(x, d)$   
 (a place blocks motions if it does not allow any of the assumed motions).
- **partial-blocking-place**( $x, dirs$ ):  
 $\neg(\exists d \in dirs)\text{allowed-motion}(x, d)$   
 (a partial blocking place blocks the specified motions)
- **possible-path**( $x_0, x_n$ ):  
 $(x_0 = x_n) \vee \exists S \triangleq (x_0, x_1, x_2, \dots, x_n) (\forall i < n)(\exists d \in \mathcal{M}(x_i))\text{transition}(x_i, x_{i+1}, d)$   
 (There is a path from place  $x_0$  to place  $x_n$  whenever there is a sequence of places with transitions between them under at least one assumed motion)

- **cycle**( $x_0, C$ ):  
 $C = (x_0, x_1, x_2, \dots, x_n, x_0) (\forall x_i \in C)(\exists d \in \mathcal{M}(x_i))\text{transition}(x_i, x_{mod(i+1, n+1)}, d)$   
 (there is a cycle of states  $C$  such that transitions between subsequent states are consistent with assumed and allowed directions of motion.)

A place vocabulary can only fulfill the required functions if the number of states and their connectedness is sufficient. Reasoning about such *topological* features is difficult in the place vocabulary itself, since it is based only on individual *boundaries* of shapes which cannot be modified individually. We use an explicit representation of the *kinematic topology* ([Faltings *et al.*,89]) of the mechanism to detect cases where the topology of particular object shapes would not permit the specified function. An example of a function defined on the basis of kinematic topology is:

- **cycle-topology**( $c, d_1, d_2$ ): if the first or second object have rotational freedom, the cycle involves  $d_1$  rotations of the first or  $d_2$  rotations of the second object. This predicate is defined directly on the kinematic topology of the mechanism.

which can be defined formally using similar behavior predicates as those which define place vocabularies.

The function of a ratchet can now be defined qualitatively as follows:

For all starting states  $a$  in which the orientation of the lever is in the interval  $[0..π]$  (pointing to the left such that the moment gravity exerts on it is positive):

- for  $\mathcal{M}_{ass} = \{(+, *)\} \wedge \mathcal{F}_{ass} = \{(*, +)\}$  (the '\*' stands for either +, 0 or -):  
  - **cycle**( $a, C$ )  $\wedge$  **cycle-topology**( $c, 1, 0$ )
  - $\neg(\exists x)\text{blocking-place}(x) \wedge \text{possible-path}(a, x)$   
 (assuming that the wheel turns counterclockwise and the lever is forced onto it, there is a cycle of states where the wheel can rotate, and no reachable blocking state from any starting state a.)
- for  $\mathcal{M}(x) = \{(-, *)\} \wedge \mathcal{F}_{ass} = \{(*, +)\}$ :  
  - $(\forall y)\text{possible-path}(a, y) \Rightarrow \{\neg\text{cycle}(y) \wedge (\exists z)(\text{blocking-state}(z) \wedge \text{possible-path}(y, z))\}$   
 (assuming that the wheel turns clockwise, no reachable state leads to a cycle and all states can eventually lead to a blocking state).

Note that due to the ambiguities inherent in qualitative envisionments, the formalism always overgenerate behaviors. It is therefore only possible to define *necessary*, but never *sufficient* specifications of behavior and, consequently, function. For example, we can express the specification that clockwise motion leads to a blocking state only in an indirect manner: if there is no possibility to cycle, and there is at least one reachable blocking state, the device must eventually reach this state.

## Creating and using new prototypes

**Explaining functions** New prototypes are defined by generalizing a particular device which implements

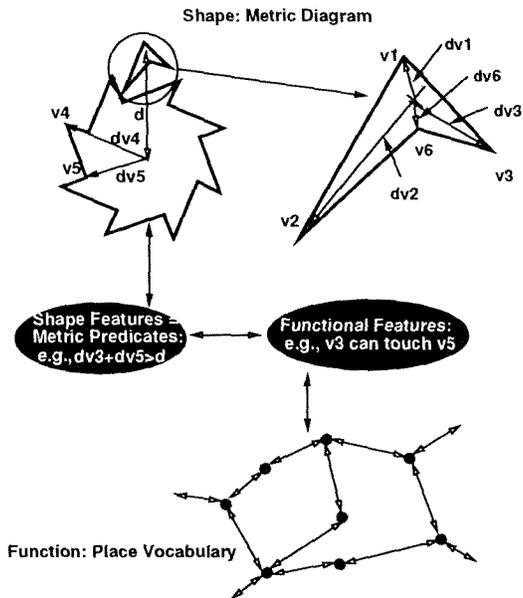


Figure 5: Analysis of a device defines a place vocabulary as a set of features, for example the possibility of touch between the tip of the lever and the bottom of the wheels teeth. Such features define metric predicates such as the one shown.

a novel function. The generalization is based on an explanation of the function in terms of the structure of the device.

A function is a quantified logical expression of behavior predicates defined on the place vocabulary. When the function is implemented in a device, the envisionment defines a set of behavior predicates. Among these predicates, there is at least one logical conjunction which satisfies the quantified condition defining the function:

$$\text{functional feature} \Rightarrow$$

$$\text{behavior-pred}_1 \wedge \text{behavior-pred}_2 \wedge \dots$$

The essence of a creative idea is now formally defined by the choice of a particular conjunction of behavior predicates which satisfy the quantified logical expression defining the function. Note that in general, finding all conjunctive propositions which satisfy a quantified logical expression is a non-computable problem, thus putting creativity beyond the scope of algorithms.

**Defining shape features** Analyzing the behavior of a device using a place vocabulary produces a set of behavior predicates. Each of these predicates is computed based on certain properties of the geometry of the device being analyzed (Figure 5). The analysis which produced the behavior predicates can be generalized to constraints which the shape must satisfy in order for the behavior predicates to remain present.

These constraints, taken together, define a qualitative shape feature which is associated to the functional feature. That is:

$$\text{behavior-pred}_1 \wedge \text{behavior-pred}_2 \wedge \dots \Rightarrow$$

$$\text{constraints on shapes} \Rightarrow \text{shape feature}$$

Reversing the causal chain of the analysis thus establishes a mapping from functional features to shape features, and we call such a process *causal inversion*. More details on the mapping between shape and qualitative behavior can be found in [Faltings,92a].

For any functional feature identified in the place vocabulary of a device, we can thus construct a corresponding shape feature which implements it. These shape features, indexed by the functions, form new prototypes which are added to the system's knowledge base.

### Kinematic pair design with prototypes

A kinematic pair is specified by a conjunction of qualitative functions. Structures which satisfy these functions are obtained by combining prototypes from the systems knowledge base such that all required functions are covered. Combining prototypes means combining shape features using the following steps:

1. choose a unification of vertices and edges of the shape features defined in the prototypes.
2. instantiate the constraints associated with the shape features and find a solution to the resulting constraint network.
3. envision the solution to determine compositional constraints which must also be considered, add them to the constraint network and iterate from step (2).

This process poses two major difficulties: satisfying the dynamic constraint network, and discovering and adding compositional constraints.

**Constraint satisfaction** The constraint network for combining shape features is dynamic and involves many nonlinear constraints. No reliable and efficient method exists for solving such constraint networks. In our current prototype, we use a process of iterative refinement where an initial partial solution, given for example by the device used to define the prototype, is incrementally modified until all constraints are satisfied. The refinement process uses two types of modification operators:

- *dimensional* modifications, where the dimensions of parts are adjusted to fit the functional requirements, and
- *topological* modifications, where vertices are added to part shapes. A topological modification is always coupled with a dimensional modification to fix the dimensions of the new features.

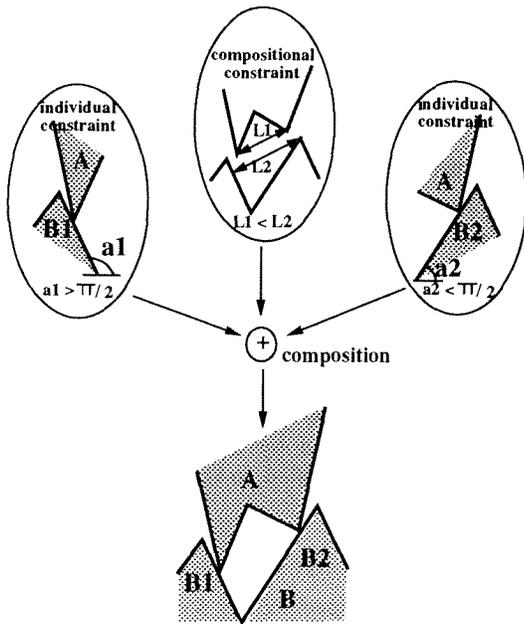


Figure 6: Inferring a compositional constraint from observing the behavior of a combination of shape features.

Dimensional modification varies the values of dimensional parameters, thus changing the appearance of one place and its properties (for example, inference rule). Topological modification are proposed when there is no dimensional modification which can satisfy additional constraints. More details about the computation of dimensional and topological modifications can be found in [Faltings and Sun,93].

**Discovering compositional constraints** As discussed in the introduction, combination of shape features often implies novel interactions which result in additional compositional constraints. In kinematics, the only interactions we have to consider are *subsumptions*, where one shape features makes the contact of another impossible or alters the way it occurs. Compositional constraints which ensure the absence of subsumptions can be formulated most easily once a subsumption has been observed. For example, if we observe the subsumption shown in the introduction (Figure 1), we can infer a novel compositional constraint as shown in Figure 6 by expressing the condition that the subsuming contact may not occur simultaneously with the subsumed contact as an algebraic inequality. Subsumption constraints are relatively simple expressions in the case of translational motion, but can be considerably more complex in the case of kinematic pairs involving rotations.

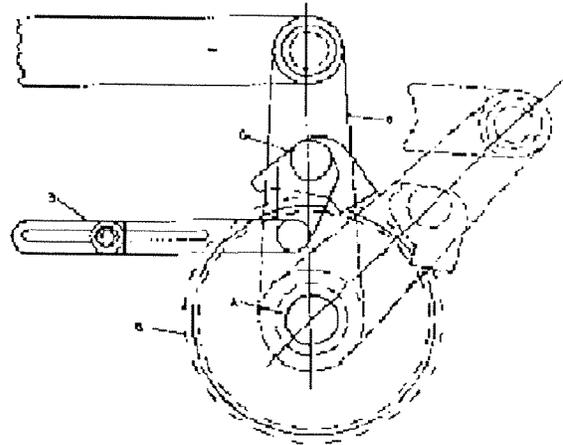


Figure 7: A forward-reverse mechanism as found in a popular mechanism book.

### Creative design of a forward-reverse mechanism by composition

As an example of using composition for creative design, we show how a novel forward-reverse mechanism can be conceived by composing two ratchets.

A forward-reverse mechanism is used to transform an oscillating motion into a rotation which advances in one direction and, after a period of rest, reverses the motion to a lesser degree. A solution for this problem in the literature ([Newell and Horton,67]) is shown in Figure 7. It uses 4 parts and a friction-based mechanism which is problematic for maintenance.

Using the functional features defined earlier, the function of a mechanism with forward-reverse movements can be specified as follows:

1.  $(\exists \text{array } \mathcal{X} = \{x_0, x_1, x_2, \dots, x_{n-1}\})$  of states such that:  
 $(\forall x_i \in \mathcal{X}) \text{transmitting-place}(x_i, (+, -)) \wedge \text{array-topology}(\mathcal{X}, 1, 0)$   
 (There is a array of states which transmits counterclockwise motion of input driver to clockwise motion of wheel.)
2.  $(\exists \text{array } \mathcal{Y} = \{y_0, y_1, y_2, \dots, y_{n-1}\})$  of states such that:  
 $(\forall y_i \in \mathcal{Y}) \text{transmitting-place}(y_i, (-, +)) \wedge \text{array-topology}(\mathcal{Y}, 1, 0)$   
 (There is a array of states which transmits clockwise motion of input driver to counterclockwise motion of wheel.)
3. for  $\mathcal{F}_{ass} = (0, +)$ ,  $\mathcal{M}_{ass} = \{(-, +), (0, +), (+, +)\}$ :  
 $(\forall x_i \in \mathcal{X}) \text{possible-path}(x_i, y_i), y_i \in \mathcal{Y}$   
 (When the input driver changes the motion from counterclockwise to clockwise, there exists a path from place  $x_i$  to  $y_i$ )
4. for  $\mathcal{F}_{ass} = (0, -)$ ,  $\mathcal{M}_{ass} = \{(-, -), (0, -), (+, -)\}$ :

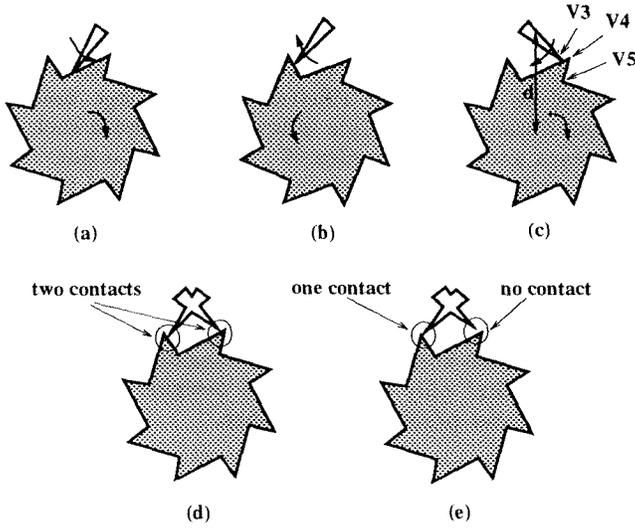


Figure 8: Creative design of forward-reverse mechanism by composing two ratchets. a), known device of start design. a), b) and c), interesting aspects of functional features discovered by envisionment. d), composition of shape features with subsumptions. e), iterative satisfaction of subsumption constraints.

- ( $\forall y_i \in \mathcal{Y}$ ) **possible-path**( $y_i, x_{mod(i+1,n)}, x_{mod(i+1,n)} \in \mathcal{X}$ )  
 (When the input driver changes the motion from clockwise to counterclockwise, there exists a path from place  $y_i$  to the place  $x_{mod(i+1,n)}$  in the array  $\mathcal{X}$ .)
5. ( $\forall x_i \in \mathcal{X}$ ) ( $\forall y_i \in \mathcal{Y}$ ) { **dist**<sub>1</sub>( $y_i, y_{mod(i+1,n)}, (+, *)$ ) < **dist**<sub>1</sub>( $x_i, x_{mod(i+1,n)}, (-, *)$ ) }  
 (The counterclockwise motion angle of wheel in each period is smaller than its clockwise motion angle.)

### Discovering functional features in the envisionment

Assume that the designer has noticed that a ratchet device, when used in the environment of forward-reverse mechanism, can achieve some of the required functions. In an envisionment of the ratchets behavior in the environment of the forward-reverse mechanism, the designer specifies the following correspondences to the functional specifications:

1. there is one array of states, Figure 8 (a), in which the lever can drive the wheel. The counterclockwise motion of the lever turns the wheel clockwise. Therefore, it can be used to satisfy functional specification (1).
2. there is another array of states, Figure 8 (b), in which the lever can also drive the wheel. The clockwise motion of the lever pushes the wheel counterclockwise, which meets the requirement of specification (2).

3. when the lever changes the motion from counterclockwise to clockwise, there is a possible path from one state of array  $\mathcal{X}$  to one state of array  $\mathcal{Y}$  in the same period, which fulfills specification (3).
4. when the lever changes the motion from clockwise to counterclockwise, there are no paths for satisfying specification (4), since from one state of array  $\mathcal{Y}$ , it returns to one state of array  $\mathcal{X}$  in the same period.
5. when the wheel is driven by the lever, clockwise and counterclockwise motion angles of the wheel are always equal, which means that specification (5) cannot be satisfied.

The designer now searches functional features to satisfy specifications (4) and (5). For solving the discrepancy with specification (5), the designer has to add a set of states where motion of the lever results in a clockwise motion of the wheel:

$$(\exists \text{array } \mathcal{Z} = \{z_0, z_1, z_2, \dots, z_{n-1}\}) \\ \forall z_i \{ \mathbf{transmitting-place}(z_i, (+, *)) \} \wedge \\ \mathbf{array-topology}(\mathcal{Z}, 1, 0) \\ \text{(States } z \text{ transmits some motion of the lever to a clockwise motion of the wheel.)}$$

The envisionment of the ratchet in fact contains such an array of states  $z$ , shown in Figure 8 (c). We assume that the designer decides to use this set of states to make it possible to satisfy specification (5). However, (5) refers to the transitions defined in specification (4), which are not yet satisfied in the simple ratchet device. Assume that the designer decides to satisfy specifications (4) and (5) by creating paths passing through intermediate states chosen from the array  $\mathcal{Z}$ . He communicates this to the system by identifying in the envisionment of a ratchet the states  $z \in \mathcal{Z}$  and the transitions which should be connected to states in  $\mathcal{X}$  and  $\mathcal{Y}$ , thus defining the shape features and their relative positions used in the design.

### Defining a new prototype

The new prototype consists of the functional feature, defined by the logical expression given earlier, and the corresponding shape feature which implements it. The shape feature is defined by the explanation underlying the envisionment. For example, the existence of state  $z$  in Figure 8 (c) can be translated to the existence of vertices  $v_3, v_4, v_5$ , the center distance  $d$ , and the following constraints:

$$C1: \sqrt{x_4^2 + y_4^2} - \sqrt{x_3^2 + y_3^2} > 0$$

$$C2: x_3^2 + (y_3 - d)^2 - \frac{(d \times (x_5 - x_4) + x_4 \times (y_5 - y_4) + y_4 \times (x_4 - x_5))^2}{(y_4 - y_5)^2 + (x_5 - x_4)^2} > 0$$

$$C3: (x_3 - x_4) \times (y_5 - y_4) - (y_3 - y_4) \times (x_5 - x_4) > 0$$

$$C4: x_3 \times (x_4 - x_5) + (y_3 - d) \times (y_4 - y_5) > 0$$

## Composing prototypes

The functional features have been mapped into two shape features, each defined as a set of constraints on the metric diagram of a single ratchet device. Furthermore, the identified transitions impose constraints on the relative positions of the shape features in the combined device. For the output member, these can be satisfied by one and the same object, but the input driver has to be a composition of two levers implementing specifications (1)-(3) and specification (4)-(5), respectively. The resulting composed device is shown in Figure 8 (d).

## Satisfying compositional constraints

Not all compositional constraints can be specified before composition, but many are only discovered when the composed device is envisioned. For example, the composed device has subsumptions as illustrated in Figure 8 (d), where tips of the levers touch two teeth of the wheel simultaneously. In this case, the composed levers are blocked from further clockwise movement. Therefore, the paths from state  $y$  to  $z$  in the replaced specification (4) are broken. A new subsumption constraint is added to avoid this behavior, starting a search for a better solution which satisfies all constraints.

We have seen that the constraints describing the state  $z$  and subsumptions involve  $v_3, v_4, v_5$  and  $d$ , which are highly nonlinear. Their satisfaction is very difficult. We attack this problem by an incremental refinement. In this example, assume that the designer chooses to change vertex  $v_3$  to search for a solution. By carrying out a region search, the system changes its position to  $v_3 = (7.58, 24.00)$ . This results in a new device as shown in Figure 8 (e). Renewed envisionment shows that it is in fact a functional forward-reverse mechanism, and the design is finished.

## Conclusions

A main shortcoming of knowledge-based CAD systems is the fact that precoded design knowledge does not allow designers to express their creative ideas. In this paper, we have presented an implemented technique which shows that knowledge-based technology and creativity are not contradictory concepts. *Qualitative physics* provides the extensible representations needed to accommodate creative ideas in a knowledge-based system. Qualitative physics has exactly the functionality required for extending design spaces, as postulated by many researchers in creative design. Using more complete domain models such as developed for IICAD ([Kiriya *et al.*, 92]), the technique is applicable to more general domains than elementary mechanisms. This points to a new and as yet unexploited application of qualitative physics as a tool to support extensible knowledge-based systems.

Our current approach is geared towards *supporting* creative designs, not generating them automati-

cally. By automatically searching the space of possible geometric structures, it would be possible to construct a fully automatic "creative" system. Such search might be made more efficient using a large mechanism library and suitable indexing techniques such as [Sycara and Navinchandra, 90]. However, it is not clear how such a generation process could be guided to only furnish functionalities which are in fact interesting. One way to do this might be to provide such a search with a set of specifications taken from a standard mechanism textbook and ask it to find novel solutions which satisfy them. However, the combinatorial problems associated with such a search are considerable, and we consider that only supporting a designer's creativity has a greater practical importance.

## References

- Boi Faltings, Emmanuel Baechler, Jeff Primus. Reasoning about Kinematic Topology. *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Detroit, 1989
- Boi Faltings. Qualitative Kinematics in Mechanisms. *Artificial Intelligence*, 44(1), 1990
- Boi Faltings. A Symbolic Approach to Qualitative Kinematics. *Artificial Intelligence*, 56(2), 1992
- Boi Faltings. Supporting Creativity in Symbolic Computation. *Second International Round-Table Conference on Computational Models of Creative Design*, Heron Island, Queensland, Australia, 1992
- Boi Faltings, Kun Sun. Computer-aided Creative Mechanism Design. *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, 1993
- Ken Forbus, Paul Nielsen, Boi Faltings. Qualitative Spatial Reasoning, the CLOCK Project. *Artificial Intelligence*, 51(3), 1991
- John Gero. Design Prototypes: A Knowledge Representation Schema for Design. *AI Magazine*, 11(4), 1990
- John Gero. Creativity, emergence and evolution in design. *Second International Round-Table Conference on Computational Models of Creative Design*, Heron Island, Queensland, Australia, 1992
- Y. Iwasaki, R. Fikes, M. Vescovi, B. Chandrasekaran. How things are Intended to Work: Capturing Functional Knowledge in Device Design. *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, 1993
- Leo Joskowicz, S. Addanki. From Kinematics to Shape: An Approach to Innovative Design. *Proceedings of the AAAI*, St. Paul, August 1988
- Takashi Kiriya, Tetsuo Tomiyama, Hiroyuki Yoshikawa. Building a Physical Feature Database for Qualitative

Modeling and Reasoning, *AAAI Fall Symposium, Design from Physical Principles*, Cambridge, MA, 1992

Dorothy Neville, Daniel S. Weld. Innovative Design as Systematic Search, *AAAI Fall Symposium, Design from Physical Principles*, Cambridge, MA, 1992

John Newell, Holbrook Horton. Ingenious Mechanism for Designers and Inventors, Volume IV. Industrial Press INC, New York, 1967

Paul Nielsen. *A Qualitative Approach to Rigid Body Mechanics*. Ph. D. Thesis, University of Illinois, 1988.

P.M. Sargent. A computational view of creative steps. *Second International Round-Table Conference on Computational Models of Creative Design*, Heron Island, Queensland, Australia, 1992

Devika Subramanian. Conceptual Design and Artificial Intelligence. *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambéry, France, 1993, pp. 800-809

Katia Sycara, D. Navinchandra. Index Transformation Techniques for Facilitating Creative Use of Multiple Cases. *Proceedings of the 12th IJCAI*, Sydney, 1991

Brian Williams. Interaction-based Invention: Designing Novel Devices from First Principles, *Proceedings of the National Conference on Artificial Intelligence*, Boston, 1990

# Qualitative Reasoning of a Temporally Hierarchical System Based on Infinitesimal Analysis

Hiroshi Tanaka and Shusaku Tsumoto  
Department of Informational Medicine  
Medical Research Institute, Tokyo Medical and Dental University  
1-5-45 Yushima, Bunkyo-ku Tokyo 113 Japan  
Email: {tanaka,tsumoto}@tmd.ac.jp

## Abstract

In ordinary qualitative reasoning (QR), qualitative behavior of the dynamical systems is predicted by assignment of qualitative values such as  $\{+,0,-\}$  into model variables based on proper transition rules. Unfortunately, due to ambiguities in these ordinarily introduced qualitative values, their arithmetic and transition rules cause predictions to be redundant, sometimes even inaccurate. In this paper, we present a new method of qualitative reasoning which, besides using hyperreal numbers, takes into account their  $\varepsilon$ - $\mathbf{H}$  ranking in describing both qualitative values and qualitative derivatives of variables and also employs a convergence filter to investigate the infinitesimal asymptotic behavior of qualitative variables. We applied this qualitative reasoning method to envision a temporally hierarchical complex system. The result shows that this method provides a more detailed and natural qualitative solution than previous methods like Kuipers's time abstraction in envisioning temporally hierarchical complex system.

## 1 Introduction

Limitations of ordinary qualitative reasoning (QR) using the  $\{+,0,-\}$  semantics have been discussed in many works [1]-[7],[11]-[17]. One of the major limitations is that scale information such as the relative magnitude of quantities or their temporal derivatives are not included so that further ambiguity arises in determining state transition among all the possible adjacent values of a current qualitative state. To compensate this too abstract qualitative representation, several researchers introduced the concept of order of magnitude ( $O(M)$ ) proposed by Raiman [11] and the hyperreal numbers to extend the definition of qualitative value [2],[16]. Davis's CHEPACHET [2] and Weld's HR-QSIM [16], though quite different in their motivations and aims, can be considered as examples of this extension from ordinary QR.

We believe, however, that the true virtue of introducing this kind of scale information lies in envisioning a complex system in which several scale hierarchies are

involved. Introducing this scale information can approximately isolate interactions of subparts having different magnitudes or time-scales, which, in ordinary QR, must be taken into account together in equal levels. This means that by using scale information we can introduce some kind of hierarchization in envisionment to prune insignificant transitions. Though infinitesimal analysis is not explicitly used, Kuipers's "abstraction by time-scale" method [6] can be considered to lie along this line. He treats the complex system as composed of different time-scales. On account of the "extra-mathematical" nature of his method, however, this intuitively appealing method has several problems when its range of application is extended.

In this paper we developed a new method to deal with temporal hierarchization by introducing infinitesimal analysis [5] to realize what Kuipers was trying to do in a more formal and natural way. But in doing so, simple introduction of hyperreal number like in [2],[16] has proved to be insufficient. For this reason we developed a new QR scheme which can handle infinitesimal qualitative behavior in a right way. Our QR method features the following two newly introduced concepts: First, infinitesimal/infinite numbers are ranked ( $\varepsilon$ - $\mathbf{H}$  ranking) to be able to evaluate their relative magnitudes. Second a convergence filter is introduced in order to investigate more precisely how variables converge towards their equilibrium.

The paper is organized as follows: in Section 2, we discuss Kuipers's time-scale abstraction and its problems. Section 3 shows extensions of the qualitative values and transition rules for our new scheme. Section 4 presents our temporal hierarchization algorithm for envisionment. Section 5 gives an example which illustrates envisionment by our temporal hierarchization applied to the same problem in [6]. Finally, in Section 6 we discuss about related work.

We assume that the reader is familiar with standard theories of envisionment, as in [1],[7], and those with qualitative hyperreals proposed by Davis [2] and Weld [16]. In this paper,  $[x]$  denotes the qualitative value of a variable  $x$ .  $\partial x$  and  $\partial^2 x$  stand for the qualitative deriva-

tive of  $x$ , and the second order of derivative of  $x$ , respectively. And  $x_i$  denotes the variable in a state  $i$ , for example,  $x_1$  is the variable in a state 1. We also use the symbols,  $\varepsilon, N$ , and  $\mathbf{H}$ , which represent infinitesimals, finite numbers, and infinite numbers respectively, in such a way that, when distance between a variable  $x$  and a landmark  $x_0$  is infinitesimal:  $[x - x_0] = \varepsilon$ , finite:  $[x - x_0] = N$ , or infinite:  $[x - x_0] = \mathbf{H}$ .

## 2 Kuipers's Hierarchization and its Problems

Kuipers [6] deals with a complex system such as a collection of interacting equilibrium submechanisms. He gives as an example the body fluid regulation and describes the whole system in terms of two mechanisms, water and sodium balances, that operate at different response time. The basic principle of this time-scale abstraction is "A faster mechanism reaches its equilibrium instantaneously and, during this process, a slower mechanism can be treated as being constant". His approach consists of the following steps: 1) Decompose a whole system into faster system and slower system and model each one separately, but with some sharing variables. 2) Faster to slower: envision first the faster system. Viewed from a faster system, slower variables are treated as constants (relative constancy of slower variables). 3) Slower to faster: from the point of view of the slower system, the faster system instantaneously reaches its equilibrium with its environment composed of the slower system. Hence faster variables move quasi-statically along with the equilibrium conditions determined by slower variables when the slower system changes (binding faster variables as a function of slower variables). 3) The behavior of the whole system is temporally joined in cascade from faster to slower (continuation).

The essence of his approach is decomposition of a model from the temporal point of view. This hierarchization reflects our naive reasoning in envisioning behaviors of such a complicated system; however, this approach has several problems when extended to be applied to more general cases: 1) Since his continuation is based on "cascade shift of attention", which handles mechanism by switching between submodels, he assumes that all the submodels are stable and reach their equilibrium in pre-determined subsequent order. If the faster mechanism does not converge (for example, oscillates with negligible magnitude), we cannot use this kind of decomposition. 2) Kuipers decomposes the structure of a system completely, so that conjunction after such decomposition may lose some important information about the interaction between faster variables and slower variables which is involved in the original system. 3) When faster variables converge to equilibrium values, their derivatives must approach the same infinitesimal order of magnitude as those of the slower variable derivatives. His ab-

straction assumes that even in that stage faster variables change more quickly.

The problems are mainly caused by the too strong nature of the decomposition of a system. Since the difference in velocities between variables can be described using the  $O(M)$  of those derivatives, it is expected that the introduction of the  $O(M)$  for the qualitative derivatives gives us a more sophisticated and natural way to use the information of difference of the variable changing rate without any decomposition of the structure.

## 3 QR with Ranked Hyperreals

### 3.1 Extensions of Qualitative Values

We extend a qualitative hyperreal representation described by Weld [16] and Davis [2] in order to describe the infinitesimal/infinite behaviors of qualitative variables more exactly. As Davis discusses in [2], one of the problems of envisionment using qualitative hyperreals is that once a parameter and its derivative both become infinitesimal or both become infinite, it becomes impossible to say anything about their relative sizes. To solve this problem, we further divide infinitesimal and infinite interval into  $\varepsilon, \varepsilon^2, \varepsilon^3, \dots$  and  $\mathbf{H}, \mathbf{H}^2, \mathbf{H}^3, \dots$ , respectively, when these high-order numbers are required to be evaluated in the course of envisionment, for example, when the convergence speeds of variables are required to be evaluated.

Note that in evaluating the interval's length we take its maximal width. Thus, whereas  $\varepsilon$  means  $\varepsilon$ -neighborhood interval around some landmark, it is simultaneously representing its interval's length so that  $\varepsilon$  can be treated also as a number in infinitesimal calculations in envisionment. Hence we can write  $\varepsilon > \varepsilon^2 > \varepsilon^3 > \dots$ . On the other hand,  $\mathbf{H}$  also means the interval greater than the hyperreal infinite number  $\mathbf{H}$  where the latter is thought as a hyperreal number. Hence we can write  $\mathbf{H} < \mathbf{H}^2 < \mathbf{H}^3 < \dots$ . We call this division of infinitesimal/infinite interval " $\varepsilon$ - $\mathbf{H}$  ranking."

This ranking is illustrated in fig.1 and fig.2. Let  $l_0$  be a certain landmark of a parameter. Fig.1 shows the former relationship between  $\varepsilon^i$  and  $\varepsilon^{i+1}$  for any integer  $i$ . In this figure, ellipse denotes the neighbor of  $l_0$  whose radius is given by  $\varepsilon$ ,  $N$ , and  $\mathbf{H}$ . Region inside the ellipse whose radius is equal to  $\varepsilon$ , and  $N$  represents the infinitesimal neighborhood of  $l_0$ , and the "finite-distance" neighborhood of  $l_0$ , respectively. The outside of this "finite-distance" neighborhood shows the infinite neighborhood of  $l_0$ .

If we zoom up the infinitesimal neighborhood, then we can define the similar structure in the ellipse by introducing power of  $\varepsilon$ . Based on this characterization, relationship between  $\varepsilon^2$  and  $\varepsilon$  can be defined as being similar to that between  $\varepsilon$  and  $N$ .

In the similar way, in general, relationship between

$\varepsilon^{i+1}$  and  $\varepsilon^i$  can be defined as being similar to that between  $\varepsilon$  and  $N$ . Also, we can define relation between  $\mathbf{H}^{i+1}$  and  $\mathbf{H}^i$  for any integer  $i$ , as shown in fig.2.

This "ε-H ranking." is formulated as:

**Definition 1 (Qualitative value with ε-H ranking)**

Let  $l_0 < l_1 < l_2 < \dots l_i (= 0) \dots < l_n (< \mathbf{H})$  be the landmark values of a parameter  $x$ . Define the hyperreal qualitative value of  $x$  as:

$$[x] = \left\{ \begin{array}{ll} \mathbf{H}[\mathbf{H}^2, \dots], & \text{if } x - \mathbf{H}[\mathbf{H}^2 \dots] \leq \mathbf{H}[\mathbf{H}^2 \dots] \\ N(= \langle l_n, \mathbf{H} \rangle), & \text{if } x - l_n > \varepsilon \text{ and } x < \mathbf{H} \\ l_n + \varepsilon[\varepsilon^2, \dots], & \text{if } 0 < x - l_n \leq \varepsilon(\varepsilon^2, \dots) \\ l_n, & \text{if } x = l_n \\ l_n - \varepsilon[\varepsilon^2, \dots], & \text{if } 0 < l_n - x \leq \varepsilon(\varepsilon^2, \dots) \\ \langle l_{n-1}, l_n \rangle, & \text{if } x - l_{n-1} > \varepsilon \text{ and } \\ & l_n - x > \varepsilon \\ \dots, & \\ l_0 + \varepsilon[\varepsilon^2, \dots], & \text{if } 0 < x - l_0 \leq \varepsilon[\varepsilon^2, \dots] \\ l_0, & \text{if } x = l_0 \\ l_0 - \varepsilon[\varepsilon^2, \dots], & \text{if } 0 < l_0 - x \leq \varepsilon[\varepsilon^2, \dots] \\ -N(= \langle -\mathbf{H}, l_0 \rangle), & \text{if } l_0 - x > \varepsilon \text{ and } \\ & x > -\mathbf{H} \\ -\mathbf{H}, [-\mathbf{H}^2, \dots], & \text{if } -\mathbf{H}[\mathbf{H}^2 \dots] - x \\ & < -\mathbf{H}[\mathbf{H}^2 \dots] \end{array} \right.$$

where

denotes the possible alternation and  $\langle p1, p2 \rangle$  is equivalent to the difference between open interval  $(p1, p2)$  and the two halos as in Weld's HR-QSIM. The quantity space of a variable  $x, QS(x)$  is defined as:

$$QS(x) = \{ (\dots, \mathbf{H}^2), \mathbf{H}, N, \\ l_n + \varepsilon(\varepsilon^2, \dots), l_n, l_n - \varepsilon(\varepsilon^2, \dots), \\ \langle l_{n-1}, l_n \rangle, \dots, \langle l_0, l_1 \rangle, \\ l_0 + \varepsilon(\varepsilon^2, \dots), l_0, l_0 - \varepsilon(\varepsilon^2, \dots), \\ -N, -\mathbf{H}, (-\mathbf{H}^2, \dots) \}$$

□

**Definition 2 (O(M) and Sign of Qualitative Value)**

Define the order of magnitude (O(M)) of a variable  $x$  as:

$$abs(x) = \left\{ \begin{array}{ll} \mathbf{H}^i, & \text{where } x \text{ is } \mathbf{H}^i\text{-infinite} \\ N(> 0), & \text{where } x \text{ is finite} \\ \varepsilon^i, & \text{where } x \text{ is } \varepsilon^i\text{-infinitesimal} \\ 0, & \text{where } x = 0 \end{array} \right.$$

where  $i$  is integer. □

Using D2 with the sign of a variable  $x \text{ sign}(x)$ , we can define the qualitative derivative(QD) of a variable. We also introduce the qualitative second-order derivatives of variables to "tame intractable branching" [8] where it is appropriate to evaluate.

**Definition 3 (Qualitative Derivative)** For  $i = 1, 2$ , define the qualitative derivative of a variable  $x$  as:

$$\partial^i x = \text{sign}\left(\frac{d^i x}{dt^i}\right) * \text{abs}\left(\frac{d^i x}{dt^i}\right).$$

□

**Definition 4 (Qualitative Representation)**

Define the qualitative representation of a variable state  $x, Q_s R(x)$  as:

$$Q_s R(x) = ([x], \partial x, \partial^2 x).$$

□

Consider an example where qualitative differential equation(QDE) is  $\partial x = -\varepsilon * [x]$  and its initial condition is  $[x] = \varepsilon$ . From QDE, we obtain  $\partial^2 x = -\varepsilon * \partial x$ . Hence the initial state:  $Q_s R(x_1) = (+\varepsilon, -\varepsilon^2, +\varepsilon^3)$  is obtained through constraint propagation. Since  $\varepsilon^2$  and  $\varepsilon^3$  appear in  $\partial x$ ,  $QS(x) = \{\mathbf{H}, N, \varepsilon, \varepsilon^2, \varepsilon^3, 0, -\varepsilon^3, -\varepsilon^2, -\varepsilon, -N, -\mathbf{H}\}$  and the next candidate is the transition from  $[x] = \varepsilon$  to  $[x] = \varepsilon^2$ . The next state  $Q_s R(x_2) = (+\varepsilon^2, -\varepsilon^3, +\varepsilon^4)$  is derived from the above constraints.

**Definition 5 (Qualitative Arithmetic)** Arithmetic between qualitative infinitesimal values is based on the O(M) reasoning proposed by Raiman[11]. We extend this arithmetic as follows: For any integers  $m$  and  $n$  such that  $m < n$ ,

- (Addition)  $\varepsilon^m + \varepsilon^n \simeq \varepsilon^m$
- (Substitution)  $\varepsilon^m - \varepsilon^n \simeq \varepsilon^m$
- (Multiplication)  $\varepsilon^m * \varepsilon = \varepsilon^{m+1}$
- (Division)  $\varepsilon^{-m} = \mathbf{H}^m$
- (Comparison)  $(\varepsilon/\varepsilon) < (1/\varepsilon) < (1/\varepsilon^2) < \dots < (1/\varepsilon^m) < \dots$

where  $A \simeq B$  means  $abs(A) = abs(B)$ . □

Note that  $\varepsilon/\varepsilon$  is finite. Intuitively, this relation is derived by an inequality  $\varepsilon^2/\varepsilon(= \varepsilon) < \varepsilon/\varepsilon < 1/\varepsilon(= \mathbf{H})$ . The proof can be found in Keisler[5].

**3.2 Transition with ε-H ranking**

Besides extensions of qualitative values, we extend definition of state interval, transition between qualitative values, and persistent and arrival time. The ε-H ranking adds many interesting characteristics to QR with hyperreals. But for limitations of space, this presentation is restricted to the extensions of transition and persistent and arrival time from Weld's HR-QSIM[16]. For more detail, see [15].

**Definition 6 (Transition of Qualitative Values)**

Qualitative values can have transitions only between two adjacent states, or inside a ε-neighborhood interval ( $\varepsilon^i \leftrightarrow \varepsilon^{i+1}$ ) or inside a infinite interval ( $\mathbf{H}^i \leftrightarrow \mathbf{H}^{i+1}$ ). The possible transitions are as follows:

Table 1: time-distance table

		distance			
		0	$\varepsilon$	N	H
$\partial x$	0	0	H	H	H
	$\varepsilon$	0	N	H	H <sup>2</sup>
	N	0	$\varepsilon$	N	H
	H	0	$\varepsilon^2$	$\varepsilon$	N

$$(\dots \leftrightarrow \mathbf{H}^2 \leftrightarrow) \mathbf{H} \leftrightarrow \text{finite} \leftrightarrow \varepsilon (\leftrightarrow \varepsilon^2 \leftrightarrow \dots) \leftrightarrow \text{point}$$

The steps which determine the transition of qualitative values are derived by envisionment using Welds' HR-QSIM method [16] except for inside  $\varepsilon$  or H interval.  $\square$

**Definition 7 (Arrival time and persistent time)**

For any variable  $x$  and any integer  $i$ ,  $I_s(x_i)$  be the  $O(M)$  of the interval's length of the state  $i$ , and  $I_a(x_{i+1})$  be the minimum  $O(M)$  of the distance of  $x$  between the present state( $i$ ) and the next one( $i+1$ ). Persistent time( $t_s(x_i)$ ) and arrival time( $t_a(x_{i+1})$ ) are represented as the following equations:

$$t_s(x_i) = I_s(x_i) / \text{abs}(\partial x_i)$$

$$t_a(x_{i+1}) = I_a(x_{i+1}) / \text{abs}(\partial x_i)$$

Their values are derived by qualitative calculus with  $\varepsilon$ -H ranking as shown in Section 3.1 and a time-distance table as shown in Table 1.  $\square$

For example, when  $I_a(x) = \varepsilon$  and  $\text{abs}(\partial x) = \varepsilon^3$ ,  $t_a(x) = (\varepsilon / \varepsilon^3) = \varepsilon^{-2} = \mathbf{H}^2$

Note that our  $\varepsilon$ -H ranking improves Weld's temporal filter[16]. For example, we can deal with  $\varepsilon$ -ordering rule[1] more concretely, which Weld a little bit trickily includes in "Temporal Continuity Rule". Consider an example where a variable  $x$  is 0, and the order of its derivative is  $\varepsilon^m$  ( $m$ :a certain integer). The next deduced transition is  $[x] = \varepsilon^m$ . Since  $I_a(x)$  is the minimum  $O(M)$  of the distance of  $x$  between 0 and  $\varepsilon^m$ , for any integer  $i$ ,  $I_a(x) < I_a(\varepsilon^{i+m}) < I_a(\varepsilon^m)$ . Hence we obtain that  $t_a(x) = (I_a(x) / \text{abs}(\partial x)) < (I_a(\varepsilon^{i+m}) / \text{abs}(\partial x)) = (\varepsilon^{i+m} / \varepsilon^m) = \varepsilon^i$ . Consequently,  $t_a(x) < \varepsilon^i$ , that is, arrival time from  $x=0$  to  $\varepsilon^m$  is less than  $\varepsilon^i$ . Since  $i$  is arbitrary, this means that a variable in a state changes faster than any other variables in an infinitesimal interval.

Then, how we can deal with the reverse case, that is, the transition of  $x$  from  $\varepsilon$  to 0 ( $x$  converges at 0)? We can classify the convergence into two types:  $x$  passes 0 after converging to 0 within a finite or an infinitesimal interval and  $x$  converges at 0 asymptotically. In the next section, we discuss about this case.

### 3.3 Convergence

When one variable converges monotonically, it would happen that envisionment is repeated infinitely. For

example, consider the case mentioned in Section 3.1. Envisionment generates the following infinite sequence:  $(x, \partial x, \partial^2 x) = (+\varepsilon, -\varepsilon^2, +\varepsilon^3), (+\varepsilon^2, -\varepsilon^3, +\varepsilon^4), (+\varepsilon^3, -\varepsilon^4, +\varepsilon^5) \dots$ . However, in this case, it is obvious that  $(x, \partial x, \partial^2 x)$  converges at  $(0, 0, 0)$ . Our formalism can examine more precisely how a variable converge, (for the above example,  $\partial x, \partial^2 x$  decreases as time passes), but it cannot judge the convergence. Hence, in addition to the extensions of QR discussed above, we must provide a rule which judges whether and how the convergence of variables occurs (In this section, for simplicity, we only deal with "monotonic convergence". However, our definitions can be easily generalized in order to deal with damped oscillation. Generalization of a convergence filter is discussed in [15].)

**Definition 8 (Convergence Filter)** Let  $x_0$  be the nearest landmark. Also let  $I_s(x_i)$  denote the length of an interval( $i$ :integer). If

$$\exists i, \quad I_s(x_i) = \varepsilon$$

$\forall k$  such that  $k > i, \exists m$

$$([x_k - x_0] = -\varepsilon^m, \partial x_k > 0) \vee ([x_k - x_0] = +\varepsilon^m, \partial x_k < 0)$$

$$\text{and if } I_s(x_{k+1}) \leq \varepsilon * I_s(x_k)$$

then we shall say that  $x$  converges at  $x_0$ .

And the arrival time  $t_a$  is defined as:

$$t_a = \sum_{j=i+1}^{\mathbf{H}} t_a(x_j).$$

$\square$

This arrival time is calculated by the ordinary methods for infinite sum in nonstandard analysis[5] and it has several important features. Unfortunately, for limitation of space, we cannot give a detailed discussion about the arrival time here. In this paper, we only present two characteristics without their proof: **if  $\text{abs}(t_a) \leq N$  then  $x$  passes  $x_0$  and if  $\text{abs}(t_a) \geq \mathbf{H}$  then  $x$  converges at  $x_0$  asymptotically.** Precise discussion is given in [15].

The above definition is clear, but insufficient to determine the convergence with finite steps. It does not reduce the infiniteness of the determining process. Note that we should judge the convergence in finite time: when we observe that some elements of the sequence of a variable  $x$  approach at a point  $x_0$ , we determine that  $x$  converges at  $x_0$ . This reasoning process is an example of "persistence" in nonmonotonic reasoning[9],[10]. According to our commonsense reasoning, we provide a convergence filter rule as follows:

**Definition 9 (Convergence Filter Rule)** If the states satisfy the condition D8 (convergent condition) until  $\varepsilon^4$  occurs, then check the next transition. And if the condition D8 is also satisfied, the system judges

that convergence has occurred. Arrival time is calculated as  $t_a = \sum_{j=i+1}^{\mathbf{H}} t_a(x_j)$  where  $i$  is a certain integer such that  $I_s(x_i) = \varepsilon^3$ . If  $\text{abs}(t_a) < N$  then  $x$  passes  $x_0$ , else if  $\text{abs}(t_a) \geq \mathbf{H}$  then  $x$  converges at  $x_0$  asymptotically.  $\square$

Consider the example mentioned above.  $I_s(x) = \varepsilon, \varepsilon^2, \varepsilon^3$  and the candidate for the next state is  $(x, \partial x, \partial^2 x) = (\varepsilon^4, -\varepsilon^5, \varepsilon^6)$ . Since the former states and the candidate satisfy the conditions D8, we check the next transition. The next is  $(\varepsilon^5, -\varepsilon^6, \varepsilon^7)$  and the condition is also satisfied. So we determine that  $x$  (and  $\partial x$ ) converges at 0. And the arrival time is derived as follows:  $t_a = (\varepsilon/\varepsilon^2) + (\varepsilon^2/\varepsilon^3) + \dots = \sum_{j=2}^{\mathbf{H}} (\varepsilon^{i-1}/\varepsilon^i) = (1/\varepsilon) * \mathbf{H} \simeq \mathbf{H}^2 > \mathbf{H}$ . Hence a variable  $x$  converges at 0 asymptotically.

## 4 QD Restriction

Time-scale abstraction is considered to be based on the two properties of the  $O(M)$  of qualitative derivatives. First, relative constancy of slower variables means that the  $O(M)$  of the derivatives of slower variables and their change are much less than the derivatives of faster variables. Second, binding of faster variables means that  $O(M)$  of the derivatives of faster variables is much larger than slower ones. These temporal ontologies of variables can be more sophisticatedly represented by specifying range of derivatives; we call this hierarchy of qualitative derivatives QD restriction.

**Definition 10 (QD Restriction)** *Quantity space of qualitative derivatives of faster variables(f) or slower ones(s) should be represented as follows:*

$$\begin{aligned} QS(\partial f) &= \{.., +\mathbf{H}^2, +\mathbf{H}, +N, \\ &\quad +\varepsilon, +\varepsilon^2, .., 0, .., -\varepsilon^2, -\varepsilon, \\ &\quad -N, -\mathbf{H}, -\mathbf{H}^2, ..\} \\ QS(\partial s) &= \{+\varepsilon, +\varepsilon^2, .., 0, .., -\varepsilon^2, -\varepsilon\} \\ QS(\partial^2 s) &= \{+\varepsilon, +\varepsilon^2, .., 0, .., -\varepsilon^2, -\varepsilon\}. \end{aligned}$$

*Direction of transition of qualitative derivative is determined by the signs of the second order derivatives of the qualitative values and precedence of transition is determined by the arrival time.*  $\square$

Note that the second order derivatives of the slower variables are also restricted. If their order of magnitude is  $N$  (finite), the derivatives transit into a finite interval. This contradicts the above definition. For example, consider  $\partial x = \varepsilon$  and  $\partial^2 x = N$ . In order for  $\partial x$  to stay at  $\varepsilon$ , the order of persistent time for  $\partial^2 x = N$  is lower than  $(N - \varepsilon)/N = N/N$ , that is, its order is equal to or lower than  $\varepsilon$ . If we consider that the third order derivatives meet the above requirements,  $\partial^3 x$  will be  $\mathbf{H}$ . Hence, this fact contradicts the definition of slower variables: slower

variables change much slower than faster variables in a finite interval. Therefore the second order derivatives are also required to be restricted.

One may say that QD restriction can be naturally embedded in QDE as shown in Davis[2], such that  $\partial x = -\varepsilon^i * [x]$  where  $x$  is a slower variable and  $i$  is integer. However, this embedding is not sufficient. When  $x$  is  $\mathbf{H}^i$ ,  $\partial x$  is  $-N$ . If the  $O(M)$  of the QD of a faster variable is  $N$ , then we cannot differentiate between a faster variable and slower one. Hence even in the above case, QD restriction is also needed.

## 5 QUASAR

We develop a program QUASAR (QUALitative reasoning using time-Scale information Analysis by epsilon-eta ranking and Restriction of qualitative derivatives) which implements QR with ranked hyperreals, the convergence filter and QD restriction. QUASAR consists of two parts: setting part, and transition analyzer. Setting part calculates the constraints of QD from given QDE and then QD restriction is set up. Finally, it derives the initial states from an initial condition. Transition analyzer envisions a transition from a certain state  $i$ . In this section, first we show algorithm for transition analysis, and then illustrate how QUASAR works.

### 5.1 Algorithm for Transition

In this algorithm, we use a operator " := " for substitution, for example, " $x := 3$ " means that 3 is substituted for  $x$ . For simplicity, we assume that there is no branching before the state  $i$ .

#### Algorithm

Let  $i, j, k, l, m, n$  and  $p$  be integer.  $x_i(m)$  stands for a variable of the system ( $1 \leq m \leq n$ ,  $n$ : the total number of the variables in QDE) in a state  $i$ . And also let  $first_i, min_i$  and  $final_i$  denote the set of first candidates, minimum candidates, and final candidates respectively as defined below. Before transition analysis, all the sets are  $\{\}$  (empty).

1. Apply the transition rules to each variable:  $x_i(j)$ , and generate the list of the nearest qualitative adjacent value of each variable considered as the candidates of the next state transition:  $first_{i+1} = \{(1 : \hat{x}_{i+1}(1)), \dots, (n : \hat{x}_{i+1}(n))\}$ .
2. Calculate each  $t_a(\hat{x}_{i+1}(k))$  ( $k$ : integer,  $1 \leq k \leq n$ ). Compare the  $O(M)$  of  $t_a(\hat{x}_{i+1}(k))$ , and choose the set of candidates whose  $O(M)$  of the arrival time is minimum: (minimum candidates:  $min_{i+1} = \{(j : \hat{x}_{i+1}(j)) \mid \forall l, \text{abs}(\hat{x}_{i+1}(j)) \leq \text{abs}(\hat{x}_{i+1}(l))\}$ ).

3. Choose a variable, say  $x_{i+1}(m)$ , from  $min_{i+1}$ :  
 $min_{i+1} := min_{i+1} - \{(m : x_{i+1}(m))\}$ . Substitute  $x_{i+1}(m)$  for  $x_i(m)$  in the state  $i$  and apply the constraint propagation. If the constraints are satisfied, add this variable to the set of the final candidates( $final_{i+1} := final_{i+1} \cup \{(m : x_{i+1}(m))\}$ ).
4. If  $min_{i+1} \neq \{\}$ , go to 3). If  $min_{i+1} = \{\} \wedge final_{i+1} \neq \{\}$ , go to 5). If  $min_{i+1} = \{\} \wedge final_{i+1} = \{\}$ , quit as failure.
5. Choose a variable, say  $x_{i+1}(p)$  from  $final_{i+1}$ .(  
 $final_{i+1} := final_{i+1} - \{p : x_{i+1}(p)\}$ ). Apply the convergence filter rule to  $x_{i+1}(p)$ :
  - (a) If the past four sequences of  $x(p)$  satisfy the convergent condition, then check the next transition.
  - (b) If the convergent condition is also satisfied, then determine that  $x(p)$  converges at a point  $x_0(x_{i+1}(p) := x_0)$ .
  - (c) If  $x(p)$  converges, calculate the arrival time( $t_a$ ). If  $abs(t_a) \leq N$  then  $x(p)$  passes at the point, else  $x(p)$  converge at the point asymptotically.
  - (d) If  $x(p)$  passes, store this state as the next state else store this state as the final state.
6. If  $final_{i+1} \neq \{\}$  then go to 5).  
If  $final_{i+1} = \{\}$  then quit as succeeded.

## 5.2 An Example

Let us consider a model of body fluid regulation. Body fluid system is regulated chiefly by amount of water( $w$ ), sodium( $n$ ) and concentration of sodium( $c = n/w$ ) which is almost equal to osmotic pressure. It is known that the amount of body fluid change(water intake and excretion) is regulated by sensing osmotic pressure deviation, and this regulation takes place with response time 10 minutes, whereas change of the amount of sodium is regulated by the amount of water with response time more than one hour. This system can be modeled as follows:

$$\begin{aligned} \partial w &= [c - c_0] \\ \partial n &= \varepsilon * [w_0 - w] \\ c &= \frac{n}{w} \end{aligned}$$

where  $w$  is a faster variable,  $n$  is a slower variable. And  $n_0, w_0, c_0 (= n_0/w_0)$  are quantities of sodium, water, and concentration of sodium at steady state, respectively. We consider the case when osmotic pressure is hypertonic( $c > c_0$ ), and both water and salt are overloading( $w > w_0, n > n_0$ ) to show how QUASAR works.

### 5.2.1 Setting Part

From the qualitative derivative equations, we calculate the constraints about derivatives and second order derivatives of faster and slower variables as follows:

$$\begin{aligned} \partial c &= \frac{\partial n - c \partial w}{w} \\ \partial^2 w &= \partial c \\ \partial^2 n &= \varepsilon * (-\partial w) \\ \partial^2 c &= \partial^2 n - 2\partial n \partial w - \partial^2 w + (\partial w)^2. \end{aligned}$$

Using these formulae, QD restriction are used to obtain into the quantity space of qualitative derivatives:

$$\begin{aligned} QS(\partial w) = QS(\partial^2 w) &= \{\dots, +\mathbf{H}^2, +\mathbf{H}, +N, \\ &+ \varepsilon, +\varepsilon^2, \dots, 0, \dots, -\varepsilon^2, -\varepsilon, \\ &-N, -\mathbf{H}, -\mathbf{H}^2, \dots\}, \\ QS(\partial n) = QS(\partial^2 n) &= \{+\varepsilon, +\varepsilon^2, \dots, 0, \dots, -\varepsilon^2, -\varepsilon\}, \\ QS(\partial c) = QS(\partial^2 c) &= \{\dots, +\mathbf{H}^2, +\mathbf{H}, +N, \\ &+ \varepsilon, +\varepsilon^2, \dots, 0, \dots, -\varepsilon^2, -\varepsilon, \\ &-N, -\mathbf{H}, -\mathbf{H}^2, \dots\}. \end{aligned}$$

After the above settings, the given initial conditions are propagated: State1 is an interval, where  $Q_s R((w - w_0)_1) = (+N, N, -N)$ ,  $Q_s R((n - n_0)_1) = (+N, -\varepsilon, -\varepsilon)$ ,  $Q_s R((c - c_0)_1) = (+N, -N, -N)$ . From this initial state, the transition process begins.

### 5.2.2 Transition

Using the algorithm mentioned in 4.3, we can derive the results of qualitative analysis as shown in Table 2. Here, we give one example: transition from state4 to state5 to illustrate how the results are obtained.

#### (Transition from state4 to state5)

1. The candidates are  $[w - w_0] : +N \rightarrow \mathbf{H}$ ,  $[n - n_0] : +N \rightarrow +\varepsilon$ ,  $[c - c_0] : +\varepsilon^3 \rightarrow \varepsilon^4$ : thus  $first_5 = \{(w - w_0)_5 : \mathbf{H}\}, \{(n - n_0)_5 : +\varepsilon\}, \{(c - c_0)_5 : +\varepsilon^4\}$ .
2. Calculate each arrival time:  $w : (\mathbf{H} - N)/N \simeq \mathbf{H}$ ,  $n : N/\varepsilon \simeq \mathbf{H}$ ,  $c : (\varepsilon^3 - \varepsilon^4)/\varepsilon \simeq \varepsilon^2$ ,  $c$  are the final candidates:  $min_5 = \{((c - c_0)_5 : +\varepsilon^4)\}$ .
3. Choose a variable  $(c - c_0)_5$  (then  $min_5 := \{\}$ ). Perform constraint propagation and the solution:  $Q_s R((w - w_0)_5) = (N, +\varepsilon^4, -\varepsilon)$ ,  $Q_s R((n - n_0)_5) = (N, -\varepsilon, -\varepsilon^5)$ ,  $Q_s R((c - c_0)_5) = (+\varepsilon^4, -\varepsilon, +\varepsilon)$  are derived. Add it to the final candidates:  $final_5 = \{(c - c_0)_5\}$ .
4.  $min_5 = \{\}$  and  $final_5 = \{(c - c_0)_5\}$ , so go to 5).
5. Choose  $(c - c_0)_5$  (then  $final_5 = \{\}$ ). It satisfy the convergent condition: that is to say, for a variable  $c$  and for integer  $i = 2, 3$ , and 4,  $Ia((c - c_0)_2) =$

$\varepsilon$ ,  $Ia((c - c_0)_{i+1}) \leq \varepsilon * Ia((c - c_0)_i)$ ,  $[c - c_0] = +\varepsilon^{i-1}$ , and  $\partial c < 0$ . Check the next transition. the next state is  $([c - c_0], \partial c, \partial^2 c) = (+\varepsilon^5, -\varepsilon, +\varepsilon)$ , and the arrival time is  $\varepsilon^3$ . So we determines that  $[c - c_0]$  converges at 0 and their arrival time is  $t_a = \sum_{j=2}^{\mathbf{H}} \varepsilon^j \simeq \varepsilon^2$ . Since  $t_a < N$ ,  $c$  passes  $c_0$ : the state is stored as the state 5.

6.  $final_5 = \{ \}$ , so quit as succeeded.

### 5.2.3 The Results

The results of qualitative analysis show that three kinds of the processes are involved. First, the amount of water( $w$ ) increases fast and the concentration of sodium( $c$ ) converges at the point( $c_0$ ). Second, when  $c$  reaches  $c_0$ ,  $w$  stops increasing. Third,  $c$  passes  $c_0$ , and  $w$  begins to decrease slowly. The variable  $c$  remains to be in the neighborhood of  $c_0$  only to give the driving force to adjust remained water imbalance after osmotic pressure is regulated.  $w$  and  $n$  decreases and, as infinite time passes,  $w, n$  and  $c$  reach their equilibrium. First and second process correspond to faster mechanisms in Kuipers's time-scale abstraction, and third process to slower mechanisms. But our results explain the interaction between faster variables( $w$ ) and slower ones( $n$ ) more clearly: while  $w$  changes quickly when  $c - c_0 > \varepsilon$ ,  $w$  changes slowly with  $n$  when  $c \simeq c_0$ . Those behaviors clearly agrees with the body fluid regulation: if the osmotic pressure changes a little, this change is compensated mainly by renal function - slower mechanism. In Kuipers's time-scale abstraction, however, if " $c \simeq c_0$ ", one constraint: " $c = n/w = c_0$ " should be given for qualitative simulation; interconnection must be always given from outside in order to simulate only the slower mechanism. Our method can cope with that case correctly.

## 6 Discussion

We combine qualitative representation based on Weld's qualitative hyperreals with envisionment based on Davis' CHEPACHET[2] and introduce  $\varepsilon$ - $\mathbf{H}$ ranking and the convergence filter with QD restriction specially for application of our method to a temporal hierarchical system.  $\varepsilon$ - $\mathbf{H}$  ranking and the convergence filter give more precise information of qualitative variables to ordinary QR. Providing some important knowledge of real numbers and the  $O(M)$  of QD makes the envisionment more accurate and reduces the ambiguities in qualitative values. Also it can represent interaction between faster and slower variables more naturally than "extra-mathematical" hierarchization, especially when derivatives of faster variables converge on the  $O(M)$  of the derivatives of slower one and extends his approach. Hence, our approach realizes Kuipers time-scale abstraction in a more mathematical way and extends his approach.

The limitation of this work is that this work will be computationally expensive when faster and slower variables are not well-defined. In other words, since QD restriction may not be applied in that case, so transition of qualitative derivatives is not restricted as in definition 10. QUASAR cannot detect whether a given model support QD restriction or not. To implement these automated detection is our future work.

## 7 Related Works

Little previous attention has been devoted to time-scale abstraction, except for Kuipers work [6]. In this section, we consider AI work related to temporal hierarchization.

Weld [16] extends qualitative values to qualitative hyperreal numbers, and develops a program that considers a role of one parameter in a system by comparing a normal system behavior with the exaggerated system behavior. He discusses about Kuipers approach and describes that his exaggeration can represent time-scale abstraction implicitly, whereas he does not discuss the methodology in detail. One may say that our QD restriction can be regarded as exaggeration of slower variables: we use the nonstandard analysis, and also introduce time-scale into the quantity space of derivatives. However, QD restriction is different from exaggeration. As shown in Section 5, the derivative of faster variables reaches the same order of those of slower variables. And in those states the derivative of slower variables is not exaggerated any more. Hence the whole behavior can be interpreted as combination of exaggerated behavior and not-exaggerated one. Original exaggeration method needs the continuation analysis to deal with time-scale abstraction, which generates the problems discussed in Section 2. So our approach includes exaggeration about time-scale and solve the problems of the continuation analysis of exaggeration.

Davis [2] combines order of magnitude reasoning with envisionment of qualitative differential equations. He divides the non-standard real line into seven disjoint intervals: -LARGE (infinite numbers), -MEDIUM (finite numbers), -SMALL (infinitesimals), ZERO, SMALL, MEDIUM, LARGE. He introduces variance of parameter, which is equal to our  $I_s(x)$ , and time duration, which is equal to persistent time. He illustrates quickly settling control parameter and observes that this example is similar to those studied by Kuipers. Our approach is also similar to his work. However, he does not discuss the cases when derivatives of faster variables converge on the order of derivatives of slower one. In those cases, the interaction between faster and slower variables necessarily appears. So, we cannot envision both of them separately. Our QUASAR can cope with this problem and simulate those cases much finer.

Iwasaki [4] discusses about the mixture of slower sys-

tem and faster system in a viewpoint from causal ordering. She regards a mixed structure  $M$  as combination of equilibrium equations  $Static(M)$  which represent a very short-term equilibrium description, and dynamic equations  $Dynamic(M)$  which represent slower mechanisms. Her approach also uses pre-decomposition of the model and deals with both systems independently. Like the approaches mentioned above, she does not discuss the interaction between faster and slower variables and the problems about the continuation analysis.

Finally, note that our framework can deal with hierarchization of variables' magnitude, such as a system which includes a subsystem of infinitesimal sustained oscillation. Detailed treatment of this kind of system is our future work.

## Acknowledgements

The authors wish to thank Prof. Peter Szolovits and Jon Doyle. Substantial parts of this work were completed by insightful discussions with them when one of the author, Tanaka was working as the visiting scientist at M.I.T. Laboratory for Computer Science.

## References

- [1] de Kleer, J., and Brown, J.S. A Qualitative Physics Based on Confluences, *Artificial Intelligence* **24**, 7-83, 1984.
- [2] Davis, E. Order of Magnitude Reasoning in Qualitative Differential Equations, in *Weld, D.S. and de Kleer, J. (eds) Readings in Qualitative Reasoning about physical systems*, 422-434, Morgan-Kaufmann, 1989.
- [3] Forbus, K.D. Qualitative Process theory. *Artificial Intelligence* **24**:85-168, 1984.
- [4] Iwasaki, Y. Causal Ordering in a Mixed Structure, in *Proc. of AAAI-88*, 313-318, 1988.
- [5] Keisler, H.J. *Foundations of Infinitesimal Calculus* Prindle, Weber and Schmidt, Boston Mass., 1976.
- [6] Kuipers, B. Abstraction by Time-Scale in Qualitative Simulation, *Proc. of AAAI-87*, 1987.
- [7] Kuipers, B. Qualitative Simulation, *Artificial Intelligence* **29**, 289-338, 1986.
- [8] Kuipers, B., Chiu, C. Taming Intractable Branching in Qualitative Simulation, in *Proc. of IJCAI-87*, 1079-85, 1987.
- [9] McDermott, D.V. A Temporal Logic for Reasoning About Processes and Plans, *Cognitive Science* **6**, 101-155, 1982.
- [10] McDermott, D.V., Doyle, J. Nonmonotonic Logic I, *Artificial Intelligence* **13**, 41-72, 1980.
- [11] Raiman, O. Order of Magnitude Reasoning, in *Proc. of AAAI-86*, 1986.
- [12] Sacks, E. Automatic Qualitative Analysis of Dynamic Systems Using Piecewise Linear Approximations, *Artificial Intelligence* **41**, 313-364, 1990.
- [13] Struss, P. Global Filters for Qualitative Behaviors, in *Proc. of AAAI-88*, 1988.
- [14] Struss, P. Problems of Interval-Based Qualitative Reasoning, in *Weld, D.S. and de Kleer, J. (eds) Readings in Qualitative Reasoning*, 288-305, Morgan Kaufmann, 1989.
- [15] Tanaka, H. and Tsumoto, S. Temporal Hierarchization in Qualitative Reasoning, MIT AI-memo (to appear), 1994.
- [16] Weld, D. Theories of Comparative Analysis, M.I.T. press, Cambridge, 1990.
- [17] Williams, B.C. Doing Time: Putting Qualitative Reasoning on Firmer Ground, in *Proc. of AAAI-86*, 105-112, 1986.

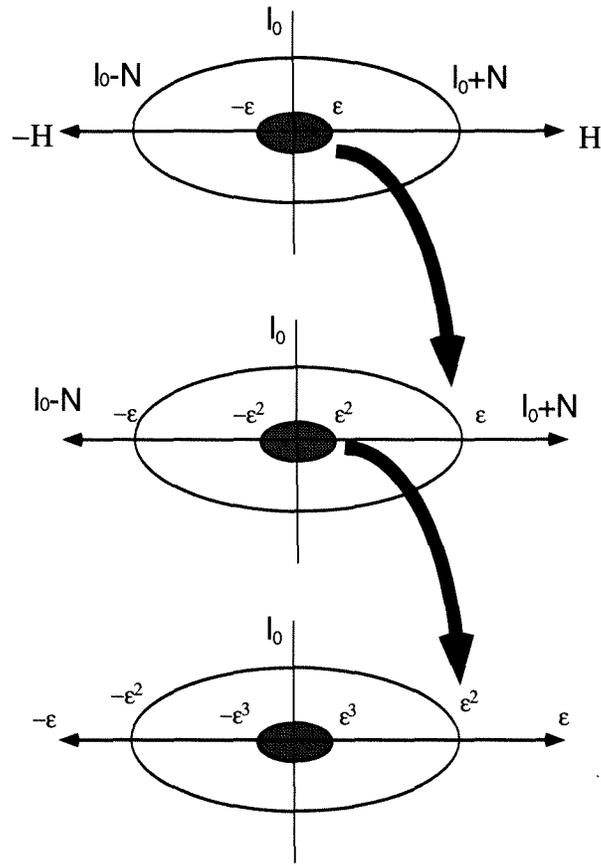


Figure 1: Relationship between the powers of  $\epsilon$

Table 2: Transition of an Example

state	1	2	3	4	(5)	6	7	8	9	10	(11)
$[w - w_0]$	N	N	N	N	N	N	N	N	$\epsilon$	$\epsilon^2$	0
$[n - n_0]$	N	N	N	N	N	N	N	N	$\epsilon$	$\epsilon^2$	0
$[c - c_0]$	N	$\epsilon$	$\epsilon^2$	$\epsilon^3$	0	$-\epsilon$	$-\epsilon$	$-\epsilon$	$-\epsilon^2$	$-\epsilon^3$	0
$\partial w$	N	$\epsilon$	$\epsilon^2$	$\epsilon^3$	0	$-\epsilon$	$-\epsilon$	$-\epsilon$	$-\epsilon^2$	$-\epsilon^3$	0
$\partial n$	$-\epsilon$	$-\epsilon$	$-\epsilon$	$-\epsilon$	$-\epsilon$	$-\epsilon$	$-\epsilon$	$-\epsilon$	$-\epsilon^2$	$-\epsilon^3$	0
$\partial c$	$-N$	$-\epsilon$	$-\epsilon$	$-\epsilon$	$-\epsilon$	$-\epsilon$	0	$\epsilon^2$	$\epsilon^2$	$\epsilon^3$	0
$\partial^2 w$	$-N$	$-\epsilon$	$-\epsilon$	$-\epsilon$	$-\epsilon$	$-\epsilon$	0	$\epsilon^2$	$\epsilon^2$	$\epsilon^3$	0
$\partial^2 n$	$-\epsilon$	$-\epsilon^2$	$-\epsilon^3$	$-\epsilon^4$	0	$\epsilon^2$	$\epsilon^2$	$\epsilon^2$	$\epsilon^3$	$\epsilon^4$	0
$\partial^2 c$	N	$\epsilon$	$\epsilon$	$\epsilon$	$\epsilon$	$\epsilon$	$\epsilon^2$	$\epsilon^2$	$-\epsilon^2$	$-\epsilon^3$	0
$t_a$		N	N	$\epsilon$	$\epsilon^2$	N	N	N	<b>H</b>	<b>H</b>	<b>H<sup>2</sup></b>

$$t_a \text{ for state4} \rightarrow \text{state5} \quad t_a = \epsilon^2 + \epsilon^3 + \dots = \epsilon * (1/(1 - \epsilon)) = \epsilon^2$$

$$t_a \text{ for state10} \rightarrow \text{state11} \quad t_a = 1/\epsilon + 1/\epsilon + \dots = \mathbf{H}/\epsilon = \mathbf{H}^2$$

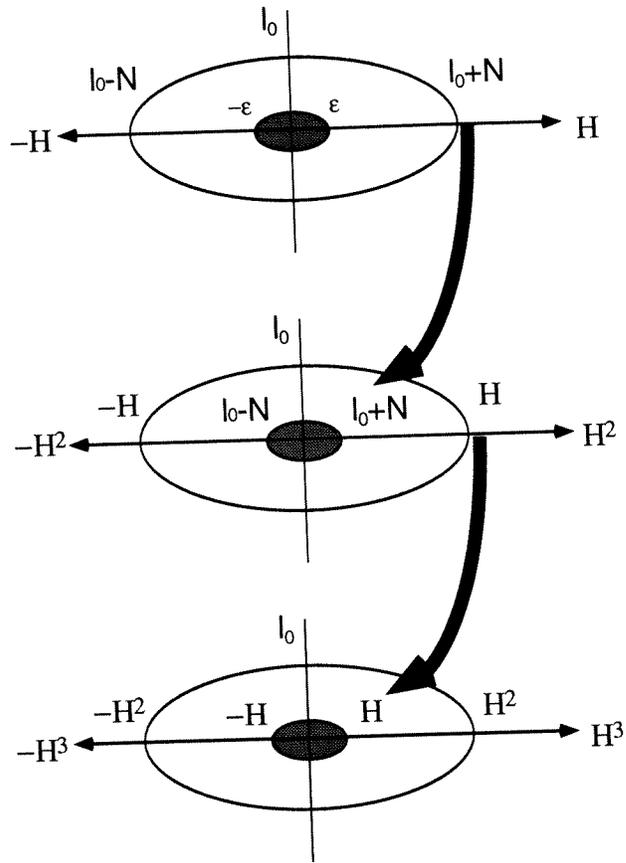


Figure 2: Relationship between the powers of  $H$

# Constructing Functional Models of a Device from its Structural Description

Sunil Thadani      B. Chandrasekaran  
Laboratory for Artificial Intelligence Research  
Department of Computer and Information Science  
The Ohio State University  
Columbus, Ohio 43210-1277  
thadani/chandra@cis.ohio-state.edu

**Topic:** Representation and reasoning

**Domain:** Electrical circuits

**Contribution:** This paper identifies a useful and ubiquitous version of structure-to-behavior reasoning problem and proposes a novel solution approach to it.

## Abstract

This paper addresses, and proposes a solution for, a version of the device understanding problem. Given the structural description of the device, the system generates hypotheses about its functions and how it achieves them. These function-specific device models (FR models) are constructed at multiple levels of abstraction. The proposed method uses the knowledge of frequently encountered abstract devices in the domain to derive functions and FR models from structure. Using a decomposition-based strategy a device is viewed as consisting of a combination of the instances of the already known abstract devices whose functions and FRs are composed to derive the functions and FRs of the whole device.

## 1 The problem

The importance of the problem of understanding how devices work cannot be overstated. Given an unfamiliar device one has to first go through the process of understanding the working of the device, that is, what the device does and how it does that, before one can perform tasks such as diagnosing the device, predicting the behavior of the device, and explaining the working of the device. In this paper we are interested in the process of understanding itself, that is, given the structural description, henceforth referred to as *struc-desc*, of a device, how an agent acquires the understanding of the working of the device. More specifically, by “understanding

a device” we mean determining the functions of the device and for each function generating a causal account of how the device function arises from the functions of its components.

## 2 Situating this work

The motivations to address this problem in the field of AI have been quite varied ranging from building more flexible and robust expert systems to understanding the cognitive aspects of common sense reasoning. The version of the problem addressed by most of the earlier work in AI in this area [dK85, For84, Kui86] was that given the *struc-desc* of the device and given an input perturbation, determine the resulting behaviors of the device. And some [dK85, IS86] also addressed the problem of generating an account of how specific behaviors of the device come about from its *struc-desc*. Even though the various approaches proposed by these various researchers differed along certain dimensions, they all shared the following characteristics. *Device models were composed out of context-independent models of the parts. Simulation was proposed as the primary method of going from structure to behavior. The device models and the behavior descriptions generated were all at a single level of abstraction. Further, the device models and the behavior descriptions generated were general purpose, in the sense that they were independent of the specific aspects of device behaviors that one might be interested in.* Such an approach to reasoning about devices is useful for, say, discovering the behavioral implications of a new design. But there are several scenarios involving reasoning about devices in which device models with the above characteristics are not adequate for the purpose [VIFC93, Cha94, CGI93, IC92, IFVC93].

For any complex device, most tasks require one to reason about the device at multiple levels of abstraction. For example, to diagnose a complex device, one first rea-

sons about the device at a high level of abstraction to localize the malfunctioning portions, and then reason at a more detailed level with the malfunctioning portions only to further narrow down the fault. Device models at multiple levels of abstraction improve efficiency of such tasks. Further, there are a lot of scenarios in which one is interested only in specific behaviors of the device. In such cases there is no need to simulate the general purpose device model generating several irrelevant device behaviors, because for most complex devices simulating such general purpose models is computationally very expensive. To efficiently generate only the behaviors of interest, we need to be able to generate function-specific device models. And finally, for most complex devices component-level behavior descriptions at a single level of abstraction are usually quite large and do not lend themselves to any efficient strategy for organizing the behavior descriptions of a large number of devices in memory.

Depending on the specific task and the specific domain, different researchers have focused on different aspects of the issues mentioned above. For example, [KC87, Str88] use information about higher order derivatives to eliminate the generation of some spurious behaviors during simulation. [ACP91, NJA91] have proposed a mechanism for using context specific component models. [FF91] builds device models at the right level of abstraction appropriate for answering specific questions about the behavior of the device. [Wel86] proposes an abstraction technique for repeating behaviors. Even though various versions of the issues mentioned in the previous paragraph have been addressed by these researchers, most of them assume the context that makes specific commitments, indicated in italics earlier in this section, to what constitutes understanding of a device.

What distinguishes our work on device understanding is the set of specific commitments made about what constitutes understanding of a device. There are at least three parts to the problem of understanding devices. The first one is determining what constitutes understanding of a device. That is, determining the form and content of the knowledge that constitutes understanding. The second part of the problem is to determine how this understanding can actually be generated starting from the *struc-desc* of the device. The third part is to determine how to use this understanding for various tasks mentioned earlier [IC92, IFVC93, CGI93, All90, VIFC93]. Naturally, the commitments made for the first part will determine the characteristics of the approach used to do the second and third parts.

For the first part we have used the proposal on functional representation (FR) [Cha94, IC92, SC88] that makes clear commitments to the content and the form of the knowledge that constitutes understanding of a

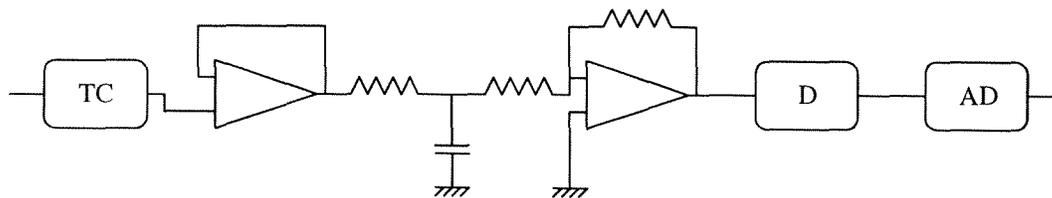
device, *independent of the specific task or the specific domain*. These commitments have been motivated by larger concerns about how the understanding of a large number of complex devices may be organized in memory and how this understanding may be used to efficiently perform various tasks mentioned earlier. Since the distinguishing characteristics of the output expected, the FR models, provide both the motivation as well as the justification for some of the characteristics of the approach proposed in this paper, a clear understanding of the commitments made in FR would help understand this work better.

### 3 Characteristics of the output

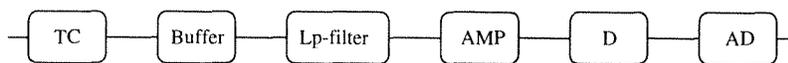
One of the important ideas stated in the proposal on FR is that the causal understanding of a device consists of a set of function-specific causal models of the device, also referred to as the FR models of the device, each of which may be at multiple levels of device-component abstraction.

Consider the circuit, shown in Figure 1(a), of a temperature measuring device. Even though the circuit is given in terms of components such as op-amps, resistors, and capacitors, an expert describing how this circuit works uses several abstractions, such as buffer, low-pass-filter, amplifier, and instrumentation-amplifier. An expert's description of how this device works would be similar to the one that follows. The device shown in Figure 1(a) is understood in terms of the abstract *struc-desc* shown in Figure 1(b) and 1(c). In the *struc-desc* in Figure 1(c) the thermocouple, *TC*, produces a voltage proportional to the input temperature. The instrumentation-amp, *IAMP*, filters and amplifies the voltage generated by *TC*. The analog display, *AD*, produces a deflection proportional to the voltage generated by the *IAMP*. The function of the instrumentation-amp in turn is explained as follows. The input is buffered. The output of the buffer, *B*, is filtered by the *lp-filter* and then amplified by the amplifier *AMP*. The output of the amplifier is further amplified by the driver, *D*, to generate the driver voltage. The functions of the amplifier, buffer, and low-pass filter are in turn explained in terms of the functions of the op-amp, resistors, and capacitors. Such a description of the function of the temperature measuring device is at multiple levels of abstractions and it bridges the descriptions at different levels.

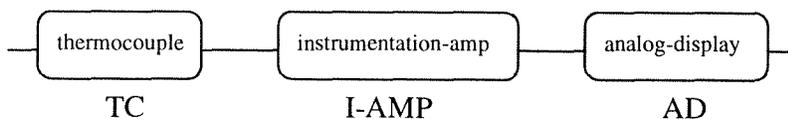
Further, this description is a function-specific description of the temperature measuring device. That is, even though there are several other aspects of this device, only the ones that are relevant to the temperature-measurement function, *tmp-meas-func*, of this device are used in this description. Such function-specific models



(a) Temperature-measuring-device circuit



(b) First level abstraction of temperature-measuring-device circuit



(c) Second level abstraction of temperature-measuring-device circuit

Figure 1: Structural descriptions of temperature measuring device

enable highly focused and computationally efficient reasoning in tasks that use these models.

The description of the temperature measuring device given above is an English-language version of the FR model of the *tmp-meas-func* function of the device shown in Figure 1(a). According to the commitments made in [SC88, Cha94] this is the kind of description an agent is expected to have if the agent claims to understand the *tmp-meas-func* function of the device shown in Figure 1(a). This is also the kind of description we expect our system to generate as an output.

#### 4 Definitions

Some of the terms used here have been used differently by different researchers. For the sake of clarity we give our definitions of these terms here. A *component* is defined by a set of ports and parameters. A component connects and interacts with other components through its *ports*. *Parameters* of a component is a set of parameter-name and parameter-value pairs. For example, a component called *rechargeable-battery* is defined by two ports, the positive and the negative terminals, and parameter names like *emf*, *internal resistance*, and *charge-capacity* each of which has a value associated with it. Associated with a component are *state variables*. The state of a component is defined by the values of its state variables. A *partial state* of a component is a wff of pred-

icates on the state variables of the component. Also associated with a component may be functions. A *function* of a component relates the dependent state variables to the independent state variables of the component. *Independent state variables* of a component are the state variables whose values are assigned or changed by an agent external to the component and *dependent state variables* are the state variables whose values depend on the values of independent state variables.

For each function the component also has a condition, called *provided*. A function of the component is applicable only if the corresponding *provided* condition evaluates to true. These conditions may check anything ranging from the values of certain component parameters to some global variables used to capture the agent's interests. For example, for a rechargeable battery in a closed circuit the charge level of the battery has to be above a certain threshold value,  $Q_P$ , for the *deliver-voltage* function of the battery to be applicable. So the *provided* condition for *deliver-voltage* function of the rechargeable battery would be "actual-charge  $>$   $Q_T$ ."

Components may be connected at the ports to form new components. *Struc-desc* of a component specifies the set of sub-components and the relationships between the sub-components. Relationships between the components are defined in terms of the connections between their ports. In the domain of electrical circuits, one

of the relations between the components is *electrically-connected* that corresponds to an electrical connection between *electrical* ports.

Associated with a specific function of a component may be a functional representation (FR), which contains, among other things, a description of how the function arises from the functions of its subcomponents. This description is called causal process description (cpd). A cpd is a directed acyclic graph [IC92, Tha93, Cha94] with nodes corresponding to the partial states of the components and directed edges, also called links, implying transition from one partial state to another partial state. Links of type *using-func-link* are annotated by the function of the component that causes the state transition, and the links of type *as-per-link* are annotated by the name of the domain law that explains the state transition. The *using-func-link* annotation is what enables the FR to relate cpds at various levels of abstractions. For a complete description of FR see [Cha94, IC92, SC88].

Components are organized in component class hierarchies. A component class may inherit ports, parameters, functions, and FRs from its parent class.

## 5 Getting to the Solution

To describe our approach we shall start backwards from the characteristics of the output expected, and identify the kinds of knowledge required to generate such an output. We shall then propose the form in which these various knowledge types can be acquired and used to perform the task. This is followed by a high level description of the algorithm.

For most non-trivial devices the FR models would have multiple levels of abstraction. FR models use both, structural and behavioral abstractions. For example, component *AMP* in Figure 1(b) abstracts the *struc-desc* in box *A* in Figure 1(a) by hiding the structural details. And the function *amplify-func* associated with the *AMP* abstracts the voltage level description given by the cpd in Figure 6(e), thus hiding the details of the cpd and also introducing a new term “*amplify-func*” for the input-output behavior described by the corresponding cpd. Since these abstractions are not given as part of the input *struc-desc*, the system needs knowledge to build structural and behavioral abstractions to build an FR like the one shown in Figure 6.

The other important characteristic of the output FR models is that these models are function-specific models of the device. To build such a model, the system has to select relevant aspects of the behaviors (subset of state variables and appropriate relations between them) for each abstract and primitive component, aspects that are relevant in describing the specific function of the device.

For example, even though there are several state variables and parameters associated with the *struc-desc* of the component *amplifier*, in the context of the function *amplify-func* only a small subset of state variables are of interest. For each device function the system needs knowledge to get appropriate relations between the relevant subset of state variables.

We represent the above mentioned knowledge types using structure-function-FR (SFF) templates. Each SFF template represents the understanding of an abstract device in the domain. For example, an expert working with electronic circuits already understands several abstract devices, such as comparator, integrator, adder, amplifier, and lp-filter. So a system reasoning about electronic circuits would be provided with SFFs corresponding to each of these abstract devices. The SFF captures the understanding of an abstract device by associating the functions and FR templates of the device with the abstract *struc-desc* of the device. The abstract *struc-desc* represents a class of *struc-desc*'s. It is defined just like the *struc-desc*, except that the sub-components are specified only by their classes and it may also define a constraint on the values of the sub-component parameters. The corresponding function and FR templates also refer to the parameter names and functions of the sub-components defined in the abstract *struc-desc*. A function (or FR) template also defines a class of functions (or FRs). As described below, the SFF can be used to select specific state variables and relations for specific functions and can also be used to build structural and behavioral abstractions for the abstract device the SFF models.

The system has access to a large number of SFFs that correspond to the abstract devices an expert working in that domain already understands. The primary method used by the system to go from a *struc-desc* to its functions and FRs is by using SFFs. First the system tries to identify the set of SFFs such that the given *struc-desc* is an instance of the abstract *struc-desc* of the SFF. The functions of the matching SFFs are hypothesized to be the functions of the given *struc-desc*. Additional methods are used to verify which functions are indeed applicable to the given *struc-desc*. The templates of the verified functions and the corresponding FRs are instantiated to get the functions and FRs for the given *struc-desc*. This process is shown in Figure 2(a).

For example, the part of the circuit in box *A* in Figure 1(a) matches the abstract *struc-desc* of the SFF for the amplifier. The function templates associated with the amplifier SFF, *amplify-func* and *clipped-amplify-func*, are hypothesized to be the possible functions for the *struc-desc* in box *A*. The system then verifies which of the hypothesized functions are actually applicable for the given structural description in the given context. The

system, as an output, then returns an instance,  $A'$ , of the abstract device *amplifier* with its abstract *struc-desc* instantiated to the *struc-desc* in box  $A$ . And instantiations of the verified and selected function and FR templates are also associated with the instance  $A'$ .

Obviously the system cannot be provided with an SFF for every structural description that the system would encounter. So there are going to be *struc-desc*'s for which the system does not have any matching SFFs. To be able to handle a large number of devices the system has to have some sort of a compositional method that enables the system to identify a given *struc-desc* as an instance of a combination of SFFs. The way our system achieves this is by decomposing the given *struc-desc* into parts, analyzing<sup>1</sup> each part separately using the method described in the previous paragraph, and then combining the analyzed parts to form a new *struc-desc* which can be analyzed again. This process is diagrammatically shown in Figure 2(b).

For example, after analyzing the *struc-desc* in box  $A$ , the part of the circuit in box  $A$  can be replaced by an instance,  $AMP$ , of the abstract device *amplifier*, with the function *amplifier-func* associated with the instance  $AMP$ . Similar things are done with other parts of the given circuit. This process results in a new *struc-desc* shown in Figure 1(b) consisting of abstract devices like  $AMP$ . The decomposition strategy can also be seen as a "structural description transformation" technique, because it takes in a *struc-desc* and generates a new *struc-desc*.

The above process is repeated on the new *struc-desc* generating more abstractions resulting in hierarchical FR models. The *struc-desc* in Figure 1(b) is further abstracted to the structural description in Figure 1(c), which is analyzed by matching it to the *measuring-instrument* SFF. The functions and FRs for the *struc-desc* in Figure 1(c) are obtained by verifying and instantiating the functions and FR templates associated with the *measuring-instrument* SFF.

The two strategies described above constitute the core of the algorithm used by our system to perform the task. In several domains, given the right set of SFFs, a large class of devices can be analyzed by recursive application of these two strategies alone. The method based on the above two strategies is complementary to the simulation based techniques for going from structure to behavior. There are at least two places where simulation can play a role. One is in function verification. Right now we use purely structural criteria to verify if a hypothesized function is in fact a function of the given *struc-desc*.

<sup>1</sup>We use the phrase "analyzing the device" to refer to the process of understanding the device, that is, generating its functions and FRs

A simulation-based function verification may be used where it may not be possible to verify function based on structural criteria alone. Another place simulation may be used is to determine the behavior of those parts of the given *struc-desc* for which the above method does not work. So if some parts of the given *struc-desc* cannot be analyzed using SFFs, one can use simulation locally to determine their behaviors.

Additional algorithms are used to achieve the following:

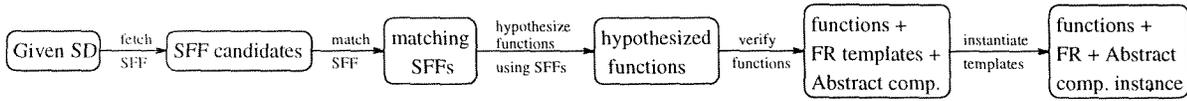
- To fetch SFF candidates that would match the given *struc-desc*. To make the search efficient we have organized the SFFs based on functions, the number of sub-components, and special sub-components.
- To match the given *struc-desc* to the abstract structural description of an SFF.
- To verify and select functions that are applicable to the given structural description and that are relevant in the given context.
- To decompose the given *struc-desc*. Various heuristics are used to control the decomposition. For example, the knowledge of the hypothesized function of the given *struc-desc* may be used to suggest decompositions.

For more details on the algorithm see [Tha94]. As we will show in the following section the resulting control in this algorithm may be top-down or bottom-up depending on the knowledge available. The heuristics used for various steps keep the complexity of the algorithm linear in most cases [Tha94]. Assuming the component models and SFFs to be correct, and assuming the *provided* conditions associated with each function in the SFF correctly verify the hypothesized functions, the algorithm will always produce sound results. Since our decomposition heuristics do not try all the possible decompositions of the given device, the algorithm is not complete with respect to the knowledge provided to it.

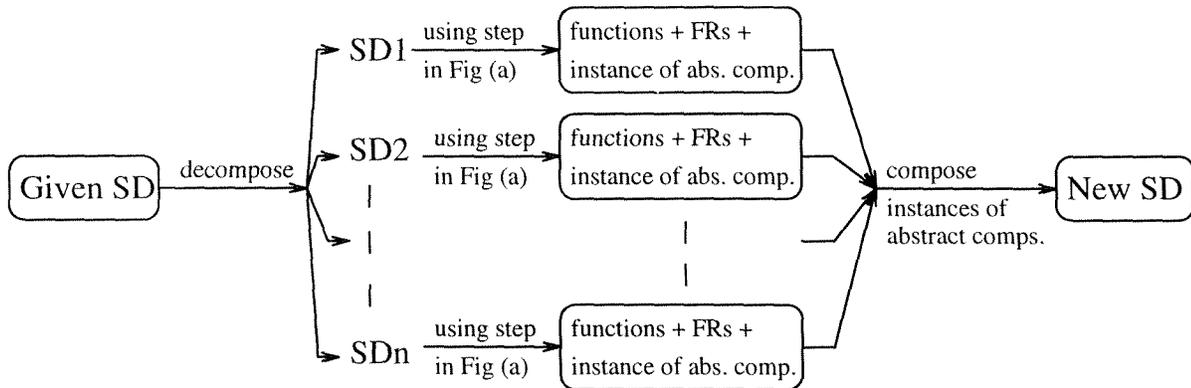
## 6 Example

The input to the system is the *struc-desc* shown in Figure 1(a). The system is to determine its functions and FRs. The system uses the models/SFFs of the op-amp, resistor, capacitor, thermocouple, analog-display, instrumentation-amp, and measuring-instrument device, inverter, amplifier, lp-filter, and integrator to analyze this device.

We will show two different executions of this example. The two executions differ in the knowledge provided to the system. In the first execution, as we will see, the system considers a large number of hypotheses to generate the function and the FR of the whole device. In the second execution the system is provided with



(a) The fetch-match-hypothesize-verify method



(b) The decomposition method

Figure 2: The two primary strategies

the additional knowledge that if the given structural description contains a sub-component of type transducer and a sub-component of type display, then hypothesize the given *struc-desc* to be an instance of the *measuring-instrument*. Once a function has been hypothesized for the given *struc-desc*, the given structural description is analyzed in a top-down fashion resulting in a highly focused reasoning.

Here are the main steps the algorithm goes through for the circuit in Figure 1(a):

- Since the given *struc-desc* does not match any SFF, partition the given circuit. A heuristic is used that produces six partitions corresponding to the *struc-desc* in the box *P, Q, R, S, T, and U* in Figure 3. Each partition is analyzed separately as shown in Figure 3 to get abstract devices.
- Since no subset of the analyzed partitions matches an SFF, it backtracks one step and tries to analyze the partitions again.
- The partitions are re-analyzed to get the results shown in Figure 4.
- A subset of analyzed partitions matches the SFF of a *sweep-generator*. A new *struc-desc* containing the instance of a *sweep-generator* is generated. The system tries to analyze the new structural description.
- Since there is no matching SFF, once again it backtracks and tries to re-analyze the partitions one

more time. Like the first attempt, this also results in a failure.

- The fourth attempt, shown in Figure 5, is finally successful. The final *struc-desc* (having the *instrumentation-amp*) is matched with the *measuring-instrument* SFF and the relevant functions are selected. The verified and selected function and FR templates are instantiated to get the FR shown in Figure 6.

Executing the same example with the additional rule, results in a highly focused problem solving. Once the hypothesized function of the device, *measuring-instrument-func*, is known, the cpds corresponding to the hypothesized function of the device are used to hypothesize functions for the sub-components. With the added rule, the system directly gets to the steps shown in Figure 5.

## 7 Conclusions

We have proposed a method for generating understanding of devices (FR models) that is function-specific and is at multiple levels of abstraction. Further the understanding at multiple levels is bridged enabling smooth transitions between levels while reasoning. The two main characteristics of the method proposed are that it uses the knowledge of frequently encountered abstract devices in the domain to derive FRs from structure and that

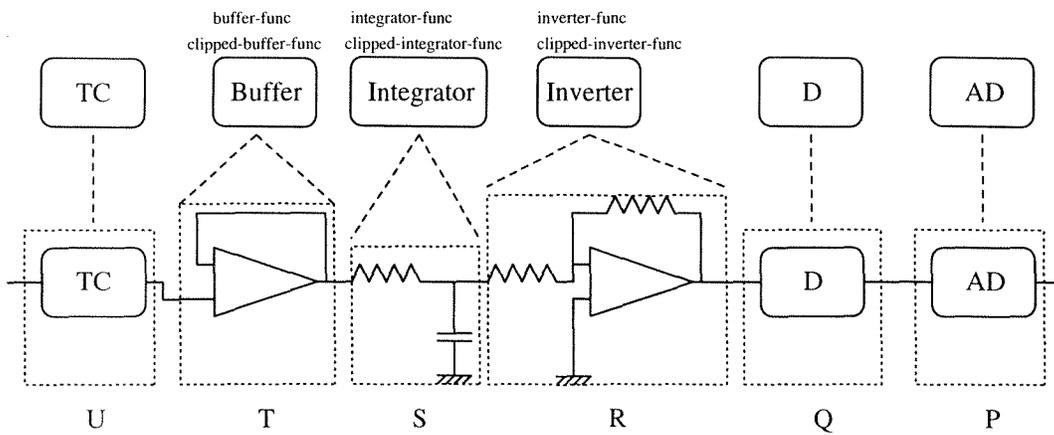


Figure 3: Analyzing the given structural description

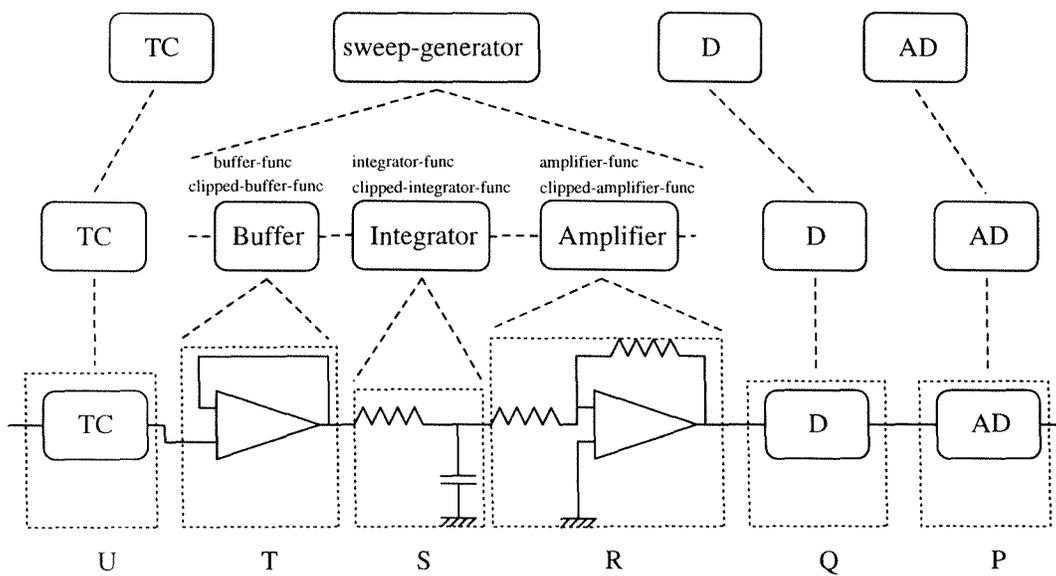


Figure 4: Analyzing the given structural description

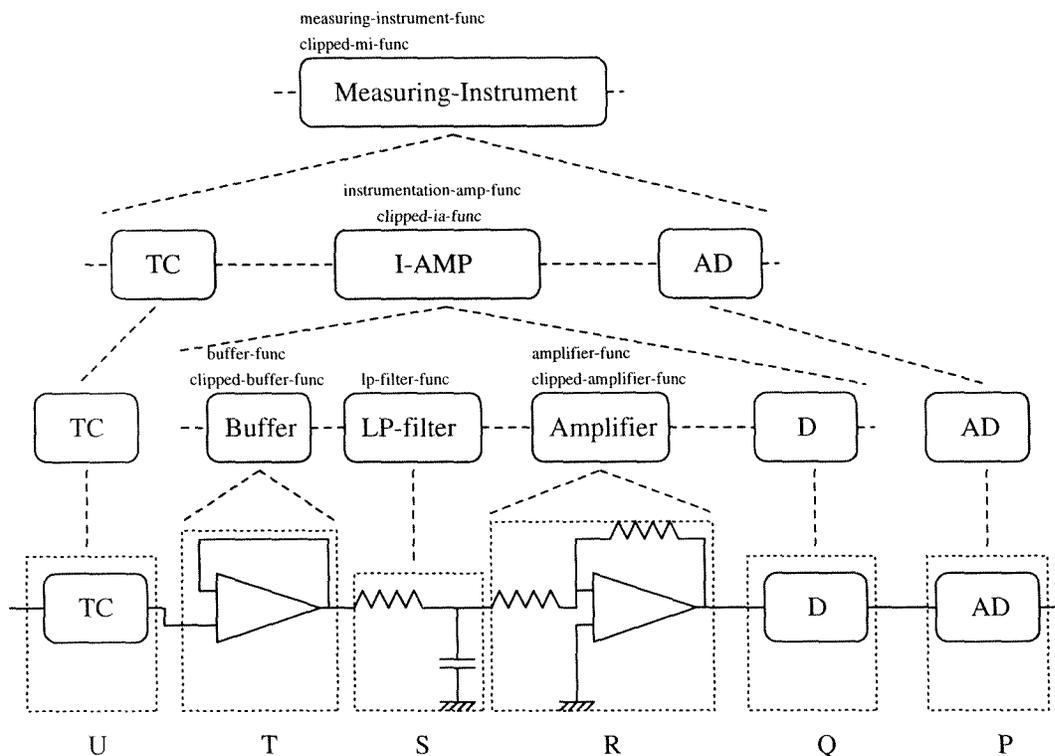
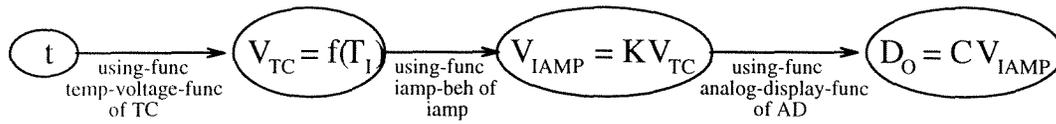


Figure 5: Analyzing the given structural description

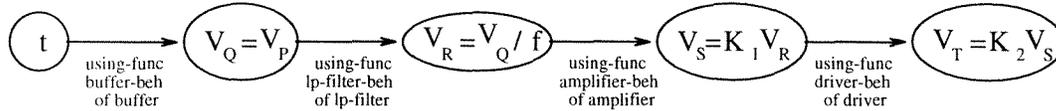
it uses a decompose-analyze-compose strategy to transform *struc-desc*'s. Given the right set of abstractions, a large class of devices can be analyzed by recursive application of just these two strategies. The specific set of SFFs provided to the system in a way captures the behaviors the user is interested in. Thus, our method also proposes a way of capturing user's interest in specific behaviors and a way of bringing that knowledge to bear upon structure-to-function reasoning. We also described how our approach complements the simulation based approach in understanding devices [IC92].

## References

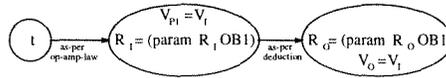
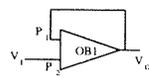
- [ACP91] Sanjay Addanki, Roberto Cremonini, and J. S. Penberthy. Graphs of models. *Artificial Intelligence*, 51(1-3):145-177, October 1991.
- [All90] D. Allemang. *Understanding Programs as Devices*. PhD thesis, The Ohio State University, 1990.
- [CGI93] B. Chandrasekaran, Ashok Goel, and Yumi Iwasaki. Functional representation as design rationale. *IEEE Computer, Special issue on concurrent engineering*, pages 48-56, Jan 1993.
- [Cha94] B. Chandrasekaran. Functional representation and causal processes. *Advances in computers*, 38, 1994.
- [dK85] J. de Kleer. How circuits work. In D. G. Bobrow, editor, *Qualitative Reasoning About Physical Systems*. MIT Press, 1985.
- [FF91] B. Falkenhainer and K. D. Forbus. Compositional modeling: finding the right model for the job. *Artificial Intelligence*, 51(1-3):95-143, Oct 1991.



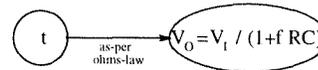
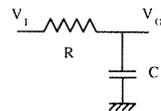
(a) *measuring-instrument-cpd*



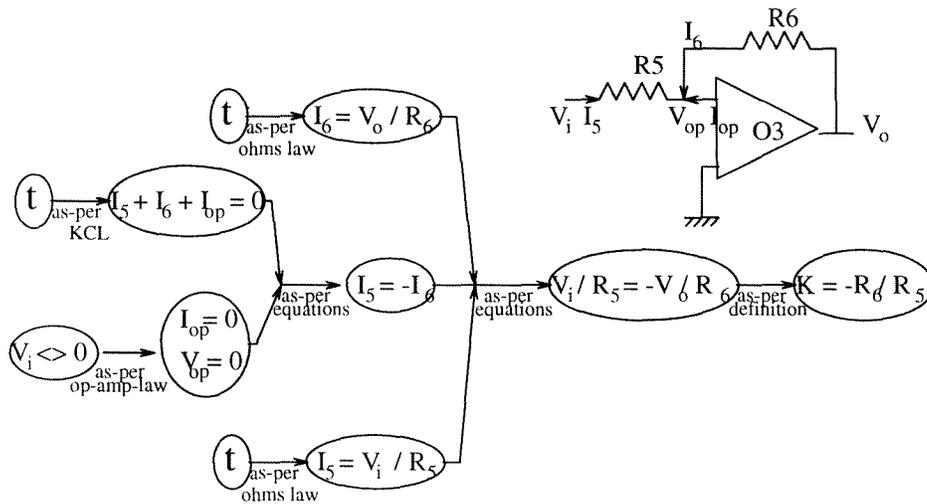
(b) *instrumentation-amp-cpd*



(c) *buffer-cpd*



(d) *lp-filter-cpd*



(e) *amplifier-cpd*

Figure 6: Cpd for the temperature measuring device

- [For84] Kenneth D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85-168, 1984.
- [IC92] Yumi Iwasaki and B. Chandrasekaran. Design verification through function- and behavior-oriented representations: Bridging the gap between function and behavior. In *Proceedings of AI in Design Conference*, pages 597-616, 1992.
- [IFVC93] Y. Iwasaki, R. Fikes, M. Vescovi, and B. Chandrasekaran. How things are intended to work: Capturing functional knowledge in device design. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1516-1522. Morgan Kaufmann, 1993.
- [IS86] Yumi Iwasaki and Herbert A. Simon. Causality in device behavior. *Artificial Intelligence*, 29, 1986.
- [KC87] Benjamin Kuipers and Charles Chiu. Taming intractable branching in qualitative simulation. In *Proceedings of the IJCAI*, pages 1079-1085, 1987.
- [Kui86] B. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:289-388, 1986.
- [NJA91] P. Pandurang Nayak, Leo Jaskowicz, and Sanjay Addanki. Automated model selection using context-dependent behaviors. Technical Report RC 16906 (#74798), IBM, IBM Research division, T. J. Watson Research Center, Yorktown Heights, NY 10598, May 1991.
- [SC88] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem-solving systems. In J. Kolodner and C. Riesbeck, editors, *Experience, Memory, and Reasoning*, pages 47-73. Lawrence Erlbaum Associates, 1988.
- [Str88] Peter Struss. Global filters for qualitative behaviors. In *Proceedings of the AAAI*, pages 275-279. AAAI, 1988.
- [Tha93] Sunil Thadani. Extending causal sequences to make teleological distinctions. In *Proceedings of The Ninth Conference on Artificial Intelligence for Applications*, Orlando, Florida, March 1993. IEEE, IEEE Computer Society Press.
- [Tha94] Sunil Thadani. *Constructing Functional Models of a Device from its Structural Description*. PhD thesis, The Ohio State University, 1994.
- [VIFC93] M. Vescovi, Y. Iwasaki, R. Fikes, and B. Chandrasekaran. Cfrl: A language for specifying the causal functionality of engineered devices. In *Proceedings of the Eleventh National Conference on AI*, pages 626-633. AAAI, July 1993.
- [Wel86] D. S. Weld. The use of aggregation in causal simulation. *Artificial Intelligence*, 30, 1986.

# A History-oriented Envisioning Method

**Takashi Washio, Ph.D.**

Research Center for Safety Engineering  
Mitsubishi Research Institute, Inc.  
1-8-1, Shimomeguro, Meguro-ku,  
Tokyo 153, Japan  
washio@mri.co.jp

## Abstract

A novel and generic approach named as "history-oriented envisioning" is proposed to qualitatively envision all possible and sound situations focusing on our intended partial behaviors and actions of an objective system. Some basic notations called "partial slices" and a "partial history" are also introduced as the extensions of the conventional history and its slices representing qualitative, temporal behaviors and actions of the system. They provide basic information of the intended partial behaviors and actions to the history-oriented envisioning. A major characteristic of the envisioning proposed here is the low complexity reduced by the specified history of behaviors and actions in addition to the conventional scenario. Another important characteristic is its incremental structure enabling the import of the measured information of the objective system in an on-line mode. These features of the history-oriented envisioning will promote new progress of qualitative envisioning theory toward its application to practical tasks of simulation, planning, design, measurements interpretation and diagnosis. The efficiency of the proposed method is demonstrated through an example to control a steam generator of a large scale plant.

## 1 Introduction

One of the primary tasks of qualitative reasoning is the envisioning of system "behaviors" established through leading works [de Kleer and Brown 1984; Forbus 1984, 1988; Kuipers 1984, 1986]. The "behaviors" are the collection of possible situation transitions resulted by system operations. The conventional framework of the envisioning consists of "attainable envisioning" and "total envisioning". The former derives the qualitatively possible situation transitions achieved from a specified initial state of a system. The later derives all possible situation transitions that may occur in the operations. The basic idea of these methods is to evaluate possible and sound behaviors of a system while maintaining a set of initially given background assumptions associated with exogenous quantity states, views and processes to the system without imposing our intentional changes of the assumptions.

In contrast with this standard methodology, the author is

interested in the envisioning in case that some portions of the evolutionary system behaviors are specified in advance by our intention. Some works to introduce exogenously specified quantity, view and process transitions into the envisioning have been done [Forbus 1989; Drabble 1993; Iwasaki et al. 1993; Vescovi et al. 1993]. Forbus defined an "action" as an exogenous replacement of some background assumptions in a system scenario, and established "action-augmented envisioning" that enumerates all possible transitions among combinations of quantity states, views, processes and actions. Besides, Drabble extended the notion of the actions to involve the exogenous specification of quantity states and to have qualitative time intervals. He also introduced a hierarchical sequence of actions to represent complex influences exogenously driven. Iwasaki and Vescovi proposed a language to specify intended functional behaviors in terms of causal transition rules. The latter two studies basically utilize the repetitions of the attainable envisioning to search their intended system behaviors.

However, a difficulty of combinatorial explosion of derived situations in the aforementioned envisioning methods have been reported, especially for the total and action-augmented envisioning, when the methods are adopted to practical scale applications [Caloud 1987; Forbus 1989; Forbus and Falkenhainer 1990, 1992; Amador et al. 1993]. For example, a self-explanatory simulation system "SimGen Mk2" to envision only local states of a system requires 4 hours to compile a simulator for a model of 9 containers and 12 pipes [Forbus and Falkenhainer 1992]. The main cause of this difficulty arises from the vast number of possible states envisioned, e.g. almost 1012 states, even for such a simple system. An efficient remedy to this difficulty is to restrict the scope of the envisioning within the partial behaviors and actions intentionally specified by following our interests or the objectives of application tasks. Many works on simulation, planning, diagnosis and design in the field of qualitative reasoning utilize the envisioning to obtain the information associated with specific system behaviors [DeCoste 1990, 1993; Drabble 1993; Forbus 1986; Forbus and Falkenhainer 1990, 1992; Ishida and Eshelman 1988; Iwasaki 1993; Pearce 1988; Umeda et al. 1991; Yannou 1993]. Their efficiency may be enhanced significantly by introducing the envisioning focused on specific and meaningful behaviors in addition to specific actions.

The work presented here is to propose a novel and generic envisioning method focused on specific partial behaviors and

actions of a system so called as "history-oriented envisioning". The efficiency of the proposed method is demonstrated through an example to control a steam generator of a large scale plant in the latter half of this paper.

## 2 Partial Slice and Partial History

An efficient representation of the partial behaviors and actions which are to be intentionally specified is defined first before the detailed discussion on the history-oriented envisioning. The fundamental structure of temporal behaviors and actions has been discussed in detail in the past works [Hayes 1979; Forbus 1984, 1989; Williams 1984; Dean and McDermott 1987]. Hayes and Forbus defined a sequence of changes of objects in a scenario as a "history" consisting of "episodes" and "events". Events always last for an instant, while episodes usually occur over a time interval. Each episode has a start and an end which are events that serve as its boundaries. Both of an event and an episode can involve the descriptions of quantity states, views, processes, actions, their relations and their transitions at a time (or a time interval). An assertion representing one of such descriptions is called a "token" [Dean and McDermott 1987]. Each token states a primitive fact in an event or an episode such as "amount of water in a pot is 1kg." or "boiling of water occurred.". In addition to these definitions, they also defined a "slice" of a history denoting a piece of a history at a particular time. Thus, a slice can be either of an episode and an event.

Based on these definitions, some new and important ideas on the history are introduced in this work as follows.

**Definition 1:** A "partial event" is a set of some tokens involved in an event in a history.  
A "partial episode" is a set of some tokens involved in an episode in a history.

**Definition 2:** A "partial slice" of a history is either of a partial event and a partial episode.

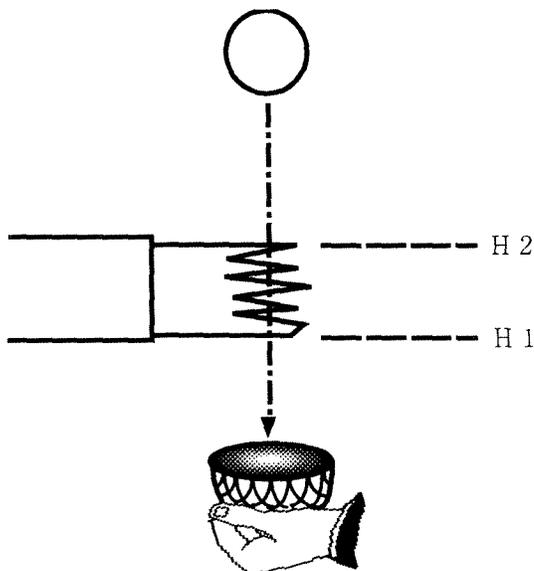


Fig.1 Catching a ball dropped through a flame.

The notation of a partial slice consists of a list of individuals that must exist, a list of quantity values and relations indicating the objects' states, a list of (in)active and (in)activated views, a list of (in)active and (in)activated processes and a list of (in)active and (in)activated actions. The (in)activated views, processes and actions mean these are (in)activated in a partial event, while (in)active ones are to be kept (in)active in a partial slice. The detailed descriptions of the contents in each list follow the notations in QP-theory [Forbus 1984]. T-operators are used to say that a particular token is true at some time, and M-operators represent the measured value of a quantity at some time.

Two of partial slices for the example of catching a ball dropped through a flame depicted in fig.1 are represented as follows.

**Partial Slice Heat-Flow-to-Ball-Active(?time)**  
 Individuals: ball a ball  
                   flame a flame  
 Quantities: (T A[temperature-of(ball)]  
                   <A[temperature-of(flame)] ?time)  
                   (M A[temperature-of(ball)] ?time) = T1 (1)  
                   (M Ds[temperature-of(ball)] ?time) = 1  
 Views:  
 Processes: (T Status(Heat-Flow(flame, ball, flame-ball),  
                   Active) ?time)  
 Actions:

**Partial Slice Catching-Ball-under-Flame(?time)**  
 Individuals: ball a ball  
                   flame a flame  
                   basket a basket  
 Quantities: (M A[position-of(ball)] ?time) = (-∞, H1)  
                   (M Ds[position-of(ball)] ?time) = -1 (2)  
 Views: (T Status(Contained-Stuff(ball, basket),  
                   Activated) ?time)  
 Processes:  
 Actions: (T Status(Catch-In(ball, basket),  
                   Activated) ?time)

The former partial slice represents that the objects of a ball and a flame exist at a particular time. Also, the amount and the derivative's sign of the temperature of the ball are T1 and positive respectively in addition to the fact that it is colder than the temperature of the flame, and the process of heat flow is operating at the same time. The latter means that the objects of a ball, a flame and a basket exist, and the ball descending under the flame is caught and settled in the basket at some time. Some lists and their contents can be left unspecified in a partial slice. For instance, any specifications of the position of the ball are not given within the former partial slice, while the quantity must be determined to specify a unique state of the ball. A distinct partial slice is "No-Specification(?time)" in which all lists are blank. This partial slice is used to represent that any behaviors and actions are unspecified at a time.

The term "?time" represents the temporal specification of a partial slice, and follows the conditions indicated below with respect to its duration and the limit hypotheses involved in its partial slice.

?time is an instant.  $\Leftrightarrow$  start(?time)=end(?time),  
 ?time is an interval.  $\Leftrightarrow$  start(?time)<end(?time), (3)  
 The duration of ?time is unspecified.  
 $\Leftrightarrow$  start(?time)  $\leq$  end(?time).

A partial slice involves some limit hypotheses. (4)  
 $\Rightarrow$  ?time is an instant.

The condition (4) states that any partial slice involving some limit hypotheses is a partial event. However, a partial event does not have to involve any limit hypothesis, because the extra portion of the event may own some limit hypotheses. On the other hand, any partial episode does not involve any limit hypothesis by definition.

The contents in the lists of individuals and quantities of a partial slice are used as the assumptions for the history-oriented envisioning. For instance, the assertion of "ball a ball" in the individual lists is an assumption specified by the partial slices in the above examples. On the other hand, the views, the processes and the actions in their lists do not represent their assumptions directly. The views and the processes in the scenario and the domain model of the QP-theory have the information of "Individuals", "Preconditions" and "Quantity Conditions" [Forbus 1984]. Also, the actions have the part of "Individuals" [Forbus 1989]. These are their assumptions to be active. Accordingly, the unifications of the contents in the view, the process and the action lists to the scenario and the domain model in the envisioning system provide their assumptions for the partial slice. For instance, the unification of "Heat-Flow(flame, ball, flame-ball)" in the process lists of the partial slice (1) to the following domain model (5) of the heat-flow process serves the contents in the "Individuals", "Preconditions" and "Quantity Conditions" of the process as the corresponding assumptions.

Process Heat-Flow(?src, ?dst, ?path)  
 Individuals: ?src an object,  
                   Has-Quantity(?src, heat)  
                   ?dst an object,  
                   Has-Quantity(?dst, heat)  
                   ?path a Heat-Path,  
                   Heat-Connection(?path, ?src, ?dst)  
 Preconditions: Heat-Aligned(?path) (5)  
 Quantity Conditions: A[temperature(?src)]  
                           >A[temperature(?dst)]  
 Relations: Quantity(flow-rate)  
                   flow-rate = (temperature(?src)  
                                   -temperature(?dst))  
 Influences: I-(heat(?dst), A[flow-rate])  
                   I+(heat(?src), A[flow-rate])

Another important assumption in a partial slice is the duration of ?time. Its specification controls the generation of the limit hypotheses in the envisioning process as explained later. The history-oriented envisioning utilizes all of the assumptions described here for a scenario involving the partial slice.

The definition of a "partial history" is given based on the partial slices.

**Definition 3** : A "partial history" of a history is a set of partial slices of the history which time intervals and instants are totally ordered in time domain.

A partial history has a list of the T-operators to say that a particular partial slice is true at some time. It also involves a list of constraints following the condition (3) on the temporal relations among the partial slice's time intervals. An example of a partial history for the ball is shown here.

Partial History Initial-and-Final-Ball

Partial Slices: (T Initial-Position-of-Ball(I0) I0)  
 (T Position-Decreasing-of-Ball-above-Flame(I1) I1)  
 (T Heat-Flow-to-Ball-Active(I2) I2)  
 (T No-Specification(I3) I3)  
 (T Position-Decreasing-of-Ball-under-Flame(I4) I4)  
 (T Catching-Ball-under-Flame(I5) I5) (6)

Time Constraints: (start(I0)=end(I0)), (end(I0)=start(I1)),  
 (start(I1)<end(I1)), (end(I1)=start(I2)),  
 (start(I2)  $\leq$  end(I2)), (end(I2)=start(I3)),  
 (start(I3)<end(I3)), (end(I3)=start(I4)),  
 (start(I4)<end(I4)), (end(I4)=start(I5)),  
 (start(I5)=end(I5))

The former half of this partial history before No-Specification(I3) specifies the partial behaviors of a ball beginning with its initial position until it touches the flame, while the latter after No-Specification(I3) describes the partial behaviors and actions associated with the ball under the flame until it is caught in a basket. Hence, this partial history specifies two clusters of partial slices mutually apart in time domain. Each partial slice of which the starting and the ending times are identical corresponds to an instant, e.g., Initial-Position-of-Ball(I0) and (T Catching-Ball-under-Flame(I5) I5). The partial slice, (T Heat-Flow-to-Ball-Active(I2) I2), is regarded as any of a partial event and a partial episode, because its duration is not specified in this example. Identical partial slices should not be neighbored mutually, because the neighboring identical partial slices are equivalent to such one partial slice, and they can be merged.

A partial history given to the history-oriented envisioning belongs to a scenario for the envisioning. Some partial histories may not be involved in the possible histories of the scenario. For example, the initiation of the heat flow from the flame to the ball before the touch of the ball to the flame is impossible. When such a partial history is specified, the history-oriented envisioning is halted at its intermediate step, and does not generate any environment consistent with the partial history.

### 3 History-oriented Envisioning

This section describes the outline and the algorithm of the history-oriented envisioning, and discusses its important features.

#### 3.1 Overview of History-oriented Envisioning

The outline of the history-oriented envisioning is depicted in fig.2. The vertical direction from the top to the bottom of the box stands for the time evolution of the behaviors and actions of an objective system. The horizontal axis represents the spectrum of the assumptions in the envisioning process. The shadowed area is the input information to the history-oriented envisioning, while the white part is its output. The domain

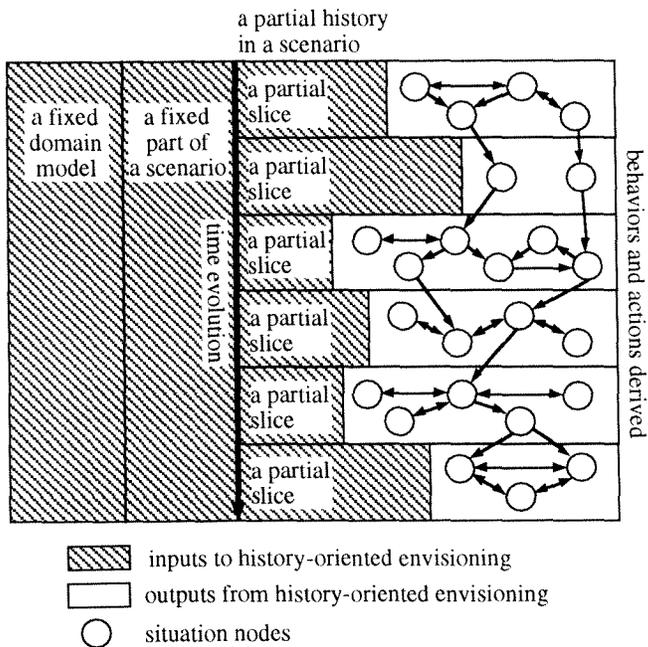


Fig. 2 The outline of the history-oriented envisioning.

model and a part of the scenario for the objective system are fixed over the entire time evolution in the envisioning. The conventional envisioning enumerates situation nodes grounded all possible and sound combinations of the rest opened assumptions associated with the system. On the other hand, the history-oriented envisioning imports the specifications on some extra portion of the assumptions for each time interval or instant in form of the partial slice. It derives situation nodes allowed within all possible and sound combinations of the remaining part of the opened assumptions while following the order of the specified time interval or instant. Accordingly, the history-oriented envisioning focuses on only the situations of the objective system within the intentionally specified partial behaviors and actions. The number of the derived situation nodes and the computation amount for the derivation are significantly reduced due to the low ambiguity of the conditions exponential to the number of the opened assumptions.

### 3.2 Algorithm

A partial history for the history-oriented envisioning must be compiled in advance to derive all assumptions associated with its partial slices. The algorithm of compiling a partial history is depicted in fig.3. This derives the set of assumptions,  $\Psi_i$  ( $i=1, \dots, n$ ), for every partial slice,  $PS(i)$ , and its time constraints,  $TC(i)$ , in the partial history by unifying their views, processes and actions to the scenario and the domain model in the envisioning system. The series of the partial slices and their time constraints in a partial history is sequentially compiled. (step 3) and (step 4) collect the assumptions of a partial slice. (step 3) obtains the assumptions explicitly represented in the individuals lists, the quantities lists and the time constraints of the partial slice, while (step 4) derives implicit ones of the views, processes and actions not directly represented in the partial slice. (step 2) and (step 5) check the violation of the condition (4) in the given partial slices, and quit the compiling if any violations are detected. (step 2)

- (step 1)  $i \leftarrow 1$ .  
 (step 2) Choose the  $i$ th partial slice  $PS(i)$  and the time constraints  $TC(i)$  on  $PS(i)$ .  
 If any explicit limit hypotheses appear in any lists of the individuals, the quantities, the views, the processes and the actions in  $PS(i)$ , check the condition (4). If it is violated, then stop.  
 (step 3) Let  $\Psi_i$  be a set of the contents in  $TC(i)$ , the individuals lists and the quantities lists of  $PS(i)$ .  
 (step 4) Unify the views, the processes and the actions in their lists to the scenario and the domain model in the envisioning system, and let  $\Psi_{su}$  be a set of the contents in the individuals, the preconditions and the quantity conditions of the unified predicates.  
 $\Psi_i \leftarrow \Psi_i \cup \Psi_{su}$ .  
 (step 5) If any implicit limit hypotheses appear in  $\Psi_i$ , check the condition (4). If it is violated, then stop.  
 If  $i < n$ , then go to (step 2), else end.

Fig.3 An algorithm of compiling a partial history.

- (step 1)  $i \leftarrow 1$ . Let the set of assumptions  $P_i$  be  $P_f \cup \Psi_i$ . Perform a total (or action-augmented) envisioning under  $P_i$ , and let  $Q$  be a set  $\{q(i,j) \mid q(i,j) \text{ is } j\text{th situation node generated in the envisioning, and } j=1, \dots, m_i.\}$ .  
 If  $Q$  is null, then stop.  
 (step 2)  $R \leftarrow \{\}$ .  
 For  $j=1$  to  $m_i$  {  
   (step 2.1) Let the set of assumptions  $P'_{i,j}$  be  $P_f \cup \text{Initial}(q(i,j))$ .  
   Perform one step attainable (or attainable action-augmented) envisioning under  $P'_{i,j}$ .  
   (step 2.2) Filter only the situation nodes which satisfies all conditions specified in  $\Psi_{i+1}$ .  
   Let  $R_f$  be a set of the filtered situation nodes, and  $R \leftarrow R \cup R_f$ . }  
 If  $R$  is null, then stop.  
 (step 3) Remove every situation node from  $Q$  which is not reachable to any  $r(j) \in R$  ( $j=1, \dots, m_r$ ).  
 (step 4)  $i \leftarrow i+1$ .  $S \leftarrow \{\}$ .  
 For  $j=1$  to  $m_r$  {  
   Let the set of assumptions  $P_{i,j}$  be  $P_f \cup \Psi_i \cup \text{Initial}(r(j))$ .  
   Perform an attainable (or attainable action-augmented) envisioning under  $P_{i,j}$ .  
   Let  $S_f$  be a set of the situation nodes generated in the envisioning, and  $S \leftarrow S \cup S_f$ . }  
 (step 5)  $S$  is represented as  $\{q(i,j) \mid j=1, \dots, m_i.\}$ .  $Q \leftarrow Q \cup S$ .  
 If  $i < n$ , then go to (step 2), else end.

Fig. 4 An algorithm of history-oriented envisioning.

checks any explicit limit hypotheses appearing in a partial episode, e.g., (T Status(Catch-In(ball, basket), Activated) Interval). On the other hand, (step 5) checks any implicit limit hypotheses appearing in the assumptions of a partial episode unified in (step 4), e.g., (Am[temperature(ball)]=100)& (Ds[temperature(ball)]=1) in the quantity conditions of a process.

Once, all assumptions are obtained, they are applied to the algorithm of the history-oriented envisioning represented in fig.4. This algorithm has an incremental structure. It accepts the assumption sets of partial slices in a partial history one by

one following their total order described in the time constraints of the history. Pf is the time-independently fixed portion of the set of background assumptions for a scenario over the envisioning. Pf corresponds to the ordinary scenario's part excluding the partial history.

(step 1) is to enumerate all possible situations for the first partial slice. The total envisioning (or action-augmented envisioning, if possible actions must be taken into account.) under the conditions of Pf and Ps1 is required, because any preceding situations have not been specified at this step. If the first partial slice is not consistent with the Pf, then no solutions are obtained, and the process is halted.

(step 2) is to identify all situations for the next partial slice which can be directly caused from the current situations. In (step 2.1), all possible one step transitions from a current situation are figured out. The one step attainable (or attainable action-augmented) envisioning is an ordinal attainable (or attainable action-augmented) envisioning under a given initial condition,  $q(i,j)$ , but its calculation is limited to one situation transition. The notation, Initial( $q(i,j)$ ), expresses that  $q(i,j)$  is a given initial condition for the envisioning. (step 2.2) filters out any inconsistent situations with the next partial slice. When the next partial slice is the No-Specification(?time), all situations obtained in (step 2.1) is filtered. On the contrary, when any current situations can not transit to consistent situations with the specifications of the next partial slice, the process is halted.

(step 3) is a sort of retrospective reasoning, while the other steps are perspective. Q contains all possible histories from the first to the current partial slice. Because the situations in the current partial slice which can transit to the next are limited as the result of the filtering in (step 2), some preceding histories in Q may not be causatively connected to the next partial slice. This step eliminates such dead end histories in Q.

(step 4) enumerates all possible situations in the new partial slice proceeded from the previous partial slice based on the attainable (or attainable action-augmented) envisioning. Each envisioning starts from a situation figured out in (step 2), and a list of all possible situations evolved for this partial slice is obtained.

(step 5) simply accumulates the situations for the new partial slice in Q. The entire envisioning is ended, and an environment following the given partial history is rested in Q, when all partial slices in the partial history have been processed.

The most of the computational load in this algorithm is caused by (step 4). The load of this step strongly depends on the number of its initial states R generated in (step 2), and the number is almost dominated by the efficiency of the situation filtering in (step 2.2) associated with the preceded partial slices. Hence, the computational load will be efficiently reduced, when many specifications are included in each partial slices. The loads of the other steps are not very significant. The total (or action-augmented) envisioning in (step 1) which algorithm is essentially efficient is performed just once, and its processing speed is greatly accelerated by the specifications of the first partial slice, unless the specifications are limited. (step 2) to perform only one step reasoning for each situation transition is also a quite cheap process. (step 3) is merely a network search of which various efficient algorithms are available. The simplicity of (step 5) is trivial.

An advantage of this algorithm is that the conventional total and attainable envisioning [Forbus 1984, 1988, 1989] can be utilized as parts of its process while reducing their solutions and processing time based on the information in a partial history. The unique difference of the envisioning utilized here from the conventional one is the imposition of the following rules to the situation node generation associated with the assumption of TC(i). They reject the situation nodes involving any limit hypotheses in a finite time interval.

(start(Ii)=end(Ii))&(The assumptions of PS(i) do not involve any limit hypotheses.)  
 $\Rightarrow$  (The assumptions generated for the situation node in the envisioning must involve some limit hypotheses.) (7)  
 (start(Ii)<end(Ii)),  
 $\Rightarrow$  (The assumptions generated for the situation node in the envisioning must not involve any limit hypotheses.),  
 where Ii is an instant or an interval for PS(i).

Another advantage is its incremental structure to process a partial history which enables its on-line application to import the new partial slice information step by step. Especially, when the amount of the specifications in each partial slice is large under its on-line import, its computation time will be applicable to the real time processing. These features of the algorithm are expected to be highly profitable for the real time applications of control, planning, measurement interpretation and diagnosis.

### 3.3 Soundness and Complexity

The standard total (or action-augmented) envisioning is sound for all possible system behaviors and actions under closed world assumptions of which the members are the only possible assumptions for the scenario [Forbus 1988]. The standard attainable (or attainable action-augmented) envisioning is also sound for its possible initial conditions under the closed world assumptions. Hence, each standard environment generated in (step 1), (step 2.1) and (step 4) in the algorithm depicted in fig.4 is sound for the given assumptions. The other steps of (step 2.2) and (step 3) reduce the generated nodes. Among these two steps, (step 2.2) is clearly sound, because it just filters situation nodes consistent with the constraints required for the transitions from the current slice to the next as well as the standard envisioning internally does. (step 3) is also sound, since it keeps all histories which do not contradict the assumptions of any partial slices and the scenario's fixed part in the context of a given partial history. These observations support the soundness of the history-oriented envisioning conducted through the algorithm of fig.4 under the closed world assumptions.

The complexity of an envisioning process sensitively depends on the number of unspecified assumptions for an environment [Forbus 1988, 1989]. Let P be the set of assumptions for a scenario, where its fixed portion is  $P_f \subseteq P$ . The set of unspecified assumptions for the standard envisioning is  $P_u = P - P_f$ , because it does not utilize any information of a partial history. If  $P_u$  consists of pairs of independent propositions  $p$  and  $\neg p$ , the number of states could increase by  $O(2^{|P_u|})$ . On the contrary, each partial slice specifies some extra portion of P in the history-oriented envisioning. The part of unspecified assumptions in P with respect to the first partial slice is  $P_{u1} = P - P_f \cup P_{s1}$ . Hence, the complexity of the

total envisioning in (step 1) of fig.4 is proportional to  $O(2^{|P_{ui}|-1})$ . The attainable envisioning through (step 2) to (step 4) is performed under the unspecified assumptions of  $P_{ui}=P-P_f \cup P_{si}$ , and its initial situations are limited to the preceding envisionment. Accordingly, its complexity could be less than  $O(2^{|P_{ui}|-1})$ . These derive the complexity of  $O(2^{|P_{ui}|-1})+O(\sum_{i=2}^n 2^{|P_{ui}|-1})$  in case of the maximum for the history-oriented envisioning. As the number of partial slices in a partial history,  $n$ , is independent with the assumptions, and also each  $|P_{ui}|$  is equal or less than  $|P_{ul}|$ , the complexity of the history-oriented envisioning can be quite small comparing with the standard.

#### 4 An Example

The basic performance of the proposed history-oriented envisioning has been evaluated through the application to the control of a steam generator commonly used in power plants for electricity generation. Figure 5 depicts the overview of the steam generator. It has a primary water tube (p-tube) passing through a secondary boiler tank (s-tank). Highly pressured hot water is supplied from a primary heat source by a pump. When the temperature of the primary water (p-water) is higher than the boiling point of the secondary water (s-water) in the low pressure tank, the heat flow from the primary to the secondary side can boil the secondary water. To compensate the decrease of the secondary water amount due to the escape of the steam (s-steam) to a turbine generator, the extra water feed (f-water) to the tank through a feed pipe (f-pipe) is required. At the beginning of its operation, the boiling of the secondary water has not occurred yet. We could qualitatively determine the future change of the primary water flow rate and its temperature based on the operational conditions of the heat source and the primary pump in the upper stream. Also, the future change of the temperature of the secondary feed water is qualitatively known based on the information of its reservoir. The temperatures of p-water and f-water are supposed to increase monotonically, while the flow rate of p-water are predicted to decrease monotonically in the mean time, and three of them are considered to settle at certain levels after some time. Our task is to plan all sound control strategies of the secondary water feed to the tank to start the boiling,

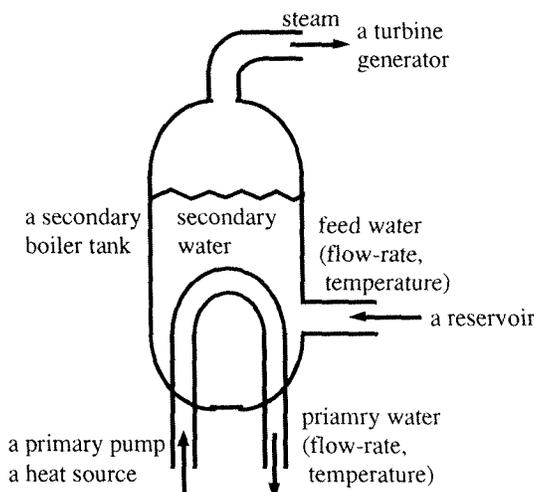


Fig. 5 A steam generator in a power plant for electricity generation.

when the three boundary quantities finish their transients. This kind of model based planing tasks have been researched in many AI literatures [Dean and Siegle 1990; Drabble 1993]. But, the most of them utilize the repetition of the attainable envisioning and its evaluations. In contrast with such conventional solution search, the history-oriented envisioning enumerates all possible plans within a finite number of envisionings.

A possible partial history corresponding to our mission can be written as shown in fig.6. It specifies the intended behaviors of the steam generator together with the predicted disturbances exogenously driven. The occurrence of boiling of the secondary water is intended at the final stage of the transients. Figure 7 represents two partial slices in the partial history. The former specifies the initial situation associated with the three boundary quantities, an endogenous quantity, i.e. temperature-of(s-water), and the intended processes. The latter specifies that the endogenous temperature-of(s-water) reaches at its boiling point, and simultaneously the boiling process is activated, when the three boundary quantities reach to their goal levels while maintaining the amount-of(s-water), the heat-flow and the fluid-flow.

The fixed portion of the scenario for this system and its partial history in fig.6 has been compiled to their assumptions and applied to the algorithm of history-oriented envisioning. A program specific to this type of examples has been developed and tested on a SPRAC-10 machine, while the development of a more generic program is currently under way. Figure 8 depicts the resultant envisionment indicating all possible and sound strategies to control the boiling under the given partial history. Totally, 29 situations were found. The author have tried to derive the total envisionment of this steam generator without specifying any partial history for comparison. However, the solution was not obtained due to the limitation of the memory capacity under the current program. The situations in the total envisionment can be at least more than 6000 even for this simple system, since it has 4 free boundary quantities. The planing of control strategies for process systems as this example is highly expensive, unless any constraints are introduced.

#### Partial History Boiling-Control

Partial Slices:	(T Initial-State(I0) I0)
	(T Start-of-Transient(I1) I1)
	(T Monotonic-Transient(I2) I2)
	(T End-of-Transient-and-Start-of-Boiling(I3) I3)
	(T Final-State(I4) I4)
Time Constraints:	(start(I0)=end(I0)), (end(I0)=start(I1)),
	(start(I1)=end(I1)), (end(I1)=start(I2)),
	(start(I2)<end(I2)), (end(I2)=start(I3)),
	(start(I3)=end(I3)), (end(I3)=start(I4)),
	(start(I4)=end(I4))

Fig. 6 A partial history to control the boiling of secondary water.

#### 5 Discussions and Related Works

One of the major characteristics of the history-oriented envisioning proposed here is the direct introduction of the specifications on the history of behaviors and actions to the envisioning process in addition to the conventional scenario. The envisioning focuses on only the specified situations and their

Partial Slice Initial-State(?time)	
Individuals:	p-tube a pipe f-pipe a pipe s-tank a container p-water a contained liquid s-water a contained liquid f-water a contained liquid
Quantities:	(T A[temperature-of(p-water)] >A[temperature-of(f-water)] ?time) (M A[temperature-of(p-water)] ?time) = T <sub>pmin</sub> (M Ds[temperature-of(p-water)] ?time) = 0 (M A[temperature-of(f-water)] ?time) = T <sub>fmin</sub> (M Ds[temperature-of(f-water)] ?time) = 0 (T A[temperature-of(s-water)] ?time) <A[t-boil(s-water)] ?time) (M Ds[temperature-of(s-water)] ?time) = 0 (M A[flow-rate-of(p-water)] ?time) = F <sub>pmax</sub> (M Ds[flow-rate-of(p-water)] ?time) = 0
Views:	
Processes:	(T Status(Heat-flow(p-water, s-water, p-tube), Active) ?time) (T Status(Fluid-flow(f-water, s-water, f-pipe), Active) ?time) (T Status(Boiling(s-water, Heat-flow), Inactive) ?time)
Actions:	
Partial Slice End-of-Transient-and-Start-of-Boiling(?time)	
Individuals:	p-tube a pipe f-pipe a pipe s-tank a container p-water a contained liquid s-water a contained liquid f-water a contained liquid
Quantities:	(T A[temperature-of(p-water)] >A[temperature-of(f-water)] ?time) (M A[temperature-of(p-water)] ?time) = T <sub>pmax</sub> (M Ds[temperature-of(p-water)] ?time) = 1 (M A[temperature-of(f-water)] ?time) = T <sub>fmax</sub> (M Ds[temperature-of(f-water)] ?time) = 1 (T A[temperature-of(s-water)] ?time) = A[t-boil(s-water)] ?time) (M Ds[temperature-of(s-water)] ?time) = 0 (M A[flow-rate-of(p-water)] ?time) = F <sub>pmin</sub> (M Ds[flow-rate-of(p-water)] ?time) = -1 (M A[amount-of(s-water)] ?time) = (0, M <sub>smax</sub> )
Views:	
Processes:	(T Status(Heat-flow(p-water, s-water, p-tube), Active) ?time) (T Status(Fluid-flow(f-water, s-water, f-pipe), Active) ?time) (T Status(Boiling(s-water, Heat-flow), Activated) ?time)
Actions:	

Fig. 7 An example of a partial slice for the control of the secondary water boiling.

histories, and derives those within small amount of computation. Iwasaki and Vescovi introduced a language named as CFRL to specify intended functional behaviors, and adopted it to a design support application [Iwasaki et al. 1993;

Vescovi et al. 1993]. However, the CFRL just filters intended behaviors from the possible behaviors resulted in the envisioning, and hence does not control the envisioning process directly. In contrast, the characteristic of efficient behavior focusing of the history-oriented envisioning highly enhances the applicability of the envisioning theory to the practical scale problems.

Another major characteristic is the explicit use of the information on the behaviors' history we intend on the objective system not only our intentional actions to eliminate unrequired or useless solutions for our reasoning tasks. Drabble developed a system named as EXCALIBUR for planning and reasoning with process systems [Drabble 1993]. The system utilizes some attainable envisioning processes, and can manage the actions changing continuous process quantities not only the ones causing discontinuous change of views and processes. Also, it can take a tree and hierarchical structure of actions sequences. But, it does not handle the explicit specifications on the behaviors evolved in process systems in the envisioning. As many applications such as planning, simulation modeling and design in practical fields are usually seeking processes of objective or intended sequences of behaviors given in advance, the history-oriented envisioning provides an efficient approach to these synthetic tasks.

The third important characteristic is the incremental structure of the history-oriented envisioning. Some works on the measurements interpretation utilize the total envisionments of the objective system to interpret the situation transition of that system [Forbus 1986; DeCoste 1990, 1993]. The total envisioning which is quite expensive for practical scale processes is essential in their approaches. In contrast, the incremental feature of the history-oriented envisioning enables to take the information of a sequence of behaviors and actions one by one in the on-line monitoring process. If the amount of the input information is not small, then its attainable envisioning in each step will be quite cheap, and its real time use will be possible. Thus, this characteristic meets the practical needs of analytic tasks such as measurements interpretation, control and diagnosis.

## 6 Conclusion

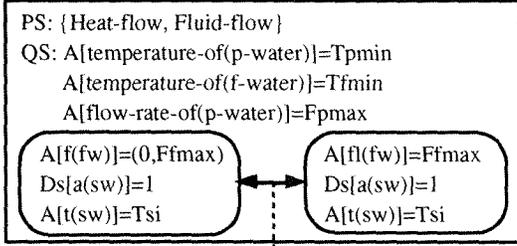
A history-oriented envisioning method has been proposed in this work together with its basic input named partial slices and a partial history. The applicability and efficiency of this method has been readily confirmed through an example of a control strategy planning for a steam generator. The major characteristic of the history-oriented envisioning are summarized as follows.

- (1) Soundness, small complexity and high efficiency comparing with the conventional envisioning.
- (2) Envisioning focused on a sequence of intended partial behaviors and actions.
- (3) Incremental envisioning to import the assumptions in an on-line manner.

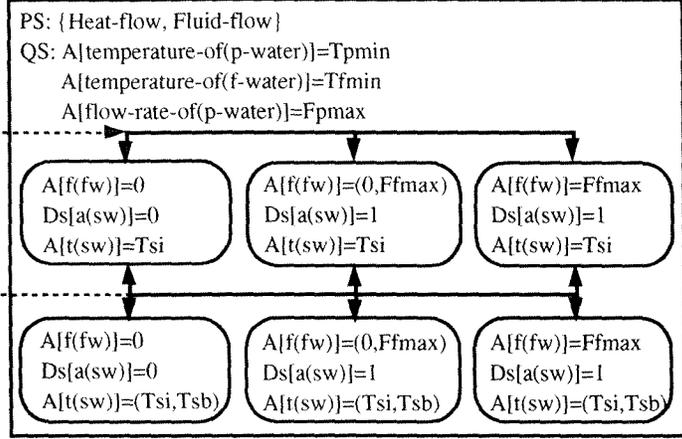
The ideas presented here will promote new progress of qualitative envisioning theory toward its application to practical tasks of simulation, planning, design, measurements interpretation, control and diagnosis.

Some following topics for our future works remains.

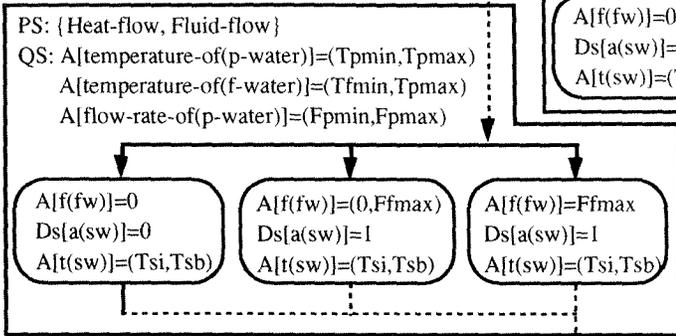
Initial-State



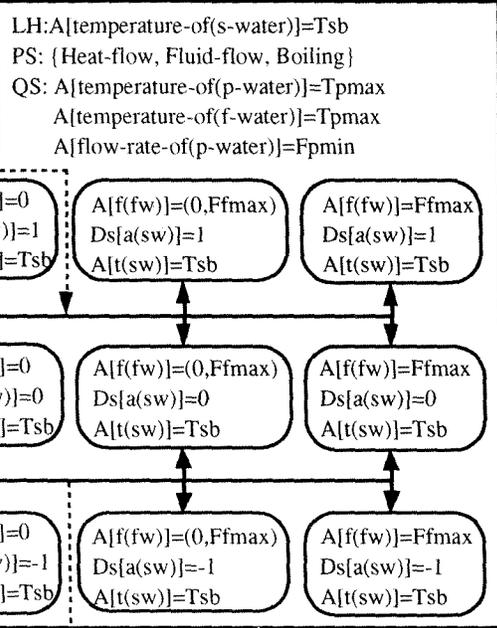
Start-of-Transient



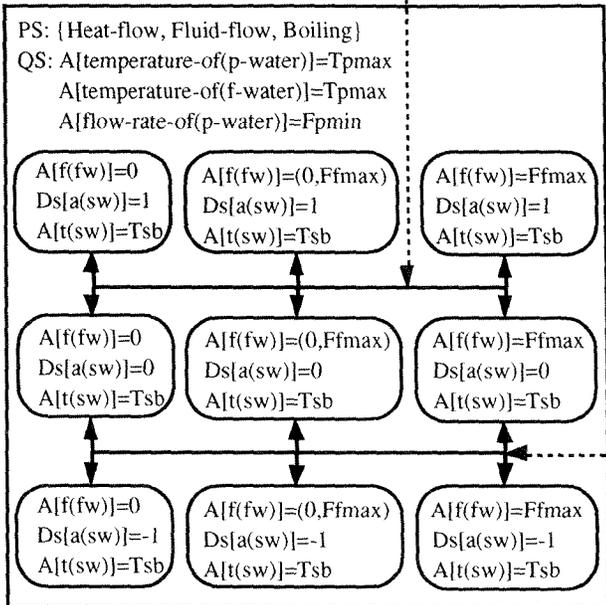
Monotonic-Transient



End-of-Transient-and-Start-of-Boiling



Final-State



bidirectional arc  $\longleftrightarrow$   
 unidirectional arc  $\dashrightarrow$

Abbreviations

A[f(fw)] : A[flow-rate-of(f-water)]  
 Ds[a(sw)] : Ds[amount-of(s-water)]  
 A[t(sw)] : A[temperature-of(s-water)]  
 Tsi: initial value of A[t(sw)]  
 Tsb: boiling temperature for A[t(sw)]

Fig.8 A situation transition diagram of a steam generator for a partial history.

- (1) Extension of a partial history: The structure of a partial history is merely a sequence of partial slices, although the sequence can be fragmented by the "No-Specification" partial slice. Its extension to tree, graph and hierarchical structures or specified transition rules from a history likewise the EXCALIBUR and the CFRL will enhance the usability of the history-oriented envisioning.
- (2) Seeking a better algorithm: The current algorithm for the history-oriented envisioning is a first version to evaluate the basic performance. Some more efficient algorithm might be developed.
- (3) Development of a code for general use: The domain of process systems to be envisioned in the current program is quite limited. The author is currently working on the development of a general code for history-oriented envisioning.

## Acknowledgments

The author wishes to express my thanks to Prof. Masaharu Kitamura in the Nuclear Engineering Department of Tohoku University for the useful discussions. The author extends the gratitude to Dr. Shuichi Koike and Dr. Hideaki Takahashi in Mitsubishi Research Institute, Inc. for their extensive support.

## References

- [Amador et al. 1993] Franz G. Amador, Adam Finkelstein and Daniel S. Weld. Real-Time Self-Explanatory Simulation. In proceedings AAAI-93, pp.562-567, 1993.
- [Caloud 1987] Philippe Caloud. Toward Continuous Process Supervision. In proceedings IJCAI-87, pp. 1086-1089, 1987.
- [Dean and McDermott 1987] Thomas Dean and Drew V. McDermott. Temporal Database Management. *Artificial Intelligence*, 36(3):375-399, 1988.
- [Dean and Siegle 1990] Thomas Dean and Greg Siegle. An Approach to Reasoning About Continuous Change for Applications in Planning. In proceedings AAAI-90, pp.132-137, 1990.
- [DeCoste 1990] Dennis DeCoste. Dynamic Across-Time Measurement Interpretation. In proceeding AAAI-90, pp.373-379, 1990.
- [DeCoste 1993] Dennis DeCoste. Dynamic Across-Time Measurement Interpretation. *Artificial Intelligence*, 51(1-3):273-341, 1993.
- [de Kleer and Brown 1984] Johan de Kleer and John Seely Brown. A Qualitative Physics Based on Confluence. *Artificial Intelligence*, 24:7-83, 1984.
- [Drabble 1993] Brain Drabble. EXCALIBUR: A Program for Planning and Reasoning with Process. *Artificial Intelligence*, 62:1-40, 1993.
- [Forbus 1984] Kenneth D. Forbus. Qualitative Process Theory. *Artificial Intelligence*, 24:85-168, 1984.
- [Forbus 1986] Kenneth D. Forbus. Interpreting Measurement of Physical Systems. In proceeding AAAI-86, pp.113-117, 1986.
- [Forbus 1988] Kenneth D. Forbus. QPE: Using Assumption-based Truth Maintenance for Qualitative Simulation. *Artificial Intelligence in Engineering*, 3(4):200-215, 1988.
- [Forbus 1989] Kenneth D. Forbus. Introducing Actions into Qualitative Simulation. In proceedings IJCAI-87, pp.1273-1278, 1989.
- [Forbus and Falkenhainer 1990] Kenneth D. Forbus and Brain Falkenhainer. Self-explanatory Simulations: An Integration of Qualitative and Quantitative Knowledge. In proceedings AAAI-90, pp. 380-387 1990.
- [Forbus and Falkenhainer 1992] Kenneth D. Forbus and Brain Falkenhainer. Self-explanatory Simulations: Scaling Up to Large Models. In proceedings AAAI-92, pages 685-690, 1992.
- [Kuipers 1984] Benjamin Kuipers. Commonsense Reasoning about Causality: Deriving Behavior from Structure. *Artificial Intelligence*, 24:169-203, 1984.
- [Kuipers 1986] Benjamin Kuipers. Qualitative Simulation. *Artificial Intelligence*, 29:289-338, 1986.
- [Hayes 1979] P. J. Hayes. The naive physics manifesto, in: D. Michie (Ed.), *Expert Systems in the Electric Age* (Edinburgh University Press, Edinburgh, 1979).
- [Ishida and Eshelman 1988] Y. Ishida and L. Eshelman. AQUA: Integrating Model-Based Diagnosis and Syndrome-Based Diagnosis, cmu-cs-87-111, 1987.
- [Iwasaki et al. 1993] Yumi Iwasaki, Richard Fikes, Marcos Vescovi, and B. Chandrasekaran. How Things Are Intended to Work: Capturing Functional Knowledge in Device Design. In proceedings IJCAI-93, pp.1516-1522, 1993.
- [Pearce 1988] D. A. Pearce. The Induction of Fault Diagnosis Systems from Qualitative Models. In proceedings AAAI-88, pp. 353-357, 1988.
- [Umeda et al. 1991] Yasushi Umeda, Tetsuo Tomiyama, and Hiroyuki Yoshikawa. A Design Methodology for a Self-maintenance Machine. DE-Vol.31, Design Theory and Methodology - DTM'91 - edited by L.A. Stauffer, Book No. G00641, 1991.
- [Vescovi et al. 1993] Marcos Vescovi, Yumi Iwasaki, Richard Fikes, and B. Chandrasekaran. CFRL: A Language for Specifying the Causal Functionality of Engineered Devices. In proceedings AAAI-93, pp.626-633, 1993.
- [Williams 1984] Brian C. Williams. Qualitative Analysis of MOS Circuits. *Artificial Intelligence*, 24:281-346, 1984.
- [Yannou 1993] Bernard Yannou. Qualitative Design with Envisionment. In working papers QR'93, pp.250-259, 1993.

# Activity Analysis: The Qualitative Analysis of Stationary Points for Optimal Reasoning

**Brian C. Williams**

Xerox Palo Alto Research Center  
3333 Coyote Hill Road,  
Palo Alto, CA 94304 USA  
bwilliams@parc.xerox.com

**Jonathan Cagan**

Department of Mechanical Engineering  
Carnegie Mellon University  
Pittsburgh, PA 15213 USA  
cagan+@cmu.edu

## Abstract

We present a theory of a modeler's problem decomposition skills in the context of *optimal reasoning* — the use of qualitative modeling to strategically guide numerical explorations of objective space. Our technique, called *activity analysis*, applies to the pervasive family of linear and non-linear, constrained optimization problems, and easily integrates with any existing numerical approach. Activity analysis draws from the power of two seemingly divergent perspectives — the global conflict-based approaches of combinatorial satisficing search, and the local gradient-based approaches of continuous optimization — combined with the underlying insights of engineering monotonicity analysis. The result is an approach that strategically cuts away subspaces that it can quickly rule out as suboptimal, and then guides the numerical methods to the remaining subspaces.

## Introduction and Example

Our goal is to capture a modeler's tacit skill at decomposing physical models and its application to focusing reasoning. This work is ultimately directed towards the construction of "self modeling" systems, operating in embedded, real time situations. This article explores the modeler's decompositional skills (Williams & Raiman 1994) in the context of *optimal reasoning* — the use of qualitative modeling to strategically guide gradient-based and other numerical explorations of objective spaces. Optimal reasoning is crucial for embedded systems, where numerical methods are key to such areas as estimation, control, inductive learning and vision. The technique we present, called *activity analysis*, applies to the pervasive family of linear and non-linear, constrained optimization problems, and easily integrates with any existing numerical approaches.

Activity analysis is striking in the way it merges together two styles of search that are traditionally viewed as quite disparate: first is the more strategic, conflict-based approaches used in combinatorial, satisficing search to eliminate finite, inconsistent subspaces (e.g., (de Kleer & Williams 1987)). The second is the

rich suite of more tactical, numeric methods (Vanderplaats 1984) used in continuous optimizing search to climb locally but monotonically towards the optimum. Activity analysis draws from the power of both perspectives, strategically cutting away subspaces that it can quickly rule out as suboptimal, and then guiding the numerical methods to the remaining subspaces.

The power of activity analysis to eliminate large suboptimal subspaces is derived from *Qualitative KT*, an abstraction in *qualitative vector algebra* of the foundational Kuhn-Tucker (KT) condition of optimization theory. The underlying algorithm achieves simplicity and completeness, by introducing the concept of generating *prime implicating assignments* of linear, qualitative vector equations. This process of ruling out feasible, but suboptimal subspaces in a continuous domain, nicely parallels the use of conflicts and prime implicant generation for combinatorial, satisficing search. The end result is a method that achieves parsimonious descriptions, guarantees correctness, and maximizes the filtering achieved from QKT.

Finally, activity analysis can be thought of as automating the underlying principle about monotonicity used by the simplex method to examine only the vertices of the linear feasible space. It then generalizes and automatically applies this principle to nonlinear programming problems.

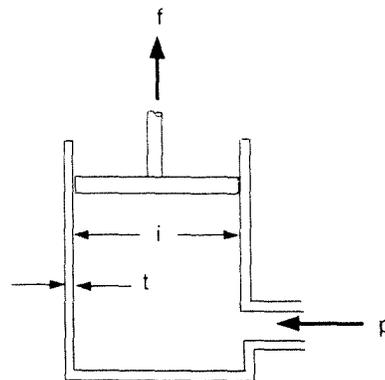


Figure 1: Hydraulic Cylinder

To demonstrate the task consider the design of a hydraulic cylinder, a classic optimization problem, introduced by Wilde (Wilde 1975) to demonstrate the related technique of monotonicity analysis. The cylinder (figure 1) delivers force  $f$ , through input pressure  $p$ . Weight is modeled as inside diameter ( $i$ ) plus twice the cylinder thickness ( $t$ ), force ( $f$ ) as pressure ( $p$ ) times cylinder area, and hoop stress ( $s$ ) as pressure times diameter acting across the thickness. The task is to find a parametric solution that minimizes cylinder weight, while satisfying constraints including positivity of variables ( $i, s, t, p, f > 0$ ), maximum pressure ( $P$ ) and stress ( $S$ ), and minimum force ( $F$ ) and thickness ( $T$ ) (design variables are in lowercase, fixed parameters in uppercase, and equality and inequality constraints are labeled  $h_i$  and  $g_i$ , respectively): Minimize  $i + 2t$ ,

subject to:

$$\begin{aligned} s - \frac{pi}{2t} &= 0, & (h_1 = 0) : & T - t \leq 0, & (g_2 \leq 0) \\ f - \frac{\pi i^2 p}{4} &= 0, & (h_2 = 0) : & p - P \leq 0, & (g_3 \leq 0) \\ F - f &\leq 0, & (g_1 \leq 0) : & s - S \leq 0, & (g_4 \leq 0) \end{aligned}$$

Given this symbolic formulation, activity analysis uses qualitative arguments to classify regions of the design space where optima might lie and where they cannot. After eliminating suboptimal regions, each remaining region identifies the solution as possibly lying on the intersection of one or more constraint boundaries. Each region reduces the dimensionality of the problem by the number of intersecting boundaries, thus significantly increasing the ease with which a solution can be found. In particular, for the cylinder problem activity analysis concludes there are two subspaces of the design space that could contain the optima, one subspace in which  $g_1$  and  $g_4$  become strict equalities, and a second in which all but  $g_4$  become strict equalities. The new problem formulation finds the optima of the two spaces and combines the results as follows (where "argmin" returns a set of optima):

Given: vector  $\mathbf{x} = (istpf)^T$ ,

1. Let  $\mathbf{Y} = \arg \min_{\mathbf{x}}(i + 2t)$ , subject to:

$$\begin{aligned} (h_1 = 0) & \quad (g_1 = 0) & \quad (g_3 \leq 0) \\ (h_2 = 0) & \quad (g_2 \leq 0) & \quad (g_4 = 0). \end{aligned}$$

2. Let  $\mathbf{Z} = \arg \min_{\mathbf{x}}(i + 2t)$ , subject to:

$$\begin{aligned} (h_1 = 0) & \quad (g_1 = 0) & \quad (g_3 = 0) \\ (h_2 = 0) & \quad (g_2 = 0) & \quad (g_4 \leq 0). \end{aligned}$$

3. Return  $\arg \min_{\mathbf{x}}(i + 2t)$ , subject to:

$$\mathbf{x} \in \mathbf{Y} \cup \mathbf{Z}.$$

Originally, the problem has a 3 dimensional space to be explored (3 degrees of freedom - DOF) resulting from 5 variables, 2 equality constraints. The reformulated problem rules out the interior and boundaries, except some intersections. The first remaining subspace corresponds to a *line* (1 DOF) produced by the intersection of the  $g_1$  and  $g_4$  constraint boundaries with the  $h_i$ . The

second remaining space is a *point* (0 DOF) produced by the intersection of  $g_1, g_2, g_3$  and the  $h_i$ . Thus finding a solution to the first problem involves a single, one dimensional line search, and the second involves solving the system of equalities to find the unique solution. Using parameter values  $F=1000$  lbf,  $T=.05$  in,  $S=30000$  psi,  $T=1000$  in, applying matlab to the original problem took 46.3 seconds. The optimal solution lies in  $\mathbf{Z}$ , which took only 8.1 seconds to run; no feasible solution exists in  $\mathbf{Y}$  for these parameter values.

Activity analysis draws inspiration from monotonicity analysis (MA) (Papalambros & Wilde 1979; Papalambros 1982). Monotonicity analysis began as a set of principles and methods used by modelers to identify ill-posed problems and to partially solve them, based on monotonic arguments alone. These principles were encoded in several rule-based implementations (Azram & Papalambros 1984; Choy & Agogino 1986; Rao & Papalambros 1987; Hansen, Jaumard, & Lu 1989), presented informally as heuristic methods.

The problem activity analysis addresses is similar in spirit to that of MA; nevertheless, the approach is quite different. First, activity analysis operates directly on an abstraction (QKT) of the Kuhn-Tucker (KT) conditions of optimization theory. While much easier to apply, QKT and KT are equivalent for the task, given only knowledge of monotonicities. Second, activity analysis provides a precise formulation of the problem in terms of *minimal stationary coverings*, that guarantees the solution is parsimonious, maximizes the filtering derived from QKT, and insures correctness. Finally, a mapping to *prime assignments* and the introduction of a simple but complete prime assignment engine guarantees that these three properties are achieved.

## Stationary Points and Kuhn-Tucker

For a point  $\mathbf{x}^*$  to be an optimum it is necessary that the point be *stationary*, that is any "down hill" direction is blocked by the constraints. Activity analysis exploits this fact to eliminate sets of points that can quickly be proven to be *nonstationary*, using a condition we call *Qualitative Kuhn-Tucker* (QKT). This section introduces the optimization problem, the concept of stationary point, and the traditional algebraic (Kuhn-Tucker) condition for testing stationary points. Activity analysis applies to the pervasive family of linear and non-linear, constrained optimization problems  $OP = \langle \mathbf{x}, f, \mathbf{g}, \mathbf{h} \rangle$ :

$$\begin{aligned} \text{Find } \mathbf{x}^* &= \arg \min f(\mathbf{x}) \\ \text{subject to: } & \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \\ & \mathbf{h}(\mathbf{x}) = \mathbf{0}, \end{aligned}$$

where column vectors are denoted in bold (e.g.,  $\mathbf{x}, \mathbf{x}^*, \mathbf{g}(\mathbf{x})$  and  $\mathbf{h}(\mathbf{x})$ ),  $f(\mathbf{x})$  is the *objective function*,  $\mathbf{g}(\mathbf{x})$  is a vector of *inequality constraints* and  $\mathbf{h}(\mathbf{x})$  is a vector of *equality constraints*. A point  $\mathbf{x} \in \mathbb{R}^n$  is *feasible* if it satisfies the constraints, and *feasible space*  $\mathcal{F} \subseteq \mathbb{R}^n$

denotes all feasible points (represented  $\mathcal{F} = \langle \mathbf{g}, \mathbf{h} \rangle$ ). A *feasible direction*  $\vec{s}$  from a feasible point is one through which a non-zero distance can be moved before hitting a constraint boundary.  $f(\mathbf{x})$  is *decreasing* at  $\mathbf{x}$  in direction  $\vec{s}$  if  $\nabla f(\mathbf{x}) \cdot \vec{s} < 0$ . Finally, a *point is stationary* (denoted  $\mathbf{x}^*$ ) if any direction that decreases the objective is infeasible. The Kuhn-Tucker (KT) conditions (Kuhn & Tucker 1951) provide a set of vector equations that are satisfied for a feasible point  $\mathbf{x}^*$  exactly when that point is stationary:

$$\nabla f(\mathbf{x}^*) + \lambda^T \nabla \mathbf{h}(\mathbf{x}^*) + \mu^T \nabla \mathbf{g}(\mathbf{x}^*) = \mathbf{0}^T \quad (\text{KT1})$$

subject to

$$\mu^T \mathbf{g}(\mathbf{x}^*) = \mathbf{0}^T, \quad (\text{KT2})$$

$$\mu \geq \mathbf{0}. \quad (\text{KT3})$$

$\mu^T$  transposes column vector  $\mu$  to a row. Gradients  $\nabla f$ ,  $\nabla \mathbf{g}$  and  $\nabla \mathbf{h}$  denote Jacobian matrices.  $\nabla f$  is a row vector  $(\frac{\partial f}{\partial x_1} \dots \frac{\partial f}{\partial x_n})$ .  $\nabla \mathbf{g}$  and  $\nabla \mathbf{h}$  are matrices  $(\frac{\partial g_i}{\partial x_j})$  and  $(\frac{\partial h_i}{\partial x_j})$ , respectively, where  $(a_{ij})$  denotes a matrix whose element in the  $i$ th row and  $j$ th column is  $a_{ij}$ , for all  $i$  and  $j$ . For example, KT1 and KT2 are equivalences between row vectors, and KT3 is a relation between column vectors.

In KT1 the  $-\nabla f$  term denotes directions of decreasing objective from  $\mathbf{x}^*$ , the term  $(\lambda^T \nabla \mathbf{h}(\mathbf{x}^*) + \mu^T \nabla \mathbf{g}(\mathbf{x}^*))$  denotes infeasible directions from  $\mathbf{x}^*$ , and the equality says the decreasing directions are all infeasible; hence,  $\mathbf{x}^*$  is stationary. More specifically,  $\vec{s}$  decreases the objective if it has a component in the  $-\nabla f$  direction ( $\vec{s} \cdot \nabla f < 0$ ). A direction is infeasible with respect to inequality constraint  $g_i(\mathbf{x}^*)$  if  $\mathbf{x}^*$  lies on the constraint boundary ( $g_i(\mathbf{x}^*) = 0$ ) and it has a component in the  $+\nabla g_i(\mathbf{x}^*)$  direction. A direction is infeasible with respect to equality constraint  $h_j(\mathbf{x}^*)$  if it has a component in either the  $-\nabla h_j(\mathbf{x}^*)$  or  $+\nabla h_j(\mathbf{x}^*)$  direction. Most importantly, if  $\mathbf{x}^*$  lies on multiple constraint boundaries, then an infeasible direction has a component which is a linear, weighted combination of the above gradients for these constraints. The weights are  $\mu$  and  $\lambda$ , (called *Lagrange multipliers*), and the combination is  $\mu^T \nabla \mathbf{g} + \lambda^T \nabla \mathbf{h}$  subject to KT2 and KT3. Hence all decreasing directions are infeasible when  $-\nabla f$  equals one of these linear combinations (KT1). Figure 2 shows an example of  $\nabla f$  and  $\nabla \mathbf{g}$  gradient vectors, and the combined weighted vector, which exactly cancels  $\nabla f$ .

A key property of KT is that it identifies *active* inequality constraints. Intuitively, a constraint  $[g_i]$  is active at a point  $\mathbf{x}$  when  $\mathbf{x}$  is *on* the constraint boundary and the direction of decreasing objective,  $\nabla f$ , is pointing into the boundary. When this is true  $\mu_i$  is positive. The basis of our approach is to conclude, by looking at signs of  $\mu$ , that the stationary points lie at the intersection of the constraint boundaries. One or more constraints have been identified as active, hence the name *activity analysis*.

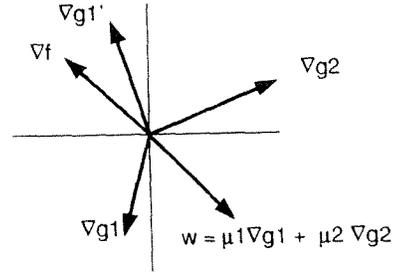


Figure 2: Example gradient vector diagram for KT.

## Qualitative KT Conditions

*Qualitative KT (QKT)* is an abstraction of KT that is a necessary, but insufficient, condition for a point being stationary. It is the means by which activity analysis quickly rules out suboptimal subspaces. Qualitative properties used by QKT to test a point  $\mathbf{x}$  include whether each constraint is active at  $\mathbf{x}$ , and the quadrant of the coordinate axes each gradient  $\nabla f$ ,  $\nabla \mathbf{g}$  and  $\nabla \mathbf{h}$  lies within. These properties can be extracted quickly and hold uniformly for large subsets of the feasible space, and parameterized families of optimization problems. QKT, its proof (see (Williams 1994)), and manipulations by activity analysis rely on a matrix version of SR1 – a hybrid algebra combining signs and reals. This algebra behaves as one expects given a familiarity with (scalar) sign algebra and traditional matrix algebra (see (Williams 1994; 1991)). Derived from KT, QKT states that a feasible point  $\mathbf{x}^*$  is stationary only if (QKT1):

$$[\nabla f(\mathbf{x}^*)] + [\lambda]^T [\nabla \mathbf{h}(\mathbf{x}^*)] + [\mu]^T [\nabla \mathbf{g}(\mathbf{x}^*)] \geq \mathbf{0}^T,$$

subject to

$$[\mu]^T [\mathbf{g}(\mathbf{x}^*)] = \mathbf{0}^T, \text{ and } (\text{QKT2})$$

$$[\mu_i] \neq \hat{-}, \quad (\text{QKT3})$$

where  $[\mathbf{v}]$ , called a *sign vector*, denotes the signs of the elements of  $\mathbf{v}$ , such that  $[v_i] \in \{\hat{-}, 0, \hat{+}\}$ . Recall KT said that to be stationary there must exist a weighted sum ( $\vec{w}$ ) of  $\nabla \mathbf{g}$  and  $\nabla \mathbf{h}$  that exactly cancels  $\nabla f$  (note  $\vec{w}$  is a row vector). QKT says a point is *nonstationary* unless there exists a  $\vec{w}$  that lies in the *quadrant* diagonal from that which contains  $\nabla f$ . For example, in figure 2  $\nabla f$  lies in the upper left quadrant; thus, a  $\vec{w}$  must exist that lies in the lower right. The sign vector  $[\mathbf{v}]$  denotes the quadrant containing a vector  $\mathbf{v}$ , and each component  $[v_i]$  describes where  $\mathbf{v}$  lies relative to the  $v_i = 0$  plane. For example,  $[\vec{w}] = (\hat{+} \hat{-})$  indicates that  $\vec{w}$  is in the lower right. Using this algebraic representation, the condition on diagonal quadrants becomes  $-\nabla f = [\vec{w}]$ .

Using only knowledge of the quadrant each constraint's gradient lies within and whether each constraint is active (indicated by the signs of the lagrange

multipliers  $[\mu]$  and  $[\lambda]$ , we know from KT that the quadrants  $\bar{\mathbf{w}}$  may lie within are a subspace of those described by  $[\mu]^T[\nabla\mathbf{g}] + [\lambda]^T[\nabla\mathbf{h}]$ . Thus,  $-\nabla f = [\bar{\mathbf{w}}] \subseteq [\mu]^T[\nabla\mathbf{g}] + [\lambda]^T[\nabla\mathbf{h}]$  (i.e., QKT1). For example, in figure 2 since  $\nabla g_1 (= (\hat{+} \hat{+}))$  lies in the upper right and  $\nabla g_2 (= (\hat{-} \hat{-}))$  lies in the lower left, it is possible for a  $\bar{\mathbf{w}}$  to lie in the lower right; thus, any  $\mathbf{x}$  satisfying these conditions may be stationary. But suppose  $\nabla g_1$  is replaced with  $\nabla g'_1$ , which lies in the upper left for points in some subspace  $\mathcal{F}_1 \subseteq \mathcal{F}$ . Then  $\bar{\mathbf{w}}$  may lie in the upper or lower left, but not the lower right; thus, all points in  $\mathcal{F}_1$  must be nonstationary. That is, evaluating  $-\nabla f = [\mu]^T[\nabla\mathbf{g}]$  for  $\nabla g_1$  and then  $\nabla g'_1$ :

$$\begin{aligned} (\hat{+} \hat{-}) \subseteq (\hat{?} \hat{?}) &= (\hat{+} \hat{+}) \begin{pmatrix} \hat{+} & \hat{+} \\ \hat{-} & \hat{-} \end{pmatrix} \text{ but} \\ (\hat{+} \hat{-}) \not\subseteq (\hat{-} \hat{?}) &= (\hat{+} \hat{+}) \begin{pmatrix} \hat{-} & \hat{+} \\ \hat{-} & \hat{-} \end{pmatrix} \end{aligned}$$

It is this second type of conclusion, made from only qualitative properties, that activity analysis uses to eliminate feasible subspaces of nonstationary points.

Next, to instantiate QKT1 on optimization problem  $OP \equiv (\mathbf{x}, f, \mathbf{g}, \mathbf{h})$ :

1. Compute Jacobians  $\nabla f$ ,  $\nabla\mathbf{g}$  and  $\nabla\mathbf{h}$  by symbolic differentiation.
2. Compute signs of Jacobians. For each element,
  - (a) replace real operators with sign operators, using properties  $[a + b] \subseteq [a] + [b]$ ,  $[ab] = [a][b]$ ,  $[a/b] = [a]/[b]$  and  $[-a] = -[a]$ .
  - (b) Substitute for sign variables  $[a]$  using positivity conditions ( $[a] = \hat{+}$ ), and perform sign arithmetic (e.g.,  $[5] \Rightarrow \hat{+}$ ,  $(\hat{-}) + (\hat{-}) \Rightarrow \hat{-}$ ).
3. Expand QKT1 by expanding matrix sums and products.

Returning to the hydraulic cylinder problem from the introduction, recall that  $\mathbf{x}$  is the vector  $(itfSp)^T$ , the objective  $f(\mathbf{x})$  is  $i + 2t$ , and the constraint vectors are:

$$\begin{aligned} \mathbf{h} &= \left( s - \frac{pi}{2t} \quad f - \frac{\pi i^2}{4} p \right)^T, \\ \mathbf{g} &= ( F - f \quad T - t \quad p - P \quad s - S )^T. \end{aligned}$$

The following shows  $[\nabla\mathbf{h}]$  after steps 2a (middle) and 2b (right):

$$\begin{aligned} [\nabla\mathbf{h}] &= \begin{pmatrix} \frac{-[p]}{[2][t]} & \frac{[p][i]}{[2][t]^2} & 0 & [1] & \frac{-[i]}{[2][t]} \\ -\frac{[\pi][i]}{[2]}[p] & 0 & [1] & 0 & -\frac{[\pi][i]^2}{[4]} \end{pmatrix} \\ &= \begin{pmatrix} \hat{-} & \hat{+} & 0 & \hat{+} & \hat{-} \\ \hat{-} & 0 & \hat{+} & 0 & \hat{-} \end{pmatrix}. \end{aligned}$$

Repeating for  $[\nabla f]$  and  $[\nabla\mathbf{g}]$ , and inserting into QKT:

$$\mathbf{0}^T \subseteq \begin{pmatrix} \hat{+} \\ \hat{+} \\ 0 \\ 0 \\ 0 \end{pmatrix}^T + \lambda^T \begin{pmatrix} \hat{-} & \hat{+} & 0 & \hat{+} & \hat{-} \\ \hat{-} & 0 & \hat{+} & 0 & \hat{-} \end{pmatrix}$$

$$+ \mu^T \begin{pmatrix} 0 & 0 & \hat{-} & 0 & 0 \\ 0 & \hat{-} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \hat{+} \\ 0 & 0 & 0 & \hat{+} & 0 \end{pmatrix}$$

Expanding matrix operations for step 3 results in equations QKT1(1)-(5):

$0 \subseteq$	$(+) - [\lambda_1] - [\lambda_2]$	$(1)$	$0 \subseteq$	$[\mu_4] + [\lambda_1]$	$(4)$
$0 \subseteq$	$(+) - [\mu_2] + [\lambda_1]$	$(2)$	$0 \subseteq$	$[\mu_3] - [\lambda_1] - [\lambda_2]$	$(5)$
$0 \subseteq$	$-[\mu_1] + [\lambda_2]$	$(3)$			

Note that the computation of sign matrices in step 2 is extremely simple, but surprisingly adequate for many problems. The symbolic algebra system Minima (Williams 1991) provides a general tool for deducing the signs of sensitivities (e.g.,  $\left[\frac{\partial f(\mathbf{x})}{\partial x_i}\right]$ ) subject to  $\mathbf{x}$  satisfying the equality and inequality constraints. Having achieved an easily evaluable condition that is sufficient for testing the suboptimality of infinite subspaces, we turn to its use for strategically focussing optimization.

## Activity Analysis and Prime Assignments

Activity analysis reduces an optimization problem to a set of simpler subproblems by "cutting" out feasible subspaces that are suboptimal. These subspaces contain all and only those points that are provably nonstationary by QKT (see (Williams 1994)). The output of activity analysis is a concise description of the remainder, called a *minimal pstationary covering* ("p-" stands for "possible" according to QKT). It is a set of feasible subspaces (and corresponding optimization problems), at least one of which is guaranteed to contain the true optimum. What is key is that the descriptions are parsimonious, they maximize the "filtering" achievable from QKT, and are always correct (these three properties are theorems, stated precisely in (Williams 1994)). This section states and demonstrates the activity analysis problem, and a sound and complete solution algorithm. The core is a mapping between *minimal pstationary subspaces* and *prime assignments*, and a general prime assignment engine for arbitrary systems of linear sign equations.

To start we say a point is *pnonstationary* if it follows from QKT that it is nonstationary; otherwise, it is *pstationary*. A *feasible subspace* is *pstationary* if all its points are pstationary, and *pnonstationary* if all its points are pnonstationary. Activity analysis maximizes its use of QKT while preserving correctness by eliminating exactly the pnonstationary subspaces from its description of the feasible space. This description is built from a set  $\Sigma$  whose elements result from strengthening one or more of the inequality constraints  $g_i \leq 0$  to strict equalities  $g_i = 0$ ; that is,  $\Sigma$  is the powerset of constraint boundary intersections. The description (called a *minimal pstationary covering*), covers the pstationary points by collecting all pstationary subspaces that are maximal under superset. These cover every pstationary subspace. The

activity analysis problem is then: *given optimization problem*  $OP = \langle \mathbf{x}, f, \mathbf{g}, \mathbf{h} \rangle$  *and instantiation of QKT* ( $=QKT(OP)$ ), *construct the minimal pstationary covering*  $C$ .

Mapping QKT(OP) to  $C$  relies on two observations: First, from QKT2 ( $\equiv [\mu_i(\mathbf{x})][g_i(\mathbf{x}) = 0]$ ) it follows that  $[\mu_i(\mathbf{x})] = \hat{+} - g_i(\mathbf{x}) = 0$  (denoted R1). That is, any point where  $[\mu_i] = \hat{+}$  must be on the  $g_i = 0$  constraint boundary. Thus, when activity analysis shows that a subspace of pstationary points makes  $[\mu_i] = \hat{+}$  for one or more  $g_i$ 's, it concludes that these points lie along the intersection of the  $g_i$  boundaries. Second, a particular set of variable assignments for QKT1, called *prime (implicating) assignments*, directly maps to the minimal pstationary covering by applying the first observation. The key here is that achieving parsimony, maximum filtering and correctness reduces to generating complete prime assignments.

The following properties, stated informally here, are given as definitions and theorems in (Williams 1994). First, a *(partial) assignment* to  $[\mathbf{x}]$  is a set  $\alpha$  which assigns each  $[x_i]$  at most one value,  $\alpha \subseteq \{[x_i] = s \mid [x_i] \in \mathbf{x}, s \in \{\hat{-}, 0, \hat{+}\}\}$ . We are interested in the *consistent assignments* to QKT1, where the  $[\mathbf{x}]$  to be assigned is a vector of lagrange multipliers ( $[\mu]^T[\lambda]^T$ ). Additionally, the consistent assignments must also satisfy the restriction of QKT3 ( $[\mu] \neq \hat{-}$ ). Note that each consistent assignment  $C$  has a corresponding subset  $S$  of feasible space, produced by applying R1 to the assignment and then adding the resulting active constraints to the original constraint set.  $S$  has the property that every point in  $S$  satisfies  $C$ .

Next, an *implicating assignment*  $\gamma$  is a consistent assignment to QKT1, such that whenever an extension to  $\gamma$  satisfies restriction QKT3, it also is consistent with QKT1. That is, assignment  $\gamma$  *implies* QKT1 under restriction QKT3. An implicating assignment has the important property that every point in its corresponding subspace  $S$  satisfies QKT. Thus  $S$  is a pstationary subspace.

Finally, a *prime assignment*  $P$  is an implicating assignment no proper subset of which is also an implicating assignment. Thus  $P$ 's corresponding  $S$  is a maximal pstationary subspace. Conversely, every maximal pstationary subspace is the corresponding subspace of some prime assignment. Thus the set of subspaces corresponding to all prime assignments is a minimal pstationary covering.

To produce all primes for QKT1, our prime assignment engine first computes the primes  $P_i$  of each scalar equation in QKT1, then combines them using minimal set covering. Pulling this all together, the activity analysis algorithm is:

*Given problem*  $OP = \langle \mathbf{x}, f, \mathbf{g}, \mathbf{h} \rangle$ :

1. Instantiate QKT1 (given earlier)  $\rightarrow QKT1(OP)$ ,
2. Compute prime assignments  $P_i$  of each  $QKT1_i(OP) \in QKT1(OP)$ ,

3. Compute minimal set covering of  $P_i \rightarrow P$ , deleting inconsistent assignments,
4. Extract minimal sets of  $[\mu_i] = \hat{+}$  assignments from  $P \rightarrow U$ ,
5. Map each element of  $U$  to a maximal pstationary subspace by applying  $[\mu_i(\mathbf{x})] = \hat{+} \rightarrow g_i(\mathbf{x}) = 0$ , producing a covering.
6. Formulate and return a new optimization problem from this covering.

Step one was demonstrated in the previous section. For steps two and three we note that QKT1 is an instance of a linear system of sign equations (denoted  $L([\mathbf{x}])$ ) and solve the prime assignment problem for arbitrary  $L([\mathbf{x}])$ . That is,  $L([\mathbf{x}])$  in vector form is  $\mathbf{0} \subseteq [\mathbf{B}] + [\mathbf{A}][\mathbf{x}]$ , with  $[\mathbf{A}]$  and  $[\mathbf{B}]$  being sign constant matrices,  $[\mathbf{x}]$  an  $n$  vector,  $[\mathbf{A}]$  an  $n$  by  $m$  matrix and  $[\mathbf{B}]$  an  $m$  vector. The  $i$ th scalar equation of  $L([\mathbf{x}])$  (denoted  $L_i([\mathbf{x}])$ ) is of the form:

$$L_i([\mathbf{x}]) \equiv 0 \subseteq [b_i] + \sum_{j=1}^m [a_{ij}][x_j].$$

For QKT1,  $\mathbf{x}^T$  is  $(\mu^T \lambda^T)^T$ ,  $[\mathbf{B}] = [\nabla f]$ , and  $[\mathbf{A}]$  is the matrix  $(\nabla \mathbf{g} \ \nabla \mathbf{h})$ . Additionally, we generalize the set of restrictions given by QKT3 (i.e.,  $[\mu_i] \neq \hat{-}$ ), to arbitrary sets of restrictions  $\mathbf{R}([\mathbf{x}]) \subseteq \{[x_i] \neq s \mid [x_i] \in \mathbf{x}, s \in \{\hat{-}, 0, \hat{+}\}\}$ . For the cylinder (table, end of QKT section), QKT1 has 5  $L_i([\mathbf{x}]$ 's, with  $\mathbf{x} \equiv (\mu_1 \mu_2 \mu_3 \mu_4 \lambda_1 \lambda_2)^T$ . For ease of reading we wrote terms  $\hat{+}[x_i]$  as  $[x_i]$ ,  $\hat{-}[x_i]$  as  $-[x_i]$ , and eliminated terms  $0[x_i]$ . The cylinder  $\mathbf{R}([\mathbf{x}])$  is  $\{[\mu_1] \neq \hat{-}, [\mu_2] \neq \hat{-}, [\mu_3] \neq \hat{-}, [\mu_4] \neq \hat{-}\}$ .

For step 2, the prime assignments of each  $L_i([\mathbf{x}])$  are constructed from three sets of scalar assignments, consistent with  $\mathbf{R}([\mathbf{x}])$ : those restricting one of the equation's terms ( $[a_{ij}][x_j]$ ) to be positive ( $P_i$ ), those making a term zero ( $Z_i$ ), and those making a term negative ( $N_i$ ), respectively:

$$\begin{aligned} P_i &\equiv \{[x_j] = [a_{ij}] \mid [a_{ij}] \neq 0, ([x_j] \neq [a_{ij}]) \notin \mathbf{R}([\mathbf{x}])\}, \\ Z_i &\equiv \{[x_j] = 0 \mid [a_{ij}] \neq 0, ([x_j] \neq 0) \notin \mathbf{R}([\mathbf{x}])\} \text{ and} \\ N_i &\equiv \{[x_j] = -[a_{ij}] \mid [a_{ij}] \neq 0, ([x_j] \neq -[a_{ij}]) \notin \mathbf{R}([\mathbf{x}])\}. \end{aligned}$$

Justifying  $P_i$ , for example, we know in general that  $[c] \neq 0 \rightarrow [c]^2 = \hat{+}$ . Thus  $[a_{ij}][x_j] = \hat{+}$  if  $[x_j] = [a_{ij}]$  and  $[a_{ij}] \neq 0$ . The derivation of  $Z_i$  and  $N_i$  is similar. Constructing the prime assignments for the cylinder  $L_i([\mathbf{x}])$  uses:

$i$	$N_i$	$Z_i$	$P_i$
1	$[\lambda_1] = \hat{-}, [\lambda_2] = \hat{+}$	$[\lambda_1] = 0, [\lambda_2] = 0$	$[\lambda_1] = \hat{-}, [\lambda_2] = \hat{-}$
2	$[\lambda_1] = \hat{-}, [\mu_2] = \hat{+}$	$[\lambda_1] = 0, [\mu_2] = 0$	$[\lambda_1] = \hat{+}$
3	$[\lambda_2] = \hat{-}, [\mu_1] = \hat{+}$	$\lambda_2 = 0, \mu_1 = 0$	$[\lambda_2] = \hat{+}$
4	$[\lambda_1] = \hat{-}$	$\lambda_1 = 0, \mu_4 = 0$	$[\lambda_1] = \hat{+}, [\mu_4] = \hat{+}$
5	$[\lambda_1] = \hat{+}, [\lambda_2] = \hat{+}$	$\lambda_1 = 0, \lambda_2 = 0, \mu_3 = 0$	$[\lambda_1] = \hat{-}, [\lambda_2] = \hat{-}, [\mu_3] = \hat{+}$

Next, recall that the prime (implicating) assignments for  $L_i([\mathbf{x}])$  must imply  $L_i([\mathbf{x}])$ . That is, they guarantee that it holds, given  $\mathbf{R}([\mathbf{x}])$ , independent of

additional consistent assignments. This is true if the right hand side of  $L_i(\mathbf{x})$  is guaranteed to be a superset of 0 (i.e., it is either 0 or  $\hat{?}$ ). The form of the assignments that achieve this for some  $L_i(\mathbf{x})$  depends on the value of  $[b_i]$ , where  $[b_i] = \left[ \frac{\partial f}{\partial x_i} \right]$  for QKT1. Suppose  $[b_i] = \hat{+}$ , then the right hand side must become  $\hat{?}$ . This holds exactly when at least one of the  $[a_{ij}][x_j]$  terms is negative (since  $0 \subseteq (\hat{-}) + (\hat{+}) = \hat{?}$ ). For example, in the cylinder QKT equation (2),  $\lambda_1 = \hat{-}$  guarantees that the equation is satisfied. The only other assignment that guarantees this is  $\mu_2 = \hat{+}$ . Thus the prime assignments for (2) are  $\{\lambda_1 = \hat{-}\}$  and  $\{\mu_2 = \hat{+}\}$ . The treatment of  $[b_i] = \hat{-}$  is analogous.

Next, suppose  $[b_i] = 0$ , then to imply  $L_i(\mathbf{x})$  the prime assignment can make the right hand side either 0 or  $\hat{?}$ . The first holds exactly when all terms are 0. The second holds when at least one term is positive and the other is negative. For example,  $\left[ \frac{\partial f(\mathbf{x})}{\partial x_i} \right] = 0$  in cylinder QKT1(3) :  $0 \subseteq -[\mu_1] + [\lambda_2]$ . Thus, the prime assignments are  $\{\lambda_2 = 0, \mu_1 = 0\}$  and  $\{\lambda_2 = \hat{+}, \mu_1 = \hat{+}\}$ . Note that  $\{\lambda_2 = \hat{-}, \mu_1 = \hat{-}\}$  is not acceptable, since by restriction  $[\mu_1] \neq \hat{-}$ . To summarize, the prime assignments of  $L_i(\mathbf{x})$  are 1)  $N_i$  if  $[b_i] = \hat{+}$ , 2)  $P_i$  if  $[b_i] = \hat{-}$ , and 3)  $\{Z_i\} \cup \{\{p, n\} | p \in P_i, n \in N_i\}$  if  $[b_i] = 0$  (where  $p$  and  $n$  in  $\{p, n\}$  do not contradict each other). Completing step two for the table of cylinder equations QKT1(1) - (5) produces:

$$\begin{aligned} \{\lambda_1 = \hat{+}\}, \{\lambda_2 = \hat{+}\} & P(1) \\ \{\lambda_1 = \hat{-}\}, \{\mu_2 = \hat{+}\} & P(2) \\ \{\lambda_2 = 0, \mu_1 = 0\}, \{\lambda_2 = \hat{+}, \mu_1 = \hat{+}\} & P(3) \\ \{\lambda_1 = 0, \mu_4 = 0\}, \{\lambda_1 = \hat{-}, \mu_4 = \hat{+}\} & P(4) \\ \{\lambda_1 = 0, \lambda_2 = 0, \mu_3 = 0\}, \\ \{\lambda_1 = \hat{+}, \lambda_2 = \hat{-}\}, \{\lambda_1 = \hat{+}, \mu_3 = \hat{+}\} \\ \{\lambda_1 = \hat{-}, \lambda_2 = \hat{+}\}, \{\lambda_2 = \hat{+}, \mu_3 = \hat{+}\} & P(5) \end{aligned}$$

The third step, constructing the composite primes for  $\mathbf{L}(\mathbf{x})$ , is based on:

$$\bigvee_{p \in P(\mathbf{L}(\mathbf{x}))} \left( \bigwedge_{a \in p} a \right) \equiv \bigwedge_{i=1}^n \left( \bigvee_{p \in P(L_i(\mathbf{x}))} \left( \bigwedge_{a \in p} a \right) \right)$$

The left hand side is a disjunction of the  $\mathbf{L}(\mathbf{x})$  prime assignments, and the right hand side is an expression in terms of the primes of  $L_i(\mathbf{x})$ , just computed. Thus, the desired primes result from reducing the expression on the right to minimal, disjunctive normal form. For this specialized case, this step is equivalent to computing minimal set covering of the  $P(L_i(\mathbf{x}))$  and then removing inconsistent assignments (see a standard algorithm text, or (Williams 1994) for our algorithm). For the cylinder, the minimal covering of P(1) - (5) produces just two prime assignments,

$$\begin{aligned} & \{[\lambda_1] = \hat{-}, [\lambda_2] = \hat{+}, [\mu_1] = \hat{+}, [\mu_4] = \hat{+}\}, \\ & \{[\lambda_1] = 0, [\lambda_2] = \hat{+}, [\mu_1] = \hat{+}, [\mu_2] = \hat{+}, [\mu_3] = \hat{+}, [\mu_4] = 0\}. \end{aligned}$$

The fourth step, extracting the minimal sets of  $[\mu_i] = \hat{+}$  assignments results in  $\{[\mu_1] = \hat{+}, [\mu_4] = \hat{+}\}$  and  $\{[\mu_1] = \hat{+}, [\mu_2] = \hat{+}, [\mu_3] = \hat{+}\}$ . The fifth step uses  $[\mu_i] = \hat{+} \rightarrow g_i(\mathbf{x}) = 0$  to map these sets to the equivalent minimal stationary covering. The sets tell us that  $g_1$  and  $g_4$  must be active, or  $g_1, g_2$  and  $g_3$ . The resulting cover is:

$$\begin{aligned} \mathcal{F}_1 & \equiv \langle \{g_2, g_3\}, \{h_1, h_2, g_1, g_4\} \rangle \text{ and} \\ \mathcal{F}_2 & \equiv \langle \{g_4\}, \{h_1, h_2, g_1, g_2, g_3\} \rangle, \end{aligned}$$

where  $(\mathbf{g}, \mathbf{h})$  is a space defined by inequality  $\mathbf{g}$  and equality  $\mathbf{h}$  constraints.  $\mathcal{F}_1$  and  $\mathcal{F}_2$  denote the line and point highlighted in the introduction to the cylinder example. The final step, formulating a new optimization problem, produces:

$$\begin{aligned} \text{Given: } S & \equiv \{\mathbf{x}^* | \mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathcal{F}} f(\mathbf{x}), \mathcal{F} \in \{\mathcal{F}_1, \mathcal{F}_2\}\}, \\ \text{Find: } & \min_{\mathbf{x} \in S} f(\mathbf{x}). \end{aligned}$$

The first part finds the minimum of each subspace in the covering. The second part selects from these the global minimum. A more expanded form was given in the introduction. Thus through this example we have demonstrated activity analysis' capability of partially solving constrained optimization problems from monotonicity constraints, and for synthesizing special purpose optimization codes.

## Discussion

As we mentioned in the introduction, activity analysis builds upon a large body of work from the mechanical engineering community on monotonicity analysis (Wilde 1975; Papalambros & Wilde 1979; Papalambros 1982), a method that uses derivative information to address the boundedness and global optimality of optimization problems. Monotonicity analysis provides two rules that test the boundedness of a formulation:

Rule 1: If the objective function is monotonic with respect to a variable, then there exists at least one active constraint that bounds the variable in the direction opposite of the objective function.

Rule 2: If a variable is not contained in the objective function then it must be either bounded from both above and below by active constraints or not actively bounded at all (i.e., in the latter case any constraint that is monotonic with respect to that variable must be inactive or irrelevant).

Both of these rules can be derived from the Kuhn-Tucker Conditions. They also follow as an instance of QKT and are embodied within activity analysis.

The result of monotonicity analysis (exhaustive application of the rules) are several sets of constraints one of which must be active for a problem to be well bounded. Various levels of rule-based implementations of monotonicity analysis have been described in (Michelena & Agogino 1988; Rao & Papalambros 1987; Azram & Papalambros 1984; Hansen, Jaumard, & Lu 1989), which guide numerical optimization codes. Choy and Agogino (Choy & Agogino 1986)

and Agogino and Almgren (Agogino & Almgren 1987) incorporate symbolic algebraic methods to aid in the evaluation of monotonicities and the solution of the optima. Cagan and Agogino (Cagan & Agogino 1987) apply monotonicity analysis to identify topological changes to designs that improve performance. While these systems address the optimal reasoning problem, they do not present algorithms proven to be sound and complete (each of these implementations has been described as "heuristic" (Rao & Papalambros 1987; Hansen, Jaumard, & Lu 1989)).

Activity analysis provides the following contributions: it formalizes the strategic way in which a modeler focuses optimization, as the process of generating minimal stationary coverings. It introduces QKT as a powerful condition for quickly eliminating large, suboptimal subspaces. Finally, it exploits this condition through a novel problem reformulation based on the prime, implicating assignments of linear sign equations. The activity analysis algorithm is sound and complete with respect to classifying the design space into stationary and nonstationary subspaces. The method of pruning suboptimal subspaces provides a continuous analog to the conflict-based approaches prevalent in combinatorial satisficing search (such as those used in model-based diagnosis (de Kleer & Williams 1987)). Activity analysis automates the intuitions about monotonicity exploited by the simplex method to examine only the vertices of the linear feasible space, most importantly, extending its application to nonlinear problems.

Activity analysis has been demonstrated on several engineering problems. The implementation is in Franz Lisp running on a Sparc 2. The problem reformulation is passed to Matlab's Optimization toolbox, where a wide variety of nonlinear gradient methods are available. (Williams 1994) describes an extension to activity analysis for cases where monotonicities are only partially known. Activity analysis is currently being pursued in the context of visual 3D matching problems and other embedded, realtime problems. Activity analysis can also be extended to provide explainable optimizers, ones that use QKT to provide common-sense explanations about optimality. Activity analysis is one of several techniques being developed that capture a modeler's expertise at strategically guiding numerical codes.

## References

- Agogino, A. M., and Almgren, A. S. 1987. Techniques for Integrating Qualitative Reasoning and Symbolic Computation in Engineering Optimization. *Engineering Optimization* 12:117-135.
- Azram, S., and Papalambros, P. 1984. An Automated Procedure for Local Monotonicity Analysis. *Trans. ASME, Journal of Mechanisms, Transmissions, and Automation in Design* 106:82-89.
- Cagan, J., and Agogino, A. M. 1987. Innovative Design of Mechanical Structures from First Principles. *AI EDAM* 1(3):169-189.
- Choy, J. K., and Agogino, A. M. 1986. SYMON: Automated SYMBOLic MONotonicity Analysis System for Qualitative Design Optimization. In *Proceedings of ASME 1986 International Computers in Engineering Conference*, 305-310.
- de Kleer, J., and Williams, B. C. 1987. Diagnosing Multiple Faults. *Artif. Intell.* 32:97-130.
- Hansen, P.; Jaumard, B.; and Lu, S. H. 1989. An Automated Procedure for Globally Optimal Design. *Trans. of the ASME, Journal of Mechanisms, Transmissions, and Automation in Design* 361-367.
- Kuhn, H. W., and Tucker, A. W. 1951. Nonlinear Programming. In Neyman, J., ed., *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*. Berkeley, CA: University of California Press.
- Michelena, N., and Agogino, A. M. 1988. Multiobjective Hydraulic Cylinder Design. *Journal of Mechanisms, Transmission and Automation in Design* 110:81-87.
- Papalambros, P., and Wilde, D. J. 1979. Global Non-Iterative Design Optimization Using Monotonicity Analysis. *Trans. ASME, Journal of Mechanical Design* 101(4):645-649.
- Papalambros, P. 1982. Monotonicity in Goal and Geometric Programming. *Transactions of the ASME, Journal of Mechanical Design* 104:108-113.
- Rao, J. R., and Papalambros, P. 1987. Implementation of Semi-Heuristic Reasoning for Bounded Analysis of Design Optimization Models. In *Advances in Design Automation, proceedings of the ASME Design Automation Conference*, 59-65.
- Vanderplaats, G. N. 1984. *Numerical Optimization Techniques for Engineering Design With Applications*. New York: McGraw-Hill.
- Wilde, D. J. 1975. Monotonicity and Dominance in Optimal Hydraulic Cylinder Design. *Trans of the ASME, Journal of Engineering for Industry* 94(4):1390-1394.
- Williams, B. C., and Raiman, O. 1994. Decompositional Modelling through Caricatural Reasoning. In *AAAI*.
- Williams, B. C. 1991. A theory of interactions: unifying qualitative and quantitative algebraic reasoning. *Artif. Intell.* 51.
- Williams, B. C. 1994. Characterizing Activity Analysis. in progress.

# Macroscopic Interpretation of Microscopic models

Kenneth Man-kam Yip\*  
Department of Computer Science  
Yale University  
P.O. Box 208285, Yale Station  
New Haven, CT 06520-8285.  
*yip-ken@cs.yale.edu*

## Abstract

The method of renormalization group and its associated geometric language for describing macroscopic phenomenology is a powerful technique for studying physics problems with many degrees of freedom. The abstract problem solving strategy embodied by the method – solving a hard problem by transforming it to a sequence of similar but simpler problems – acquires new power in the context of sophisticated physical theories. This paper describes a procedural implementation of the idea and suggests new research problems to turn this powerful technique into a qualitative reasoning method.

## Introduction

Model interpretation – extracting useful consequences out of a mathematical model – is a key problem in many areas of science and engineering. So it is not surprising that many qualitative reasoning research efforts are devoted to this task. Causal and incremental analysis of devices [de Kleer, 1984; Williams, 1984; Weld, 1988], envisionment and qualitative simulation of qualitative equations [Forbus, 1984; Kuipers, 1986], dimensional and order of magnitude analysis of algebraic and differential equations [Bhaskar and Nigam, 1990; Mavrovouniotis and Stephanopoulos, 1988; Raiman, 1991; Yip, 1993], and phase space analysis of dynamical systems [Sacks, 1991; Yip, 1991; Zhao, 1991; Nishida *et al.*, 1991; Bradley and Zhao, 1993] – these machineries have found applications in many domains.

Interpretation problems can be characterized according to three dimensions: (1) the essential degrees of freedom in the system (or roughly its size),<sup>1</sup> (2) the kind of information given as input, and (3) the kind of information required as output. In general, the difficulty of analysis increases rapidly as the degrees of freedom or the uncertainty of the input or the demand for precision of the output increase.

\*Supported in part by NSF NYI Award ECS-9357773

<sup>1</sup>Essential degrees of freedom measures the extent to which the parts of the systems are essentially coupled.

So far there has not been much attention paid to systems with many (or practically infinite) degrees of freedom. Many problems in physics have this character: phase transition of fluids, spontaneous magnetization of ferromagnetic material, and effective transport in turbulence, just to name a few. Common among these problems is the task of extracting macroscopic or large-scale behaviors of a system from its microscopic properties. It is not immediately obvious how the extraction can be done because neither direct numerical simulation nor analytical solutions of equations involving a huge number (could be on the order of  $10^{23}$ ) of interacting variables is feasible.

A macroscopic description is possible whenever the nitty-gritty of the microphysics can be subsumed into a few phenomenological parameters. Water, oil, or gas are complicated systems made up zillions tiny molecules interacting with some complicated force laws. However, their macroscopic properties can be summarized by similar functional relationships among few material constants like density and viscosity. The functional relationship is usually universal for a large class of fluids, while the specific values of the material constants are the detail-sensitive parts.

About twenty years ago Ken Wilson invented a technique known as the renormalization group (RNG), which becomes a standard method for constructing macroscopic theories from microscopic models [Wilson, 1975]; he was awarded a Nobel prize for this work. The technique has been applied in many areas other than critical phenomena, and continues to be a subject of current research. The purpose of this paper is two-fold: (1) to give an elementary description of the RNG method in terms of procedures, and (2) to suggest problem areas that might be fruitful to work on from the qualitative reasoning perspective.

There are several reasons why we propose to study RNG. First, RNG is based on a surprisingly simple idea: one solves a hard problem by transforming it to a similar but simpler one with the same answer, and by iterating the transformation until one arrives at a problem that is almost trivial to solve. Second, viewed as an abstract problem solving strategy, RNG is not-

ing novel. The method acquires new power when it is combined with problem-specific knowledge structures. Isolating the essence of the method and understanding its scope might provide a new source of problems for investigation into the fundamental issues of descriptive language, styles of reasoning, and representation techniques in qualitative reasoning. Third, explicit procedural encoding of RNG has an educational benefit: it might provide a better medium for beginners to learn and use this technique. Fourth, RNG has solved some of the hardest problems in physics, and theoretical scientists are applying it to all sorts of problems: percolation, onset of superfluidity, polymer conformation, elementary particle excitation, and turbulence, just to take a few examples [Wilson, 1983]. Therefore, automating aspects of the RNG will likely have a large payoff.

Despite the appearance of automating a technique applicable to a specialized class of problems, we want to stress our more general concerns for this line of research:

- To study the nature of scientific reasoning as practiced in normal science. We would like to codify some of the skills that professionals have in formulating problems, making approximations, explaining data, and testing theories.
- To solve real problems in an area of significance to modern science.
- To provide scientists with an intelligent workbench consisting of a library of powerful heuristic and qualitative methods.

The paper is organized as follows. We begin by describing the task of extracting macroscopic properties. Next we explain intuitively how and why the RNG works. Then we illustrate the procedural implementation of a particular type of RNG method. Finally, we conclude with problem areas that might need most "cognitive" help.

## The task: extracting macroscopic behaviors

Given a microscopic model with many degrees of freedom, the goal is to predict macroscopic behaviors that are independent of the inessential details of the microscopic model. This task is in general very difficult. The first difficulty is the large number of interacting variables; the second is that one does not really know which aspects of the microscopic model are inessential until one has solved the problem.

As an illustration of this task, we will consider a theoretical model for the spontaneous magnetization of ferromagnetic material. The theoretical model is known as the two-dimensional Ising model, one of the rare statistical mechanics models that can be solved exactly [Onsager, 1944]. Its study is still of considerable interest for two reasons: (1) it is probably the

simplest nontrivial problem to illustrate the essence of RNG, and (2) many variants of the model, such as the 3D Ising model, useful for the study of other critical phenomena cannot be solved explicitly, but RNG is still applicable to them.

At room temperature a piece of iron is ferromagnetic. At the microscopic level, the iron can be thought of as consisting of many tiny little atomic magnets spinning perpetually. The interaction forces among them are such that at low temperature two neighboring magnets tend to align in the same direction: both up or both down. As a result many more magnets will point to one direction than any other direction, creating a net magnetization at the macroscopic level. Thus, the piece of iron behaves like a bar magnetic. If the iron is heated, the atomic magnets will flip randomly due to the increasing thermal energy, and the alignment will be disturbed. At a critical temperature, known as the Curie temperature (770 C), the net magnetization vanishes. The critical temperature marks the transition of the iron from the ferromagnetic to paramagnetic phase.

The net magnetization  $M$ , which for our purpose can be defined as the absolute value of the average excess of atomic magnets pointing up over down, is found to obey a power law:

$$M \begin{cases} \propto |T - T_c|^\beta & T < T_c \\ = 0 & T > T_c \end{cases}$$

where  $T_c$  is the critical temperature, and  $\beta$  is called a critical exponent. Experiments have found  $\beta$  to be approximately 0.12 for two dimensional ferromagnetic systems (Fig. 1). It is conveniently to rewrite the power law in terms of a dimensionless temperature called the reduced temperature defined by  $t = \frac{T - T_c}{T_c}$ :  $M \propto |t|^\beta$ . The quantity  $M$  is also called an **order parameter** because it signifies the degree of orderliness of the system. At zero temperature the order parameter attains its maximum value.

A second important phenomenological quantity is the **correlation length**, which measures the maximum range of distance over which fluctuations in one part of the system (say the flipping of a magnetic spin) are correlated or have influence on fluctuations on another part of the system. When the correlation length is small, say on the order of a few separation distance of the atomic magnets, the system can be partitioned into a large number of statistically independent cells. As the critical point is approached, the correlation length grows rapidly and it eventually becomes comparable to the size of the system. Experiments have found the correlation length, denoted by  $\xi$ , *diverges* near the critical temperature and obeys the power law:

$$\xi \propto |t|^{-\nu}$$

where the critical exponent  $\nu$  is approximately 1 for two dimensional systems.

One reason why the critical exponents are significant is that they seem to be universal, i.e., they are

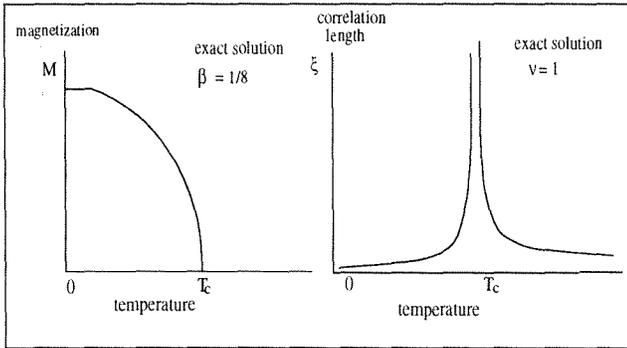


Figure 1: Schematic diagrams for the critical behaviors of the magnetization  $M$  and correlation length  $\xi$  for the 2D Ising model. For 3D Ising model, no exact solutions have been found; experimental data suggest  $\beta \approx 0.33$  and  $\nu \approx 0.63$ .

remarkably insensitive to the microscopic details of the system. A whole range of fluids and magnets have exponents that fall in a narrow range of the  $\beta$  and  $\nu$  values. The coincidence is particularly impressive when the critical exponents are not simple fractions like  $\frac{1}{2}$  or  $\frac{1}{3}$ ; some of them are believed to be irrational numbers. That means simple dimensional analysis will give *wrong* answers to the exponents.

The question of interest is: *Can the values of the critical exponents  $\beta$  and  $\nu$  be predicted from a microscopic description in terms of atomic magnets?*

Let's describe what input is required for the calculation. The input is a microscopic model with many degrees of freedom. Three ingredients are needed: (1) the microscopic variables and the values they can take, (2) a description of how the microscopic variables interact, and (3) a prescription for calculating averages.

Let's see what these ingredients are in the context of the 2D Ising model. Imagine a triangular lattice of spins (Fig. 2a), each of which can take one of the two values: +1 or -1. Physically it means the spins are constrained to point either in the up or down direction. The  $N$  spins, where  $N$  is of  $O(10^{23})$ , define  $2^N$  possible configurations for the system.

Each spin interacts with its nearest neighbors in such a way the interaction energy is lowered if the spins are aligned in the same direction: both up or both down. Mathematically the interaction can be described by a Hamiltonian  $H$ :

$$H = -K \sum_{\langle i,j \rangle} s_i s_j$$

where  $K > 0$  is the **coupling constant**, measuring the interaction strength between the nearest-neighbors  $s_i$  and  $s_j$ . One could complicate the model by adding an external field, triple interactions, quadruple interactions, and so forth. Physically, the Hamiltonian defines the total energy of a particular configuration. Because nature favors lower energy states, we put the negative

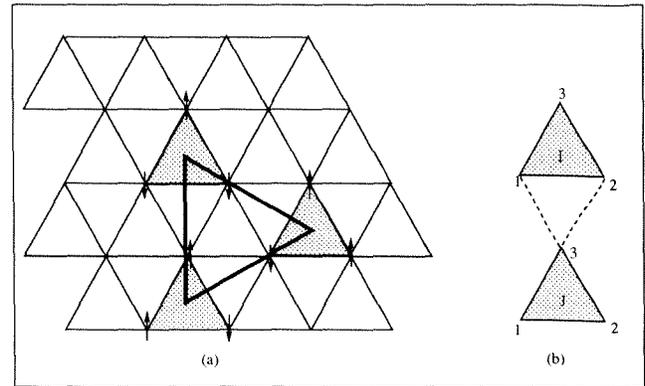


Figure 2: (a) Ising model defined on an infinite triangular lattice. Each site spins can point either up or down. Nine of them are shown. A block spin is formed by grouping three site spins within a shaded triangle. The value of a block spin is determined by a majority rule. The block spins form a coarse-grained system which is a scaled version of the original one. (b) Neighboring block spins can interact via two ways in the first order calculation. The interactions are indicated by dashed lines.

sign in front of  $K$  so that the lowest energy states correspond to the configurations in which all the spins are perfectly aligned: all up (+1) or all down (-1).

To define the averaging operator, we appeal to a fundamental result in statistical mechanics, namely, the probability  $p_s$  of a configuration  $s$  with Hamiltonian  $H_s$  is given by:

$$p_s = \frac{e^{-\frac{H_s}{k_B T}}}{Z}$$

where  $k_B$  is the Boltzmann constant,  $T$  the temperature, and  $Z$  the partition function defined by:

$$Z = \sum_s e^{-H_s}$$

where the sum is taken over all possible configurations  $s$ . It is conventional to absorb the factor  $\frac{1}{k_B T}$  into the coupling constant  $K$ . The inverse dependence on temperature means that as the temperature is raised, the coupling strength will decrease, thereby increasing the tendency of spins to misalign.

The average of a quantity  $A$  is defined as the configuration sum weighed by the Boltzmann probability  $p_s$ :

$$\langle A \rangle = \sum_s A_s p_s$$

where  $A_s$  is the value of  $A$  in a configuration  $s$ .

## Renormalization Group: a method and a new language

The universality of the critical exponents has an important consequence. Since the exponents are universal, it

is almost tautological to assert that they do not depend on the microscopic details. And therefore the critical exponents will be invariant under any transformation of the system that removes the details but preserves large-scale behaviors. We'll be more precise with this statement shortly.

The central idea of RNG is the iterative removal of degrees of freedom from the microscopic model in such a way that at every stage the new system we get is similar to the old one we start with. Consider a portion of the infinite triangular lattice with nine spins (Fig. 2a). Instead of keeping track of all the individual spins, we might group the spins into blocks of three and represent the entire block by a single new spin whose value is determined only by the spins inside the block. For instance, the value of the new spin, called it a **block spin**, can be decided by a majority rule: if two or more of the spins inside the block are up (+1), then the block spin is up (+1); otherwise it is down (-1).

Notice the effect of the projection. The new system looks exactly like the old one – well not exactly because the coupling constant will in general be changed and new coupling constants for triple or higher order interactions might be introduced. Let us write the transformation of the old to new system symbolically as:

$$K' = R(K)$$

where the prime denotes the coupling constant for the new system. To anticipate the generation of new coupling constants, one usually starts with a more elaborate Hamiltonian with a set of coupling constants  $K_1 \dots K_n$  most of which are set to zero initially. The transformation  $R$  is called the **renormalization group transformation**. It is a renormalization because we expect the transformation to change only the values of the coupling constants – they are said to be renormalized – and not the structure of interaction. It is a group (in fact only a semi-group) because the transformation can be iterated but the inverse of the transformation is not well-defined as information is lost by the projection procedure.

The transformation has to satisfy one important constraint, namely, it must preserve the partition function of the system:

$$\sum_{s'} e^{-H_{s'}} = \sum_s e^{-H_s}$$

where the sum on the left-hand side extends over the possible configuration  $s'$  of the block spins, and the sum on the right-hand side the possible configurations  $s$  of the original site spins.

With the projection and the requirement that the partition function be invariant, the transformation achieves the two basic goals of RNG: (1) to reduce the degrees of freedom (by a factor of 3 per iteration for this particular projection), and (2) to ensure the macroscopic quantities calculated from the new system are the same as those from the old one.

The significance of the transformation is revealed when we show that the critical exponents are calculable from the properties of the transformation equation. The first point to note is that the transformation  $R$  increases the lattice spacing by a factor of  $l$  ( $l = \sqrt{3}$  in our example) per iteration, thereby *reducing* the correlation length  $\xi$  by the same factor in units of the original microscopic lattice spacing. For almost all values of  $K$ , if the transformation is iterated infinitely many times, the correlation length will be reduced to zero. This reduction will occur except at a value of  $K$  where the correlation length is infinite to begin with, i.e., at a  $K$  value corresponding to the system's critical temperature. Physically it means that a system starts off with a critical temperature will remain at the critical temperature under the transformation. At all other temperature – no matter how close it is to the critical temperature – the system will be driven away from its critical state.

This last observation points to the importance of the unstable fixed point of the transformation equation: it corresponds to the system at its critical temperature and therefore the critical exponents should be calculable from properties of the unstable fixed point.

Fixed points of the equations are determined by the equation:

$$K = R(K)$$

Since  $R$  is in general a nonlinear mapping, there may be one or more fixed points.<sup>2</sup> The stability of a fixed point  $K^*$  is determined by the value of the derivative<sup>3</sup> of the mapping  $R$  with respect to  $K$  evaluated at the fixed point:

$$\lambda_t = \left. \frac{dR(K)}{dK} \right|_{K=K^*}$$

where  $\lambda_t$  is the eigenvalue; the subscript  $t$  indicates it is related to temperature. If  $\lambda_t$  is known, then the critical exponent  $\nu$  can be calculated by (see Appendix A):

$$\nu = \frac{\log l}{\log \lambda_t}$$

where  $l$  is the factor by which length scale changes under the transformation.

A fixed point is physically significant because the system at its fixed point is **scale-invariant**, i.e, the correlation length is either 0 or  $\infty$ . A fixed point is called **trivial** if it corresponds to zero correlation length, and called **critical** if the correlation length is infinite.

More significant than the calculation of critical exponents is the fact that the geometry around a critical fixed point explains **universality**: it explains why different physical systems near its critical point have the

<sup>2</sup>Of course, the equation might not have any fixed point at all. In fact it is an open question to decide when a renormalization equation can have a fixed point.

<sup>3</sup>Or the Jacobian if  $K$  is a set of couple constants.

same behavior. To understand this, we exploit an analogy with dynamical systems. We first note that the set of coupling constants (say,  $K_1, K_2, \dots, K_n$ ) of the Hamiltonian defines a  $n$ -dimensional  $K$ -space, called the **coupling space**. Each point in the  $K$ -space represents a physical system defined by the Hamiltonian with the values of  $K$ 's at that point. The iterated action of the transformation  $R$  generates an orbit of points in the  $K$ -space. The simplest limiting behavior of the orbit is towards a fixed point. The basin of attraction of a fixed point defines a class of systems whose large-scale behaviors are determined by the geometry of orbits around that fixed point.

Consider as an example an Ising model with three coupling constants ( $K_1, K_2$ , and  $K_3$ ). If it turns out that near a critical fixed point, the two directions  $K_2$  and  $K_3$  are contracting towards the fixed point, then these two coupling constants are called **irrelevant** because regardless of their initial values they tend to 0 under the action of  $R$ . In other words, at the critical fixed point, the system acts as if the irrelevant coupling constants don't exist.

## Procedural implementation of a renormalization group method

In the previous section, we explain the significance of the renormalization group equation  $K' = R(K)$ . In particular, its critical fixed point determines the critical behaviors of the system. The heart of a RNG calculation is to find an explicit form for the transformation  $R$ , which except in a few rare cases cannot be calculated exactly. In the following we will explain the details of an implementation of a particular RNG method, the so-called real-space renormalization. An RNG calculation consists of 5 steps:

1. formulation of the microscopic model
2. projection
3. derivation of the transformation equation
4. solution of the fixed point equation
5. calculation of eigenvalues and critical exponents

### Formulation of the microscopic model

Formulation of a novel problem within the RNG framework is arguably the most difficult step; it requires knowledge of the underlying physics of the problem and an articulation of what macroscopic properties one can reasonably expect RNG to be able to calculate. Applying RNG to problems in fully developed turbulence is a good illustration of the kind of difficulties one typically encounters [Yakhot and Orszag, 1986].

Fortunately the theoretical frameworks for studying a large class of problems in statistical mechanics (and quantum field theory) are already in a form amenable to RNG analysis. The Hamiltonian is represented by a higher-order procedure of two arguments, configuration and  $K$ . Representing objects like hamiltonian

or configuration as procedures has two advantages: (1) the object can be manipulated by algebraic operations, and (2) the object can be evaluated to give numbers.

```
(define hamiltonian
  (lambda (configuration K)
    (mult -1
      (mult K
        (sum (lambda (i j)
              (mult (configuration i)
                    (configuration j)))
            nearest-neighbor?
            (pick-2-combinations
              (configuration 'names)))))))
```

We have used a few auxiliary procedures in the definition of the hamiltonian. The **mult** procedure is a generic multiplication, handling both numbers and algebraic expressions. The **sum** procedure takes three arguments: (1) a term to be summed over, which is itself a procedure taking  $n$  indices (2) a filter predicate, which removes indices not satisfying the predicate, and (3) the set of all indices. In the example, **nearest-neighbor?** is the filter predicate; it returns true if two site indices are nearest neighbors. The expression **(configuration 'name)** gives all site labels, and the procedure **pick-2-combinations** returns all possible pairs of site labels.

The partition function is represented by:

```
(define partition-function
  (lambda (hamiltonian coupling-constant configs)
    (sum (lambda (config)
          (exp* (mult -1 (hamiltonian
                        config
                        coupling-constant))))
        identity
        configs)))
```

The **exp\*** is a generic exponentiation procedure.

### Projection

This step is the projection or coarse-graining of the system: the grouping of spins into blocks. The purpose of coarse-graining is to reduce the original problem to a problem with fewer degrees of freedom. Many choices of projection are possible, and each choice determines a renormalization transformation. Some choices just affect the ease of calculation, while others might produce transformations that either have no fixed point or generate non-physical fixed points. Although no sufficient condition on the validity of a projection operator has been proven, several necessary conditions are known. The most important requirement is that the coarse-grained system must be similar to the microscopic system, which means symmetry of the system is preserved under the projection. For instance, a projection that changes the dimension of the system or maps a scalar spin variable to a vector will certainly produce spurious results.

The majority rule described in previous section works very well with 2D Ising models. The value of

a triangular block spin is +1 when two or more of its spins are +1, and is -1 otherwise.

### Derivation of the renormalization equation

The constraint that the partition function be invariant relates the coarse-grained and the original Hamiltonian (see Appendix B):

$$-H'(S) = \log Z_0 + \langle V \rangle_0$$

where  $Z_0$  represents the energy contributed by *intra-block* spin interactions, while  $\langle V \rangle_0$  is the first term in the approximation for the energy contributed by *inter-block* spin interactions (Fig. 2b). The term  $\log Z_0$  can be ignored in the calculation of critical exponents because the term is analytic and contains no singularities [Goldenfeld, 1992].

In the second term on the right-hand side, the subscript 0 in the average operator indicates that the average is partial: the sum extends over the local site configurations inside one block as opposed to over the entire block configuration. Physically, it means the effect of the smaller site spins is averaged out during the coarse-graining operation.

The partial average operator has nice properties:

1. it is linear, i.e.,  $\langle A+B \rangle_0 = \langle A \rangle_0 + \langle B \rangle_0$  and  $\langle cA \rangle_0 = c \langle A \rangle_0$  where  $c$  is a constant,
2. it is separable, i.e.,  $\langle AB \rangle_0 = \langle A \rangle_0 \langle B \rangle_0$  if A and B belong to separate blocks,
3. it is symmetrical with respect to site spins, i.e.,  $\langle s_i \rangle_0 = \langle s_j \rangle_0$  if for any two site spins  $s_i$  and  $s_j$  belonging to the same block.

These rules allow the expression for  $V$  to be simplified. The end result is (Appendix C):

$$\langle V \rangle_0 = 2K \left( \frac{e^{3K} + e^{-K}}{e^{3K} + 3e^{-K}} \right)^2 \sum_{\langle IJ \rangle} S^I S^J$$

Comparison with the form of the coarse-grained Hamiltonian:

$$H'(S) = -K' \sum_{\langle IJ \rangle} S^I S^J$$

yields the renormalization equation:

$$K' = 2K \left( \frac{e^{3K} + e^{-K}}{e^{3K} + 3e^{-K}} \right)^2$$

The algebraic simplifier to evaluate these expressions has two parts: (1) a set of rewrite rules incorporating the properties of the partial average operator, and (2) explicit calculation of the partial average for spin variables.

### Solution of the fixed point equation

To find the fixed point, we just use the Newton's method:

```
> (newton
  (lambda(k)
    (- k
      (* 2
        (* k
          (* (- (+ (EXP (* -1 K)) (EXP (* 3 K)))
                2)
            (- (+ (* 3 (EXP (* -1 K)))
                (EXP (* 3 K))) -2)))))))
1)
0.3356134
```

An arbitrary initial guess 1 is used. The answer 0.336 compares reasonably well to the exact value  $K_c = 0.275$ .

### Calculation of the critical exponents

The derivative is done symbolically and then numerically evaluated at the fixed point to get  $\lambda_t$ . The correlation length exponent  $\nu$  follows from the formula:  $\nu = \frac{\log l}{\log \lambda_t}$ , where  $l = \sqrt{3}$ . The answer 1.133 is reasonably close to the exact value 1.

Calculation of the magnetization exponent  $\beta$  requires the addition of an external field to the starting Hamiltonian. The algebra with two coupling constants is little more complicated, but no new idea is involved.

### Evaluation

*How good is the block spin renormalization?*

The method carried to first order gives reasonable estimates for critical  $K_c$  and the correlation length  $\nu$ . But the estimate for  $\beta$  is not as good; it in fact gets the wrong sign. When the calculation is carried to second order, we get big improvements: within one percent accuracy for  $\beta$  and four percent for  $\nu$ . But the approximation seems to be asymptotic at best because third order calculations give *worse* answer.

The last conclusion seems to be generally true for real-space renormalization methods. The reason appears to be the uncontrolled proliferation of new coupling constants as the calculation is carried to higher order. More accurate renormalization methods are available: for dimension greater than two, the Fourier-space (or momentum-space)  $\epsilon$  expansion [Goldenfeld, 1992] is most accurate, while for dimension two or less, the Monte Carlo renormalization method seems most promising [Swendson, 1979].

Despite the accuracy problem, the block spin renormalization is simple to understand and relatively easy to carry out. So whenever applicable it is still the first method to try in order to develop a qualitative feel for the problem.

*How general is the procedural implementation?*

The basic steps of RNG are very much the same for real-space and momentum-space methods. In the momentum-space method, integrals replace discrete partition sums. The projection operator is simpler: at

each stage fluctuations higher than certain cutoff frequencies are averaged out. But the partial averaging is harder: the integrals get complicated quickly. Formal diagrammatic methods like Feynman diagrams are often used to simplify the calculation of these integrals.

The implementation assumes the microscopic interactions are given by a Hamiltonian and the transformation invariant is the partition function. These assumptions limit its applicability to problems like turbulent transport which has no Hamiltonian formulation and is far away from equilibrium. Generalizing RNG analysis beyond the equilibrium statistical mechanics formulation is an active research area in physics.

### Three open problems to explore

*“Where the renormalization group approach has been successful, a lot of ingenuity has been required: one cannot write a renormalization group cookbook.”*

— Ken Wilson, 1975.

Even within the realm of classical real-space and momentum space renormalization methods, there are still many areas that would benefit from computer help. By computer help, I don't mean numerical methods nor algebraic manipulations. Rather I mean cognitive help to aid a scientist in making judicious choice of the projection operator, the systematic exploration of the coupling space, and strategic formulation of the microscopic model.

### Intelligent choice of projection operator

The projection operator and the order parameter for the ferromagnetic Ising model are easy to construct because the ground states are extremely simple: all spins up or all spins down. For most other cases, the choice is not obvious. I will give two examples.

The antiferromagnetic Ising model [Creswick, 1992] is identical to the ferromagnetic one except the nearest-neighbor coupling constant  $K$  is *negative*. Physically, it corresponds to a situation in which the tiny atomic magnets prefer to be antiparallel to each other. Inside a triangular block of three spins, at least one of the bonds will be frustrated because there is no way to arrange the three spins so that they are all antiparallel to each other. A blind application of the 3-spin majority rule will lead to totally wrong answer.

The second example is the so-called XY-model [Creswick, 1992], a generalization of the 2D Ising model where the spins can point to any direction on a plane – like a compass needle. The model is proposed as a theoretical model to study the behavior of superfluid  $He^4$ . Again a naive application of the majority rule gives disastrous results. It turns out that the ground states contain “vortices” and the easiest way to deal with them is by a momentum-space type integration done in the real space.

These two examples are fairly typical. The failure of a projection operator can usually be traced to the

misidentification of the ground states and the violation of the symmetry group of the ground states. The open problem is: *Is there a general rule to construct a projection operator that will respect the symmetry group of the given ground states?*

### Systematic exploration of the coupling space

The geometry of orbits in the  $K$ -space contains information not only about critical behaviors but also about non-critical macroscopic properties over the entire phase diagram (e.g., the liquid-gas first order transition). The open problem is: *Can the fixed points of an RNG transformation and their connection in the  $K$ -space be mapped out intelligently?* This problem is reminiscent of the qualitative reasoning research in automatic phase space analysis.

### Strategic formulation of the microscopic model

Many physics problems have no obvious Hamiltonian formulation. The open problem is: *Is there a general characterization of the basic ingredients necessary for the application of RNG?* After all the twin principles of RNG – the systematic removal of degrees of freedom and the preservation of large-scale behaviors – do not seem to depend on the Hamiltonian formulation.

## Conclusion

In this paper, I have explained the essence of RNG, illustrated the procedural implementation of a particular real-space renormalization method, and proposed some open problems for qualitative reasoning research. The interest in RNG lies not so much in the calculation of critical exponents but in its methodology to extract macroscopic properties from microscopic descriptions without explicitly solving a huge number of coupled equations. Covering a qualitatively new class of problems, RNG could be a welcome addition to the qualitative reasoning arsenal. The abstract problem solving strategy that RNG embodies – solving a hard problem by reducing it to a sequence of similar but simpler problems – acquires new power in the context of sophisticated physical theories. Without the guide of problem-specific knowledge, the method remains sterile. Articulation of these specific knowledge structures and use of them to guide the application and interpret the results of RNG methods – these two tasks might hold the key to helping scientists solve some of the hardest problems in science.

### Appendix A: Calculation of critical exponent $\nu$

Let  $K_c$  be the critical fixed point of the RNG equation and  $K'$  near  $K_c$ .

$$\begin{aligned} K_c = R(K_c) &\Rightarrow K' - K_c = K' - R(K_c) \\ &\Rightarrow K' - K_c = R(K) - R(K_c) \\ &\Rightarrow K' - K_c = \lambda_c(K - K_c) + O((K - K_c)^2) \\ &\Rightarrow K' - K_c = l^{\nu_c}(K - K_c) \end{aligned}$$

But  $\xi(K) = l\xi(K')$ . Substituting the above result, we get  $\xi(K - K_c) = l\xi(l^{y_t}(K - K_c))$ . Since  $\xi \sim |K - K_c|^{-\nu}$ , we finally get  $\nu = \frac{1}{y_t} = \frac{\log l}{\log \lambda_t}$ .

## Appendix B: Calculation of the transformation invariant

The transformation must preserve the partition function:

$$\sum_{s'} e^{-H'_{s'}} = \sum_s e^{-H_s}$$

Rewrite the sum on the left on right-hand side as a double sum: first sum over the site configuration  $\{\sigma\}$  consistent with a given block spin configuration  $s'$  and then sum over all block spin configurations. We get:

$$\sum_{s'} e^{-H'_{s'}} = \sum_{s'} \sum_{\sigma} e^{-H_{s',\sigma}}$$

which gives

$$e^{-H'_{s'}} = \sum_{\sigma} e^{-H_{s',\sigma}}$$

Write the Hamiltonian  $H$  in terms of two parts:

$$H = H_0 - V$$

where  $H_0$  is the intra-block interaction, and  $V$  is the inter-block interaction. The negative sign in front of  $V$  simplifies some algebra. Then,

$$e^{-H'_{s'}} = \sum_{\sigma} e^{-H_{s',\sigma}} = \sum_{\sigma} e^{-(H_0 - V)} = Z_0 \langle e^V \rangle_0$$

where  $Z_0 \equiv \sum_{\sigma} e^{-H_0}$  and  $\langle e^V \rangle_0 \equiv \frac{\sum_{\sigma} e^{-H_0} e^V}{Z_0}$ . Finally, take logarithm on both sides of the equation and keep the first term in the cumulant expansion [Goldenfeld, 1992, Chap 9]:

$$\log \langle e^V \rangle_0 = \langle V \rangle_0 + O(V^2)$$

we arrive at:  $-H' = \log Z_0 + \langle V \rangle_0$ .

## Appendix C: Calculation of inter-block spin interactions

Write the inter-block spin interaction  $V$  as a sum of nearest-neighbor block interaction:

$$V = \sum_{\langle I, J \rangle} V_{IJ}$$

Then

$$\langle V \rangle_0 = \sum_{\langle I, J \rangle} \langle V_{IJ} \rangle_0$$

The partial average of  $V_{IJ}$  can be calculated as follows (Fig. 2b):

$$\langle V_{IJ} \rangle_0 = \langle K S_3^J (S_1^J + S_2^J) \rangle_0 = 2K \langle S_3^J \rangle_0 \langle S_1^J \rangle_0$$

where  $S_3^J$  denotes the site spin 3 insider block spin  $S^J$ . The partial average of a site spin is given by:

$$\langle S_1^J \rangle_0 = \frac{e^{3K} + e^{-K}}{e^{3K} + 3e^{-K}} S^J$$

## References

- Bhaskar, R and Nigam, A 1990. Qualitative physics using dimensional analysis. *Artificial Intelligence* 45.
- Bradley, E and Zhao, F 1993. Phase space control system design. *IEEE Control Systems Magazine* 13.
- Creswick, R. J 1992. *Introduction to renormalization group methods in physics*. John Wiley. ISBN 0-471-60013-X.
- Kleer, Johande 1984. How circuits work. *Artificial Intelligence* 24.
- Forbus, Kenneth D 1984. Qualitative process theory. *Artificial Intelligence* 24.
- Goldenfeld, Nigel 1992. *Lectures on phase transition and the renormalization group*. Addison Wesley.
- Kuipers, Benjamin 1986. Qualitative simulation. *Artificial Intelligence* 29:289-338.
- Mavrovouniotis, M.L. and Stephanopoulos, G. 1988. Formal order-of-magnitude reasoning in process engineering. *Computer Chemical Engineering* 12.
- Nishida, Toyooki; Mizutani, Kenji; Kubota, Atsushi; and Doshita, Shuji 1991. Automated phase portrait analysis by integrating qualitative and quantitative analysis. In *Proceedings AAAI-91*.
- Onsager, L 1944. *Physical Review* 65.
- Raiman, Olivier 1991. Order of magnitude reasoning. *Artificial Intelligence* 51(1).
- Sacks, Elisha P. 1991. Automatic analysis of one-parameter planar odes by intelligent numerical simulation. *Artificial Intelligence* 48.
- Swendsen, Robert 1979. Monte Carlo renormalization group. *Physical Review Letters* 42(14).
- Weld, Daniel 1988. Comparative analysis. *Artificial Intelligence* 36.
- Williams, Brian 1984. Qualitative analysis of mos circuits. *Artificial Intelligence* 24.
- Wilson, Kenneth G. 1975. The renormalization group: critical phenomena and the kondo problem. *Reviews of Modern Physics* 47(4).
- Wilson, Kenneth G. 1983. The renormalization group and critical phenomena. *Reviews of Modern Physics* 55(3).
- Yakhot, Victor and Orszag, Steven 1986. Renormalization group analysis of turbulence: basic theory. *Journal of Scientific Computing* 1.
- Yip, Kenneth 1991. *KAM: A System for Intelligently Guiding Numerical Experimentation by Computer*. Artificial Intelligence Series. MIT Press.
- Yip, Kenneth 1993. Model simplifications in fluid mechanics. In *Proceedings of the 11th National Conference on Artificial Intelligence*.
- Zhao, Feng 1991. Extracting and representing qualitative behaviors of complex systems in phase spaces. In *Proceedings IJCAI-91*.

## Examination of Deep Knowledge in Knowledge Compilers

Shinji YOSHIKAWA\*, Akira ENDOU  
Power Reactor and Nuclear Fuel Development Corporation  
4002 Narita-cho, O-arai-machi, Ibaraki-pref. 311-13, Japan  
Telephone:+81-292-67-4141 ext. 3121  
Facsimile:+81-292-66-3868

Yoshinobu KITAMURA, Munehiko SASAJIMA, Mitsuru IKEDA,  
and Riichiro MIZOGUCHI  
The Institute of Science and Industrial Research  
Osaka University  
8-1, Mihogaoka, Ibaraki, 567 Japan  
Telephone:+81-6-877-5111 ext. 3565  
Facsimile:+81-6-877-4977  
E-mail:miz@ei.sanken.osaka-u.ac.jp

Length:4078 words(excluding the abstract and references)  
8 figures

### **Abstract**

A new method is proposed to examine a deep knowledge in knowledge compilers based on qualitative reasoning(QR)[1]. It is profitable if a knowledge compiler can examine a deep knowledge base to judge whether a thorough shallow knowledge can be generated or not, and can request the knowledge engineer to add some knowledge to the deep knowledge base if necessary, just like ordinary compilers check program source lists before generating executable objects and let the programmer know about defects of the source lists. Deep knowledge in QR-based knowledge compilers can be represented as simultaneous qualitative equations(SQEs). And the examination method reported in this paper is based on a structure analysis of a set of SQEs, and enables compilers to suggest necessary additional qualitative equation(QE)s. The way how this examination function works is explained with a sample model of a heat transport system of a nuclear power plant.

## Background

In the conventional numerical calculations, the device under consideration is analyzed using numerical models, and analysis accuracy can be improved as the models become sophisticated. However, such a method doesn't show us how the device behaviors are determined by its descriptions. For example, we cannot know "why this temperature increases when this pump stops". And we need another whole analysis to know "how this behaves if this pipe were 1cm shorter than it is".

In contrast with numerical analyses, qualitative reasoning can be viewed as aiming at "to make it clear how the device behaviors are determined by its descriptions, in exchange for a compromise on the behavior accuracy."

Advantages of qualitative reasoning [2][3][4] has been claimed that reasoning can be started from incomplete information, and all possible behaviors can be derived. And explanation generation capacity has been recognized as one of the greatest advantages. However, combinatorial explosion is the heaviest drawback and makes crucial tradeoffs with the advantages described above. There are many efforts to achieve disambiguation and to maintain explanation capability and simplicity of qualitative reasoning, mostly by combining qualitative reasoning/simulation with numerical processing. Within most of these researches, disambiguation is done by choosing one among all qualitative solutions by matching with numerically obtained result or human intuitive knowledge. [5] It is undoubtedly profitable if disambiguation process is explained also in a qualitative manner. In other words, qualitative reasoning will be much more effective if it shows why the other solutions should be denied as spurious ones, even when the conflicting change propagations have comparable effect [6]. For this reason, some systems employ additional qualitative constraints for disambiguation. However, identification of the additional constraints is done manually in an ad hoc manner. And, in case that qualitative reasoning is done using insufficient information, users may need to know what information can determine the system behavior uniquely. If a deep knowledge base is viewed as a set of qualitative constraint, there should be a clear condition to attain disambiguation. However, researches to identify this condition and to utilize this condition for examining deep knowledge are hardly found [7]. What the authors want to do in this research is to enable the system to examine initially given set of qualitative knowledge base and to suggest additional qualitative constraints useful for disambiguation of the solution. In this paper, an attempt to realize this suggesting function is presented. The proposed method deals with equilibrium equations. First, a qualitative model of a heat transport system of nuclear plant is shown, which has plural feedback loops, to emphasize the

importance of additional qualitative constraints. Then, mechanism of ambiguity is analyzed. Finally, the method to check the initial deep knowledge and to suggest additional knowledge effective for disambiguation is explained and demonstrated.

## A Qualitative Model of Heat Transport System of Nuclear Power Plant

A qualitative model of a heat transport system in a nuclear power plant is shown below.

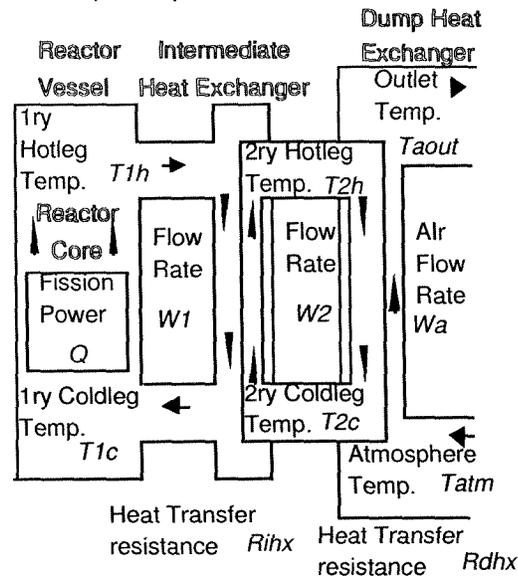


Fig.1 One loop Heat Transport System Model of A Nuclear Plant

The knowledge of change propagation in each component of the model, which can be primarily acquired easily, is shown below:

### Reactor

- 1ry hotleg coolant temperature ( $T_{1h}$ ) increases if:
- (1) 1ry coldleg coolant temperature ( $T_{1c}$ ) increases,
  - (2) reactor fission power ( $Q$ ) increases, or
  - (3) 1ry coolant flow rate ( $W_1$ ) decreases.

— QK-1

### Intermediate Heat Exchanger (IHx)

- 1ry coldleg coolant temperature ( $T_{1c}$ ) increases if:
- (1) 1ry hotleg coolant temperature ( $T_{1h}$ ) increases,
  - (2) 2ry coldleg coolant temperature ( $T_{2c}$ ) increases,
  - (3) 1ry coolant flow rate ( $W_1$ ) increases,
  - (4) 2ry coolant flow rate ( $W_2$ ) decreases, or
  - (5) heat transfer resistance of IHx ( $R_{ihx}$ ) increases.

— QK-2

- 2ry hotleg coolant temperature ( $T_{2h}$ ) increases if:
- (1) 1ry hotleg coolant temperature ( $T_{1h}$ ) increases,
  - (2) 2ry coldleg coolant temperature ( $T_{2c}$ ) increases,
  - (3) 1ry coolant flow rate ( $W_1$ ) increases,

(4) 2ry coolant flow rate(W2) decreases, or  
 (5) heat transfer resistance of IHX(Rihx) decreases. — QK-3

**Air Dump Heat Exchanger(DHX)**

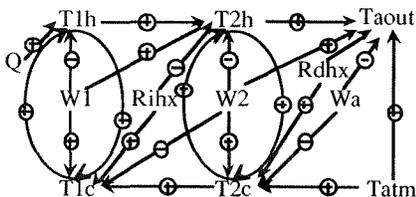
2ry coldleg coolant temperature(T2c) increases  
 if:(1) 2ry hotleg coolant temperature(T2h) increases,  
 (2) Atmosphere temperature(Tatm) increases,  
 (3) 2ry coolant flow rate(W2) increases,  
 (4) air flow rate(Wa) decreases, or  
 (5) heat transfer resistance of DHX(Rdhx) increases. — QK-4

Air outlet temperature(Taout) increases  
 if:(1) 2ry hotleg coolant temperature(T2h) increases,  
 (2) Atmosphere temperature(Tatm) increases,  
 (3) 2ry coolant flow rate(W2) increases,  
 (4) air flow rate(Wa) decreases, or  
 (5) heat transfer resistance of DHX(Rdhx) decreases. — QK-5

Now we can try to determine the qualitative value of the 1ry coldleg temperature(T1c) when the primary coolant flow rate(W1) decreases. However, too many qualitative value combinations of the endogenous parameters(T1h,T1c,T2h,T2c,Taout) are derived:----

([-],[+],[+],[+],[+]),([0],[+],[+],[+],[+]),([+],[+],[+],[+],[+]),  
 ([+],[0],[+],[+],[+]),([+],[+],[+],[+],[+]),([+],[+],[+],[+],[+]),  
 ([+],[0],[0],[0],[0]),([+],[+],[0],[0],[0]),  
 ([-],[+],[+],[+],[+]),([0],[+],[+],[+],[+]),  
 ([+],[+],[+],[+],[+]), and ([+],[+],[+],[+],[+]) ----from  
 which we get no meaningful result:

From causal network of this model shown in Fig.2, it is easily observed that this model has many feedback loops and that most of endogenous parameters cannot be causally ordered<sup>[8]</sup>. In case that a device under consideration has a particular characteristic like this model, qualitative reasoning can hardly be applied without additional qualitative constraints.



**Fig.2 Causal Network of the Heat Transport System Model**

**Knowledge Representation by Qualitative Equation**

Before the detailed discussions, representative equations has to be presented.

QK-1 means that:  
 a) if (T1h=[+]  
 or[0])and(Q=[+]  
 or[0])and(W1=[-]  
 or[0])

and(not (T1c=Q=W1=[0]))  
 then T1h=[+]  
 b)if (T1c=[-]  
 or[0])and(Q=[-]  
 or[0])and(W1=[+]  
 or[0])and(not (T1c=Q=W1=[0]))  
 then T1h=[-]  
 c)if (T1c=Q=W1=[0]) then T1h=[0]

These propositions can be transformed as:  
 a')(T1h=[0]  
 or[-])and(T1c=[+]  
 or[0])and(Q=[+]  
 or[0])and(W1=[-]  
 or[0])and(not(T1h=T1c=Q=W1=[0]))  
 IS FALSE  
 b')(T1h=[0]  
 or[+])and(T1c=[-]  
 or[0])and(Q=[-]  
 or[0])and(W1=[+]  
 or[0])and(not(T1h=T1c=Q=W1=[0]))  
 IS FALSE

In this paper, this set of propositions is represented as an qualitative equation as shown below:

$$[-]T1h+[+]T1c +[+]Q+[-]W1=0 \quad \text{--- QE-1}$$

This equation means that:  
 "there is at least 1 pair of terms having opposite signs, or all the terms are zero".

In case that the right-hand side is [+], this equation means that:  
 "At least one term is plus".

Above transformation of a qualitative knowledge(QK-1) into a qualitative equation(QE-1) is called transposition thereafter. And a set of qualitative knowledge can be represented as a set of simultaneous qualitative equations(SQE) by transposition. Knowledge representation in the form of SQE is convenient for visualizing the mechanism of ambiguity and for discussion of the method of disambiguation. Furthermore, this representation method leads to a new qualitative reasoning algorithm, which derives the solution(s) based on qualitative constraint satisfaction. This algorithm can deal both with local change propagation knowledge and other kind of qualitative constraint effective for disambiguation , in the same manner. From a viewpoint of explanation generation, this algorithm may have a disadvantage to the standard method of qualitative reasoning which traces change propagation along with causal network<sup>[1]</sup>. However, the focus of this paper is to analyze the given deep knowledge base, to check whether a thorough shallow knowledge can be obtained with satisfactory disambiguation, and to suggest about necessary additional qualitative knowledge, if any.

The qualitative knowledge from QK-1 through QK-5 can be represented as follows by transposition and in the form of matrix calculus similar to that of

ordinary linear equations:

$$\begin{pmatrix} [-] T_{1h} + [+] T_{1c} \\ [+] T_{1h} + [-] T_{1c} & + [+] T_{2c} \\ [+] T_{1h} & + [-] T_{2h} + [+] T_{2c} \\ & [+] T_{2h} + [-] T_{2c} \\ & [+] T_{2h} & + [-] T_{aout} \end{pmatrix} + \begin{pmatrix} [+] Q + [-] W_1 \\ & [+] W_1 + [+] R_{ihx} + [-] W_2 \\ & [+] W_1 + [-] R_{ihx} + [-] W_2 \\ [+] T_{atm} & & [+] W_2 + [+] R_{dhx} + [-] W_a \\ [+] T_{atm} & & [+] W_2 + [-] R_{dhx} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

**Fig. 3 Initial Qualitative Knowledge about the Heat Transport Model Represented as a Set of Simultaneous Qualitative Equations**

In the above figure, parameters affected by no other parameter in the whole model are classified as exogenous parameters, and parameters affected by other parameter(s) in this model are classified as endogenous parameters.

**Mechanism of Ambiguities**

In order to identify the method for checking initially constructed SQEs and for suggesting about additional QEs effective for disambiguation, We need to observe and formulate ambiguity generation mechanism.

In QE-1 and QE-2 as follows:

$$\begin{aligned}
 &[-]T_{1h} + [+]T_{1c} & + [+]Q + [-]W_1 & = 0 & \text{--- QE-1} \\
 &[+]T_{1h} + [-]T_{1c} + [+]T_{2c} & + [+]W_1 + [+]R_{ihx} + [-]W_2 & = 0 & \text{--- QE-2}
 \end{aligned}$$

,suppose that only W1 becomes [-] among the exogenous parameters. Then QE-1 and 2 are transformed as follows for the endogenous parameters:

$$\begin{aligned}
 &[-]T_{1h} + [+]T_{1c} & & = [-] & \text{--- QE-1'} \\
 &[+]T_{1h} + [-]T_{1c} + [+]T_{2c} & & = [+] & \text{--- QE-2'}
 \end{aligned}$$

QE-2' is always satisfied as long as QE-1' is satisfied. In other words, QE-2 cannot contribute to suppress the ambiguity when reasoning about W1 deviation.

This observation can be generalized as:  
*"In case that all the endogenous terms and one term of exogenous parameter Ve of QE-m are involved in another equation QE-n, QE-n does not represent any effective constraint when Ve deviates."*

---- Condition-1  
 Condition-1 can be used for checking ambiguities specific to particular parameter's deviation. Now we need another method for checking common ambiguities among all the

exogenous deviation. In numerical calculus, it is apparen that a set of independent equations represents enough constraint if the set is as large as the number of the endogenous parameters, and that no endogenous parameter can deviates when no exogenous parameter deviates. However, this is not applicable to qualitative calculus. For example, all the endogenous parameters in Fig.3 can have qualitative values:([+],[+],[+],[+],[+]) when the exogenous parameters take [0]. It can be proven that some endogenous parameter is allowed to deviate when no exogenous parameter deviates, this parameter cannot have unique qualitative value against any exogenous deviation. Therefore, a necessary condition for SQE to suppress the ambiguity is:

*"None of the endogenous parameters can be other than zero when no exogenous parameter deviates."*  
 ---- Condition-2

It is worth emphasizing that Condition-1 is to inhibit a par of spurious solutions specific to each exogenous deviation, and that Condition-2 is to inhibit common spurious solutions for all exogenous deviations. These 2 conditions can be used to examine originally constructed SQE, and to suggest the user/KE to add QEs, by showir candidate QEs which can suppress the ambiguity. The detailed procedures are shown in the next section.

**Suggestion Method of Additional QEs**

Condition-1 can be used for checking independent effectiveness of a particular QE's qualitative constraint, in comparison to another QE. On the other hand, Condition 2 is for checking the whole SQE. If Condition-2 is appliced first for checking the initially given SQE, followed by filteri by Condition-1, Condition-2 must be applied again, because filtering by Condition-1 can throw away QEs necessary for satisfying Condition-2.

Therefore, Condition-2 has to be checked and satisfied the set of QEs after filtered by Condition-1. The propose method to suggest additional QEs consists of 2 steps: 1)excluding QEs which cannot contribute to suppressior of ambiguity when one particular parameter deviates,applying condition-1, 2)derivation of candidate additional QEs to inhibit nonzero qualitative values for the endogenous parameters when no exogenous parameter deviates. And Condition-1 is checked whenever a new C is added.

It should be noticed that excluded QEs by Step-1 should be included in knowledge compilation because these can contribute for disambiguation when deviated exogenous parameter differs from one corresponding to Ve in Condition-1.

**Demonstration of Suggestion about Adding QI**

In this section, the method for suggesting about adding QEs is demonstrated about the heat transport system model, assuming that only QK-1 through QK-5 representing local change propagations are initially included in the deep knowledge.

The procedures of these 2 steps are explained below for the heat transport model of a nuclear power plant describ before:

**Step-1)**

About exogenous parameter  $T_{atm}, Q, R_{ihx}, R_{dhx}$  and  $W_a$ , there is no pair equations to match the condition-1.

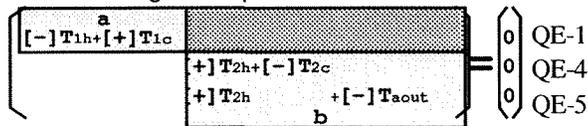
About  $W_1$ , QE-2 represents no additional constraint to QE-1. then QE-2 is excluded.

About  $W_2$ , QE-3 represents no additional constraint to QE-4. then QE-3 is excluded.

Then QE-1,4, and 5 are handed to Step-2.

**Step-2)**

When no exogenous parameter deviates, the SQE for the endogenous parameters are as follows:



**Fig. 4 SQEs of QEs 1, 4, and 5**

If the block-b) satisfies Condition-2, the dark shaded area can be neglected to judge whether the whole matrix satisfies Condition-2. Only block-a) has to be judged. Thus it is seen that a SQE satisfies Condition-2 if and only if every diagonal block satisfies Condition-2. In this case, examination of Condition-2 need to be done on 3 blocks.

**Examination of diagonal block-b) in Fig.4:**

Solutions of :  
 $[+]T2h + [-]T2c = [0]$ , and  
 $[+]T2h + [-]Taoout = [0]$   
 are  $([-], [-], [-])$ ,  $([0], [0], [0])$ , and  $([+], [+], [+])$ .  
 The candidate QEs to inhibit  $([-], [-], [-])$  and  $([+], [+], [+])$  are:  
 $[+]T2h \quad +[?]Q + \dots + [?]Wa = 0$ ,  
 $[+]T2c \quad +[?]Q + \dots + [?]Wa = 0$ ,  
 $[+]T2h + [+]T2c \quad +[?]Q + \dots + [?]Wa = 0$ ,  
 $[+]T2h \quad +[+]Taoout + [?]Q + \dots + [?]Wa = 0$ ,  
 $[+]T2c + [+]Taoout + [?]Q + \dots + [?]Wa = 0$ , and  
 $[+]T2h + [+]T2c + [+]Taoout + [?]Q + \dots + [?]Wa = 0$ ,  
 Note that  $[?]$  stands for any of  $[-], [0]$ , and  $[+]$  and that description " $[?]Q + \dots + [?]Wa$ " in each of above list means :  
 $"[?]Q + [?]W_1 + [?]R_{ihx} + [?]W_2 + [?]R_{dhx} + [?]T_{atm} + [?]W_a"$ .

Therefore, these candidate EQs are not concrete yet. For example, the first candidate EQ can match many EQs:  $[+]T2h + [-]Q = 0$ ,  $[+]T2h + [+]Q + [-]T_{atm} + [-]W_a = 0$ , and so on. Each candidate in the above list can match  $3^7 = 2187$  possible concrete EQs. In this sense, candidate EQs like those in the above list are called just "type" of EQ hereafter. What the user has to do at this point is to inspect the candidate EQ types one by one, carefully suspecting whether he/she has a qualitative knowledge to match the type.

In this case, the following knowledge matches the third type:

"About Air flow of DHX,

heat reduction rate by air flow in DHX is proportional to (air outlet temperature - atmosphere temperature)\*(air flow rate)

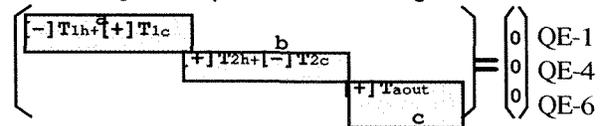
— QK-6

, and the heat reduction rate is equal to the fission power of the reactor".

, and the following QE is obtained by transposition:  $[+]T_{aout} + [-]T_{atm} + [-]Q + [+]W_a = 0$

— QE-6

It is important that the new set of SQEs consisting QE-1,4,5, and 6 has to be examined about Condition-1, because constraint of newly added QE-6 can make some of the constraints of QE-1,4, and 5 ineffective about some exogenous deviation. In this case, QE-5 is excluded from the SQE for the examination about Condition-2, and the SQE for the endogenous parameters change as follows:



**Fig. 5 SQEs of QEs 1, 4, and 6**

In this case, diagonal blocks-b) and a) have to be examined, because diagonal block-c) apparently satisfies Condition-2, and cannot be excluded by Condition-1 when new EQs are added in diagonal blocks-b) and a).

**Examination of diagonal block-b) in Fig.5:**

The solutions to inhibit are  $([-], [-])$  and  $([+], [+])$ . The candidate QEs to add are:

$[+]T2h \quad +[?]T_{aout} + [?]T_{atm} + [?]Q + \dots + [?]W_a = 0$ ,  
 $[+]T2c + [?]T_{aout} + [?]T_{atm} + [?]Q + \dots + [?]W_a = 0$

, and  
 $[+]T2h + [+]T2c + [?]T_{aout} + [?]T_{atm} + [?]Q + \dots + [?]W_a = 0$ .

This time, the user can be expected to pick up the following knowledge from his/her brain when checking the third QE type in the above list :

"About DHX,

energy transport rate through DHX is proportional to  $(2ry \text{ average temperature} - \text{air side average temperature}) / (\text{heat transfer resistance of DHX})$

— QK-7

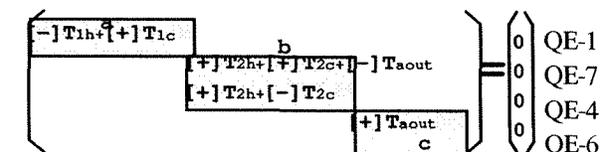
, and the heat transport rate is equal to the fission power"

, and transposed QE is:

$[+]T2h + [+]T2c + [-]T_{aout} + [-]T_{atm} + [-]Q + [-]R_{dhx} = 0$

— QE-7

, and no EQ is excluded by Condition-1 this time, and the SQE to be examined is now:



**Fig. 6 SQEs of QEs 1, 7, 4, and 6**

**Examination of diagonal block-a) in Fig.6:**

The solutions to inhibit are again (-,-) and (+,+). The candidate QEs to add are:

$$[+]T1h+ [?]T2h+[?]T2c+[?]Taout+...=0,$$

$$[+]T1c+[?]T2h+[?]T2c+[?]Taout+...=0, \text{ and}$$

$$[+]T1h+[+]T1c+[?]T2h+[?]T2c+[?]Taout+...=0.$$

This time the following knowledge can be reminded by the user when checking the third type:

"About IHX,

energy transport rate through IHX is proportional to (1ry average temperature - 2ry average temperature)/(heat transfer resistance of IHX)

— QK-8

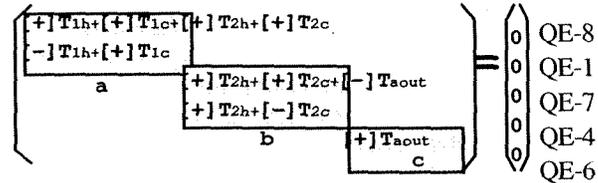
,and the energy transport rate is equal to the fission power."

and transposed EQ is:

$$[+]T1h+[+]T1c+[-]T2h+[-]T2c+[-]Q+[-]Rihx=0$$

— QE-8

, and no EQ is excluded by Condition-1 this time, and Condition-2 is finally satisfied by the set of SQEs:



**Fig.7 SQEs of QEs 8,1,7,4, and 6**

The final set of QEs resulted from above procedures are:QE-1,4,6,7, and 8, and whole SQE for reasoning consists of QE-1,2,3,4,5,6,7, and 8. Ambiguity is drastically reduced by newly added EQs(:QK-6,7, and 8) as follows.

Deviation of dependent variables	T	T	T	T	T
	1	1	2	2	a
	h	c	h	c	o
exogenous deviation					t
Tatm=[+]->	(+)	(+)	(+)	(+)	(+)
Q=[+]->	(+)	(+)	(+)	(+)	(+)
W1=[+]->	?	?	?	?	?
Rihx=[+]->	?	?	?	?	?
W2=[+]->	?	?	?	?	?
Rdhx=[+]->	(+)	(+)	(+)	(+)	?
Wa=[+]->	(-)	(-)	(-)	(-)	(-)

Reasoning result by initially constructed SQE



Deviation of dependent variables	T	T	T	T	T
	1	1	2	2	a
	h	c	h	c	o
exogenous deviation					t
Tatm=[+]->	?	?	?	+	+
Q=[+]->	+	+	+	+	+
W1=[+]->	-	+	0	0	0
Rihx=[+]->	+	+	0	0	0
W2=[+]->	?	?	-	+	0
Rdhx=[+]->	+	+	+	+	0
Wa=[+]->	?	?	?	-	-

Reasoning result by enlarged SQE

**Fig. 8 Disambiguation Effect of Additional Qualitative Equations**

(note: (+) and (-) mean that these qualitative signs are determined only when reasoned along with the causal orders. This method assumes that the device is in perfectly normal condition before the assumed exogenous deviation, in contrast with that derived informations from SQE of the device's equilibrium state after the assumed deviation are independent of the prior states to the assumed deviation.)

**Discussions**

1) Conditions of qualitative matrix for disambiguation

It has been demonstrated that conditions-1 and 2 can be used to suggest necessary additional qualitative equations, but still have a weak point. There is one more qualitative knowledge as follows, which has not been picked up by the proposed method:

"About 2ry loop,

Energy transport rate through 2ry loop is proportional to (2ry hotleg coolant temperature - 2ry coldleg coolant temperature)\*(2ry coolant flow rate),

and the energy transport rate is equal to the fission power".

And, this knowledge suppresses the remaining ambiguities of the resulting SQE about deviations of  $T_{atm}$  and  $W_a$ .

This means that satisfying Conditions-1 and 2 is not enough to suppress the reasoning ambiguity, at least by the algorithm briefed in the section of "Knowledge Representation by Qualitative Equation".

More conditions of SQE structure to suppress ambiguity have to be identified, and, if possible, it is desirable that an efficient algorithm is available to judge all the conditions.

### 2) Identification of the right QE from suggested QE types

In the demonstration, the way to select right QE type and to identify right QE to match the type was not discussed. It is definitely impossible to derive concretely the right QE based only on the initially constructed knowledge base and the examination or reasoning algorithm. The purpose of this research is to support users of knowledge compilers in building more effective deep knowledge bases more easily. The method proposed in this paper can help to limit the space of knowledge to search for effective disambiguation, although this method cannot guarantee the successful knowledge building. And, there is still some room for improvements. First, parameter combinations in the suggested equation type can be limited based on the device topology. For example, about the heat transport system model shown before, if a suggested equation type has only  $T_{1h}$  and  $T_{aout}$ , excluding all the parameters in the secondary loop, this QE cannot represent naive knowledge and can be thrown away from the suggestion list. Second, suggested equation type can be more intuitive by introducing physical dimensions. In general, equations to represent physical laws are expressed in a form like " $PV=nRT$ " or " $F=ma$ ". Physical dimensions can be useful to transform the suggested type of QE into the form like "<Product of Parameters is equal/proportional to Product of Parameters>". For example, about the heat transport system model again, a suggestion like "Do you have any knowledge to show that ' $Q[J/sec]$  are positively proportional to <product of  $T_{1h}[K]$  and  $W_1[kg/sec]$ > and inversely proportional to <product of  $T_{1c}[K]$  and  $W_1[kg/sec]$ >?' " is expected to much more understandable than the other one like "Do you have any knowledge to show that ' $[+T_{1h}+[-]T_{1c}+[+]W_1+[-]Q=0$ ' ?"

### 3) Possibility to support another knowledge category construction

It has been considered that deep knowledge for knowledge compilation has to consist of a) physical principles and commonsense causalities, and b) informations of the device's topology. Category a) can be commonly used at least in the specific domain, and category b) is thought to be specific to the object device. However, category b) is still

common in the sense that behaviors of devices having same configuration can differ to each other. This implies that a new knowledge category is necessary to reason particular device behavior without excessive ambiguity. Examples are demonstrated below:

In the previous heat transport system of a nuclear power plant, qualitative influence of 2ry coolant flow rate on 1ry coolant temperature cannot be determined from local change propagations, general principles and device topologies. This qualitative influence depends on initial plant heat balance. In other words, this qualitative influence depends on individual quantitative specifications. After complete condition of SQE is identified, the function to suggest necessary additional qualitative knowledge can derive the criterion to determine whether the influence is positive or negative. And it will be possible to efficiently construct a knowledge base for *qualitative knowledge depending on individual quantitative specifications*. Let us consider another example. In general, heat transfer coefficient is dependent on Reynolds number. Suppose that the secondary coolant were gas, which makes the net heat transfer ratio much more dependent on Reynolds number,. If so, both of the 2ry hotleg and coldleg temperature can decrease if 2ry flow rate increases. In this case, suggestion function of additional qualitative equations enables the user to make the qualitative model of heat exchanger more common, and to describe additional qualitative constraint dependent on whether the fluids are liquid or gas into a knowledge base for *qualitative knowledge depending on individual quantitative specifications*.

In case that it is necessary to develop a deep knowledge base for another similar plant, this new category can be separated, to enhance reusability of the deep knowledge base. As seen so far, this function improves re-usability of the whole deep knowledge base.

### Conclusion

A method to examine deep qualitative knowledge and to suggest necessary additional knowledge has been proposed and demonstrated. Knowledge representation by qualitative equations plays an important role in this method. Currently derived conditions of simultaneous qualitative equations for disambiguation is still incomplete, Followings are thought to be the main works to reach the next step of this research:

#### 1) Qualitative mathematics

In order to identify the complete condition of SQE for disambiguation, an integrated theory of qualitative mathematics has to be developed. As seen so far, it can happen that a set of SQEs larger than the number of endogenous parameters still remains under constraining. And, if a QE is viewed as a set of signs in an ordinary linear equations, a set of SQEs larger than the number of endogenous

parameters involves some information about absolute values of the coefficients. Theory of qualitative mathematics has to deal with these special characteristics of SQEs.

## 2) Reasoning algorithm modification

In order to make sense of establishing a new knowledge category for *qualitative knowledge depending on individual quantitative specifications*, priorities among QEs have to be considered during the reasoning procedures. Knowledge used in the explanations generated along with reasoning process has to be shifted to deeper ones. In other words, shallow knowledge derivable from general principles only must not be explained by individual quantitative specifications.

After these works are completed, knowledge compilers based on QR will be greatly enhanced both in effectiveness of shallow knowledge to be generated, and in reusability of deep knowledge base, mainly due to being less dependent on heuristic approaches.

## Acknowledgement

And the authors are greatly indebted to Mr. K. Suda of Power Reactor and Nuclear Fuel Development Corporation, who provide his original heat balance calculation and visualization software to enable the authors to verify the qualitative reasoning system.

## References

- [1] Hirai, K. and Mizoguchi, R. et al.: "The organization of the domain knowledge oriented to shareability - Domain-specific Tool Based on the Deep Knowledge (KCII-DST)", *Proceedings of the 5th Annual Conference of JSAI*, Vol.1, pp.325-328, 1991 (in Japanese)
- [2] de Kleer, J. and Brown, J. S.: "A Qualitative Physics Based on Confluences", *Artificial Intelligence*, Vol.24, pp7-83, 1984
- [3] Benjamin Kuipers: "Qualitative Simulation", *Artificial Intelligence*, 29:289-338, 1986.
- [4] Kenneth D. Forbus: "Qualitative Process Theory", *Artificial Intelligence*, 24:85-168, 1984
- [5] Yoshida, K. and Motoda, H.: "An Approach to Hierarchical Qualitative Reasoning"
- [6] O. Raiman, "Order of Magnitude Reasoning," AAAI-86
- [7] Jean-Luc Dormoy, et al.: "Assembling a Device", *Proceedings of AAAI-88*, pp.330-335, 1988
- [8] Iwasaki, Y. and Simon, H. A.: "Causality in Device Behavior", *Artificial Intelligence*, Vol.29, pp3-32, 1986

# Intelligent Computing About Complex Dynamical Systems\*

Feng Zhao

Laboratory for Artificial Intelligence Research and  
Department of Computer and Information Science

The Ohio State University

Columbus, OH 43210 U.S.A.

E-mail: `fz@cis.ohio-state.edu`

## Abstract

We develop computational mechanisms for intelligently simulating nonlinear control systems. These mechanisms enhance numerical simulations with deep domain knowledge of dynamical systems theory and control theory, a qualitative phase-space representation of dynamical systems, symbolic and geometric manipulation capabilities, and a high-level interface. Programs equipped with these capabilities are able to autonomously simulate a dynamical system, analyze the simulation results, and utilize the analysis to perform design tasks. We demonstrate the mechanisms with an implemented computational environment called the Control Engineer's Workbench.

**Keywords.** Qualitative reasoning, scientific computing, numeric/symbolic processing, control system design.

## Introduction

*The purpose of computing is insight, not numbers.*

— R. W. Hamming

Computationally simulating complex physical systems in engineering design has become a common practice. Yet most of today's simulations rely entirely on extensive numerical computations and laborious human analysis. Human engineers have to shoulder the burden of translating physics and constraints into models, preparing numerical simulations, interpreting consequences of the experiments, and performing design tasks. Moreover, nonlinear systems can exhibit extremely complicated behaviors that defy human analysis and pure numerical simulations. The complexities of these systems are largely due to nonlinearities, high

dimensionality, and uncertainties of the systems and environments the systems operate in.

The difficulties in the traditional engineering simulation arise from the lack of (1) parsimonious representations capturing the essence of physical systems and amenable to efficient computations, (2) efficient modeling algorithms for constructing the representations, and (3) effective reasoning methods that can use the representations to compute and synthesize useful properties for the systems. The lack of computable representations for physical dynamical systems hinders the exploitation of the special properties of the systems and the attainment of the maximum performance for the design. The simulation and design of the dynamical systems are limited by the available computational power and the complexities of the systems.

While the traditional numerical computing has successfully attacked many practical problems, we can greatly enhance its effectiveness and significantly expand the scope of what can be done with this style of engineering computing by integrating the numerical computation with advanced artificial intelligence technology and symbolic computing methods. For example, programs equipped with AI reasoning techniques and deep domain knowledge have already helped engineers solve an open problem in hydrodynamics [Yip, 1991], given new insights into behaviors of a heart model in cardiology [Sacks & Widman, 1993], and designed a high-performance nonlinear controller in maglev engineering [Zhao & Thornton, 1992]. Abelson *et al.* described a collection of computer programs that analyze dynamical systems at the level of expert dynamicists [Abelson, 1989]. Other related work is discussed in [Nishida *et al.*, 1991; Bradley, 1992; Kant *et al.*, 1992; Amador, Finkelstein & Weld, 1993].

## Task Domain: Control System Simulation and Design

We study the analysis and design of nonlinear control systems. A real-world control system is a complex closed-loop system with extremely rich dynamics. Computation and reasoning are pervasive in the design and operation of the controller. Sensors collect a large

---

\*This research was supported in part by the National Science Foundation grants CCR-9308639 and MIP-9001651, and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-89-J-3202. An early version of this paper appeared in *Mathematics and Computers in Simulation*.

amount of quantitative information. State and parameter estimators infer hidden information about the system from the sensed data. The system is modeled with a representation appropriate for further analysis and design based on available information. The model is then analyzed to extract behaviors that are considered significant for the control objective. To meet the control objective, a control law is synthesized to change the natural dynamics of the system.

The domain of automatic control brings together issues of sensing, estimation, control synthesis, and control execution. The study of their common themes—computation and reasoning—serves as a framework for coherently addressing these issues and makes it possible to employ advanced computational techniques to drastically improve modern control design. We focus on the control synthesis that *maps* a model of a physical system together with some control objective to a synthesized control law:

*Control Design:*

model + control objective  $\mapsto$  control law.

A control engineer goes through the following design steps to synthesize a controller for a given dynamical system:

1. *Analysis:* analyze the model of the dynamical system. The physics and the constraints of the system are often modeled with a quantitative mathematical model, typically a set of differential equations. This step examines the model and analyzes the behaviors of the system.
2. *Design:* design a controller for the system. Based on the analysis, this step arrives at a control design according to the prespecified control constraints.
3. *Verification:* verify the control design. This step ensures that the design meets the control specification.

The steps 1, 2, and 3 of the above design procedure are often iterated before a reasonable control law is synthesized. Except for very few cases in which analytic-form solutions are available, computer simulations are the main tools for analyzing nonlinear systems and for designing and verifying the controllers.

Existing control simulation softwares are inadequate for automatically designing highly complex nonlinear systems. Commercially available programs like MATLAB and SIMULAB [MathWorks, 1989] rely on numerical simulations. These programs are, at their very best, semi-automatic and serve as interactive design aids to human engineers. Although they are equipped with elaborate graphic interfaces, these programs provide only fragmented, limited capabilities such as integration and root finding for performing the simulation task; human users need to prepare the simulation and to interpret the result. The specialized control toolboxes embedded in these programs are “shallow” expert systems; they lack deep domain knowledge and do

not have mechanisms for computationally representing and manipulating a control design.

## Smart Simulation of Dynamical Systems

We address the difficulties of traditional numerical computing by developing computer representation and simulation technologies necessary for enabling programs to autonomously perform and interpret numerical simulations of control systems. The design of complex control systems requires powerful computational mechanisms to represent, reason about, and manipulate the dynamics of nonlinear systems. We demonstrate that difficult control synthesis tasks can be automated, using a computational workbench that actively exploits knowledge of nonlinear dynamics and phase space. More specifically, we develop a computer representation for dynamical systems in terms of qualitative, geometric phase-space features and equivalence classes of behaviors. We employ hybrid computation integrating symbolic and numerical methods with AI reasoning and representation techniques and exploiting sophisticated mathematical domain knowledge. We provide a high-level interface for communicating the result of analysis in qualitative terms and for visualizing the phase-space dynamics.

We have constructed a computational environment, the Control Engineer's Workbench, integrating a suite of programs that automatically analyze and design high-performance, global controllers for a large class of nonlinear systems using the qualitative phase-space representation [Zhao, 1992]. Given a model of a physical system and a control objective, the Control Engineer's Workbench analyzes the system and designs a control law achieving the control objective. A user typically interacts with the Workbench in the following way.

The user first tells the Workbench about the system: he inputs a system model in terms of an ordinary differential equation, parameter values, and bounds on state variables for analysis in the form of a phase-space region. The user also tells the Workbench about the requirements on the control design: he specifies the desired state for the system to settle in, the initial states of the system, the allowable control parameter values, and the constraints on the control responses.

The user then asks the Workbench to analyze the system within the parameter ranges of the model. The Workbench visualizes the totality of the behaviors of the system over the parameter ranges; it represents the qualitative aspects of the system in a data structure and reports to the user a high-level, symbolic summary of the system behaviors and, if necessary, a graphic visualization of the phase-space qualitative features.

Next, the user instructs the Workbench to synthesize a control law for the system, subject to the specified design requirements. The Workbench searches for the global control paths that connect the initial

```

equation_of_motion:
  {
    dx1/dt = x2
    dx2/dt = -p1x1 - p2x1^3 - p3x2 + u
  }
state_variable:    x1
state_variable:    x2
parameter:         p1 = -2.0
parameter:         p2 = 1.0
parameter:         p3 = 0.2
bounding_box:      x1 ∈ [-3.0,3.0],
                  x2 ∈ [-4.0,4.0]
goal_state:        (0.0, 0.0)
initial_state:     (-1.0, -3.0)
range_of_control:  u ∈ [-0.2,0.2]

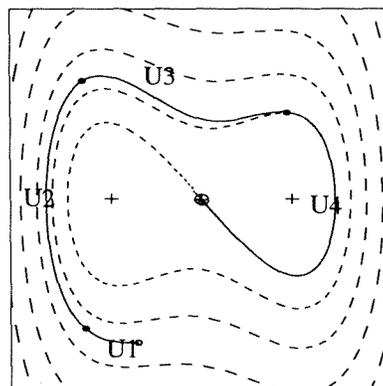
```

Figure 1: Workbench Input: the model for a buckling steel column and the control objective of stabilizing the buckling motion.

states of the system and the desired goal state, using the qualitative description about the system. More specifically, the search is conducted in a collection of discrete entities representing trajectory flows in phase space. After the global control paths are established, the Workbench determines the controllable region of the system and the switching surfaces where control parameters should change values. A synthesized trajectory reference consists of a sequence of trajectory segments, each of which is under a constant control.

The following example illustrates how the Workbench autonomously analyzes the buckling motion of a steel column under compression and synthesizes a control law to stabilize the motion. Figure 1 shows the input to the Workbench—the model and control objective—and Figure 2 reports the synthesized control reference trajectory superimposed on the phase space of the column. The global portion of the reference trajectory shown in Figure 2 consists of four segments, each of which starts at a switching state marked as a small circle; the reference trajectory connects the initial state with the goal state at the origin of the phase space. The control parameter value is held constant for each segment denoted by  $U_1$ ,  $U_2$ ,  $U_3$ , and  $U_4$ , respectively.

The analysis of the nonlinear dynamics of the buckling column accounts for a large percentage of the computation in arriving at the desired control design. The equation of motion for the column is described as a nonlinear ordinary differential equation. An exhaustive numerical simulation can be very expensive. The Workbench employs a geometric analysis of dynamical systems that uses a phase-space representation to capture the qualitative features of the system, first developed by Poincaré at the beginning of the century.



```

;; The Synthesized Control Law specifying the time
;; instance, switching state, and corresponding
;; control value for each switching:
((time 0.)
 (switching-state #(-1 -3))
 (control .2))
((time .284)
 (switching-state #(-1.82 -2.71))
 (control 0.))
((time 1.06)
 (switching-state #(-1.86 2.49))
 (control -.2))
((time 2.71)
 (switching-state #(1.35 1.82))
 (control 0.))
((time 6.76)
 (switching-state #(-.0023 -.0692))
 (control *local-control*))

```

Figure 2: Workbench Output: the control reference trajectory and the control law for stabilizing the column. In the plot, the reference trajectory is drawn in solid lines, and the phase-space stability boundaries of the uncontrolled column in dashed lines.

## Representation, Simulation, and Interface

The Workbench has demonstrated the following capabilities in designing a control law for the buckling column:

- Deciding what behaviors are significant. The Workbench looks for qualitative features like equilibrium points, stability regions, and trajectory flows.
- Describing the behaviors qualitatively in computational terms. The Workbench models the stability regions and trajectory flows geometrically.
- Reasoning about the geometry of a phase space.
- Performing the control design autonomously.

These capabilities are supported in the Workbench by (1) qualitative representation consisting of dimension-independent geometric constructs, (2) hierarchical extraction of the behaviors, (3) modeling and

manipulation mechanisms for trajectory flows, and (4) algorithms implementing the geometric, combinatorial, and numerical computations.

Our intelligent simulation environment has three generic layers: (1) computer representations of the physical world, (2) simulation technology that makes feasible the computation about the physical world, and (3) high-level interfaces for communications between human users and the computer. For our intended task domain of control design and analysis, we present the following structure for our environment:

#### 1. Computer representation:

We develop a qualitative representation describing a dynamical system in terms of its phase-space geometric features: the qualitative phase-space structure. This qualitative representation captures asymptotic and transient behaviors of a dynamical system and essential constraints of the system useful for the control synthesis task.

#### 2. Simulation technology:

Our simulation of a control system comprises numerical and symbolic computation about the system model and geometric modeling of phase space. The Workbench also embodies deep domain knowledge of dynamical systems theory and control theory that can guide quantitative simulation and reduce the amount of computation necessary for analyzing the system. We choose Scheme, a dialect of LISP, as the implementation language for the Workbench [Hanson, 1991] and use extensively the Scheme mathematical library supporting generic numerical and symbolic manipulations. Scheme supports procedural abstraction that facilitates the extraction of common patterns in numerical computation and the composition of numerical procedures.

#### 3. High-level interface:

The Workbench presents a high-level, qualitative description of the result of its analysis and design to human designers; this level of interaction is more intuitive and direct than that of pure quantitative presentation. Other programs in the Workbench can efficiently access and manipulate the result. The internal data structure for the analysis ensures that the result is sensible to human engineers and manipulable by other programs.

The rest of the paper describes our representation for control systems and an implemented simulation environment.

## Representing and Manipulating Constraints of a Control Design

The complexity of a nonlinear system necessitates the need for a design vocabulary capable of describing implicit constraints of a control design and providing means to manipulate and reason about these constraints and to build abstractions. We present a com-

putational mechanism that allows one to represent and manipulate constraints of a control design in terms of phase-space geometry and topology of a dynamical system. A control design will be specified in terms of the composition of geometric objects in phase space.

## A qualitative representation

We have interpreted a control design as a *mapping* from the model of a physical system to be controlled and the control objective to the control law, under the influence of which the physical system will behave in the desired way. The synthesized control law alters natural dynamics of the system through the selection and composition of the natural behaviors.

We describe a qualitative representation for complex behaviors of dynamical systems in phase space and a design vocabulary for computationally expressing and manipulating these behaviors [Zhao, 1991; Zhao, 1993]. A phase space of a dynamical system is an  $n$ -dimensional geometric space, each dimension of which represents a state variable of the system. We are interested in representing the qualitative behaviors of dynamical systems for control analysis and design. One useful qualitative representation of the phase space of a dynamical system is in terms of equilibrium points and limit cycles, stability regions, trajectory flows exhibiting the same qualitative features, and the spatial arrangement of these geometric objects in phase space. This qualitative representation captures the gross aspects of dynamics in a relational graph of phase-space structure and a set of discrete objects called *flow pipes*—the equivalence classes of behaviors. The design vocabulary describes a control design task in terms of well-defined geometric, combinatorial operations on the flow pipes. This vocabulary formalizes aspects of implicit expert reasoning of control engineers in solving control design problems with the phase-plane method. The representation and the vocabulary are developed independently of the orders of systems, *i.e.*, the dimensionality of phase spaces.

## Designing control by manipulating equivalence classes of trajectories

The flow pipes group infinite numbers of distinct behaviors into a manageable discrete set that becomes the basis for establishing control reference trajectories; each flow pipe models an equivalence class of trajectories exhibiting similar qualitative features.

The geometric modeling of a phase space with flow pipes makes the phase-space control planning and navigation possible. Given a discrete set of possible control actions, the search for a control path from an initial state to a destination is a reachability problem, *i.e.*, the problem of finding a sequence of connected path segments each of which is under a single control action, as schematically illustrated in Figure 3. This point-to-point planning can be naturally described within the

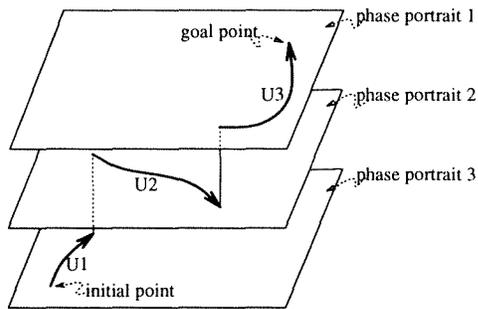


Figure 3: Search for a control path from an initial point to a goal point in a stack of phase spaces.

flow-pipe representation of phase space: a system under control jumps from one flow pipe to another upon the switching of control, eventually arriving at the destination.

To make this approach computationally feasible, the phase spaces of the dynamical system indexed by different control actions are first parsed into a discrete set of trajectory flow pipes. These flow pipes are then aggregated to intersect each other and are pasted together to form a graph, the *flow-pipe graph*. The flow-pipe graph is a directed graph where nodes are intersections of flow pipes and edges are segments of flow pipes. The initial state and the goal state are nodes in the graph. Each edge of the graph is weighed according to traveling time, smoothness, etc. With this graph representation, the search for optimal paths is formulated as a search for shortest paths in the directed graph.

### An Environment: the Control Engineer's Workbench

The Control Engineer's Workbench is an implemented system that analyzes and designs control systems and interacts with users qualitatively. The Workbench serves as an intelligent assistant to control engineers. The components of the Workbench are shown in Figure 4:

- MAPS program for simulation and interpretation
- Phase Space Navigator for control synthesis
- A graphic program for visualizing the design
- A user interface for communication with the system.

MAPS, standing for Modeler and Analyzer for Phase Spaces, is an autonomous phase-space analysis and modeling program that extracts and represents qualitative phase-space structures of nonlinear dynamical systems [Zhao, 1991]. MAPS generates a high-level description of the behaviors of a dynamical system, sensible to humans and manipulable by other programs. Phase Space Navigator visualizes the phase-space structure of a given system computed by MAPS,

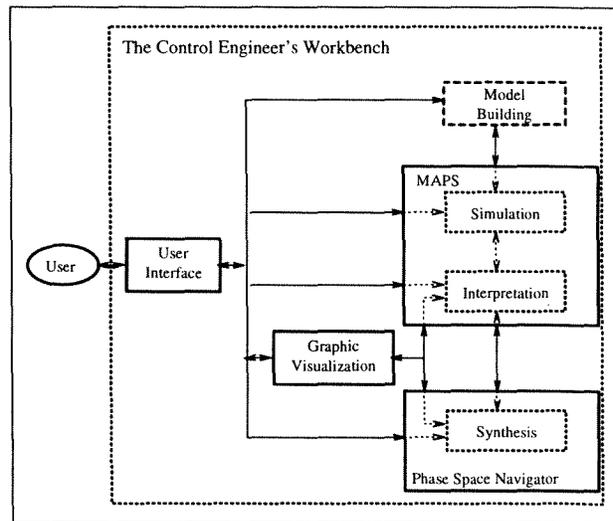


Figure 4: The Control Engineer's Workbench.

plans global control reference trajectories, and navigates the system along the planned trajectories [Zhao, 1992]. The component for model building in the figure has not yet been implemented.

The programs in the Workbench implement various algorithms: symbolic differentiation, numerical algorithms on differential equations, modeling of geometric structures, clustering of equivalence classes, graph algorithms, etc. The inference mechanism of the Workbench uses these programs to construct a qualitative phase-space structure for representing a system, to check for the consistency of the structure, and to reason about and manipulate the representation through a graph of flow pipes.

The flow of computation within the Workbench is illustrated in Figure 5. Given a model of a system, a bounded phase-space region of interest, allowable parameter values, and control objectives and constraints, the Workbench performs stability and trajectory flow analysis for the system in phase space and interprets the result in a phase-space graph. The Workbench then explores the control space to synthesize a desired control law subject to the design constraints. It reports the control law specifying reference trajectories and performance properties. The control design for steering towards an equilibrium is performed in this way: for a point-to-point control, the output is a reference trajectory, whose control law consists of a sequence of tuples of time, switching state and the corresponding control value; if an initial operating region is given, the output is a controllable region geometrically represented as a polyhedral structure.

The current implementation of the Workbench takes as input the model for a dynamical system in terms of an ordinary differential equation. Incorporating model formulation capability that constructs models

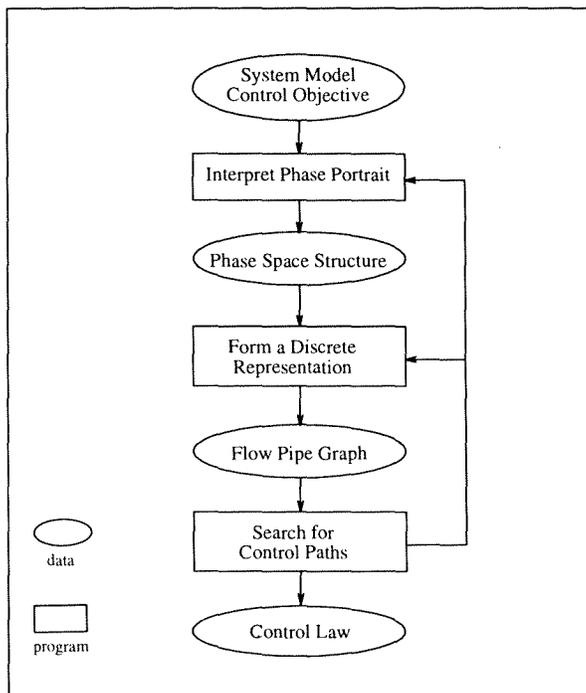


Figure 5: The flow of computation in the Workbench.

from physical principles or measurements can broaden the scope of control systems the Workbench can design. Although the analysis and design algorithm of the Workbench applies to dynamical systems of any order, the computational complexity of high-dimensional systems remains as a future research topic.

## Conclusion

We have developed the Control Engineer's Workbench comprising programs MAPS and Phase Space Navigator that automate a significant portion of a control engineer's simulation and design task. The Workbench employs computational means to represent and manipulate dynamics of control systems, using a qualitative representation for encoding dynamics and a flow-pipe based phase-space method for designing nonlinear controllers; it combines numerical simulation with symbolic techniques and AI reasoning; it provides an interface for visualizing the result of control design and analysis. The Workbench has been applied to the design of a nonlinear controller for a magnetic levitation vehicle. We plan to extend the capabilities of the Workbench to support interactive editing of phase-space geometric representation of dynamical systems for the purpose of experimenting and testing new ideas in control design.

The Control Engineer's Workbench complements and enhances human design activities. By providing manipulation and visualization mechanisms for the design, the Workbench relieves engineers from routine,

tedious low-level tasks of simulation and interpretation, allows the engineers to focus on higher-level design issues, and enlarges the design space the engineers can explore.

## References

- H. Abelson, M. Eisenberg, M. Halfant, J. Katzenelson, E. Sacks, G.J. Sussman, J. Wisdom, and K. Yip, "Intelligence in Scientific Computing." *CACM*, 32(5), May 1989.
- F. G. Amador, A. Finkelstein, and D. S. Weld, "Real-Time Self-Explanatory Simulation." *Proc. of QR-93*, 1993.
- E. Bradley, "Taming Chaotic Circuits." *Technical Report AI-TR-1388*, MIT Artificial Intelligence Lab, 1992.
- J. Guckenheimer and S. Kim, "Kaos: Dynamical System Toolkit with Interactive Graphic Interface." *Technical Report (Draft)*, Cornell University, 1990.
- C. Hanson, "MIT Scheme Reference Manual." *AI-TR-1281*, MIT Artificial Intelligence Lab, 1991.
- E. Kant et al. (eds.) *Intelligent Scientific Computation*. Working notes of AAAI Fall Symposium Series, 1992.
- The MathWorks, *SIMULAB, a program for simulating dynamic systems, a user's guide*. The MathWorks, Inc., 1990 and *MATLAB User's Guide*. The MathWorks, Inc., 1989.
- T. Nishida, K. Mizutani, A. Kubota, and S. Doshita, "Automated Phase Portrait Analysis by Integrating Qualitative and Quantitative Analysis." *Proc. of AAAI-91*, July 1991.
- M. F. Russo and R. L. Peskin, "Automatically Identifying the Asymptotic Behaviors of Nonlinear Singularly Perturbed Boundary Value Problems." *J. Automated Reasoning*, 8(3), June 1992.
- E. Sacks, "Automatic Analysis of One-Parameter Planar Ordinary Differential Equations by Intelligent Numeric Simulation." *Artificial Intelligence*, 48(1):27-56, 1991.
- E. Sacks and L. Widman, "Nonlinear Heart Model Predicts the Range of Heart Rates for 2:1 Swinging in Pericardial Effusion." *American Journal of Physiology*, 1993.
- K. M. Yip, *KAM: A System for Intelligently Guiding Numerical Experimentation by Computer*. MIT Press, 1991.
- F. Zhao, "Extracting and Representing Qualitative Behaviors of Complex Systems in Phase Spaces." *Proc. 12th Int'l Joint Conf. on Artificial Intelligence*, Morgan Kaufmann, 1991. To appear in *Artificial Intelligence journal*.
- F. Zhao, "Automatic Analysis and Synthesis of Controllers for Dynamical Systems Based on Phase Space

Knowledge." *Technical Report AI-TR-1385*, MIT Artificial Intelligence Lab, 1992.

F. Zhao and R. Thornton, "Automatic Design of a Maglev Controller in State Space." *Proc. of the 31st IEEE Conf. on Decision and Control*, Tucson, Arizona, December 1992.

F. Zhao, "Computational Dynamics: Modeling and Visualizing Trajectory Flows in Phase Space." *Annals of Mathematics and Artificial Intelligence*, 8(3-4), 1993.

## Author Index

Bell, David	1	Narayanan, N. Hari	197
Bobrow, Daniel	1	Pisan, Yusuf	205
Bradley, Elizabeth	13	Ramachandran, Sowmya	212
Brajnik, Giorgio	81	Saraswat, Vijay	1
Bredeweg, Bert	24	Sasajima, Munehiko	224,310
Cagan, Jonathan	295	Schut, Cis	24
Chandrasekaran, B.	276	Shen, Q.	234
Chi, Diane	36	Shirley, Mark	1
Clancy, Daniel J.	45	Shults, Benjamin	164
Compton, Paul	176	Steele, A. D.	234
Cunningham, Pdraig	90	Stefanelli, M.	144
de Koning, Kees	24	Struss, Peter	246
Doyle, Richard J.	55	Sun, Kun	256
Dressler, Oskar	63	Suwa, Masaki	197
Durrant-Whyte, Hugh F.	114	Tanaka, Hiroshi	266
Endou, Akira	224, 310	Thadani, Sunil	276
Falkenhainer, Brian	1, 98	Tsumoto, Shusaku	266
Faltings, Boi	69, 256	Washio, Takashi	286
Farquhar, Adam	81	Whalley, Peter B.	106
Finn, Donal P.	90	Williams, Brian C.	295
Forbus, Kenneth D.	98, 106	Yip, Kenneth Man-kam	302
Freitag, Hartmut	63	Yoshikawa, Shinji	224, 310
Fromherz, Markus	1	Zhao, Feng	318
Gao, Yang	114		
Gelsey, Andrew	124		
Han, Kyungsook	124		
Hibler, David	136		
Ikeda, Mitsuru	224, 310		
Ironi, L.	144		
Itoh, Kiyoshi	156		
Iwasaki, Yumi	36		
Kitamura, Yoshinobu	224, 310		
Kuipers, Benjamin	45, 164, 212		
Lee, Maria	176		
Leitch, R. R.	234		
Lundell, Monika	187		
Mizoguchi, Riichiro	224,310		
Mooney, Ray J.	212		
Motoda, Hiroshi	197		

