The Qualitative Process Engine: A study in Assumption-based Truth Maintenance

Kenneth D. Forbus
Qualitative Reasoning Group
Department of Computer Science
University of Illinois
1304 W. Springfield Avenue
Urbana, Illinois, 61801
Internet: Forbus@uiuc. ARPA
Voice: (217) 333-0193

Abstract

This paper describes how to use an assumption-based truth maintenance system (ATMS) to significantly speed up qualitative reasoning. Specifically, we introduce three organizing abstractions for ATMS-based problem solvers (*many-worlds databases*, *justify/interpret cycles*, and *closed-world tables*). We illustrate their utility by describing the *Qualitative Process Engine* (QPE), an implementation of Qualitative Process theory that is roughly 95 times faster and signficantly simpler than the previous implementation. After analyzing QPE's performance, we draw some general conclusions about the advantages and disadvantages of assumption-based truth maintenance systems.

Program: ENGINEERING
Area: Commonsense Reasoning
Subarea: Qualitative Physics
Length: 3977 words
Draft of: February 5, 1987

1 Introduction

Recently there has been a great deal of interest in *assumption-based truth maintenance systems* (ATMS) [1,2,3] as a tool for building AI programs. This paper describes several ATMS techniques applicable to a broad class of problems. We illustrate these techniques by outlining their use in an envisioner for Qualitative Process (QP) theory [8,9,10] which is roughly 95 times faster than our previous implementation[1]. Based on this experience we comment on some advantages and disadvantages of ATMS technology.

Qualitative simulation programs take as input a domain model and a scenario, and produce as output a description of possible behaviors. The low resolution of qualitative models means there typically are several possible behaviors. *Envisioning* is the process of generating a description of *all* possible behaviors. Representing and reasoning with these various possibilities is complex, and makes heavy computational demands relative to generating specific histories. Still, envisioning is extremely valuable for developing new qualitative models, and produces knowledge bases which may potentially be compiled for further applications (c.f. [13]).

We view efficient envisioners as essential to continued progress in qualitative physics. de Kleer has claimed that an ATMS is superior for these kinds of problems, and this paper can be viewed in part as evidence for this claim. Qualitative Process theory takes on more modelling work than device-centered models (such as [4,22]), making implementations of it more complex and thus a better test of ATMS technology. For instance, a QP implementation must *apply* the idealizations of a physics to descriptions of objects and relationships, rather than starting with a network of idealized devices specified by the user. Changes in existence, such as steam appearing and water vanishing due to boiling, must be taken into account. While we believe these extra capabilities make QP theory more suitable for a broader range of modeling tasks, they also pose formidable challenges to the implementor (see [12] for details).

This paper describes a set of useful abstractions for building ATMS problem solvers to meet these and similar challenges. The advantages of these abstractions are both conceptual and computational; using them can both speed up and radically simplify representations and algorithms. They are:

1. *The Many-Worlds database:* The ATMS notion of environment can be used to eliminate complicated temporal reference schemes for many tasks.

2. *The Justify/Interpret cycle:* Organizing problem solving as alternating phases which install justifications and construct partial solutions can reduce the computational complexity of algorithms.

3. *Closed World Tables:* The Justify/Interpret organization allows efficient implementation of a simple yet expressive version of closed-world assumptions.

[1]QPE is written in ADB, an ATMS-based problem-solving language being developed jointly with Johan de Kleer of Xerox PARC.

We begin by elaborating these ideas. Next we illustrate their use in the *Qualitative Process Engine* (QPE), a high-performance envisioner for QP theory. Finally, we analyze QPE's performance, and summarize some advantages and pitfalls we have experienced with ATMS technology.

2 ATMS nomenclature

The ATMS is similar to the original, justification-based TMS's (or JTMS, e. g. [21]). Each fact has an associated *node*, whose status of IN or OUT indicates whether or not it is believed. Justifications are Horn clauses; when all antecdent nodes are IN, the consequent node will also be IN[2].

Unlike JTMS's, *assumptions* are a distinct datatype in the ATMS. Roughly, an assumption can be viewed as the choice to assume a particular fact (but see [3]). A set of assumptions is called an *environment*. The *label* of an ATMS node is the set of minimal environments under which that node is believed. Environments provide the ATMS notion of 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. ATMS justifications primarily serve to propagate environments. Since the ATMS maintains multiple environments, and one can easily test if a fact is believed in an environment, reasoning in multiple contexts is greatly simplified.

Logical constraints are installed by providing justifications for certain nodes marked as *contradictory*. Any environment which appears in the label of a contradictory node must itself be contradictory, and hence is called a *nogood*. Any environment which contains a nogood is itself nogood (i. e., we forbid non-monotonicity). For simplicity we sometimes also refer to justifications of contradictory nodes as nogoods since their effect is to generate contradictory environments.

3 The Many-worlds database

Using environments as contexts is very powerful. For example, a common inferential operation is finding consistent extensions of a partial solution. The ATMS reduces this operation to finding whether or not the fact F (representing the extension) is consistent with the environment E (representing the partial solution). The answer is yes if some environment E_1 in F's label can be consistently combined[3] with E. However, fully exploiting this notion requires a different problem-solver organization than other TMS's.

A useful interpretation of facts in an ATMS problem solver is as statements about individuals in all possible worlds, rather than as statements about those individuals at some specific time. Any particular world, be it different temporally or hypothetically, is defined by a particular environment. The set of consequences of the defining

[2]Non-monotonic justifications are irrelevant to this paper.

[3]Environments are combined by taking the union of the sets of assumptions involved, marking the result as inconsistent if it subsumes an environment already known to be contradictory.

environment specify what is true in that world. This scheme avoids the complicated pattern manipulation required to directly implement notations for temporal reference, including situational calculus [16] and histories [14] (c.f. [20,10]). For example, in this scheme we would simply write `Left-of(Block-A, Block-B)` instead of `Left-of(Block-A, Block-B, SO)` or `Left-of(at(Block-A, SO), at(Block-B, SO)`, assuming that the label of the fact contained an environment which is a subset of the environment corresponding to `SO`.

One has to think differently about a database organized in this way. Metaphorically, consider databases as if they were physical worlds. In a JTMS or logic-based TMS (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 particular answers, just as taking a physical measurement provokes the collapse of the QM wave function. Consequently, we call this organizational scheme the *many-worlds* database.

While not as general as explicit temporal notations — we cannot say `Left-of(at(Block-A, SO), at(Block-B, S1))`, for instance — the many-worlds database has clear advantages for some kinds of problems. The most obvious overhead in temporal reference notations is the storage involved in making copies of assertions, new justifications, and so forth. In the many-worlds database the same assertions and justifications are used, only new environments are created. Less obvious, but more serious, is the additional complication required for pattern-matching, as anyone who has implemented such schemes can attest. `CONNIVER`-style databases [17] can also be used to model time implicitly, but without the composability of the ATMS structures: Each access to a fact must work backward up the tree of contexts to see what the actual status of that fact is. The ATMS allows contexts to be synthesized as needed, rather than sprouting new, monolithic contexts.

The many-worlds database is the closest to the original conception of the ATMS [1], and is used in de Kleer's qualitative physics programs as well as `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 simpler than the Viewpoint mechanism of `ART` or the Worlds mechanism of `KEE` [18], two current commercial systems which use ATMS-like technology. These mechanisms are closer in spirit to `CONNIVER`-style databases, using assumptions to model markers corresponding to assertion and deletion of facts within particular named contexts. This provides the ability to retract facts from contexts. In the many-worlds database contexts are only identified with environments, so retraction does not make sense.

4 The Justify/Interpret cycle

The descriptions in a many-worlds ATMS database form the framework for casting solutions to problems. The justifications determine the consequences which follow from every environment, as well as weed out inconsistent environments. A *solution* in a problem-solver organized along these lines is a particular environment and its attendent consequences. There are a number of ways to construct such solutions (c.f. [2,6]). A particularly useful technique for problems where the entir, space of solutions is sought, such as envisioning, is to organize the problem solver into *justify/interpret cycles.* Given initial facts and partial solutions,

```
1. Until the problem is solved,
   1.1 Justify: Create justifications representing the conclusions
       and constraints that follow from the current set of facts.
   1.2 Interpret: Extend the partial solutions, based on the
       new conclusions and constraints.
```

The standard N-queens problem provides a simple example. The goal is to find all the ways N queens can be placed on an $N \times N$ chessboard so that no two queens can capture each other. Clearly there must be exactly one queen in each column, so each justify phase of the cycle consists of creating statements corresponding to the possible locations of a queen within a particular column, and installing nogoods to rule out combinations of queen placements in the previous columns that would result in a capture. Each interpret phase extends solutions by trying to add assumptions about the queen's location in the current column consistent with previous choices. (The process of extending solution environments with new choices is called *interpretation construction*[2].) After N cycles the solutions are complete.

Very simple combinatorial problems, such as the N-queens problem, can actually be solved by a single justify/interpret cycle. More complex problems require decomposition into several distinct justify/interpret cycles. The reason is that interpretation construction, left to itself, can lead to immense combinatorial explosions. By breaking the process up into several cycles, partial solutions from each stage can guide the construction of justifications and nogoods at the next stage.

Consider for example a program to generate feasible travel plans. The choices in such a plan include means of transportation (such as airplane, train, bus, or car) and routes. In principle, a set of plans can be generated by using a set of pattern-directed inference rules (as in AMORD[5] or DEBACLE[10]) to construct all the relevant justifications and nogoods, and then using interpretation construction to find all consistent plans. This technique fails in practice because it fails to take into account logical dependencies between the choices. Suppose the choice of vehicle was made last, and time constraints ruled out every choice but air travel. The interpretation construction process will have generated all possible routes by train, bus, and car when in fact all of these choices are irrelevant. Aside from gross inefficiency, sometimes the problem is not solved at all because the environments generated during intermediate phases of interpretation construction overflow the address space of the computer, even though the final answer fits comfortably. We call this problem *intermediate interpretation bulge*, by analogy with a similar problem in symbolic algebraic manipulation systems (see [19]).

By decomposing the solution process into distinct phases the combinatorial explosions plaguing a uniform problem-solver organization can be avoided. For instance, if the mode of transport were fixed first, the number of intermediate solutions would remain quite small in the example above. Similar dependencies appear in most kinds of problem-solving. In QP theory, for example, there is no reason to calculate the effects of two processes P_1, P_2 taken together if they can never consistently be active at the same time. Although it does not affect the correctness of the final answers, it is wasted effort. Empirically, we have found careful decompositions to be essential in envisioning.

5 Closed-World Tables

Closed-world assumptions are often required in problem solving. A simple but expressive form of closed-world assumption concerns set membership. The form is, roughly, "the known members of the set are the only ones". Examples from QP reasoning include the set of active processes, the direct influences or qualitative proportionalities that constrain a quantity, and the inequalities comprising a quantity space. Many common closed-world assumptions in other domains can be cast in this form as well, including the possible suspects for a crime, the observations a theory must explain, and the premises underlying a contradiction.

While clearly useful, the techniques used in other TMS's for implementing these closed-world assumptions are unsuitable for the ATMS. Typically one implements this reasoning as follows: At some point in the computation it becomes necessary to know the members of a set, given the assumptions made so far. The set is *closed* by first fetching the membership statements currently believed. An assertion that these members comprise the set at this point is justified by these statements and an explicit statement of the closed-world assumption. Since new members can be added or removed at any time, a mechanism for retracting the closed-world assumption is necessary. This can be done in the TMS itself [7] or by pattern-directed rules which detect and signal such conditions [10]. Unfortunately, these techniques assume a single, global context: directly applying this technique to an ATMS typically results in an exponential number of (mostly irrelevant) assertions.

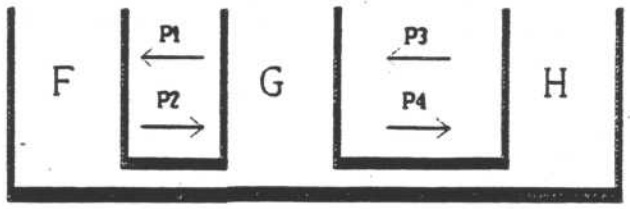
The organization provided by the justify/interpret cycles can be exploited to provide a highly efficient alternative. There must be some stage in the solution process at which all information required to determine the possible compositions of a set is known. If the set is closed then, only sets belonging to some consistent solution will be generated. This requires a means of determining the possible members of each set. We introduce *closed world tables* to provide this information.

A closed-world table is a collection of entries whose form is

(<*member*> . <*justifying form*>)

The *member* is a potential element of the set, such as a particular person being a murder suspect or a quantity having a particular influence. The *justifying form* is

Figure 1: Example of closed-world tables
A subproblem facing a QP interpreter is finding how the amount of water in G might change, given the various combinations of liquid flow processes $P_{1...4}$ that can occur.



a statement whose belief justifies that member being in the set. The set of members listed in the closed world table is implicitly assumed to be complete.

A closed-world table can be used as soon as the labels for the justifying forms are complete. Finding all consistent combinations of the justifying forms yields the possible compositions of the set. Each statement about the membership of the closed set is justified by the union of justifying forms for members and the negation of the justifying forms for non-members. Every combination represents the set's members for some class of solutions.

Consider the scenario in Figure 1. One subproblem in reasoning about this scenario is figuring out all the ways that the amount of water in container G can change, by calculating the sign of its derivative (the Ds value). If the only processes that can occur are liquid flow, then the amount of water will tend to increase when P_2 or P_3 are occuring, and tend to decrease when P_1 or P_4 are occuring. (See [9] for details of influence resolution.)

To solve this subproblem we first must find all combinations of these four processes which can be active at the same time, and for each combination calculate their net effect. Figure 2 illustrates the consistent combinations found by interpretation construction on the justifying forms of a closed-world table. The inconsistencies come from shared quantity conditions; a flow in one direction implies a pressure difference that rules out a flow in the opposite direction. Figure 3 shows the net effect on the amount of water in G calculated for each consistent combination. When influenced in both directions the program creates alternate worlds corresponding to each choice, installing assumptions about the relative magnitudes of rates. Since these result only depend on the information in the closed world table, they will be available in all situations which contain the relevant processes.

Figure 2: Generating consistent sets of active processes
Finding what combinations of processes can influence the amount of water in G requires making closed-world assumptions. Justifying forms from a closed-world table define the search space.

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

Figure 3: Finding `Ds[Amount-of(G)]`
The net effect of each consistent collection of processes can be simply found. Since the results depend only on the information in the closed-world table, they will be applicable in every relevant situation.

Active	{}	$\{P_1\}$	$\{P_2\}$	$\{P_3\}$	$\{P_4\}$	$\{P_1, P_3\}$	$\{P_1, P_4\}$	$\{P_2, P_3\}$	$\{P_2, P_4\}$
Ds values	0	-1	1	1	1	-1, 0, 1	-1	1	-1, 0, 1

6 The Qualitative Process Engine

We outline QPE's algorithms only in enough detail to illustrate how these ideas can be applied to organizing problem-solvers; a detailed description is beyond the scope of this paper (see [12]).

QPE is organized as a collection of justify/interpret cycles. The processing occurs in five major steps:

1. Load the domain model.
2. Load the scenario.
3. Find the Process and View structures
4. Resolve influences
5. Perform Limit Analysis

We describe each in turn.

6.1 Step 1: Load the domain model.

Domain models describe types of quantities, objects, relationships, individual views (i.e., time-varying relationships), and processes (see [10]). A syntaxer turns the domain model into ADB assertions and pattern-directed rules. Given a scenario, these rules find possible instances of processes and individual views and instantiate definitions of predicates and relations. Unlike de Kleer's consumers, these rules are run on all assertions matching their triggers irregardless of their current belief status. The network of assertions, justifications, and nogoods they build are used by later steps to prune possibilities.

6.2 Step 2: Load the scenario.

Next the facts corresponding to the particular scenario are asserted, causing the domain rules to run. Aside from creating assertions and justifications, the rules also create the necessary closed-world tables. These include potential influencers of quantities and sets of preconditions and quantity conditions. At the end of this step all potential occurrences of processes and views are known, but nothing is known about whether they actually occur and in what combinations. This step ends the justify portion of the first justify-interpret cycle.

6.3 Step 3: Find the Process and View structures.

In QP theory a process or individual view is active exactly when its *quantity conditions* (inequalities, plus the status of other processes or individual views) and *preconditions* (other kinds of facts) are true. Finding what collections of processes and views can be active together is the first step in generating situations.

Situations are found by generating all consistent combinations of preconditions and quantity conditions, using closed-world tables constructed during Step 2. Each environment so formed is the seed of a situation. Generally several situations can have the same active processes and views (since negating any precondition or quantity condition suffices to deactivate a process or view), so they are divided into equivalence classes called *sclasses*. The computation of situations and sclasses completes the first justify/interpret cycle.

6.4 Step 4: Resolving Influences

This step determines for every quantity in every situation the sign of its derivative (i.e., the `Ds` value, one of `-1, 0, 1`). In any particular situation `Ds` values are computed by gathering the set of *influences* for each quantity and determining their net effect. As the example in Section 5 indicated, in `QPE` this process occurs in two phases. First, the closed world tables built for each quantity and each type of influence (direct or indirect; see [9] for details) are used to gather all possible sets of influences and determine their effects where unambiguous. This part concludes the justify phase of the second justify/interpret cycle.

Second, the environments just generated are used to determine for each situation the `Ds` value for unambiguously influenced quantities. Ambiguities in influence resolution (caused by multiple effects, such as simultaneous flow into and out of a container) force `QPE` to create new situations by extending the affected ones with additional assumptions about relative rates[4] and Ds values. The second justify/interpret cycle is now complete.

6.5 Step 5: Limit Analysis

Limit analysis finds state transitions. A *limit hypothesis* (LH) concerns possible changes in quantity conditions or relative rates from influence resolution. Limit analysis begins by considering each consistent combination of an inequality and the Ds values of its associated quantities. Table-lookup suffices to find how these inequalities can change [9], yielding the initial set of LHs (the justify phase of the third justify/interpret cycle). Potential simultaneous changes are found by testing combinations of LHs to see which of them are mutually consistent. As with influence resolution, these computations are performed once per *scenario*, rather than once per *situation*, greatly improving efficiency.

The rest of the computation proceeds by weaving these partial results together to find transitions between states. First, the LHs applicable to each situation are found. Second, for each situation/LH pair the state that would result from its occurrence is computed. The temporal inheritance algorithm employed [11] explicitly manipulates sets of assumptions, which in other TMS's requires constantly changing belief states of a substantial part of the TMS database. The ATMS ability to explicitly manipulate sets of assumptions substantially simplifies this algorithm and improves its efficiency.

Being local, these results must be tested to ensure they satisfy various compatability and continuity constraints (see [9] for details). These tests can be complicated when objects vanish, since the relevant inequalities no longer exist. In other TMS' these tests require reinstalling the relevant inequalities and testing the new situation so created. In the ATMS each situation contains an explicit assumption that certain laws pertaining to quantities hold; to make these tests simply requires testing the environment resulting from subtracting this assumption. Finally durations are calculated using the Equality Change Law [9], splitting situations as required.

The third justify/interpret cycle is now complete, and so is the total envisionment.

7 Performance Analysis

Here we compare QPE's performance against GIZMO, the first implementation of QP theory. A caveat: GIZMO was designed as a conceptual tool to explore QP theory. Every theoretical assumption and reasoning step was explicitly represented in GIZMO's logic-based TMS [15]. Thus a typical conclusion in GIZMO might depend on 250 assumptions, of which only three were scenario-specific. While inefficient, these explicit assumptions were invaluable in debugging the theory. Conversely, QPE

[4]Assumptions about relative rates are added to the relevant quantity spaces so that changes in them may be detected.

Figure 4: QPE and GIZMO performance figures

These numbers were generated using a Symbolics 3670 with 474MB Eagle disk and 4MB RAM, using Release 6.1 Zetalisp.

Measured data

	GIZMO		QPE	
Example	Time	#Situations	Time	#Situations
Two Containers	NA	2	32	6
Boiling	465	6	28	25
Three Containers	6300	14	198	66
Four Blobs	14400	86	376	1275

Figure 5: Average performance, in seconds per situation

Example	GIZMO	QPE
Two Containers	NA	5.33
Boiling	77.5	1.12
Three Containers	450.0	3.0
Four Blobs	167.4	0.29
AVERAGE	231.6	2.44

is designed for speed, and thus generates as few justifications and assumptions as possible.

Figure 4 shows comparative run times on several examples from [10]. Normalization is required, since GIZMO generates all situations arising from a given initial state (*attainable envisionment*) and QPE generates all situations possible from every initial state (*total envisionment*), typically a much larger number. Figure 5 shows the average rate of situation production for each program on each example. Averaging these rates across examples provides a rough index of performance improvement. The result is that QPE is, on the average, about 95 times faster than GIZMO.

On closer examination, the three containers example is clearly the most complex. It has more view and process instances than the others, hence more quantities and inequalities are involved. However, QPE is significantly faster on the four blobs problem than on the boiling example. Two factors seem relevant:

1. *Paging:* As the number of situations grows, GIZMO's performance is page-bound. The four blobs example nearly exhausts a 200MB swap space, and paging in the logic-based TMS accounts for 70% of the time. QPE uses signficantly less memory and hardly pages at all (around 1-2%).

2. *Number of quantities:* The Four Blobs problem has the fewest inequalities, since temperature is the only quantity blobs have. The other problems involve

liquids and gasses, and hence more quantities and inequalities. Consequently the search space is larger, with more ambiguous influence resolutions.

8 Conclusions

This paper introduces three abstractions for organizing ATMS-based problem solvers: *Many-worlds databases*, *justify/interpret cycles*, and *closed-world tables*. The utility of these ideas was illustrated by outlining QPE, our new implementation of Qualitative Process theory with substantially improved performance (roughly 95 times faster than GIZMO). We believe QPE will be an essential tool in building the next generation of qualitative models. Furthermore, we believe these ideas will be useful for any problem where generating many solutions is desirable.

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

1. *Speed:* By allowing many deductions to be done independently 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 simplicity:* Avoiding explicit temporal references and providing the ability to explicitly manipulate assumptions allows programs to be substantially smaller and cleaner. For example, QPE consists of just over 4,000 lines of code, while GIZMO is just over 15,000 lines.[5]

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

1. *Justifications must be written carefully.* Installing too few justifications causes combinatorial explosions during interpretation construction. Conversely, redundant justifications causes vast inefficiencies inside 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 decide how each kind of fact is used and which facts will be assumptions. An ATMS which used disjunctive normal form for justifications instead of Horn clauses, if efficient, could overcome this limitation.

2. *Intermediate interpretation bulge:* While theoretically interpretation construction is order-independent, in practice considering choices in different orders leads to dramatic performance differences. Early experiences with QPE showed that choosing the wrong order could slow performance by a factor of 6, or even overflow memory. A useful heuristic is to order choice sets by logical dependency.

[5]Both figures ignore user-interface code and the underlying inference engines, ADB and DEBACLE, which are roughly the same size.

9 Acknowledgements

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

This research was supported by the Office of Naval Research, Contract No. N00014-85-K-0225.

References

[1] de Kleer, J. "Choices without Backtracking", AAAI-84, Austin, Texas, August, 1984

[2] de Kleer, J. "An assumption-based truth maintenance system", *Artificial Intelligence*, **28**, 1986

[3] de Kleer, J. "Extending the ATMS", *Artificial Intelligence*, **28**, 1986

[4] de Kleer, J. and Brown, J. "A qualitative physics based on confluences", *Artificial Intelligence*, **24**, 1984

[5] de Kleer, J., Doyle, J., Steele, G., and Sussman, G. "Explicit control of reasoning" in Winston, P. & Brown, R., (Eds.), *Artificial Intelligence: An MIT Perspective: Volume 1*, The MIT Press, Cambridge, Mass, 1979

[6] de Kleer, J. and Williams, B. "Back to Backtracking: Controlling the ATMS" AAAI-86, Philadelphia, Pennsylvania, August, 1986

[7] Doyle, J., "A truth maintenance system", *Artificial Intelligence*, **12(3)**:231-272, 1979.

[8] Forbus, K. "Qualitative reasoning about physical processes" IJCAI-7, Vancouver, B.C., August, 1981

[9] Forbus, K. "Qualitative Process theory" *Artificial Intelligence*, **24**, 1984

[10] Forbus, K. "Qualitative Process theory" MIT AI Lab Technical report No. 789, July, 1984.

[11] Forbus, K. "The problem of existence", Proceedings of the Cognitive Science Society, 1985.

[12] Forbus, K. "The Qualitative Process Engine" University of Illinois Department of Computer Science Technical Report No. UIUCDCS-R-86-1288, December, 1986

[13] Forbus, K. "Interpreting measurements of physical systems" Proceedings of AAAI-86, August, 1986.

[14] Hayes, P. "The naive physics manifesto" in *Expert systems in the micro-electronic age*, D. Michie (Ed.), Edinburgh University Press, 1979

[15] McAllester, D. "An outlook on truth maintenance" MIT AI Lab Memo No. 551, August, 1980.

[16] McCarthy, J. and Hayes, P. "Some philosophical problems from the standpoint of artificial intelligence" *Machine Intelligence 4*, Edinburgh University Press, 1969

[17] McDermott, D. and Sussman, G. "The CONNIVER reference manual" MIT AI Lab Memo No. 259, Cambridge, May, 1972

[18] Morris, P., and Nado, R. "Representing actions with an assumption-based truth maintenance system", AAAI-86, Philadelphia, Pennysylvania, August, 1986

[19] Moses, J. "Algebraic simplification: A guide for the perlexed" *Communications of the ACM*, (**14**)8, August 1971

[20] Simmons, R. "Representing and reasoning about change in geologic interpretation", MIT Artificial Intelligence Lab TR-749, December, 1983

[21] Stallman, R., and Sussman, G. "Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis", *Artificial Intelligence*, (**9**), October, 1977, pp. 135-196

[22] Williams, B. "Qualitative analysis of MOS circuits", *Artificial Intelligence*, **24**, 1984