FROM: Proceedings of AAAI '92
July 12-16, 1992
San Jose, California

Self-Explanatory Simulations: Scaling up to large models

Kenneth D. Forbus
Qualitative Reasoning Group
The Institute for the Learning Sciences
Northwestern University
1890 Maple Avenue, Evanston, IL, 60201

Brian Falkenhainer
System Sciences Laboratory
Xerox Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto CA 94304

Abstract

Qualitative reasoners have been hamstrung by the inability to analyze large models. This includes *self-explanatory simulators*, which tightly integrate qualitative and numerical models to provide both precision and explanatory power. While they have important potential applications in training, instruction, and conceptual design, a critical step towards realizing this potential is the ability to build simulators for medium-sized systems (i.e., on the order of ten to twenty independent parameters). This paper describes a new method for developing self-explanatory simulators which scales up. While our method involves qualitative analysis, it does not rely on envisioning or any other form of qualitative simulation. We describe the results of an implemented system which uses this method, and analyze its limitations and potential.

Introduction

While qualitative representations seem useful for real-world tasks (c.f. [1; 15]), the inability to reason qualitatively with large models has limited their utility. For example, using envisioning or other forms of qualitative simulation greatly restricts the size of model which can be analyzed [14; 4]. Yet the observed use of qualitative reasoning by engineers, scientists, and plain folks suggests that tractable qualitative reasoning techniques exist. This paper describes one such technique: a new method for building *self-explanatory simulators* [10] which has been successfully tested on models far larger than previous qualitative reasoners can handle.

A self-explanatory simulation combines the precision of numerical simulation with the explanatory power of qualitative representations. They have three advantages: (1) *Better explanations:* By tightly integrating numerical and qualitative models, behavior can be explained as well as predicted, which is useful for instruction and design. (2) *Improved self-monitoring:* Typically most modeling assumptions underlying today's numerical simulators remain in their author's heads. By incorporating an explicit qualitative model, the simulator itself can help ensure that its results are consistent. (3) *Increased automation:* Explicit domain theories and modeling assumptions allow the simulation compiler to shoulder more of the modeling burden (e.g., [7]).

Applying these ideas to real-world tasks requires a simulation compiler that can operate on useful-sized examples. In [10], our account of self-explanatory simulators required a total envisionment of the modeled system. Since envisionments tend to grow exponentially with the size of the system modeled, our previous technique would not scale.

This paper describes a new technique for building self-explanatory simulations that provides a solution to the scale-up problem. It does not rely on envisioning, nor even qualitative simulation. Instead, we more closely mimic what an idealized human programmer would do. Qualitative reasoning is still essential, both for orchestrating the use of numerical models and providing explanations. Our key observation is that in the task of simulation writing reification of global state is unnecessary. This suggests developing more efficient local analysis techniques. While there is room for improvement, SIMGEN.Mk2 can already write self-explanatory simulations for physical systems which no existing envisioner can handle.

Section outlines the computational requirements of simulation writing, highlighting related research. Section uses this decomposition to describe our new method for building self-explanatory simulations. Sections and discuss empirical results. We use Mk1 below to refer to the old method and implementation and Mk2 to refer to the new.

The task of simulation writing

We focus here on systems that can be described via systems of ordinary differential equations without simultaneities. Writing a simulation can be decomposed into several subtasks:

1. Qualitative Modeling. The first step is to identify how an artifact is to be described in terms of conceptual entities. This involves choosing appropriate perspectives (e.g., DC versus high-frequency analysis) and deciding what to ignore (e.g., geometric details, capacitive coupling). Existing engineering analysis tools (e.g., NASTRAN, SPICE, DADS) provide little support for this task. Qualitative physics addresses this problem by the idea of a domain theory ($\mathcal{DT}$) whose general descriptions can be instantiated to form models of specific artifacts (e.g., [7]). Deciding which domain theory fragments should be applied in building a system can require substantial reasoning.

2. Finding relevant quantitative models. The conceptual entities and relationships identified in qualitative analysis guide the search for more detailed models. Choosing to include a flow, for instance, requires the further selection of a quantitative model for that flow (e.g., laminar or turbulent). Current engineering analysis tools sometimes supply libraries of standard equations and approximations. However, each model must be chosen by hand, since they lack the deductive capabilities to uncover non-local dependencies between modeling choices. Relevant AI work includes [3; 7; 17].

3. From equations to code. The selected models must be translated into an executable program. Relevant AI work includes [2; 21].

4. Self-Monitoring. Hand-built numerical simulations are typically designed for narrow ranges of problems and behaviors, and rarely provide any indication when their output is meaningless (e.g., negative masses). Even simulation toolkits tend to have this problem, relying on the intuition and expertise of a human user to detect trouble. Forcing a numerical model to be consistent with a qualitative model can provide automatic and comprehensive detection of many such problems [10].

5. Explanations. Most modern simulation toolkits provide graphical output, but the burden of understanding still rests on the user. Qualitative physics work on complex dynamics [19; 16; 20] can extract qualitative descriptions from numerical experiments. But since they require the simulator (or equations) as input and so far are limited to systems with few parameters they are inappropriate for our task. The tight integration of qualitative and numerical models in self-explanatory simulators provides better explanations for most training simulators and many design and analysis tasks.

Simulation-building by local reasoning

Clearly envisionments contain enough information to support simulation-building; The problem is they contain too much. The author of a FORTRAN simulator never enumerates the qualitatively distinct global states of a complex artifact. Instead she identifies distinct behavior regimes for pieces of the artifact (e.g., whether a pump is on or off, or if a piping system is aligned) and writes code for each one. Our new simulation-building method works much the same way. Here we describe the method and analyze its complexity and trade-offs. We use ideas from assumption-based truth maintenance (ATMS) [6], Qualitative Process theory [8], Compositional Modeling [7], and QPE [9] as needed.

Qualitative analysis

Envisioning was the qualitative analysis method of MK1. The state of a self-explanatory simulator was defined as a pair $\langle \mathcal{N}, \mathcal{Q} \rangle$, with $\mathcal{N}$ a vector of continuous parameters (e.g., `mass(B)`) and booleans corresponding to preconditions (e.g., `Open(Valve23)`), and $\mathcal{Q}$ ranged over envisionment states.

Envisioning tends to be exponential in the size of the artifact $\mathcal{A}$. Many of the constraints applied are designed to ensure consistent global states using only qualitative information. For example, all potential violations of transitivity in ordinal relations must be enumerated. The computational cost of such constraints can be substantial. For our task such effort is irrelevant; the extra detail in the numerical model automatically prevents such violations.

The domain theory $\mathcal{DT}$ consists of a set of model fragments, each with a set of antecedent conditions controlling their use and a set of partial equations defining *influences* [8] on quantities. The *directly influenced* quantities are defined as a summation of influences on their derivative $dQ_0/dt = \sum_{Inf(Q_0)} Q_i$ and the *indirectly influenced* quantities are defined as algebraic functions of other quantities $Q_0 = f(Q_1, \ldots, Q_n)$. The qualitative analysis identifies relevant model fragments, sets of influences, and transitions where the set of applicable model fragments changes. The algorithm is:

1. Establish a dependency structure by instantiating all applicable model fragments into the ATMS. The complexity is proportional to $\mathcal{DT}$ and $\mathcal{A}$.

2. Derive all minimal, consistent sets of assumptions (called local states) under which each fragment holds (i.e., their ATMS labels). The labels enumerate the operating conditions (ordinal relations and other propositions) in which each model fragment is active.

3. For each quantity, compute its derivative's sign in each of its local states when qualitatively unambiguous (QPT influence resolution). This information is used in selecting numerical models and in limit analysis below. The complexity for processing each quantity is exponential in the number of influences on it. Typically there are less than five, so this step is invariably cheap in practice.

4. Find all limit hypotheses involving single inequalities (from QPT limit analysis). These possible transitions are used to derive code that detects state transitions. This step is linear in the number of ordinal comparisons.

This algorithm is a subset of what an envisioner does. No global states are created and exponential enumeration of all globally consistent states is avoided (e.g., ambiguous influences are not resolved in step 3 and no limit hypothesis combinations are precomputed in step 4). Only Step 2 is expensive: worst case exponential in the number of assumptions due to ATMS label propagation. We found two ways to avoid this cost in practice. First, we partially rewrote the qualitative analysis routines to minimize irrelevant justifications (e.g., transitivity violations). This helped, but not enough.

The second method (which worked) uses the fact that for our task, there is a strict upper bound on the size of relevant ATMS environments. Many large environments are logically redundant [5]. We use labels for two purposes: (1) to determine which model fragments to use and (2) to derive code to check logical conditions at run-time. For (1) having a non-empty label suffices, and for (2) shorter, logically equivalent labels produce better code. By modifying the ATMS to never create environments over a fixed size $\mathcal{E}_{max}$, we reduced the number of irrelevant labels. The appropriate value for $\mathcal{E}_{max}$ can be ascertained by analyzing the domain theory's dependency structure.[1] Thus, while Step 2 is still exponential, the use of $\mathcal{E}_{max}$ greatly reduces the degree of combinatorial explosion.[2]

A new definition of state for self-explanatory simulators is required because without an envisionment, $\mathcal{Q}$ is undefined. Let $\mathcal{N}$ be a vector of numerical parameters, and let $\mathcal{B}$ be a vector of boolean parameters representing the truth value of the non-comparative propositions which determine qualitative state. That is, $\mathcal{B}$ includes parameters representing propositions and the status of each model fragment, but not comparisons. (Ordinal information can be computed directly from $\mathcal{N}$ as needed.) The state of a self-explanatory simulator is now defined as the pair $\langle \mathcal{N}, \mathcal{B} \rangle$. In effect, each element of $\mathcal{Q}$ can be represented by some combination of truth values for $\mathcal{B}$.

Finding relevant quantitative models

The qualitative analysis has identified the quantities of interest and provided a full causal ordering on the set of differential and algebraic equations. However, because the influences on a quantity can change over time, a relevant quantitative model must be found for each possible combination.

This aspect of simulation-building is identical with Mk1. The derivative of a directly influenced parameter is the sum of its active influences. For indirectly influenced parameters, a quantitative model must be selected for each consistent combination of qualitative proportionalities which constrain it For instance, when a liquid flow is occurring its rate might depend on the source and destination pressures and the conductance of the path. The numerical model retrieved would be

$$\mathit{FluidConductance}(\,?path) \times (\,\mathit{Pressure}(\,?source) - \mathit{Pressure}(\,?dest))$$

If N qualitative proportionalities constrain a quantity there are at most 2^N distinct combinations. This worst case never arises: typically there are exactly two consistent combinations: no influences (i.e., the quantity doesn't exist) and the conjunction of all N possibilities (i.e., the model found via qualitative analysis). N is always small so the only potentially costly aspect here is selecting between alternate quantitative models (See Section).

The only potential disadvantage with using $\mathcal{B}$ over $\mathcal{Q}$ in this computation is the possibility that a combination of qualitative proportionalities might be locally consistent, but never part of any consistent global state. This would result in the simulator containing dead code, which does not seem serious.

Code Generation

The simulation procedures in a self-explanatory simulator are divided into *evolvers* and *transition procedures*. An evolver produces the next state, given an input state and time step dt. A transition procedure takes a pair of states and determines whether or not a qualitatively important transition (as indicated by a limit hypothesis) has occurred between them.[3] In Mk1 each equivalence class of qualitative states (i.e., same processes and Ds values) had its own evolver and transition procedure. In Mk2 simulators have just one evolver and one transition procedure.

An evolver looks like a traditional numerical simulator. It contains three sections: (1) calculate the derivatives of independent parameters and integrate them; (2) update values of dependent parameters; (3) update values of boolean parameters marking qualitative changes. Let the *influence graph* be the graph whose nodes are quantities and whose arcs the influences (direct or indirect) implied by a model (note that many can't co-occur). We assume that the subset of the influence graph consisting of indirect influence arcs is loop-free. This *unidirectional assumption* allows us to update dependent parameters in a fixed global order. While we may have to check whether or not to update a quantity (e.g., the level of a liquid which doesn't exist) or calculate which potential direct influences are relevant (e.g., which flows into and out of a container are active), we never have to change the order in which we update a pair of parameters (e.g., we never have to update level using pressure at one time and update pressure using level at another within one simulator). The code generation algorithm is:

[1] Empirically, setting $\mathcal{E}_{max}$ to double the maximum size of the set of preconditions and quantity conditions for $\mathcal{DT}$ always provides accurate labels for the relevant subset of the ATMS. The factor of two ensures accurate labels when computing limit hypotheses.

[2] Under some tradeoffs non-exponential algorithms may be possible: See Section .

[3] Transition procedures also enforce completeness of the qualitative record by signalling when the simulator should "roll back" to find a skipped transition [10].

Sample of direct influence update code

```
(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))
  QuantityConditions ((greater-than (A (temperature ?src))
                                    (A (temperature ?dst))))
  Relations ((quantity flow-rate)
             (Q= flow-rate (- (temperature ?src) (temperature ?dst))))
  Influences ((I+ (heat ?dst) (A flow-rate))
             (I- (heat ?src) (A flow-rate))))
```

```
(SETF (VALUE-OF (D (HEAT (C-S WATER LIQUID F))) AFTER) 0.0)
(WHEN (EQ (VALUE-OF (ACTIVE PI0) BEFORE) ':TRUE)
  (SETF (VALUE-OF (D (HEAT (C-S WATER LIQUID F))) AFTER)
        (- (VALUE-OF (D (HEAT (C-S WATER LIQUID F))) AFTER)
          (VALUE-OF (A (HEAT-FLOW-RATE PI0)) BEFORE))))
(WHEN (EQ (VALUE-OF (ACTIVE PI1) BEFORE) ':TRUE)
  (SETF (VALUE-OF (D (HEAT (C-S WATER LIQUID F))) AFTER)
        (+ (VALUE-OF (D (HEAT (C-S WATER LIQUID F))) AFTER)
          (VALUE-OF (A (HEAT-FLOW-RATE PI1)) BEFORE))))
(SETF (VALUE-OF (A (HEAT (C-S WATER LIQUID F))) AFTER)
      (+ (VALUE-OF (A (HEAT (C-S WATER LIQUID F))) BEFORE)
        (* DELTA-T
          (VALUE-OF (D (HEAT (C-S WATER LIQUID F))) AFTER))))
```

Sample of indirect influence update code

```
(defentity (Contained-Liquid (C-S ?sub liquid ?can))
  (quantity (level (C-S ?sub liquid ?can)))
  (quantity (Pressure (C-S ?sub liquid ?can)))
  (Function-Spec Level-Function
             (Qprop (level (C-S ?sub liquid ?can))
                    (Amount-of (C-S ?sub liquid ?can))))
  (Correspondence ((A (level (C-S ?sub liquid ?can)))
                   (A (bottom-height ?can)))
                  ((A (amount-of (C-S ?sub liquid ?can))) zero))
  (Function-Spec P-L-Function
             (Qprop (pressure (C-S ?sub liquid ?can))
                    (level (C-S ?sub liquid ?can)))))
```

```
(COND ((EQ :GREATER-THAN
          (COMPUTE-SIGN-FROM-FLOAT
           (VALUE-OF (A (AMOUNT-OF-IN WATER LIQUID F)) BEFORE)))
       (SETF (VALUE-OF (A (LEVEL (C-S WATER LIQUID F))) AFTER)
             (/ (VALUE-OF (A (AMOUNT-OF (C-S WATER LIQUID F))) AFTER)
               (* 31.353094 (VALUE-OF (A (DENSITY WATER)) AFTER)
                 (VALUE-OF (A (RADIUS F)) AFTER)
                 (VALUE-OF (A (RADIUS F)) AFTER))))
       (SETF (VALUE-OF (D (LEVEL (C-S WATER LIQUID F))) AFTER)
             (- (VALUE-OF (A (LEVEL (C-S WATER LIQUID F))) AFTER)
               (VALUE-OF (A (LEVEL (C-S WATER LIQUID F))) BEFORE))))
      (T (SETF (VALUE-OF (A (LEVEL (C-S WATER LIQUID F))) AFTER)
               (VALUE-OF (A (LEVEL (C-S WATER LIQUID F))) BEFORE))
         (SETF (VALUE-OF (D (LEVEL (C-S WATER LIQUID F))) AFTER)
               0.0)))
```

Sample of boolean update code

```
(defprocess (Liquid-flow ?sub ?src ?dst ?path)
   Individuals ((?sub :type Substance)
                (?src :type Container)
                (?dst :type Container)
                (?src-cl :bind (C-S ?sub LIQUID ?src))
                (?dst-cl :bind (C-S ?sub LIQUID ?dst))
                (?path :type Fluid-Path
                   :conditions
                    (Fluid-Connection ?path ?src ?dst)))
   Preconditions ((aligned ?path))
   QuantityConditions
      ((greater-than (A (pressure ?src-cl)) (A (pressure ?dst-cl))))
   Relations ((quantity flow-rate)
             (Q= flow-rate (- (pressure ?src-cl) (pressure ?dst-cl)))
             • • •)
   Influences ((I+ (Amount-of-in ?sub LIQUID ?dst) (A flow-rate))
              • • •))
```

```
(SETF (VALUE-OF (ACTIVE PI0) AFTER)
   (IF (AND
         (EQ :GREATER-THAN
           (COMPUTE-INEQUALITY-FROM-FLOATS
             (VALUE-OF (A (PRESSURE (C-S WATER LIQUID F))) AFTER)
             (VALUE-OF (A (PRESSURE (C-S WATER LIQUID G))) AFTER)))
         (EQ (VALUE-OF (ALIGNED P1) AFTER) ':TRUE))
     ':TRUE ':FALSE))
```

Figure 1: Code fragments produced by MK2. The relevant model fragments are shown on the left, the corresponding sample code fragments are shown on the right.

1. Analyze the influence graph to classify parameters as directly or indirectly influenced, and establish a global order of computation.

2. Generate code for each directly influenced quantity. Update order is irrelevant because the code for each summation term is independent.

3. Generate code to update indirectly influenced quantities using the quantitative models found earlier. Updates are sequential, based on the ordering imposed by the influence graph.

4. Generate code to update $\mathcal{B}$, using label and dependency information.

Figure 1 shows part of an evolver produced this way. Step 1 is quadratic in the number of quantities and the rest is linear, so the algorithm is efficient. The code generation algorithm for transition procedures is linear in the number of comparisons:

1. Sort limit hypotheses into equivalence classes based on what they compare. For instance, the hypothesis that two pressures become unequal and the hypothesis that they become equal both concern the same pair of numbers and so are grouped together.

2. For each comparison, generate code to test for the occurrence of the hypotheses and for transition skip (see [10] for details). To avoid numerical problems, place tests for equality first whenever needed.

Explanation generation

Explanations in MK1 were cheap to compute because the envisionment was part of the simulator. The value of $\mathcal{Q}$ at any time provided a complete causal structure and potential state transitions. In MK2 every self-explanatory simulator now maintains instead a *concise history* [18] for each boolean in $\mathcal{B}$. The temporal bounds of each interval are the time calculated for that

Table 1: MK2 on small examples
All times in seconds. The envisioning time is included for comparison purposes.

Example	Qualitative Analysis	Code Generation	Envisioning
Two containers	19.4	3.4	40.2
Boiling water	21.8	3.4	45.6
Spring-Block	4.9	1.5	6.2

interval in the simulation. Elements of $\mathcal{N}$ can also be selected for recording as well, but these are only necessary to provide quantitative answers. A compact *structured explanation* system, which replicates the ontology of the original QP model, is included in the simulator to provide a physical interpretation for elements of $\mathcal{B}$ in a dependency network optimized for explanation generation.

Surprisingly, almost no explanatory power is lost in moving from envisionments to concise histories. For instance, histories suffice to determine what influences and what mathematical models hold at any time. What is lost is the ability to do cheap counterfactuals: e.g., asking "what might have happened instead?". Envisionments make such queries cheap because all alternate state transitions are precomputed. Such queries might be supported in MK2's simulators by incorporating qualitative reasoning algorithms that operated over the structured explanation system.

Self-Monitoring

In MK1 clashes between qualitative and quantitative models were detected by a simulator producing an inconsistent state: i.e., when $\mathcal{N}$ could not satisfy $\mathcal{Q}$. This stringent self-monitoring is impossible to achieve without envisioning. To scale up we must find a good compromise between stringency and performance. Our compromise is to search the nogood database generated by the ATMS during the qualitative analysis phase for useful local consistency tests. These tests are then proceduralized into a *nogood checker* which becomes part of the simulator. Empirically, few nogoods are useful since they rule out combinations of beliefs which cannot hold, given that $\mathcal{B}$ is computed from $\mathcal{N}$. Thus so far nogood checkers have tended to be small. How much self-monitoring do we lose? At worst MK2 produces no extra internal consistency checks, making it no worse than many hand-written simulators. This is a small price to pay for the ability to produce code for large artifacts.

Examples

These examples were run on an IBM RS/6000, Model 530, with 128MB of RAM running Lucid Common Lisp. Table reports the MK2 runtimes on the examples of [10]. Here, MK2 is much faster than human coders. To explore MK2's performance on large problems we tested it on a model containing nine containers connected by twelve fluid paths (i.e., a 3×3 grid). The liquid in each container (if any) has two independent variables (mass and internal energy) and three dependent variables (level, pressure, and temperature). 24 liquid flow processes were instantiated, each including rates for transfer of mass and energy. We estimate a total envisionment for this situation would contain over 10^{12} states, clearly beyond explicit generation. The qualitative analysis took 16,189 seconds (over four hours), which is slow but not forever. Generating the code took only 97.3 seconds (under two minutes), which seems reasonably fast.

Analysis

The examples raise two interesting questions: (1) why is code generation so fast and (2) can the qualitative analysis be made even faster?

Code generation is fast for two reasons. First, in programming framing the problem takes a substantial fraction of the time. This job is done by the qualitative analysis. Transforming the causal structure into a procedure given mathematical models is easy, deriving the causal structure to begin with is not. The second reason is that our current implementation does not reason about which mathematical model to use. So far our examples included only one numerical model per combination of qualitative proportionalities.[4] This will not be unusual in practice, since typically each approximation has exactly one quantitative model (e.g., laminar flow versus turbulent flow). Thus the choice of physical model typically forces the choice of quantitative model. On the other hand, we currently require the retrieved model to be executable as is, and do not attempt to optimize for speed or numerical accuracy (e.g. [2; 17]).

The qualitative analysis for large examples could be sped up in several ways. First, our current routines are culled from QPE, hence are designed for envisioning, not this task. Just rewriting them to minimize irrelevant dependency structure could result in substantial speedups. Second, using an ATMS designed to avoid internal exponential explosions could help [5].

A more radical possibility is to not use an ATMS. Some of the jobs performed using ATMS labels in Section can be done without them. Consider the problem of finding quantitative models for indirectly influenced parameters, which requires combining labels for qualitative proportionalities. For some applications it might be assumed that if no quantitative model is known for a combination of qualitative proportionalities then that combination cannot actually occur. Computing the labels of influences is unnecessary in such cases. Sometimes ignoring labels might lead to producing code which would never be executed (e.g., boiling iron in a steam plant). At worst speed in qualitative analysis

[4]If there are multiple quantitative models the current MK2 selects one at random

can be traded off against larger (and perhaps buggy) simulation code; At the best faster reasoning techniques can be found to provide the same service as an ATMS but with less overhead for this task.

Discussion

SIMGEN.Mk2 demonstrates that qualitative reasoning techniques can scale up. Building self-explanatory simulators requires qualitative analysis, but does not require calculating even a single global state. By avoiding envisioning and other forms of qualitative simulation, we can build simulators for artifacts that no envisionment-based system would dare attempt. Although our implementation is not yet optimized, already it outspeeds human programmers on small models and does reasonably well on models within the range of utility for certain applications in instruction, training, and design. Our next step is to build a version of Mk2 which can support conceptual design and supply simulators for procedures trainers and integration into hypermedia systems.

Acknowledgements

We thank Greg Siegle for intrepid CLIM programming. This work was supported by grants from NASA JSC, NASA LRC, Xerox PARC, and IBM.

References

[1] Abbott, K. "Robust operative diagnosis as problem-solving in a hypothesis space", Proceedings of AAAI-88, August, 1988.

[2] Abelson, H. and Sussman, G. J. The Dynamicist's Workbench: I Automatic preparation of numerical experiments MIT AI Lab Memo No. 955, May, 1987.

[3] Addanki, S., Cremonini, R., and Penberthy, J.S. "Graphs of Models", *Artificial Intelligence*, **51**, 1991.

[4] Collins, J. and Forbus, K. "Building qualitative models of thermodynamic processes", *unpublished manuscript.*

[5] DeCoste, D. and Collins, J. "CATMS: An ATMS which avoids label explosions", *Proceedings of AAAI-91*, Anaheim, CA., 1991.

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

[7] Falkenhainer, B and Forbus, K. D. Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 51(1-3):95–143, October 1991.

[8] Forbus, K. D. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[9] Forbus, K. The qualitative process engine, A study in assumption-based truth maintenance. *International Journal for Artificial Intelligence in Engineering*, 3(3):200–215, 1988.

[10] Forbus, K. D. and Falkenhainer, B. Self-Explanatory Simulations: An integration of qualitative and quantitative knowledge. *AAAI-90*, July, 1990.

[11] Franks, R.E. *Modeling and simulation in chemical engineering*, John Wiley and Sons, New York, 1972.

[12] Haug, E.J. *Computer-Aided Kinematics and Dynamics of Mechanical Systems Volume I: Basic Methods*, Allyn and Bacon, 1989.

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

[14] Kuipers, B. and Chiu, C. "Taming intractable branching in qualitative simulation", *Proceedings of IJCAI-87*, Milan, Italy, 1987.

[15] LeClair, S., Abrams, F., and Matejka, R. "Qualitative Process Automation: Self-directed manufacture of composite materials", *AI EDAM*, 3(2), pp 125-136, 1989.

[16] Sacks, E. "Automatic qualitative analysis of dynamic systems using piecewise linear approximations", *Artificial Intelligence*, **41**, 1990.

[17] Weld, D. "Approximation reformulations", Proceedings of AAAI-90, August, 1990.

[18] Williams, B. "Doing time: putting qualitative reasoning on firmer ground" *Proceedings of AAAI-86*, Philadelphia, Pa., 1986.

[19] Yip, K. "Understanding complex dynamics by visual and symbolic reasoning", *Artificial Intelligence*, **51**, 1991.

[20] Zhao, F. "Extracting and representing qualitative behaviors of complex systems in phase spaces" *Proceedings of IJCAI-91*, Sydney, Australia, 1991.

[21] Zippel, R. Symbolic/Numeric Techniques in Modeling and Simulation. In *Symbolic and Numerical Computations – Towards Integration*. Academic Press, 1991.