QUALITATIVE SYSTEM IDENTIFICATION

by

Ahmet Celal Cem Say

B.S. in Computer Engineering, Boğaziçi University, 1987

Submitted to the Institute for Graduate Studies in Science and Engineering in partial fulfillment of the requirements for the degree of

Doctor

of

Philosophy

Boğaziçi University

1992

ACKNOWLEDGEMENTS

I would like to thank Prof. Selahattin Kuru who proposed the dissertation topic and advised me in all stages of this work. Thanks are also due to Dr. Levent Akın, Dr. Benjamin Kuipers, Arif Harova and Dr. Ethem Alpaydın who supplied some of the references used in this dissertation. I am grateful to Dr. Levent Akın, Dr. Yorgo Istefanopulos, Bülent Özden, Arif Harova and Hakan Çivi for the valuable discussions, and to Dr. Serdar Biyiksiz, who gave the System Identification course in our department. Fedon Kadifeli played a very important role in the writing of this text. I also thank Prof. Selahattin Kuru, Dr. Levent Akın, Prof. Yorgo Istefanopulos, Prof. Nadir Yücel, and Dr. Taflan Gündem for their careful reviews. Special thanks go to my parents and to my country.

A. C. Cem Say

QUALITATIVE SYSTEM IDENTIFICATION

The main contribution of this research in the qualitative reasoning area of Artificial Intelligence is the development of the qualitative system identification algorithm QSI. QSI's input is a description of the qualitative behaviors of the system to be identified. Its output is a constraint model (possibly containing "deep" parameters absent in the input) of that system, in the format of Kuipers' qualitative simulation algorithm QSIM. The QSI approach to qualitative modeling makes no assumptions and requires no knowledge about the "meanings" of the system parameters. QSI is discussed in detail.

Other contributions are a new method of eliminating a class of spurious QSIM predictions, and an algorithm for postdiction.

Unlike other approaches to spurious behavior reduction, the method presented here does not require restricting assumptions about the input model. A particular kind of spurious behavior is shown to be caused by pure QSIM's insistence on assigning only point values to "corresponding value tuples" associated with model constraints. The solution put forward here preserves the overall complexity of the algorithm, while producing fewer incorrect predictions, as shown by the presented reports of the case runs and proofs.

Postdiction is the task of finding out the possible pasts of the system under

consideration, given the laws of change and the current state. For obtaining the algorithm, a different scheme of interpreting the tree built by simulation is imposed, as well as the handling of the "flow" of time. Issues of this reasoning task, which is promising for diagnosis applications, are discussed.

NİTEL SİSTEM TANILAMA

Yapay Zeka'nın nitel uslamlama alanındaki bu araştırmanın temel katkısı, nitel sistem tanılama algoritması QSI'ın geliştirilmesidir. QSI'ın girdisi, tanılanacak sistemin nitel davranışlarının bir betimlemesidir. Çıktısı bu davranışları gösteren sistemin Kuipers'in QSIM nitel benzetim algoritmasının biçeminde ve sistemin girdide belirtilmemiş "derin" parametrelerini de içerebilen bir kısıt modelidir. Nitel modellemeye QSI yaklaşımı sistem parametrelerinin "anlamları" hakkında hiç bir varsayım yapmaz ve bilgiye gereksinmez. QSI geniş biçimde tartışılmakta ve örneklenmektedir.

Diğer katkılar, bir grup "sahte" QSIM davranışını elemek için yeni bir yöntem ve bir sonradan tahmin algoritmasıdır.

Sahte davranışları azaltma konusuna başka yaklaşımların aksine, burada sunulan yöntem girdideki model hakkında kısıtlayıcı varsayımlar gerektirmez. Belli bir tür sahte davranışın saf QSIM'in model kısıtları için kullanılan karşılık değer takımlarında sadece nokta değerlerin tutulması konusundaki ısrarı nedeniyle ortaya çıktığı saptanmıştır. Sunulan çözüm, verilen işletim örnekleri ve ispatların da gösterdiği gibi, algoritmanın genel karmaşıklığını kötüleştirmeden daha doğru çıktıların oluşmasını sağlar.

Sonradan tahmin, değişim yasaları ve şimdiki durum verildiğinde olası

geçmişleri bulma işidir. Algoritmayı elde etmek için, zamanın "geçişi" ile ilgili değişikliklerin yanısıra, benzetimce üretilen durum ağacını yorumlamanın farklı bir yöntemi getirilmiştir. Tanı uygulamaları için umut vaadeden bu uslamlama türüyle ilgili konular tartışılmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

Artificial Intelligence (AI) has been defined as "the field of Computer Science that concerns itself with the production of programs which accomplish tasks which, when performed by a human, can be considered intelligent." [1] In the 35 years since the Dartmouth Conference, widely regarded as the start of AI, considerable success has been achieved in the automation of some such tasks. Programs which play chess (and some other board games) better than most humans have been written [2]. Expert systems which possess some of the knowledge of a human expert in a specific domain are commercially available and used in various areas [2,1].

However, some other mental tasks performed routinely by humans (so easily, in fact, that they are not normally considered very intelligent,) have proven to be very hard to program, against the early optimistic expectations of some researchers.

Upon comparison of these "basic" tasks (like vision, understanding natural language, or the possession of what may be called "commonsense" knowledge of the world) to the more "advanced" ones mentioned in the first paragraph, one sees that the *knowledge representation* required for tackling the "simple" tasks is much more complicated. Most games have a simple, standard and obvious representation, the board. Expert systems store their knowledge in relatively simple representations (using, for example, *production rules* or *frames*.) On the other hand, the internal representation of things like languages or "the world" is certainly bound to be more complicated. Actually, it is because of their simplistic representation schemes that expert systems cannot solve problems that belong to their domain but are *too simple* for the expertise level that they are "geared" to. They lack *deep* knowledge, that is, a model of the underlying mechanism causing the observable features in the domain. Some variables of this model may not even be directly visible. Knowledge of such a model enables one to perform flexible reasoning, answering a wide variety of questions, and providing *causal explanations* (e.g.

"the rise in the pressure in tank A *led to* an increase in the pressure difference, which in turn *caused* a flow from tank A.") of system behaviors. Most of today's expert systems work with shallow knowledge; their conclusions are based on empiric relationships with the observations. As Hobbs [3] says:

"An expert system in geology may know the characteristics of a good prospect site without knowing that a rock is a physical object and that a geologist is a person."

Considering the importance of representation, McCarthy and Hayes [4] have pointed out that intelligence has two parts: *epistemological* (representation of the world) and *heuristic* (the problem solving mechanism.) In order to construct the epistemological part of an AI, they say:

"The first task is to define even a naive, common-sense view of the world precisely enough to program a computer to act accordingly. This is a very difficult task in itself."

In 1979, Hayes wrote "The Naive Physics Manifesto," [5] in which he called for "a formalization of our knowledge of the everyday physical world: of naive physics," and set out the guidelines for this project. This paper had a great effect, and many researchers were actively involved in the various branches of this endeavor [3].

Humans generally do not use numerical or exact information when performing commonsense reasoning. Rather, the knowledge they utilize is *qualitative* (and, sometimes, wrong.) For example, a human can tell what a ball thrown upward will do, even though (usually) neither the value of its velocity, nor the exact equations governing its flight are known to him. That is why the reasoners and theories developed for automating the task of understanding the behavior of physical systems have adopted a qualitative representation of system models and parameter values, and the associated field of research [6] has been given names like *qualitative physics* or *qualitative reasoning*.

Generally, these programs start with a qualitative model of the system under consideration and reason about its time behavior. This *model-based* reasoning capability is a candidate for solving the deep knowledge problem of expert systems. It also provides for the generation of causal explanations of device behaviors, with potential applications to design and diagnosis aids, and tutoring systems, where capturing and making use of the user's intuition is of great importance. The ability to express and reason with incomplete knowledge is another desirable property of these reasoners. Early applications of this

technology have already appeared, in domains of expertise as diverse as electrical circuits [7] and physiological systems [8,9]. (Also see [10].)

Most of the research in the field of qualitative reasoning has been in the area of *qualitative simulation* [11-14]. A qualitative simulator is a program which takes as input the initial state of a physical system and a qualitative model composed of *constraints* representing time-independent relations between the system's parameters, and produces an account of *all* possible future behaviors of the system, represented as sequences of *qualitative states* (parameter value collections.) Because of the incomplete nature of the information they deal with, qualitative simulators can produce *spurious* behaviors, i.e. those which do not correspond to any actual physical behavior that the system can exhibit, for some inputs.

After pioneering work by researchers including de Kleer and Brown [11], Forbus [12], and Williams [13] on this topic, Benjamin Kuipers developed the QSIM [14,15] algorithm, which is distinguished from the others by its implementation efficiency and its ability to discover qualitatively important *landmark* values about the modeled system.

QSIM has been used as a tool in the development of other reasoners, and the work in qualitative reasoning that will be explained in this dissertation has also been performed in the context of QSIM, using its representation and formalism, so that the compatibility of its products to this "standard" is guaranteed.

1.1. Contribution of This Work

Where do the qualitative models used by the above-mentioned reasoners come from? Surely, modeling the systems in the first place is a very important task, if model-based reasoning is to be performed. Almost all presently available qualitative physical reasoners get these models as input, leaving the whole modeling task to the user. Some authors [11,14] even describe methods of hand-transforming ordinary differential equations governing the input system

(which are to be obtained from, say, textbooks on that domain,) to qualitative form. Some (like Forbus [12]) provide methodologies where the user gives a description of the individuals, configuration and quantities of a physical scene, and the reasoner builds the model using this input, together with already present information about a great variety of physical *processes* that may be active and the relations on the system parameters imposed by these processes. Although these are admirable approaches to using available knowledge and building correct models, the question remains: Can one automate the task of *finding out* this information about the relations between the parameters of a given system?

Humans can form a mental model of the dynamics of a newly seen physical system after observing its working for some time. This suggests that a kind of *learning* process is going on, whose input consists of system behaviors and whose output is a model of the system. There is also a well-established field (quite unrelated to AI) of modeling dynamic systems from (*numerical*) experimental data, known as *system identification*. In this study, it has been decided to adopt such an approach to the problem, and a *qualitative system identification* algorithm, QSI, which takes a set of system behaviors expressed in the QSIM format as input, performs constraint generation and testing on these data, and proposes a QSIM-style model of the system as output, has been developed. QSI also has the capability of finding an "appropriately" deep model, a feature that also exists in conventional system identification. Despite their conceptual similarity, the methods have great internal differences, because of the different natures of the data they handle.

Research in qualitative physical reasoning has mostly striven to produce qualitative analogs of existing "standard" methods for performing various analysis tasks. The already mentioned technique of qualitative simulation, for instance, is the equivalent of numerical simulation or analytical solution of equations, in this domain. Similarly, Daniel Weld's methods of *comparative analysis* [16,17] are qualitative analogs of the standard task of perturbation analysis. As remarked above, QSI adds the system identification task to the repertory of qualitative reasoning.

In addition to its suitability as a model generator for qualitative simulators, the QSI method also suggests an approach to the *model structure determination* problem of conventional system identification itself. Its ability of preparing models for systems whose behaviors are observed, without

making any restrictive assumptions on the meaning or function of the systems, also makes QSI a good candidate for diagnosis applications.

As mentioned above, QSI achieves learning; a topic of much AI work. How it compares to established learning procedures has been investigated. Proofs of correctness and a performance evaluation which shows that its time complexity is comparable to that of other qualitative reasoners have also been given.

Prediction of future behavior is only one of a variety of temporal reasoning tasks [12] which can be accomplished using qualitative models. One such task, which has been little-explored because of some inherent difficulties, is *postdiction*, inferring what happened in the past from the current state of the system. Certain modifications on the "time-passing" module of QSIM have been made and a new way of interpreting the *state graph* it produces during simulation has been developed, resulting in an algorithm for postdiction by qualitative simulation. Postdiction is promising in relation to diagnosis tasks, where it can be used to enumerate the various ways in which "things may have gone wrong," given a problematic system state.

As explained before, qualitative simulators may predict spurious behaviors. Work has been going on [18,19] to decrease the number of such spurious solutions produced by QSIM. In the course of this study, a shortcoming in the *corresponding value filters*, which QSIM uses during simulation to prevent states which do not satisfy the constraint model from being generated, has been discovered. This anomaly causes a particular class of "illegal" states to pass the filters, thereby producing spurious behaviors. Improved qualitative arithmetic and corresponding value recording routines have been designed as a solution to this problem. These modifications have been integrated into an implementation of QSIM, and a provably improved program has been obtained. The method requires no changes in the assumptions about the input information, unlike the other approaches to spurious behavior reduction.

1.2. Structure of the Dissertation

The rest of this dissertation is structured as follows: Chapter 2 provides the background and gives a technical overview of previous qualitative reasoners, with an emphasis on their features relevant to this work. The qualitative simulation algorithm QSIM is examined in a similar manner in Chapter 3. The filtering problem of QSIM which leads to an avoidable class of spurious behaviors and the solution for it are explained in Chapter 4. The work on postdiction by qualitative simulation is detailed in Chapter 5. Chapter 6 starts the coverage of qualitative system identification with a detailed presentation of the algorithm. Complexity and correctness analyses of QSI are given in Chapter 7. Chapter 8 is a discussion of the various considerations that arise when performing the QSI task, and the place of the algorithm in the wider reasoning scene. Various QSI examples are presented in Chapter 9. Chapter 10 contains recommendations for future work in the area and a conclusion. The appendix is about the implemented programs.

II. QUALITATIVE PHYSICAL REASONING

The behavior of physical systems takes place in time, and reasoning about them requires an adequate and efficient way of representing time and change. This chapter of the dissertation will begin with an overview of the "classical" AI approaches to temporal representation. Various qualitative reasoners will be presented, with their features relevant to the present areas of interest highlighted. Finally, other AI topics with a relation to the presently investigated ones will be briefly covered.

2.1. Temporal Representations

Temporal reasoning needs a specialized representation. The approaches summarized in this section have greatly affected and shaped further AI research in the area. As will be seen in the next section, qualitative physical reasoning researchers have also adopted and specialized the methods that will be mentioned here.

2.1.1. The Situation Calculus

The best-known temporal representation scheme is the *situation calculus*, developed by McCarthy [20] and Hayes [4]. In this formalism, a *situation* is the state of the whole universe at a given time. (Of course, the reasoner's knowledge of "the whole universe" is very limited. What is meant here is "all the *facts* known to the reasoner.") A situation persists until some facts about the world change, in which case a new situation begins. Facts can be changed by *actions* (e.g. shooting somebody causes the facts about his health to be

changed, therefore leading to a new situation,) or *events* (the falling of a vase changes its price.) Transitions between situations (i.e. the passing of time) are shown by the *result* function:

$$new\_s = result\ (s,\ e) \tag{2.1}$$

means that if event e occurs in situation s, the situation new_s results. The state of the world *during* the change is not modeled and therefore cannot be reasoned about. Laws of change, which dictate how particular events cause change are then represented as in the following example [21]:

$$\forall\ s,\ x,\ c:\ color(result(s,\ paint(x,\ c)),\ x) = c \tag{2.2}$$

That is, in the situation that results when object x is painted to color c, the color of x is c. Note how facts have situations as arguments. This is necessary because one is performing *temporal* reasoning; the facts are time-dependent.

A famous drawback of the situation calculus is the *frame problem*. Consider a reasoner which supports "color" facts about objects as in the above example together with lots of other facts about other things, such as the name of the prime minister, for instance. In situation s_1, many such facts are known. Then, *paint(object_1, blue)* occurs, resulting in situation s_2. Now, if the reasoner is asked about the name of the prime minister in situation s_2, will it be able to answer the question? That fact is known in s_1, but how can one deduce that it is still there in s_2? As this example illustrates, in classical situation calculus, one has to include a huge number of *frame axioms* that specify which aspects of the world do not change (i.e. are attached to the "frame") when a particular event occurs, for each aspect and for each kind of event.

But the real problem with situation calculus from the point of view of reasoning about physical systems is that it cannot deal with *continuous* change in a healthy manner. As McCarthy [21] says, the situation calculus "applies only when it is reasonable to reason about discrete events, each of which results in a new total situation. Continuous events and concurrent events are not covered."

2.1.2. Hayes' Work

In "The Naive Physics Manifesto" [5], Hayes introduced a new ontological primitive, the *history*, as an alternative to situation calculus. The history of an object or an event is the 4-dimensional piece of spacetime that it occupies, bounded spatially and temporally by its "natural" boundaries. This scheme has no frame problem, since only objects and events whose histories interact in some manner can affect each other; all other things are isolated from them and thus do not change.

In the same paper, while enumerating the general commonsense concepts that humans have, Hayes gives a definition of *quantities* as qualities that can be measured. He goes on to point out that the measuring scale involved may not impose a strict linear order on the space of quantities being measured. He adds that fuzzy values [22] may be used in this context. Interestingly, as will be discussed later, a qualitative representation quite different from the fuzzy setup has been adopted in the field.

2.1.3. Allen's Theory of Time

Allen [23,24] proposed a model of naive temporal reasoning which is based solely on time intervals. He argued that time "points," i.e., "instants" in the representation produce some confusing interpretation problems and claimed that effective temporal reasoning can be achieved without them. His classic example is as follows: Someone turns on the light. This means that two time intervals exist: One during which the light was off, followed by another one during which it is on. If these intervals are both open, there is a time point in between "during" which the light is neither on nor off. This is counterintuitive. If the intervals are both closed, then there is a time point when the light is both on and off. This is even worse. And there is no good reason for one interval being closed and the other being open. So one should not use "the real line as a model of the time line" in naive physical reasoning.

Allen says that the commonsense notion of time point, sometimes used by humans, is actually an interval which is too small for the *scale* of the current

discussion, and can always be decomposed to smaller intervals at smaller scales[1]. (Consider the different time scales used by astrophysicists and politicians, for instance.)

In Allen's model, two intervals, one of which is immediately followed by the other, like those in the light switch example, are said to *meet*. Twelve other "primitive" relations that are possible between intervals are defined and a constraint propagation algorithm for performing reasoning of the kind of being able to answer queries like "Did event_1 happen before or after event_8?" is developed. These sort of problems can arise in story understanding.

Allen's theory caused some controversy. Galton [25] says that it is not suitable for representing continuous change, and proposes that instants should be included in the ontology as part of his solution. Williams, who has adopted the real line representation for time in his qualitative physical reasoning system, criticizes Allen's ontological decisions in [13]. His arguments will be presented in Section 2.2.3.

2.2. Qualitative Reasoning

The basic difference of the work to be explained here from that of the researchers referenced above is that, while those researchers' aim was to achieve *generality* in AI, (i.e. they were after a scheme which was up to representing a wide range of the temporal reasoning tasks that humans perform,) the aim of what has become known as the *qualitative physical reasoning* field is to automate reasoning about the smaller domain of the working of physical systems. Special emphasis on the relation of the *structure* of these systems (or mechanisms, devices, etc.) to their *behavior* is made. A model of the system, describing the relations between its various elements

[1]Kuipers has used this notion of time scale in designing an abstraction method for QSIM (Section 3.2.1.)

(parameters, quantities, etc.) is used to generate not only a description, but also an explanation of various aspects of its behavior. Since this task domain is much more specialized both representationally and computationally than what, say, the situation calculus set out to handle, more efficient representations and algorithms that deal with it can be, and have been, developed.

2.2.1. Qualitative Process Theory

Forbus' *Qualitative Process* (QP) *Theory* [12] is one of the foundations of qualitative reasoning about physical systems. Many concepts and techniques used in the field were firmly established in the context of that work.

QP theory is a model-building methodology based on the notion of physical *processes*. All change is viewed to be caused by processes. The reasoner is assumed to have a (big) library of *model fragments* [26]; knowledge about various kinds of processes (under what conditions and in what configurations they occur, what changes they impose on various quantities,) and of relations that various states impose on the continuous parameters involved in them (e.g. the existence of water in a container necessitates a relation between the amount of water and the pressure at the bottom.)

Forbus introduced the *quantity space* representation for the magnitudes of parameters. The quantity space of a parameter is a partially ordered set of *landmarks*, symbols representing qualitatively important real values (like *0*, *boiling_point*, or *maximum_capacity*) that this parameter can take. Magnitudes are then represented as points or intervals in the quantity space. Each parameter's value is composed of its magnitude and time derivative. In this way, the directions of change of parameters are easily represented in their values.

The relations between parameters can be either in the form of arithmetic operations, as a parameter being the sum of two others, or in terms of *qualitative proportionalities*, as in

$$\text{pressure } \alpha_{Q+} \text{ amount\_of\_liquid} \qquad (2.3)$$

which means that, all other things held constant, pressure will increase if amount is increased, and it will decrease if amount is decreased. No details about

the function except its monotonicity are known. α_{Q-} similarly represents inverse proportionality. Additional information about the function can be specified using *corresponding values* in the quantity spaces of the related parameters that define points which the function crosses. For example, $(0,0)$ is a corresponding value tuple for the above proportionality, since the pressure is zero when the amount is zero.

Given a description of the individuals, configuration, and certain parameter values, if one makes the *Closed World Assumption* (CWA) that all the relevant knowledge about the part of the world in which one is interested is available, QP theory can be used to determine the currently active processes, and therefore to assemble a model of the laws of motion driving the system. The future behavior of the system is then predicted by qualitatively combining the effects on the quantities to see which threshold values they will cross, possibly leading to changes in the process structure. Because of the ambiguity inherent in the representation of quantities and relations, usually many alternative futures are predicted, each of them corresponding to an actual scenario that satisfies the model.

The *limit analysis* technique used by Forbus to achieve the "passing" of time, i.e. to find the "next" interval (Allen's model of time is adopted) in which some aspects are qualitatively different from the current one, evolved to be a standard method used by *qualitative simulation* programs. The idea is this: The sign of the time derivative of each parameter indicates its direction of change. So all changing parameters are nearing certain points in their quantity spaces. When one of them arrives at such a point, a qualitative change occurs. Barring some cases, in which corresponding value information may be used to eliminate some such transitions, one generally has no way of knowing which parameter will change qualitative value first, so the prediction branches to consider all cases. For instance [12], suppose the existence of two containers with fluids in them, and an open fluid path between them has been described in the input. A fluid flow process is activated from the container with the greater pressure to the other one. (Equation 2.3 is also part of the present information.) Given all this, a reasoner employing QP theory is able to deduce two alternative futures for the system with limit analysis. In one of them, the pressures eventually become equal (since the flow process influences the "target" amount positively, and the "source" amount negatively,) and the process terminates as the pressure difference that triggered it vanishes. In the

other alternative, the source container runs out of fluid (i.e. that amount reaches the landmark *0*) and the process stops, since a flow needs the existence of a source. Which of the alternatives actually occurs cannot be determined without further (configurational) information about the system.

Forbus also introduces *encapsulated histories* to represent "destructive" events, like breakings, collisions, and explosions, whose underlying mechanism is hard to model. These are just accounts of what happens when such an event occurs.

Since qualitative simulators are mostly outgrowths of one another, a detailed example of their complete working will only be given for QSIM, in Chapter 3.

Forbus argues that QP theory can be helpful for the generation of Hayes-style histories. He proposes ways of identifying components of physical scenes that do not affect each other, and therefore can be reasoned about independently.

The origin of the huge number of model fragments which contain knowledge of dynamics that are an essential part of QP theory is not made clear. In this study, it is claimed that such knowledge of structure can be obtained by examining the behavior exhibited by the system, and observing the relations between the parameter values. An algorithm based on this claim will be presented in Chapter 6.

2.2.2. Qualitative Physics and Causality

de Kleer and Brown's ENVISION [11] program, based on their *Qualitative Physics*, assumes a mechanistic world view. The input system models are formed by connecting component models in the program's component library according to the device topology of the system. Zero is (practically always) the only landmark used. There are *conduit* laws which have to hold in the connections between the components, representing physical conservation rules, as well as the time-invariant *confluences* (qualitative differential equations) of each component, governing its working. Furthermore, each

component may have various *states*[2] in which different confluences apply, determined by the values of certain variables. The component confluences have to have been previously written by a human, either employing "commonsense," or directly transforming them from the conventional physical equations (to be found in, say, a textbook,) to the qualitative form, where only the signs of variables and their derivatives come into play.

Making the assumption that the system is always near or at equilibrium, and starting with the initial values of some of the variables, ENVISION solves the system of equations to obtain the intrastate behavior; that is, a complete assignment of values to variables. Descriptions of the variable values and directions in all different states that the system can enter can be calculated. Since the directions indicate the "next" values that the variables may obtain, a diagram showing all possible state transitions that the system can go through can then be constructed. One must note that this cannot be considered a simulation process like the scheme discussed in relation to QP theory; in simulation, if event A comes after event B in "real life," this relation is reflected in the order they are processed by the program. ENVISION, on the other hand, "calculates" all states that the system can have, and then starts an analysis of possible transitions between them.

de Kleer and Brown also developed a way of producing causal explanations of intrastate behavior, trying to show how the change in the value of a variable *causes* changes in the values of other variables in the same confluence. Causality is very hard to formalize [27], and de Kleer and Brown were not entirely successful. Consider the very simple equation

$$X = Y. \tag{2.4}$$

ENVISION produces explanations like "X increased, and this caused Y to increase," but this seems to imply that there was a time period during which $X \neq Y$, which doesn't make sense. de Kleer and Brown had to introduce the concept of *mythical time*, during which the laws may be temporarily violated. No "normal" time passes during mythical time. When the confluences are slightly more complicated than the form above, ambiguities result as to how causality should be propagated among the variables. de Kleer and Brown use

---

[2] Note that de Kleer and Brown's notion of state is somewhat different from the one that will be used in most of this text.

canonicality heuristics which encompass the intuition "If you do not know that something is changing, assume that it is not," to help decide in propagation in such cases. Iwasaki and Simon [28,29] pointed out the weaknesses of mythical causality and showed how the method of *causal ordering*, developed by Simon in the 1950's for the field of econometrics, provides better solutions.

2.2.3. Temporal Qualitative Analysis

Brian Williams [13] developed the *temporal qualitative* (TQ) *analysis* technique as the basis of a CAD tool for high-performance circuits. Specialized to this domain, the model building and explanation-generation features of TQ analysis are similar to those of de Kleer and Brown. However, the interstate transitions are handled with *transition analysis*, in which the limit analysis task introduced by Forbus is performed using a comprehensive set of rules. This task is divided into the *transition recognition* (Which parameters are likely to change qualitative value in the next instant?) and *transition ordering* (Which of these transitions will occur first?) parts, guided by continuity considerations and device laws. These make TQ analysis a major qualitative simulation technique, and an important predecessor of QSIM. Since *operation region changes*, where the equations governing the device have to be replaced by another set are common in this domain, they are handled in detail. Williams adopts the real line representation of alternating points and open intervals for both parameter values and time. A parameter which is positive and decreasing (or negative and increasing) takes an instant (a time point) to cross zero, and an interval of time to cross an interval, and since the device behavior is an account of the ways in which the collection of parameters are changing, it has to be presented as a sequence of alternating time points and intervals. Countering Allen's light switch argument (see Section 2.1.3) against time points, Williams points out that the intensity of the light is a continuous parameter, and the notorious "point" in between the two intervals actually contains a (very short) interval during which the intensity is increasing from 0 to its standard level. He puts forward the example of a ball thrown upwards (a very popular "system" in commonsense reasoning research) and says that, intuitively, the ball "stops" at the top of its trajectory for only an instant, i.e, a time point between the two intervals of rising and falling, and therefore time points have a place in intuition in domains where continuous change is present.

2.2.4. Kuipers' Work

Early in his work on qualitative simulation of mechanisms, Benjamin Kuipers developed the ENV [30] program. Unlike its predecessors, ENV does not have a library of model fragments, and for each input, the user is expected to specify the laws of the mechanism in terms of time-invariant *constraints* between the parameters. The allowable constraint types include arithmetic, functional, and derivative, which later formed QSIM's constraint vocabulary. *Functional* constraints state that the two parameters linked by them have a functional relationship which is either monotonically increasing (in the case of the M+ constraint) or monotonically decreasing (in the case of the M- constraint.) Since this definition is different from that of Forbus' qualitative proportionalities, (where *other* parameters may have opposing effects on the parameters involved,) the sign of the time derivative of a parameter in a functional constraint can be immediately determined if the sign of the other parameter's derivative is known.

ENV leaves the whole modeling task to the user, but its flexible constraint vocabulary allows any system with continuous time-varying parameters governed by differential equations to be described to it, making it a most general-purpose qualitative simulator. Kuipers' interest in the medical domain, [8,9] where most "component" laws are only very incompletely known, may have influenced this decision.

Starting with a description of the system's initial state, ENV executes a propagation/prediction cycle: Propagation completes information about the current state, prediction uses a great number of rules, similar to Williams' transition analysis rules, to determine the next state. If intractable branching (i.e. too many alternative predictions) occurs, the system model is simplified to avoid this situation and simulation continues. Cyclic system behaviors and equilibria can be recognized by the reasoner.

ENV allows the use of nonzero landmarks, and, as an improvement to all its predecessors, is able to discover new landmarks of the parameters during simulation.

Kuipers later developed QSIM [14] as a successor to ENV. QSIM adopts a total-ordered, independent quantity space representation for parameters. It contains many improvements (e.g, much clearer methods of transition analysis and constraint filtering, which make the algorithm amenable to proofs of correctness and lead to very acceptable execution times) over ENV. Work on QSIM is continuing around the world to get rid of *spurious* behaviors (i.e. those that do not correspond to any actual solution of the underlying differential equations) that it can produce for some input systems, and to make it work feasibly for large models.

A modification to the QSIM algorithm that leads to improved constraint filtering, which in turn helps to eliminate some spurious behaviors, will be presented in Chapter 4. Certain other modifications which enable the structure of QSIM to be used for performing postdiction, or "predicting the past," will be explained in Chapter 5.

As well as the many people from the expert systems, monitoring and control, and simulation fields who were attracted to QSIM for its applications, other qualitative reasoning researchers too have used it as a tool to build their own reasoners (See the section on Weld's work) because of its simple representation and powerful simulator. The same approach has been adopted in the development of the qualitative system identification method, QSI. Since QSIM plays such an integral role in this study, the basic algorithm and the many improvements to it will be presented and discussed in detail in Chapter 3.

2.2.5. Weld's Work

Weld [31] introduced the technique of *aggregation* for qualitative simulators, and developed the PEPTIDE program as a first attempt to implement it. Aggregation involves simulating the behavior of a process (which may be discrete or continuous) and trying to recognize cycles in this behavior. Unlike the method used by QSIM, which identifies a cycle when a qualitative state is *identical* to a previous one, aggregation recognizes cycles if the states are *similar* to each other. The net effect of one "turn" of the cycle (e.g. reducing the amount of a finite resource) is determined, and this repeating cycle of processes is replaced by a single continuous process summarizing their effect. Since it is continuous, this higher-level abstraction of the system can be

simulated using transition analysis leading to faster and more powerful reasoning. Some problems remain with the aggregation method, but the fact that it provides insights about dealing with discrete processes in qualitative simulation is interesting.

Weld has developed two different methods for *comparative analysis*, the task of determining how small perturbations cause changes in the system behavior. The QSIM format and algorithm are used in both of them. The input to comparative analysis is a QSIM model of a system, its initial state, and a QSIM-produced behavior leading from this initial state, together with a perturbation specifying the direction(s) of the perturbed parameter(s). The expected output is an account of how and why the behavior changes in response to the perturbation. The first method, *differential qualitative* (DQ) *analysis* [16], involves the assumption that the "structure" of the behavior does not change, and achieves its results by propagating the input changes through the constraints. DQ analysis may fail to solve some problems, but gives correct answers for all the ones that it solves.

The second method, *exaggeration* [17], approaches the comparative analysis problem by considering extreme perturbations in the direction indicated in the input. For instance, when asked about the change in the period of the oscillating block/spring system in response to increased block mass, a system model where the block mass is infinite is simulated. A modified version of QSIM that can deal with hyperreal values (infinity and infinitesimals) is used for this process. Assuming that the system responds monotonically to the perturbation, the set of deviations in the result obtained for the extreme case is presented as output. Exaggeration may give wrong answers for some problems.

Weld points out [16,17] that QSIM's representation of behaviors as sequences of states may force it to predict an unnecessarily large number of behaviors that have no interesting difference. (This happens when the model is large and the quantity spaces have relatively many landmarks in them.) Hayes' development of histories was an attempt to avoid this problem, which is also present in the situation calculus, where the smallest change in the facts causes the whole world to enter a new situation. Williams [32] devised a temporal constraint propagator which embodies these ideas to forgo a total ordering on non-interacting events, and therefore to produce fewer behaviors.

2.2.6. Concluding Remarks on Literature Survey

This section has presented a brief overview of the qualitative physical reasoning field. Emphasis was on the section of literature that bears a direct relation to and can provide a background for the concepts and tools used in this study. For this reason, a lot of important work (for instance, on *spatial* reasoning [33,34]) not immediately relevant to the undertaken research was left out.

The power and efficiency that qualitative reasoners have achieved in the past few years is evident in the various applications [35,1,36] they are being integrated to. Comparisons are being made [37-39] of qualitative reasoning methods, which originally set out to automate human commonsense, to the well-established science of conventional dynamic systems theory. This underlines the potential importance of the new methods of qualitative reasoning that will be introduced later in this dissertation.

2.3. Related Topics

This section is about certain AI topics that are related to, and sometimes confused with, qualitative reasoning. Brief comparisons will be made between these and the present subject, and the differences will be underlined. So in a sense, what this dissertation is *not* about is explained here.

The fuzzy representation [22] for variable values and the relationships among them is a well-established approach to dealing with incompletely known systems, and has an extensive literature of its own. The qualitative reasoning methods' inherent property of representing parameter values and functions incompletely may lead to an impression that a type of fuzzy representation is employed. This is untrue. Although both approaches use the technique of *summarizing* a whole range of real numbers to a single "value," the manner in which they accomplish this is fundamentally different. The

mapping of a real number to a fuzzy value is a "possibilistic" issue, depending on the *grade of membership* of that number in the *fuzzy subset* describing the fuzzy value. Furthermore, a single real number can be mapped to several fuzzy values; it may be the case that a given temperature turns out to be both "hot" and "cold," (with varying degrees.) This is never the case in the quantity space representation. Each real number is mapped to exactly one qualitative magnitude, (though one may not know *which* one, if the numerical values of the landmarks are not known.) In the absence of further information, two real numbers in the same interval in the quantity space are indistinguishable, the idea being that if they did have an important difference, a landmark between them, making ordinal comparison possible, would be provided. (A method that makes use of program-generated history information to compare two instances of the same interval is presented in Chapter 4.)

Suzuki et al. [40] have proposed a qualitative simulator using fuzzy logic. Their program does not predict multiple behaviors, since this is not desirable in the particular application (plant control) that they have in mind.

As mentioned above, "temporal reasoning" in AI means much more than this study's particular area of interest. Any program that answers some kind of time-related query can be regarded as performing temporal reasoning. The techniques in the field have been developed mainly to deal with two separate issues [2, Ch.7]: One "family" of programs specialize as "databases" and strive to provide efficient organization and retrieval of large amounts of time-indexed data. Methods of the second group focus on inference; i.e. on how to deduce new items of information from the already present ones. The inferred items usually reflect later events which are results of the ones used as antecedents. It has already been pointed out that qualitative physical reasoning is a quite specialized subfield of this inference approach; the laws of change it employs (e.g. "If a quantity is increasing now, it either goes on increasing or stops in the near future") are much more basic compared to those used in the general case (like "If you kick a dog, it will want to bite you.")

As to the relation of conventional simulation techniques with qualitative simulation, see Section 3.1.4 for a discussion.

III. QSIM AND ITS EXTENSIONS

Since Kuipers' first presentation of QSIM in [14], many extensions, improving the algorithm in a variety of ways, have been produced, mostly by Kuipers and his team in the University of Texas at Austin. But the "core" of QSIM remained intact, and other researchers using QSIM for their own purposes always started out with the *pure* version of the algorithm. Kuipers sends the code to interested researchers. In the course of this study, the program was written according to the specifications in [14] in Turbo PROLOG Version 2.0, and the work reported in this dissertation was implemented upon that basis. (The original QSIM is in LISP.) The next section contains an explanation of pure QSIM. Design decisions that were made on points where [14] is not clear are indicated. The terminology is slightly modified for purposes of clarity. Section 3.2 is about the various extensions to this algorithm that appeared in literature. Most of these are relevant to the present work too.

3.1. QSIM

This section is a detailed description of QSIM: Its representation and algorithm are explained in depth. Examples are given of the working of, and problems with, the algorithm. The correctness and complexity issues are discussed, and a comparison of QSIM with classical simulation methods is presented.

3.1.1. Parameters, Constraints and Value Transitions

The *parameters* of the physical system under consideration are continuously differentiable functions of time. Both the domains and the ranges of these

functions are closed intervals in the extended real number line $\mathbf{R}^*$, i.e. $[-\infty,\infty]$. QSIM requires the parameters to be *reasonable* functions, which means they have to have only finitely many critical points in any interval.

Each parameter has a *quantity space*. This is a totally ordered set of symbols, i.e. *landmarks*, representing interesting real values that the function can have, such as its values at its critical points. The three landmarks $-\infty$, 0, and ∞, appear in all quantity spaces. All other landmarks are meaningful only in their own quantity spaces. One cannot represent much of the numerical value of a landmark in this setup. All one can say stems from its ordinal relations with the other landmarks in its quantity space. At least, the sign of each landmark is evident, since 0 is in every quantity space.

The *qualitative magnitudes* that a parameter can take are the points and intervals between adjacent points in its quantity space. So, for instance, with the basic quantity space $\{-\infty, 0, \infty\}$, the whole extended real numbers are mapped to five qualitatively distinct qualitative magnitudes: $-\infty$, $(-\infty, 0)$, 0, $(0, \infty)$, and ∞. This definition makes the rationale for including $-\infty$ and ∞ in every quantity space evident.

The *qualitative direction* of a parameter is the sign of its derivative at that time. In that context, the symbols **inc** (increasing,) **std** (steady) and **dec** (decreasing) are used for +, 0, and -, respectively.

The *qualitative value* of a parameter at a particular time is the pair consisting of its qualitative magnitude and qualitative direction.

The time points in a parameter's domain, in which its magnitude changes to or from a landmark are called its *distinguished time-points*. t_0, the time at which the simulation begins, is the first distinguished time-point. The *qualitative behavior of a parameter* is then the sequence of its qualitative values in the alternating time points and open intervals t_0, (t_0, t_1), t_1, (t_1, t_2), ... In the time intervals, the parameter is either increasing or decreasing in an interval in its quantity space, or "sitting" at a landmark. At the time points, it either arrives at (and maybe passes) a landmark, or leaves a landmark that it has been sitting at. So only the changes are represented by the behavior. Nothing is known about the numerical values of the time points, except their ordering. The "length" of (i.e, the number of qualitative values in) the behavior of a parameter is just a measure of the qualitative changes that occur,

not of the time that passes. The height of a rocket that is launched from the Earth and continues in the same direction for a million years will have a qualitative behavior consisting of only two values, those at t_0 and (t_0, t_1), while the height of a bouncing ball thrown up will have a much longer behavior.

The *distinguished time-points of the system* being simulated are the union of the distinguished time-points of the parameters of the system. The *qualitative state* of the system is the collection of the qualitative values of its parameters at that time. The *qualitative behavior of a system* is then the sequence of its qualitative states at its distinguished time-points and the intervals between them. So the system changes state when one or more of its parameters exhibit a qualitative value change.

At each step of the simulation, QSIM considers *all* the possible qualitative values that each parameter may take on in the next qualitative state. Kuipers has proven [14] that each parameter is restricted to the qualitative value transitions shown in Table 3.1.1. In that table, l_{j-1}, l_j, and l_{j+1} are three ordered landmarks of the parameter. *P-transitions* are transitions from qualitative values at distinguished time-points to intervals, while *I-transitions* are transitions from intervals to time-points. Table 3.1.1 (along with most of this section) has been taken from [14], but transitions P4 and P5 have been interchanged. This has no effect on pure QSIM, the reason for this change will be explained in Chapter 5.

Transitions I8 and I9 give QSIM an ability nonexistent in other reasoners; they discover new landmarks. The parameter "comes to a halt" in an interval in its quantity space, its value at that point is inserted as a new landmark between l_j and l_{j+1}, preserving the total ordering.

The "laws" that the system has to obey are represented by *qualitative constraints* between the parameters that have to be satisfied in each state. These form a model of the system, reflecting the underlying structure behind the behavior. QSIM assumes that there is an ordinary differential equation (ODE) governing the system, and the constraint set is its qualitative counterpart, that is, it is a *qualitative differential equation* (QDE.) The ODE need not be actually known, it is just assumed to exist. A QDE may correspond to many ODEs, as will be discussed below. Performing qualitative simulation on the QDE is

analogous to solving the ODE, they both find out the time behavior of the system.

TABLE 3.1.1. The transitions

P-transitions

name	at t_i	in (t_i , t_{i+1})
P1	$< l_j$, std >	$< l_j$, std >
P2	$< l_j$, std >	$< (l_j , l_{j+1})$, inc >
P3	$< l_j$, std >	$< (l_{j-1} , l_j)$, dec >
P4	$< (l_j , l_{j+1})$, inc >	$< (l_j , l_{j+1})$, inc >
P5	$< l_j$, inc >	$< (l_j , l_{j+1})$, inc >
P6	$< l_j$, dec >	$< (l_{j-1} , l_j)$, dec >
P7	$< (l_j , l_{j+1})$, dec >	$< (l_j , l_{j+1})$, dec >

I-transitions

name	in (t_i , t_{i+1})	at t_{i+1}
I1	$< l_j$, std >	$< l_j$, std >
I2	$< (l_j , l_{j+1})$, inc >	$< l_{j+1}$, std >
I3	$< (l_j , l_{j+1})$, inc >	$< l_{j+1}$, inc >
I4	$< (l_j , l_{j+1})$, inc >	$< (l_j , l_{j+1})$, inc >
I5	$< (l_j , l_{j+1})$, dec >	$< l_j$, std >
I6	$< (l_j , l_{j+1})$, dec >	$< l_j$, dec >
I7	$< (l_j , l_{j+1})$, dec >	$< (l_j , l_{j+1})$, dec >
I8	$< (l_j , l_{j+1})$, inc >	$< l^*$, std >
I9	$< (l_j , l_{j+1})$, dec >	$< l^*$, std >

For most realistic systems, one set of equations cannot cover the whole range of operation of the system. QSIM acknowledges this with the notion of *operating regions*. A set of constraints apply only in a specific operating region. When one of the parameters exceeds its *legal range*, that is, a previously specified interval in its quantity space, in an operating region, a new set of constraints associated with the new operating region, specified by the parameter that caused the change, become valid.

There are six types of constraints in pure QSIM: ADD, MULT, MINUS, DERIV, M+, and M-.

The constraint DERIV(X,Y) indicates that the parameter Y is the time derivative of parameter X. That is, the direction of X should always be equal to the sign of the magnitude of Y.

MINUS(X,Y) means that the parameters X and Y are always the negatives of each other.

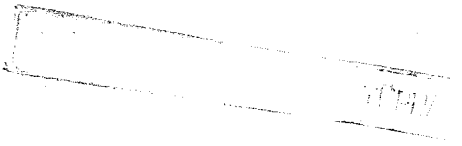
ADD(X,Y,Z) means that the sum of X and Y is always Z.

MULT(X,Y,Z) means that the product of X and Y is always Z.

M+(X,Y) says that there is a monotonically increasing functional relationship between X and Y, and M-(X,Y) means that the function is monotonically decreasing, (See [14] for the exact definitions.) If one has M+(X,Y), the directions of X and Y are always the same. If M-(X,Y) or MINUS(X,Y), the directions are the negatives of each other. Note that the *functional* constraints M+ and M- correspond to an infinite number of "actual" functions, and therefore a QDE which contains them will be an abstraction of many ODEs at once.

All constraints except DERIVs can have *corresponding value* (CV) tuples of landmarks that define known points of the relation. The corresponding values $(0,0)$ of the constraint M+(amount_of_liquid, pressure) were mentioned in Chapter 2. Some corresponding value tuples are "natural," e.g. every ADD and MULT has a $(0,0,0)$ triple associated with it. Others may be found during simulation, when all parameters involved in the constraint are noticed to have landmark magnitudes.

The constraints are used to *filter* proposed "next" states produced by the transition rules, and corresponding value information plays an important role in this process. For instance, if the magnitudes $A*$, $(0,B*)$, and $C*$ are proposed for the parameters in ADD(A,B,C), and this constraint already has the CV triple $(A*,B*,C*)$ in its records, the proposed transition will be eliminated, since it is contrary to the algorithm's knowledge of addition and known values. The exact mechanism of this filtering will be detailed later.



This is a good place to reiterate that qualitative arithmetic is ambiguous: An arithmetic operation can have more than one result. Consider Table 3.1.2, which QSIM uses when checking whether the directions of three proposed values satisfy the ADD constraint. "any" in this table means any of **inc**, **std**, or **dec**, and is necessary since the relative magnitudes are not known.

TABLE 3.1.2. Qualitative addition

		Y		
		dec	std	inc
X	dec	dec	dec	any
	std	dec	std	inc
	inc	any	inc	inc

Given any ODE defined only in terms of addition, multiplication, negation, and functions with continuous and strictly nonzero derivative, a set of parameters and constraints can be written, such that any reasonable function that satisfies the ODE also satisfies the constraints [14]. All the models that will be discussed in this study are subject to these restrictions.

Example

As a classic [15] example that will also be used later in the discussion, consider how a simple U-tube (Figure 3.1.1) is modeled. The U-tube, in its "healthy" state, is made of two tanks connected by a pipe. The QDEs for the cases where tank A or tank B are burst will also be considered. So one has three operating regions to model: NORMAL, A_BURST, and B_BURST.

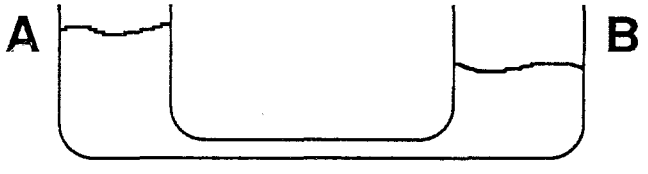


Figure 3.1.1. U-tube in operating region NORMAL

After a lot of simplifying assumptions, the parameters of the system are identified as in Table 3.1.3. There are also the *invariants* that the amount and pressure parameters are never negative.

TABLE 3.1.3. Parameters of the U-tube system

PARAMETER	QUANTITY SPACE	REMARKS
amount_A	$\{-\infty, 0, AMAX, \infty\}$	$AMAX$ is maximum capacity
amount_B	$\{-\infty, 0, BMAX, \infty\}$	$BMAX$ is maximum capacity
flow_AB	$\{-\infty, 0, \infty\}$	flow from A to B
flow_BA	$\{-\infty, 0, \infty\}$	flow from B to A
pressure_A	$\{-\infty, 0, \infty\}$	pressure at bottom of A
pressure_B	$\{-\infty, 0, \infty\}$	pressure at bottom of B
p_diff_AB	$\{-\infty, 0, \infty\}$	pressure_A - pressure_B

The constraints for operating region NORMAL are listed in Table 3.1.4. In this and the following tables describing QDEs, the "natural" CVs of the arithmetic constraints (like $(0,0)$ for MINUS) are not shown; note that the ones that *are* shown need not appear for *any* M+ constraint.

TABLE 3.1.4. U-tube constraints in region NORMAL

CONSTRAINT	CVs
M+(amount_A, pressure_A)	$(0,0)$ and (∞, ∞)
M+(amount_B, pressure_B)	$(0,0)$ and (∞, ∞)
DERIV(amount_A, flow_BA)	
DERIV(amount_B, flow_AB)	
ADD(pressure_B, p_diff_AB, pressure_A)	
M+(p_diff_AB, flow_AB)	$(0,0)$ and (∞, ∞)
MINUS(flow_AB, flow_BA)	

Suppose that when an amount parameter exceeds its maximum capacity, the corresponding tank bursts. If B exceeds $BMAX$ in region NORMAL, the constraints for the ensuing operating region, B_BURST, are then as in Table 3.1.5. The changes are caused by the fact that amount_B is fixed at 0 in this operating region. The QDE for A_BURST is similar.

TABLE 3.1.5. U-tube constraints in region B_BURST

CONSTRAINT	CVs
M+(amount_A, pressure_A)	$(0,0)$ and (∞, ∞)
M+(amount_B, pressure_B)	$(0,0)$ and (∞, ∞)
DERIV(amount_A, flow_BA)	
ADD(pressure_B, p_diff_AB, pressure_A)	
M+(p_diff_AB, flow_AB)	$(0,0)$ and (∞, ∞)
MINUS(flow_AB, flow_BA)	

Consider the state where some water has been instantly poured to tank A, and tank B is empty. This point state, which can be completed by propagation of values from this initial information using QSIM's knowledge of constraints, is shown in Table 3.1.6.

TABLE 3.1.6. Initial state of U-tube system

PARAMETER	VALUE
amount_A	<(0,*AMAX*), dec>
amount_B	< 0, inc>
flow_AB	<(0, ∞), dec>
flow_BA	<(-∞, 0), inc>
pressure_A	<(0, ∞), dec>
pressure_B	< 0, inc>
p_diff_AB	<(0, ∞), dec>

Now consider, for example, amount_A. Only transition P7 (which does not change the parameter's value) from Table 3.1.1 is suitable for it, so amount_A's value has to be again <(0,*AMAX*), dec> in the next state. This example will be concluded in Section 3.1.2.

3.1.2. The QSIM Algorithm

The input to QSIM is the following: The names, quantity spaces, and legal ranges of the system's parameters, invariant assertions that some parameters may have to satisfy, (like "being constant,") one or more operating region descriptions with their associated constraint sets, and a complete description of all the parameter values and the operating region at t_0, (that is, the *initial state*.)

Since more than one behavior can be predicted, QSIM builds a tree of qualitative states, whose root is the initial state, and every path from the root to a leaf is a distinct system behavior which is output.

The algorithm starts by putting the initial state in a list which always contains the states whose successors have not been created yet. Then the following is repeated until the list is empty, or one has to stop for external reasons (the tree may be infinite):

1. Take a state from the list.
2. "Open" it, (i.e, generate all its successors,) doing the following:
   2.1. For each parameter, find its possible transitions to the next state using Table 3.1.1.
   2.2. For each constraint, eliminate all tuples of transitions of its parameters that do not satisfy the constraint in the next state.
   2.3. Check that each proposed transition of every parameter P in a constraint also appears in other constraints involving P. Otherwise, eliminate that transition tuple.
   2.4. Construct all the possible next states by using the join of the remaining transition tuples. If all the parameters have made transitions in the set {I1, I4, I7}, that state is identical to its predecessor, so do not consider it a "next" state. If, in a next state, all magnitudes of a non-DERIV constraint are landmarks, add that tuple of magnitudes to the CV list of that constraint in that branch. If, for any parameter, a new landmark has been found, augment that parameter's quantity space in that branch.
3. Put all the new states created in Step 2 to the list, except when:
   a) A new state is identical to a state S that appears in the path from the root to it; a cycle has been detected, put a pointer from the new state to S, you do not have to continue in this branch anymore.
   b) Any parameter has an infinite value in the new state.

Heuristics that assume that the system has achieved quiescence and stop simulation when all qualitative directions are `std`, and do not allow any transitions to infinite values, are also used..

In the constraint filtering phase (Step 2.2 above,) the newly proposed values are checked for consistency of both their directions and their magnitudes with the constraints they participate in. Apart from the well-known rules that MINUS, ADD and MULT impose on the signs of parameter magnitudes, the CV lists are also a valuable source of information for filtering. As always, the ordinal relations among landmarks are the only guide. For the two-place constraints, consistency with previous CV information is easy to check. In all such constraints, for each CV tuple (p,q), the magnitudes of the two parameters should either be both *at* those landmarks, or both away from them. This simply follows from the nature of being *corresponding* values. For the ADD and MULT constraints, Kuipers presents [14] the following rules, which greatly simplify the filtering process:

- When a triple of magnitudes (m_A, m_B, m_C) is proposed for parameters A, B, C, and the constraint ADD(A,B,C) exists, the following has to hold for each CV triple (p, q, r) of that constraint:

$$(m_A - p) + (m_B - q) = (m_C - r) \qquad (3.1)$$

Proof. It is known that $p + q = r$ and also $m_A + m_B = m_C$ should be true.

Since p, q, and r are point values, each term in (3.1) has an unambiguous value (either +, 0, or -,) and this sign addition's correctness is checked by using Table 3.1.2. (Remember that **dec**, **std**, and **inc** are just names for signs.) The rule for MULT is similar:

- When a triple of magnitudes (m_A, m_B, m_C) is proposed for parameters A, B, C, and the constraint MULT(A,B,C) exists, the following has to hold for each nonzero CV triple (p,q,r) of that constraint:

$$\left(\frac{m_A}{p}\right)\cdot\left(\frac{m_B}{q}\right)=\left(\frac{m_C}{r}\right) \tag{3.2}$$

The logic is the same. *Qualitative division* $\left(\frac{X}{Y}\right)$, yielding one of the symbols **lto** (<1), **one**, or **gto** (>1), can be applied when X and Y have the same (nonzero) sign. The multiplication table (Table 3.1.7) then facilitates the consistency check.

TABLE 3.1.7. Qualitative multiplication

		Y		
		lto	one	gto
X	lto	lto	lto	any
	one	lto	one	gto
	gto	any	gto	gto

How QSIM handles operating region changes is not made clear in [14]. In later work, (Section 3.2.5) QSIM is explicitly restricted to run in a single operating region, another program taking over when a region transition occurs. Using [34] as a guide, this version of QSIM was implemented such that it does the following:

If a state in which a parameter is about to exceed its legal range is obtained, the next operating region's name is found by looking at the appropriate input record of this parameter, and another state, which is the direct descendant of this state, is created. Since the newly created state is the first one of the new operating region, it has to be given special treatment. Usually, the new constraint set has only a few differences from the old one. Constraints which "survive" the operating region change are identified from

the region description and their CV lists are retained. Similarly, surviving parameters retain their quantity spaces. Operating region changes are very similar to Forbus' encapsulated histories (Section 2.2.1) in that they are an easy way of representing "jumps" in the values of some parameters. For example, the level of liquid in a burst container starts from zero (and stays so!) and this is facilitated by including an input fact specifying that this parameter obtains the value zero at the start of this region, disregarding continuity. Parameters which "inherit" their values from the previous region are also indicated in the input. If there are still unknown parameter values, this version of QSIM generates all completions of the state obeying the constraints, leading to a new branching in the tree. Contrary to intra-region transitions, an inter-region transition is from a time-point to a time-point, with the interval in between not being modeled. Or, alternatively, the change may be viewed as instantaneous and discontinuous. Simulation within the new operating region then continues as usual.

Example (continued)

The three behaviors that QSIM predicts for the input of Section 3.1.1, (Tank A contains liquid, tank B is empty,) seen in Tables 3.1.8 thru 3.1.10, are:

- Behavior #1: The amounts stabilize at landmarks below the maximum capacities.

- Behavior #2: amount_B stabilizes just at $BMAX$, narrowly avoiding a burst.

- Behavior #3: Tank B bursts, the liquid in tank A drains away from the "hole."

The quantity spaces of parameters for which new landmarks have been discovered have been shown after the behavior in the tables.

During the region transition from NORMAL to B_BURST, amount_B is set to zero. Discontinuous changes are also seen in the values of pressure_B, p_diff_AB, and the flows, which are linked to amount_B and each other by constraints. For this run of QSIM, the state tree produced has the shape shown in Figure 3.1.2. The point states are shown as circles in that figure, and the time value corresponding to each level is indicated. The reason why all behaviors have the same first two states is obvious from the figure.

TABLE 3.1.8. Behavior #1 of the U-tube

time	t_0	(t_0,t_1)	t_1
amount_A	<(0,*AMAX*), dec>	<(0,*AMAX*), dec>	<*NewA*, std>
amount_B	<0, inc>	<(0,*BMAX*),inc>	<*NewB*, std>
flow_AB	<(0, ∞), dec>	<(0, ∞), dec>	<0, std>
flow_BA	<(-∞, 0), inc>	<(-∞, 0), inc>	<0, std>
pressure_A	<(0, ∞), dec>	<(0, ∞), dec>	<*PA*, std>
pressure_B	<0, inc>	<(0, ∞), inc>	<*PB*, std>
p_diff_AB	<(0, ∞), dec>	<(0, ∞), dec>	<0, std>

Quantity space of amount_A: {-∞, 0, *NewA*, *AMAX*, ∞}
Quantity space of amount_B: {-∞, 0, *NewB*, *BMAX*, ∞}
Quantity space of pressure_A: {-∞, 0, *PA*, ∞}
Quantity space of pressure_B: {-∞, 0, *PB*, ∞}

TABLE 3.1.9. Behavior #2 of the U-tube

time	t_0	(t_0,t_1)	t_1
amount_A	<(0,*AMAX*), dec>	<(0,*AMAX*), dec>	<*NewA*, std>
amount_B	<0, inc>	<(0,*BMAX*),inc>	<*BMAX*, std>
flow_AB	<(0, ∞), dec>	<(0, ∞), dec>	<0, std>
flow_BA	<(-∞, 0), inc>	<(-∞, 0), inc>	<0, std>
pressure_A	<(0, ∞), dec>	<(0, ∞), dec>	<*PA*, std>
pressure_B	<0, inc>	<(0, ∞), inc>	<*PB*, std>
p_diff_AB	<(0, ∞), dec>	<(0, ∞), dec>	<0, std>

Quantity space of amount_A: {-∞, 0, *NewA*, *AMAX*, ∞}
Quantity space of pressure_A: {-∞, 0, *PA*, ∞}
Quantity space of pressure_B: {-∞, 0, *PB*, ∞}

TABLE 3.1.10.a Behavior #3 of the U-tube (in operating region NORMAL)

time	t_0	(t_0,t_1)	t_1	**Operating region change from NORMAL to B_BURST occurs now**
amount_A	<(0,*AMAX*), dec>	<(0,*AMAX*), dec>	<*NewA*, dec>	
amount_B	<0, inc>	<(0,*BMAX*),inc>	<*BMAX*, inc>	
flow_AB	<(0, ∞), dec>	<(0, ∞), dec>	<*NewF*, dec>	
flow_BA	<(-∞, 0), inc>	<(-∞, 0), inc>	<*NewR*, inc>	
pressure_A	<(0, ∞), dec>	<(0, ∞), dec>	<*PA*, dec>	
pressure_B	<0, inc>	<(0, ∞), inc>	<*PB*, inc>	
p_diff_AB	<(0, ∞), dec>	<(0, ∞), dec>	<*NewD*, dec>	

Quantity space of amount_A: {-∞, 0, *NewA*, *AMAX*, ∞}
Quantity space of flow_AB: {-∞, 0, *NewF*, ∞}
Quantity space of flow_BA: {-∞, *NewR*, 0, ∞}
Quantity space of pressure_A: {-∞, 0, *PA*, ∞}
Quantity space of pressure_B: {-∞, 0, *PB*, ∞}
Quantity space of p_diff_AB: {-∞, 0, *NewD*, ∞}

TABLE 3.1.10.b Behavior #3 of the U-tube (in operating region B_BURST)

time	t_1	(t_1,t_2)	t_2	(t_2,t_3)	t_3
amount_A	<*NewA*, dec>	<(*0*,*NewA*), dec>	<(*0*,*NewA*), dec>	<(*0*,*NewA*), dec>	<*0*, std>
amount_B	<*0*, std>	<*0*, std>	<*0*, std>	<*0*, std>	<*0*, std>
flow_AB	<*NewF2*, dec>	<(*NewF*,*NewF2*),dec>	<*NewF*, dec>	<(*0*,*NewF*),dec>	<*0*, std>
flow_BA	<*NewR2*, inc>	<(*NewR2*,*NewR*),inc>	<*NewR*, inc>	<(*NewR*,*0*),inc>	<*0*, std>
pressure_A	<*PA*, dec>	<(*0*, *PA*), dec>	<(*0*, *PA*), dec>	<(*0*, *PA*), dec>	<*0*, std>
pressure_B	<*0*, std>	<*0*, std>	<*0*, std>	<*0*, std>	<*0*, std>
p_diff_AB	<*NewD2*, dec>	<(*NewD*,*NewD2*),dec>	<*NewD*, dec>	<(*0*,*NewD*),dec>	<*0*, std>

Quantity space of amount_A: $\{-\infty, 0, NewA, AMAX, \infty\}$
Quantity space of flow_AB: $\{-\infty, 0, NewF, NewF2, \infty\}$
Quantity space of flow_BA: $\{-\infty, NewR2, NewR, 0, \infty\}$
Quantity space of pressure_A: $\{-\infty, 0, PA, \infty\}$
Quantity space of pressure_B: $\{-\infty, 0, PB, \infty\}$
Quantity space of p_diff_AB: $\{-\infty, 0, NewD, NewD2, \infty\}$

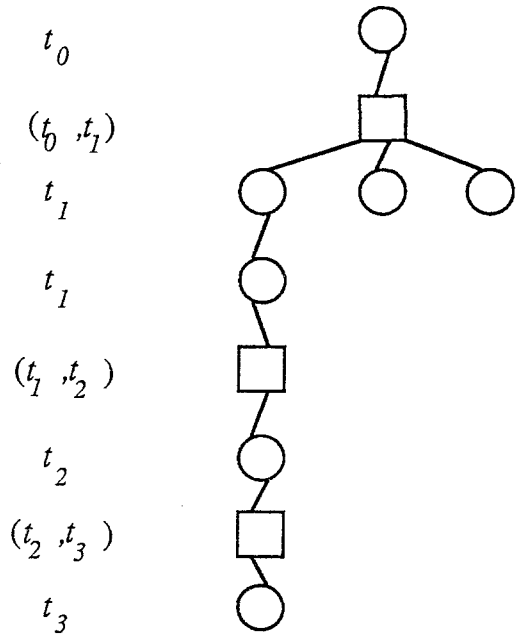


Figure 3.1.2. State tree for U-tube simulation (time values shown)

3.1.3. Complexity and Correctness

Complexity

Considering the time QSIM requires to generate all successors of a given state, Kuipers shows that there are cases where one state yields 2^p successors, where

p is the number of parameters. So one has an algorithm where opening one node is exponential in p, and this in a tree where nodes can have up to 2^p successors! Fortunately, these pathological cases are rare, and in practice [14], opening one node seems to be $O(c\,s)$, where c is the number of constraints, (which is $O(p)$,) and s is the current length of the behavior being generated. s enters the consideration in two places: The CV lists grow as one goes down the tree, and all the previous states in the behavior have to be checked for cycle detection in Step 3.a.

Let us propose an "improvement" to QSIM and then try to see whether it really is an improvement or not. Consider the tree of Figure 3.1.3, which has been produced by QSIM after running for some time. The nodes (states) marked X are identical. (Actual parameter values and constraints that yield such a situation can easily be written.) The subtrees having these nodes as roots will, of course, be identical, and, depending on the specifics of the situation, these subtrees may be huge in size. Clearly, there is no need to open both of these nodes, only a pointer need be set from the "second" to the "first" one; this will save a lot of time. However, QSIM does not realize this situation, since each new state is only compared with the states on the path to the root. One can change the algorithm such that it checks for identical states *all over* the tree, when a new state is produced. What will be the complexity of this new algorithm?

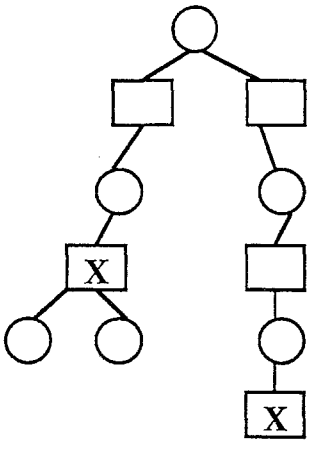


Figure 3.1.3. A partial state tree

Previously, identicalness detection took $O(p\,s)$ time. If one checks all the states, instead of those on a single path, the time required will be *exponential* in s, since this is the relation between the number of nodes in a tree and its depth. Furthermore, this amount of time will be required in all successor

generations, not just in pathological cases. Since worst-case complexity is being considered, one must assume that no reduction in the overall number of states is obtained, (which may well be the case most of the time,) and one is left with a worse algorithm. Obtaining the average-case complexity is much harder. How the number of states varies with p and c cannot be determined with this information alone.

As part of this study, a simple "extension" to QSIM, where a (possibly incomplete) "target" state description may be included in the input, has been implemented. QSIM runs as usual, but only behavior(s) which are paths from the initial to the target state (if any) are output. All that is required is that each new state be compared with the target description, which is $O(p)$.

Many researchers are working towards improving QSIM's performance, in order to apply it to large-scale systems.

Correctness

Kuipers has proven that QSIM's inference is sound, but incomplete. That is, all the actual behaviors of a real system modeled by the QDE are predicted by QSIM, but there may be some *spurious* output behaviors which do not correspond to any actual behaviors.

Soundness is proven as follows: QSIM starts with a correct description of the system state. At each next state generation, all the possible transitions of each parameter are generated, and only the combinations that do not satisfy the constraints (the system's laws) are filtered out. So there is no way that a real state can be absent from the output. Note that this proof is valid within a single operating region.

Incompleteness is proven by exhibiting a constraint set which causes QSIM to produce some spurious behaviors. The example Kuipers gives in [14] is particularly interesting: In the spring/block system of Figure 3.1.4, X is the horizontal position of the block, (*0* when the spring is "at ease,") V and A are velocity and acceleration, respectively.

This system, defined by the constraints of Table 3.1.11, and initial state

X = <*0*, inc>,
V = <*V**, std>,
A = <*0*, dec>,

where $V*$ is a positive landmark, is periodic, i.e. has a single actual behavior, that shown in Table 3.1.12. But QSIM computes an infinite number of behaviors, the first branching occurring at t_4, as shown in Tables 3.1.13 and 3.1.14. The algorithm has no way of knowing that the values 0 and $V*$ of the parameters X and V "belong" together, (they are *not* corresponding values) and mistakenly tries out all transitions for V. Ways of eliminating spurious behaviors are active topics of research. A way of eliminating some spurious predictions of pure QSIM will be introduced in Chapter 4. For the spring system, Kuipers says [14] :

"By changing the problem to take into account of conservation of total energy, an expanded view of the spring mechanism allows QSIM to determine that there is a single, periodic behavior. A physicist can look at [the spring QDE] and recognize or derive the fact that it represents an energy conserving system, and therefore that the behavior must be periodic. Part of this knowledge is the ability to recognize the physical system described by a set of constraints, and to know that there is a better structural description for it; one which adds parameters and constraints (e.g. energy) that illuminate the actual behavior. This approach takes us outside the realm of qualitative simulation, and into the realm of problem formulation. [...]

Returning to the larger problem of qualitative causal reasoning about mechanisms, an important problem is to *formulate* a suitable set of constraints given a physical situation, using the device-topology approach of de Kleer, Brown and Bobrow, the process-based approach of Forbus, or some approach yet to be discovered."

Such an approach is presented in Chapter 6.

Figure 3.1.4. The spring/block system

TABLE 3.1.11. QDE of spring/block system

CONSTRAINT	CVs
DERIV(X, V)	
DERIV(V, A)	
M-(A, X)	$(0,0)$

TABLE 3.1.12. Stable oscillatory behavior of spring/block system

X	V	A	time
<*0*, inc>	<*V**,std>	<*0*, dec>	t_0
<(*0*, ∞), inc>	<(*0*,*V**),dec>	<(-∞,*0*), dec>	(t_0,t_1)
<*X1*, std>	<*0*, dec>	<*A1*, std>	t_1
<(*0*,*X1*), dec>	<(-∞,*0*), dec>	<(*A1*,*0*), inc>	(t_1,t_2)
<*0*, dec>	<*V1*, std>	<*0*, inc>	t_2
<(-∞,*0*), dec>	<(*V1*,*0*), inc>	<(*0*,∞), inc>	(t_2,t_3)
<*X2*, std>	<*0*, inc>	<*A2*, std>	t_3
<(*X2*,*0*), inc>	<(*0*,*V**), inc>	<(*0*,*A2*), dec>	(t_3,t_4)
<*0*,inc>	<*V**,std>	<*0*, dec>	t_4

Cycle detected: States at t_0 and t_4 are identical.

Quantity space of X:	{-∞, *X2*, *0*, *X1*, ∞}
Quantity space of V:	{-∞, *V1*, *0*, *V**, ∞}
Quantity space of A:	{-∞, *A1*, *0*, *A2*, ∞}

TABLE 3.1.13. A spurious behavior of spring/block system (first nine states)

X	V	A	time
<*0*, inc>	<*V**,std>	<*0*, dec>	t_0
<(*0*,∞), inc>	<(*0*,*V**),dec>	<(-∞,*0*), dec>	(t_0,t_1)
<*X1*, std>	<*0*, dec>	<*A1*, std>	t_1
<(*0*,*X1*), dec>	<(-∞,*0*), dec>	<(*A1*,*0*), inc>	(t_1,t_2)
<*0*, dec>	<*V1*, std>	<*0*, inc>	t_2
<(-∞,*0*), dec>	<(*V1*,*0*), inc>	<(*0*,∞), inc>	(t_2,t_3)
<*X2*, std>	<*0*, inc>	<*A2*, std>	t_3
<(*X2*,*0*), inc>	<(*0*,*V**), inc>	<(*0*,*A2*), dec>	(t_3,t_4)
<*0*,inc>	<*V2*, std>	<*0*, dec>	t_4

Quantity space of X:	{-∞, *X2*, *0*, *X1*, ∞}
Quantity space of V:	{-∞, *V1*, *0*, *V2*, *V**, ∞}
Quantity space of A:	{-∞, *A1*, *0*, *A2*, ∞}

TABLE 3.1.14. Another spurious behavior of spring/block system (first nine states)

X	V	A	time
<*0*, inc>	<*V**,std>	<*0*, dec>	t_0
<(*0*,∞), inc>	<(*0*,*V**),dec>	<(-∞,*0*), dec>	(t_0,t_1)
<*X1*, std>	<*0*, dec>	<*A1*, std>	t_1
<(*0*,*X1*), dec>	<(-∞,*0*), dec>	<(*A1*,*0*), inc>	(t_1,t_2)
<*0*, dec>	<*V1*, std>	<*0*, inc>	t_2
<(-∞,*0*), dec>	<(*V1*,*0*), inc>	<(*0*,∞), inc>	(t_2,t_3)
<*X2*, std>	<*0*, inc>	<*A2*, std>	t_3
<(*X2*,*0*), inc>	<(*0*,*V**), inc>	<(*0*,*A2*), dec>	(t_3,t_4)
<(*X2*,*0*), inc>	<*V**, inc>	<(*0*,*A2*), dec>	t_4

Quantity space of X: {-∞, *X2*, *0*, *X1*, ∞}
Quantity space of V: {-∞, *V1*, *0*, *V**, ∞}
Quantity space of A: {-∞, *A1*, *0*, *A2*, ∞}

3.1.4. QSIM as Simulation

Qualitative simulation is interesting when viewed from the classical simulationist's perspective. Numerical simulation can be divided into two different kinds: Continuous and discrete. Continuous simulation is suitable for systems whose values are changing at all time points, and which have been formulated as differential equations. Typically, these are fluid, mechanical, thermal, or electrical circuit systems. Discrete simulation, on the other hand, deals with event-driven systems, described by discrete event models, whose states change discretely and are constant at intermediate points. Various queueing and "traffic" problems are suitable for this kind of simulation, in which stochastic processes are employed to model the random influences on the system. Because of these differences, the algorithms used for continuous and discrete simulation are markedly different. While continuous simulation algorithms focus on more efficient ways of computing the next set of values that satisfy the equations, discrete simulation's main concern is to schedule the "next" event that will occur within the system.

The QSIM method, although its area of application is almost exactly that of continuous simulation as described above, involves an algorithm that resembles the discrete-event kind of numerical simulation algorithms. In QSIM, just as in discrete numerical simulation, time is "advanced" between distinguished time-

points in which what can be called an "event" (a qualitative value change) occurs. The qualitative representation allows the algorithm to treat time intervals in which the parameters are changing between landmark values as single states. QSIM's transition rules may be viewed as analogues of discrete event state-change rules. On the other hand, QSIM inputs have no "unknown" or stochastic component in the sense of discrete simulation. Unlike any kind of numerical simulation, a QSIM run may in general produce more than one future behaviors, when the algorithm cannot decide which of a number of transitions will occur earliest. This is desirable in certain applications for which qualitative reasoners may be used; for example, in tutoring systems, where explaining the changes in the behavior of the system in response to a parameter value exceeding or staying below a given threshold is important.

3.2. Extensions to QSIM

Kuipers and his colleagues have developed many extensions to the pure QSIM algorithm. Several of these will be briefly summarized in this section. Some parts of the present work (see Chapters 4 and 5) is also comprised of improvements or extensions to QSIM; a comparison with the research described here will be possible.

3.2.1. Time-Scale Abstraction

One way to deal with the execution time problem that QSIM faces when the input system is big, (i.e. has many parameters,) is to decompose it to many small systems. This can be achieved when it is known that certain subsets of the constraint set representing various "mechanisms" in the system operate at widely different *time-scales*. Kuipers [41] has proposed a method of time-scale abstraction of systems, based on this idea. In this setup, mechanisms identified as "fast" view "slower" ones as constant, whereas the slow ones view faster ones as instantaneous. The whole collections of constraints representing the fast mechanisms are abstracted to single M+ or M- constraints in the slower ones.

Simulation begins with the fastest mechanism. That mechanism is simulated until equilibrium. Parameters of this mechanism shared with the next fastest mechanism pass their values to that one, and so the whole system can be simulated as a composite of manageable subsystems in this manner.

3.2.2. Ways of Producing Smaller Trees

Kuipers and Chiu ([42]; also see [18]) present two different methods to eliminate a situation that sometimes arises: QSIM may produce an intractably large tree, in which most behaviors of the system are practically the same, the only difference between them being the behavior of one (or a few) parameters, which are "chattering." A parameter *chatters* (i.e. it may increase, then stop, then decrease, stop again, decrease again, etc.) if the constraints are satisfied for all the qualitative directions that it may take. For example, if ADD(X,Y,Z), and

$$X=\langle(0, \infty), \text{dec}\rangle \quad \text{and} \quad Y=\langle(-\infty,0), \text{inc}\rangle,$$

and there are no other constraints involving Z, it chatters. (Actually, it does more than that, since it can take any qualitative *magnitude* as well, because of the magnitudes of X and Y in the example, and the ambiguity of qualitative addition.) This, in itself, is not a spurious prediction, there may be real systems which correspond to each behavior. The trouble is, chattering causes QSIM to produce such large trees that the size of the output reduces its usefulness.

The first method of handling this problem is simple: The user specifies that he does not care about the directions that certain parameters (those likely to chatter) may take on. QSIM then uses only the symbol **ign** (standing for "ignore") to represent these parameters' directions. So changes in their directions do not cause branchings in the tree, and simulation ends in a reasonable time. Additional care is taken to ensure that unreal behaviors for these parameters are not predicted, i.e. that a continuous assignment of **inc**, **std** or **dec**s for all the **ign**s exists.

The second method requires making the *sign-equality assumption* about the system being simulated. This practically means that all the M constraints in the QDE reflect linear functional relationships. Once this assumption is made, automatic calculation of the higher-order derivatives of the system parameters is possible, and this information is used to eliminate branchings in the tree resulting from qualitative direction ambiguities. The directions are derivatives,

and knowledge of *their* derivatives restricts the possible values they can take on. Application of this method thus involves a round of preprocessing in which the QDE is checked for parameters likely to chatter, and a qualitative algebraic manipulator is employed to derive expressions for the signs of the higher order derivatives. During QSIM's execution, the *HOD* (Higher Order Derivative) constraint has to be satisfied, as well as the "classical" types of constraints. Since, in general, M constraints may not obey the sign-equality assumption, higher-order derivative filtering is not *conservative*, i.e., it may eliminate consistent transitions. The method for spurious behavior reduction that will be presented in Chapter 4 entails conservative filters.

3.2.3. Incorporation of Quantitative Knowledge into QSIM

Kuipers and Berleant [43,44] introduced a way for including available *quantitative* information into QSIM's reasoning, and produced the Q2 program. Two kinds of quantitativeness are allowed: 1) Numerical values of the possible lower and upper bounds of the ranges that landmarks lie in, and, 2) Bounds on the "shapes" of the functions represented by the M constraints. One can specify upper and lower *envelopes* (numerically computable partial functions) for each monotonic constraint, meaning that the value of its function is to remain between the two envelopes. Bounds on the first and second derivatives of these functions can also be specified.

Q2 does the following: QSIM runs as usual. The behaviors produced are examined, making use of the quantitative information. This information can be propagated across the algebraic equations implied by the relations between the landmarks. Some enhanced quantitative knowledge about the landmarks (and even the time points) can be obtained from this propagation. In some cases, contradictions result, which mean that that behavior is impossible to occur, given the quantitative knowledge available. This results in that behavior being pruned off, leading to a less ambiguous output.

3.2.4. Qualitative Phase Space and Other Constraints

The *phase space* representation is well-known in mathematics, and can be used practically in relation to differential equations. In the phase space, there is an

axis for each independent parameter of the system. Each state (tuple of parameter values) of the system corresponds to a point in phase space. Various time behaviors of the system can then be represented as trajectories in the space. There is a theorem which states that trajectories in phase space cannot cross themselves, unless they are closed curves [45,46].

This knowledge, if adequately represented, can enable QSIM to rule out some spurious predictions. Lee and Kuipers [19] report on an extension to QSIM which does just that. They use the above-mentioned theorem as an additional type of constraint, named the *non-intersection* constraint. Their implementation of it is limited to second-order systems. Interesting considerations arise because the "points" in this *qualitative* phase space can be points, line segments, or even rectangles, because of the nature of qualitative magnitudes.

Other proposals for reducing the number of spurious behaviors are also mentioned in [19]. These involve adding yet other constraints to the QDE, making comparisons of the energy and phase properties of an oscillatory system at different extreme points possible [15]. Note that these are *system-specific* features, not generally applicable to an arbitrary model.

3.2.5. Using QP Theory to Build QSIM Models

Recognizing the importance of the task of *building* the qualitative models in the first place, Crawford et al. [26] present QPC; a compiler which can assemble model fragments into QDEs. Knowledge about the current situation in the physical scene of interest is input; this information is used to identify the model fragments relevant to the currently active processes. (Section 2.2.1) Use of QP theory and the Closed World Assumption is made to obtain the constraint set. QSIM then runs on those constraints until an operating region transition occurs. In that case, control returns to QPC, and the new processes and quantities are identified to obtain a new constraint set that will describe the new operating region. QSIM is employed with this new QDE, and this model building/simulation cycle continues until the system reaches quiescence.

The difference between this form of model building and the QSI method that will be presented in Chapter 6 is fundamental: Here, the relations that apply in various physical situations are stored in a library, and the model

builder's task is to find the relevant constraints and bring them together to form the QDE, given a description of the configuration of the system at a single (the "current") state. QSI, on the other hand, has no such library; the constraints are *inferred* from accounts of the behavior of the system over a time period.

3.2.6. QSIM for Monitoring

Dvorak and Kuipers [35] have reported on MIMIC, a method of monitoring dynamic systems. Monitoring, in this context, is a form of diagnosis (i.e, the task of understanding that something is wrong and identifying the problem,) that has to be performed while the system is operating, since it may be too expensive or impossible to shut down. Another issue with monitoring is that only a few of the parameters can be observed.

The basic idea of MIMIC is as follows: The system's (visible) parameters are observed over time. All the while, QSIM (with incomplete quantitative reasoning capability, see Section 3.2.3.) is parallelly run with the "normal" and a number of "faulty" models of the system. According to the observations, some of these models can be discarded if their predictions are not being satisfied, and new "suspected" fault models can be activated. The currently "active" models (i.e., the ones whose simulations are producing results that match the observations) are reported to the system operators.

This approach requires somebody to write down the normal and (all kinds of) faulty models of all types of components in the system. A great number (but still, only a fraction of all the possibilities) of system models with various kinds of faults are then built from these component models. QSIM is run on all these models and from each possible initial state, to produce a complete state tree for each case. Now that the "states" (collections of visible values) that can be created at some time by each kind of fault under consideration are known, they are stored in a *decision tree* that MIMIC will use during its operation to see which faults may be producing the observed behavior. So all the procedure described in this paragraph has to be completed before monitoring by MIMIC even starts.

A simpler approach to the monitoring problem is seen to be desirable. QSI's (Chapter 6) capability of finding the constraints that hold in a sequence of states may provide an answer.

IV. IMPROVED CONSISTENCY FILTERING FOR QUALITATIVE SIMULATION

As demonstrated in the previous chapter, qualitative simulation programs may occasionally predict spurious behaviors. Among the reasons for this are the *local* nature of simulation (i.e, states are determined by their predecessors, and the farther past of the system is generally not considered,) and the inherent incompleteness of the information being dealt with. Methods for reducing the number of spurious predictions of pure QSIM have been proposed; these require restrictions on the possible relationships between the system's parameters, like the sign-equality assumption (Section 3.2.2,) or are system-specific [19] and not generally applicable. They generally necessitate major additions to the algorithm. In this chapter, a modification to the algorithm which allows it to detect and eliminate a class of spurious behaviors is introduced. This method has none of the above-mentioned requirements, since it makes use of information already possessed but not used by pure QSIM.

4.1. A Class of Spurious Behaviors

This section contains examples showing the kind of spurious behavior that will be dealt with in this chapter. Two systems, in both of which little balls are thrown upwards at the beginning, will be considered.

4.1.1. Example 1: The Elevator/Ball System

Consider a system consisting of a ball thrown upward in a descending elevator (Figure 4.1.1.) The elevator is going down with constant speed in an underground shaft. The parameter representing the elevator's upward velocity, EL_V, is therefore fixed at a negative landmark value. The elevator floor's height, EL_H, is at the negative landmark elh_0 at the beginning of the simulation, t_0. Ground level corresponds to zero in EL_H's quantity space. One is interested in the ball's height relative to the elevator floor; the parameter REL_H represents this quantity. Two positive landmarks of REL_H are known; $relh_0$ is the relative height of the ball at t_0, and win_h is the height of the elevator's window ($relh_0 < win\_h$). The parameter BALL_H is the height of the ball relative to the Earth; zero is ground level. Note that, as a result of these definitions,

$$\text{EL\_H} + \text{REL\_H} = \text{BALL\_H}$$

at all times. BALL_H has two negative landmarks; blh_0 is its magnitude at t_0 (therefore, $elh_0 + relh_0 = blh_0$) and blh_1 is another landmark, such that $blh_0 < blh_1$. The ball's flight is governed by gravity. The upward acceleration of all falling bodies, including the ball, is the parameter ACC, which has the constant magnitude g (a negative landmark.) The velocities of the ball relative to the elevator, REL_V, and relative to the Earth, BALL_V, are also parameters.

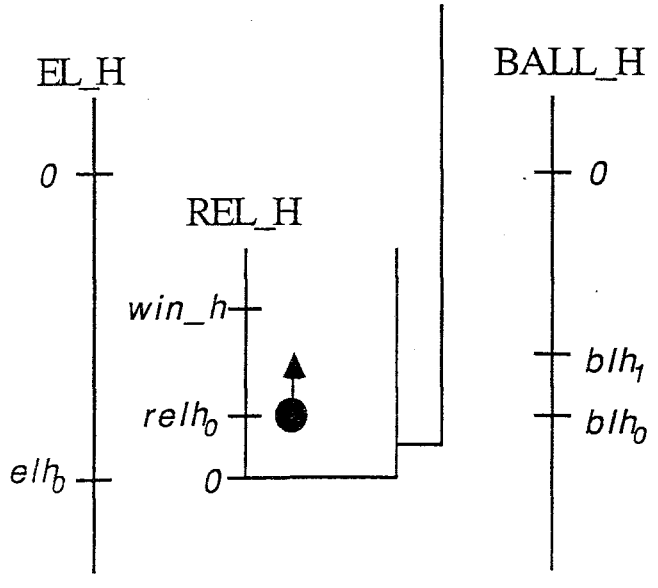


Figure 4.1.1. The elevator/ball system at t_0

This system is described by the constraints shown in Table 4.1.1.

TABLE 4.1.1. QDE of elevator/ball system

CONSTRAINT	CVs
DERIV(EL_H, EL_V)	
DERIV(REL_H, REL_V)	
DERIV(BALL_H, BALL_V)	
DERIV(REL_V, ACC)	
DERIV(BALL_V, ACC)	
ADD(EL_H, REL_H, BALL_H)	$(elh_0, relh_0, blh_0)$
ADD(EL_V, REL_V, BALL_V)	

Note that various "realities" which would violate these constraints and are not relevant to the present discussion have not been modeled. So the shaft is infinitely deep, the elevator has no ceiling which might interrupt the ball's flight, and one does not care what happens after the ball hits the elevator floor.

Now consider Table 4.1.2. This is part of one of the behaviors that pure QSIM predicts when started on this system with the initial state described above. Only the values and quantity spaces of the height parameters have been shown in the table. This behavior is actually "longer," i.e, there is more to come after $t4$, however, the sequence in the table is enough for the present discussion. (QSIM can be made not to generate the successors of nodes which are "deeper" in the tree than a specified level, by the inclusion of a simple control.)

TABLE 4.1.2. Spurious behavior of elevator/ball system

EL_H	REL_H	BALL_H	time
$<elh_0$, dec>	$<relh_0$, inc>	$<blh_0$, inc>	t_0
$<(-\infty, elh_0)$, dec>	$<(relh_0, win\_h)$,inc>	$<(blh_0, blh_1)$, inc>	(t_0,t_1)
$<(-\infty, elh_0)$, dec>	$<win\_h$, inc>	$<blh_1$, inc>	t_1
$<(-\infty, elh_0)$, dec>	$<(win\_h, \infty)$, inc>	$<(blh_1, 0)$, inc>	(t_1,t_2)
$<(-\infty, elh_0)$, dec>	$<(win\_h, \infty)$, inc>	<NewB, std>	t_2
$<(-\infty, elh_0)$, dec>	$<(win\_h, \infty)$, inc>	$<(blh_1, NewB)$, dec>	(t_2,t_3)
$<(-\infty, elh_0)$, dec>	<NewR, std>	$<(blh_1, NewB)$, dec>	t_3
$<(-\infty, elh_0)$, dec>	$<(win\_h, NewR)$,dec>	$<(blh_1, NewB)$, dec>	(t_3,t_4)
$<(-\infty, elh_0)$, dec>	$<win\_h$, dec>	$<(blh_1, NewB)$, dec>	t_4

Quantity space of EL_H: $\{-\infty, elh_0, 0, \infty\}$
Quantity space of REL_H: $\{-\infty, 0, relh_0, win\_h, NewR, \infty\}$
Quantity space of BALL_H: $\{-\infty, blh_0, blh_1, NewB, 0, A2, \infty\}$

To see that this is indeed a spurious behavior, inspect the parameter values at t_1 and t_4. At t_1, the ball is passing the level of the elevator's window in its upward journey, and at that moment, its "absolute" height is known to be blh_1. As expected, the ball rises for a while, and then starts descending, passing by the window for a second time at t_4. QSIM predicts that at that time, the ball's (and so the window's) absolute height is in the interval $(blh_1, NewB)$, in other words, higher than blh_1. But this does not make sense, since the elevator is known to be descending all along, its window cannot be higher at t_4 than at t_1. So the behavior of Table 4.1.2. is a spurious prediction; this sequence of events will never occur.

4.1.2. Example 2: The Ball/Shadow System

As a second example, the behavior of a little ball thrown upward from the ground, and the position of its shadow on the ground will be examined. (See Figure 4.1.2.) A small and powerful light source is fixed at a location of a certain height to the left of the point of takeoff of the ball. It is assumed that the ball can never reach the height of the light source. The height, velocity, and acceleration of the ball are parameters Y, V, and A, respectively. A is fixed at a negative landmark, like in the previous example. One is also interested in the position of the ball's shadow on the ground, represented by parameter X. *0* (zero) is the point of takeoff of the ball in both X and Y's quantity spaces. The ground is level (i.e. has no "bumps") so that X is a reasonable function. The highest altitude that the ball has ever reached before is the landmark *alt_rec* in Y's quantity space. There is a dead bug lying at a point to the right of the ball's takeoff point. X has the positive landmark *bug_pt* when the shadow is on the bug. Light travels infinitely fast (for the commonsense time scale at which the system is being viewed, of course.) The set of constraints is that of Table 4.1.3.

TABLE 4.1.3. QDE of ball/shadow system

CONSTRAINT	CVs
DERIV(V, A)	
DERIV(Y, V)	
M+(X,Y)	$(0, 0)$

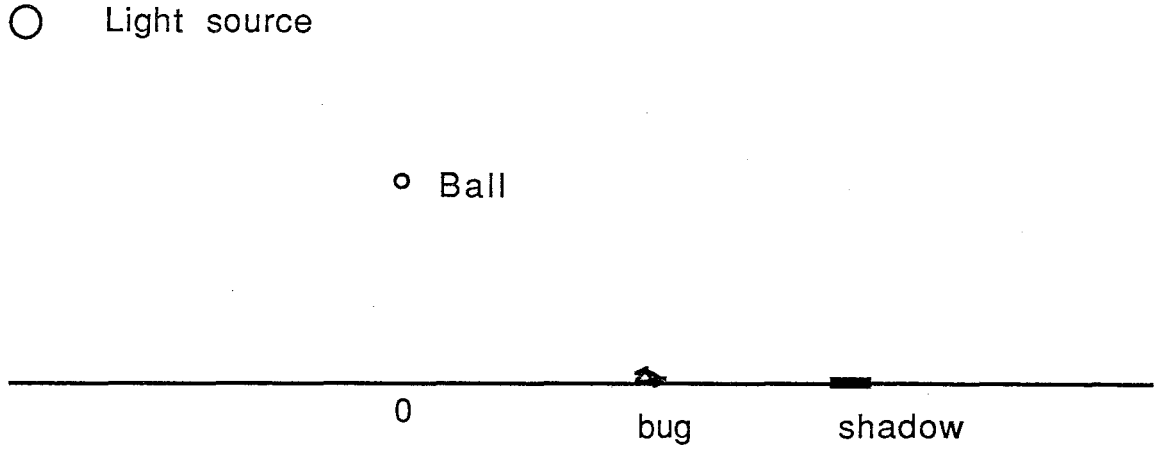


Figure 4.1.2. The ball/shadow system

The ball is shot up with initial velocity v_0 at t_0. Table 4.1.4 contains part of one of the spurious behaviors that QSIM predicts.

What is wrong with this behavior? The ball is shot up at t_0. At t_1, it breaks the old altitude record and goes on climbing. At t_2, when the ball is at a point above *alt_rec*, its shadow falls on the bug. At t_3, both the ball and its shadow stop for an instant, and their magnitudes at that point are recorded as CVs. After that, the ball starts going down, crossing *alt_rec* at t_4. But the shadow has still not reached the bug for a second time. This is inconsistent with the available knowledge about the function from Y to X at t_1, so Table 4.1.4 contains a spurious behavior.

TABLE 4.1.4. Spurious behavior of ball/shadow system

Y	V	A	X	time
<*0*, inc>	<v_0, dec>	<*g*, std>	<*0*, inc>	t_0
<(*0*, *alt_rec*), inc>	<(*0*, v_0), dec>	<*g*, std>	<(*0*, *bug_pt*), inc>	(t_0,t_1)
<*alt_rec*, inc>	<(*0*, v_0), dec>	<*g*, std>	<(*0*, *bug_pt*), inc>	t_1
<(*alt_rec*, ∞), inc>	<(*0*, v_0), dec>	<*g*, std>	<(*0*, *bug_pt*), inc>	(t_1,t_2)
<(*alt_rec*, ∞), inc>	<(*0*, v_0), dec>	<*g*, std>	<*bug_pt*, inc>	t_2
<(*alt_rec*, ∞), inc>	<(*0*, v_0), dec>	<*g*, std>	<(*bug_pt*, ∞), inc>	(t_2,t_3)
<*NewY*, std>	<*0*, dec>	<*g*, std>	<*NewX*, std>	t_3
<(*alt_rec*, *NewY*),inc>	<(-∞, *0*), dec>	<*g*, std>	<(*bug_pt*, *NewX*), dec>	(t_3,t_4)
<*alt_rec*, dec>	<(-∞, *0*), dec>	<*g*, std>	<(*bug_pt*, *NewX*), dec>	t_4

Quantity space of X: {-∞, *0*, *bug_pt*, *NewX*, ∞}
Quantity space of Y: {-∞, *0*, *alt_rec*, *NewY*, ∞}

4.2. The Cause of Spurious Behaviors

The spurious behaviors of the previous section are not results of the "weakness" of the sets of constraints, or a fundamental shortcoming of qualitative simulation. All the information required for identifying that the qualitative states at t_4 (in both examples) are inconsistent with the past behaviors of the systems is available to the algorithm: The constraints between the parameters are given as input, and the states up to t_4 are created by the algorithm itself. The problem is the inability of the algorithm to utilize this information.

QSIM uses corresponding value information to check the consistency of newly proposed parameter magnitudes with previous knowledge about the arithmetic and functional relationships in the system. However, the tuple of magnitudes at t_1 in Section 4.1.1, and those at t_2 in Section 4.1.2, which allows one to recognize that something is wrong, are *not* recorded as CVs of the constraints by the algorithm. The reason for this is that only landmarks are allowed in CV tuples, and in both cases under discussion, there are parameters which have interval magnitudes at the times of interest. In fact, EL_H of the elevator example has an interval magnitude in each state after t_0 in all behaviors of the system, so no CV tuples will be recorded for any constraint involving it.

Moreover, it would not be of much use in the elevator example, even if pure QSIM recorded such CV tuples. Equation 3.1 is used as the ADD CV consistency check, which would, when comparing the values at t_4 and t_1, require that

$$((-\infty, elh_0) - (-\infty, elh_0)) + (win\_h - win\_h) = ((blh_1, NewB) - blh_1) \qquad (4.1)$$

be satisfied. This would reduce to

$$? + 0 = +$$

where "?" can be any one of +, 0, or -, so the consistency check would be satisfied, the state at t_4 would be generated, and the same spurious behavior would still be produced.

The reason for pure QSIM's insistence on landmarks as CVs may be the fact that neither qualitative subtraction nor (provided the signs are the same) qualitative division are ambiguous when one of the values participating in them is guaranteed to be a landmark. When both operands are allowed to be intervals, one may be faced with a situation where one has to compare an interval with itself, as in Equation 4.1, which leads to the condition being satisfied trivially.

However, it was demonstrated that the use of only-landmark CVs causes the generation of spurious behaviors. What is needed is a way of recording *interval CVs* (*ICVs*), and an extension of the consistency filtering and qualitative arithmetic rules to handle this generalization.

4.3. Filtering Spurious Behaviors

To enable it to use more of the available information about the system's arithmetic and functional relationships, the following changes have been made to pure QSIM, (Section 3.1.2) resulting in the *improved* QSIM algorithm, as it will be called in the rest of this text:

1) Whenever a new landmark is discovered for parameter P, augment the CV lists of constraints in which P participates to reflect this new information.

2) In Step 2.4 of the original algorithm, for every newly created state, add the parameter magnitudes appearing in all non-DERIV constraints as tuples to the CV lists of these constraints, regardless of whether any of the magnitudes are landmarks or not.

3) During the CV consistency filtering phase (in Step 2.2,) the tuples of proposed magnitudes will be eliminated unless they have the following relationships with all of their constraints' CV tuples:

i. For MINUS and M- constraints, if the proposed magnitude tuple is (m_A, m_B) and the CV tuple is (p, q), the signs of $(m_A - p)$ and $(m_B - q)$ should be the opposites of each other.

ii. For M+ constraints, if the proposed magnitude tuple is (m_A, m_B) and the CV tuple is (p, q), the signs of $(m_A - p)$ and $(m_B - q)$ should be the same.

iii. For ADD and MULT constraints, equations 3.1 and 3.2 should be satisfied, respectively.

iv. When a subtraction, or, (in the case of MULT,) division operation required for the above controls has an ambiguous result, the predicate in question will be automatically satisfied for that CV tuple.

4) Qualitative subtraction will yield one of +, 0, or -, according to the ordinal relation of its operands. However, when this relation cannot be determined from the operands, $(m_A - p)$ will yield:

+, if both its operands are the same interval magnitude, the parameter has had this same magnitude since the recording of p, and its direction is **inc**,

-, if both its operands are the same interval magnitude, the parameter has had this same magnitude since the recording of p, and its direction is **dec**,

?, (the ambiguous value,) otherwise.

5) Qualitative division will yield one of **lto**, **one**, or **gto**, according to the signs and ordinal relation of its operands. However, when this relation cannot be determined from the operands, $\left(\frac{m_A}{p}\right)$ will yield:

gto, if both its operands are the same positive interval magnitude, the parameter has had this same magnitude since the recording of p, and its direction is **inc**,

gto, if both its operands are the same negative interval magnitude, the parameter has had this same magnitude since the recording of p, and its direction is **dec**,

lto, if both its operands are the same positive interval magnitude, the parameter has had this same magnitude since the recording of p, and its direction is **dec**,

lto, if both its operands are the same negative interval magnitude, the parameter has had this same magnitude since the recording of p, and its direction is **inc**,

?, (the ambiguous value,) otherwise.

As an example to (1), consider the behavior of BALL_H in Table 4.1.2. In (t_1,t_2), the ball is in the interval $(blh_1, 0)$. This fact is recorded in the CV list of the ADD, as modification (2) requires. At t_2, BALL_H stops at its new landmark, $NewB$. This tells one that the *previous* magnitude of BALL_H can be more

correctly described as $(blh_1, NewB)$, and the related CV tuple is changed to contain the new information. (Actually, this application of (1) does not contribute to the elimination of the spurious behavior in this example. A "system" where (1) is necessary to prune the spurious prediction will be presented at the end of this section.)

The rationales for (3.i) and (3.ii) stem from the nature of inverse and direct proportionality, respectively. If M-(A,B), then for every CV tuple (A_mag, B_mag) of this constraint, A and B's magnitudes should always be at opposing sides of A_mag and B_mag, (unless they are both exactly "on" them,) because they always "go" in opposing directions. (When A_mag and B_mag are both landmarks, this is very easy to see, the rules of (4) and (5) allow it to be generalized to interval CVs.) For the same reason, parameters linked by M+s should always be at the same sides of their CVs.

The justification for the rules of (4) and (5) is as follows: For any reasonable function f, if $f'(t) > 0$ for all $t \in [t_{beg}, t_{end})$, then $f(t_1) < f(t_2)$ for all $t_1, t_2 \in [t_{beg}, t_{end})$ such that $t_1 < t_2$. (An analogous proposition holds when $f'(t) < 0$; then $f(t_1) > f(t_2)$.) That is, if a quantity is continuously increasing (decreasing) in an interval, its value at a later time in that interval will be greater (less) than its value at an earlier time in the same interval. Whether the parameter has had the same magnitude at all times since the recording of the CV to the proposal of the new values can be checked using information in the CV list or the state tree.

Table 4.3.1 shows the calculations made by the improved algorithm when the spurious state at t_4 is proposed in the elevator simulation. Each new CV triple is added to the beginning of the list, so older values are used later in the check. As can be seen in the table, Equation 3.1 fails to be satisfied when the proposed tuple is checked against the CV of t_1. As a result of this, a state containing that combination of magnitudes will not be created, and the spurious behavior of Table 4.1.2 will not be predicted.

Table 4.3.2 similarly shows how the spurious state (and behavior) of Table 4.1.4 is eliminated. Every CV tuple's Y magnitude is subtracted from the proposed Y magnitude to obtain the sign of the Y difference. The same thing is done for X. According to (3.ii), the signs should agree. When the CV triple of t_2 is considered, they definitely do not, so the proposed magnitudes are wrong.

TABLE 4.3.1. CV triples used to test ADD(EL_H,REL_H,BALL_H) at t_4

proposed triple: $((-\infty, elh_0), win\_h, (blh_1, NewB))$

CV triple	Equation 3.1	O.K.?
$((-\infty, elh_0), (win\_h, NewR), (blh_1, NewB))$	(-) + (-) = (-)	Yes
$((-\infty, elh_0), NewR, (blh_1, NewB))$	(-) + (-) = (-)	Yes
$((-\infty, elh_0), (win\_h, NewR), (blh_1, NewB))$	(-) + (-) = (-)	Yes
$((-\infty, elh_0), (win\_h, NewR), NewB)$	(-) + (-) = (-)	Yes
$((-\infty, elh_0), (win\_h, NewR), (blh_1, NewB))$	(-) + (-) = (?)	Yes
$((-\infty, elh_0), win\_h, blh_1)$	(-) + (0) = (+)	NO

TABLE 4.3.2. CV tuples used to test M+(Y,X) at t_4

proposed tuple: $(alt\_rec, (bug\_pt, NewX))$

CV tuple	Sign of Y diff.	Sign of X diff.	Signs equal?
$((alt\_rec, NewY), (bug\_pt, NewX))$	-	-	Yes
$(NewY, NewX)$	-	-	Yes
$((alt\_rec, NewY), (bug\_pt, NewX))$	-	?	Yes
$((alt\_rec, NewY), bug\_pt)$	-	+	NO

A spurious prediction which cannot be eliminated without modification (1) is presented here. Consider a (decidedly very simple) model, consisting of three parameters A, B, and C, with a single constraint among them: ADD(A,B,C). Table 4.3.3 is a spurious prediction of QSIM about this system.

Like in the previous examples, the last state (the one at t_5) is spurious. To detect this, the information at t_1 must be considered together with that of t_5. At t_5, the A magnitude is less than at t_1. The C magnitudes are the same. So B at t_5 has to be greater than B at t_1 for the ADD to hold. At first sight, it seems one cannot decide on the relative ordering of $(b1, NewB)$ and $(b1, b2)$, and the CV check will be satisfied as it is in ambiguous situations, but the fact that B stops at $NewB$ at time point t_2 after decreasing in $(b1, b2)$ in the interval (t_0,t_2) tells one that B was actually in $(NewB, b2)$ in that time period. Now it is clear that the tuple of t_5 is inconsistent. Without modification (1), the algorithm cannot

detect this inconsistency. Table 4.3.4 shows the invocations of Equation 3.1 performed in the CV filtering of this state.

TABLE 4.3.3. A spurious prediction

initial quantity space of A: $\{-\infty, 0, a1, a2, a3, \infty\}$
initial quantity space of B: $\{-\infty, 0, b1, b2, \infty\}$
initial quantity space of C: $\{-\infty, 0, c1, c2, c3, \infty\}$

A	B	C	time
<$a3$, dec>	<$b2$, dec>	<$c3$, dec>	t_0
<($a2$, $a3$), dec>	<($b1$, $b2$), dec>	<($c2$, $c3$), dec>	(t_0,t_1)
<$a2$, dec>	<($b1$, $b2$), dec>	<$c2$, dec>	t_1
<($a1$, $a2$), dec>	<($b1$, $b2$), dec>	<($c1$, $c2$), dec>	(t_1,t_2)
<($a1$, $a2$), dec>	<$NewB$, std>	<($c1$, $c2$), dec>	t_2
<($a1$, $a2$), dec>	<($b1$, $NewB$), dec>	<($c1$, $c2$), dec>	(t_2,t_3)
<$a1$, std>	<($b1$, $NewB$), dec>	<($c1$, $c2$), dec>	t_3
<($a1$, $a2$), inc>	<($b1$, $NewB$), dec>	<($c1$, $c2$), dec>	(t_3,t_4)
<($a1$, $a2$), inc>	<($b1$, $NewB$), dec>	<$c1$, std>	t_4
<($a1$, $a2$), inc>	<($b1$, $NewB$), dec>	<($c1$, $c2$), inc>	(t_4,t_5)
<($a1$, $a2$), inc>	<($b1$, $NewB$), dec>	<$c2$, inc>	t_5

final quantity space of B: $\{-\infty, 0, b1, NewB, b2, \infty\}$

TABLE 4.3.4. CV triples used to test ADD(A, B, C) at t_5

proposed triple: $((a1, a2), (b1, NewB), c2)$

CV triple	Equation 3.1	O.K.?
$((a1, a2), (b1, NewB), (c1, c2))$	(+) + (-) = (+)	Yes
$((a1, a2), (b1, NewB), c1)$	(+) + (-) = (+)	Yes
$((a1, a2), (b1, NewB), (c1, c2))$	(+) + (-) = (+)	Yes
$(a1, (b1, NewB), (c1, c2))$	(+) + (-) = (+)	Yes
$((a1, a2), (b1, NewB), (c1, c2))$	(?) + (-) = (+)	Yes
$((a1, a2), NewB, (c1, c2))$	(?) + (-) = (+)	Yes
$((a1, a2), (NewB, b2), (c1, c2))$	(?) + (-) = (+)	Yes
$(a2, (NewB, b2), c2)$	(-) + (-) = (0)	NO

4.4. Correctness and Complexity of Improved QSIM

To prove that improved QSIM is actually better and not worse than pure QSIM, one has to show that:
1) Improved QSIM does not predict some of the spurious behaviors that pure QSIM predicts,
2) Improved QSIM does not predict any spurious behaviors that pure QSIM does not predict, and,
3) Improved QSIM is sound, i.e., it does not fail to predict any actual behavior.

(1) has already been proven by demonstration in the previous section. It is also easy to show (2), i.e., improved QSIM causes no extra spurious behaviors: Consider the changes that were made to the algorithm. They cause it to record and check more CVs than before. CV tuples which contain only landmark values (i.e., the only kind present in the pure version,) are handled in the same way as before, so all the spurious behaviors to whose eliminations they contribute in pure QSIM are also eliminated in improved QSIM. The only difference that improved QSIM has is that it checks proposed states against interval corresponding values; a control totally absent in pure QSIM. If no inconsistencies are detected in this extra checking, improved QSIM will give the same output as pure QSIM. Otherwise, it will predict less behaviors. In no case can it predict some behavior that pure QSIM does not predict. So (2) has been proven.

To prove (3), one has to show that the filtering criteria given in the previous section eliminate only those tuples which do not satisfy their constraints, i.e., that the improved filters are *conservative* (Section 3.2.2.)

For the ADD and MULT constraints, this is already proven, since Kuipers' equations 3.1 and 3.2 (Section 3.1.2) are used.

For the M+, M-, and MINUS constraints, the filtering criteria presented here subsume those of Kuipers, and are still conservative. Consider M+(A,B), the arguments for M- and MINUS are similar. A geometric interpretation of the

rules will be given. When intervals are allowed in CV tuples, each tuple defines either a point, a segment, or a rectangle in the plot of the two parameters against each other. It is known that the plot of A vs. B passes through these CV areas. Each proposed magnitude tuple also defines such an area, call it PA. For M+ to hold, some point in PA has to be "higher" than some other points in all the CV areas to the "left" of PA, and "lower" than some points in all the CV areas to the "right" of PA. That is because it is known that the plot will be "rising" as one goes from left to right in the graph (whichever way you look at it; A vs. B or B vs. A.) The requirement that both proposed magnitudes should be "on the same sides" of the values in all CVs embodies this necessity: *Both* coordinates of each point in the graph of an M+ function are either less or greater than the coordinates of other points. The filter eliminates tuples only when it is certain that they do not fulfill this condition. So no magnitudes which truly satisfy the M+ are eliminated. Consider the consistency check between the CV tuple $((alt\_rec, NewY), bug\_pt)$ and the proposed magnitude tuple $(alt\_rec, (bug\_pt, NewX))$ (Section 4.3) which leads to the elimination of the spurious behavior of the ball/shadow system. Figure 4.4.1 is the geometric interpretation of that check. In the figure, the horizontal segment is the proposed magnitude area, and the vertical segment is the CV area. Clearly, no monotonically increasing function can cross both these segments. Therefore, the proposed tuple cannot be accepted. For M- and MINUS, the reasoning is similar, with the slope of the curve now negative. In all cases, only values which violate the constraints are filtered out, so the filters are conservative.

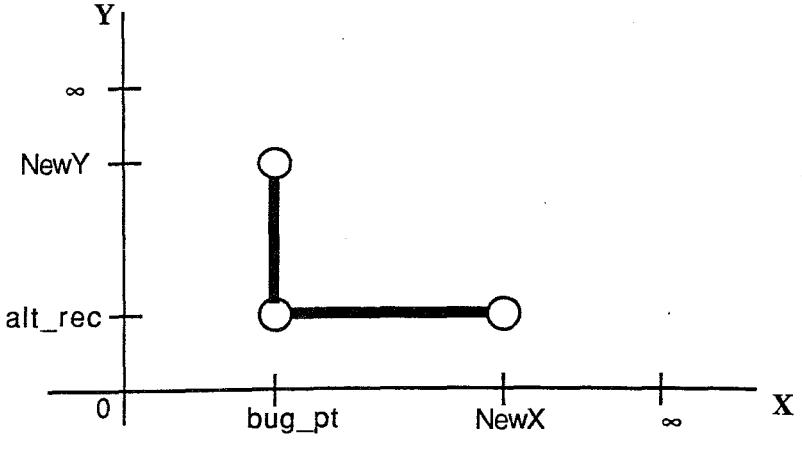


Figure 4.4.1. Geometric interpretation of M+ consistency check

Now, the complexity of the resulting algorithm will be considered. If c is the number of constraints and s is the current length of the behavior being generated, constraint filtering takes $O(cs)$ time in pure QSIM [14]. In the improved version, the task of determining whether there is an unbroken string consisting of the same magnitude between a proposed magnitude and its recorded counterpart can be done in parallel with the already present scanning of the CV list. This means that, although the improved version is using more CVs, the time required for using them is still on the order of cs. Updating the CVs after landmark discovery takes time linear in c. The worst-case time requirement of the algorithm is, again, caused by successor generation, and is exponential in the number of parameters.

TABLE 4.4.1. Execution times of improved QSIM case runs

Problem	number of constraints in input	number of parameters in input	number of operating region transitions	number of states in final tree (pure)	number of states in final tree (improved)	pure QSIM execution time (s)	improved QSIM execution time (s)
Single Ball	2	3	0	5	5	0.16	0.17
Kidney	8	10	0	3	3	5.78	5.85
Kidney	8	10	0	3	3	4.50	4.71
U-tube	18	7	1	8	8	2.24	2.30
Spring	6	6	0	9	9	6.79	6.93
Ball/ Shadow	3	4	0	79	63	8.37	6.78
Heat Exchanger	3	5	0	7	7	0.78	0.79

The fact that improved QSIM's additional space and time requirements are not significant has also been shown by case runs of both versions of the algorithm on various input systems, as can be seen in Table 4.4.1. In the table, the section numbers indicate the location in this text where the relevant QDE is described; some of these are famous examples from literature. The second and third problems are runs of the same model with different initial states. In cases where it is able to prune additional behaviors, improved QSIM also shows (naturally) a clear time and space advantage. As indicated in the table, the improved version eliminates 16 states when running on the ball/shadow problem. These states cause four spurious behaviors to be predicted by pure QSIM, in addition to the 11 "actual" behaviors. The "un-improved" version

creates such a huge output when confronted with the elevator problem that our PC implementation of QSIM (Appendix A) has to be modified to handle it; that is why that item is absent in the table. When restricted to produce the tree only until t_4, pure QSIM predicts 147 behaviors of the elevator/ball system, again four of which are spurious ones that are eliminated by the improved algorithm.

4.5. Concluding Remarks

In this chapter, it was shown how interval corresponding values can be used to strengthen existing qualitative simulation methods, and the exact list of modifications to QSIM which enable it to support and make use of ICVs were given. One must emphasize that improved QSIM does not detect and eliminate *all* spurious behaviors; Those of the spring system of Section 3.1.3 are still predicted, for instance. Kuipers and his colleagues have concentrated on this problem, and the non-intersection, energy, and system property constraints mentioned in Section 3.2.4 handle that system and many of its variants. This suggests that these constraints, the interval corresponding value modification, and also the HOD constraint of Section 3.2.2 can be used in conjunction to produce simulations tighter than any one of them can provide individually. In the following paragraphs, a justification for this claim is presented.

Such a unification of methods would not cause a harmful "interference" where the implementation of one idea hinders another, since each involves separate constraints. ICVs are to be applied only on the "classical" constraints, so the other ones are not affected. (The algorithm applies each constraint independently of the others. Even a single unsatisfied constraint causes a proposed state to be rejected, which is what one expects from the use of constraints. The addition or removal of one type of constraint does not affect the filtering properties or applicability of the other types.) Lee and Kuipers [19] state that all the additional constraint types can be used profitably together.

To show that the addition of the ICV modification will make simulation by the unified algorithm even tighter, it is sufficient to demonstrate a single spurious behavior which can only be detected by improved QSIM and not by the additional constraints. The ball/shadow system of Section 4.1.2 provides a good example. Derivatives of all necessary orders of the system parameters are included in that model, so the HOD constraint will not be used. The system is not oscillatory, so the energy and system property constraints will not be derived. No choice of independent variables produces a phase space with a self-intersecting trajectory, so the non-intersection constraint does not eliminate the spurious behavior either. Therefore, the claim about the utility of the proposed unified algorithm is justified.

Improved QSIM's input and output interfaces are exactly those of pure QSIM, so one can easily *replace* pure QSIM with it; in the applications mentioned in Section 2.2.5, Section 3.2, or the reasoners that will be described in the rest of this dissertation, leading to better performance of the final product.

V. POSTDICTION BY QUALITATIVE SIMULATION

5.1. Postdiction

Postdiction is the task of "inferring how a particular state of affairs might have come about." [12] This task has also been named *abductive projection* [2] and *retrodiction* [47] in the literature. The basic idea is to utilize the existing laws of change of the domain in the "reverse" of their normal direction; i.e. to obtain the causes from the results. If a law of change of the form "A results in B" exists, and B is a current fact, A may be inferred as a cause of B. This kind of inference is called *abduction*. Abduction is not a "legal" form of inference, i.e., it does not always produce correct results. (To put this more formally, it allows false conclusions from true axioms.) That is because there may be a lot of other things besides A that can result in B, and just knowing B does not necessitate that its cause was A.

Reasoners using a situation calculus representation of change for performing abductive projection (in story understanding [2], for instance) have to deal with this problem. There are simply too many laws of change that may lead to a given fact being true, (Suppose you see some blood on the street. Try to enumerate the possible courses of events that may have led to this.) and although, realistically, most of the laws will not even be "written down," (i.e., known to the reasoner,) the program still has to "go back" on all the laws that it does have, and to use other kinds of information to decide which "road" back is the most sensible one. When to stop generating causes that are further and further in the past is another important issue for these reasoners.

In this chapter, it is suggested to use a QSIM-like reasoner (actually, a modified version of QSIM itself) to perform postdiction. This will have the same advantages that pure QSIM has over a situation calculus-based reasoner in the prediction task: It will be able to handle continuously changing values, and the fact that its domain is much more specialized will lead to more efficient and elegant results. The problems mentioned above for the abductive projection managers will have natural solutions.

Some straightforward but important changes need to be made in the QSIM algorithm to make it perform postdiction; these are the subject of the next section.

5.2. QSIM for Postdiction

Most of the "mechanism" of QSIM used to construct the state tree (e.g. the constraint filtering phase) is independent of the direction in which time is "running" as new states are created. By restricting the changes to the algorithm to the modules which deal with the "passing" of time, the continued validity of the already present correctness proofs and complexity analyses of the remaining parts of QSIM will be ensured.

What is wanted is an algorithm which takes a "current state" of the system at t_0, together with the QDEs describing the system in all its operating regions, as input. The output of this algorithm will again be a state tree; with the state at t_0 as the root. Each node in this tree will be a possible temporal *predecessor* of its father, rather than a successor, as in the case of pure QSIM. The *interpretation* of this tree will also be different than that of pure QSIM.

There are two ways in which a state change can occur in QSIM: Either all the parameters obey the transitions of Table 3.1.1 and obtain new values, or a parameter exceeds its legal range and a new operating region is activated. "Reversing" the first kind of change to perform backward simulation is especially easy, while the second kind is more involved; as explained below, the

parts of the algorithm dealing with operating region changes have to be completely rewritten.

The modifications that will be made to QSIM algorithm, as given in Section 3.1.2, for performing postdiction are the following:

1) In Step 2.1, use Table 5.2.1 (instead of Table 3.1.1) to generate each parameter's possible transitions to the previous state.

2) For each newly created state, check whether this state can be the *first* state of the current operating region, i.e., whether this is a state that the system can have immediately after an operating region change. If this is possible, create the *last* state of the previous operating region as a possible predecessor of this state, and continue with the new region's QDE down that branch.

3) In Step 3, do not put a state in the list of states to be opened if any parameter in it is about to exceed its legal range (by the transitions of Table 5.2.1.)

4) In the resulting state tree, every path from a (leaf or non-leaf) node to the root is a distinct *possible past* of the system.

As can be seen, Table 5.2.1 is simply the reversed version of Table 3.1.1; all the possible transitions to previous values are obtained by reversing the arrow of time. Also note that the P-transitions have become I-transitions, and vice versa; this follows from their definitions in Section 3.1.1. The two "new landmark discovering" transitions have been moved to the new I-list, so the new algorithm has QSIM's ability to detect previously unknown and interesting parameter magnitudes, this time in the system's past. The reason for swapping Kuipers' original transitions P4 and P5 also becomes clear now; the part of the algorithm in Step 2.4, which checks the "no change" transitions {I1, I4, I7} can work correctly without being modified, since that set contains the same elements in both Table 3.1.1 and Table 5.2.1.

Whether an operating region change may have preceded the state at hand (say, S) or not can be checked as follows: All parameters which are designated (in the input) to have specific values at the start of the current operating region must have those values in S. Furthermore, there has to be another operating region description, Pr_OR, in the input; and a parameter in Pr_OR, which causes a transition into the current operating region when it exceeds its legal range. If these conditions are satisfied, a new state in Pr_OR

where that parameter is just exceeding its range is created. Parameters designated not to change values in this transition keep their values at S in the new state, while the remaining ones receive their values by completion according to the new constraints. See the next section for an example to this process.

TABLE 5.2.1. The reverse transitions

I-transitions

name	in (t_i , t_{i+1})	at t_i
I1	$< l_j$, std >	$< l_j$, std >
I2	$< (l_j , l_{j+1})$, inc >	$< l_j$, std >
I3	$< (l_{j-1} , l_j)$, dec >	$< l_j$, std >
I4	$< (l_j , l_{j+1})$, inc >	$< (l_j , l_{j+1})$, inc >
I5	$< (l_j , l_{j+1})$, inc >	$< l_j$, inc >
I6	$< (l_{j-1} , l_j)$, dec >	$< l_j$, dec >
I7	$< (l_j , l_{j+1})$, dec >	$< (l_j , l_{j+1})$, dec >
I8	$< (l_j , l_{j+1})$, inc >	$< l^*$, std >
I9	$< (l_j , l_{j+1})$, dec >	$< l^*$, std >

P-transitions

name	at t_{i+1}	in (t_i , t_{i+1})
P1	$< l_j$, std >	$< l_j$, std >
P2	$< l_{j+1}$, std >	$< (l_j , l_{j+1})$, inc >
P3	$< l_{j+1}$, inc >	$< (l_j , l_{j+1})$, inc >
P4	$< (l_j , l_{j+1})$, inc >	$< (l_j , l_{j+1})$, inc >
P5	$< l_j$, std >	$< (l_j , l_{j+1})$, dec >
P6	$< l_j$, dec >	$< (l_j , l_{j+1})$, dec >
P7	$< (l_j , l_{j+1})$, dec >	$< (l_j , l_{j+1})$, dec >

The intuition behind the rule of (3), which prevents the opening of a state in which a parameter is backing out of its legal range can be illustrated by the following example: If one sees a descending elevator, one may think that it was on the upper floor a short time ago. However, if one sees a descending elevator at the top floor, no such conclusion about the past can be made.

The definition of possible pasts reflects a fundamental difference between postdiction and prediction: One has no way of knowing when the system was "started up" during postdiction. The examples in the next section will help illustrate this issue.

5.3. Examples

This section will illustrate the working of the postdiction algorithm on two very familiar systems. Although very simple, these systems are able to reflect important features of the algorithm.

5.3.1. The Ball Postdiction

Once again, a ball thrown upwards from ground level will be considered. There is a single operating region, which is exited when the ball's height is about to become negative, (i.e., when it hits the ground.) The constraints are, as usual,

DERIV(Y,V)

and

DERIV(V,A),

where Y is the height, V is upward velocity, and A is the (fixed) acceleration. (Note that this particular set of constraints models a large family of actual systems; objects thrown at any angle, and balls on frictionless inclined planes being included. A vertically flying ball is the easiest of these to visualize. The concepts illustrated here are, of course, applicable to any QDE.)

Let the current state be

Y = <*0*, dec>,

V = <(-∞, *0*), dec>.

A = <*g*, std>,

i.e., the ball is hitting the ground. The postdiction algorithm produces a single-branched tree of five states when run with this input. That branch is: (values of Y shown only)

<0, dec> <-- <(0, ∞), dec> <-- <*NewY*, std> <-- <(0, *NewY*), inc> <-- <0, inc>

where the root is at the left end and the arrows show the well-known direction of time.

If possible pasts were paths between leaves and the root, the only one found in this example would be the following:

"The ball has been shot up from the ground, risen a while, then fallen back."

But intuition tells one that this is not the only possibility, given the current state and this QDE. The ball may have been dropped from an initial position above the ground, for instance; this would again result in the input state. The other possibilities are the ones where the ball is *shot* (i.e., given some initial nonzero velocity) upwards or downwards from a point above the ground.

Note that the state tree contains all these different possible pasts, according to the definition given in the previous section. As a result, the algorithm presents the information in Table 5.3.1 as the output in this case. (Again, only Y is shown.) Figure 5.3.1 shows the paths taken by the ball in each of the possible pasts (*PP*s.)

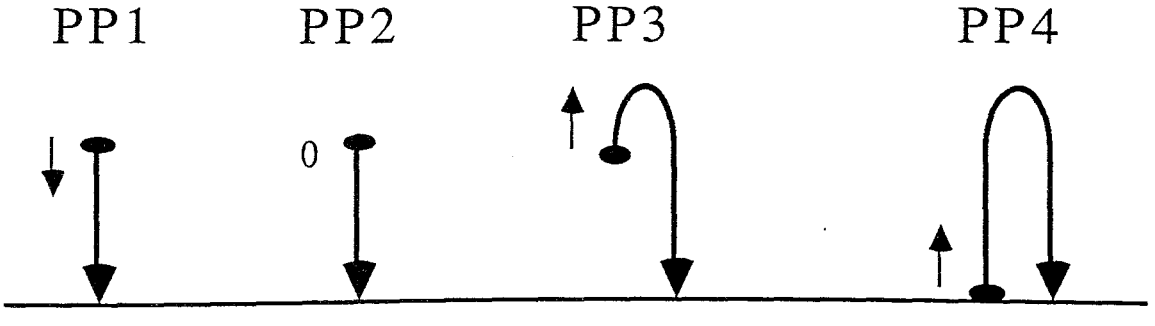


Figure 5.3.1. Possible pasts for a ball hitting the ground

TABLE 5.3.1. Output of ball postdiction

Possible Past #1

Y	time
<*NewY2*, dec>	t_{-1}
<(0, ∞), dec>	(t_{-1},t_0)
<0, dec>	t_0

Quantity space of Y: {-∞, 0, *NewY2*, ∞}

Possible Past #2

Y	time
<*NewY*, std>	t_{-1}
<(0, ∞), dec>	(t_{-1},t_0)
<0, dec>	t_0

Quantity space of Y: {-∞, 0, *NewY*, ∞}

Possible Past #3

Y	time
<*NewY3*, inc>	t_{-2}
<(0, *NewY*), inc>	(t_{-2},t_{-1})
<*NewY*, std>	t_{-1}
<(0, ∞), dec>	(t_{-1},t_0)
<0, dec>	t_0

Quantity space of Y: {-∞, 0, *NewY3*, *NewY*, ∞}

Possible Past #4

Y	time
<0, inc>	t_{-2}
<(0, *NewY*), inc>	(t_{-2},t_{-1})
<*NewY*, std>	t_{-1}
<(0, ∞), dec>	(t_{-1},t_0)
<0, dec>	t_0

Quantity space of Y: {-∞, 0, *NewY*, ∞}

Table 5.3.1 shows how the algorithm "pads" a point state with all magnitudes at newly designated landmarks to the beginning of possible pasts starting with interval states, to keep to the custom that QSIM behaviors start with point states. The values in times (t_{-1},t_0) do not contain the new landmarks

which appear in previous states; this is a result of the fact that the previous states are actually computed later by the algorithm, and can be modified easily, by the same method used to update the CV lists to reflect the new landmark information in Chapter 4.

5.3.2. Burst Tank Postdiction

As a second example, consider the U-tube system of Section 3.1.1. If the system is currently in the operating region B_BURST, (Figure 5.3.2) what could have happened in the past? In the input state, a description of the system, where the liquid in tank A is just flowing out of the pipe between tank A and the now nonexistent tank B, is given (Table 5.3.2.) The "current" value of amount_A is A_{now}, a landmark in $(0,AMAX)$. This postdiction will clearly involve a backward operating region change.

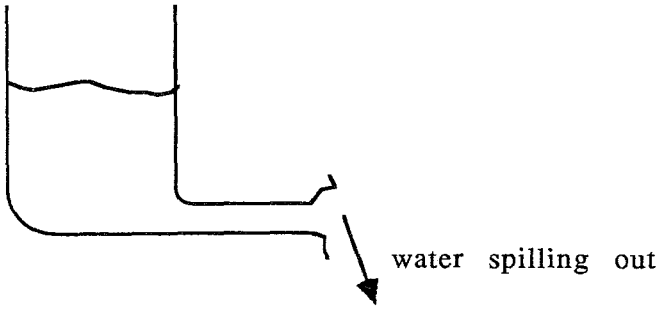


Figure 5.3.2. U-tube in operating region B_BURST

TABLE 5.3.2. Starting state of U-tube postdiction

PARAMETER	VALUE
amount_A	$<A_{now}$, dec>
amount_B	< 0, std>
flow_AB	$<(0, \infty)$, dec>
flow_BA	$<(-\infty, 0)$, inc>
pressure_A	$<(0, \infty)$, dec>
pressure_B	< 0, std>
p_diff_AB	$<(0, \infty)$, dec>

Since the program's output is too large to reproduce here in a concise manner (the tree contains many branchings caused by ambiguities about the value of the pressure difference,) verbal descriptions of the families of possible pasts found are given below:

1: The system starts in region B_BURST, with amount_A in $(A_{now}, AMAX)$, and moves to the input state.

2: The system starts in region B_BURST, with amount_A at $AMAX$, and moves to the input state.

3: The system starts in region NORMAL, with the U-tube completely full (i.e., amount_A at $AMAX$ and amount_B at $BMAX$.) Tank B explodes immediately. Liquid in tank A drains for a while, and the input state is obtained.

4: The system starts in region NORMAL, with amount_A in $(A_{now},AMAX)$, and amount_B at $BMAX$ and increasing. This is immediately followed by the explosion of tank B. Liquid in tank A drains for a while, and the input state is obtained.

5: The system starts in region NORMAL, with amount_A in $(A_{now},AMAX)$, and amount_B in $(0,BMAX)$ and increasing. After a while, tank B explodes. Liquid in tank A drains for a while, and the input state is obtained.

6: The system starts in region NORMAL, with amount_A in $(A_{now},AMAX)$, and amount_B 0 and increasing. After a while, tank B explodes. Liquid in tank A drains for a while, and the input state is obtained.

7: The system starts in region NORMAL, with amount_A at $AMAX$ and amount_B in $(0,BMAX)$ and increasing. After a while, tank B explodes. Liquid in tank A drains for a while, and the input state is obtained.

8: The system starts in region NORMAL, with amount_A at $AMAX$ and amount_B 0 and increasing. After a while, tank B explodes. Liquid in tank A drains for a while, and the input state is obtained.

9: The system starts in region NORMAL, with amount_A at its value in the input state, and amount_B at $BMAX$ and increasing. This is immediately followed by the explosion of tank B, which results in the input state.

10: The system starts in region NORMAL, with amount_A in $(A_{now},AMAX)$, and amount_B in $(0,BMAX)$ and increasing. After a while, tank B explodes, and at that moment, the input state is obtained.

11: The system starts in region NORMAL, with amount_A in $(A_{now},AMAX)$, and amount_B 0 and increasing. After a while, tank B explodes, and at that moment, the input state is obtained.

12: The system starts in region NORMAL, with amount_A at $AMAX$ and amount_B in $(0,BMAX)$ and increasing. After a while, tank B explodes, and at that moment, the input state is obtained.

13: The system starts in region NORMAL, with amount_A at $AMAX$ and amount_B 0 and increasing. After a while, tank B explodes, and at that moment, the input state is obtained.

Note that all the qualitatively distinct possibilities involving various combinations of amount_A and amount_B values are listed. This ability of exhaustive postdiction is a desirable feature, as will be discussed later.

5.4. Discussion

When QSIM is used for postdiction, some important issues of this reasoning task are resolved naturally. One does not have to worry about the possibility of overlooking some laws of change, because, (within a single operating region,) *all* the laws are already known; they are the transitions of Table 5.2.1. Kuipers has proven that the rules of Table 3.1.1 cover all possible transitions from "this" state to the "next" state, and here they are used to retrieve the *same* information, albeit in the other direction, so the proof stands. The fact that the same laws of change apply to each input problem is another advantage of this algorithm, stemming from the definition of the domain.

There is also no problem of choosing the most "sensible" law of change among many alternatives, since the level of description of the rules makes them all of equal caliber, so all possibilities are generated by the algorithm.

There is a well-defined rule about when to stop postdiction on a branch: The algorithm does not try to find the predecessors of a leaf, if no other

operating region can have a state which would cause an immediate transition to this one, and: 1) The transition rules do not lead to any different state satisfying the constraints, or 2) The end of a legal range has been reached, or 3) An "all-steady" state (i.e. one in which all the qualitative directions are **std**) in the system's past has been reached, (note that some of the *higher order* derivatives, which are not represented as parameters, must have been nonzero at that point; this is contrary to the quiescence heuristic used during forward simulation, which assumes that all higher order derivatives are also zero when the represented ones are,) or 4) A cycle in the path to the root has been detected.

Condition (4) above is interesting: What does a cycle in the state tree signify in postdiction? Consider the spring/block system of Section 3.1.3. One branch of the tree produced for that input contains an eight-state cycle (corresponding to stable oscillation) of which the input state is a member. (This is the backward-generated equivalent of Table 3.1.12.) The existence of an infinite number of distinct possible pasts is concluded from that branch, because the system may have started just one state ago, or two states ago,..., or 222 states ago,..., and so on. Once again, the fact that the initialization time of the system is not known leads one to consider all the alternatives.

All qualitative model-based reasoners make the Closed World Assumption (Section 2.2.) The CWA is the reason why paths from the state tree's root to its leaves are defined as possible futures (behaviors) in QSIM: It is known that no "external" influence (i.e., one not mentioned in the input) will affect the system and interrupt its behavior. The balls in the previous examples will not be hit by ("anti-ball") missiles, for example. So the system will "run" until the end of the branch representing its behavior is reached.

The CWA is also made in postdiction during the generation of the predecessor states: All the entities and relationships that may come about at any relevant situation are known; so all possible predecessors can be computed. But the very notion of an *initial* state defies the CWA: There has to be "someone" (whose mode of operation is unknown or unmodeled) who "initializes" the system. In the presented examples, this is the "person" who throws the ball, or pours the water into the tank, or stretches the spring. Since the models do not know about this entity, which can "set" the system to any state (which satisfies the QDE) that it wishes, the algorithm has no way of deciding whether a state in the past is the initial state or not, so it presents both these possibilities in its

output. (This also corresponds to human intuition. People, when they see a non-quiescent physical system, seem to postulate the existence of a "creator" which set the system going and then left the scene.)

The correctness considerations of the postdiction algorithm are the same as those of pure QSIM. Soundness (i.e., that *all* possible pasts are found) follows from the fact that QSIM is sound, and the above discussion. The incompleteness property (i.e., the possibility of generating spurious answers) is also inherited from QSIM, since this does not have anything to do with the direction in which the transition rules are being used.

The computational complexities of the two algorithms are again the same. The possible past which starts with the newly created state can be printed out immediately after each state creation, this takes $O(ps)$ time and does not affect the overall worst-case complexity. Case runs also show that the QSIM and postdiction algorithms have the same time requirements when run with inputs leading them to produce trees of similar size. An implemented PROLOG program embodying the postdiction algorithm is described in Appendix A.

5.5. Applications

Postdiction has an important place in the qualitative reasoning repertory. Diagnosis of malfunctioning physical systems is a natural area of application for postdiction. In diagnosis, there is a conceptual "going back" in time, from the occurrence of "something wrong" to its cause, so postdiction immediately suggests itself as a method. A fault can manifest itself either as an unexpected parameter value, or a change in the QDE. In the first case, postdiction from the current state gives an exhaustive list of all possible value combinations that the parameters may have possessed, which can be examined by a human expert. The fact that the algorithm enumerates all the qualitatively distinct pasts, some of which may escape the notice of a human, is an advantage in this task. The more "exotic" and less obvious causes of faults are not missed, thanks to this feature. In the burst U-tube problem, the possibilities in which the system starts out as a one-tank system are overlooked by a sizable proportion of

humans invited to "postdict" it, for instance, maybe because destructive events are psychologically dominant. When faults cause changes in the QDE, several constraint sets, representing various "faulty models" (like in Section 3.2.6) can be used to perform postdiction from the current state. The ones that have the system's (known) initial state in their possible pasts are candidate causes of the problem.

Since the algorithm has only very local differences from QSIM, (the syntaxes of their input sets are the same,) all the extensions to QSIM explained in the previous chapters can also be applied to the postdiction algorithm, resulting in corresponding improvements in performance.

VI. THE QUALITATIVE SYSTEM IDENTIFICATION ALGORITHM

Before performing any kind of model-based reasoning, one has to have a model of the system which will be reasoned about. The modeling methods used by current reasoners, which require possession of large amounts of information about physical laws and the various kinds of components or mechanisms that can be used to build systems, are fundamentally inadequate for general-purpose reasoning. When faced with a novel situation, or a new mechanism whose description is not available in the library, these reasoners cannot achieve modeling, even though it is in these cases that the modeling task is the most important and interesting. Leaving the preparation of the models completely to the "user," on the other hand, is clearly not a way out, from the point of view of *artificial* intelligence, which aims to automate human behavior.

When one examines what humans do in similar situations, it is seen that a "mental" model of the "laws" of the system under consideration can be formed, after a period of observation of the system's behavior, which suggests an "algorithm" whose input is the behavior of the system, and whose output is the system model. This is essentially the reverse of what simulation, qualitative or quantitative, does.

This task of data-based model construction is the subject of an already mature field, named *system identification.* Extensive research has been made and widespread applications of efficient algorithms which perform system identification in the numerical domain have been produced. In this chapter, QSI, an algorithmic method of performing *Qualitative* System Identification, using the qualitative representation, is presented. In the following, "conventional" system identification will be called CSI to distinguish it from QSI.

6.1. QSI as CSI

First of all, a potentially confusing difference in terminology will be clarified. What has been called *parameters* in this text are generally called *variables* in CSI. In CSI, the *parameters* are constants which appear in the equations (models) describing the system, and the main concern is to identify their values precisely. In the qualitative representation, a constant can be described, if necessary, as a parameter "stuck" at a landmark. For example, consider the acceleration of the balls in Chapters 4 and 5.

Generally, there are two kinds of variables in CSI: *input* and *output* variables. The input variables can be controlled by "us," and changing their values to "excite" the system properly is an important task. A CSI *experiment* consists of this excitation and the recording of the variable values for some time. Almost always, the measurements are real-valued and are made at (usually equidistant) discrete time points. As a complicating factor, *noise*, which may corrupt these values, is usually present, and has to be taken into account. Once the data are collected, the first thing to do is to determine the *form* of the equation that is being searched. This *model structure determination* problem is still an important issue of CSI, [48] which involves the following questions: What should the equation "look like," i.e., how should it "link" the variables together so that it is an acceptable description of the physical system? What should be its basic parameterization?

Once a model structure has been decided, the parameters in that equation are estimated, using statistics-based algorithms. The aim is to find the parameter values which, when "inserted" to their places in the model, will predict the variable values seen in the experiment.

The model which emerges as a result of this procedure is then tested, and accepted only if it seems to describe the system at hand appropriately. Otherwise, one has to go back to the parameter estimation, structure

determination, or even the experiment stages, to try it with new decisions all over again.

The most extensively researched and accomplished part of CSI is the parameter estimation step. Elaborate numerical algorithms for this task have been developed.

There has been some work [49] on performing CSI with fuzzy values and models, aimed at handling cases where the available information is incomplete.

QSI's input is a set of QSIM behaviors of the system to be identified, and its output is a QSIM-style QDE describing the system. Apart from its ability to handle incomplete information, the adoption of the QSIM representation also has the advantage that QSI fits naturally to the "modeling" gap, discussed above, in the qualitative reasoning repertory.

QSI does not cover the experiment design and execution stages of CSI: It starts with ready (qualitative) data about the behaviors. It treats input and output[3] parameters in the same manner, (actually, it has no distinction of them,) note that, in the QSIM representation, all parameters are "equal" in this sense. Various issues that arise about QSI's input will be discussed later.

The QSI algorithm may be viewed as a way of finding better and better model structures, as will be explained shortly. QSI has the ability of postulating deep variables of the system, which are not visible in its input. The model testing stage is also a part of QSI, but the "testing" here has a different meaning than that of CSI: QSI tests its models to see whether they are "deep" enough; it does not need to test whether they really describe the input behaviors, because the models are created in such a way that they are provably correct, see Section 7.2.

Although the qualitative representation itself is resilient to noise, qualitative noise filters, based on simple observations about the nature of noise, have also been designed for incorporation to QSI.

QSI's relation to CSI is similar to those of other qualitative reasoning methods to their quantitative counterparts: The qualitative methods suppress

---

[3] In some applications, such as signal processing, CSI may be performed without any input (i.e. directly controllable) variables, as well.

irrelevant (or unknown) information, keep the qualitatively important distinctions, and arrive at useful results through much simpler computation.

The actual algorithm that QSI uses to generate the models is fundamentally different than anything that CSI uses. This underlines the traditional difference of AI programs, which make symbolic computation, from "non-AI" programs, which perform numeric computation. QSI performs a search in the *space* of models; since the building blocks of the equation that describes the system are already known and are finite, (the "operands" are the parameters and the "operators" are the constraints,) a well-defined method of trying out all the combinations until the correct one is found can be developed.

6.2. The QSI Algorithm

This section includes a comprehensive presentation of the basic QSI algorithm; the requirements on the input, the syntaxes of input and output, examples, and detailed discussions of the algorithm's individual stages.

6.2.1. Input and Output

The input to QSI consists of one or more behaviors of the system to be identified, and the quantity spaces of the parameters seen in these behaviors. As mentioned before, it may be the case that only some of the parameters that would appear in a deep model of the system are easily observable, and therefore "at first sight," one may think that the system consists only of these parameters. For this reason, QSI allows the possibility that its input does not contain all the system parameters, and tries to find the deeper parameters by itself. On the other hand, the input should contain as many qualitatively distinct system behaviors as possible, if QSI is expected to find an appropriately deep model.

Since a QSIM model that produces it will be looked for, the input should be generable by QSIM, i.e., there should be a QSIM input set (unknown, of

course, at this stage,) that would cause QSIM to produce it as output. This means that the input behaviors cannot be just any sequence of qualitative states:

Definition 6.2.1. A behavior is *T-legal* if all the parameters in it obey the transition rules of Table 3.1.1 throughout the behavior.

The QSIM transition rules embody all kinds of change that a continuous-valued quantity can undergo. Barring operating region changes, all quantities that are dealt with in this domain obey these rules. All "real" systems behave like this, (at least, at the commonsense scale in which one is viewing them.) All QSIM outputs which do not contain operating region changes are, by construction, T-legal.

QSI requires that its input behaviors are T-legal, so in a single run, it should only be "shown" a single operating region of a system. In consecutive runs, by feeding QSI by the system behaviors at different operating regions, the QDEs of all the operating regions can be obtained.

Apart from operating region changes, another source of *T-illegal* behaviors is the following: Suppose one is monitoring a system, as in Section 3.2.6. Because of one's measurement intervals, one may "jump" over some states (especially time-point states) that appear in the actual qualitative behavior of the parameter being measured. This may lead to discontinuous changes in the "behavior" constructed as a result of the measurement.

Actually, the constraint determination stage (Section 6.2.4) of QSI works equally well for T-illegal and T-legal behaviors, i.e., it finds all constraints valid on the parameters in the input behaviors, but the nature of the model depth test and extension stages requires the T-legality assumption, as will be seen.

To represent some properties of behaviors that QSIM is able to indicate in its output, the QSI input marker symbols, EQU and CYC, are defined. These markers may appear after each input behavior. Their meanings are as follows:

-EQU requires that in the last state of the behavior it precedes, all qualitative directions are `std`, and means that the system is quiescent from that time on. (This conclusion is heuristic, of course, see Section 3.1.2.)

-CYC requires that the last state of the behavior it precedes has appeared before in that behavior, and means that the rest of the behavior is cyclic.

The landmarks discovered during simulation can be distinguished from the other ones in QSIM's output, and QSI also requires that such landmarks be specified in the input quantity spaces, by preceding their names by the string "*disclm*" (standing for "discovered landmark.")

Note that none of these requirements about the input violates the "spirit" of system identification and let the algorithm know more than it is "allowed" to: Equilibrium and cyclic behavior are generally easily observable things, and a simple method of understanding which landmarks are discovered during the observed behavior is to designate all nonzero values at which the parameter becomes **std** for some time as that parameter's discovered landmarks in that behavior.

Actually, QSI starts execution with much *less* information that it is "entitled" to: It has no idea at all about *what* the parameters are; unit (or even, dimension) information on the parameters, which goes without saying in CSI, is nonexistent, and even the most natural invariant knowledge (like "amounts are never negative") cannot be used. The fact that QSI is still able to find the models, as will be demonstrated, shows the algorithm's potential strength.

QSI's input may also contain an integer representing the maximum number of allowed iterations for the algorithm. When this item is absent, the number is assumed to be infinite. The allowable number of excess behaviors in depth testing is also an input item. (See Section 6.2.6 for explanations.)

Finally, if he wishes, the user may include postulation and search mode selectors in the input; these specify certain restrictions on the model search that will be performed, and can be utilized for efficiency reasons, especially when additional information ("hints") about the sought model is available, as will be explained.

QSI's output consists of one or more constraint sets, which are models of the system exhibiting the input behaviors. Each QDE in this sequence is deeper (i.e. has more constraints and invisible parameters) than its predecessors, with the last one being an appropriate description of the system.

6.2.2. The Algorithm

To avoid conceptual cluttering, the preprocessors, which are used for converting the possibly numerical input to qualitative form and qualitative noise filtering, will not be explained until a later section. This section will be devoted to the "core" of QSI, the basic algorithm which constructs system models from their behaviors.

The algorithm (Figure 6.2.1) starts with a stage of constraint determination on the input behaviors. The QDE obtained as a result of this stage is tested to see whether it is appropriately deep or not. If it passes the test, the model has been found. Otherwise, the model (and, therefore, the behaviors,) are extended to contain new parameters, and constraint determination is made on this set, followed by a new test. This loop is exited when a "good" model is found. The model is enhanced by making use of dimension information inherent in the arithmetic constraints, and the algorithm terminates. Here is the algorithm in a pseudo-high-level language:

```
      BS := set of system behaviors from input
      perform Constraint Determination on BS, resulting in system QDE
loop: print the QDE
      if the QDE passes the Depth Test
         then
           blockbegin
             impose Dimension Consistency on the QDE, resulting in final model
             print final model
             terminate
           blockend
      { Depth Test not passed }
      postulate new parameters; EBS := BS ∪ [ the new parameter behaviors ]
      perform Constraint Determination on EBS, resulting in the extended
        system QDE
      BS := set of system behaviors involving parameters that appear in the QDE
      go to loop
```

The constraint determination stage finds *all* the constraints valid in the behaviors given to it, using a simple method. It considers all possible constraints on the given set of parameters, and controls each of them to see whether it holds throughout the sequence of input states. However, not every constraint found in this manner is included in the resulting QDE; only the

"useful" constraints that are *not* algebraic consequences of already existing ones are added to the model.

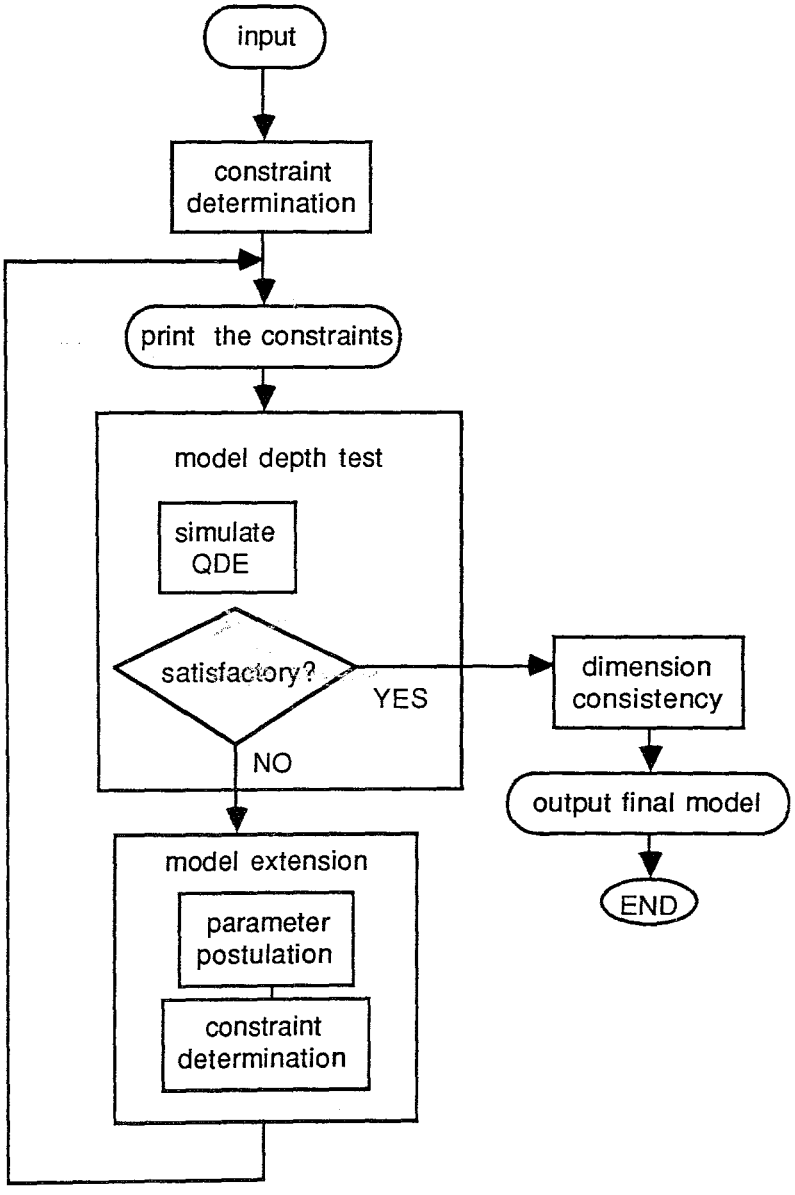


Figure 6.2.1. The QSI algorithm

The model depth test stage uses a slightly modified version of the QSIM algorithm to make its decision. The QDE produced by the previous stage is simulated by QSIM for each distinct initial state appearing in QSI's input. The output of QSIM is then examined. Since the constraint determination stage performs correct system identification on its input, QSIM's output in this stage is bound to contain all of QSI's input behaviors. (This is proven in Section 7.2.) What is really checked in this stage of the algorithm is the number of QSIM

behaviors that do *not* appear in the QSI input. If these are above an "acceptable" level, (see discussion in Section 6.2.6) the QDE is deemed "loose," and model extension is performed. Otherwise, the QDE is accepted and the algorithm terminates after the dimension consistency stage.

The model extension stage involves adding new variables into the equation of the system. These new parameters are obtained from the old ones; they are the derivatives, sums, squares, etc. of the old parameters. If interesting relationships which may tighten QSIM simulation in this extended set of parameters are found by constraint determination, the involved parameters are permanently added to the model; i.e. they are "discovered" by QSI.

The dimension consistency stage converts the obtained model to a "real" one where the discovered relationships among the quantities still hold, but the simple dimension rules imposed by the constraints on their parameters (such as the ADD and MINUS constraints' requirement that their parameters have the same units) have been satisfied by the postulation of possible "buffer" parameters and M constraints.

After an example which illustrates these concepts, each stage will be discussed in detail.

6.2.3. An Example

As an example to the operation of QSI, the U-tube (in operating region NORMAL) of Section 3.1 will be considered again. Since the QDE of this system has already been seen, one has an idea of what the underlying model is. Of course, QSI has no such information when it starts. Suppose that only the amount parameters appear in the input. (It is very likely that only these two would be recognized as parameters of this system after a "shallow" observation.) Two behaviors of this system are input: One of them starts with amount_A decreasing and amount_B zero and increasing; the other describes the opposite case. (To keep the example as simple as possible, the maximum capacity limits of the tanks are not considered at all. The algorithm would work equally correctly in the case where they are included, and the following discussion would still apply. The number of input behaviors would rise in that case, to cover the various ordinal relations that the amounts could have with

their maximum landmarks at the end of the behavior in this operating region.) So the input behaviors are as in Tables 6.2.1 and 6.2.2.

TABLE 6.2.1. U-tube identification, input behavior #1

amount_A	amount_B	time
<(0, ∞), dec>	<0, inc>	t_0
<(0, ∞), dec>	<(0, ∞), inc>	(t_0, t_1)
<*disclmA*, std>	<*disclmB*, std>	t_1
	EQU	

TABLE 6.2.2. U-tube identification, input behavior #2

amount_A	amount_B	time
<0, inc>	<(0, ∞), dec>	t_0
<(0, ∞), inc>	<(0, ∞), dec>	(t_0, t_1)
<*disclmA*, std>	<*disclmB*, std>	t_1
	EQU	

The constraint determination stage tries out all constraints syntactically possible on amount_A and amount_B. For example, DERIV(amount_A, amount_B) is tried, but it fails in the very first state in the input, so it is discarded. The only constraint that is satisfied throughout the input is M-(amount_A, amount_B), so it forms the initial QDE on its own.

Note that no such constraint appears in the U-tube model of Section 3.1.1. However, simple reflection about the system confirms that the amounts in the tanks are indeed inversely proportional in the operating region NORMAL. The human who wrote the QDE of Section 3.1.1. chose not to include the M-. (That model still adequately describes the system.) On the other hand, QSI, which is designed not to miss any significant constraints on the known parameters, has found it. (The "human" aspects of modeling vs. QSI will be discussed further in Section 8.1.)

This single-constraint model is simulated in the depth-test stage from both initial states in the input. As expected, the model cannot pass the test; it is too shallow. The single constraint cannot represent the inner mechanism which causes the system to arrive at equilibrium. Among the behaviors generated by QSIM at this stage are those where one amount starts increasing

from zero, while the other one arrives at, and even goes below, zero. So a model extension is necessary.

The model extension stage begins with the computation of the behaviors of the newly postulated parameters. (The extent of postulation can be modified. For certain problems, more efficient solutions with less postulation are possible; see Chapter 9 for a complete list of new behaviors for this example in full postulation mode.) Since the new parameters are linked by constraints to the old ones, whose values are already known, their values at each state can be calculated. Possible ambiguities are resolved using certain heuristics. (See Section 6.2.5.) For example, consider two new parameters, say, PX and PY, which are defined to be the time derivative of amount_A, and the sum of the two amounts, respectively. The defining constraints of these parameters are therefore

DERIV(amount_A, PX) and ADD(amount_A, amount_B, PY).

By the use of the heuristics, which basically say that "things change as infrequently as possible," the new parameter behaviors are calculated, and the system behaviors are augmented to include them, as shown in Table 6.2.3. Already, another important relationship has been discovered: The sum of the amounts is fixed, i.e. mass is conserved.

TABLE 6.2.3. U-tube identification, behaviors of two of the postulated parameters

System Behavior #1

amount_A	amount_B	...	PX	...	PY	...	time
<(0, ∞), dec>	<0, inc>	:	<(-∞,0),inc>	:	<*nlm*, std>	:	t_0
<(0, ∞), dec>	<(0, ∞), inc>	:	<(-∞,0),inc>	:	<*nlm*, std>	:	(t_0, t_1)
<*disclmA*, std>	<*disclmB*, std>	:	< 0, std>	:	<*nlm*, std>	:	t_1
			EQU				

System Behavior #2

amount_A	amount_B	...	PX	...	PY	...	time
<0, inc>	<(0, ∞), dec>	:	<(0, ∞), dec>	:	<*nlm*, std>	:	t_0
<(0, ∞), inc>	<(0, ∞), dec>	:	<(0, ∞), dec>	:	<*nlm*, std>	:	(t_0, t_1)
<*disclmA*, std>	<*disclmB*, std>	:	< 0, std>	:	<*nlm*, std>	:	t_1
			EQU				

Quantity space of PY: {-∞, 0, *nlm*, ∞}

Constraint determination on these larger behaviors is more involved. Parameters which appear in two or more constraints which do not algebraically imply each other are added to the model, and the constraints involving them are made part of the QDE. A lot of constraints which are implied by others are not even checked, which is good for efficiency. The QDE found after one iteration of model extension in derivative postulation/half search mode and fed to the depth test module is presented in Table 6.2.4. The simulation of this model from the initial states predicts only the input behaviors, so an acceptable model has been obtained.

TABLE 6.2.4. Constraints found in the U-tube identification

CONSTRAINT
M-(amount_A,amount_B)
DERIV(amount_A,P1)
DERIV(amount_B,P2)
ADD(amount_A,P1,amount_B)
ADD(amount_B,P2,amount_A)

The ADD constraints in this model involve the addition of a quantity with its time derivative, which is arithmetically not legal. To legalize the situation, while keeping the valuable ADD relation, three buffer parameters for the arguments of each ADD are postulated. The buffer parameters are linked by M+ constraints to the ADD arguments, and each has the same quantity space structure as the corresponding ADD argument. The resulting model of the U-tube in operating region NORMAL is the one shown in Table 6.2.5, which is, although slightly different than the model of Section 3.1.1, a correct and deep description of the system. The newly postulated parameters are seen to correspond to the following actual quantities:

P1: Flow into tank A
P2: Flow into tank B
P3, P8: Pressure at the bottom of tank A
P5, P6: Pressure at the bottom of tank B
P4: The pressure difference between tank B and tank A
P7: The pressure difference between tank A and tank B

The method's power of hinting at meaningful deep parameters is thus demonstrated. (See Section 8.1 for deep parameters which are not so meaningful, and an interpretation for them.)

TABLE 6.2.5. Final U-tube model after identification

CONSTRAINT	CVs
M-(amount_A,amount_B)	
DERIV(amount_A,P1)	
DERIV(amount_B,P2)	
M+(amount_A,P3)	(0, 0), (∞, ∞), (-∞, -∞)
M+(P1,P4)	(0, 0), (∞, ∞), (-∞, -∞)
M+(amount_B,P5)	(0, 0), (∞, ∞), (-∞, -∞)
ADD(P3,P4,P5)	
M+(amount_B,P6)	(0, 0), (∞, ∞), (-∞, -∞)
M+(P2,P7)	(0, 0), (∞, ∞), (-∞, -∞)
M+(amount_A,P8)	(0, 0), (∞, ∞), (-∞, -∞)
ADD(P6,P7,P8)	

Various more detailed features and some problems will be discussed in further examples in the text. An in-depth discussion of each of the individual stages follows now.

6.2.4. Constraint Determination

The constraint determination process is summarized in the pseudo-high-level language algorithm below. Remember that the input is a set of system behaviors, and the output is a set of constraints. In this sense, this stage is the part of QSI where system identification itself is performed, the others deal with improving the model in some way or another.

```
for each constraint type CT do
 for each tuple ARG of parameters that can be arguments to CT do
  if existing constraints do not overrule CT(ARG)
  then
   if CT(ARG) is a consequence of existing constraints
   then
    write(CT(ARG))
   else
    blockbegin
     for each qualitative state in the input do
      if CT(ARG) does not hold
      then
       break out and go to blockend
     {At this point, CT(ARG) is a novel constraint valid throughout the input}
     write(CT(ARG))
     add CT(ARG) to the QDE of the system
    blockend
```

In the algorithm above, an accumulation and control of possible CVs of the CT is also part of the check about whether it "holds" or not. When CT(ARG) is added to the QDE, any discovered CVs go with it.

How many different qualitative constraints can be written on p parameters? There are six constraint types, and all of them have to be considered for every combination of parameters.

M+ has to be checked on all pairs of parameters. However, since M+ is commutative, M+(Y,X) need not be checked if M+(X,Y) has already been checked. The same applies for M- and MINUS. (MINUS is a special case of M- anyway.) For each of these types, the number of constraints that will be checked is thus

$$\binom{p}{2} = \frac{p.(p-1)}{2} = \frac{p^2 - p}{2} \qquad (6.1)$$

DERIV is not commutative, so twice as many of those has to be considered as any one of the above-discussed ones, that is:

$$2.\binom{p}{2} = p^2 - p \qquad (6.2)$$

DERIVs will be checked.

ADD and MULT are commutative, so that their first two arguments can be interchanged. (Not all "additive" or "multiplicative" relationships among the parameters are noticed at this stage: Note that any addition or multiplication of more than two operands can be expressed as a *set* of three-argument ADD or MULT constraints as defined in Chapter 3. If one's initial set of parameters is {A,B,C,D} and the relationship A + B + C = D holds among these, the first constraint determination does not add the constraints representing this equation to the QDE, since an additional parameter is required to write them in the QSIM format: ADD(A,B,P), ADD(P,C,D). Such "cascades" of constraints are discovered later in the model extension stage; see Section 6.2.5.) The formula for the number of controls of ADD and MULT constraints on three different parameters is thus

$$3.\binom{p}{3} = 3.\frac{p.(p-1).(p-2)}{6} = \frac{p^3 - 3p^2 + 2p}{2} \qquad (6.3)$$

since this is a matter of choosing three parameters, and deciding which of this will be third argument.

Have all the possibilities been exhausted? There is still one more meaningful relationship which can exist between parameters, and which is expressible in the present vocabulary. The parameter X can be the *square* of parameter Y, that is, MULT(Y,Y,X) may be valid. Since this is a noncommutative binary relationship like DERIV, the number of MULTs that will be tried in this manner is again $p^2 - p$. (The reader may note that there is also a "twice" relationship which can be expressed as ADD(Y,Y,X). Since this is qualitatively equivalent to M+(Y,X), and also not very common in practice, this combination is not checked.)

The total number of possible constraints on p parameters is therefore the sum of the above, i.e.

$$\frac{2p^3 + p^2 - 3p}{2} \qquad (6.4)$$

for $p \geq 3$, and only 7 for $p = 2$.

But the actual number of constraints that get checked against the input states is usually much less than that, since the semantics of the constraints can be used to decide on most of them without checking any values. Consider the following scenario: Constraint generation and testing has been going on for some time. The constraint M+(A,B) has been found to be valid. Now, the constraint MINUS(A,B) is considered. The algorithm can decide to skip this possibility immediately, since the MINUS has no hope of being satisfied, given the M+.

The M constraints' defining properties can be used extensively to detect constraints which are logical *consequences* of already discovered constraints, as well. For example, if M+(A,B) and M+(B,C) are already known, there is no need to check M+(A,C) against the input behaviors; it *is* valid. There are interestingly many rules like this one; the ones QSI uses, together with their proofs, are listed in Chapter 7, which is about such "technical" issues.

Consequence constraints like the one mentioned above, are written out, but not included in the QDE that is fed to QSIM for the model depth test. The reason for this is twofold: First, consequence constraints do not change

anything in the QSIM output if their antecedents are already in the input, (This is a result of their being consequences,) second, QSIM's time requirements are linear in the number of constraints, so their inclusion slows down execution considerably, without contributing anything.

The consequence detection check, which has just been explained, speeds up the algorithm especially in later constraint determinations, made after model extension, when p is relatively great, and the number of values to be checked can be big. (See Sections 6.2.5 and 7.1.)

6.2.5. Model Extension

After constraint determination has been performed on a particular set of behaviors, no new constraints, other than those already found, can be written on the set of parameters appearing in these behaviors, since constraint determination is exhaustive. So if the model at hand is found to be too loose by the depth test stage (Section 6.2.6) and has to be extended by the addition of new constraints, one has to introduce new system parameters to the model, so that constraints involving them can be searched for. Since the set of behaviors is all that QSI knows about the system, it is used in the *postulation* of the new parameters. Each newly postulated parameter is a "neighbor" of an existing parameter. Two parameters are *neighbors* if they appear in the same constraint. Parameter postulation is then seen to be composed of two steps:

1) Postulation of a new constraint which links one or two "old" (i.e. known) parameters to a new one; this will be called the *defining constraint* of the new parameter,

2) Calculation of the behavior of this parameter from its defining constraint and the values of its neighbors.

Both of these steps give rise to important issues, which will now be described.

To make QSI search as wide an area of the "space" of models mentioned before as possible, virtually all neighbors of the known parameters have to be postulated. If some neighbors are left out, and the "real" equation describing the system contains them, one is faced with the possibility of failing to find a

good model. On the other hand, parameter postulation is an expensive process (Section 7.1) and a number of QSI problems, where the solution can be obtained efficiently with the postulation of only some neighbors, exist. (Examples to this are presented in Chapter 9.) Because of this, QSI has been made flexible about the extent of postulation, and can be run in any one of a number of "postulation modes." The following analysis is about the *full* postulation mode, where, in response to the lack of any "hints" about the actual model,virtually all the neighbors are created, i.e. the worst case.

Full postulation mode involves the generation of the following neighbors:

-The derivative of every non-constant parameter
-The sum and differences of every pair of parameters
-The product and, if possible, ratios of every pair of parameters
-The negative of every parameter
-The square of every parameter

As can be seen, all types of constraints, except the Ms, are utilized as defining constraints. The reason for the fact that not all syntactically possible neighbors are created will be clear when the second step of parameter postulation, that is, the behavior calculation procedure, is discussed.

QSI decides that a parameter is constant when all its values are seen, or can be assumed, to have the direction **std**, and all its magnitudes are the same landmark. The discovery of such constants is desirable, since they greatly limit the proliferation of QSIM outputs, and are conceptually helpful in modeling, as will be further discussed. Derivatives of constants need, of course, not be postulated, since values fixed at zero can be eliminated from equations. (A "lonely" zero on one side of an equation can always be handled by using ADD and/or MINUS constraints.)

Neighbors whose defining constraints are already *in* the QDE, (found by previous constraint determinations,) will also not be postulated.

How many neighbors of p parameters are there? Assuming that none of the reducing conditions above apply, one has

* p derivatives,
* p negatives,
* p squares,
* $\binom{p}{2}$ sums,

* $2.\binom{p}{2}$ differences,
* $\binom{p}{2}$ products, and
* $2.\binom{p}{2}$ ratios.

Therefore, the worst-case number for the full postulation mode is the sum of these, that is, $3p^2$ new parameters will be created, provided all old parameters are nonzero throughout the input (which is unlikely.) If a parameter X does have the magnitude zero even once, no ratios of the form $\frac{Y}{X}$ (where Y is another old parameter) are postulated, so the above number is usually not reached. In cases where limited postulation is acceptable, for example, in the "derivative postulation mode," where only the derivatives of the existing parameters are created, the number of neighbors, and the time required, are of course accordingly less.

To perform extended constraint determination on the new set of parameters, QSI must assign behaviors to each of the newly postulated parameters. The values of the old parameters at each state and the defining constraint are known, value sequences of the new parameter which satisfy both the constraint and the transition rules can be found. The problem is that, in most cases, there is an *infinite* number of legal behaviors that may be assigned to the new parameter.

To see this, one behavior of the parameter amount_A from Section 6.2.3 will be examined more closely. (Table 6.2.6.) If the derivative of amount_A, that is, PX, where DERIV(amount_A,PX), is being postulated, knowledge of the constraint alone yields the information in Table 6.2.7 about PX's behavior in $[t_0, t_1]$. It is also known that PX will have magnitude zero after t_1.

TABLE 6.2.6. Behavior of amount_A

amount_A	time
<(0, ∞), dec>	t_0
<(0, ∞), dec>	(t_0, t_1)
<*disclmA*, std>	t_1

EQU

Knowledge of the transition rules lets one conclude that PX's direction should be **inc** just before t_1, and it should be **std** at t_1 and after it.

TABLE 6.2.7. Behavior of PX in $[t_0, t_1]$

PX	time
<A negative landmark or interval, ?>	t_0
<A negative landmark or interval, ?>	(t_0, t_1)
<0, ?>	t_1

But this still leaves an infinite number of possibilities for the behavior of PX, the ones depicted in Tables 6.2.8 thru 6.2.10 being among them. There is nothing wrong about the "length" of the behavior in Table 6.2.10; remember that the number of states in a behavior is just a measure of the changes that occur, so by adding new parameters to a system, one always faces the possibility that the description of its behavior may get longer to reflect the changes in the new parameters. If PX indeed has that behavior, the behavior of the system with this parameter included will be as shown in Table 6.2.11, with the period designated $[t_0, t_1]$ in the "PXless" form of the behavior now being described by five states from t_0 to t_2.

TABLE 6.2.8. Possible behavior for PX

PX	time
<(-∞, 0), inc>	t_0
<(-∞, 0), inc>	(t_0, t_1)
<0, std>	t_1
EQU	

TABLE 6.2.9. Another possible behavior for PX

PX	time
<*lm1*, std>	t_0
<(*lm1*, 0), inc>	(t_0, t_1)
<0, std>	t_1
EQU	

Quantity space of PX: {-∞, *lm1*, 0, ∞}

All three behaviors of PX shown in these tables, and, actually, all the infinitely many qualitatively distinct behaviors where PX "wanders" in various ways in negative magnitudes before settling at zero, are physically

possible for the input of Table 6.2.1. The defining constraint's restrictions are simply unable to help one decide at one of them. ADD and MULT constraints are often faced with the same situation.

TABLE 6.2.10. Yet another possible behavior for PX

PX	time
<(*lm1*, *0*), dec>	t_0
<(*lm1*, *0*), dec>	(t_0, t_1)
<*lm1*, std>	t_1
<(*lm1*, *0*), inc>	(t_1, t_2)
<*0*, std>	t_2

EQU

Quantity space of PX: $\{-\infty, lm1, 0, \infty\}$

TABLE 6.2.11. Possible system behavior for input of Table 6.2.1

amount A	amount B	PX	time
<(*0*, ∞), dec>	<*0*, inc>	<(*lm1*, *0*), dec>	t_0
<(*0*, ∞), dec>	<(*0*, ∞), inc>	<(*lm1*, *0*), dec>	(t_0, t_1)
<(*0*, ∞), dec>	<(*0*, ∞), inc>	<*lm1*, std>	t_1
<(*0*, ∞), dec>	<(*0*, ∞), inc>	<(*lm1*, *0*), inc>	(t_1, t_2)
<*disclmA*, std>	<*disclmB*, std>	<*0*, std>	t_2

EQU

One might be tempted to design the algorithm to explore each qualitatively distinct possibility, as qualitative reasoners often do, but the previous discussion showing that there can be an infinite number of possibilities overrules that approach. One has to use rules of heuristic nature to assign the most "reasonable" of its possible behaviors to each new parameter.

The heuristics that have been adopted after thorough experimentation are:

"Prefer behaviors in which the qualitative direction changes the fewest times."

and

"If the parameter can be constant (i.e. **std** throughout at the same landmark) prefer that behavior and designate the parameter as constant."

These rules have many desirable features. They are easy to implement. They correspond to commonsense and scientific intuition in more than one way. When there are many alternative explanations for a given event, the most reasonable thing to do is to choose the simplest. It is simpler to assume that something ("thing" meaning derivatives as well as values) is not changing, when one does not know whether it is changing or not[4].

The more times the direction of a parameter changes, the stronger is the suggestion that the derivative of that parameter is driven by an even deeper mechanism, leading to a presumably unnecessarily complicated model. Especially constant parameters are important in QSIM models, and contribute to the production of smaller trees. (See the examples in Chapters 4 and 5.) They also usually correspond to important "natural" quantities. The impressive number of examples in which they actually work is another important factor in the justification of the heuristics.

QSI does the following when postulating a new parameter: It determines the shortest length that the new parameter's behavior can have, (short behaviors can contain fewer changes than longer ones, by definition) and produces *all* behaviors of that length that the parameter can exhibit, obeying the defining constraint and T-legality. The heuristics are then employed to choose one of these behaviors and assign it to the parameter. If two or more behaviors are indicated to be equally preferable by the heuristics, one is picked randomly. Since *system* behaviors are the input of constraint determination, the input behaviors are lengthened if necessary (i.e. if the new parameters require more values in their behaviors) as well as being "widened" by the behaviors of the new parameters. For more details of the behavior calculation process, see the discussion in Chapter 9.

In the U-tube example, application of the heuristics results in the behavior of Table 6.2.8 being selected for the derivative of amount_A, the enlarged system behaviors in this case would indeed look like those in Table 6.2.3.

It now becomes clear why it was decided not to postulate, for example, the parameter IX, with the defining constraint DERIV(IX,X) from any old parameter

---

[4] Note the parallels to Newton's first law and de Kleer and Brown's canonicality heuristics (Section 2.2.2.)

X, or the square roots of existing (necessarily nonnegative) parameters. In the former case, absolutely nothing about the new parameter's magnitude is known, while in the latter case, there are generally two alternative possibilities for the new parameter's values, and no hint about which one to choose. The procedure described above can be applied to find behaviors for such neighbors just as easily, but it will always come to a random selection, with no particularly good reason that the behavior selected is the most sensible one.

The "bigger" behaviors obtained as a result of parameter postulation have a tentative nature. Not all of the neighbors of the input parameters have to be important parts of the model, therefore their permanent addition into the system description is deferred until they are seen to be "significant" in some manner. QSI's criterion for significance of a model component is the following: Its addition to a QSIM input should contribute to the elimination of some additional behaviors. This is reasonable, since the whole aim of the model extension stage is to eliminate as many behaviors that do not appear in the input of QSI as possible from the output of QSIM. Immediately after behavior calculation, only the defining constraints of new parameters which are seen to be constant throughout their behaviors are permanently added to the system QDE, with invariant information indicating their fixedness being included in the QSIM input set. Fixed parameters are significant by the above criterion, since they eliminate two transitions at each point state. (See Table 3.1.1.)

The constraint determination procedure is called to find the significant constraints on the new and wider set of system behaviors. For efficiency reasons, the number of tuples that get considered in this process can be modified by specifying one of various "search modes," similar to the already mentioned postulation modes, in the input. *Full search mode* tries all combinations, just as initial constraint determination does when acting on the original input. *Half search mode* requires at least one old parameter to appear in each considered tuple.

As mentioned above, constraint determination at this point also involves an *insignificance* check, which is similar to the consequence detection check. Not every constraint that holds on the behaviors is included in the QDE. Insignificant constraints are the ones that can be proven to hold without being tested on all the states, using present information. Defining constraints by themselves have this property; the computer "knows" that they hold, because it

postulated them, and then calculated the new parameter values so that they hold. Constraints of the form ADD(X,Y,Z), where Z is any parameter, and MINUS(X,Y) is asserted or can be derived, are also insignificant. Since the qualitative addition of values of opposing sign is ambiguous, such constraints can be satisfied for lots of Zs, without reflecting any actual relationship; their generation is just a by-product of QSI's policy of testing every combination.

Significant constraints found by this stage will generally contribute to the elimination of QSIM behaviors, since the old parameters in them now have to satisfy more constraints. This usually implies a smaller number of transitions, and therefore, less behaviors. (See proof in Section 7.2.)

When a significant constraint is found, it, and the defining constraint(s) of the new parameter(s) appearing in it are added to the QDE permanently, and the new parameters' behaviors are permanently "pasted" to the system's input behaviors. Postulated parameters and their behaviors which are still out of the permanent model at the end of constraint determination are dropped, and the extended model, (now with a greater number of "old" parameters,) is again fed to the model depth test stage, to see whether it will produce only the input behaviors or not.

If no significant additions can be made to the model by this stage, the derivatives of all parameters are appended all the same, in the hope of finding a better QDE in a later iteration.

6.2.6. Model Depth Testing

The model extension procedure is a well-defined way of deepening models by adding (presumably) directly unobservable but important entities and relationships. Whether the new version of the model is satisfactory for simulation modeling purposes or not is determined by the model depth test stage. It must be emphasized that the purpose is to obtain a model which produces all, and only, the input behaviors. That the QSI-produced models yield all the input behaviors is guaranteed. (See proof in Section 7.2.) To make them produce as few of other behaviors as possible, the obvious thing to do is to add more constraints to them. To check the intermediate models to see whether they meet the requirements, the obvious approach is to simulate them using QSIM.

The model depth test stage starts by preparing the QSIM inputs necessary for the simulation. The QDE is already formed by the previous constraint determinations. Recall that QSI assumes that its input originates in a single operating region. Initial quantity spaces are prepared by stripping the discovered landmarks from QSI's input, and the quantity spaces of postulated parameters (if any.) Invariant information, specifying which parameters are constant, is discovered earlier, as discussed, and incorporated here. Each different initial state that appears in the input of QSI is entered into QSIM's input separately; QSIM will run from each of them.

Pure QSIM creates (at least, tries to create) the complete state tree for each initial state. In this application, one only wants to see that the model predicts the given behaviors correctly, so one only needs simulate for the length of these behaviors. Levels of the tree corresponding to events occurring after the end of the input behaviors are not created. (This *level limiting* feature was first mentioned in Section 4.1.1.)

Since QSIM can predict spurious behaviors, and one has no way of knowing whether spurious predictions will appear (or even, have appeared) in a particular simulation or not, the ideal aim of finding a model which will generate only the input behaviors is not generally reachable. So the model depth test stage must not strictly require that the number of QSIM outputs and QSI inputs be equal; an "acceptable" number of "excess" output behaviors have to be allowed. In view of the fact that this number changes widely from problem to problem, it has been decided to let the user specify it in the input. If no allowable excess number is specified, it is set to zero.

A shortcut is possible during the simulation. If the number of predicted behaviors exceed the allowed limit before the generation of the state tree arrives the specified level, simulation is cut off, and the current model is deemed unsatisfactory. This method is very easy to implement in the very first model testing, just after the initial constraint determination. One can always find the minimum number of behaviors implied by an incomplete state tree by simply counting its present leafs. In further iterations things are complicated by the fact that postulated parameters may cause a proliferation of system behaviors. In the following example, assume that X and Y are input (old) parameters, and Z is a new parameter with defining constraint ADD(X,Z,Y). (Obviously, these would normally be part of a bigger, meaningful system.

Attention is focused on this part of it, for the sake of the discussion.) Suppose that X and Y's input behavior is as in Table 6.2.12.

TABLE 6.2.12. Input behavior of X-Y system

X	Y
<(*0*, ∞), inc>	<(*0*, ∞), dec>
<(*0*, ∞), inc>	<(*0*, ∞), dec>
<*disclmX*, std>	<*disclmY*, std>
EQU	

In the model depth test stage, the parameters have the initial values

X=<(*0*, ∞), inc>,

Y=<(*0*, ∞), dec>,

Z=<(*0*, ∞), dec>,

and QSIM creates the behaviors in Table 6.2.13, which differ only in the final magnitude of Z, for the X-Y-Z system.

TABLE 6.2.13. Behaviors of X-Y-Z system

X	Y	Z
<(*0*, ∞), inc>	<(*0*, ∞), dec>	<(*0*, ∞), dec>
<(*0*, ∞), inc>	<(*0*, ∞), dec>	<(*0*, ∞), dec>
<*newX*, std>	<*newY*, std>	<*newZ*, std>
	EQU	

Quantity space of Z: {-∞, *0*, *newZ*, ∞}

X	Y	Z
<(*0*, ∞), inc>	<(*0*, ∞), dec>	<(*0*, ∞), dec>
<(*0*, ∞), inc>	<(*0*, ∞), dec>	<(*0*, ∞), dec>
<*newX*, std>	<*newY*, std>	<*0*, std>
	EQU	

X	Y	Z
<(*0*, ∞), inc>	<(*0*, ∞), dec>	<(*0*, ∞), dec>
<(*0*, ∞), inc>	<(*0*, ∞), dec>	<(*0*, ∞), dec>
<(*0*, ∞), inc>	<(*0*, ∞), dec>	<*0*, dec>
<(*0*, ∞), inc>	<(*0*, ∞), dec>	<(-∞, *0*), dec>
<*newX*, std>	<*newY*, std>	<*newZ2*, std>
	EQU	

Quantity space of Z: {-∞, *newZ2*, *0*, ∞}

Should the model test fail, since three behaviors were obtained when one was wanted? Clearly not, because a closer examination of the QSIM output shows that it actually contains only the single input behavior, when one restricts attention to the parameters in the input. So the current model in this example (whatever it is) is acceptable.

If a *subsystem* is defined to be a subset of the set of parameters, the specification of the model depth test stage may be worded as follows: The model will be labeled satisfactory if the number of the input subsystem's behaviors in the QSIM output is acceptably close to the number of QSI input behaviors. The simulation cutoff mechanism should check this number, and the level limiting mechanism should keep the "elasticity" of the behaviors in mind when making its decision.

Note that the above-mentioned features mean that this stage will generally take less time than a similar number of pure QSIM simulations with the same input model would require.

If the "QDE" to be tested is empty, then model testing automatically fails without any simulation performed; model extension is clearly necessary. This trivial case may occur only immediately after initial constraint determination. If any non-constant parameter which appears in the input is missing from all of the constraints in the QDE, the depth test again fails without simulation, since an unconstrained parameter would lead to an infinite simulation. (For examples, see Chapter 9.)

Model testing is automatically satisfied if the number of iterations has exceeded the specification in the input. This guarantees that the algorithm terminates even for pathologically unrelated parameters in the input.

6.2.7. Dimension Consistency

The final stage of QSI is a procedure of model rationalization, where the previously discovered relations are made to fit into arithmetically sensible constraints. QSI is totally ignorant about the nature of the input quantities in the beginning. But when the constraints are found, simple rules of mathematics imply certain relations among the dimensions of the parameters

in the constraints. If these relations are contradictory, the model can be rationalized by the use of buffer M+ constraints and parameters.

If a DERIV(X,Y) exists, for example, this implies that X and Y's dimensions are not the same, (Y has X's dimension divided by time) so they can not appear in additive constraints together, since ADD and MINUS obviously require all their arguments to have the same dimensions. So if, for instance, ADD(X,Y,Z) also appears in the QDE, it is not acceptable, and is removed from the model. But one does not want to lose the addition relation whose existence in the system has been discovered. Therefore, the M+ constraint type, which can be viewed as a "dimension converter," is used. Three new (buffer) parameters B1, B2, and B3, whose quantity space structures are identical to those of X, Y, and Z, are added to the model, together with the constraints M+(X,B1), M+(Y,B2), and M+(Z,B3), which have CVs linking each of B1, B2 and B3's landmarks to (respectively) X, Y and Z's landmarks. Since B1, B2 and B3 will have exactly the same behaviors as X, Y and Z, the constraint discovered among X, Y and Z will exist between them too, so ADD(B1,B2,B3) is also added. One now has the same model, (from a simulation point of view,) but without the inconsistency.

The actual mechanism of this stage is a little more complicated than the one just described, since some inconsistencies can be discovered only by considering (possibly long) chains of constraints. Suppose, in the above case, one did not have DERIV(X,Y), but the two constraints DERIV(X,P) and DERIV(P,Y). Understanding that something is wrong with ADD(X,Y,Z) would then require traveling along this chain of DERIVs. As another facet of the dimension consistency imposing problem, consider that the arithmetic constraints may form such chains too. Suppose one has

DERIV(A,E),
ADD(A,B,C),
ADD(C,D,E).

The fact that ADDs and MINUSes which share parameters in this manner form equivalence classes of parameters of the same dimension has to be recognized and handled by the buffering algorithm.

Since dimension consistency always applies in the real world, the buffer parameters and constraints created in this stage usually hint at actual deep model components, as the example of Section 6.2.3 showed. As extra constraints

which do not contribute to any behavior pruning, the buffer M+s would certainly slow down a QSIM simulation of the model, but this is not a problem, since QSI does not make any simulation after their creation.

The interpretation of what QSI's output actually means is quite involved, and will be the subject of a later section.

6.3. Noise Filtering

Depending on the specifics of the application, two preprocessors may be involved in the preparation of the QSI input. If the robot is obtaining the knowledge about the behavior of the system from actual numerical measurements, what it originally has is a group of parallel sequences of visible parameter values; with a real number for each parameter value at each discrete point of measurement. This (possibly long) input can be converted to a (usually much shorter) qualitative behavior by a preprocessor. Whole sequences of measurement points in which each parameter value changes in only one direction are collapsed to single qualitative states. This operation can be accomplished in time linear in the number of measurements. Each qualitative behavior in the input is obtained by a separate run of this simple algorithm. Other qualitative reasoners which have to perform this quantitative to qualitative behavior conversion (e.g. for *tracking* monitored systems) also employ similar methods. For a detailed (and much more advanced) discussion of the issues about this conversion, see [50].

QSI's input has to be a correct description of the system's behaviors if it is expected to perform successfully. If the input is stemming from measurements of the real world, it may be corrupted by noise. *Noise* will be defined as the differences between the measurements and the actual parameter values, caused by any conceivable reason. Note that the qualitative representation is particularly suitable (in fact, it was designed) for abstracting away unimportant value fluctuations. Therefore, the noise may well have been eliminated if the input has been prepared by a human, maybe even inadvertently. In some cases, noise has no effect on the qualitative description.

Consider a parameter increasing in $(0,\infty)$, with no known positive landmarks. The measurement of this parameter is being continuously corrupted by noise so that, at each reading, say, five units more than the actual value is presented. The resulting qualitative behavior will again contain the value <$(0,\infty)$, inc> for this parameter, and the noise will have been "in vain."

Despite these resilient features of the representation, a qualitative noise filter has been developed. The filter is aimed at individual parameters, specified in its input. (Because of the particular configuration of the experiment, it may be the case that only some parameters are subject to noise, and others are not.) If QSI (without running this preprocessor) fails to find a good model within the allowed iteration limit, this can lead one to suspect the existence of noise. Possibly noisy parameters can be identified as the ones with an unusually great number of distinguished time-points in their behaviors.

The filter's input is the set of system behaviors, its desired sensitivity (see below,) and the names of parameters to be filtered. Its output is a shorter (less noisy) set of system behaviors, and smaller quantity spaces for the filtered parameters. The following is an explanation of its working.

Many kinds of noise exist, but attention in this study will be restricted to *white noise*, which can be modeled as a sequence of independent and identically distributed random variables of zero mean. It is also assumed that, when it exists at all, the variance of the noise is not very big, so it causes the measurements to read values "slightly" greater or less than they normally would, with equal probability. For example, if the plot of the magnitude of parameter X is "really" as shown in Figure 6.3.1, one intuitively expects its noisy version to be as in Figure 6.3.2. By the same reasoning, if one sees Figure 6.3.2 and is told that noise is present, then one would propose something like Figure 6.3.1 as the noiseless (filtered) version. This is the qualitative analog of the *convolution* technique, used in, for example, the *early processing* phase of the vision process [2], to "smooth" the lines that will be obtained. As with all filters, there is an inherent tradeoff involved in this technique: If you go too far with the smoothing, real features of the behavior may get wiped out too, if you are too cautious to avoid this, however, you run the risk of leaving actual noise unfiltered. There is no perfect solution to this problem, and this kind of noise filters are "heuristic" by their nature.

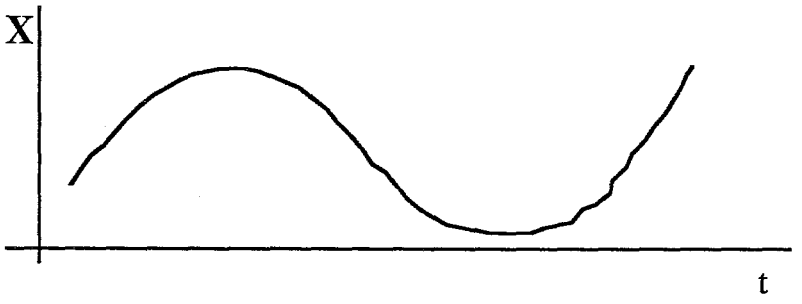


Figure 6.3.1. Actual behavior of X

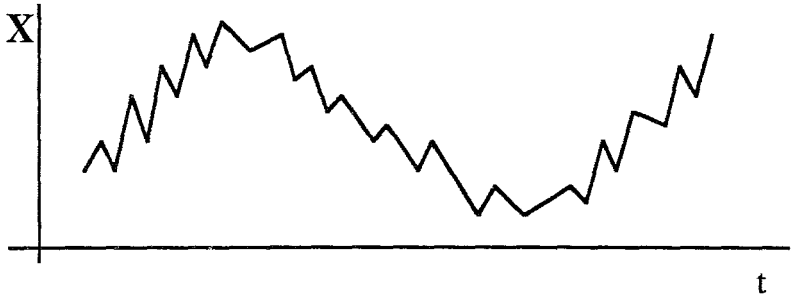


Figure 6.3.2. Noisy behavior of X

The implementation of the qualitative noise filter is quite different from its quantitative analog. This is to be expected, since the qualitative noise filter takes *qualitative* behaviors as input, and the averaging process of convolution cannot be applied in this format, since there are no ordinary "numbers" to be averaged. The qualitative filter again uses the ordinal relationships among landmarks to achieve its aim.

An examination of Figure 6.3.2 reveals the undesirable features of the noisy behavior, in addition to its being incorrect. A huge number of landmarks have to be kept in its quantity space to describe this behavior. Note that Figure 6.3.1 requires only two landmarks outside the basic set. The noise landmarks cause the behavior to have an unacceptably great number of distinguished time-points and states. Even worse, they decrease the intelligibility of the behavior and make it lose the advantages of qualitativeness.

A sequence of values in which the parameter's direction starts as **inc** (or **dec**) in the first one or more values, becomes **std** once, and then is **dec** (or **inc**) for one or more steps is called a *tooth*, because of the way it looks in a plot of the parameter, like Figure 6.3.2. The basic idea of the filter is to replace long sequences of teeth during which there is a general increasing (or decreasing) of magnitude with single values with direction **inc** (or **dec.**)

The sensitivity of the filter is an integer specifying a lower limit for the length of tooth sequences to be smoothed. After all, the behavior of Figure 6.3.1 is a (short) sequence of teeth itself, and one does not want such things to be smoothed.

A tooth sequence to be filtered in a given parameter behavior is determined as follows: Sequences (as long as possible) of values where a ziczac of directions (like {inc..., std, dec..., std, inc... }, where the ellipsis (...) means "zero or more of the preceding,") exists are identified. Filtering can be accomplished if: a) The number of **std**s are above the filter's sensitivity, b) The parameter never has the value <*lm*, std> outside this sequence for any landmark *lm* for which it has such a value in this sequence, c) The value <*0*, std> does not appear in the sequence, and d) The odd-numbered (i.e. 1^{st}, 3^{rd}, 5^{th},...) landmarks on which the parameter becomes **std** in this sequence are in increasing (decreasing) order, and the same applies for the even-numbered landmarks.

If all these conditions hold, this sequence of values is replaced by the value <Mag, Dir> where Mag is the interval in the quantity space of the parameter which is formed after deleting all the landmarks on which it became **std** during the sequence, and Dir is the direction of the ordering determined in condition (d) above.

Conditions (b) and (c) are required to prevent the filter from destroying useful landmarks by mistaking them for products of noise.

Condition (d) is the check for the "general increasing (or decreasing) of magnitude" mentioned before.

Here is an example showing how a noisy sequence is smoothed by the filter.

<(B1,B2),inc>
<B2,inc>
<(B2,T1),inc>
<T1,std>
<(B2,T1),dec>
<B2,std>
<(B2,T1),inc>
<T1,inc> --------> <(B1,T3),inc>
<(T1,B3),inc>
<B3,inc>
<(B3,T2),inc>
<T2,std>
<(B3,T2),dec>
<B3,std>
<(B3,T2),inc>
<T2,inc>
<(T2,T3),inc>

The system behaviors have to be renewed after parameter filterings are complete. Since successful filtering shortens parameter behaviors, i.e. reduces the number of the parameter's distinguished time-points, the system composed of these parameters will have much shorter behaviors, too. This reduction in the size of the QSI input (keeping in mind that the quantity spaces are also smaller now) is naturally an improvement from the point of view of the time requirement. (See Chapter 7.)

Noise filtering of a single parameter can be accomplished in a single pass, i.e. linear time. The same applies for the global behavior shortening.

VII. COMPLEXITY AND CORRECTNESS ANALYSIS OF QSI

This and the following two chapters will focus on various aspects of the QSI algorithm presented in Chapter 6. Section 7.1 examines the time complexity issue, while formal derivations of various claims made without proof in Section 6.2 will be presented in Section 7.2. The final section contains the rules that QSI uses in order to detect consequence and insignificant constraints (Sections 6.2.5 and 6.2.6,) together with their proofs.

7.1. Complexity

The computational complexity of QSI will now be determined stage by stage. Most of the required analysis has already been done in the Section 6.2. In the following, s_0 is the number of states in the input, p_0 is the number of input parameters, s_i and p_i are these numbers in the i^{th} iteration.

7.1.1. Analysis

Constraint Determination

Since worst-case complexity is being considered, assume none of the conditions (Section 6.2.4) which let the algorithm skip testing a constraint are fulfilled. Also assume that each constraint is satisfied for the first $s_i - 1$ states, so no shortcut is obtained. The constraint determination after the i^{th} iteration then requires $O(p_i^3 s_i^2)$ time for large p_i. In further iterations, p_i, and, generally, s_i will increase. Always, $s_{i+1} = O(s_i)$, since all new parameter behaviors can be expressed simply by replacing (in the worst case) each interval state in the

system behavior by three-state sequences of interval-point-interval states. If constraint determination has to be performed for a second time, and if full postulation mode is active, p_1, the total number of parameters on which the algorithm will work, will be $O(p_0^2)$, so the second determination will take $O(p_0^6 s_0^2)$ time. Experience shows that only a small fraction of the new parameters are actually added to the model after determination (especially in half-search mode,) so $p_{i+1} = O(p_i)$ for $i \geq 1$, and constraint determinations in later iterations also require $O(p_0^6 s_0^2)$ time. If one considers a (very) pathological case in which *all* neighbors get added to the model at each iteration, this stage's time requirement would be on the order of $p_0^3, p_0^6, p_0^{12}, p_0^{24}$, etc. in successive iterations, i.e. it would be exponential in the current iteration number.

Model Depth Testing

Again assuming that no shortcuts are possible, this stage consists of a number of QSIM runs with level limiting. QSIM's worst-case complexity is exponential in the number of parameters. As mentioned above, the number of QSIM input parameters is usually linear in the QSI input parameters, but the worst-case analysis about the parameter number stated in the above paragraph still stands. Note that this stage is exponential in s, because of QSIM's nature.

Model Extension

The points made above about the growing number of parameters apply here, too. The time requirement is linear in the number of postulated parameters, which is $O(p_0^2)$ in normally all iterations, but can rise as $p_0^2, p_0^4, p_0^8, \ldots$ in the worst case. Note that this problem can only occur in postulation modes which involve sums, differences, products, or ratios, since only the numbers of these kinds of neighbors involve squared terms. In all other postulation modes, the number of neighbors is linear in the number of the old parameters, so the "explosion" does not occur even if all the new parameters are added to the model, which is itself a very rare situation.

The behavior calculation procedure, which is performed for every parameter that is postulated, is generally linear in the number of states, (note that the *quantitative* version of this task is linear in the length of the input,) but unfortunately, the fact that one calculates all possible behaviors of the

minimum length and the ambiguity of qualitative arithmetic mean that pathological cases (involving ADD or MULT as the defining constraint) in which the time requirement is exponential in s can occur. Consider the two parameters X and Y, which have the following values throughout the (long) system behavior:

X=<(0, ∞), inc>
Y=<(-∞, 0), dec>

Now consider the new parameter Z, whose defining constraint is ADD(X,Y,Z). Clearly, Z can have *any* value at any time, only restricted by T-legality. To see that the calculation of all of Z's behaviors is exponential in s, note that this procedure is equivalent to the production of several trees whose depths are equal to s. Each possible value that Z may take at t_0 is a root. Each transition that it may undergo is a link between nodes.

The application of the behavior selection heuristics is linear in both s and the number of alternative behaviors, which can be exponential in s, by the reasoning of the above paragraph. This is another factor which suggests the use of specific postulation modes for greatly improved efficiency.

The complexity of constraint determination, which is also part of the model extension stage, was discussed earlier.

Dimension Consistency

The last stage of QSI is also the fastest. Since it simply involves scanning the constraints in the QDE to find dimension relations among the parameters, it can be completed in time polynomial in c_i, the current number of constraints. Note that c_i itself is linear in the current number of parameters in the model.

7.1.2. Remarks

The "good news" about the complexity of QSI is that most of the analysis just performed entailed very pessimistic assumptions. In practice, the consequence and insignificance detection checks omit a lot of constraints in constraint determination. Constraints that do get checked against states are usually "shot down" very early in this process. Full postulation mode is not necessary for a

wide class of problems, similarly for full search mode. A lot of problems have been solved elegantly using the derivative postulation mode, see Chapter 9. p_0 and s_0 are usually quite small, so the grim expectations suggested by the determined time requirements are not realized. The input sizes generally used in this text are typical in the qualitative reasoning literature; also keep in mind the basic assumption that QSI "sees" only some of the system parameters, which reduces the number of parameters in its input compared to other reasoners. If the algorithm will be used to find QDEs for individual model fragments, as currently envisioned, the input sizes will rarely turn out to be problematically big. As explained in Sections 3.1.3 and 3.2, active research is going on [1] to improve QSIM's performance on medium and large-scale systems. The results of such research will certainly be useful for QSI as well, since it uses QSIM as a subroutine.

Although clearly very high when compared to algorithms dealing with simpler forms of data processing, the complexity of QSI is similar to those of other qualitative reasoners, and is quite acceptable, considering the nature of the task performed. For an idea about the actual performance, see Table 7.1.1, which lists the execution times of some of the problems in Chapter 9 in the current PC implementation (Appendix A.) The section numbers indicate where each problem has been presented. Details can be found in Chapter 9.

TABLE 7.1.1. Execution times of QSI case runs

Problem*	number of states in input	number of parameters in input	number of model extensions	number of constraints in final QSIM input	number of constraints in final model	execution time (s)
U-tube	6	2	1	7	11	3.25
9.1.2	3	1	1	5	9	1.18
9.1.3	3	2	1	3	6	0.72
9.1.4	4	2	1	17	41	42.51
9.1.5	5	3	1	6	13	19.14
9.1.6	5	1	2	2	2	1.01

*The derivative postulation and half search modes have been selected in all of the above.

7.2. Correctness

The discussion will begin with the "heart" of the QSI process, namely, the constraint determination algorithm. The following assumes that full search mode has been selected in the input.

Definition 7.2.1. A constraint is *significant* if: a) it cannot be proven using already known constraints as axioms, and b) it is not of the form ADD(A,B,C) where MINUS(A,B) is already known.

The reasons for such a distinction between constraints were already discussed; the formal definition is given here so that it can be invoked in the following propositions.

Proposition 7.2.1. All significant constraints valid in the behaviors that constraint determination obtains as input appear in its output.

Proof. Assume for the moment that the contradiction, consequence and insignificance checks are absent, and one has a "pure" constraint determination algorithm, as seen below. The proposition will first be proven for this algorithm. (Note that the omitted checks were there to improve the efficiency. They will later be incorporated again to show that the proof stands.)

```
for each constraint type CT do
 for each tuple ARG of parameters that can be arguments to CT do
  blockbegin
   for each qualitative state in the input do
     if CT(ARG) does not hold
     then
      break out and go to blockend
   {At this point, CT(ARG) is a novel constraint valid throughout the input}
   write(CT(ARG))
   add CT(ARG) to the QDE of the system
  blockend
```

Assume that the above algorithm has terminated without a constraint C that is valid throughout the input being written out. C must have been

generated at the "top" of the algorithm by the for statements, since *all* possible constraints are generated there (by construction.) This means that the check in the innermost for failed, that is, there is an input state in which C does not hold. But this contradicts the assumption that C is valid in the input, so pure constraint determination has been proven to find all valid constraints.

The consequence and insignificance checks result in certain constraints being written out without being tested; therefore their inclusion cannot cause any valid constraints to be missed.

The contradiction check causes constraints rendered impossible by present information to be skipped. The rules that may be used are:

M+(A,B)	----->	MINUS(A,B) is impossible.
M+(A,B)	----->	M-(A,B) is impossible.
MINUS(A,B)	----->	M+(A,B) is impossible.
M-(A,B)	----->	M+(A,B) is impossible.
NOT(M-(A,B))	----->	MINUS(A,B) is impossible.

These are easily seen to hold, except in the case where both A and B are fixed, which makes it possible for both M+(A,B) and M-(A,B) to be trivially satisfied at the same time. However, the special treatment given to fixed parameters by the algorithm means such constraints will not be necessary for simulation, and it is not sensible to talk about such relations between constants anyway. So the contradiction check will eliminate no significant and valid constraints, and the proof is complete.

Proposition 7.2.2. No constraint that is not valid throughout the constraint determination stage's input appears in its output.

Proof. Consider the "pure" constraint determination algorithm again. For any constraint C to be written out, the innermost for statement has to be completed; that is, C has to hold in each input state. Therefore, the proposition holds for the pure version.

The contradiction check does not change the output, in particular, it does not add anything to it, (see discussion above,) so the proof stands.

Insignificant constraints which do not satisfy the requirement of Definition 7.2.1(b) are *not* written out by the algorithm. All other constraints

which satisfy the consequence or insignificance tests *are* valid; see discussion in Sections 6.2.4, 6.2.5 and proofs in Section 7.3. This means the incorporation of the checks does not cause the inclusion of any invalid constraint in the output; the proof is complete.

The previous two propositions can be combined to form the following statement of the correctness of the constraint determination procedure:

Proposition 7.2.3. The constraint determination stage finds all, and only, the valid constraints that hold among the parameters in its input.

Although there is already strong intuitive evidence for it, the fact that QSI really achieves correct system identification, i.e. the QDEs that it finds really produce the input behaviors when simulated, will now be formally established. It is first shown that there is an (admittedly easy) solution to any system identification problem.

Proposition 7.2.4. For any T-legal behavior, a constraint set which will produce it when simulated from its initial state can be found.

Proof. The empty set (∅) has this property for any T-legal behavior. When started by the initial state of the behavior, QSIM will produce an infinite tree, each branch of which corresponds to a qualitatively distinct account of the manner the parameters change value, constrained only by continuity. (Part of) one of the branches will be the given behavior.

Of course, this is the trivial case. One is really interested in bigger constraint sets. Note that, by the same reasoning as above, a "model" can be found, given *any number* of behaviors of a system. One should also point out that, given QSI's lack of knowledge of where its input comes from, there is always the possibility that the "parameters" in the input are really unrelated to each other, in which case the empty model is the correct solution.

Proposition 7.2.5. When the constraint set found by the constraint determination procedure is used as the QSIM input together with the initial states of the input behaviors, all the input behaviors appear in the QSIM output; that is, correct system identification is performed.

Proof. The model ∅ does produce the input behaviors when simulated from their initial states, as already discussed. The addition of constraints to this model

will cause the infinite trees it would produce to get smaller. More specifically, the addition of any constraint C will prune all, and only, the states in which C does not hold, together with their descendants, from each tree. (See QSIM description in Chapter 3.) However, every constraint found by constraint determination holds in every state of the input behaviors, (by Proposition 7.2.2) which means they will not be pruned, and all these behaviors will appear in the simulation output.

Note that by "the input behaviors," the data on which the constraint determination procedure operates are meant; these will be larger than QSI's initial input in later iterations. So the above proofs stand for each model found in successive iterations of the algorithm.

In the previous chapter, it was established that a heuristic method is necessary for behavior assignment to neighbor parameters, since there are cases where an infinite number of alternative behaviors for a single parameter exist. This inevitably means that QSI outputs may lack some possible relationships among the deep model parameters, if model extension has been performed. The choice of the heuristics was made with this fact in mind, aiming to minimize the number of overlooked relationships.

Finally, it will be proven that model extension never produces "shallower" models according to QSI's criterion of model depth, that is, fewness of QSIM behaviors predicted by the model. Note that this is not obvious; in model extension, both the number of constraints and parameters increase, more constraints tend to decrease the number of behaviors, but more parameters generally mean more behaviors. The following shows that, after model extension, one never obtains more QSIM behaviors than those obtained before extension:

Definition 7.2.2. The number of behaviors of the input subsystem that would be predicted at the i^{th} execution of the model depth test stage if the behavior count cutoff and empty model controls were absent is called the *i^{th} input subsystem behavior count*, or $ISBC_i$.

If the result of initial constraint determination is the empty model, then $ISBC_1 = \infty$, as already mentioned.

Proposition 7.2.6. For any QSI run in which the model depth test stage is executed more than once, say, n times, $ISBC_i \geq ISBC_{i+1}$, for all i, where $i < n$.

Proof. Assume that an input subsystem behavior predicted at a later execution of depth testing was not predicted at an earlier execution. This means that there was at least one constraint in the model which caused that behavior to be filtered out during the earlier testing, and this constraint was not present in the later one. However, this is impossible, since constraints added to the model at the end of model extension are never removed, i.e. the QDE can never get smaller. Therefore, all behaviors of the later stage must also be present in the earlier stage, that is, $ISBC_i < ISBC_{i+1}$ is never the case.

The reason why an integer representing the maximum number of allowed iterations was included in the input also becomes clear now. We have no proof that $ISBC_i$ is strictly greater than $ISBC_{i+1}$ for all i. Without this, one cannot prove that the algorithm will terminate for all cases (although the examples show that it does for a lot of useful ones,) so an iteration cutoff is necessary, to be on the safe side.

7.3. Consequence Constraints

In Chapter 6, the consequence and insignificance checks, which are parts of the constraint determination algorithm, were mentioned. Both these checks are used to see whether a particular constraint is already implied by the current knowledge of constraints or not. If it is implied, the process of checking the constraint against each input state is unnecessary and is skipped. It is a nontrivial task to impart the total knowledge of qualitative algebra required to identify all consequence constraints to the program, and it is not claimed that the following list is complete. Recall that the existence or absence of consequence constraints in the constraint determination stage's output affects only the efficiency of the algorithm, not its correctness.

In the following, the left hand side of the arrow contains conjunctions of "known" constraints; these are either in the QDE, or previously discovered consequences of those in the QDE. The right hand sides are the consequences.

M+(A, B), M+(B, C) --------> M+(A, C)

Proof. Recall that A, B, and C are functions of time. Furthermore, the M+s mean that there exist functions F and G such that A(t) = F(B(t)) and B(t) = G(C(t)) (with the proper domains and ranges,) and both F' and G' are positive everywhere in their domains [14]. But this means that

$$A(t) = F(G(C(t)))$$

and since the derivative of the composite function FoG ≡ H is the product of F' and G', $H' > 0$ throughout its domain; A(t) = H(C(t)) is the very definition of M+(A, C), and the proof is complete.

A more "qualitative" proof for the same rule is as follows: Consider all the qualitative directions that the parameters may take on. Given the antecedents, the possible direction tuples are those of Table 7.3.1. Obviously, M+(A,C) "holds" in each possibility. (CV information of the consequence constraint is also handled by the CVs of the two antecedent constraints, using parameter B as a sort of "bridge.")

TABLE 7.3.1. Possible directions of A, B and C

A	B	C
inc	inc	inc
dec	dec	dec
std	std	std

The first two of the following rules have similar proofs as the one above.

M+(A, B), M-(B, C) --------> M-(A, C)

M-(A, B), M-(B, C) --------> M+(A, C)

ADD(A, B, C), M-(A, C) --------> M-(A, B)

The proof is again very simple. The antecedents say that

$$A(t) + B(t) = F(A(t))$$

where F' is negative. Rearranging, one gets

$$B(t) = -A(t) + F(A(t))$$

Clearly, there exists a function G such that

$$B(t) = G(A(t))$$

and G' is negative in its domain. This means that M-(B, A).

Similarly,

ADD(A, B, C), M+(A, B) --------> M+(A, C).

The fact that the derivative of a constant is zero implies the following rules, where K is a parameter already known to be fixed at a landmark:

ADD(A, B, K) --------> M-(A, B)

ADD(A, K, B) --------> M+(A, B)

Other rules make use of the properties of MINUS and the fact that rearranged equations still "say" the same thing:

MINUS(A, B) --------> M-(A, B)

ADD(A,B,C), MINUS(A,D) --------> ADD(C, D, B)

ADD(A,B,C), ADD(C,D,A) --------> MINUS(B, D)

Since all constraints except DERIV are commutative, the left hand sides of the above rules may be changed to reflect this fact; they will still apply.

VIII. QSI: A DISCUSSION

This chapter aims to put QSI in perspective: Its relation with existing techniques of modeling, aspects of its utilization, and ideas about using it for different applications, such as diagnosis, are discussed. QSI and the method of inductive learning are compared.

8.1. QSI as Modeling

As well as being a natural counterpart to QSIM, (i.e. system identification vs. system simulation,) QSI also provides a new approach to the qualitative model formulation problem. Its ability of finding significant deep relationships among the system's quantities can be used to write the "best" model, given a system. A striking example where QSI can propose a better model than the obvious one is the spring/block system of Section 3.1.3. Recall that the three-parameter, three-constraint model of Table 3.1.11, which, although mathematically adequate to produce only a single periodic behavior, (Table 3.1.12) leads to a simulation with infinite spurious solutions, because of inherent representation problems. It was mentioned in the same section that a more comprehensive model containing energy laws applying in that situation does produce the single behavior output, but it is not obvious for the user how the model should be formulated in the beginning. Now suppose that the behavior of Table 3.1.12 has been presented to QSI as input. The initial constraint determination finds the constraint set of Table 3.1.11, as expected, plus the constraint MINUS(X,A). (Remember that dimension consistency is not imposed until QSI terminates.) The model is found to be unsatisfactory by the depth test stage, since three behaviors (those of Tables 3.1.12 thru 3.1.14) are predicted by QSIM, so model extension is performed. Among the new

components added to the model by this stage are the parameters P9 and P10, defined by the constraints

MULT(X,X,P9) and MULT(V,V,P10),

and the constraint

M-(P9,P10)

The bigger model *is* satisfactory, and QSI ends.

P9 and P10 correspond to the potential and kinetic energies, respectively. The M- constraint among them represents the fact that the sum of these two energies, i.e. the total energy, is constant. The relevant "real-world" equation [51] is

$$\frac{1}{2}mv^2 + \frac{1}{2}kx^2 = E \qquad (8.1)$$

Note how constants like m, the mass of the block, k, the spring constant, and the total energy are "buried" in the constraints QSI finds; this will be taken up in Section 8.1.2. QSI's usefulness as a modeling tool has thus been demonstrated. The rest of this section is comprised of further discussions of some aspects of QSI's utilization.

8.1.1. How to Prepare QSI's Input

As pointed out in Section 6.1, the procedures of observation (and possibly, excitation) of the system to obtain the accounts of its behaviors are outside QSI's specification. The algorithm operates with the assumption that the input has been obtained so that: a) Each qualitative behavior correctly describes the corresponding family of actually exhibited (or, in design applications, expected) behaviors, and, b) As many distinct qualitative behaviors that the system can exhibit as possible have been included. Both of these conditions may be difficult to meet in practice in some cases (see the section on qualitative noise filtering in relation to condition (a),) and automation of the data collection task has to be an important target for future research.

But before the considerations mentioned above can even arise, one has to decide (even roughly) *what* the system *is*, that is, which observable

quantities to include in a single QSI input as shallow parameters. In all the examples in this text, and probably in the QSI modeling applications in the foreseeable future, the problem will be clearly defined by the user, who knows it is about a tank system, spring, etc. An independent intelligent robot would have to perform this problem definition by itself. Suppose such a robot enters a big room in, say, a chemical plant, which it is "seeing" for the first time. The room contains many tanks with fluids in them; some of them are connected by pipes, some are not. Most probably, all observable quantities in the room would not be parameters of the same system. Rather, it would be more suitable to partition them to a number of independent systems. This allocation task involves many perception and modeling issues, some of which are related to QP theory, some others outside the scope of the qualitative reasoning area itself, let alone this study.

Another feature which has already been mentioned is QSI's total disregard of possibly useful information about the "natures" of the parameters, including their dimensions. Thus, QSI views its input simply as accounts of the changing of some collection of quantities over time. The output that it produces then reflects various mathematically possible relations on these nameless quantities. Humans clearly do not "act" like this when performing modeling, as will soon be discussed.

In this regard, the work of Bhaskar and Nigam [52] in which the *dimensional representations* of the relevant variables of the system are presented as input to the qualitative reasoner, without explicitly stating the physical laws, can be seen as an interesting opposite of the approach taken by QSI.

8.1.2. How to Interpret QSI's Output

The constraints in QSI's output provably hold on the set of parameters, as already seen. But the interpretation of these constraints to obtain a "real-world," e.g. verbal, model involves some issues arising from the natures of the representation and the algorithm.

The qualitative representation tends to "look over" constants, since systems can usually be modeled just as tightly without them. The monotonic

constraints are especially useful for this purpose. Take the spring/block example again. The well-known formulae for the force on the block are

$$F = ma \tag{8.2}$$

and

$$F = -kx \tag{8.3}$$

which can, if wished, be translated into the qualitative representation directly by defining parameters for all the quantities in the equations, and two MULT constraints. But one generally does not do this, and uses instead M-(X,A) with CVs *(0,0)* as in [14], since it is more sensible from a simulation point of view to choose the smaller of two equally strong models.

So the M constraints presented by QSI can "hide" other relationships in them, which may be interesting to examine if even deeper modeling is desired. Only two very simple examples will be considered. Each M may be the abstraction of an arbitrarily long chain of Ms, for instance, M+(A,B) can mean M+(A,P1), M+(P1,P2), M-(P2,P3), M-(P3,P4), M+(P4,B). (For an example to such chains of Ms in real models, see the water balance mechanism in the human kidney, modeled in [8,9], and also Section 8.1.4.) M-(X,Y) may have been derived from, say, an equation of the form $XY = K$ $(K > 0)$ among many others. Since the input can be an abstraction of not one, but a family of systems, the output also reflects all of these possibilities. The user can choose the most suitable one from among the alternatives implicit in the output of QSI, in the role of a modeling aid.

Another issue that may come up is the "discovery" of some deep parameters that do not seem meaningful when considered in a real-world context, given their defining constraints, and the meanings humans give to their defining neighbors, something QSI is unable to do. For example, in the U-tube problem of Section 6.2.3, if full postulation is used, the following new constraints and parameters are among the ones added to the model by the extension stage, in addition to those already discussed:

MULT(amount_A,amount_B,P10) MULT(amount_A,amount_A,P11)
ADD(P10,P11,amount_A)

Even after dimension consistency is imposed, this relationship does not reflect any intuitively obvious feature of the U-tube. The very idea of multiplying the amounts in the tanks by each other, or squaring them, does not make sense. The answer is, of course, this *need* not be part of a model of the U-tube. The relationship has been discovered, because it is mathematically implied by the input behaviors. The existence of this *coincidental* constraint is the proof of existence of a "system" (which would probably be much harder to physically visualize, compared to the U-tube,) in which the parameters A and B behave as specified in the input, and whose QDE includes the squares and products of A and B and links them in a "meaningful" way. Again, the user, with his knowledge of the natures of the quantities, can choose the meaningful constraints from among the ones QSI presents. Note that, even if no such selection is made, the QSI output is still guaranteed to produce the input when simulated, i.e. even the unintuitive components cannot filter out the input behaviors.

8.1.3. QSI vs. Modeling by Humans

Modeling is a tremendously important mental activity, which is very hard to automate. QSI must be viewed as an early step in this direction. It is not known which processes go on in the human mind when one attempts to solve problems of the kind discussed in these chapters, but most probably, the approach taken by the brain is not QSI's method of trying out all the possibilities. Many very "human" capabilities, among them, the use of analogies [2], and pure "insight" may come into play. In this regard, QSI is taking what Rothenberg [53] calls the "engineering" approach to AI: Exploiting the computer's abilities to come up with methods for solving the problems, without caring whether humans solve them in the same way.

Having said this about QSI's relation to the naive modeling activity, let us briefly compare it with what can be called "expert modeling," i.e. the task of writing down the algebraic or differential equations describing a system. This task is normally performed by scientifically oriented people, so one might expect there is a more formal way in which it can be described. However, this does not seem to be the case. The modelers use their previous knowledge of laws that may apply in the current situation, and again seem to employ "insight" to

recognize the particular law instances that do apply. A QSI-style search is absent. Iwasaki and Simon [28] say that "good" models should contain each different law in a different *structural* equation, like equations 8.2 and 8.3, rather than combining them, like writing

$$mA = -kx \tag{8.4}$$

for the spring/block system. In this manner, modifications in the physical situation which cause different laws to apply can be handled nicely. Note that QSI does not (and can not) impose such a form on the output models; this would again be the responsibility of the model user, using the guidelines of Section 6.4.2. As Iwasaki and Simon point out in [28], "Establishing the structural equations for a system is as much an empirical as a formal matter, and certainly not a syntactical exercise."

8.1.4. QSI for Diagnosis

Kuipers [8,9] has proposed a medical diagnosis expert system based on a "hypothesize-and-match" architecture which combines a first generation expert system with QSIM. Clinical findings are fed to the first-generation system to obtain a set of "candidate" diseases. Previously prepared QDEs describing each of the diseased mechanisms are simulated one by one, and the QSIM predictions are compared with the clinical observations, completing the cycle. Diagnosis is achieved when the observations match the predictions.

When behavior data about a reasonably big subset of the parameters are available for both the healthy mechanism and the present state of the mechanism, QSI offers an alternative approach to the diagnosis problem. Identifications of both sets of behaviors can be performed. By this method, the disease QDE can be found immediately, without going to the trouble of simulating a lot of non-answers. The first generation expert system is rendered unnecessary. If the "QDE-base" of possible diseases mentioned in the above paragraph is present, matching its entries with the QSI output leads to diagnosis. Even if such a disease dictionary is not available, comparison and contrasting of the models of the healthy and diseased mechanisms may give useful hints to a human expert of the domain about the nature of the problem.

For example, Tables 8.1.1 and 8.1.2 contain healthy and diseased models [8,9,41] of the water balance mechanism (Table 8.1.3) of the human kidney, respectively. This mechanism governs the relationship between the ingestion and excretion (in the urine) of water in the body. The disease of Table 8.1.2 is called SIADH (Syndrome of Inappropriate Antidiuretic Hormone Secretion.) A detailed discussion of what goes on in the kidney in the normal and abnormal cases, together with explanations of the parameters, can be found in the references. Assume that QSI has been shown the behaviors (including most of the parameters; see below) of both the normal and post-SIADH periods in two separate runs. (These behaviors are also presented in [8], they are quite simple accounts of the variables nearing and settling at the equilibrium values in each case.) As has been proven, the QDEs of Tables 8.1.1 and 8.1.2 will be presented as outputs. An expert physician will notice that the differences of the diseased model from the healthy one (the loss of the direct proportionality of the sodium concentration to the ADH concentration, and the fixed higher-than-normal value for the ADH concentration) are signs of SIADH, and concentrate on the problem more quickly, since the many other measured parameters do not appear in the difference set of the two models, and presumably do not contribute to the trouble.

TABLE 8.1.1. Healthy water balance mechanism

CONSTRAINT

MULT(amt(water,P),c(Na,P),amt(Na,P))
M+(amt(water,P),c(natriuretic hormones,P))
M+(c(Na,P),c(ADH,P))
M+(c(natriuretic hormones,P),flow(water,P->U))
M+(c(ADH,P),reabsorbed flow(water,U->P))
ADD(reabsorbed flow(water,U->P),netflow(water,P->U),flow(water,P->U))
ADD(net flow(water,P->U), netflow(water,out->P), netflow(water,ingest->P))
DERIV(amt(water,P),netflow(water,out->P))

{amt(Na,P) and netflow(water,ingest->P) are fixed at positive landmarks. All constraints have CVs at the "equilibrium" values.}

Of course, the above discussion entailed the basic assumption that all the deep parameters were already identified and could be measured; this is a little bit too optimistic, as has been stressed before. But QSI can also be employed to hint at a deeper structure. Chapter 9 contains a modest-sized problem about the

kidney system in which only the most easily observable parameters are in the input, and the QSI solution to it in the derivative postulation mode.

TABLE 8.1.2. Water balance model with SIADH

CONSTRAINT

MULT(amt(water,P),c(Na,P),amt(Na,P))
M+(amt(water,P),c(natriuretic hormones,P))
M+(c(natriuretic hormones,P),flow(water,P->U))
M+(c(ADH,P),reabsorbed flow(water,U->P))
ADD(reabsorbed flow(water,U->P),netflow(water,P->U),flow(water,P->U))
ADD(net flow(water,P->U), netflow(water,out->P), netflow(water,ingest->P))
DERIV(amt(water,P),netflow(water,out->P))

{amt(Na,P) and netflow(water,ingest->P) are fixed at positive landmarks. c(ADH,P) = fixed at a landmark higher than its equilibrium value in Table 8.1.1. All constraints have CVs at the "equilibrium" values.}

TABLE 8.1.3. Water balance model's parameters

PARAMETER	MEANING
amt(water,P)	amount of water in plasma
amt(Na,P)	amount of sodium in plasma
c(Na,P)	concentration of sodium in plasma
c(natriuretic hormones,P)	concentration of natriuretic hormones in plasma
c(ADH,P)	concentration of antidiuretic hormone in plasma
flow(water,P->U)	rate of water filtration from plasma into the tubules
reabsorbed flow(water,U->P)	rate of water reabsorption from tubules back into plasma
netflow(water,P->U)	net rate of water excretion from the blood via the tubules
netflow(water,ingest->P)	rate of water ingestion
netflow(water,out->P)	net rate of change of water in plasma

8.1.5. QSI's Limitations

In addition to the issues already discussed in this section, some other limitations of the QSI algorithm, and possible ways out, will be briefly repeated here.

QSI is very sensitive to possible errors in its input. A single "wrong" state may cause it to fail to find the correct system model, even if all the rest of the input behaviors are described correctly, since the algorithm insists that all output constraints should be satisfied on all the input states. Noise filtering (Section 6.3) may be useful in eliminating such problematic states.

The considerable worst-case computational complexity of the algorithm in the full modes is another limitation, (at least, for the current PC implementation,) practically restricting its application to relatively small-scale systems. (Note that even within a limit of a few parameters, a huge number of different systems can be considered, because of the versatility of the representation.) Still, as the examples in Chapter 9 illustrate, a sizeable class of QSI problems can be solved using the limited postulation and search modes, which reduces the time requirements significantly. As a general remark, one should point out that deep (i.e. invisible) parameters seem usually to be linked by the derivative relation to the visible ones, which explains the success of the derivative postulation mode in finding relevant models. For instance, hard-to-visualize things as acceleration and variable rate of flow are derivatives of more easily-seen things like displacement and amount of liquid in a container.

8.2. QSI as Learning

QSI's relation to well-known machine learning approaches will be examined in this section. Ways of modifying QSI so that it fits the classical inductive learning framework will be explained. Related methods will be briefly discussed.

Induction is the best known method used in machine learning. Here is a simplified definition [2] of the inductive concept learning problem: One is trying to learn a *concept*, satisfied by a particular set of *patterns*, and not satisfied by patterns outside that set. From time to time, one observes different patterns, and is told (by the "teacher") whether each pattern one observes is in the set or not. Using this (ever-increasing) knowledge, the learner is expected to infer a *general* description of the patterns satisfying the concept, which will enable him to classify a given pattern by himself. This description can be too general, that is, it may lead one to believe that a pattern which is actually not in the set is an element. In this case, it has to be *specialized* to exclude the problematic pattern and its likes. On the other hand, one may go too far in this specialization and exclude some genuine patterns which satisfy the concept

from the description; when this is noticed, again a *generalization* is necessary. So the description undergoes a kind of "diminishing oscillation" as more information keeps coming in, being generalized and specialized again and again, becoming more correct after each such update.

An obvious way of specializing a logical formula (i.e. letting it be satisfied for less cases) is to add a conjunct to it, while one can remove a conjunct, or add a disjunct, to perform generalization. (There are many other ways of doing these.)

Now consider the following interpretation of QSI as a form of inductive learning: The observed qualitative states of the system are the patterns. The description QSI is trying to learn is the system model. Each QDE is a conjunction of constraints; adding a conjunct (a new constraint) to a model specializes it, i.e. causes it to produce tighter simulations. Removing a constraint would have the opposite effect.

View QSI as starting with the assumption that all possible constraints do hold, implying a description with a great number of conjuncts. That is, QSI *overspecializes* in the beginning. Considering the input states causes the unsatisfied conjuncts to be dropped from the model, i.e. generalization. The depth test "tells" the algorithm whether it has *overgeneralized* (obtained a model that would predict unwarranted behaviors) or not. If so, the complete set of constraints involving the neighbor parameters is assumed to hold, (overspecialization again,) followed by the checking and probable elimination of the constraints against the extended states (generalization again) and a new depth test. This goes on until the algorithm terminates.

As can be seen, this is quite a special case of the general learning algorithm presented above. There are no "negative instances" (i.e. patterns that do not satisfy the concept) in QSI's input. It takes all of its input in one batch, and works and reworks it until it finds a satisfying model. A general-purpose learning "algorithm" would generally "run" for a very long time (ideally, the lifetime of the "processor" that is running it,) and would accept its input items piece by piece, with sometimes considerable periods between them. QSI's reason for requiring all distinct behaviors of the system to appear in the input is clearer now: The completeness of this information ensures the "health" of the induction, in the sense that it helps the "teacher" (the model depth test stage) to know the correct answers.

This suggests a natural way to convert QSI to an interactive algorithm, using a human (the modeler) as the teacher in the depth test stage. All that is needed is a small modification (actually, a simplification) to that stage. The idea is as follows: The model depth test stage simply simulates the proposed QDE from each initial state without counting the number of behaviors, presents each behavior that does not appear in the QSI input to the user, and asks whether this is a genuine behavior of the system or not. Behaviors identified as genuine are incorporated to the QSI input. The number of the remaining ones is used to decide for or against a new iteration. This modified algorithm, as well as fitting more nicely to the concept learning outline by having a distinct teacher, is also very suitable for the prospective modeler, since it informs the user of possibly unexpected behaviors of the system in an on-line manner, allowing him to form decisions during the modeling session. Thus, user-friendliness is gained at the cost of independence.

Allowing the user to actually specify parts of the system model before the search to QSI, which normally starts with no idea at all about the sought model, would be a simple instance of learning *by being told*, another important learning approach.

As work related to the presently explained one in this regard, the fact that inductive algorithms have been used to generate model fragments in discrete-event simulation must be mentioned. Quinlan's inductive learning algorithm[5] ID3 has been incorporated, for instance, in the package EXPERT-EASE [54], where it can produce simulation rules, in the form of nested if statements in which the conditional expressions depend on aspects of the "previous" state, and the statements in the then and else parts indicate the activities to be started in the "next" state. ID3 requires a table completely describing all the previous-next combinations as input, and is not able to handle uncertainty. In addition to the fundamental dissimilarity of their domains, ID3 and QSI are also different in the sense that the "rules" (constraints) found by the latter are much more general; they do not contain explicitly specified values in them as those of ID3, and describe conceptually more basic relations.

---

5 A similar algorithm is used to produce the decision trees for MIMIC (Section 3.2.6.)

As Forbus notes in a survey [55] of the qualitative physical reasoning field, different aspects of the problem of the machine learning of physical models [36,56] have been studied. QSI is distinguishable among these as a method which starts out with no model (instead of refining an existing one) and produces a qualitative constraint model ready for simulation, given only the qualitative behaviors of the system.

IX. QSI AT WORK

This chapter is comprised of two sections. First, several additional examples which further illustrate the working of the QSI algorithm are presented. The common small scale of the inputs is a result of the insistence that these be actually tested on the computer, and the fact that Turbo PROLOG imposes a maximum memory limit of 640 K (See Appendix A.) Also keep in mind that all the "semantical" comments about the systems and their models have been added by a human for the benefit of the readers; the actual input and output of QSI are free of that. The second section discusses some detailed features and ways of handling certain remaining difficulties with the method.

9.1. Examples

9.1.1. U-tube with Full Postulation

As shown in Chapter 6, most QSI problems, including that of the U-tube in region NORMAL, can be solved easily in the derivative postulation mode. For completeness' sake, however, an account of the algorithm's execution in the full postulation mode is presented here.

The input is again that of Tables 6.2.1 and 6.2.2. Since the postulation and search modes do not affect the initial constraint determination, the initial model is again found to be

M-(amount_A, amount_B),

and depth testing fails because of the great number of QSIM behaviors which are predicted. Now, all neighbors of the two visible parameters are postulated and their behaviors are calculated; this results in an equivalent of Table 9.1.1 being constructed by the program. In that table, the names of the new parameters (which are shown in the leftmost column) are chosen so that the reader can understand their defining constraints, for example, A' is the parameter which is defined to be amount_A's derivative. The assigned behaviors can be seen to be "sensible" enough, though (A-B) and (B-A) can raise a few thoughts, see Section 9.2 for a discussion on this.

TABLE 9.1.1. Old and postulated parameters for U-tube identification

	BEHAVIOR #1			BEHAVIOR #2		
time	t_0	(t_0,t_1)	t_1	t_0	(t_0,t_1)	t_1
A	<(0, ∞), dec>	<(0, ∞), dec>	<*disclmA*, std>	<0, inc>	<(0, ∞), inc>	<*disclmA*, std>
B	<0, inc>	<(0, ∞), inc>	<*disclmB*, std>	<(0, ∞), dec>	<(0, ∞), dec>	<*disclmB*, std>
A'	<(-∞, 0), inc>	<(-∞, 0), inc>	<0, std>	<(0, ∞), dec>	<(0, ∞), dec>	<0, std>
B'	<(0, ∞), dec>	<(0, ∞), dec>	<0, std>	<(-∞, 0), inc>	<(-∞, 0), inc>	<0, std>
-A	<(-∞, 0), inc>	<(-∞, 0), inc>	<*lm1*, std>	<0, dec>	<(-∞, 0), dec>	<*lm1*, std>
-B	<0, dec>	<(-∞, 0), dec>	<*lm2*, std>	<(-∞, 0), inc>	<(-∞, 0), inc>	<*lm2*, std>
A+B	<*lm3*, std>	<*lm3*, std>	<*lm3*, std>	<*lm3*, std>	<*lm3*, std>	<*lm3*, std>
A-B	<(0, ∞), dec>	<(0, ∞), dec>	<0, std>	<(-∞, 0), inc>	<(-∞, 0), inc>	<0, std>
B-A	<(-∞, 0), inc>	<(-∞, 0), inc>	<0, std>	<(0, ∞), dec>	<(0, ∞), dec>	<0, std>
A*B	<0, inc>	<(0, ∞), inc>	<*lm4*, std>	<0, inc>	<(0, ∞), inc>	<*lm4*, std>
A^2	<(0, ∞), dec>	<(0, ∞), dec>	<*lm5*, std>	<0, inc>	<(0, ∞), dec>	<*lm5*, std>
B^2	<0, inc>	<(0, ∞), dec>	<*lm6*, std>	<(0, ∞), dec>	<(0, ∞), dec>	<*lm6*, std>

If half search mode (which is more efficient than full search mode, especially in full postulation, where the number of old parameters is only a little fraction of the new ones,) is active, the constraints listed in Table 9.1.2 are found to hold. The constraints marked as "consequence" or "insignificant" in the table are not checked against the values of Table 9.1.1, and they are not added to the QSIM input. New parameters that appear in the remaining entries of Table 9.1.2 are made permanent. The fact that (A+B) is fixed is also marked in the QSIM invariants. Simulation with the extended model produces only the input behaviors, the dimension consistency stage adds the required buffer parameters and constraints, and QSI terminates successfully. Since this

system's model has already been discussed, let us concentrate on the additional constraints that were found in this section.

The last two ADDs, relating the products of amount_A and amount_B with their squares, are coincidental; they do not reflect any fundamental property of the U-tube, but may be the components of another system with similarly behaving quantities, as explained in Section 8.1.2. Another such coincidence has caused the four ADD constraints involving (A+B). In fact, these constraints do not make any arithmetic sense; they survive the contradiction check since there exist trivial cases where the equations they imply can be satisfied, though not with the particular values here. They happen to hold throughout the behaviors, and will not cause any negative effects (except cluttering the output and slowing simulation,) so they have been allowed to stay. By enabling the contradiction check to examine old parameter values, this sort of constraints can be totally eliminated.

TABLE 9.1.2. U-tube (NORMAL) constraints found after model extension

CONSTRAINT	REMARK
ADD(A,A',B)	
M-(A,A')	consequence
M+(B,A')	consequence
ADD(B,B',A)	
M+(A,B')	consequence
M-(B,B')	consequence
M+(B,(-A))	consequence
ADD(B,(-A),A')	consequence
M+(A,(-B))	consequence
ADD(A,(-B),B')	consequence
M+(A,(A-B))	consequence
M-(B,(A-B))	consequence
DERIV(B,(A-B))	see 9.2
ADD((A+B),(A-B),A)	
ADD(B,(A-B),(A+B))	
M+(B,(B-A))	consequence
M-(A,(B-A))	consequence
DERIV(A,(B-A))	see 9.2
ADD((A+B),(B-A),B)	
ADD(A,(B-A),(A+B))	
ADD((A-B),(B-A),A)	insignificant
ADD((A-B),(B-A),B)	insignificant
ADD(A,(-A),(A*B))	insignificant
ADD(B,(-B),(A*B))	insignificant
ADD(A,(-A),A^2)	insignificant
ADD((A*B),A^2,A)	
ADD(B,(-B),B^2)	insignificant
ADD((A*B),B^2,B)	

As illustrated here, full postulation tends to produce a big output, increasing the user's (already important) responsibility of retrieving the relevant model constraints from among the ones in it. For this reason, it is suggested that full postulation should be used in the last resort, if an initial approach using the derivative postulation mode proves to be unsuccessful.

9.1.2. Single Leaking Tank

In Chapter 6, it was pointed out that the QDEs of different operating regions of the same system can be obtained by different runs of QSI. (Actually, QSI has no distinction of two operating regions of the "same" system and two different systems.) Here is an account of a U-tube half of which has burst (or, equivalently, a bathtub whose plug has been pulled off after the bath) is identified by QSI: (Figure 5.3.2)

Assume the only initially recognizable parameter is the amount of water in the tank, and this decreases and stops at zero. (Table 9.1.3.)

TABLE 9.1.3. Input of bathtub identification

amount
<*init*, dec>
<(*0*, *init*), dec>
<*0*, std>
EQU

No constraints can be defined on a single parameter, which means the initial model is empty. Depth testing fails automatically, and the model extension stage (with derivative postulation mode) is activated, with the single parameter amount' being postulated. (See Table 9.1.4.)

TABLE 9.1.4. Old and postulated parameters for bathtub identification

amount	amount'
<*init*, dec>	<(-∞, *0*), inc>
<(*0*, *init*), dec>	<(-∞, *0*), inc>
<*0*, std>	<*0*, std>
EQU	

The constraints determined on this "bigger" behavior are again quite limited:

M-(amount,amount') with CVs *(0,0)*,
MINUS(amount,amount'),
MULT(amount',amount',amount),
DERIV(amount',amount),

and, of course, the defining constraint

DERIV(amount,amount').

Simulation produces the single input behavior, dimension consistency requires the MINUS constraint to be dropped and buffers to be created, so the output is as shown in Table 9.1.5.

TABLE 9.1.5. Output of bathtub identification

CONSTRAINT	CVs
M+(amount,P1)	*(0,0)*,(∞,∞),(-∞,-∞)
DERIV(P1,P2)	
M+(P2,P3)	*(0,0)*,(∞,∞),(-∞,-∞)
M+(amount,P4)	*(0,0)*,(∞,∞),(-∞,-∞)
DERIV(P3,P4)	
M-(amount,P2)	*(0,0)*
M+(P2,P5)	*(0,0)*,(∞,∞),(-∞,-∞)
M+(amount,P6)	*(0,0)*,(∞,∞),(-∞,-∞)
MULT(P5,P5,P6)	

As for an interpretation, the square relation is again a coincidental one. The second DERIV is also coincidental. The remaining relationships correctly describe a leaking tank with no inward flow. The rate of *increase* of the amount (its derivative) is inversely proportional to it, and the flow ceases when the amount vanishes. The deep parameter "flow" has been hinted at by the algorithm. The derivative postulation mode's utility has once again been demonstrated; the following examples will further underline this.

9.1.3. Bathtub with Constant Inflow

Now suppose that a tap exists on top of the bathtub (as is often the case,) and water pours out of it at a constant rate. Assuming that the constant inflow is recognizable as a parameter (although this is not necessary,) the input

behavior (Table 9.1.6) will be one in which the amount in the tub (starting from zero) arrives at equilibrium at some positive landmark. (Again, the possibility of overflow has been overlooked; see discussion in Section 6.2.3.) No constraints are found by the initial determination, and model extension is required. Since inFlow is constant, its derivative is not postulated, and extended constraint determination will be performed on the behavior shown in Table 9.1.7.

TABLE 9.1.6. Input for filling bathtub identification

inFlow		amount
<*inF*, std>		<*0*, inc>
<*inF*, std>		<(*0*, ∞), inc>
<*inF*, std>		<*disclmA*, std>
	EQU	

TABLE 9.1.7. Input of second constraint determination in filling bathtub identification

inFlow	amount	amount'
<*inF*, std>	<*0*, inc>	<(*0*, ∞), dec>
<*inF*, std>	<(*0*, ∞), inc>	<(*0*, ∞), dec>
<*inF*, std>	<*disclmA*, std>	<*0*, std>
	EQU	

The QDE found (and accepted by the simulation) can be seen in Table 9.1.8. The dimension consistency stage results in the final model of Table 9.1.9.

TABLE 9.1.8. Sufficient QDE for filling bathtub identification

CONSTRAINT

M-(amount,amount')
ADD(amount,amount',inFlow)
DERIV(amount,amount')

The ADD in the final model represents the fundamental relation Net Flow = I n Flow - Out Flow. As well as the "discovered" parameter outFlow, the direct proportionality between it and the amount is again established.

TABLE 9.1.9. Output of filling bathtub identification

CONSTRAINT	CVs
DERIV(amount,P1)	
M-(amount,P1)	
M+(amount,P2)	(0,0),(∞,∞),(-∞,-∞)
M+(P1,P3)	(0,0),(∞,∞),(-∞,-∞)
M+(inFlow,P4)	(0,0),(∞,∞),(-∞,-∞)
ADD(P2,P3,P4)	

9.1.4. Water Balance in Kidney

Consider the situation in which only the two (supposedly) most easily observable parameters of Table 8.1.3, namely, the amount of water in plasma, and the net rate of water excretion, are used to form the input behaviors. One can say right away that QSI simply can not be expected to find the complicated model of Table 8.1.1 from these data; the input is inadequate to uncover the features of the complete system. Still, it will be demonstrated that the algorithm comes up with a model that does predict only the specified behaviors (which is about all what a human novice can do "at first sight,") and gives some hints about possible deep parameters.

Of the two input behaviors, one (Table 9.1.10) has been obtained by observing the "normal" state of things for some time. Both parameters are at their equilibrium values.

TABLE 9.1.10. First input behavior for water balance identification

amt(water,P)	net flow(water, P->U)
<*A**, std>	<*NF*, std>
EQU	

The second behavior (Table 9.1.11) is a result of observing what happens when the amount is rapidly increased by an outside intervention, i.e. a sudden large drink. This has caused an immediate increase in the excretion rate, with both quantities gradually returning to their normal values.

Initial constraint determination results in the model of Table 9.1.12, but, as expected, simulation shows that this model is not deep enough. Again having

selected the derivative postulation mode, two new parameters with defining constraints

DERIV(amt(water,P), P1)

and

DERIV(netflow(water, P->U), P2)

are created, both starting at negative and increasing, and settling at zero as the same instant when their defining neighbors arrive equilibrium. Extended constraint determination adds the constraints of Table 9.1.13 to the QDE. (Various consequences of the Ms which are found but not included in the simulation model, and which can be also useful for interpretation, are not presented here because of space considerations.) This model predicts only the two inputs, and is accepted. For the conditions presented in the input, a "robot physician" can use this as the basis of a model of the human water balance system. For a human physician, it would at least provide some pointers to start with in an attempt to form a deeper model. (Not for this particular system, of course; its model is already known.) For example, the existence of a parameter which is the derivative of amt(water,P) is implied. There really is such a parameter in Table 8.1.1. Furthermore, that quantity really is inversely proportional to netflow(water, P -> U) if netflow(water, ingest->P) is constant, which is the case in the table. The direct proportionality of amt(water,P) and netflow(water, P->U) also follows from Table 8.1.1. All the discovered MULTs are coincidental. For a more specific identification, more parameters, more behaviors, the full postulation and search modes, and a user with some idea of what to expect in the model would be required.

TABLE 9.1.11. Second input behavior for water balance identification

amt(water,P)	net flow(water, P->U)
<(*A**, ∞), dec>	<(*NF*, ∞), dec>
<(*A**, ∞), dec>	<(*NF*, ∞), dec>
<*A**, std>	<*NF*, std>
EQU	

TABLE 9.1.12. Initial model in water balance identification

CONSTRAINT

M+(amt(water,P), netflow(water, P->U))
MULT(amt(water,P), amt(water,P), netflow(water, P->U))
MULT(netflow(water, P->U), netflow(water, P->U), amt(water,P))

TABLE 9.1.13. Constraints added by second constraint determination

CONSTRAINT

DERIV(amt(water,P),P2)
DERIV(netflow(water, P->U),P1)
ADD(amt(water,P),P1,netflow(water, P->U))
ADD(amt(water,P),P2,netflow(water, P->U))
ADD(P1,amt(water,P),netflow(water, P->U))
ADD(P2,amt(water,P),netflow(water, P->U))
M-(netflow(water, P->U),P1)
M-(netflow(water, P->U),P2)
MULT(netflow(water, P->U),P1,P2)
MULT(netflow(water, P->U),P2,P1)
MULT(amt(water,P),P1,P2)
MULT(amt(water,P),P2,P1)

9.1.5. Heat Exchanger

The heat exchanger system (Figure 9.1.1) to be used in this example is from [16]. There is cold water in the bath shown as the box in the figure. Hot liquid enters from one end of the pipe and leaves, cooler because of the heat flow, from the other end. There are three different behaviors, determined by whether the heat flow stops when the unit volume of liquid that we are interested in is in the pipe, and if so, where. Supposing that the parameters in the input are X, (position of liquid in the pipe; the entry end is the negative landmark x^* and the exit end is 0,) Q, (surplus heat of liquid; 0 when thermal equilibrium is reached, a positive value at the start,) and F, (the heat flow in the liquid,) the algorithm starts with the three behaviors in Table 9.1.14. thru 9.1.16.

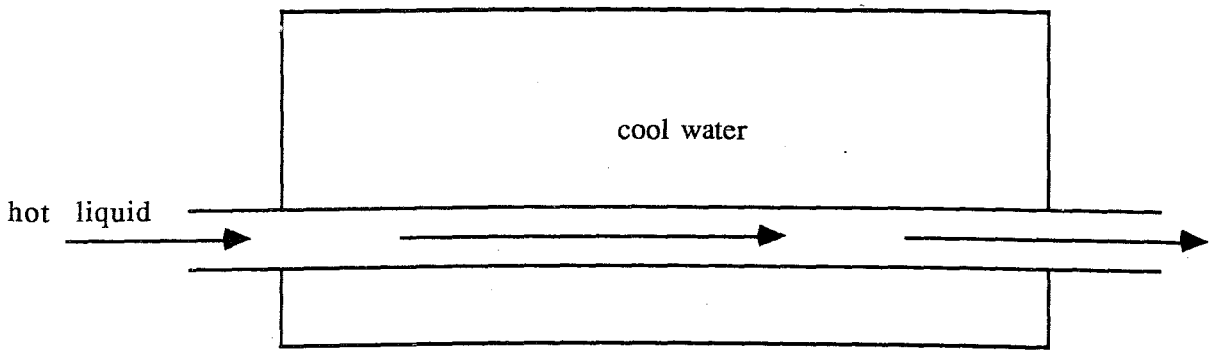


Figure 9.1.1. The heat exchanger

TABLE 9.1.14. Input behavior #1 for heat exchanger identification

X	Q	F
<x^*, inc>	<q^*, dec>	<f^*, inc>
<(x^*, 0), inc>	<(0, q^*), dec>	<(f^*, 0), inc>
<0, inc>	<0, std>	<0, std>

TABLE 9.1.15. Input behavior #2 for heat exchanger identification

X	Q	F
<x^*, inc>	<q^*, dec>	<f^*, inc>
<(x^*, 0), inc>	<(0, q^*), dec>	<(f^*, 0), inc>
<0, inc>	<(0, q^*), dec>	<(f^*, 0), inc>

TABLE 9.1.16. Input behavior #3 for heat exchanger identification

X	Q	F
<x^*, inc>	<q^*, dec>	<f^*, inc>
<(x^*, 0), inc>	<(0, q^*), dec>	<(f^*, 0), inc>
<(x^*, 0), inc>	<0, std>	<0, std>
<(x^*, 0), inc>	<0, std>	<0, std>
<0, inc>	<0, std>	<0, std>

Initial constraint determination comes up with the model of Table 9.1.17, which really covers the heat flow relationships; the first constraint is the definition of flow, whereas the fourth one has the equation

$$F = -KQ$$

(where -K is the thermal conductivity) "embedded" in it. (Even in the case where only the longest behavior is entered, those relationships are still discovered; the figures in Table 7.1.1 reflect that situation.) The other constraints are coincidental. Note, however, that DERIV(F,Q) is what one would expect to be discovered if those two names were swapped in the input, i.e. if the liquid in the pipe was warming instead of cooling.

Depth testing on the above-mentioned constraint set fails automatically without simulation, because X does not appear in any of the constraints. In the model extension stage, the derivative of only X will be postulated, since the derivatives of both Q and F are already there. The heuristics lead to a fixed

positive value for that parameter to be determined. The resulting model's simulation is satisfactory, and QSI terminates after the dimension consistency stage. A fixed value for the derivative of X is sensible, since it simply means that the speed of the liquid in the pipe is constant.

TABLE 9.1.17. Initial constraints for heat exchanger identification

CONSTRAINT	CVs
DERIV(Q,F)	
DERIV(F,Q)	
MINUS(F,Q)	
M-(F,Q)	*(0,0)*
MULT(F,F,Q)	

A justifiable remark about this problem is that instead of the heat and its flow, the *temperatures* of the liquids would be more appropriate as shallow parameters. See Section 9.2 for a discussion.

9.1.6. The Upward Thrown Ball

No discussion about a new qualitative reasoner would be complete without a version of the upward thrown ball problem; this tradition was observed in the descriptions of improved QSIM and the postdiction algorithm. This section will be concluded by an account of the execution of QSI when fed a single input behavior (Table 9.1.18) describing the height of a ball which rises for a while and then falls back.

TABLE 9.1.18. Behavior of ball height

Y
<(*0*, ∞), inc>
<(*0*, ∞), inc>
<*disclmY*, std>
<(*0*, *disclmY*), dec>
<*0*, dec>

After an empty initial model, derivative postulation will result in the extended behavior of Table 9.1.19, but the single DERIV linking the two parameters is not sufficient for a tight simulation, and model extension has to

be performed for a second time. The derivative of Y' is decided to be fixed at a negative value and permanently added to the model, which passes the test. From a single account of the height of a thrown object, the "laws" governing such bodies have been identified. Two deep parameters representing the velocity and acceleration have been correctly suggested. Consult Table 7.1.1 for the execution times of the examples of this section.

TABLE 9.1.19. Input to second constraint determination in ball system identification

Y	Y'
<(*0*, ∞), inc>	<(*0*, ∞), dec>
<(*0*, ∞), inc>	<(*0*, ∞), dec>
<*disclmY*, std>	<*0*, dec>
<(*0*, *disclmY*), dec>	<(-∞, *0*), dec>
<*0*, dec>	<(-∞, *0*), dec>

9.2. Further Issues

The behavior calculation procedure, invoked in the model extension stage to assign one of possibly infinitely many alternative behaviors to each postulated parameter, is the only "heuristic" part of the algorithm (as explained in Chapter 6.) The behaviors it comes up with are guaranteed to be mathematically plausible with regard to their neighbors specified in the input, but the extent to which they match the corresponding deep parameters in the semantical interpretation that we give to the system is dependent on the "rules of thumb" of behavior selection embedded into QSI. This means that there is always room for improvement in that procedure, and possible new heuristics and representation schemes may increase the number of problems that can be solved satisfactorily by the algorithm. Examples of some such issues about behavior selection will be presented in this section.

Consider the heat exchanger of Section 9.1.5 again. This time, the more realistic assumption that the initial parameters are X (meaning the same as before,) T_out (the fixed temperature of the cool water bath,) and T_in (the

temperature of the unit volume of liquid that one is tracking in the pipe,) will be made. The input behaviors are shown in Tables 9.2.1 thru 9.2.3.

TABLE 9.2.1. Input #1 for heat exchanger identification (temperature version)

X	T_in	T_out
<x^*, inc>	<T_{begin}, dec>	<T_{cool}, std>
<(x^*, 0), inc>	<(0, T_{begin}), dec>	<T_{cool}, std>
<0, inc>	<*disclmT*, std>	<T_{cool}, std>

TABLE 9.2.2. Input #2 for heat exchanger identification (temperature version)

X	T_in	T_out
<x^*, inc>	<T_{begin}, dec>	<T_{cool}, std>
<(x^*, 0), inc>	<(0, T_{begin}), dec>	<T_{cool}, std>
<0, inc>	<(0, T_{begin}), dec>	<T_{cool}, std>

TABLE 9.2.3. Input #3 for heat exchanger identification (temperature version)

X	T_in	T_out
<x^*, inc>	<T_{begin}, dec>	<T_{cool}, std>
<(x^*, 0), inc>	<(0, T_{begin}), dec>	<T_{cool}, std>
<(x^*, 0), inc>	<*disclmT*, std>	<T_{cool}, std>
<(x^*, 0), inc>	<*disclmT*, std>	<T_{cool}, std>
<0, inc>	<*disclmT*, std>	<T_{cool}, std>

By using our knowledge of the domain, we can "cheat" and write down the constraints that QSI is "supposed" to find if it is to identify the system as we interpret it. The rate of increase of T_in will have the sign of (T_out - T_in); i.e. we expect QSI to find the relations DERIV(T_in,P) and ADD(T_in,P,T_out) along with the already seen one about the derivative of X. The trouble is, three alternative behaviors for the derivative of T_in in Table 9.2.2 exist, among which the heuristics cannot make a preference: 1) P negative and fixed, 2) P negative and increasing, and 3) P negative and decreasing. Of these, one will

be chosen randomly. (2) is the one which fits our understanding of the system, and the expected model will be found if it is selected, otherwise, it will not.

This illustrates the motivation for the ongoing research for better heuristics. A way of representing the state tree's structure in the input could also provide a solution, since it would cause values in several behaviors to be identified as a single one, imposing an additional restriction which would probably reduce the number of alternatives. Next, some issues that can be resolved by the use of a more flexible representation will be briefly considered.

In the U-tube identification of Section 9.1.1, the difference parameters (A-B) and (B-A) are assigned behaviors in which they both have the value zero at equilibrium. This is a possibility, but two other possibilities, corresponding to the cases where either amount_A or amount_B is the greater of the two, also exist. Since the behaviors representing these possibilities have more than three values in them, they are not even considered by the algorithm. It must also be mentioned that QSI takes special care in the behavior calculation of new parameters which are negatives or reciprocals of each other, like P1 and P2 defined by ADD(X,P1,Y) and ADD(Y,P2,X) or P3 and P4 defined by MULT(A,P3,B) and MULT(B,P4,A), so that no inconsistency is allowed between the new behaviors.

Finally, consider the subsystem of Table 9.2.4. The derivative of X is to be postulated by QSI.

TABLE 9.2.4. Behavior of X-Y subsystem

X	Y
<*neglm*, std>	<(*0*, ∞), inc>
<(*neglm*, *0*), inc>	<(*0*, ∞), inc>
<*0*, inc>	<(*0*, ∞), inc>
<(*0*, *poslm*), inc>	<(*0*, ∞), inc>
<*poslm*, inc>	<*Ylm*, std>
<(*poslm*, *maxlm*), inc>	<(*0*, *Ylm*), dec>
<*maxlm*, std>	<(*0*, *Ylm*), dec>

The new parameter's magnitude clearly has to be zero at the endpoints of the behavior, and positive within it. Once again, there is more than one choice, even when one restricts attention to behaviors with seven values and applies the heuristics (Table 9.2.5.)

TABLE 9.2.5. Two choices for the behavior of X'

X	choice #1 for X'	choice #2 for X'
<*neglm*, std>	<*0*, inc>	<*0*, inc>
<(*neglm*, *0*), inc>	<(*0*, ∞), inc>	<(*0*, ∞), inc>
<*0*, inc>	<*lm1*, std>	<(*0*, ∞), inc>
<(*0*, *poslm*), inc>	<(*0*, *lm1*), dec>	<(*0*, ∞), inc>
<*poslm*, inc>	<(*0*, *lm1*), dec>	<*lm1*, std>
<(*poslm*, *maxlm*), inc>	<(*0*, *lm1*), dec>	<(*0*, *lm1*), dec>
<*maxlm*, std>	<*0*, dec>	<*0*, dec>

Furthermore, there is no good reason that the derivative should arrive at its landmark just when another parameter is crossing one of its own landmarks, and the "real" description may well be one with a greater number of qualitative states. All these different possibilities would cause different constraints involving X' to be found, or not to be found. For example, M+(X',Y) is found only if choice #2 is selected for X. To deal with such situations so that the chances of constraint discovery are maximized, a more flexible representation scheme for the directions of postulated derivatives has been designed. The idea is to defer the decision on when the new parameter actually stops at its landmark until a constraint involving it is found in the constraint determination phase. Till then, the postulated parameter's direction in the period between its two zeros is set to either **isd** (increasing-steady-decreasing) or **dsi** (decreasing-steady-increasing,) depending on its sign. After the discovery of the first significant constraint involving the parameter, its behavior is translated into the conventional format. In the example, the situation at the end of the postulation stage will be as shown in Table 9.2.6.

TABLE 9.2.6. Extended behavior of subsystem

X	Y	X'
<*neglm*, std>	<(*0*, ∞), inc>	<*0*, inc>
<(*neglm*, *0*), inc>	<(*0*, ∞), inc>	<(*0*, ∞), isd>
<*0*, inc>	<(*0*, ∞), inc>	<(*0*, ∞), isd>
<(*0*, *poslm*), inc>	<(*0*, ∞), inc>	<(*0*, ∞), isd>
<*poslm*, inc>	<*Ylm*, std>	<(*0*, ∞), isd>
<(*poslm*, *maxlm*), inc>	<(*0*, *Ylm*), dec>	<(*0*, ∞), isd>
<*maxlm*, std>	<(*0*, *Ylm*), dec>	<*0*, dec>

Consistency checks involving **isd** or **dsi** are automatically satisfied. When M+(X',Y) passes the test in this manner, the directions of X' are updated

with **inc**, **std** or **dec** so that it really satisfies the M+; this amounts to the choice #2 in Table 9.2.5 being finally selected.

X. CONCLUSION

In this dissertation, three separate contributions in the AI area of qualitative reasoning about physical systems were presented. The reported work has focused on the representation and algorithm of the qualitative simulation program QSIM. The ideas exemplified in that context can also be applied in the wider qualitative simulation and modeling scene.

10.1. Summary of Results

The first of the contributions is about the use of tuples of corresponding values of system parameters, a common technique in qualitative simulation. Corresponding value information about parameter magnitudes is used by the algorithm during the consistency filtering of newly proposed states, in addition to the information supplied by the constraints themselves. The existence of a class of inputs for which the QSIM algorithm predicts mathematically impossible behaviors, although the information required to detect and eliminate the inconsistency already exists in the state sequence produced during the simulation, was demonstrated. It was shown that the cause of this problem is the current practice of allowing only point magnitudes as corresponding values. The notion of interval corresponding values was introduced as a solution, and instructions on how to incorporate this to the QSIM algorithm and the modification of the qualitative arithmetic routines to handle operations where both operands are interval values were given. The resulting algorithm, named improved QSIM, was proven to be better than pure QSIM, in the sense that: 1) They both find all the correct behaviors, 2) Improved QSIM does not predict some spurious behaviors that QSIM predicts, and 3) Improved QSIM does not predict any spurious behaviors that QSIM does

not predict. The utility of improved QSIM was shown in example problems. As stated in the analysis and the reports on the case runs, the modification does not change the overall time complexity of the algorithm. A feature of the new technique which distinguishes it from other ways of spurious behavior reduction is that it needs no extra input information and just "squeezes" more knowledge from the available input to achieve better results. Because of the nature of the changes, pure QSIM can be replaced by the improved version with minimum effort.

Another item of this research is the development of the postdiction algorithm for systems of continuous-valued parameters. Again, QSIM was used as the basis. Various issues exist about this reasoning task in the general case: If one is informed about the current scene, and asked to find out what may have happened in the past so that this result has been obtained, there is usually a formidably large number of possibilities for the situation in the "previous" state, only a fraction of which may have been known. Furthermore, since causes of "events" can in general be traced back to the beginning of the universe, this very large "branching" of possible previous states will occur for each node of a very big tree. Choosing among the alternative pasts and deciding when to stop looking for even earlier events are typical issues. These are resolved naturally for the domain of continuous systems by the adoption of the QSIM representation. By modifying only the value transition and operating region change modules, the already analyzed correctness and complexity properties of the algorithm were ensured to remain. The postdiction algorithm produces a tree of states where each path from each node to the root is a possible past. This different interpretation of the output stems from the fact that one has no way of knowing whether a particular node in the state tree was the initial state or not. The Closed World Assumption was discussed in this context. The algorithm inherits the soundness of QSIM; it does not miss any possible pasts that the model implies. For diagnosis applications, this exhaustive listing could be useful. Example runs were used to illustrate the postdiction algorithm's working.

Finally, the qualitative system identification algorithm was presented. QSI is able to propose qualitative constraint models for systems whose qualitative behavior it takes as input. The fact that it uses the QSIM format makes the incorporation of the two algorithms in a unified reasoner very easy. QSI's work corresponds to the model structure determination phase of the

general system identification enterprise. The algorithm's structure was explained and exemplified in detail. Starting with (as many as possible of) the qualitatively distinct behaviors and associated quantity spaces of the system's parameters, the constraint determination stage finds all the constraints that hold on the parameters by generating all syntactically possible constraints and eliminating the ones that do not hold. The model obtained in this manner is fed to the depth test stage where QSIM itself is used to check it not for correctness, but for adequacy. If simulation of the model yields an unacceptably large number of behaviors which are not present in the input, it is passed to the model extension stage to be further tightened with the addition of more parameters and constraints. "Useful" system parameters that do not appear in the input are searched by postulating new parameters linked to the old ones by various types of constraints, calculating their behaviors, and performing constraint determination anew on the resulting extended system behaviors. During the behavior calculation for the postulated parameters, use of heuristics is made to choose the "likeliest" of the alternative behaviors. This means that the model extension stage may result in the production of models, which, although mathematically consistent with the raw input information, are different from the actual models that could be built given context information about the system and the "natures" of the parameters. Unit consistency rules are imposed on the final model that has passed the depth test successfully; "buffer" parameters and constraints are created to meet the arithmetic requirements about the parameter dimensions. QSI requires its input to be completely correct (albeit qualitative.) A single wrong qualitative value in the input can hinder the discovery of a correct model. A noise filter that will be used as an optional preprocessor to smooth out suspected fluctuations in individual parameter behaviors has been designed.

As the analysis shows, QSI's time complexity is similar to those of other qualitative reasoners (which are, unfortunately, not very fast.) The fact that each QSI run has (possibly multiple) QSIM invocations in it makes this obvious. Execution times of the current implementation for various inputs were listed in Table 7.1.1.

Proofs of several properties of the algorithm were presented. The constraint determination procedure finds all, and only, the constraints valid in its input. All models produced by the algorithm (even the ones which are not deep enough) therefore correctly describe the system's quantities, and, when

simulated, are guaranteed to predict each behavior in the input. The "extended" models are never worse, and usually better, than their predecessors, by QSI's criterion of model goodness. The consequence detection rules used to skip testing many constraints were established.

The preparation of QSI's input (with special emphasis on the lack of various important kinds of information in it,) and ways of interpreting its output, were discussed in detail. QSI's applicability to diagnosis tasks was examined, as well as its evident utility in qualitative modeling. The algorithm's approach was compared and contrasted to other (more "human") methods of modeling. Its place in the general framework of machine learning was determined.

Several case runs were used to help illustrate the algorithm's working.

In addition to naturally filling the system identification gap in the existing body of qualitative reasoning research, QSI may form a part of the basis of a much more involved modeling and reasoning procedure to be invoked by autonomous intelligent robots in the future.

Those interested may obtain (possibly later versions of) the PROLOG source codes of the programs embodying the above-mentioned algorithms in magnetic media from the author for research purposes.

10.2. Suggestions for Future Work

The research reported in this text has suggested some areas of future work. Ongoing experimentation on more and more example problems may result in new behavior selection heuristics for QSI being found. As already discussed, in its present form, QSI is very "mechanical," arriving at the models by what one may call a brute-force search. This is the price paid for not relying on the existence of any input information except accounts of the changing of some parameter values. While this research has demonstrated that model structure identification at a quite impressive scale is possible even with this much data, a real-world reasoner must certainly be able to make use of any additional

knowledge that is available. In this regard, two possible roads for development are evident. The first alternative is to use the "pure" QSI algorithm, as presented here, as part of a much bigger modeling program, which combines the many approaches to this difficult mental task. QP theory and the method of dimension analysis would be among the methods incorporated, and hints about the nature of the system under consideration, units of its parameters, and model fragments that are known to be definitely there, would be some of the additional input items. QSI's role in this setup could be to prepare initial model proposals, from which the later stages prune off the coincidental and noninteresting constraints. This super-modeler need not even restrict itself to systems of continuous parameters; by spanning several domains, its usefulness would increase. Research into what humans actually "do" during the model-building task would certainly provide valuable pointers for the construction of such a program.

The second, more modest alternative is to modify QSI so that it can use some of the above-mentioned information itself. A knowledge base of "typical" features of various model structures may be constructed. This knowledge would then be used to help QSI to make more intelligent decisions, maybe forgoing testing for some unlikely constraints, and therefore produce sleeker outputs, more efficiently.

Our approach to the problem of deciding what constitutes an "easily observable" parameter and what does not has been quite intuitive. Research about this topic will certainly be useful for further work on QSI.

Attempting to apply the postdiction algorithm to perform diagnosis also raises some issues to be handled by future work. In addition to the final state of the system, one also usually has some (at least partial) knowledge about the starting state during diagnosis. Various other kinds of additional information may be available, which could be used to discredit or totally eliminate many of the alternative possible pasts. For instance, known probabilities of occurrence of various kinds of faults may be utilized. If the algorithm is augmented to handle available quantitative knowledge (like in Section 3.2.3,) even MTBF data could help in imposing a likeliness order on the possible pasts.

For some purposes, the exhaustive listing of possible pasts by the postdiction algorithm, presented in Chapter 5 as a desirable feature, may be too cumbersome. A postprocessor which collapses several possible pasts which are

equivalent on a specified scale to a single one, therefore summarizing the output in higher-level terms, can be written. For the burst tank postdiction in Chapter 5, for example, all the pasts involving an explosion would be summarized in a single one.

The construction of a unified temporal reasoner, which is a combination of not only a temporal database and a simulator, but also a modeler, is another possibility. The storage and retrieval of time-indexed information, prediction and postdiction from known or "what-if" facts in the database, and inference of laws that hold in the domain from these facts, all in one program, can be achieved by such a reasoner.

The quantity space representation is not as impoverished as it seems at first sight. One *can* represent such information as $b = 2a$ for landmarks a and b in the same quantity space, using only pure QSIM's format, for instance. An investigation of how much of such "quantitative" information can be squeezed into the standard qualitative representation, and the power that such exploitations of algebra and arithmetic can impart to the various reasoners, is among the future work planned.

Computer implementations (Appendix A) and experimentations have been accompanying the development of the reported algorithms. Their porting to computers with greater space and speed is an immediate practical aim.

APPENDIX A. IMPLEMENTATION

For hardware availability reasons, the programs embodying the algorithms described in the text were implemented on IBM-compatible PCs. The "European" AI programming language PROLOG (as opposed to LISP, which is popular among American researchers) was used. Borland's Turbo PROLOG Version 2.0 [57] provides a very acceptable program development environment, however, two things must be mentioned in this regard. First, Turbo PROLOG differs in many respects from the "standard" version in [58], and, although this did not cause serious difficulties during implementation, it may have obvious negative effects in future work to port the programs to other environments. Secondly, Turbo PROLOG imposes a very low (for this kind of programs, at least) limit on the memory that can be used (640 K) by the programs. This is the cause of the small size of the problems used for the QSI case runs reported in Chapter 9. Larger problems were tested by "chopping" them up in various not-so-elegant ways, so that the program stayed within the memory limit.

Each of the two programs developed in relation to this research is a Turbo PROLOG *project*: a collection of many files. The first such project, named IQSIMP, is a "unified" program, which has both the postdiction ability of Chapter 5, and the interval CV recording and using features of Chapter 4. To ensure flexibility, the "direction of time" in which the simulation will be performed is specified by the user in the input file. If "forward" is specified, "normal" QSIM (but with interval CVs) runs. If "backward" is specified, postdiction is performed.

The guidelines observed for the PROLOG implementation of QSIM were explained in Chapter 3. The "nature" of the language makes a depth-first approach to the creation of the state tree (Section 3.1.2, Step 1) the easiest to write, so we followed this route.

The second project, QSI, is an implementation of the algorithm as described in Section 6.2. As already stated, QSI calls a slightly modified version

of QSIM as a subroutine, so it is not surprising to see that parts the QSI project resemble those of IQSIMP very closely. Actually, the version of QSIM that appears in the source code of QSI which we make available is the "un-improved" one, so that one can see the differences required by the ICV features by comparing the CV recording and qualitative arithmetic predicates in these two programs.

Both programs invoke the Turbo PROLOG editor to aid the user in the preparation of the input file and the examination of the output file. On-line help during the input's preparation is also available.

As programmers know, large programs are like "living" beings; they undergo many changes during their lifetimes. The programs described here will also be the objects of various modifications, both to improve the space efficiency, and to better explore many new ideas, some of which have been mentioned in the text. (For instance, the proposals of Section 9.2.) All the proper examples in this text, for which execution times were presented, and many more, have been run with these programs, and the reported results have been obtained as output.

The (quite long) source codes and help files can be obtained from the author at the BITNET electronic mail address SAY@TRBOUN.

BIBLIOGRAPHY

1. L. E. Widman, K. A. Loparo and N. R. Nielsen (Eds.) *Artificial Intelligence, Simulation and Modeling*. New York: John Wiley and Sons, 1989.

2. E. Charniak and D. McDermott, *Introduction to Artificial Intelligence.* Reading, MA: Addison-Wesley, 1985.

3. J. R. Hobbs and R. C. Moore (Eds.) *Formal Theories of the Commonsense World.* Norwood, NJ: Ablex, 1985.

4. J. McCarthy and P. J. Hayes, "Some philosophical considerations from the standpoint of artificial intelligence," B. Meltzer and D. Michie (Eds.), *Machine Intelligence* Vol. 4, Edinburgh, Scotland: Edinburgh University Press, 1969.

5. P. J. Hayes, "The second naive physics manifesto," J. R. Hobbs and R. C. Moore (Eds.) *Formal Theories of the Commonsense World.* Norwood, NJ: Ablex, 1985.

6. D. S. Weld and J. de Kleer (Eds.) *Readings in Qualitative Reasoning About Physical Systems.* Los Altos, CA: Morgan Kaufmann, 1990.

7. J. de Kleer, "How circuits work," *Artificial Intelligence,* Vol. 24, pp. 205-280, 1984.

8. B. Kuipers, "Qualitative simulation as causal explanation," *IEEE Transactions on Systems, Man, and Cybernetics* , Vol. 17, pp. 432-444, 1987.

9. B. Kuipers, "Qualitative reasoning with causal models in diagnosis of complex systems," L. E. Widman, K. A. Loparo and N. R. Nielsen (Eds.) *Artificial Intelligence, Simulation and Modeling*. New York: John Wiley and Sons, 1989.

10. J. Shrager, D. S. Jordan, T. P. Moran, G. Kiczales and D. M. Russell, "Issues in the pragmatics of knowledge modeling: Lessons learned from a xerographics project," *Communications of the ACM,* Vol. 30 pp. 1036-1047, 1987..

11. J. de Kleer and J. S. Brown, "A qualitative physics based on confluences," *Artificial Intelligence* , Vol. 24, pp. 7-83, 1984.

12. K. D. Forbus, "Qualitative process theory," *Artificial Intelligence,* Vol. 24 pp. 85-168, 1984.

13. B. C. Williams, "Qualitative analysis of MOS circuits," *Artificial Intelligence* , Vol. 24, pp. 281-346, 1984.

14. B. Kuipers, "Qualitative simulation," *Artificial Intelligence* , Vol. 29, pp. 289-338, 1986.

15. B. Kuipers, "Qualitative reasoning: Modeling and simulation with incomplete knowledge," *Automatica* , Vol. 25, pp. 571-585, 1989.

16. D. S. Weld, "Comparative analysis," *Artificial Intelligence* , Vol. 36, pp. 333-373, 1988.

17. D. S. Weld, "Exaggeration," *Artificial Intelligence* , Vol. 43, pp. 311-368, 1990.

18. B. Kuipers, C. Chiu, D. T. Dalle Molle and D. Throop. *Higher-order derivative constraints in qualitative simulation*. Artificial Intelligence Laboratory, The University of Texas at Austin, AI89-117, 1989.

19. W. W. Lee and B. J. Kuipers, "Non-intersection of trajectories in qualitative phase space: A global constraint for qualitative simulation," *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88).*

20. J. McCarthy, "Programs with common sense," R. J. Brachman and H. J. Levesque (Eds.) *Readings in Knowledge Representation*. Los Altos, CA: Morgan Kaufmann, 1985.

21. J. McCarthy, "Generality in artificial intelligence," *Communications of the ACM*, Vol. 30, pp. 1030-1035, 1987.

22. L. A. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Transactions on Systems, Man, and Cybernetics* , Vol. 3, pp. 28-44, 1973.

23. J. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM,* Vol. 26, pp. 832-843, 1983.

24. J. F. Allen and H. A. Kautz, "A model of naive temporal reasoning," J. R. Hobbs and R. C. Moore (Eds.) *Formal Theories of the Commonsense World.* Norwood, NJ: Ablex, 1985.

25. A. Galton, "A critical examination of Allen's theory of action and time," *Artificial Intelligence,* Vol. 42, pp. 159-188, 1990.

26. J. Crawford, A. Farquhar and B. Kuipers, "QPC: A compiler from physical models into qualitative differential equations," *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-90),* 1990.

27. E. Tın, "Nedensel formal sistemlerle çıkarım," *Bilkent Üniversitesi Elektrik-Elektronik ve Bilgisayar Mühendisliği Konferansı*, Ankara, 1991.

28. Y. Iwasaki and H. A. Simon, "Causality in device behavior," *Artificial Intelligence*, Vol. 29, pp. 3-32, 1986.

29. Y. Iwasaki and H. A. Simon, "Theories of causal ordering: Reply to de Kleer and Brown," *Artificial Intelligence* , Vol. 29, pp. 63-72, 1986.

30. B. Kuipers, "Commonsense reasoning about causality: Deriving behavior from structure," *Artificial Intelligence* , Vol. 24, pp. 169-203, 1984.

31. D. S. Weld, "The use of aggregation in causal simulation," *Artificial Intelligence*, Vol. 30, pp. 1-34, 1986.

32. Williams, "Doing time: Putting qualitative reasoning on firmer ground", D. S. Weld and J. de Kleer (Eds.) *Readings in Qualitative Reasoning About Physical Systems.* Los Altos, CA: Morgan Kaufmann, 1990.

33. B. Faltings, "Qualitative kinematics in mechanisms," *Artificial Intelligence* , Vol. 44, pp. 89-119, 1990.

34. D. R. Throop, *Spatial unification: Qualitative spatial reasoning about steady state mechanisms*. Artificial Intelligence Laboratory, The University of Texas at Austin, AI89-95, 1989.

35. D. Dvorak and B. Kuipers, "Model-based monitoring of dynamic systems," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, 1989.

36. N. Lavrac and I. Mozetic, "Methods for knowledge acquisition and refinement in second generation expert systems," *SIGART Newsletter* 108 (1989) 63-69.

37. E. Sacks, "A dynamics systems perspective on qualitative simulation," *Artificial Intelligence* , Vol. 42, pp. 349-362, 1990.

38. E. Sacks, "A dynamics perspective on comparative analysis," *IEEE Expert*, pp. 67-69, February 1991.

39. D. S. Weld, "Comparative analysis and qualitative reasoning," *IEEE Expert*, pp. 70-72, February 1991.

40. J. Suzuki, N. Sueda, Y. Gotoh and A. Kamiya, "Plant control expert system coping with unforeseen events - Model-based reasoning using fuzzy qualitative reasoning," *Proc. 3rd Int'l. Conf. Industrial and Engineering Applications of AI and Expert Systems*, July 15-18, 1990, Columbia, S. Carolina.

41. B. Kuipers, "Abstraction by time-scale in qualitative simulation," *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, 1987.

42. B. Kuipers and C. Chiu, "Taming intractable branching in qualitative simulation," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, 1987.

43. B. Kuipers and D. Berleant, "Using incomplete quantitative knowledge in qualitative reasoning," *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, 1988.

44. B. Kuipers and D. Berleant, *A smooth integration of incomplete quantitative knowledge into qualitative simulation*. Artificial Intelligence Laboratory, The University of Texas at Austin, AI90-122, 1990.

45. P. M. DeRusso, R. J. Roy and C. M. Close, *State Variables for Engineers*. John Wiley & Sons, 1965.

46. P. Struss, "Global filters for qualitative behaviors," D. S. Weld and J. de Kleer (Eds.) *Readings in Qualitative Reasoning About Physical Systems*. Los Altos, CA: Morgan Kaufmann, 1990.

47. R. Penrose, *The Emperor's New Mind*. Oxford: Oxford U. Press, 1989.

48. L. Ljung, "Issues in system identification," *IEEE Control Systems*, Vol. 11, pp. 25-29, 1991.

49. C. W. Xu and Y. Z. Lu, "Fuzzy model identification and self-learning for dynamical systems," *IEEE Transactions on Systems, Man, and Cybernetics* , Vol. 17, pp. 683-689, 1987.

50. K. D. Forbus, "Interpreting observations of physical systems", D. S. Weld and J. de Kleer (Eds.) *Readings in Qualitative Reasoning About Physical Systems*. Los Altos, CA: Morgan Kaufmann, 1990.

51. D. Halliday and R. Resnick, *Physics*. John Wiley & Sons, 1978.

52. R. Bhaskar and A. Nigam, "Qualitative physics using dimensional analysis," *Artificial Intelligence* , Vol. 45, pp. 73-111, 1990.

53. J. Rothenberg, "The nature of modeling," L. E. Widman, K. A. Loparo and N. R. Nielsen (Eds.) *Artificial Intelligence, Simulation and Modeling*. New York: John Wiley and Sons, 1989.

54. R. M. O'Keefe, "The role of artificial intelligence in discrete-event simulation," L. E. Widman, K. A. Loparo and N. R. Nielsen (Eds.) *Artificial Intelligence, Simulation and Modeling*. New York: John Wiley and Sons, 1989.

55. K. D. Forbus, "Qualitative Physics: Past, present and future," D. S. Weld and J. de Kleer (Eds.) *Readings in Qualitative Reasoning About Physical Systems*. Los Altos, CA: Morgan Kaufmann, 1990.

56. R. J. Doyle, "Reasoning about hidden mechanisms," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, 1989.

57. *Turbo PROLOG Reference Guide Version 2.0*. Borland International, 1988.

58. W. F. Clocksin and C. S. Mellish, *Programming in Prolog*. Berlin: Springer, 1987.

REFERENCES NOT CITED

H. Abelson, M. Eisenberg, M. Halfant, J. Katzenelson, E. Sacks, G. J. Sussman, J. Wisdom and K. Yip, "Intelligence in scientific computing," *Communications of the ACM,* Vol. 32, pp. 546-562, 1989.

T. Bylander, "A critique of qualitative simulation from a consolidation viewpoint," *IEEE Transactions on Systems, Man, and Cybernetics* , Vol. 18, pp. 252-263, 1988.

C. Chiu, "Constructing qualitative domain maps from qualitative simulation models," L. E. Widman, K. A. Loparo and N. R. Nielsen (Eds.) *Artificial Intelligence, Simulation and Modeling.* New York: John Wiley and Sons, 1989.

M. O. Cordier, "SHERLOCK: An expert system with hypothetical reasoning capabilities," *Proceedings of the Second International Symposium on Computer and Information Sciences (ISCIS-II)* , İstanbul, 1987.

D. R. Coughanowr and L. B. Koppel, *Process Systems Analysis and Control.* New York, NY: McGraw-Hill, 1965.

P. T. Cox and T. Pietrzykowski, "Causes for events: Their computations and applications," *Proceedings of the Eighth International Conference on Automated Deduction,* Oxford, 1986.

B. D'Ambrosio, "Extending the mathematics in qualitative process theory," L. E. Widman, K. A. Loparo and N. R. Nielsen (Eds.) *Artificial Intelligence, Simulation and Modeling.* New York: John Wiley and Sons, 1989.

C. J. Date, *An Introduction to Database Systems.* Reading, MA: Addison-Wesley, 1989.

T. Dean, "Using temporal hiearchies to efficiently maintain large temporal databases," *Journal of the ACM,* Vol. 36, pp. 687-718, 1989.

T. L. Dean and D. V. McDermott, "Temporal data base management," *Artificial Intelligence,* Vol. 32, pp. 1-55, 1987.

J. de Kleer, "An assumption-based TMS," *Artificial Intelligence,* Vol. 28, pp. 127-162, 1986.

J. de Kleer, "Extending the ATMS," *Artificial Intelligence,* Vol. 28, pp. 163-196, 1986.

J. de Kleer, "Problem solving with the ATMS," *Artificial Intelligence,* Vol. 28, pp. 197-224, 1986.

J. de Kleer and J. S. Brown, "Theories of causal ordering," *Artificial Intelligence,* Vol. 29, pp. 33-61, 1986.

J. de Kleer and B. Williams, "Diagnosing multiple faults," *Artificial Intelligence,* Vol. 32, pp. 97-130, 1987.

V. Dhar and H. E. Pople, "Rule-based versus structure-based models for explaining and generating expert behavior," *Communications of the ACM,* Vol. 30, pp. 542-555, 1987.

D. W. Franke, *Proposal for research: Representing, utilizing and acquiring teleological descriptions.* Artificial Intelligence Laboratory, The University of Texas at Austin, AI89-112.

F. Gardin and B. Meltzer, "Analogical representations of naive physics," *Artificial Intelligence,* Vol. 38, pp. 139-159, 1989.

T. Güngör, "Generating Causal Relations from Mathematical Models", M. S. Thesis, Boğaziçi University, 1989.

P. J. Hayes, "Naive physics I: Ontology for liquids," J. R. Hobbs and R. C. Moore (Eds.) *Formal Theories of the Commonsense World.* Norwood, NJ: Ablex, 1985.

D. R. Hofstadter, *Gödel, Escher, Bach: Ein Endloses Geflochtenes Band.* Translated from the english original *Gödel, Escher, Bach: An Eternal Golden Braid* by P. Wolff-Windegg and H. Feuersee, Münich: DTV, 1991.

D. Israel, "A short companion to the naive physics manifesto," J. R. Hobbs and R. C. Moore (Eds.) *Formal Theories of the Commonsense World.* Norwood, NJ: Ablex, 1985.

S. Kuru and T. Güngör, "Sign analysis technique for predicting system behavior," to appear in *International Journal of Intelligent Systems.*

S. Kuru and A. W. Westerberg, "A Newton-Raphson based strategy for exploiting latency in dynamic simulation," *Computers and Chemical Engineering*, Vol. 9, No.2, pp. 175-182, 1985.

N. J. Nilsson, *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga, 1980.

O. O. Oyeleye and M. A. Kramer, "The role of causal and noncausal constraints in steady-state qualitative modeling," L. E. Widman, K. A. Loparo and N. R. Nielsen (Eds.) *Artificial Intelligence, Simulation and Modeling*. New York: John Wiley and Sons, 1989.

J. A. Palesis, R. W. Dwyer, D. L. Leister and J. W. Kao, "Transforming mathematical product evaluation models into expert systems for product design," *Proc. 3rd Int'l. Conf. Industrial and Engineering Applications of AI and Expert Systems*, July 15-18, 1990, Columbia, S. Carolina.

A. C. C. Say and S. Kuru, "Postdiction by qualitative simulation," *Proceedings of the Fifth International Symposium on Computer and Information Sciences (ISCIS-V)*, Nevşehir, 1990.

A. C. C. Say and S. Kuru, "Nitel model tanılama," *Bilkon-91 1991 Bilkent Elektrik-Elektronik ve Bilgisayar Mühendisliği Konferansı*, Ankara, 1991.

A. C. C. Say and S. Kuru, "Qualitative system identification," *Abstracts of the Fifteenth IFIP Conference on System Modeling and Optimization (IFIP-91)*, Zurich, 1991.

T. Söderström and P. Stoica, *System Identification.* Prentice Hall, 1989.

R. K. Stobart and N. R. Shadbolt, "Process control supervision using qualitative models," *Proc. 3rd Int'l. Conf. Industrial and Engineering Applications of AI and Expert Systems*, July 15-18, 1990, Columbia, S. Carolina.