QPE: Using assumption-based truth maintenance for qualitative simulation

Kenneth D. Forbus

Qualitative Reasoning Group, Department of Computer Science, University of Illinois, 1304 W. Springfield Avenue, Urbana, Illinois 61801, USA

Efficient qualitative simulators are crucial to continued progress in qualitative physics. Assumption-based truth maintenance systems (ATMS) were developed in part to simplify writing such programs. This paper identifies several abstractions for organizing ATMS-based problem-solvers which are especially useful for envisioning. In particular, we describe the *many-worlds database*, which avoids complex temporal reference schemes; how to organize problem-solving into *justify/assume/interpret cycles* which successively construct and extend partial solutions; and *closed-world tables*, which provide a mechanism for making closed-world assumptions. We sketch the design of the *Qualitative Process Engine*, QPE, an implementation of Qualitative Process theory, to illustrate the utility of these abstractions. On the basis of our experience in developing QPE and analysing its performance, we draw some general conclusions about the advantages and disadvantages of assumption-based truth maintenance systems.

Key Words: qualitative simulation, envisioning, assumption-based truth maintenance.

1. INTRODUCTION

Qualitative physics has made substantial progress in modelling simple situations. But continued progress will require tackling larger situations, higher fidelity models, and by building systems which use qualitative simulation as a module in a larger task. All of these research directions require substantially more computation. In fact, existing models often strain current computers. While advances in hardware technology will provide some of the needed power, we must also seek better implementation techniques. This paper describes how the use of assumption-based truth maintenance in qualitative simulations can substantially improve performance and reduce system complexity, without imposing onerous restrictions.

1.1 Qualitative simulation: The problem

Qualitative simulation programs input a domain model and a scenario encoded in that domain model, and produce a description of possible behaviours. Since qualitative reasoning is fundamentally ambiguous, there are often several possible behaviours. *Envisioning* is the process of generating a description of all possible behaviours. Representing and reasoning with these various possibilities is complex. One common simplification is to examine only a single possible behaviour, often chosen by the user[27,37,40]. This may suffice for some tasks, particularly when other sources of information are available, but often this limitation is unacceptable. For example, in designing complex physical systems exploring all behaviours may be necessary to reveal all failure modes. A behaviour that the designer did not choose to explore (or that was pruned from consideration by some heuristic) could hide a potential catastrophe.

Envisioning is also an important methodology in developing new qualitative models of a domain. The domain model provides a vocabulary for describing a wide variety of scenarios. For example, a domain model for thermal control systems should be useful for describing a wide variety of particular systems. For each such scenario, the qualitative model should be capable of generating all the behaviours which are possible under some particular choice of numerical parameters for the system*. By exploring the entire space of behaviours predicted by the model for a variety of examples, we can gain some confidence that our models are correct. Consequently, efficient envisioners are essential to continued progress in qualitative physics.

The more of the modelling burden a theory takes on, the harder it is to implement. QSIM[28], for example, is simple, small, and fast. However, it also provides very little. Its input is a model for a particular scenario, in terms of qualitative equations. There is no provision for creating the equations automatically from a structural description, nor for creating and maintaining a library of abstract descriptions to simplify building future models. QSIM was written to explore qualitative mathematics and landmark introduction, and it is indeed quite useful for that. However, to develop the kind of wide-coverage qualitative physics we desire, more powerful implementations are required.

The device-centred ontology[10,40] provides more facilities for the modeller. System dynamics, after all, is a well-developed methodology in engineering disciplines, and quantitative models for a number of domains have been developed which can be translated into qualitative

* Not all paths through an envisionment necessarily correspond to physically possible behaviours[28], but this only means that generating histories from envisionments is more complicated than first thought[9].

terms. However, device-centred theories have difficulty capturing many natural phenomena, such as objects whose existence can change (e.g., steam appearing and water vanishing in a boiler), or systems where the connectivity between parts changes (e.g., bouncing balls).

A more subtle limitation is that device-centred theories do not formalize the critical step of moving a description of objects and relationships in the world to an appropriate abstract description in models terms. This limitation is subtle because for many domains of interest the modelling step is straightforward. Electronic components, for example, are pretty good approximations to their idealizations for low frequencies, and there is a simple one-to-one mapping from real world objects to the vocabulary of devices. But in many domains the mapping is not so obvious. A simple metal plate, for example, can act as a spring, a mass, or a damper depending on the specifics of its parameters and the system to which it is connected to[36]. In traditional system dynamics the mapping of real-word objects to 'devices' is left to the intuition of the engineer. But providing an account of such intuitions is one of the goals of qualitative physics, so theories which do not provide a means of capturing this expertise fail to address a crucial aspect of the problem.

Qualitative Process theory[15,16] takes on this extra modelling work. Viewed as the specification of a modelling language, it provides facilities for building up general-purpose domain models, which are automatically instantiated to describe particular scenarios. It includes interfaces to external representations and ability to make explicit modelling assumptions[14]. These extra abilities provide additional challenges to the implementor. For example, an implementation of QP theory must perform the pattern matching required to find occurrences of processes and instantiate the relevant time-varying relationships, such as the potential existence of liquid in a container. Allowing changes in existence means that the set of qualitative equations governing the quantities in a system can vary, so closed-world assumptions must be made and updated during the course of simulation to track the relationships between quantities.

Efficient envisioners for a device-centred qualitative physics have existed for some time[9]. Can modelling languages based on Qualitative Process theory be implemented with similar efficiency? The answer is yes, but building them has required developing some new techniques for using assumption-based truth-maintenance systems in problem-solving. We believe these techniques are useful for general qualitative simulation and problem-solving, as well as implementing QP theory. This paper explains these techniques.

The next section describes the three major organizational ideas for using an ATMS for qualitative simulation: the *many-worlds database*, *justify/assume/interpret cycles*, and *closed-world tables*. Section 3 illustrates these ideas by sketching how they are used in the design of the *Qualitative Process Engine* (QPE), our current implementation of QP theory. Finally, we analyse QPE's performance by comparing it to GIZMO, the first QP implementation, which used a logic-based TMS. At this writing, QPE is between 7 and 16 times faster than our previous implementation (according to the setting of various mode switches, discussed below), providing some indication that these ideas are useful. Finally, we suggest some general guidelines for designing ATMS-based problem solvers, and outline directions for future research.

2. ABSTRACTIONS FOR USING AN ATMS

First, we briefly review assumption-based truth maintenance systems.

2.1 Assumption-based Truth Maintenance Systems

The simple, classical TMS uses horn-clauses for justifications, labels nodes with IN and OUT (indicating belief and lack of belief, respectively), and maintains belief in a node in terms of well-founded support. We call this kind of system a *JTMS*. Those which allow justifications to be based on other nodes not being believed will have prefix 'NM', indicating 'nonmonotonic'. The 'fact garbage collector' in EL[39] and London's original dependency system[29] are both JTMS's. Doyle's[12] TMS was the first NMJTMS. A TMS which labels nodes as TRUE, FALSE, or UNKNOWN and uses disjunctive normal form clauses as justifications will be called an 'LTMS'. The LTMS was invented by McAllester[30,31]. We will mention these again in Section 4, but for now we focus on the ATMS.

The ATMS is similar to the original JTMS. Associated with each fact in the problem-solver is a *node*, with status of IN or OUT depending on whether or not it is believed. Justifications take the form of Horn clauses, in that they have a single *consequent* and a list of *antecedents*, all of which are TMS nodes. The consequent has the status of IN when all the antecedents have the status of IN.

Unlike justification-based TMS's, the ATMS stores with each node the various sets of assumptions under which that node is believed. Each set of assumptions is called an *environment*. The environments under which some node is believed is called that node's *label*. Assumptions in the ATMS are a primitive datatype, corresponding to a choice the problem-solver can make. For the purposes of this paper we will assume that each assumption corresponds to the choice to believe a particular node. (The relationships between nodes and assumptions can be complicated[6], but this simple formulation will do for our purposes.)

Certain nodes can be marked as contradictory. Installing justifications for these nodes provides a means of stating logical constraints. Any set of assumptions which would support a contradictory node must itself be contradictory. An environment (a set of assumptions) which is marked as contradictory is called a *nogood*. Sometimes we will refer to justifications of contradictory nodes themselves as nogoods for simplicity, since their effect is to produce contradictory environments.

Justifications in the ATMS primarily serve to propagate environments. A node is IN only when it has a nonempty label, i.e., there is at least one environment in which the fact corresponding to that node is believed. Unlike other TMS's, whether or not a node is IN is usually unimportant. What typically matters in using the ATMS is whether or not a fact holds in some particular context of interest.

Environments provide the ATMS's notion of context. The set of assumptions which comprise the environment, plus their consequences, form a particular context. There is no need to copy their consequences explicitly, since the label of the fact can be used to compute whether or not a fact is in a particular context. A fact is IN in a particular

environment **E** only if there is some environment in its label that is a subset of **E**.

A *choice set* is an exhaustive collection of alternatives. The choice sets which comprise a problem-solver's search space provide the space of ATMS assumptions. The subset of environments which support the problem-solver's goals can be considered the goal states of the search space. This mapping simplifies problem-solver operations. For example, a common operation is determining how a partial solution might be consistently extended. In ATMS terms this question can be reduced to the simpler question of whether or not a particular fact **F** is consistent with a particular set of assumptions **E**. The answer is 'yes' if some environment $\mathbf{E}_1$ in **F**'s label can be consistently combined with **E***.

Given a set of choice sets, the process of *interpretation construction*[4] computes all consistent combinations of them, selecting one assumption from each set. The choices in each set are assumed to be mutually exclusive and exhaustive. Thus a simple model for arranging an ATMS-based problem solver is:

1. Build a network of nodes and justifications.
2. Identify the choice sets which form the 'basis set' for solutions, and assume each individual element.
3. Gather solutions by interpretation construction on the choice sets.

Alas, this model is too simple to be used for most problems, and we shall introduce a slightly more complex organization in Section 2.3. The *consumer architecture*[5] varies from the simple model only in that construction of nodes and justifications can be interleaved with making assumptions. The intent of the consumer architecture is to reduce the ATMS structures to just those which potentially appear in solutions, by seeing if the antecedents for a justification could be believed together before installing the consequent. As described below, consumers are insufficient (and in many cases inefficient) since they do not provide enough control for our purposes.

2.2 The many-worlds database

Fully exploiting the notion of environment as context requires organizing problem-solvers somewhat differently than for other TMS systems. In particular, it pays to return to an implicit represenation of time and situation instead of using more modern notations of situation-calculus[32] and histories[25]. Each fact in the database can be interpreted as a number of statements about individuals in several possible worlds, rather than as about individuals at some specific time or in some specific circumstance. Any particular world, be it different in time or in some other way, is defined by a particular environment. What is true in that world is represented by the set of facts which are the consequences of the defining environment.

This interpretation can be understood in terms of the following metaphor. Consider the database as a description of the physical world. In a JTMS or LTMS the world is deterministic, although perhaps not totally determined: to find out if something is true we need only look at the appropriate TMS node. By contrast, the ATMS database is something like a quantum mechanical wave function. Looking at a particular node, one sees only possibilities. But probing the database with a particular environment yields a particular answer, just as taking a physical measurement provokes the collapse of the QM wave function. Consequently, we call this organizational scheme the *many-worlds* database.

The many-worlds database has clear advantages for some kinds of problems. First, it eliminates the overhead of making copies of assertions required by explicit temporal reference schemes. For example, in QPE we write

(Left-of Block-A Block-B)

while the same fact represented in GIZMO would be

(Left-of (at Block-A S1) (at Block-B S1))

Describing different times in GIZMO required creating new assertions; describing different times in QPE requires only creating new environments. In addition to storage economy, this increases the advantage of inference caching provided by the TMS, since the same justifications serve in the structure of many states. It also eliminates the complicated pattern-matching that explicit temporal reference schemes require in order to avoid creating nonsense assertions, such as fluid paths that exist between temporally nonoverlapping objects[17,37].

The many-worlds database is essentially the database interpretation implied in the original conception of the ATMS[3], and is used in de Kleer's qualitative physics programs as well as in QPE. However, it has never been adequately explained as a strategy per se, nor does it appear to have been widely adopted. For example, it appears to be simpler than the viewpoint mechanism of ART or the Worlds mechanism of KEE[34], the two current commercial systems which use ATMS-like technology. These mechanisms are closer in spirit to CONNIVER-style databases[33], using assumptions to model markers corresponding to assertion and deletion of facts within particular named contexts. They suffer from the same limitation, namely having to work backwards up a tree of contexts in order to find the actual status of a fact, and the inability to share cached inferences across time*.

The obvious disadvantage of this scheme is that, since one cannot name specific situations and times, one cannot state explicit comparisons between them. Thus one cannot say that

A[Amount-of(at(Water-in-can,After(boiling)))]
< A[Amount-of(at(Water-in-can,Before(boiling)))]

However, the advantages gained by the many-worlds database suggest that the best way to overcome this problem is to support multiple notations, rather than to force explicit temporal references everywhere. By constructing *registrations*[21], histories can be described in terms of occurrences of qualitative states from an envisionment. Most of the information relevant to a particular episode of the history can be inherited from

* Combining environments consists of taking the union of the sets of assumptions involved. The result is considered to be contradictory if it subsumes a set of assumptions already known to be contradictory. See Ref. 4 for details.

* What these schemes do provide is the ability to retract facts from contexts. In the many-worlds database contexts are only identified with environments, so retraction does not make sense.

correspondong qualitative state in the envisionment, while still using explicit temporal notations for episode-specfic facts.

2.3 The justify/assume/interpret cycle

Recall that justifications in the ATMS determine the set of consequences which follow from any environment, and weed out inconsistent sets of assumptions. A 'solution' is a particular environment and its attendent consequences. There are a number of ways to construct such solutions[4,11,23]. A particularly useful technique for problems where the entire space of solutions is sought, such as envisioning, is to organize the problem-solver into *justify/assume/interpret cycles*.

A justify/assume/interpret cycle works like this:

1. Begin with an initial set of facts and initial partial solutions.
2. Until the problem is solved,
   - 2.1 Until the subspace of assumptions is complete,
     - 2.1.1 *Justify:* Create justifications representing the conclusions and constraints that follow from the current set of facts.
     - 2.1.2 *Assume:* Install assumptions suggested by the results so far.
   - 2.2 *Interpret:* Extend the partial solutions, based on the new conclusions and constraints.

We include the creation of special-purpose datastructures which contain pointers to ATMS constructs in the *Justify* and *Interpret* phases. The crucial difference between this organization and the simple organization discussed in Section 2.1 is the use of iteration to decompose the problem-solving activities into layers, each exploiting the partial results of the layer before it.

This scheme further requires that all pertinent information is provided as input before processing starts. That is, computations organized in this way shall be treated as atomic operations when used in a larger problem-solving system. This stipulation has two advantages. First, we can dispense with the standard ATMS consumer architecture and its associated overhead. Second, it allows us to use a form of negation by failure. We can know, for nodes representing certain classes of facts, that by a particular stage of processing that the labels of these nodes are as large as they will ever be. (Their labels could shrink, due to environments becoming nogood as a consequence of other constraints.) We will call this label the *maximal* label for the node. If such a node's label is empty, for example, then we can conclude that it is not believed under any circumstance that will lead to a complete solution, and so not consider that node further.

The **N**-queens problem provides a simple example. Consider an $\mathbf{N} \times \mathbf{N}$ chessboard. The goal is to find all the possible ways to place **N** queens on the board so that no two queens can capture each other. Let the initial facts be the set of statements about the location of the queen in the first column. The initial solutions are simply the **N** assumptions that the first queen can be legally placed in each of the **N** rows. The justify phase of the cycle consists of constructing statements corresponding to the possible locations of a queen in the second column, along with the nogoods which rule out combinations of queen placements in the first and second columns that would allow a capture. The assume phase consists of assuming that each statement about queen locations in the second column could hold. The interpret phase of the cycle extends the solutions by adding to each partial solution (i.e., the assumptions about queen locations for the first column) an assumption about the location of the queen in the second column, filtering out those possibilities which are inconsistent. The result will be a new set of environments, each consisting of consistent positions for a queen in the first column and a queen in the second column. Carrying out this procedure **N** times, constructing nogoods linking each column with all the previous columns, followed by extending the solution environments, will generate all solutions to the **N**-queens problem.

A single justify/assume/interpret cycle corresponds to the simple ATMS organization described in Section 2.1. For complex problems, the simple model results in grotesque combinatorial explosions during interpretation construction. By decomposing the program's operation into several distinct justify/assume and justify/interpret cycles such explosions can usually be avoided. The partial solutions from each stage can guide the construction of justifications, nogoods, and choice sets at the next stage. Consider, for example, a program which generates feasible plans for travelling from Urbana to Palo Alto. The choices in such a plan include the means of transportation (such as airplane, train, bus, or car) and the particular path (highway, track, flight number, etc.). A set of plans can be generated in principle by turning loose a set of pattern-directed inference rules (as in AMORD[8], RUP[31], DEBACLE[17]) to construct all the relevant justifications and nogoods, and then using interpretation construction to find all consistent plans. However, in practice this technique does not work very well. Suppose the choice of transportation mode was made last and that time constraints rule out any solution but flying. The interpretation construction process has generated all possible paths by train, bus, and car – an enormous set of possibilities – when in fact all of these choices are irrelevant.

The problem can be worse than mere inefficiency because sometimes the intermediate results of interpretation construction are larger than can fit in the address space of the machine, even though the final set of solutions fits quite comfortably. We call this problem *intermediate interpretation bulge*, by analogy with a similar problem in symbolic algebraic manipulation systems*.

A uniform problem-solver organization does not adequately exploit the logical dependencies between the choices involved in a solution. By breaking the solution process into distinct phases, the combinatorial explosion can be abated. In this example, considering the mode of transportation first would drastically reduce the number of solutions, since there is no reason to work on any solutions which assume the mode is any thing but flying. Similar dependencies appear in most kinds of problem-solving. In Qualitative Process theory, for example, there is no reason to calculate the effects of two processes taken together if they can never consistently be active at the

* In symbolic algebra systems, *intermediate expression swell*[35] refers to the problem of the size of intermediate expressions growing larger than can fit in memory, even though the final answer does fit. Usually an alternate solution path would have yielded the same solution but with smaller intermediate expressions.

same time. Although doing this work does not affect the correctness of the final answer, it is wasted effort.

In general there are two ways to minimize intermediate interpretation bulge. The first is to minimize the total number of assumptions (i.e., program with 'Occam's machete'). The second is to ensure that nogoods are found as quickly as possible, so that large environments which will eventually be ruled out are never created in the first place. The justify/assume/interpret organization supports both techniques. By decomposing the assumption-making part of the computation into distinct justify/assume cycles, irrelevant assumptions can be avoided. By decomposing the interpretation phase we can more easily avoid building large intermediate solutions which will eventually be thrown away as irrelevant. We have found empirically that in qualitative physics problems, careful decomposition of the program into distinct justify/assume/interpret cycles can mean the difference between a problem being solved quickly and using little memory, versus running for hours without finding solutions.

2.4 Closed-world tables

Closed-world assumptions are ubiquitous in problem-solving. A detective must figure out who had access to a bank account in order to solve a case of embezzelment. A scientist must sort through a mass of observations and decide which of them form a reasonable basis for theory generation. In daily life, we all use plausible reasoning to quickly come to conclusions. In QP theory, closed-world assumptions are essential to combine partial descriptions into a complete description of a scenario. An implementation of QP theory must assume that, for example, it knows all the influences on a quantity in order to figure out how it will change.

All of these examples of closed-world assumptions can be represented in terms of assumptions about set membership. In particular, we assume that the members of the set we know about are the only members. Then we can carry on with whatever computation needs to be performed. For example, suppose we know that two processes are directly influencing a quantity Q. If we assume those two are the only influences and that each contribution is positive then we can conclude $D[Q] >$ ZERO*.

Implementing such assumptions in a problem-solver that allows new information to be added at any time is tricky, but possible[17]. We can exploit the justify/interpret cycle organization to simplify this operation. For each set requiring a closed-world assumption, there is some stage in the solution process during which all the information required to determine it is known. That is, the labels for the statements implying membership will be maximal, as defined previously. If the set is closed at that point, only construals of the set (i.e., hypotheses about the members of the set) which belong to some consistent solution will be generated.

To carry out this computation we need to cache the possible members of the set. That information is provided by *closed-world tables*. A closed-world table is a collection of entries whose form is

$$(\langle antecedents \rangle \cdot \langle member \rangle)$$

Each $\langle member \rangle$ is a potential element of the set. The $\langle antecedents \rangle$ are a set of statements whose conjunction implies $\langle member \rangle$ is in the set. Thus if process instance PI3 introduced a direct influence $N1$ on $Q1$, then ((Active(P13)). $I + (Q1, N1)$) would be in the closed-world table for the direct influences on $Q1$. We assume (implicitly) that the set of members listed in the closed-world table are the only potential members of the set.

A closed-world table can be used as soon as the labels for the antecedents are maximal. Since the antecedents are either elements of choice sets or consequences of some choice set elements, this means placing the computation in the justify phase of a cycle which is after the assume phase of the cycle which establishes the last of the antecedents. In QPE, for example, the potential sets of influences may be calculated as soon as all of the assumptions concerning preconditions and quantity conditions are made, since these determine which processes and views are active, which in turn determine what direct and indirect influences hold.

To construct all of the possible construals of a set requires systematically computing all consistent combinations of antecedents. Each particular membership statement is then justified by the union of the antecedents for the potential elements which are in that particular construal and the disjunction of the negation of the antecedents for the potential elements which are not in that particular construal of the set.

For example, consider the subproblem of figuring out all the possible ways that the amount of water in container G of Fig. 1 can change. P_1, P_2, P_3, P_4 are instances of liquid flow, with their direction indicated by arrows in the figure. If these processes are the only ones, then the amount will tend to increase when P_2 or P_3 are occurring, and tend to decrease when P_1 or P_4 are occurring. (Influence resolution is discussed further in Section 3.6.) The closed-world table for these influences is shown in Fig. 2.

To solve this problem, we first need to find out which combinations of P_1, P_2, P_3, P_4 can be active at the same time, and hence which combinations of effects actually occur. Then for each combination we must determine the net result. This computation is depicted in Fig. 3. When nothing is happening (i.e., all pressures are equal), the set of influences is empty. There are four ways for just one process to be active (a state of affairs that won't last long, of course), and in this case each provides just one influence. The pairs of processes are more constrained: flows are active when there is a pressure differential, so clearly P_1 and P_2 cannot be active at the same time, nor can P_3 and P_4. All other pairs of processes are consistent, so the sets of influences they give rise to become solutions in turn. Finally, every triple of processes leads to a contradiction, because it contains one of the pairs of processes which cannot be active together. No larger combination can be consistent if the triples are not, so we can stop generating sets of influences at this stage.

Now consider each construal of the set of influences. When there are no influences the Ds value is 0, corresponding to no change. The antecedent for this condition is that none of P_1 through P_4 are active. When all the influences are positive or negative the Ds value takes on that sign. When the influences are in both directions the system branches, installing assumptions about the relative magnitudes of rates. Fig. 4 illustrates this. Importantly, since the results depend only on the

* Some QP notation: A quantity consists of an amount, denoted A, and a derivative D, both of which are numbers, under the usual interpretation. The sign of the derivative is denoted Ds.

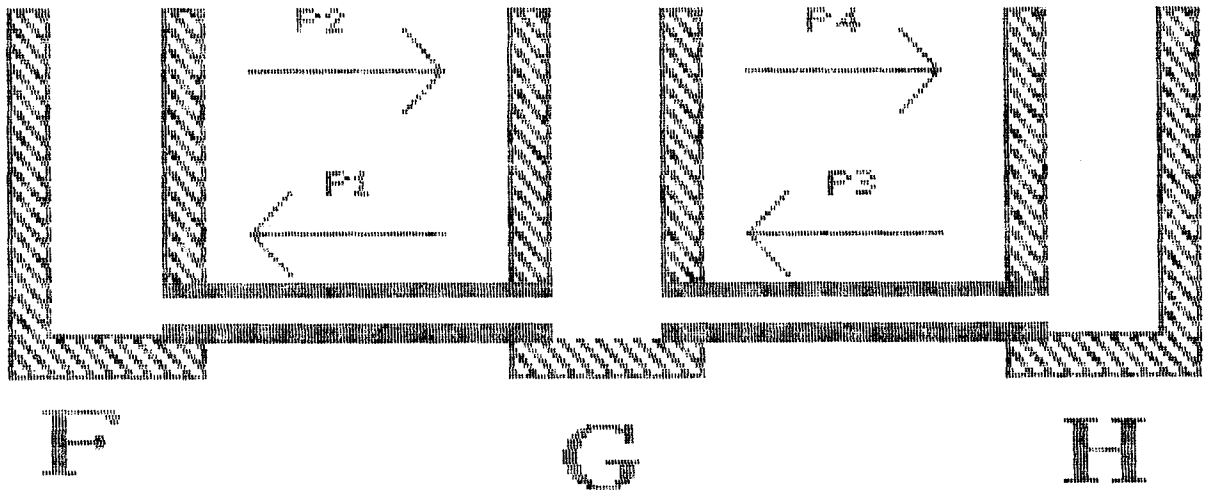

Fig. 1. A subproblem in influence resolution. Each potential flow process is indicated by a labelled arrow. By determining which combinations of flows can be active at any time, we can compute the derivative of the amount of liquid in G

```
AMOUNT-OF-IN(WATER,LIQUID,G) :
    ACTIVE(P1),
contributes I-(AMOUNT-OF-IN(WATER,LIQUID,G),A[FLOW-RATE(P1)]).
    ACTIVE(P2),
contributes I+(AMOUNT-OF-IN(WATER,LIQUID,G),A[FLOW-RATE(P2)]).
    ACTIVE(P3),
contributes I+(AMOUNT-OF-IN(WATER,LIQUID,G),A[FLOW-RATE(P3)]).
    ACTIVE(P4),
contributes I-(AMOUNT-OF-IN(WATER,LIQUID,G),A[FLOW-RATE(P4)]).
```

Fig. 2. Closed-world table for Amount-of-in(C-S(*water, liquid*,G)). *The combinations of flow process which can occur together determine all possible ways the amount of liquid in* G *can be changing. By using closed-world assumptions, we allow a local conclusion to be correctly applied across an arbitrary number of situations. This excerpt from a QPE dump for this example shows the appropriate closed-world table*

Step #1: {}

Step #2: $\{P_1\}$, $\{P_2\}$, $\{P_3\}$, $\{P_4\}$

Step #3: $\{P_1, P_3\}$, $\{P_1, P_4\}$, $\{P_2, P_3\}$, $\{P_2, P_4\}$

Step #4: Finished – all larger combinations would subsume nogoods

Fig. 3. Finding consistent sets of influences

information in the closed-world table, we have now solved this sub-problem for *every* situation in which Amount-of(*C-S*(water,liquid,*G*)) is directly influenced. The ATMS ensures this answer will be available in every context which requires it.

3. THE QUALITATIVE PROCESS ENGINE

In this section we first briefly describe ATMoSphere, the inference engine underlying QPE. Then, we describe how the organizational ideas of the previous section can be used to efficiently perform the computations sanctioned by QP theory. We begin with an important sub-problem, reasoning about inequalities. We then show the flow of information and processing in QPE by stepping through a simple example.

3.1 The ATMoSphere language

QPE is built on top of ATMoSphere, a problem-solving language which provides pattern-directed rules and a clean interface to the ATMS*. The details of ATMoSphere are not important here, but understanding some general features of the language will aid in understanding subsequent sections.

Inside this ATMS data is organized into *variables* which take on *values*, reflecting its heritage as support for a constraint language. We maintain this data orgnaization in ATMoSphere, treating propositions as functions whose range is the set {:TRUE;:FALSE}. This convention is useful because equations involving functional terms are common in qualitative physics (i.e., $Ds[Q1] = -1$). However, general equality reasoning, especially substitution of equals for equals, is not supported (unlike RUP[31]). In out experience, such computations are rarely worth the language constraints required to support them. Users are insulated from this data model if desired, by appropriate syntaxers and printers.

ATMoSphere provides pattern-directed antecedent inference rules, similar to those of RUP[31] and DEBACLE[17], for automatically installing justifications and nogoods as required. Implementing justify/assume/interpret cycles requires rules which trigger on facts being mentioned, rather than being believed (the :INTERN condition). By using :INTERN rules, we can build a complete subnetwork of justifications and constraints first, and then propagate the consequences of assumptions, and perform interpretation construction on that subnetwork. ATMoSphere also supports other rule

* ATMoSphere was developed in collaboration with Johan de Kleer of Xerox PARC.

Active	{}	{P1}	{P2}	{P3}	{P4}	{P1, P3}	{P1, P4}	{P2, P3}	{P2, P4}
Ds value	0	-1	1	1	-1	-1, 0, 1	-1	1	-1, 0, 1

Fig. 4. Summary of influence resolution results

strategies, including the standard ATMS consumer architecture and the *implied-by* strategy[23]. Like DEBACLE, ATMoSphere also supports integration of special-purpose datastructures and procedures with the assertional databases and rules. For critical computations like inequality reasoning this speed-up is essential.

3.2 Reasoning with inequalities

Efficient reasoning about inequalities is a central problem in implementing qualitative physics programs[7,17,27,37,40,41]. The key inferences required of an inequality reasoning system concern *transitivity*, such as:

$$\mathbf{A}>\mathbf{B} \wedge \mathbf{B}>\mathbf{C} \Rightarrow \mathbf{A}>\mathbf{C}$$
$$\mathbf{A}=\mathbf{B} \wedge \mathbf{B}=\mathbf{C} \Rightarrow \mathbf{A}=\mathbf{C}$$
$$\mathbf{A}>\mathbf{B} \wedge \mathbf{B}=\mathbf{C} \Rightarrow \mathbf{A}>\mathbf{C}$$

The standard strategy for reasoning with inequalities is to create a graph where the nodes are numbers and the links are inequality relationships. When a query about the relationship between two numbers is made, a simple search provides the answer. Typically, a TMS is used to cache the dependence of the relationship found on other links in the graph[17,38]. While efficient for problem-solvers that only examine one state at a time, this scheme requires modification for an envisioner. Ideally, we want to have inequality information available incrementally during our computations, since transitivity is a powerful filter, but we also want to avoid performing this computation once per situation, as the consequent reasoning strategy requires.

We have explored a number of strategies for efficient inequality processing within an ATMS, and have settled on an algorithm with certain novel aspects. Suppose there ultimately will be **N** numbers. In practice, few of the potential $\mathbf{N}^2$ relationships between these numbers are of interest. Pairs of numbers which are 'interesting' will be mentioned at some point in an inequality relationship by the problem-solver. Consequently, an efficient inequality algorithm should not introduce ordering relationships between pairs of numbers which have not already been deemed of interest by the problem solver. Since inequalities are a substantial portion of the fabric from which qualitative states are woven, many of these inequalities will be assumptions. We want to find out quickly which combinations of them are inconsistent, since such combinations cannot be part of any state. This suggest a forward-chaining, incremental algorithm.

Whenever a pair of numbers is mentioned in some inequality relationship, we build a compound link which contains representations for *all* possible relationships between that pair of numbers. In other words, associated with each link are ATMS nodes corresponding to the pair of numbers being $>$, $<$, $=$, or $\downarrow$. (In QP theory objects and their properties can have limited temporal extent, i.e., they can ease to exist. $N_1 \downarrow N_2$, read 'N_1 is *unrelated to* N_2', indicates that the quantity of which N_1 or N_2 belongs to does not exist.)

The key observation is that every inference involving transitivity occurs in cycles in this graph of links. Given a new pair of interesting numbers, we incrementally install the consequences of transitivity involving the pair by (a) finding every cycle in the graph of links which includes this new link, and (b) installing a set of justifications which enforces transitivity in each newly-found cycle. The set of justifications consists of concluding some relationship for a link in the cycle based on consistent combinations of relationships from the other links in the cycle. For instance, consider the cycle $\mathbf{A} \leftrightarrow \mathbf{B} \leftrightarrow \mathbf{C} \leftrightarrow \mathbf{A}$. When this cycle is first created, the consequences shown at the beginning of this section will be among those installed*.

Since all transitivity consequences of a new number pair are installed antecedently, queries are free, which simplifies subsequent computations. We have explored a wide variety of ATMS-based transitivity algorithms, and this has given us the best results to date.

3.3 QPE: Input, output, and organization

QPE takes two inputs, a domain model and a particular scenario described in terms of that model. For concreteness, we will use the scenario depicted in Fig. 5. Here, a container called Can is connected thermally to a heat source Stove via a heat path Burner. The only kind of substance is water, which can be in either the liquid or gas state. Notice that we have not specified any equations or processes. It is part of QPE's job to generate those, based on the domain model. We will say more about this procedure below. The output is an envisionment, consisting of a set of *situations* and transitions between them. An *attainable* envisionment consists of all states that can be reached by some set of transitions from a distinguished initial state, and a *total* envisionment consists of the union of attainable envisionments from every possible initial state. A situation is a qualitative state representing some class of behaviours. Each situation is represented as an ATMS environment. The facts which hold in that situation are simply the facts implied by the environment.

QP theory provides a set of basic deductions which can be woven together to perform particular tasks. The simplest way to generate histories given an initial state, for instance, is

1. Find what processes and views are active in the current situation.
2. Resolve influences to determine *Ds* values.
3. Perform limit analysis to determine what transitions are possible.
4. Select one of these transitions as corresponding to the next state.
5. Until finished, go to 1.

Attainable envisionments may be generated by carrying out the same procedure, but by exploring all transitions and 'collapsing' states which look identical into single states. This in fact is the algorithm used in GIZMO[17]. This algorithm is inadequate for generating total envisionments. In total envisionments there is no distinguished initial state. *Every* state consistent with the given scenario must be generated. For example, if the

* The number of justifications installed per cycle can be greatly reduced by either using $\leqslant$ & $\geqslant$ or equivalently by using negation.

```
;;; Background information
(assert! '(substance water))          ; "water" is a substance
(assert! '(state liquid))             ; "liquid" is a state
(assert! '(state gas))                ; "gas" is a state
;;; Describe the objects
(assert! '(Container Can)))           ; "Can" is a container.
(assert! '(Heat-Source Stove)))       ; "Stove" is a heat source.
(assert! '(Heat-Path Burner))         ; "Burner" is a heat path.
;;; Relationships between them - bidirectional thermal contact
(assert! '(Heat-Connection Burner Stove (c-s water gas can)))
(assert! '(Heat-Connection Burner Stove (c-s water liquid can)))
```

Fig. 5. A QPE scenario

purpose of the envisionment is to support measurement interpretation[20], determining the initial state is part of the problem, so we cannot start with an attainable envisionment.

Total envisioning requires a different organization, and it turns out this re-organization gains us considerable efficiency. For instance, in the algorithm above influences are resolved on each quantity once per situation. But, as we saw earlier in Section 2.4, many of these inferences rely on tightly constrained subsets of a total solution. Thus the ATMS techniques described here allow us to resolve each quantity once for whatever patterns of influence exist, and 'inherit' that solution whenever it is consistent, independently of the number of total solutions. This significantly reduces the computational complexity of envisioning.

QPE is organized as a collection of several justify/assume/interpret cycles. Each cycle adds a new kind of information to the solutions, i.e., the situations and transitions table. The broad steps are:

1. Expanding the scenario model.
2. Installing initial assumptions.
3. Resolving unambiguous influences.
4. Constructing initial situations and Sclasses.
5. Resolving ambiguous influences.
6. Performing limit analysis.

Next we outline how each step works, showing how the organizational ideas interact to make an efficient envisioner.

3.4 Expanding the scenario model

As mentioned above, QP theory takes on more modelling work than other theories of qualitative physics. Thus a QP implementation must determine what parts of the given domain model to apply to the initial description of the scenario. This involves ascertaining what individual views are applicable, what instances of processes exist, and installing their enabling conditions and direct consequences. We call this step *expanding* the scenario model.

To understand how this step works requires describing how domain models are implemented. The QP modeller writes descriptions like those of Figs 6, 7, and 8. Forms like defview and defprocess are automatically transformed into two ATMoSphere intern rules. The first rule finds instances of views or processes of that type, creating named descriptions (such as PI3) to stand for particular instances. The second rule installs the consequences and constraints associated with instances of that type, such as the conditions under which it is active and what follows from that. (Other forms, like defentity, define type predicates. A complete listing of domain model constructs can be found in the QPE Manual[24].)

The many-worlds database interpretation allows a single process or view instance to be used for every situation, rather than re-running rules to determine for each new situation what instances occur. The primitives in these descriptions (such as arithmetic operators and qualitative proportionalities) are expanded at rule compilation time by a syntaxer to allow various internal actions to be hidden from the modeller. For instance, several actions are required to instantiate a $\propto_{Q+}$. First, it

```
(defentity physob
  ;;; Random physical objects have several continuous properties
  (quantity (Amount-of ?self))        ;; amount of it there is
  (quantity (Heat ?self))             ;; internal energy
  (quantity (Temperature ?self))      ;; other standard thermo parameters
  (quantity (Pressure ?self))
  (quantity (Volume ?self))
  ;;; There are a few state-independent relationships
  (Qprop (Temperature ?self) (Heat ?self))
  (not (less-than (A (Amount-of ?self)) ZERO))
  (not (less-than (A (Temperature ?self)) ZERO)))
```

Fig. 6. Defining types of objects. A physob *is a generic physical object, with various thermodynamical properties*

```
(defview (Contained-Stuff (C-S ?s ?st ?c))
 Individuals ((?c :type container)
              (?s :type substance)
              (?st :type state))
 Preconditions ((Can-Contain-Substance ?c ?s ?st))
  ;; A paper cup can contain water, but not sulpheric acid
 QuantityConditions ((greater-than (A (Amount-of-in ?s ?st ?c))
                                   ZERO))
  ;; The can has a non-zero amount of that substance in that state
 Relations ((there-is-unique (C-S ?s ?st ?c))
          ; Exists only when this view active
           (Q= (amount-of (C-S ?s ?st ?c))        ; Define its amount
               (amount-of-in ?s ?st ?c))
           (physob (C-S ?s ?st ?c))))             ; It is a physob
```

Fig. 7. Defining time-varying relationships. QP theory provides individual views *to describe properties that change over time. This view defines fluid stuffs that are individuated by location*

```
(defprocess (Heat-Flow ?src ?dst ?path)
  Individuals ((?src :conditions (Quantity (Heat ?src)))
               (?dst :conditions (Quantity (Heat ?dst)))
               (?path :type Heat-Path
                      :conditions
                       (Heat-Connection ?path ?src ?dst)))
  Preconditions ((heat-aligned ?path))
    ;; No thermal insulators
  QuantityConditions ((greater-than (A (temperature ?src))
                                    (A (temperature ?dst))))
    ;; Must be a temperature difference
  Relations ((quantity flow-rate)                    ; Local quantity
             (Q= flow-rate (- (temperature ?src)     ; Constraint on flow rate
                              (temperature ?dst))))
  Influences ((I+ (heat ?dst) (A flow-rate))
                ; Acts to increase heat of destination
              (I- (heat ?src) (A flow-rate))))
                ; Acts to decrease heat of source
```

Fig. 8. A process description. Here is the definition of heat flow used by QPE in this example

must be justified on the basis of the antecedents of the description in which it appears. In the case of processes, for instance, the antecedent will be an assertion that the process is active. Furthermore, a provision is made to add an entry at run time to the closed-world table of indirect influences for the quantity constrained by that $\propto_{Q+}$. Similarly, each inequality mentioned in a domain model must be expanded into (a) justifications which force that inequality to hold when the antecedents hold, (b) an entry into tables of choice sets maintained by QPE to define situations, and (c) a form which causes the transitivity graph to be updated, if necessary. The syntaxer is data-driven, and thus provides a facility for user-supplied extensions to QPE*.

The assertions and justifications installed in this step are part of the justify phase of the first justify/assume cycle, within the first justify/assume/interpret cycle. All possible process and view instances have been found, and all of their direct consequences and constraints have been installed. In our example, there are 30 inequalities in the Q state and 4 entries in the table of preconditions. Two view instances and three process instances have been found, as shown in Fig. 9**.

3.5 Creating the initial assumptions

Perhaps the most important choice in organizing an ATMS-based envisioner is selecting which types of assumptions will comprise a situation. There are several strategies which might be used with QP theory. For instance, one might base situations on assumptions about the existence of objects and the status of views and processes. There are two problems with this alternative. First, it is not invertable – while a process must be active if all of its preconditions and quantity conditions hold, it is inactive if any of them fail to hold. Ergo in this representation we could not distinguish between situations which differ only in which conditions make some process or view inactive. Second, in subsequent processing we will need to introduce inequality assumptions (e.g., relative rates and derivative comparisons), so assumptions about existence and status assignments are insufficient.

The alternative used in QPE is to make preconditions and quantity conditions the assumptions which comprise the bulk of a situation. Typically, having assumed these completely fix the status of processes and views. We call a QP model *well-conditioned* when a set of inequalities and preconditions always suffices to establish the status of processes and views. It is possible to write *ill-conditioned* QP models, since quantity conditions can include the status of processes and views. However, we have yet to find a case where well-conditioned models are not sufficient, and so we restrict QPE to these. GIZMO operated under the same restriction*.

QPE distinguishes several kinds of preconditions. If the modeller guarantees that choices of other preconditions and inequalities will suffice to determine the validity of all predicate instances of a certain kind, the modeller can declare that predicate *computable*. Suppose Touches (⟨*stuff*⟩, ⟨*port*⟩) holds just when the contained liquid ⟨*stuff*⟩ touches the portal ⟨*port*⟩ in its container. Under the right modelling assumptions we can define Touches in terms of the relationship between the height of ⟨*port*⟩ and the level of ⟨*stuff*⟩, in which case we would declare Touches as a computable predicate. Also, the modeller can declare choice sets which QPE must explore. For instance, to force QPE to look at all the states of a particular kind of valve, the modeller can say

(Choices (State valve) (Open Closed))

QPE would then ensure that every situation assumes

```
---- View and Process Instances ----

View instances:

VI0 = CONTAINED-STUFF(WATER,CAN,GAS)

VI1 = CONTAINED-STUFF(WATER,CAN,LIQUID)

Process instances:

PI0 = HEAT-FLOW(C-S(WATER,GAS,CAN),STOVE,BURNER)

PI1 = HEAT-FLOW(C-S(WATER,LIQUID,CAN),STOVE,BURNER)

PI2 = BOILING(WATER,CAN,C-S(WATER,LIQUID,CAN),PI1)
```

Fig. 9. View and process instances for the example

* For instance, John Collins has used the parser to implement products and ratios in terms of qualitative proportionalities and correspondences.

** The domain model used here for illustration is very simple, and does not represent many of the consequences of containment. For example, boiling temperature is assumed to be constant, and the possibility of the can bursting is not considered.

* Extending QPE to ill-conditioned models seems straightforward. At minimum, it would require expanding the choice sets which define solutions as the union of all preconditions, quantity conditions, existence predications, and the statuses of views and processes. Since these choice sets are highly redundant, many more nogoods would be created and efficiency would suffer. I suspect the temporal inheritance algorithm would also have to undergo changes, but this is not obvious.

either that the valve is open or that it is closed.

The union of all inequalities mentioned in the envisionment (i.e., the contents of every possible quantity space) is called the *Qstate*. The rules derived from the domain model automatically construct tables of preconditions and quantity conditions for the current scenario during model expansion. This step makes each entry of these tables into an assumption*. By simply creating these assumptions the ATMS propagates them through justifications to update labels. Assuming well-conditioned models, the labels for the nodes corresponding to the status of views and processes are now maximal. We do not have situations yet, of course, since we have not yet performed interpretation construction to ascertain which complete combinations of choices are consistent. However, we do have enough information to use the closed-world tables associated with influence resolution, which we do next.

3.6 Resolving unambiguous influences

Recall that influence resolution computes the *Ds* values for each quantity. Some quantities are *directly influenced* by processes, when I^+ and/or I^- relationships involving them hold*. In this case, $D[Q]$ equals the sum of the influences. For instance, in our example the heat flow instance PI1 directly influences Heat(*C*-*S*(water,liquid, can)) and Heat(stove), each by A[flow-rate(PI1)]. Some quantities are *indirectly influenced*, in that they are functions of other quantities which are themselves influenced. Functional dependencies are expressed in QP theory by *qualitative proportionalities* (the operators $\propto_{Q+}$ and $\propto_{Q-}$). Both operators only provide partial information about the function relating two quantities; for instance, Temperature(?*c*) $\propto_{Q+}$ Heat(?*c*) indicates that the function which determines Temperature(?*c*) is increasing and monotonic in its dependence on Heat(?*c*), but does not indicate what the function is in detail or what else it may depend upon. Consequently, closed-world tables must be used for indirect influences as well. Quantities which are neither directly nor indirectly influenced are said to be *uninfluenced*.

Influences cannot always be successfully resolved. Conflicting indirect influences, for instance, cannot be resolved within basic QP theory since we do not know the underlying function. But in most cases resolution is straightforward, so we take care of these cases first. We defer handling ambiguous cases until after the initial situations are constructed, for reasons described in Section 3.8.

Part of the procedure for resolving direct influences was demonstrated in Section 2.4. Closed-world tables describing all direct influences for each quantity are constructed as a side-effect of expanding the domain model, and this same procedure is carried out for indirect influences as well. The conditions for a quantity being uninfluenced are established by taking the union of the negation of the antecedents in these tables. Uninfluenced quantities, by assumption, have a *Ds* value of 0. Should both direct and indirect closed-world tables for a quantity Q be empty, $Ds[Q]=0$ is asserted as universally true. Should the quantity not exist at some time, we say $Ds[Q]=\downarrow$.

As noted above, in QP theory direct influences combine by addition. This means that in some cases information about the relative magnitude of rates can allow otherwise ambiguous direct influences to be resolved. For instance, suppose that there are two flows into a container and two flows out. If we assume that each flow out is individually bigger than each flow in, then we can conclude that the sum of the out-flows is larger than the sum of the inflows, and hence the net effect is $Ds=1$. The computation for resolving direct influences can be set up to make these assumptions about relative rates. However, this is not always profitable. As the number of direct influences goes up, for instance, the percentage of cases in which pairs of inequality assumptions suffice to resolve the ambiguity drops rapidly. Consequently, QPE has several modes for handling ambiguous direct influences.

The simplest mode is to do nothing, in which case the normal constraint satisfaction process described in Section 3.8 handles the problem. The problem with this mode is that for some purposes the situation descriptions computed are not detailed enough. The obvious consequence is that you don't know when a particular collection of assumptions about rates would resolve the ambiguity without further inferential work. The more subtle consequence is that transitions involving rates can no longer be detected, since the inequalities involving rates must be explicitly mentioned before limit analysis will consider the possibility that they will change.

The second mode introduces *sum quantities* for each case of ambiguous direct influences. One sum quanity represents the net positive effect and the other represents the net negative effect. Each is qualitatively proportional to the positive and negative influences, respectively. These quantities replace old sets of direct influences, and an assumption concerning their relative magnitude is installed to distinguish between the possible outcomes. The qualitative proportionalities enable transitions between these assumptions to be detected; if in a case where the net positive effect was assumed to be larger, the positive influences are decreasing and the negative ones are increasing or remaining constant, then eventually the effects will cancel.

The third mode provides the most detailed answers, and is the most expensive computationally. All potential relationships between each pair of opposing direct influences are assumed, and that pair is added to the *Q*state. The combinations of assumptions which result in an unambiguous answer are found by a simple modification of the algorithm outlined in Section 7.2.7 in the MIT Report[17]. This algorithm worked quite well when reasoning within a single state, since the inequality information is either known or not. But making these extra assumptions is quite expensive. Suppose there are **N** positive direct influences and **N** negative direct influences on a particular quantity. The $\mathbf{N}^2$ inequality assumptions will be created, each of which becomes part of the constituents of state, and hence increases the cost of most subsequent operations. Empirically, we have found that the amount of detail produced by this mode is almost

* Assuming all possible preconditions is the default mode of operation for QPE. There is also a language for declaring particular preconditions to hold or not, so that the user (or other programs) can choose to explore only particular subsets of the total envisionment. Also, a debugging mode is proposed for asserting all preconditions as true, rather than assuming them.

* By definition, $I+$ and $I-$ are only found in the *influences* fields of processes. This means processes are ultimately the cause of all dynamical changes in the model.

never useful, but it is provided for completeness.

3.7 Constructing initial situations

The previous step comprised a second justify/assume subcycle, since (depending on the mode) we interleave adding justifications with additional assumptions. Now the stage is set for the first interpretation phase, which constructs the initial set of 'solutions', i.e., situations. The choice sets include the alternatives for each precondition and quantity condition. Any inequality assumptions made in the course of the previous influence resolution step are included as well.

Standard ATMS interpretation construction routines are used to build all consistent combination of alternatives from these choice sets. Each such combination forms an environment which is, by definition, a situation. The consequences of these environments include the situation's *process structure* (i.e., the set of active processes), the *view structure* (similarly defined) and the unambiguous results of influence resolution.

It is useful to impose additional structure on the set of situations. We define an *Sclass* to be a set of situations which are identical with respect to what objects exist in them, process structure, view structure, and *Ds* values. Sclasses divide the set of situations into equivalence classes, whose elements differ only in inequalities or preconditions. Sclasses are useful for two reasons. First, they provide a natural summarization of the envisionment – the differences between members of an Sclass are irrelevant for many inferences. Second, resolving ambiguous influences requires making assumptions, and as the next section demonstrates, this can be done more efficiently for the whole Sclass at once.

Situations are partitioned into Sclasses by finding collections with common objects, process structure, and view structure. These commonalities ensure that the influences which hold are identical for each situation in the Sclass. Consequently, the known *Ds* values are cached with the Sclass, and any quantities whose *Ds* values are ambiguous are noted.

3.8 Resolving ambiguous influences

When all *Ds* values are known, Sclass is said to be *r-complete*[17]. This step extends all Sclasses which have unknown *Ds* values into r-complete Sclasses. Basically, we do this by extending the incomplete situations with assumptions about the unknown *Ds* values. Since these assumptions are not independent, it is more efficient to add them in an order determined by direction of functional dependency: If Q_2 depends on Q_1, for example, then Q_2 should be examined after Q_1, since an assumption about $Ds[Q_1]$ may fix $Ds[Q_2]$. This suggests sorting the quantities by functional dependency*. Since the same set of influences hold for every situation in an Sclass (i.e., the set of influences is determined by the view and process structures), this sorting step can be done just once for each Sclass. In fact, the process of generating r-completions is simply a slightly more clever version of interpretation construction, using the initial situations for the initial solutions and not adding an assumption if the *Ds* value was fixed by some previous choice. The situations which result are sorted into new Sclasses and are indexed under the old Sclass as it r-completions.

3.9 Limit analysis

Limit analysis identifies potential state transitions and determines their consequences. State transitions are heralded by changes in ordering relationships between numbers; their potential occurrence is described by *limit hypotheses*. These potential transitions must be filtered carefully, to ensure that they lead to consistent states and that physical properties such as continuity are preserved. In QP theory these tests can require subtle measures, since objects can have finite temporal extents.

Suppose $A[Q_1]$ and $A[Q_2]$ are compared as part of the quantity conditions for some process instance, or to resolve an ambiguous direct influence. In addition to being marked as something to assume, during model expansion the comparison between their derivatives is noted as a property of interest. The reason is that limit hypotheses can be detected by examining the derivative comparison; for example, if $A[Q_1] > A[Q_2]$ and $D[Q_1] < D[Q_2]$, then eventually $A[Q_1] = A[Q_2]$.

Typically the derivative comparison will already be known as a consequence of the *Ds* values, but some derivative comparisons in some situations may not be known. Before limit analysis begins it is crucial to ensure that each important derivative comparison is known in every situation, which happens by extending situations as necessary with assumptions about derivative comparisons. These extensions complete the interpret phase of the first justify/assume/interpret cycle. The result is a collection of completely specified situations, organized into Sclasses. The rest of limit analysis consists of a second justify/assume/interpret cycle to determine the set of transitions linking these states.

A limit hypothesis LH is the conjecture that a particular set of inequality relationships will change in a certain way. LH is *applicable* to a situation S_i if the initial state of the inequalities mentioned in LH hold in S_i. Each application of an LH denotes a potential state transition. The first step in finding limit hypotheses is to find *single-change* limit hypotheses, i.e., those involving only a single inequality. This can be accomplished by iterating over each assumed relationship between quantities in the *Q*state, seeing which of the combinations of amount and derivative relationships that lead to a potential transition are consistent. This determines if the change is locally possible. Next this combination of inequalities is tested to see if they are part of any situation. If they are, then a limit hypothesis is created for this combination, since we are assured that it will be applicable to at least one situation.

The previous step is carried out one per scenario. The next step is to combine the single-change hypotheses to create limit hypotheses corresponding to multiple simultaneous changes*. Continuing in the 'once per scenario' spirit, the obvious algorithm to construct limit hypotheses corresponding to multiple simultaneous changes is to combine all consistent combinations of single-change limit hypotheses. Unfortunately, this algorithm turns out to be a disaster for most real problems. Many combinations of limit hypotheses are

* QP theory places two restrictions on the graph of influences believed at any time, which ensure this ordering is always well-defined[16]. First, the set of qualitative proportionalities is always loop-free. Second, no quantity may be directly and indirectly influenced at the same time.

* A common misconception is that such transitions are 'unlikely' and can be ignored. When two pairs of numbers are functionally related, often simultaneous transitions *must* occur.

```
LH0: A[TBOIL(WATER,CAN)] > A[TEMPERATURE(C-S(WATER,LIQUID,CAN))] ⟼ =
LH1: A[FLOW-RATE(PI1)] > ZERO ⟼ =
LH2: A[TEMPERATURE(STOVE)] > A[TEMPERATURE(C-S(WATER,LIQUID,CAN))] ⟼ =
LH3: A[FLOW-RATE(PIO)] > ZERO ⟼ =
LH4: A[TEMPERATURE(STOVE)] > A[TEMPERATURE(C-S(WATER,GAS,CAN))] ⟼ =
LH5: A[AMOUNT-OF-IN(WATER,LIQUID,CAN)] > ZERO ⟼ =
LH6: A[AMOUNT-OF-IN(WATER,GAS,CAN)] = ZERO ⟼ >
```

Fig. 10. Single-change limit hypotheses

locally consistent, but not applicable to any situation. Consequentely, the best next step is to find the applicable single-change limit hypotheses for each situation, and combine *them* to form consistent multiple-chage limit hypotheses. Clearly, any such hypotheses will be applicable without further testing. It is useful to keep a global cache of multiple-change limit hypotheses, both to speed up the generation process and to 'uniquize' them across situations for more coherent explanations.

The creation of limit hypotheses and their assignment to situations completes the justify phase of the second justify/assume/interpret cycle. The interpret phase consists of determining which limit hypotheses correspond to legal state transitions. For every LH applicable to each S_i, this test consists of two steps. First, the new situation S_j that results from assuming that LH occurs in S_i is generated. This is a *temporal inheritance* problem, and since QPE uses a variant of the algorithm described elsewhere[18], we will only describe how the ATMS has allowed us to simplify the algorithm.

Roughly, we 'subtract' from the set of situation assumptions those for which LH conjectures a change, and 'add' in the new inequalities that LH conjectures. Complications arise from allowing objects to have finite temporal extent (i.e., changes in existence can occur). For example, if steam appears in the can, all relationships involving its properties (such as comparing its temperature to that of the stove) must change from unrelated to something else. If all the water boils away, all of the relationships its quantities participate in become unrelated. The consistent sets of these possibilities is generated by interpretation construction. Since we have generated all consistent situations in advance, any result of this process which is not already known as a situation environment can be ruled out.

Even with these constraints, it is still possible for interpretation construction to produce more than one possible next state. The final filter is to select the state which is 'closest' to the previous state. This intuition is applied quite literally, by counting the number of assumptions in common between the candidates and the previous state, and picking the candidate with the most in common. Since the candidates differ only in whether or not certain individuals exist (assuming well-conditioned models), this always results in a unique answer.

It should be noted that constructing these possibilities and performing this filtering without an ATMS requires very complicated bookkeeping. In GIZMO, for example, this computation was carried out by carefully making and retracting assumptions in a 'scratchpad' database. Each such operation can cause propagation through a substantial part of the justification database, which in turn can lead to thrashing. No such propagation is required in the ATMS implementation.

What we have done so far is compute S_j given S_i and LH. Just because S_i and S_j are themselves consistent does not mean that LH constitutes a consistent transition between them. The transition must be tested to ensure that it does not violate continuity. This test compares the inequalities which hold between each pair of numbers in the Qstate before and after the transition. If some inequality 'jumps' from $<$ to $>$ or from $>$ to $<$, then continuity is violated and the transition is ruled out. Combinations of amount and derivative relationships are tested as well, to ensure that the mean value theorem is respected.

QPE contains two additional mode switches for experimenting with different varieties of continuity. The tests above hold for all switch settings. The first switch forbids any transition in which one pair of numbers becomes unequal at the same time another pair of numbers becomes equal. This implements a consequence of classical continuity, namely that all transitions from equality take an instant and all transitions to equality require an interval of time*. The second switch allows discontinuous changes from equality by weakening the mean value law, as was done by Williams[40]. For instance, the transition from

$$A[Q_1] = A[Q_2] \wedge D[Q_1] = D[Q_2]$$

to

$$A[Q_1] > A[Q_2] \wedge D[Q_1] > D[Q_2]$$

is allowed. The underlying intuition is that such transitions correspond to changes from an unstable equilibrium. (This switch sometimes interacts in odd ways with changes of existence, earning it the nickname of the 'spontaneous generation' switch.)

Extra work is required to ensure that continuity is not violated when changes in existence occur. Suppose we have two containers on a level surface connected by a pipe. If water is flowing from one to the other, the flow will be stopped when the pressures equalize. Limit analysis will also hypothesize that the source container may run out of water. If the water always existed, this transition would be ruled out since it violates continuity. (In any reasonable model, the pressure will be a function of the amount, and running out of water corresponds to a change in the pressure in the source from being greater than the pressure in the destination to being less than it.) However, without extraordinary measures this violation would not be detected since the water in the projected situation has vanished, taking its quantities with it! QPE solves this problem by performing special tests when a potential transition involves a vanishing quantity. Basically, it re-computes the standard continuity tests while temporarily suspending the QP laws governing the relationship between existence and quantities. This is accomplished easily in the ATMS by including with each situation the explicit assumption that these laws hold. To suspend this assumption simply requires subtracting it from the situation and inserting its negation instead.

* QP theory explicitly deviates from classical continuity, assuming an infinitesimal model for numbers. Transitions to equality where the two numbers differ only by an infinitesimal amount are assumed to take only an instant.

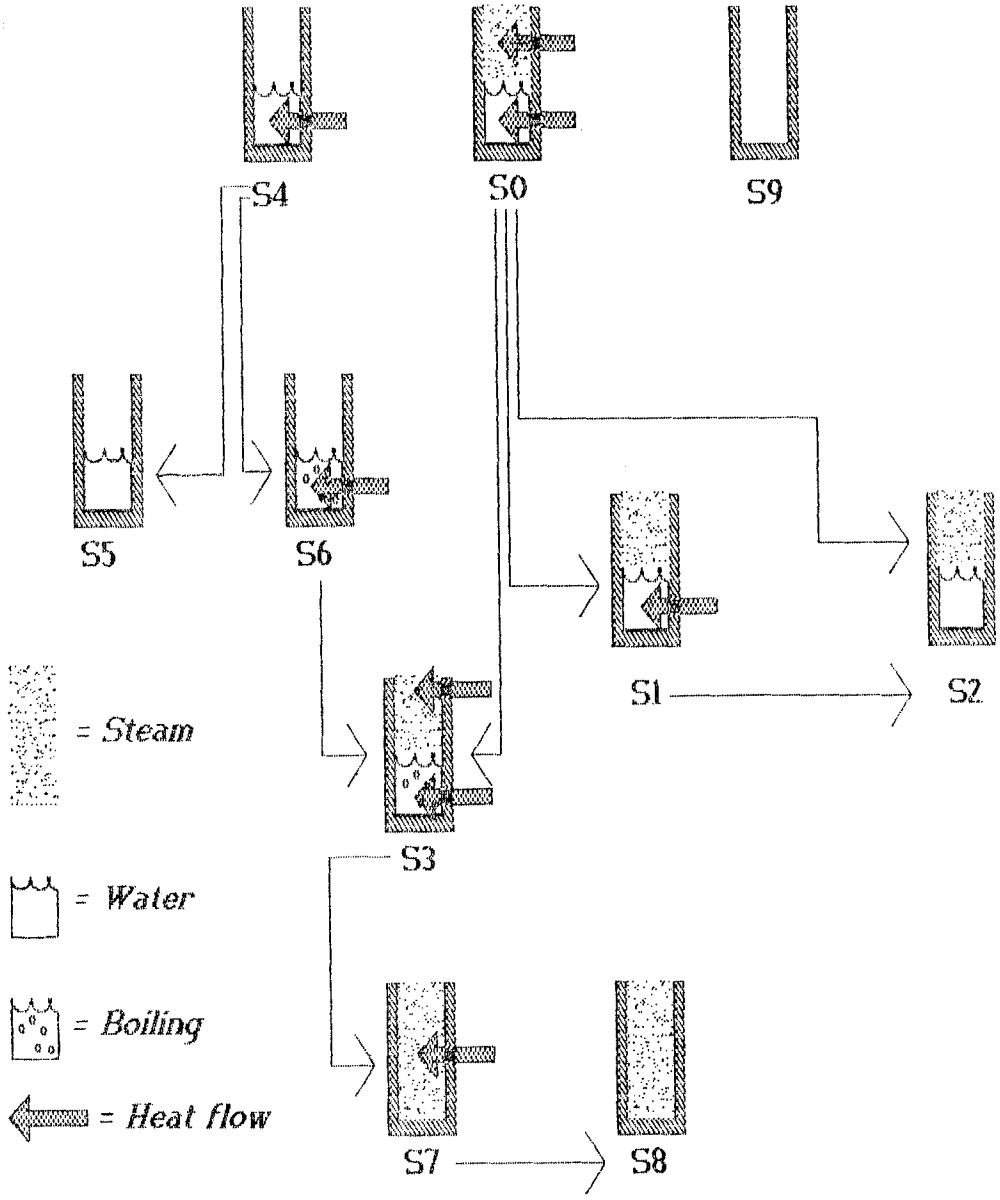

Fig. 11. The total envisionment for the boiling example

After the inconsistent transitions have been weeded out, all remaining transitions are examined using the Equality Change Law[16] to calculate the situation's duration. Each Sclass is examined to determine if the situations which comprise it last for an instant or an interval of time. If both kinds of situations occur, the Sclass is split. At this point, the envisionment is complete. The total envisionment for the example is shown in Fig. 11.

3.10 Current status

The first version of QPE (QPE v.1) was finished in March, 1986[19]. QPE v.1 was used to develop a variety of domain models, and was used as a module in research on planning[26], measurement interpretation[20], reasoning about alternate ontologies[1], and learning[13]. Our experience with this version uncovered various bugs and limitations, as well as suggesting some valuable additions.

QPE v.2 incorporates many improvements over QPE v.1, in addition to bug fixes. For instance, we have added optional extra consistency tests and a 'flight recorder' for debugging purposes, and augmented our graphical display system to show graphs of inequalities and influences as well as envisionments. We finished QPE v.2 in November, 1987. It has been tested with a variety of domain models since then, and has been successfully as a component in several projects[14,22]. Although we are still tuning its performance, we basically consider it finished. A formal complexity analysis of the algorithms in QPE v.2 is in progress.

3.11 Performance

Here we demonstrate the value of our ATMS techniques in two ways. First, we compare QPE's performance with GIZMO, the first implementation of QP theory. Second, we comment on how various factors affect the program's performance.

3.11.1 Comparison to GIZMO

Comparing programs is difficult, and hence a number of caveats are in order. GIZMO was designed as a conceptual tool to explore QP theory. Roughly, every occurrence of what would be an application of universal instantiation in a natural deduction system gave rise to a

Example	GIZMO			QPE v.2		
	Run time	Situations	Sec/Sit	Run time	Situations	Sec/Sit
Two containers	231	3	77	33	6	5.5
Boiling	210	6	35	43	19	2.3
Three Containers	1131	14	81	148	32	4.6
Four Blobs	3176	89	36	801	159	5
Sec/Sit:		57			4.35	

Fig. 12. QPE and GIZMO performance figures. The mode switches for QPE v.2 are set to approximate GIZMO's operation. Since GIZMP produces attainable envisionments and QPE produces total envisionments we must normalize their results. Here we divide the performance on each example by the number of situations produced, and average this number over several examples to provide an index of performance. By this measure, QPE v.2 is roughly 13 times faster than GIZMO

distinct node in the underlying logic-based TMS. Thus a typical conclusion in GIZMO might depend on 250 assumptions, only three of which are specific to the particular situation. Clearly this is not the most efficient implementation strategy for a production program. But as a tool for exploring theories having these assumptions explicitly available is invaluable, since it makes it easier to distinguish debugging the theory from debugging the program. QPE, on the other hand, was designed to be a 'production' program. It is designed to make as few assumptions as possible, and to minimize the number of justifications generated.

Fig. 12 shows comparative run times on several of the original GIZMO examples[17]. All data reported here was generated on the same Symbolics 3670, with 24MB of RAM and 200MB of virtual memory, running Symbolics Lisp Release 6.1. In these runs ambiguous direct influences were resolved using sum quantities, as described in Section 3.6. Simply comparing run times is not an accurate measure of performance, since GIZMO generates only those situations that can arise from a given particular initial state (i.e., attainable envisionments) whereas QPE generates all situations possible from all initial states (i.e., total envisionments), typically a much larger number. Consequently we also show the number of situations produced, and normalize by dividing the run time by the number of situations. This number is averaged to produced a rough index of performance for each program. As these results indicate, in this mode QPE v.2 is, on average, a respectable 13 times faster than GIZMO.

Two additional comments should be made about these figures. First, the original GIZMO run time reported[17] were from a Symbolics 3600 with only 4MB of RAM and a slower disk. On that machine, GIZMO averaged 240 seconds/situation. Increasing the amount of RAM by a factor of six improved performance by a factor of four. Thus we can generate envisionments about 55 times faster than we could in 1984. QPEv.2 is also more space:efficient; we can now produce envisionments that GIZMO could never do.

3.11.2 Factors affecting performance

There are two sources of problem complexity. One is simply how large the descriptions are: The number of objects, quantities, process and view instances, etc. The other factor is how many inequality assumptions are introduced in the course of envisioning. This is a property of how constrained the model is. For example, in certain modes if QPE cannot deduce the sign of a rate, it creates explicit assumptions for each alternative. Multiple simultaneous direct influences also can cause extra assumptions to be added (see Section 3.6), as can lack of knowledge of derivative comparisons (see Section 3.9).

Empirically, the number of additional assumptions introduced during the analysis has a major impact on performance. Fig. 13 illustrates how QPE v.2's performance varies when varying the modes which introduce assumptions. The degradation from additional assumptions can be quite severe. For example, QPE adds seven assumptions during influence resolution in the three containers problem, and two additional

Table 1	QPE v.2		
	Run time	Situations	Sec/Sit
Two Containers	33	6	5.5
Boiling	43	19	2.63
Three Containers	149	32	4.66
Four Blobs	200	135	1.48

Table 2	QPE v.2		
	Run time	Situations	Sec/Sit
Two Containers	33	6	5.5
Boiling	43	19	2.26
Three Containers	148	32	4.625
Four Blobs	801	159	5.04

Table 3	QPE v.2		
	Run time	Situations	Sec/Sit
Two Containers	34	6	5.67
Boiling	49	19	2.59
Three Containers	703	44	16

Fig. 13. How mode switches affect performance. QPE v.2 contains several switches which control what classes of assumptions are introduced, and consequently affect both the detail of the answer and performance. The first table shows performance with no additional assumptions, the second table adds assumptions about sums of direct influences, and the third adds assumptions about specific pairs of opposing direct influences as well

comparison assumptions about derivatives of rates during limit analysis. These nine assumptions add 12 more situation to the envisionment, but increase the computation time by a factor of four! The degradation is even more severe for the four blobs problem, which had run for hours and generated over 5200 situations before the machine was halted manually.

There are two factors at work here. First, combinations of introduced assumptions tend to be relatively unconstrained. After all, they were introduced exactly to make up for ambiguity in the state directly implied by the model. Fortunately, the growth is fairly restrained in reasonable models. In the worst case, N extra inequality assumptions could make the envisionment grow by a factor of 4^N. Second, the runtime increases more rapidly than the growth of the envisionment because the ATMS still has to compute new nogoods for the inconsistent combinations of these new assumptions.

We have found several strategies for avoiding these performance problems. First, constrain quantities, especially rates, as much as possible when building domain models. Second, generate low-resolution envisionments whenever possible. Third, when high-resolution envisionments are needed they can be generated by successive refinement: Use a low-resolution envisionment to determine what subspace of the behaviours are interesting, and then generate a high-resolution description of just that subspace by including additional assumptions about behaviour.

4. CONCLUSIONS

We have described QPE, a new implementation of Qualitative Process theory with substantially improved performance (between 7 and 16 times faster than GIZMO, depending on the desired degree of resolution in the answer). We believe that QPE v.2 provides a valuable tool for building the next generation of qualitative models and reasoning systems.

Based on our experiences with QPE we offer several observations on using an ATMS in building problem-solvers. The advantages are:

1. *Speed:* By allowing most deductions to be done independent of specific situations, the ATMS can provide significant performance improvements. Instead of drawing conclusions once per situation, inferences can be made for sub-contexts and woven together to form complete solutions.
2. *Program simplicitly:* Simplifying the interpretation of facts by avoiding explicit temporal references and by providing the ability to explicitly manipulate assumptions allows programs to be substantially smaller and cleaner. For example, QPE consists of just over 5000 lines of code, while GIZMO is just over 15 000 lines*.

However, there can also be significant disadvantages in using an ATMS:

1. *Justifications must be written carefully.* Too few justifications cause combinatorial explosions during interpretation construction. Installing too many justifications leads to vast inefficiencies in the ATMS. Unlike a logic-based TMS, where rapid prototyping is facilitated by allowing the user to assert arbitrary propositional logic statements, the user of an ATMS must very carefully lay out the way in which each domain fact is to be used and specify which facts are to serve as assumptions. An ATMS which used disjunctive normal form for clauses instead of horn clauses, if it can be made to run efficiently, could help overcome this limitation.
2. *Intermediate interpretation bulge:* While, in theory, interpretation construction is order-independent, in practice considering choices in different orders leads to dramatic performance differences. Early experiences with QPE v.1 showed that choosing the wrong order could slow performance by a factor of 6, or even make the machine run out of memory. A useful heuristic used in both versions of QPE is to order the choice sets by logical dependency.

4.1 *Future work*

While we are still tuning QPE v.2 to improve performance, the basic program is stable and we are moving on to other problems and extensions. Some of these extensions include:

- *Support for defining new sets in domain models:* Currently there is no facility for defining domain-dependent sets, such as the set of contents in a container or the set of forces on an object. The mechanism of closed-world tables appears general enough to handle such sets, assuming that conventions for when to close the sets can be established. We plan on first implementing a version where the modeller specifies when the set must be closed, and examine how this facility is used toward providing QPE with the ability to make these decisions automatically.
- *A portable Common Lisp version:* While QPE v.2 is written in Common Lisp, it still assumes the Symbolics user interface. We are expunging all Symbolics-specific parts of the code, banishing them to a separate interface package. The core program will be 'source-portable', in that the same source should compile on any machine running a reasonably complete Common Lisp. A character-oriented generic interface is being built, as well as new graphical interfaces for Symbolics, TI Explorers, and Lucid Common Lisp on IBM RT's. We intend to make the portable version publically available for research purposes. A manual is in progress, as well as a tutorial on domain modelling.
- *Domain analysis tools:* We are identifying several classes of bugs in qualitative models, such as underconstrained inequalities and undeclared quantities and relationships, which can easily be detected by cross-reference and simple static analysis. We plan on developing tools to isolate and flag such problems in order to speed model development. Brian Falkenhainer has written the first version of such a system which, although simple, has turned out to be extremely useful.
- *An implementation-independent modelling language:* While QP theory places many constraints on a modelling language, it is not itself a fully specified modelling language *per se*. We are beginning to

* Both figures ignore user-interface code. GIZMO included a simple generator of English descriptions which we eventually plan to make available with QPE. QPE includes better graphical debugging tools as well as a window-oriented query system.

develop an implementation-independent formal modelling language for expressiing domain models, so that researchers can begin to accumulate knowledge bases that can be used with any implementation of QP theory.

ACKNOWLEDGEMENTS

Several of the ATMS reasoning techniques described here, and the ATMoSphere problem-solving language, were developed in collaboration with Johan de Kleer. Several of the algorithms were developed in collaboration with John Collins, with lots of kibbitzing by Brian Falkenhainer. John Collins, Dennis DeCoste, Brian Falkenhainer, John Hogge, Barry Smith, Gordon Skorstad have all suffered through alpha testing of QPE, uncovered many bugs, and provided useful advice, encouragement, and in some cases, code. John Hogge's ZGRAPH display system has significantly sped the development of QPE.

This research was supported by the Office of Naval Research, Contract No. N00014-85-K-0225, by an NSF Presidential Young Investigator Award, and an equipment grant from IBM.

REFERENCES

1 Collins, J. and Forbus, K. Reasoning about fluids via molecular collections, Proceedings of AAAI-87, July 1987
2 de Kleer, J Causal and teleological reasoning in circuit recognition, MIT AI Lab. Technical Report No. 529, September 1979
3 de Kleer, J. Choices without Backtracking, AAAI-84, Austin, Texas, August 1984
4 de Kleer, J. An assumption-based truth maintenance system, *Artificial Intelligence*, 1986, **28**
5 de Kleer, J. Problem solving with the ATMS, *Artificial Intelligence*, 1986, **28**, 197–224
6 de Kleer, J. Extending the ATMS, *Artificial Intelligence*, 1986, **28**
7 de Kleer, and Brown, J. A qualitative physics based on confluences, *Artificial Intelligence*, 1984, **24**
8 de Kleer, J., Doyle, J., Steele, G. and Sussman, G. Explicit control fo reasoning. In *Artificial Intelligence: An MIT Perspective: Volume 1*, (Eds P. Winston and R. Brown), The MIT Press, Cambridge, Mass., 1979
9 de Kleer, J. Causal and teleological reasoning in circuit recognition, MIT AI Lab. Technical Report No. 529, September 1979
10 de Kleer, J. and Brown, J. A qualitative physics based on confluences, *Artificial Intelligence*, 1984, **24**
11 de Kleer, J. and Williams, B. Back to Backtracking: Controlling the ATMS, AAAI-86, Philadelphia, Pennsylvania, August 1986
12 Doyle, J. A truth maintenance system, *Artificial Intelligence*, 1979, **12**(3), 231–272
13 Falkenhainer, B. An examination of the third stage in the analogy process: Verification-based analogical learning, Proceedings of IJCAI-87, August 1987
14 Falkenhainer, B. and Forbus, K. Setting up large-scale qualitative models, Proceedings of AAAI-88, August 1988
15 Forbus, K. Qualitative reasoning about physical processes, IJCAI-7, Vancouver, BC, August 1981
16 Forbus, K. Qualitative Process theory, *Artificial Intelligence*, 1984, **24**
17 Forbus, K. Qualitative Process theory, MIT AI Lab. Technical Report No. 789, July 1984
18 Forbus, K. The problem of existence, Proceedings of the Cognitive Science Society, 1985
19 Forbus, K. The Qualitative Process Engine, Technical Report No. UIUCDCS-R-86-1288, December 1986
20 Forbus, K. Interpreting observations of physical systems, *IEEE transactions on systems, man, and cybernetics*, May/June 1987, Vol. SMC-17, No. 3
21 Forbus, K. The logic of occurrence, Proceedings of IJCAI-87, Milan, Italy, 1987
22 Forbus, K. Introducing actions into qualitative simulation, UIUC Department of Computer Science technical report, in preparation
23 Forbus, K. and de Kleer, J. Focusing the ATMS, Proceedings of AAAI-88, August 1988
24 Forbus, K. The QPE manual, Technical report in preparation
25 Hayes, P. The naive physics manifesto. In *Expert systems in the microelectronic age*, (Ed. D. Michie), Edinburgh University Press, 1979
26 Hogge, J. Compiling plan operators from domains expressed in Qualitative Process theory, Proceedings of AAAI-87, July 1987
27 Kuipers, B. Common sense causality: deriving behavior from structure, *Artificial Intelligence*, 1984, **24**
28 Kuipers, B. Qualitative Simulation, *Artificial Intelligence*, September 1986, **29**
29 London, P. E. Dependency networks as a representation for modelling in general problem solvers, Rep. No. TR-698, Computer Science Department, University of Maryland, 1978
30 McAllester, D. An outlook on truth maintenance, MIT AI Lab. Memo No. 551, August 1980
31 McAllester, D. Reasoning utility package user's manual: version one, MIT AI Lab. Memo No. 667, April 1982
32 McCarthy, J. and Hayes, P. Some philosophical problems from the standpoint of artificial intelligence, *Machine Intelligence 4*, Edinburgh University Press, 1969
33 McDermott, D. and Sussman, G. The CONNIVER reference manual, MIT AI Lab. Memo No. 259, Cambridge, May 1972
34 Morris, P. and Nado, R. Representing actions with an assumption-based truth maintenance system, AAAI-86, Philadelphia, Pennsylvania, August 1986
35 Moses, J. Algebraic simplification: A guide for the perplexed, *Communications of the ACM*, August 1971, **8**(14)
36 Shearer, J., Murphy, A. and Richardson, H. *Introduction to System Dynamics*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1967
37 Simmons, R. Representing and reasoning about change in geologic interpretation, MIT Artificial Intelligence Lab. TR-749, December 1983
38 Simmons, R. Commonsense arithmetic reasoning, AAAI-86, Philadelphia, Pennsylvania, August 1986
39 Stallman, R. and Sussman, G. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis, *Artificial Intelligence*, October 1977, (9), 135–196
40 Williams, B. Qualitative analysis of MOS circuits, *Artificial Intelligence*, 1984, **24**
41 Weld, D. The use of aggregation in qualitative simulation, *Artificial Intelligence*, October 1986, **30**(1)