Qualitative System Identification: Deriving Structure from Behavior

A. C. Cem Say and Selahattin Kuru
Department of Computer Engineering, Boğaziçi University, Bebek 80815, İstanbul, Turkey

ABSTRACT
Qualitative reasoning programs (which perform simulation, comparative analysis, data interpretation, etc.) either take the model of the physical system to be considered as input, or compose it using a library of model fragments and input information about how to combine them. System identification is the task of creating models of systems, using data about their behaviors. We present the qualitative system identification algorithm QSI, which takes as input a set of qualitative behaviors of a physical system, and produces as output a constraint model of the system. QSI's output is guaranteed to produce its input when simulated. Furthermore, the QSI-made models usually contain meaningful "deep" parameters of the system which do not appear in the input behaviors. Various aspects of QSI and its applicability to diagnosis, as well as the model fragment formulation problem, are discussed.

1. Introduction

Research in qualitative reasoning about physical systems [36] has resulted in many programs designed to achieve various tasks of commonsense reasoning being produced. These reasoners take a "deep" model of the underlying mechanism of the system under consideration as part of their input and analyze or predict its behavior in one of a variety of ways.

Before performing any kind of model-based reasoning, one has to have a model of the system which will be reasoned about. The model composition methods used by some 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, overlook the issue of initial creation of model fragments, using which the complete system models are built. When faced with a novel situation, or a new mechanism whose description is not available in the library, reasoners employing such an approach may be unable to 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 behavior-based model construction is the subject of an already mature field, named *system identification.* As a result of extensive research in this field, widespread applications of efficient algorithms which perform system identification in the numerical domain have been produced. In this paper, QSI, a program

which performs ***Q****ualitative* **S**ystem **I**dentification, using the qualitative representation, is presented. QSI's input is a set of qualitative behaviors of a physical system, and its output is a constraint model of that system.

The outline of the paper is as follows: Section 2 is an overview of the aspects of qualitative physical reasoning relevant to the modeling problem. The device-centered and process-centered views of modeling are briefly examined. The QSIM [19,22] representation and algorithm, which we have borrowed in the construction of QSI, are summarized. Section 3 puts QSI into the broader system identification perspective. Section 4 contains a detailed explanation of the algorithm, and analyses of its complexity and correctness properties. The qualitative noise filtering method, designed to serve as a preprocessor for QSI, is presented in Section 5. Section 6 contains a comprehensive discussion on the strengths and weaknesses of the QSI algorithm, its applicability to diagnosis as well as model formulation, and its relation to other work in the field. Section 7 is a conclusion. The Appendices cover some technical issues, and contain several examples of the program in action.

2. Qualitative Physical Reasoning: An Overview

This section is an overview of the technique of qualitative simulation; the major approaches will be briefly discussed in chronological order.

de Kleer and Brown [7] established the foundations of the qualitative calculus. The basic idea is that real (continuous) quantities are represented by a finite number of qualitative values: their signs. The time derivatives of each quantity are similarly represented. The fact that a quantity is increasing, for instance, is represented by its derivative having the value +. A mechanistic world view is adopted; every system to be simulated is assumed to be a mechanism composed of simple components. The input system models are formed by connecting component models in the simulator's component library (which obviously has to be large if the program is supposed to be able to deal with a large variety of systems,) according to the device topology of the system. Each component has its own "law," an equation relating the variables involved with it. The laws of all the components of a mechanism have to be satisfied; this means that they can be solved as a system of equations to determine the qualitative values of all the variables in them. The derivatives are examined to determine which transitions to other *system states*, where certain variables have qualitatively different values, are possible. A graph of system states, each path through which represents a different prediction for the behavior of the simulated system, is thus constructed.

The method of de Kleer and Brown embodies a *component-centered* approach to modeling, as discussed. An alternative is the *process-centered* approach of qualitative process theory [13]. In this theory, the relationships holding among the quantities in the considered scene are determined by the processes that are currently active. A *process* is something which causes changes; like heating, cooling, boiling, stretching, etc. Given information about the values, individuals, and their configuration, use of a process library (which should be as big as possible, again for the above-mentioned reasons) is made to come up with the system model, properly composed of various *model fragments* contributed by the active processes, reflecting the constraints that they impose on the system quantities. Since time derivative information is also kept for quantities, it is possible to determine which quantity is nearing which landmark. (A *landmark* is a symbol representing a point value which is significant for the purposes of the model.) The system state, and, sometimes, the set of active processes, change when a quantity

crosses a landmark value. A state graph, similar to the one mentioned in the previous paragraph, is constructed by linking predecessor states to successor states determined in this manner.

Williams [38] played an important role in the perfection of qualitative simulation. The method of *transition ordering*, which can be used to determine which of a group of related quantities in the system will change qualitative value earlier, was first presented in [38]. Williams focused his work on the domain of electrical circuits, where the need for tutoring, design, and diagnosis aids which can explain the workings of the circuits in terms of causal explanations based on the simple component laws is evident.

The QSIM algorithm [19] is in many ways the most advanced qualitative simulator. We chose to adopt QSIM's representation of qualitative models in our research.

QSIM leaves the modeling task entirely to the user; a correct and complete system model has to be written in the qualitative format, which we will now explain in detail, before simulation can begin.

Each ODE (Ordinary Differential Equation) obeying certain restrictions can be translated to a corresponding QDE (Qualitative Differential Equation,) that is, a set of *qualitative constraints*, which describes the same system. These constraints are time-invariant relationships between the *parameters* (continuous valued functions of time) comprising the system. There are six types of constraints (Table I.) Each constraint (except those of the DERIV type) may possess tuples of *corresponding values* (CVs) of particular values of the parameters it binds; in this manner, additional information about the relation embodied by the constraint can be represented. QDEs are formed of instances of constraints linking the system parameters. A system may have several operating regions, each corresponding to cases in which it is governed by a different QDE, as exemplified below.

As a classic [22] example that will also be used later in the discussion, consider how a simple U-tube (Figure 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.

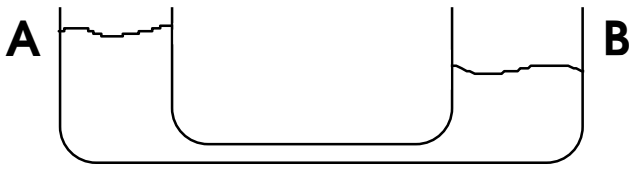

Figure 1. U-tube in operating region NORMAL

After a lot of simplifying assumptions, the parameters of the system are identified as in Table II. (A *quantity space* is an ordered set of the landmarks of a parameter. The landmarks -∞, 0 and ∞ are members of every quantity space.) There are also the *invariants* that the amount and pressure parameters are never negative.

The constraints for operating region NORMAL are listed in Table III. 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.

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 IV. 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.

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 V.

Note the qualitative representation for parameter states; the magnitudes are shown as landmarks or intervals between consecutive landmarks, and the sign of the time derivative of each parameter is part of its qualitative state: **inc**, **std**, and **dec** mean +, 0, and -, respectively. Since the point-interval representation is used for time as well, a *qualitative behavior* of the system will be a sequence of its states at t_0, (t_0,t_1), t_1, (t_1,t_2), etc.

QSIM builds a tree of system states whose root is the initial state. Each path in this tree from the root to a leaf is a predicted behavior of the system. The successors of each node are created (i.e. time is "advanced") as follows: All the system states comprising of all the possible qualitative states that each parameter can take on in the next time point or interval (obeying continuity) are implicitly created. (For example, there are only four "next" states that a quantity whose current state is <(a,b),inc> can take on: <(a,b),inc> again, <b, inc>, <b, std>, or <x, std>, where x is a new landmark between a and b.) Of these system states, only the ones in which all the constraints in the QDE hold are acceptable as possible "next" states of the system, i.e. successors of the current node. In this manner, the prediction of all the possible future behaviors of the system is guaranteed.

The three behaviors that QSIM predicts for the input of Tables III and V, (Tank A contains liquid, tank B is empty,) seen in Tables VI thru VIII, 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. As can be seen, simulation stops when all parameters have the qualitative direction **std**, a heuristic lets the reasoner assume that the system has reached equilibrium in such cases.

During the region transition from NORMAL to B_BURST in Table VIII, 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 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.

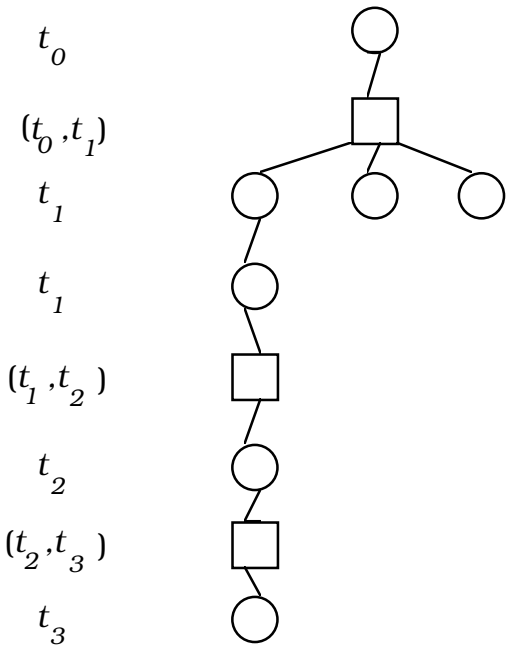

Figure 2. State tree for U-tube simulation (time values shown)

For some inputs, qualitative simulation programs predict *spurious* behaviors, which can not be exhibited by any system with that model. Although some of these spurious predictions are results of the manner in which corresponding value tuples are kept in the original QSIM algorithm, and can be eliminated by using an improved version [34], which is actually the one we incorporated into QSI, an important class of spurious behaviors remain undetected even by this improved algorithm. The inherent information loss in the nature of the qualitative representation is the cause of this problem. In the case of the spring-block system [19] of Figure 3, the QDE of which can be seen in Table IX, spurious behaviors in which the block stops at a different point at each period are produced, for instance. (The parameters X, V, and A are, respectively, the horizontal position, velocity, and acceleration of the block.) Kuipers points out [19] that this problem can be overcome by providing a deeper system model to the simulator, and therefore work in the realm of model preparation is needed. QSI is a step in this direction.

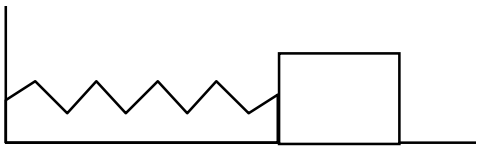

Figure 3. Spring/block system

3. QSI as System Identification

First of all, a potentially confusing difference in terminology will be clarified. The things we call *parameters* in this paper are generally called *variables* in "conventional" system identification (SI). In SI literature, 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.

Generally, there are two kinds of variables in SI: *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 SI *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 SI, [26] 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 parametrization?

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 SI is the parameter estimation step. Elaborate numerical algorithms for this task have been developed.

There has been some work [40] on performing SI 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 SI: It starts with ready (qualitative) data about the behaviors. It treats input and output 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 SI: 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 4.9.

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 SI is similar to those of other qualitative reasoning methods to their quantitative counterparts: The qualitative methods suppress 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 SI 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.

4. The QSI 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 [32,31,33] which constructs system models from their behaviors. The requirements on the input, the formats of input and output, examples, detailed discussions of the algorithm's individual stages, and complexity and correctness analyses will be presented.

4.1. 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, (in fact, *all* the behaviors that would be expected/observed for each initial state which appears in the input should be present,) 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 4.1. A behavior is *T-legal* if all the parameters in it obey the transition rules of [19] 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 [9]. Because of one's measurement intervals, one may "jump" over some 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 4.4) 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, QSI employs the input marker symbols EQU and CYC. 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 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 SI, 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 4.6 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.

4.2. The Algorithm

The algorithm 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.

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 5.) 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 4.6) 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.

4.3. An Example

As an example to the operation of QSI, the U-tube (in operating region NORMAL) of Section 2 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 X and XI.

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 2. 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 2 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 6.)

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 Appendix B 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 4.5.) 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 XII. Already, another important relationship has been discovered: The sum of the amounts is fixed, i.e. mass is conserved.

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. For instance, in the behaviors of Table XII, the constraint ADD(amount_A,PX,amount_B) is seen to be satisfied. Parameter PX's defining constraint DERIV(amount_A,PX) is (of course) also satisfied, because of the way we assigned the values for PX. These two constraints are "independent" in the sense that neither one of them can be obtained from the other one by algebraic manipulation, no matter how many additional assumptions are made. (In fact, if one considers the dimensions that the parameters involved have to possess, the two constraints are inconsistent; we handle this issue in the dimension consistency stage.) So QSI would add these two constraints to the system model at this point. Note that not all defining constraints need end up in the final model; if a new parameter does not appear in any constraint independent of its defining constraint, it will not be incorporated in the QDE. During this constraint determination, a lot of constraints which can be proven to be implied by others already in the model are not even checked tuple by tuple, which is good for efficiency.

The U-tube 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 XIII. The simulation of this model from the initial states predicts only the input behaviors, so an acceptable model has been obtained.

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 XIV, which is, although slightly different than the model of Section 2, 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 6 for deep parameters which are not so meaningful, and an interpretation for them.)

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.

4.4. 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 contradict 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

$$\binom{p}{2} = \frac{p.(p-1)}{2} = \frac{p^2 - p}{2}. \qquad (4.1)$$

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

$$2.\binom{p}{2} = p^2 - p \qquad (4.2)$$

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 Section 2. 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 4.5.) The formula for the number of controls of ADD and MULT constraints on three different parameters is thus

$$3.\binom{p}{3} = 3.\frac{p.(p-1).(p-2)}{6} = \frac{p^3 - 3p^2 + 2p}{2} \qquad (4.3)$$

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

$$\frac{2p^3 + p^2 - 3p}{2} \qquad (4.4)$$

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+. This feature, represented by the first if statement in the description of the algorithm, is called the *contradiction check*.

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 Appendix A.

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 4.5 and 4.8.)

QSI also checks each parameter to see whether it is fixed at the same landmark throughout all the input behaviors at this stage. If such a parameter is found, invariant information indicating that it is a constant is incorporated to the model. Later versions of QSIM have a unary constraint named CONSTANT for fixed parameters, see [15], for instance.

4.5. 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 4.6) 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 4.8) 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 Appendix B.) 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 Y/X 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 4.3 will be examined more closely. (Table XV.) 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 XVI about PX's behavior in $[t_0, t_1]$. It is also known that PX will have magnitude zero after t_1.

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.

But this still leaves an infinite number of possibilities for the behavior of PX, the ones depicted in Tables XVII thru XIX being among them. There is nothing wrong about the "length" of the behavior in Table XIX; 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 XX, with the period designated $[t_0, t_1]$ in the "PX-less" form of the behavior now being described by five states from t_0 to t_2.

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 X. 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.

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 common sense 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 [1] .

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. 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 Appendix B.

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

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

---

[1] Note the parallels to Newton's first law and de Kleer and Brown's canonicality heuristics [7].

The discovery of significant constraints is made in two stages: 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 can have only one possible "next" state at any given time, and will probably help further constrain the system behavior through their neighbors in subsequent simulations.

The constraint determination procedure is then called to find the other 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.

To hinder the incorporation of constraints that would not "tighten" the simulation, constraint determination at this point also involves an *insignificance* check, which is mostly 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 deemed insignificant. Since the qualitative addition of values of opposing sign is ambiguous, such constraints can be satisfied for lots of (nonzero) Zs, which is arithmetically wrong; their generation is just a by-product of QSI's policy of testing every combination. See Appendix A for the other rules used for determining insignificant constraints.

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 4.9.)

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.

4.6. Model Depth Testing

Whether the version of the system model at hand 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 4.9.) 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 feature is called *level limiting*.

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 XXI.

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 XXII, which differ only in the final magnitude of Z, for the X-Y-Z system.

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 Appendix B.)

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.

4.7. 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 4.3 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.

4.8. Complexity

The computational complexity of QSI will now be determined stage by stage. Most of the required analysis has already been done. 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.

4.8.1. Analysis

Constraint Determination

Since worst-case complexity is being considered, assume none of the conditions (Section 4.4) 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 at the i^{th} iteration then requires $O(p_i^3 s_i^2)$ time for large p_i. (The CV lists, which will have to be checked for each tuple for most constraint types, grow with s_i ; hence the second s_i factor in this formula.) 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.

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 $p_{0,}^{2}$, $p_{0,}^{4}$, $p_{0,}^{8}$,... 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.

4.8.2. 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 Appendix B. 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 paper 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. Active research is going on [37] 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 XXIII, which lists the execution times of some of the problems in Appendix B in the current PC implementation. The section numbers indicate where each problem has been presented. Details can be found in Appendix B.

4.9. 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 4.2. 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 4.1. 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 4.2. 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 4.2(b) are *not* written out by the algorithm. All other constraints which satisfy the consequence or insignificance tests *are* valid; see discussion in Sections 4.4, 4.5 and proofs in Appendix A. 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 4.3. 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 4.4. 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 ($\varnothing$) 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 4.5. 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 $\varnothing$ 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 [19].) However, every constraint found by constraint determination holds in every state of the input behaviors, (by Proposition 4.2) 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 section, 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 4.3. 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 4.6. 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.

5. Qualitative Noise Filtering

Depending on the specifics of the application, two preprocessors, (which are not fully implemented at the present time, unlike the the core algorithm explained in Section 4,) 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 floating point 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 detailed discussions of the issues related to this conversion, see [14] and [6].

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, and the fact that the quantitative-to-qualitative behavior conversion preprocessor itself can employ a method for "numerical" noise elimination, we have designed a qualitative noise filter as well, mainly to demonstrate that this process too has a meaningful qualitative counterpart. 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* [19] 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 was 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 4, one intuitively expects its noisy version to be as in Figure 5. By the same reasoning, if one sees Figure 5 and is told that noise is present, then one would propose something like Figure 4 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 [4], 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.

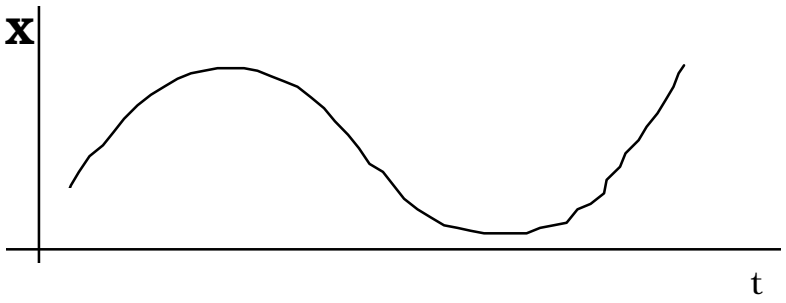

Figure 4. Actual behavior of X

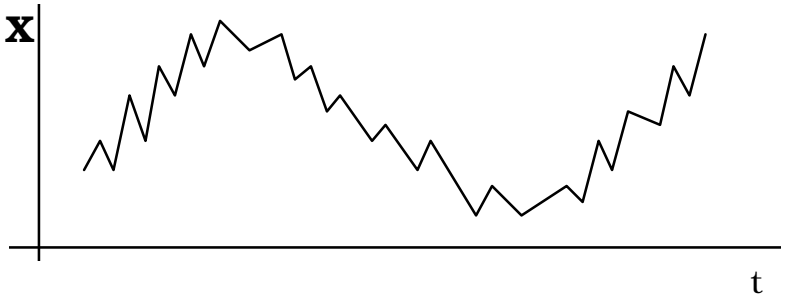

Figure 5. Noisy behavior of X

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 5 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 4 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 parameter states in which the parameter's direction starts as **inc** (or **dec**) in the first one or more states, 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 5. The basic idea behind 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 4 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 states 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 state <*lm*, std> outside this sequence for any landmark *lm* for which it has such a state in this sequence, c) The state <*0*, std> does not appear in the sequence, and d) The odd-numbered (i.e. 1st, 3rd, 5th,...) 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 states is replaced by the state <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.

Figure 6 is an example showing how a noisy sequence is smoothed by the filter. Plots of the noisy and filtered versions of this segment of this parameter's behavior are presented in Figure 7. One should keep in mind that several such jagged segments could be expected to appear in a noisy parameter behavior, each corresponding to one actual interval state of the form <(*a*, *b*), inc> or <(*a*, *b*), dec>.

<(b1,b2),inc>		<(b1,u3),inc>
<b2,inc>		<(b1,u3),inc>
<(b2,u1),inc>		<(b1,u3),inc>
<u1,std>		<(b1,u3),inc>
<(b2,u1),dec>		<(b1,u3),inc>
<b2,std>		<(b1,u3),inc>
<(b2,u1),inc>		<(b1,u3),inc>
<u1,inc>	-------->	<(b1,u3),inc>
<(u1,b3),inc>		<(b1,u3),inc>
<b3,inc>		<(b1,u3),inc>
<(b3,u2),inc>		<(b1,u3),inc>
<u2,std>		<(b1,u3),inc>
<(b3,u2),dec>		<(b1,u3),inc>
<b3,std>		<(b1,u3),inc>
<(b3,u2),inc>		<(b1,u3),inc>
<u2,inc>		<(b1,u3),inc>
<(u2,u3),inc>		<(b1,u3),inc>

Figure 6. Noise filtering of a parameter behavior segment

Parameters indicated to be noisy by the user are treated one by one in this fashion, and their states modified by the filter are updated in the system behaviors. When the parameter filterings are over, the system behaviors may contain sequences of system states that are indistinguishable except for their time values. This is to be expected, since the filters contribute to a reduction of the filtered parameters' distinguished time-points by erasing some of their landmarks. The preprocessor terminates after shortening the system behaviors by collapsing such sequences

of identical system states into single interval states. This reduction in the size of the QSI input is naturally an improvement from the point of view of the time requirement as well.

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.

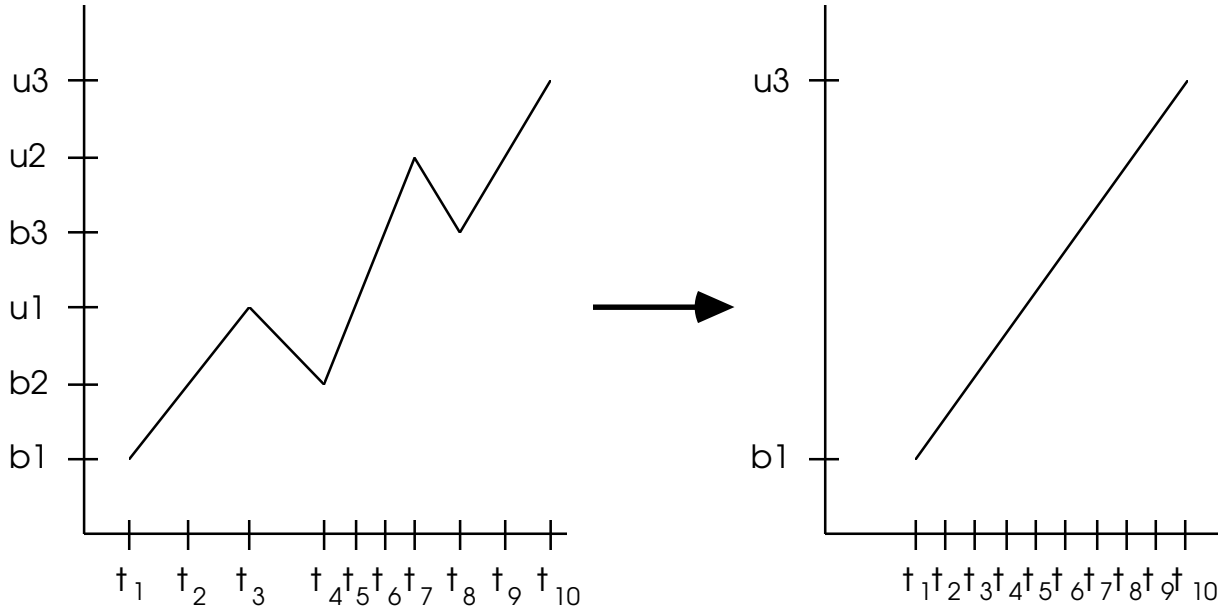

Figure 7. Plots of noisy and filtered behavior segments

6. Discussion

This section 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. Related work on automatic model acquisition is examined.

6.1. 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 2. Recall that the three-parameter, three-constraint model of Table IX, which, although mathematically adequate to produce only a single periodic behavior, leads to a simulation with infinite spurious solutions, because of inherent representation problems. As Kuipers [19] points out, 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 periodic behavior of this system with the three parameters X, V, and A, has been presented to QSI as input. The initial constraint determination finds the

constraint set of Table IX, 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 (two of which are spurious) 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 terminates.

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 [16] is

$$\frac{1}{2}mv^2 + \frac{1}{2}kx^2 = E \qquad (6.1)$$

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 6.1.2. 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.

6.1.1. How to Prepare QSI's Input

As pointed out in Section 3, 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 Section 5 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. See Section 6.3 for an account of how dimensional information has been used in modeling by other researchers.

6.1.2. 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

$$F = ma \tag{6.2}$$

and

$$F = -kx \tag{6.3}$$

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 [19], 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 [21], and also Section 6.1.4.) 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 4.3, 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.

6.1.3. 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 this paper, 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 [12], may come into play. In this regard, QSI is taking what Rothenberg [30] 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 seem to employ "insight" to recognize the particular law instances that do apply. A QSI-style search is absent. Iwasaki and Simon [17] say that "good" models should contain each different law in a different *structural* equation, like equations 6.2 and 6.3, rather than combining them, like writing

$$mA = -kx \qquad (6.4)$$

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 6.1.2. As Iwasaki and Simon point out in [17], "Establishing the structural equations for a system is as much an empirical as a formal matter, and certainly not a syntactical exercise."

Some computer programs which attempt to automate certain human modeling skills will be examined in Section 6.3.

6.1.4. QSI for Diagnosis

Kuipers [21,23] 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. (See [9] for a description of MIMIC, a different approach to QSIM-model-based diagnosis.)

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 XXIV and XXV contain healthy and diseased models [21,23,20] of the water balance mechanism (Table XXVI) 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 XXV 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 [21]. 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 [21], 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 XXIV and XXV 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.

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. Appendix B 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.

6.1.5. 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 may be useful in eliminating such problematic states, but that process is itself heuristic and may "oversmooth" the input.

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 Appendix B illustrate, a sizeable class of QSI problems can be solved using the limited postulation and search modes, which reduce 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.

6.2. 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.

Induction is the best known method used in machine learning. Here is a simplified definition [4] 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 program, 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.

Inductive learning algorithms have been used by other researchers for qualitative modeling; see the next section for the details.

6.3. Related Work

Even when one restricts attention to programs which produce system models from behavior information, as opposed to obtaining them by combining smaller models using the methods mentioned in Section 2 on structural system descriptions, there still remains a remarkable number of studies which deal with the automatic model formulation problem.

The primary feature which can be used to compare the approaches adopted by these programs is the amount of information *other than* the system behaviors required by each to achieve the modeling task. We will examine them starting with the *knowledge-intensive* ones, which require a great deal of such additional information, and go on to the *knowledge-weak* ones, of which QSI is among the weakest.

Falkenhainer's PHINEAS program [12] constructs models of systems whose behaviors are input by drawing *analogies* to other models which are remembered to have produced "similar" behaviors. A (preferably big) library of qualitative process models and structure and behavior descriptions about each of those models is required for the program to work. Input behaviors which (partially) match the "previously seen" behaviors cause the relevant parts of the model which generated the previous behavior to be used in the "target" model. This target model is qualitatively simulated to verify its correctness: The simulation output and the PHINEAS input are compared and possible discrepancies are attempted to be handled by a revision stage.

Mozetic's QuMAS [25] used partial model knowledge and behavior examples to automatically complete the model. Doyle [8] built a program which took qualitative behaviors and some structural information about the considered systems as input and used a rich library of commonly found device components to hypothesize models for the systems. Amsterdam's MM [1] takes as input an even greater amount of structural information in the form of geometric figures and/or component-connection descriptions in addition to the qualitative behavior, in which the user has to specify the *type* of each appearing quantity. These types, which are to be selected from a set which includes `acceleration`, `flow-rate`, `torque`, `momentum`, and `voltage`, among others, are used to index into a library consisting of several model building/augmenting rules, written using knowledge of the energy-based modeling framework. The several candidate models produced in this manner are analyzed in a multi-step stage, which includes a QSIM simulation to see if they produce the input behavior after all. MM allows partial specification of parameter values in the input; e. g. the qualitative directions of some parameter states may be missing.

When so much modeling knowledge is not available, one has to turn to other methods of behavior-based modeling. An early example to the use of inductive algorithms to generate model fragments for simulation was the incorporation of Quinlan's ID3 algorithm in the package EXPERT-EASE [27], where it could produce

discrete-event simulation rules. These rules were 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.

Several "general-purpose" inductive learning programs have been used for the identification of QSIM models from QSIM behaviors: Bratko *et al.* [3] used GOLEM to identify a U-tube, starting with an input containing only the amount and flow parameters, and obtained a complete and correct model. Dzeroski [10] reports a similar experiment with the mFOIL program. The representational setup of such learners seems to be particularly unsuited to the qualitative modeling framework. In both cases, the MULT constraint and the whole corresponding values feature were left unimplemented so that the execution time would be reasonable. Furthermore, these programs require negative examples (i.e. unexhibitable behaviors) in addition to the correct system behaviors for their operation.

Coiera's GENMODEL [5] was the first "special purpose" program for QSIM model identification. GENMODEL basically corresponds to what we called the initial constraint determination stage. Since it has no parameter postulation feature, it would terminate with a shallow model if some parameters were omitted from its input.

MISQ [29] and QSI are similar programs in many regards. MISQ has an implemented optional preprocessor which can convert numerical sensor data to qualitative behaviors. Unlike QSI, it allows partially specified qualitative values (as described above for MM) and dimension information (also used by Bhaskar and Nigam [2] for qualitative relation formulation) about the parameters in its input. MISQ's equivalent of the initial constraint determination stage makes use of this dimension information, in addition to value checking, during its operation. Since incomplete value information is allowed, MISQ can find an inconsistent constraint set at the initial constraint determination. This is handled by erasing problematic constraints. Several alternative models, each obtained by erasing one such constraint, can be created. Dimensional inconsistencies in the constraint set are handled similarly. MISQ deems a model satisfactory when it is consistent and *connected*, that is, each parameter is linked by a "chain" of constraints to all other parameters. Because of this, MISQ would terminate with the shallow single-constraint model M-(amount_A, amount_B), when given the input of Tables X and XI. When it obtains an unsatisfactory model at the end of the initial constraint determination, MISQ postulates new variables using a method called *relational pathfinding*, which is functionally almost equivalent to what we call model extension at full postulation-half search mode. (Relational pathfinding searches a somewhat smaller number of constraints than our version.) After one application of this method, MISQ terminates, even if the model is still unconnected.

The two programs' respective performances on the two-parameter U-tube input seem to indicate that QSI's criterion for model "goodness," i.e, number of simulation behaviors, can lead to better results compared to MISQ's connectedness condition. QSI's deferment of the imposition of dimensional consistency until the very end helps it output a single model which contains all "model-tightening" relationships discovered during the execution, as explained in Section 4.

Dzeroski and Todorovski's QMN [11] takes tuples of quantitative parameter values, and builds a QSIM model directly, without converting its input to the qualitative format. QMN requires the user to explicitly specify the system *order* (i.e. the length of the longest DERIV chain in the output,) the maximum *depth* of new parameters that can appear (e.g. a new parameter that is the sum of five other parameters is two levels "deeper" than one which is the sum of only three of the others,) optional parameter dimensions, and tolerance values used to disregard numerical deviations hopefully caused by precision and noise problems. The algorithm first postulates new parameters up to the specified order and depth, and then performs a numerical version of constraint determination on this big set of parameters. Unlike QSI, (and, to an extent, MISQ,) QMN has no mechanism which lets it terminate without unnecessarily complicating the output when a satisfactory model has been found at an intermediate step during the execution.

LAGRANGE [11] is a relative of QMN which produces *quantitative* models (ODEs) as output. The set of "new" parameters is postulated in a manner similar to QMN. Significant equations composed of various combinations of these parameters are determined by linear regression. The above comments about the lack of a "depth-test" mechanism apply to LAGRANGE as well.

Ramachandran *et al.* [28] have proposed a technique for recognizing that an input qualitative behavior contains several operating regions. Their method can work in conjunction with any one of the single-region qualitative model induction algorithms described in this section, as well as QSI.

Behavior-based modelers like QSI can be considered to be performing a kind of "discovery," like [24]. As Dzeroski and Todorovski [11] point out, the fact that they usually ask for additional experimental data during the execution, and their requirement that the parameters should be designated as dependent or independent, are among the factors that distinguish machine discovery programs from the qualitative modelers.

7. Conclusion

We have presented an algorithm for qualitative system identification; the task of model-building when incomplete information in the qualitative format about system behavior is available. QSI is a model fragment formulation tool; in contrast to various model "composers" which build models by appropriately piecing together already available model fragments, it creates them from scratch, using no information about the particulars of the system domain. The correctness of the constraint determination procedure was proven, complexity analyses for each part of the algorithm were performed, and several examples illustrating various aspects of the problem and possible application areas were presented.

QSI embodies a very "mechanical" approach, 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. This research has shown that model structure identification at a quite impressive scale is possible even with only this much data.

As well as being used in a stand-alone fashion for automatic model acquisition, QSI may form a part of the basis of a much more involved modeling procedure which combines the many approaches to this difficult mental task in the future. In such a setup, QSI's role would be to prepare initial model proposals, from which the more

knowledge-intensive stages (as defined in Section 6.3) would prune the coincidental and noninteresting constraints. Research into what humans actually "do" during the model-building task would certainly provide valuable pointers for the construction of such a program.

As for more immediate areas of improvement for QSI, one can mention the ongoing experimentation on new behavior selection heuristics, and the planned incorporation of the capability of using optional dimensional knowledge (similar in format to that used in [29]) to the dimension consistency stage, to eliminate unnecessary buffer parameters.

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

Appendix A. Consequence Constraints

In Section 4, 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. See [18] for another application of similar ideas about M constraints, and [39] for the description of MINIMA, a symbolic algebra system which supports Q1, a qualitative-quantitative hybrid algebra, also described in that paper.

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 [19]. 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 A.I. 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.")

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.

Appendix B. QSI in Action

This appendix is comprised of two parts. 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. 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 part discusses some detailed features and ways of handling certain remaining difficulties with the method.

B.1. Examples

B.1.1. U-tube with Full Postulation

As shown in Section 4, 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 X and XI. 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 B.I 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 B.2 for a discussion on this.

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 B.II are found to hold. The constraints marked as "consequence" or "insignificant" in the table are not checked against the values of Table B.I, and they are not added to the QSIM input. New parameters that appear in the remaining entries of Table B.II 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 6.1.2. 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 could be totally eliminated.

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 as a last resort, if an initial approach using the derivative postulation mode proves to be unsuccessful.

B.1.2. Single Leaking Tank

In Section 4, 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 how 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:

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

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 B.IV.)

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 B.V.

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 usefulness has once again been demonstrated; the following examples will further underline this.

B.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 B.VI) 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 4.) 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 B.VII.

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

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

B.1.4. Water Balance in Kidney

Consider the situation in which only the two (supposedly) most easily observable parameters of Table XXVI, 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 XXIV 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 B.X) has been obtained by observing the "normal" state of things for some time. Both parameters are at their equilibrium values.

The second behavior (Table B.XI) 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 B.XII, 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 B.XIII 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 XXIV. 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 XXIV. 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.

B.1.5. Heat Exchanger

The heat exchanger system (Figure B.1) to be used in this example is from [35]. 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 Tables B.XIV thru B.XVI.

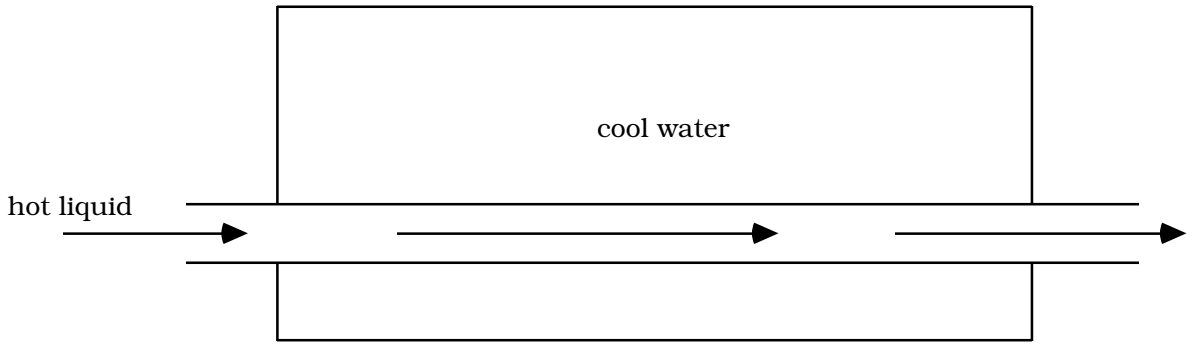

Figure B.1. The heat exchanger

Initial constraint determination comes up with the model of Table B.XVII, 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 XXIII 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.

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 B.2 for a discussion.

B.1.6. The Upward Thrown Ball

This section will be concluded by an account of the execution of QSI when fed a single input behavior (Table B.XVIII) describing the height of an upward thrown ball which rises for a while and then falls back.

After an empty initial model, derivative postulation will result in the extended behavior of Table B.XIX, 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 XXIII for the execution times of the examples of this section.

B.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 Section 4.) 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 B.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 B.XX thru B.XXII.

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 B.XXI exist, among which the heuristics have to make a preference: 1) P negative and fixed, 2) P negative and increasing, and 3) P negative and decreasing. Of these, (1) will be chosen because of the heuristic which tries to keep the order of the model as small as possible. However, (2) is the behavior which fits our understanding of the system, and the expected model will not be found.

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 B.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 B.XXIII. The derivative of X is to be postulated by QSI.

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 B.XXIV.)

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 B.XXV.

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 B.XXIV being finally selected.

Thanks are due to Bradley Richards, Levent Akın, Yorgo Istefanopulos, Kuban Altınel, Necati Aras, Ethem Alpaydın and Nadir Yücel for their comments on this work. We are grateful to the anonymous referees for their helpful remarks. Those interested may obtain the PROLOG source code of the QSI program in magnetic media from the authors for research purposes.

REFERENCES

1. J. Amsterdam, Automated qualitative modeling of dynamic physical systems, Technical Report, MIT AI Laboratory (1993).
2. R. Bhaskar and A. Nigam, Qualitative physics using dimensional analysis, Artif. Intell. **45** (1990) 73-111.
3. I. Bratko, S. Muggleton and A. Varsek, Learning qualitative models of dynamic systems, in: Proceedings Eighth International Workshop on Machine Learning (Morgan Kaufmann, San Mateo, CA, 1991) 385-388.
4. E. Charniak and D. McDermott, Introduction to Artificial Intelligence (Addison-Wesley, Reading, MA, 1985).
5. E. Coiera, Generating qualitative models from example behaviours, Technical Report 8901, Department of Computer Science, University of New South Wales (1989).
6. D. DeCoste, Dynamic across-time measurement interpretation, Artif. Intell. **51** (1991) 273-341.
7. J. de Kleer and J. S. Brown, A qualitative physics based on confluences, Artif. Intell. **24** (1984) 7-83.
8. R. J. Doyle, Reasoning about hidden mechanisms, in: Proceedings Eleventh International Joint Conference on Artificial Intelligence, Detroit, Michigan (1989) 1343-1349.
9. D. Dvorak and B. Kuipers, Model-based monitoring of dynamic systems, in: Proceedings Eleventh International Joint Conference on Artificial Intelligence, Detroit, Michigan (1989) 1238-1243.
10. S. Dzeroski, Learning qualitative models with inductive logic programming, Informatica, **16** (4) (1992) 30-41.
11. S. Dzeroski and L. Todorovski, Discovering dynamics: From inductive logic programming to machine discovery, Journal of Intelligent Information Systems (To appear).
12. B. Falkenhainer, A unified approach to explanation and theory formation, in: J. Shrager and P. Langley, eds., Computational Models of Scientific Discovery and Theory Formation (Morgan Kaufmann, San Mateo, CA, 1990) as Chapter 6.
13. K. D. Forbus, Qualitative process theory, Artif. Intell. **24** (1984) 85-168.

14. K. D. Forbus, Interpreting observations of physical systems, in: D. S. Weld and J. de Kleer, eds., Readings in Qualitative Reasoning About Physical Systems (Morgan Kaufmann, Los Altos, CA, 1990).

15. P. Fouché and B. J. Kuipers, Reasoning about energy in qualitative simulation, IEEE Trans. on Systems, Man, and Cybernetics **22** (1992) 47-63.

16. D. Halliday and R. Resnick, Physics (John Wiley and Sons, New York, 1978).

17. Y. Iwasaki and H. A. Simon, Causality in device behavior, Artif. Intell. **29** (1986) 3-32.

18. B. Kuipers, Commonsense reasoning about causality: Deriving behavior from structure, Artif. Intell. **24** (1984) 169-203.

19. B. Kuipers, Qualitative simulation, Artif. Intell. **29** (1986) 289-338.

20. B. Kuipers, Abstraction by time-scale in qualitative simulation, in: Proceedings Sixth National Conference on Artificial Intelligence, Seattle, WA (1987) 621-625.

21. B. Kuipers, Qualitative simulation as causal explanation, IEEE Trans. on Systems, Man, and Cybernetics **17** (1987) 432-444.

22. B. Kuipers, Qualitative reasoning: Modeling and simulation with incomplete knowledge, Automatica **25** (1989) 571-585.

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

24. P. Langley, H. Simon, G. Bradshaw and J. Zytkow, Scientific Discovery: Computational Explorations of the Creative Processes (The MIT Press, Cambridge, Mass., 1987.)

25. N. Lavrac and I. Mozetic, Methods for knowledge acquisition and refinement in second generation expert systems, SIGART Newsletter 108 (1989) 63-69.

26. L. Ljung, Issues in system identification, IEEE Control Systems **11** (1991) 25-29.

27. R. M. O'Keefe, The role of artificial intelligence in discrete-event simulation, in: L. E. Widman, K. A. Loparo and N. R. Nielsen, eds., Artificial Intelligence, Simulation and Modeling (John Wiley and Sons, New York, 1989).

28. S. Ramachandran, R. J. Mooney and B. J. Kuipers, Learning qualitative models for systems with multiple operating regions, in: Proceedings QR-94 (1994)

29. B. L. Richards, I. Kraan and B. J. Kuipers, Automatic abduction of qualitative models, in: Proceedings Tenth National Conference on Artificial Intelligence (1992) 723-728.

30. J. Rothenberg, The nature of modeling, in: L. E. Widman, K. A. Loparo and N. R. Nielsen, eds., Artificial Intelligence, Simulation and Modeling (John Wiley and Sons, New York, 1989.)

31. A. C. C. Say, Qualitative system identification, Ph.D. Thesis, Boğaziçi University, İstanbul (1992).

32. A. C. C. Say and S. Kuru, Nitel model tanılama, in: Proceedings Bilkon-91 1991 Bilkent Elektrik-Elektronik ve Bilgisayar Mühendisliği Konferansı, Ankara, Turkey (1991) 363-366.

33. A. C. C. Say and S. Kuru, An algorithm for qualitative system identification, in: Proceedings of the Seventh International Symposium on Computer and Information Sciences (ISCIS VII), Antalya, Turkey (1992) 229-235.

34. A. C. C. Say and S. Kuru, Improved filtering for the QSIM algorithm, IEEE Trans. on Pattern Analysis and Machine Intelligence **15** (1993) 967-971.

35. D. S. Weld, Comparative analysis, Artif. Intell. **36** (1988) 333-373.

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

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

38. B. C. Williams, Qualitative analysis of MOS circuits, Artif. Intell. **24** (1984) 281-346.

39. B. C. Williams, MINIMA: A symbolic approach to qualitative algebraic reasoning, in: Proceedings AAAI-88, Saint Paul, Minn. (1988) 264-270.

40. Xu C. W. and Lu Y. Z., Fuzzy model identification and self-learning for dynamical systems, IEEE Trans. on Systems, Man, and Cybernetics **17** (1987) 683-689.

FOOTNOTE

1. Note the parallels to Newton's first law and de Kleer and Brown's canonicality heuristics [7].

TABLE I. The Qualitative Constraint Types

Constraint	Explanation
ADD(X,Y,Z)	Z(t) = X(t)+Y(t)
DERIV(X,Y)	dX/dt = Y
M+(X,Y)	X(t) = f(Y(t)), where f'>0 throughout
M-(X,Y)	X(t) = f(Y(t)), where f'<0 throughout
MINUS(X,Y)	X(t) = -Y(t)
MULT(X,Y,Z)	Z(t) = X(t) * Y(t)

TABLE II. 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

TABLE III. 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)	

TABLE IV. 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)	

TABLE V. 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>

Tables VI, VII and VIII are missing from the electronically available version.

Sorry for the inconvenience.

TABLE IX. QDE of spring/block system

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

TABLE X. 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 XI. 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	

TABLE XII. 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*, ∞}

TABLE XIII. 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)

TABLE XIV. 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)	

TABLE XV. Behavior of amount_A

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

TABLE XVI. 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

TABLE XVII. Possible behavior for PX

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

TABLE XVIII. 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: $\{-\infty, lm1, 0, \infty\}$

TABLE XIX. 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 XX. Possible system behavior for input of Table XV

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

TABLE XXI. Input behavior of X-Y system

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

EQU

TABLE XXII. 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*, ∞}

TABLE XXIII. Execution times of QSI case runs

Problem[a]	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
B.1.2	3	1	1	5	9	1.18
B.1.3	3	2	1	3	6	0.72
B.1.4	4	2	1	17	41	42.51
B.1.5	5	3	1	6	13	19.14
B.1.6	5	1	2	2	2	1.01

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

TABLE XXIV. 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(netflow(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.}

TABLE XXV. 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(netflow(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 XXIV.

All constraints have CVs at the “equilibrium” values.}

TABLE XXVI. 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

TABLE A.I. Possible directions of A, B and C

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

TABLE B.I. 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,∞), inc>	<*lm5*, std>
B^2	<0, inc>	<(0,∞), inc>	<*lm6*, std>	<(0, ∞),dec>	<(0,∞), dec>	<*lm6*, std>

TABLE B.II. 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 B.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 B.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)	

TABLE B.III. Input of bathtub identification

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

TABLE B.IV. Old and postulated parameters for bathtub identification

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

TABLE B.V. Output of bathtub identification

CONSTRAINT	CVs
M+(amount,P1)	$(0,0),(\infty,\infty),(-\infty,-\infty)$
DERIV(P1,P2)	
M+(P2,P3)	$(0,0),(\infty,\infty),(-\infty,-\infty)$
M+(amount,P4)	$(0,0),(\infty,\infty),(-\infty,-\infty)$
DERIV(P3,P4)	
M-(amount,P2)	$(0,0)$
M+(P2,P5)	$(0,0),(\infty,\infty),(-\infty,-\infty)$
M+(amount,P6)	$(0,0),(\infty,\infty),(-\infty,-\infty)$
MULT(P5,P5,P6)	

TABLE B.VI. Input for filling bathtub identification

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

Table B.VII is not present in the electronically available version.

TABLE B.VIII. Sufficient QDE for filling bathtub identification

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

TABLE B.IX. Output of filling bathtub identification

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

TABLE B.X. First input behavior for water balance identification

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

TABLE B.XI. 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 B.XII. 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 B.XIII is not present in the electronically available version.

TABLE B.XIV. 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 B.XV. 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 B.XVI. 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>

TABLE B.XVII. 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)	

TABLE B.XVIII. Behavior of ball height

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

Table B.XIX is not available, sorry.

TABLE B.XX. 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 B.XXI. 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 B.XXII. 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>

TABLE B.XXIII. 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>

TABLE B.XXIV. 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>

TABLE B.XXV. 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>