

Real-Time Self-Explanatory Simulation

Franz G. Amador and Adam Finkelstein and Daniel S. Weld*

Department of Computer Science and Engineering, FR-35

University of Washington

Seattle, Washington 98195

franz, adam, weld@cs.washington.edu

Abstract

We present Pika, an implemented self-explanatory simulator that is more than 5000 times faster than SimGen Mk2 [Forbus and Falkenhainer, 1992], the previous state of the art. Like SimGen, Pika automatically prepares and runs a numeric simulation of a physical device specified as a particular instantiation of a general domain theory, and it is capable of explaining its reasoning and the simulated behavior. Unlike SimGen, Pika's modeling language allows arbitrary algebraic and differential equations with no prespecified causal direction; Pika infers the appropriate causality and solves the equations as necessary to prepare for numeric integration.

Introduction

Science and engineering have used numeric simulation productively for years. Simulation programs, however, have been laboriously hand-crafted, intricate, and difficult to understand and change. There has been much recent work on automating their construction (*e.g.* [Yang, 1992, Rosenberg and Karnopp, 1983, Abelson and Sussman, 1987, Palmer and Cremer, 1992]). To this, the Qualitative Physics community has contributed the idea of a *self-explanatory simulator* [Forbus and Falkenhainer, 1990, Forbus and Falkenhainer, 1992]. When using such a system, a person need only specify the basic entities, quantities, and equations governing the system to be simulated. From these, the program automatically prepares and runs a numeric simulation. It also keeps a record of its reasoning so it can explain the simulated behavior.

Such a simulator has three primary advantages [Forbus and Falkenhainer, 1990].

*Many thanks to the members of the E³ project, especially Mike Salisbury and Dorothy Neville. Pandu Nayak kindly provided Common Lisp causal ordering code. Elisha Sacks and Eric Boesch kindly provided the Runge-Kutta numeric integration code. This research was funded in part by National Science Foundation Grant IRI-8957302, Office of Naval Research Grant 90-J-1904, and the Xerox corporation.

- **Improved automation:** Because the user specifies the simulated system declaratively using equations, creating and modifying simulations is much easier.
- **Improved self-monitoring:** The simulator can analyze the equations to produce checks that detect problems with the simulation, such as numerical instability.
- **Better explanations:** Because the simulator records the deductions needed to prepare the simulation, it can generate custom English-language or graphical explanations for the simulated behavior. Such explanations assist debugging the simulated equations, and they can form the core of an automated tutor that allows the user to explore and learn about the behavior of a simulated system.

The first and third of these properties are of particular interest to the Electronic Encyclopedia/Exploratorium (E³) project at the University of Washington. We are constructing a program via which the user may interact with simulated versions of various engineered artifacts to learn how they work [Amador *et al.*, 1993]. The user can perturb the environment of the device to see how it reacts and even modify the device itself as it is "operating," all the while receiving English or graphical explanations for its behavior. Thus a central part of the E³ program is effectively a combined CAD system and self-explanatory simulator.

Unfortunately, existing self-explanatory simulators, namely the SimGen Mk1 and SimGen Mk2 programs [Forbus and Falkenhainer, 1990, Forbus and Falkenhainer, 1992], do not meet our needs.

- **Too slow:** Both compile the equation model of an artifact into a custom program that simulates it. While SimGen Mk2 is much faster than SimGen Mk1, this compilation process still takes much too long to allow prompt response to user manipulations of the artifact model. For example, SimGen Mk2 requires 4 hours to compile a simulator for a model of 9 containers and 12 pipes [Forbus and Falkenhainer, 1992].
- **Restrictive modeling language:** As with their

predecessor Qualitative Process Theory [Forbus, 1984], the SimGen programs require equations to be written as uni-directional *influences*. Thus the flow of causality and information through the model must be chosen by the modeler and remains fixed. We claim it is easier and more natural to describe a model using ordinary non-directional equations. (See "Equations" below.)

In this paper we present Pika¹, an implemented self-explanatory simulator which overcomes these limitations. Its first, unoptimized implementation prepares the above SimGen Mk2 example simulation in under 3 seconds—more than 5000 times faster. Furthermore, the modeling language is based upon ordinary differential and algebraic equations, which Pika automatically manipulates as necessary to simulate the model.

The modeling language

As with the SimGen programs, our modeling language (known as the Quantified Modeling Language (QML)) derives from the Qualitative Process Theory [Forbus, 1984]. The user defines a model in two parts. The physics that apply to the device are defined in a *domain theory*, which is general and can be reused when modeling different devices. This theory is instantiated according to a *scenario description*, which specifies a particular device for simulation. For example, a domain theory might describe the various types of electrical components while a corresponding scenario description would specify a particular circuit.

QML has two distinguishing features: a simplified modeling language and non-directional equations.

Model Fragments

Qualitative Process Theory's *entity*, *process*, and *view* definitions are replaced by *model fragments* (MFs), of which there are two kinds. *Unquantified* MFs (similar to QP Theory's entities) are instantiated explicitly in the scenario description. *Quantified* MFs (similar to QP Theory's processes and views) are instantiated automatically by the system when their preconditions are met, and their instances are likewise destroyed when those preconditions no longer hold. Here, for example, is a simplistic characterization of boiling.

```
(define-MF boiling (?fluid)
  (preconditions
    (instance-of Liquid ?fluid)
    (>= (temperature ?fluid)
      (boiling-temperature ?fluid)))
  (effects
    (dyn-infl (mass ?fluid)
      (- (/ (heat-absorption-rate ?fluid)
        (latent-heat ?fluid))))))
```

The *dyn-infl* (dynamic influence) is a non-directional version of a QP-theory "direct influence"

¹A pika (pronounced pee-kuh) is a small mammal that lives in alpine rockpiles.

written using terminology first advocated by Woods [Woods, 1991]. Pika sums all dynamic influences upon a quantity to form an equation that constrains its derivative. Thus if boiling is the only active MF, the derivative of mass is constrained to be equal to the ratio of absorption rate and latent heat of vaporization. However, if an MF encoding fluid flow into the container were active then that influence would be added to the sum constraining the derivative. Algebraic influences (*alg-infl*) are similar—they specify an implicit summation constraining the influenced quantity itself.

Equations

Pika treats all QML equations as non-directional constraints. This follows standard scientific practice better than do QP Theory's one-directional influences. Scientists express almost all physical laws as constraint equations. Ohm's Law ($V = IR$), for example, makes no commitment as to which variables are dependent and which are independent. In different contexts, such an equation can determine the value of any of its variables. If a resistor MF containing an Ohm's-Law equation appears in a model in which it is connected to a constant-voltage source, Pika will use the equation to find the current through the resistor. If the resistor is instead connected to a constant-current source, the equation determines the voltage drop across it.

Non-directional equation constraints make model writing much easier. The modeler can write the ideal gas law ($PV = nRT$) in its familiar form, without having to decide how it will be used in future simulations. Even QML's "influences" specify non-directional equations. The dynamic influence in the boiling MF above provides the modularity of QP Theory's direct influences without making any commitment to causal direction. Such modular specification of non-directional summation equations provides the modeler with considerable expressive power. For example, to encode Kirchhoff's current rule for electrical nodes, the modeler need simply include the equation ($=$ (net-current ?self) 0) in a Node MF, together with (*alg-infl* (net-current ?node) (current ?terminal)) in an MF that will be instantiated for each connected node and terminal. From this, Pika will create an equation for each node that constrains the sum of the currents of its connected terminals to be zero. Note that the "influenced" quantity remains constant; causality flows among the "influencing" terminal currents as appropriate. This use of influences is impossible in QP Theory.

The simulation algorithm

Unlike the SimGen programs (known collectively hereafter as "SimGen"), Pika does not compile the artifact model before simulating it. However, it does compile the domain theory once after it is written or changed. Pika's domain theory compiler translates each model

fragment into a set of functions that speed simulation: a model fragment's preconditions are compiled into functions that query the knowledge base to find bindings for the MF's arguments, that test whether the quantitative part of the preconditions is satisfied, and that generate numeric integration bounding conditions that halt integration when those quantitative preconditions cease to be satisfied (or become satisfied, given that the non-quantitative preconditions are met). The model fragment's effects are compiled into functions that assert those effects when the MF is activated and that retract them when it is deactivated.

This compilation requires little time (less than the LISP compiler requires to compile the resulting code). Furthermore, one need not suffer even this small cost except when the domain theory changes, which happens quite infrequently compared with changes to the model being simulated.

Since Pika is still being integrated with the E³ user interface [Salisbury and Borning, 1993], its current implementation runs in a "batch" manner. Pika takes as input the compiled domain theory, the scenario description, and a period of time for which to simulate. It simulates for the requested time, recording its reasoning and the device's behavior, and then stops to answer the user's questions.

Pika's simulation algorithm is as follows:

```

Instantiate unquantified MFs from scenario description
Repeat until time bound reached
  Instantiate and destantiate quantified MFs based
    upon world state
  Causally order the equations
  Solve the equations for the quantities they determine
  Create integration bounds from MF preconditions
  Numerically integrate until a bound is violated

```

Pika's algorithm differs from SimGen's in three important ways: MF activation, numeric integration, and equation manipulation.

Model fragment activation

Both SimGen programs use an assumption-based truth maintenance system (ATMS) [de Kleer, 1986] to perform substantial analysis during model compilation. SimGen Mk1 generates a total envisionment of the model's qualitative state space, which is computationally infeasible for large models. SimGen Mk2 reduces this cost by finding only the "local states" in which each MF is active. This analysis allows them to reduce the run-time checking needed to determine changes in the set of active MFs. If the simulation is in a qualitative state from which there is only one possible transition, then the simulator need check only the limit hypothesis corresponding to that transition, and it can switch directly to the set of active MFs determined during compilation to be active in the next qualitative state. This speeds simulation, but it exacts an enormous cost during compilation.

Since Pika does no such compile-time analysis, it must test all quantified MF preconditions at runtime to determine the active set. For each MF it queries the knowledge base for all argument bindings that meet the non-quantitative preconditions (e.g. the *instance-of* test in the boiling example). In the worst case, this is exponential in the number of MF arguments, but their number is under the modeler's control and is always small.² Pika then tests the quantitative preconditions of these candidate MF activations, which requires time linear in the size of the quantitative expressions.

Numeric integration

SimGen uses custom-generated *evolver procedures* (which use Euler's method) to do numeric integration and *state transition procedures* to detect transitions in qualitative state. Because Pika does not compile the model, it must instead use a general-purpose numeric integrator. Its current implementation uses a fourth-order Runge-Kutta integrator with adaptive step-size control [Press *et al.*, 1986], which, at a given accuracy, is much faster than Euler's method.³

In addition to the simulation equations, initial quantity values, and integration limit, the integrator also takes as input a set of *integration bounds*. Each bound is an expression and an interval; if the expression's value ever leaves the interval, the integrator halts (at the time step immediately before the bound is violated). Pika supplies bounds representing the quantitative preconditions of all currently active model fragments (known as *deactivation bounds*) and other bounds representing the quantitative preconditions of all MFs that are inactive only because of their quantitative preconditions (*activation bounds*).

Equation manipulation

All the equations in a SimGen model are written either as direct influences or as qualitative proportionalities (indirect influences). SimGen converts these into numeric integration equations by 1) summing the direct influences upon each quantity's derivative, 2) sorting the indirect influences into a graph (causal ordering) such that all quantities are determined, and 3) converting the influence subgraph that determines each quantity into an algebraic equation via a table (known as the math library). Since this table contains a different equation for each possible combination of influences upon each quantity, any given qualitative proportionality can "mean" different things in different contexts.

This arrangement implies a "two-tiered" process of quantitative model construction: the domain theory is instantiated based upon MF preconditions to produce

²Note that SimGen must also confront this exponential when instantiating MFs into the ATMS.

³It is important to emphasize that Pika's performance advantage over SimGen is not due to the underlying integration technology—the important speed-up is in simulation preparation.

a qualitative model, and then the math library is instantiated based upon the qualitative model influences to produce the quantitative model. This structure may make writing some kinds of models easier, but it requires the modeler to write every equation twice: once qualitatively for the domain theory, and once quantitatively for the math library. It also sacrifices possible modularity. Influences allow the modeler to specify qualitative equations in pieces that are automatically assembled by SimGen; however, the modeler must fully specify all possible quantitative model equations.

With Pika, the modeler specifies the domain theory using equations which Pika automatically combines and symbolically manipulates as needed to form the quantitative model. QML thus effectively collapses SimGen's two-tiered structure into one, allowing modular specification of quantitative model equations.

Pika must convert the model's non-directional equations into the following directional form expected by the Runge-Kutta integrator:

$$\begin{aligned} \frac{dX_1}{dt} &= f_1(X_1, \dots, X_n, t) & Y_1 &= g_1(X_1, \dots, X_n, t) \\ &\vdots & &\vdots \\ \frac{dX_n}{dt} &= f_n(X_1, \dots, X_n, t) & Y_m &= g_m(X_1, \dots, X_n, t) \end{aligned}$$

The X 's are the state variable used by the integrator to advance time; the Y 's are all other variables calculated from the state variables. Pika classifies any quantity that has a derivative (generally due to a *dyn-infl*) as a state variable. It rearranges the equations into the above form by first using a causal ordering routine [Iwasaki and Simon, 1986, Serrano and Gossard, 1987] to find an order in which the equations can be evaluated so as to determine values for all quantities; this ordering may include sets of simultaneous equations. Pika then uses Mathematica's [Wolfram, 1988] *Reduce* function to solve each equation for the quantity it determines (unless it is already in "solved" form, i.e. (= *determined-quantity expression*)). Mathematica also solves any simultaneity for the quantities it determines. However, since causal ordering abstracts equations to sets of quantities, it can falsely group non-independent equations as simultaneities. Here we rely upon the fact that *Reduce*, if it cannot find a solution, will reformulate the equations, discarding non-independent ones, so that another attempt at causal ordering will not make the same mistake. This process repeats until the causal ordering contains no simultaneities.

The modeler may denote some quantities as constants. Non-constant quantities retain their previous values if they are not determinable from the state variables. Implementing these semantics requires that the equation-directionalizing process make several passes. First, all constants are marked as exogenous, and causal ordering and equation solving discover which quantities are determinable from them. Next, those state variables that remain undetermined (usually all of them) are marked as exogenous, and the algorithm

Q: Summarize the simulated behavior.

A: At time 0, heat started flowing from STOVE to CAN-OF-WATER.

At time 55.96147, the temperature of CAN-OF-WATER reached 100.0, and it started boiling.

At time 55.969383, a gas appeared in CAN-OF-WATER.

At time 95.961464, the liquid in CAN-OF-WATER boiled away, and it stopped boiling.

At time 165.31618, the pressure of CAN-OF-WATER exceeded 150.0, and the container exploded.

Q: What is the value of (TEMPERATURE CAN-OF-WATER) at time 40?

A: (TEMPERATURE CAN-OF-WATER) is 82.10323 at time 40.

Q: How is (TEMPERATURE CAN-OF-WATER) changing?

A: (TEMPERATURE CAN-OF-WATER) is increasing at time 40.

Q: What happens next?

A: At time 55.96147, the temperature of CAN-OF-WATER reached 100.0, and it started boiling.

Figure 1: Example of explanation generation. Queries are translations from a specialized query language; answers are actual program output.

runs again. Lastly, all remaining undetermined quantities are given their previous values and treated as constants under the current set of equations. A feature of this algorithm is that the modeler can force a state variable to be constant during one operating region while allowing it to vary during another.

Explanations

By keeping a record of its equation manipulations, the history of model fragment activations and deactivations, and the data returned by numeric integration, Pika can answer the class of questions answerable by SimGen Mk2. This includes summarizing so-far-simulated behavior in qualitative terms, reporting the equation that determines the value of any quantity at any simulated time, and reporting the value of a quantity at any simulated time. It objects if the quantity does not exist at the requested time. Because it does no global envisioning, Pika (like SimGen Mk2) cannot answer some questions answerable by SimGen Mk1, such as summarizing currently unsimulated future behavior and describing alternative behaviors. See figure 1 for an example.

Implementation status

PIKA is fully implemented. It is written in Allegro Common Lisp and uses the LOOM knowledge representation system (version 1.4.1) [Brill, 1991], the Mathematica symbolic math system (version 2.0) [Wolfram,

Test	Prep ⁴	Total ⁵	RK % ⁶	Mma % ⁷
SimGen Mk2: ⁸ 9 cont & 12 pipe	2.7s	3.1s	6	0
36 cont & 60 pipe ⁹	34s	38s	6	0
2-rung RC ladder ¹⁰	4.9s	5.1s	1.5	70
5-rung RC ladder	38s	38s	0.0006	87
Exploding can ¹¹	0.5s	1.8s	5	40

Figure 2: Timing data (Sun SPARCstation IPX).

1988], and a Runge-Kutta numeric integration package that is written in C [Press *et al.*, 1986]. See figure 2 for timing data.

The “prep” column is what we are comparing to SimGen’s model-compilation time; it gives the time needed to prepare each model for simulation. This table demonstrates PIKA’s speed and scalability for models that do not produce large sets of simultaneous equations. However, the 2-rung and 5-rung “RC ladder” electrical circuit tests require solving sets of 21 and 51 simultaneous equations, respectively, and suffer accordingly.

Second prototype

One way to reduce the impact of equation manipulation is to avoid redoing it when unnecessary. Pika regenerates the simulation equations “from scratch” every time the set of active MFs changes. We have reimplemented Pika (as Pika2) using SkyBlue [Sannella, 1992], a hierarchical constraint manager. SkyBlue effectively maintains a causal-ordering graph which can be updated incrementally as MFs activate and deactivate. Only those equations whose causal direction changes (or which form new simultaneities) must Pika2 resolve. Also, Pika2 caches solutions of individual equations, though not of simultaneities.

SkyBlue offers other advantages over “traditional” causal ordering methods. Each constraint (a set of variables) has a specified strength. SkyBlue builds the causal-ordering graph from the highest-strength, consistent set of constraints, leaving some lower-strength constraints unused if necessary. Pika2 uses this strength hierarchy to implement the semantics of con-

stants, state variables, and value persistence without having to repeatedly run a causal-ordering procedure. For example, every quantity has an associated low-strength equation that sets it equal to its most recent simulated value. SkyBlue includes the constraint representing this equation in the causal ordering only if the quantity is not otherwise determined.

SkyBlue also allows one-way constraints, which Pika2 uses to represent the fact that the derivative of a state variable is numerically integrated to determine the state variable’s next value, but not vice versa. This allows a more accurate definition of a state variable than Pika uses: a state variable is a quantity having a derivative that can be causally determined if the quantity is assumed to be a state variable (and hence exogenous to each time step). This definition better reflects the cyclic nature of numeric integration, and SkyBlue will use a one-way constraint between derivative and possible state variable only when the definition is satisfied.

Pika2’s initial simulation-preparation times are about the same as Pika’s, but unfortunately it is not yet stable enough for timings demonstrating the value of incremental constraint management.

Related work

An important inspiration for much work in self-explanatory simulation was the STEAMER project [Holland *et al.*, 1984], which produced an impressive interactive simulator/tutor for a naval propulsion system, though all the simulations and explanations were hand-crafted in advance.

Many people have worked on easing the construction of fast, accurate numeric simulations. The iSIMILE and MISIM systems [Yang and Yang, 1989, Yang, 1992], for example, provide tools for constructing a variety of electrical and optical circuit simulations. The modeler must define new components using a subset of FORTRAN, however, and the systems do no equation manipulation. The ENPORT program [Rosenberg and Karnopp, 1983] generates numeric simulations from the more declarative bond-graph system representation, and the *Dynamicist’s Workbench* project [Abelson and Sussman, 1987] generates simulations from equation models. None of these systems, however, allow changes in the equations during simulation.

Besides SimGen, the system closest in spirit to our own is SimLab [Palmer and Cremer, 1992], which allows a model-fragment-like specification of equation-based models that it symbolically manipulates to produce numeric simulations. SimLab does not, however, allow changes in the equations during simulation, nor does it generate explanations.

We note that the “How Things Work” project at Stanford [Fikes *et al.*, 1992] is addressing issues similar to those those tackled by Pika; however, the Stanford work is too preliminary to discuss extensively.

⁴Elapsed time spent before the first numeric integration.

⁵Total simulation elapsed time. The container/pipe and RC ladder tests were simulated until “quiescence”, i.e. until all quantities had completed 99% of their possible change.

⁶Percent of total time spent doing numeric integration.

⁷Percent of total time spent solving equations.

⁸Our implementation of the example described in [Forbus and Falkenhainer, 1992].

⁹The SimGen Mk2 example container grid quadrupled.

¹⁰Each “rung” of an “RC ladder” is a capacitor with some initial voltage in series with a resistor. All rungs are connected in parallel.

¹¹See figure 1.

Future work

Pika is fast, but it isn't quite fast enough to drive a truly interactive simulation for the E³ project. We estimate that we need another factor of ten for practical use and are working on several ways to speed it up.

Pika currently uses LOOM [Brill, 1991] for its knowledge base, but it uses almost none of LOOM's inferencing power. Switching to LOOM's CLOS subset, or abandoning LOOM altogether, should significantly speed Pika.

Using Mathematica to solve equations dramatically slows Pika; solving a set of a dozen linear simultaneous equations can take several seconds. Pika prepares the SimGen Mk2 example in under 3 seconds partly because that model requires no equation solving. Using a dedicated linear-equation solver instead (when possible) should help.

Conclusions

We have presented Pika, a self-explanatory simulator 5000 times faster than SimGen Mk2, the previous state of the art. Pika also provides a more natural and more expressive modeling language based upon non-directional algebraic and differential equation constraints. Pika and the SimGen programs represent points along a continuum: SimGen Mk1 does an enormous amount of model analysis prior to simulation, SimGen Mk2 does less, and Pika does almost none. Pika must therefore pay a greater cost when changing the model during simulation. The highly interactive nature of simulation in the E³ project demands such an architecture. However, the performance results suggest that Pika's strategy may work well for other applications.

References

- H. Abelson and G.J. Sussman. The Dynamicist's Workbench: I Automatic Preparation of Numerical Experiments. AI Memo 955, MIT AI Lab, May 1987.
- Franz G. Amador, Deborah Berman, Alan Borning, Tony DeRose, Adam Finkelstein, Dorothy Neville, David Notkin, David Salesin, Mike Salisbury, Joe Sherman, Ying Sun, Daniel S. Weld, and Georges Winkenbach. Electronic "How Things Work" Articles: Two Early Prototypes. *IEEE Transactions on Knowledge and Data Engineering*, To Appear 1993.
- D. Brill. *LOOM Reference Manual*. USC-ISI, 4353 Park Terrace Drive, Westlake Village, CA 91361, version 1.4 edition, August 1991.
- J. de Kleer. An Assumption-based Truth Maintenance System. *Artificial Intelligence*, 28, 1986.
- R. Fikes, T. Gruber, and I. Iwasaki. The Stanford How Things Work Project. In *Working Notes of the AAAI Fall Symposium on Design from Physical Principles*, pages 88-91, October 1992.
- K. Forbus and B. Falkenhainer. Self-Explanatory Simulations: An integration of qualitative and quantitative knowledge. In *Proceedings of AAAI-90*, pages 380-387, 1990.
- K. Forbus and B. Falkenhainer. Self-Explanatory Simulations: Scaling Up to Large Models. In *Proceedings of AAAI-92*, page To Appear, 1992.
- K. Forbus. Qualitative Process Theory. *Artificial Intelligence*, 24, December 1984. Reprinted in [Weld and de Kleer, 1989].
- J. Holland, E. Hutchins, and L. Weitzman. STEAMER: An interactive inspectable simulation-based training system. *AI Magazine*, Summer 1984.
- Y. Iwasaki and H. Simon. Causality In Device Behavior. *Artificial Intelligence*, 29(1):3-32, July 1986. Reprinted in [Weld and de Kleer, 1989].
- R.S. Palmer and J.F. Cremer. SimLab: Automatically Creating Physical Systems Simulators. In *ASME DSC-Vol. 41*, November 1992.
- W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, editors. *Numerical Recipes*. Cambridge University Press, Cambridge, England, 1986.
- R.C. Rosenberg and D.C. Karnopp. *Introduction to Physical System Dynamics*. McGraw Hill, New York, 1983.
- M. Salisbury and A. Borning. A User Interface for the Electronic Encyclopedia Exploratorium. In *Proceedings of the 1993 International Workshop on Intelligent User Interfaces*, Orlando, FL, January 1993.
- M. Sannella. The SkyBlue Constraint Solver. Technical Report 92-07-02, Department of Computer Science and Engineering, University of Washington, December 1992.
- D. Serrano and D.C. Gossard. Constraint Management in Conceptual Design. In *Knowledge Based Expert Systems in Engineering: Planning and Design*, pages 211-224. Computational Mechanics Publications, 1987.
- D. Weld and J. de Kleer, editors. *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, San Mateo, CA, August 1989.
- S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley, Redwood City, CA, 1988.
- E. Woods. The Hybrid Phenomena Theory. In *Proceedings of the 5th international workshop on qualitative reasoning*, pages 71-76, May 1991.
- A.T. Yang and S.M. Yang. iSMILE: A Novel Circuit Simulation Program with Emphasis on New Device Model Development. In *Proceedings of ACM/IEEE 1989 Design Automation Conference*, pages 630-633, 1989.
- A.T. Yang. *MISIM User's Manual*. Department of Electrical Engineering, University of Washington, 1992.

Appendix: QML Grammar

```
DOMAINDESCRIPTION ::= {DEFINEPREDICATE* DEFINEMODELFRAGMENT*}+

DEFINEPREDICATE ::= (define-predicate (PREDICATENAME VARIABLE+))

DEFINEMODELFRAGMENT ::=  DEFINEQUANTIFIEDMF
                        |  DEFINEEXTENDERMF
                        |  DEFINEUNQUANTIFIEDMF

DEFINEQUANTIFIEDMF ::= (define-MF MFNAME (MFARG+) PRECONDITIONS
                        EFFECTS)
DEFINEEXTENDERMF ::= (define-MF MFNAME (?self) PRECONDITIONS
                        EFFECTS)
;; Extends the description of any entity meeting its preconditions.
DEFINEUNQUANTIFIEDMF ::= (define-MF MFNAME () EFFECTS)

MFNAME ::= NAME
MFARG ::= VARIABLE ;; excluding "?self"

PRECONDITIONS ::= (preconditions PRECONDITION*)
PRECONDITION ::=  MEMBERSHIPTEST
                 |  ENTITYSAMENESSTEST
                 |  QUANTITATIVETEST
                 |  PREDICATETEST

MEMBERSHIPTEST ::= (instance-of MFNAME PREENTITYPATH)
ENTITYSAMENESSTEST ::= (same PREENTITYPATH PREENTITYPATH)
                    | (different PREENTITYPATH PREENTITYPATH)
QUANTITATIVETEST ::= (COMPARATOR PREEXPRESSION PREEXPRESSION)
PREDICATETEST ::= (PREDICATENAME PREENTITYPATH+)

PREENTITYPATH ::= ENTITYPATH ;; excludes "?self" if QuantifiedMF
PREQUANTITYPATH ::= QUANTITYPATH ;; excludes "?self" if QuantifiedMF

EFFECTS ::= (effects EFFECT*)
EFFECT ::=  INHERITANCE
          |  DEFINEQUANTITY
          |  EQUATION
          |  ALGEBRAICINFLUENCE
          |  DYNAMICINFLUENCE
          |  ASSERTPREDICATION
          |  RETRACTPREDICATION
          |  DEFINESLOT
          |  CREATESLOTFILLER
          |  FILLSLOT
          |  CONSISTENCYTEST
          |  REPORTQUANTITIES
          |  SETDEFAULTINITIALVALUE
```



```

INHERITANCE ::= (inherits-from UNQUANTIFIEDMFNAME ENTITYPATH)
UNQUANTIFIEDMFNAME ::= MFNAME ;; of an UnquantifiedMF

DEFINEQUANTITY ::= (define-quantity QUANTITYPATH
                    [ :persistent
                      | :history
                      | :parameter]
                    [:default-initial-value
                     [DEFAULTINITIALVALUE]])
;; Defaults to :persistent. :History is optional hint to state-variable finder.
DEFAULTINITIALVALUE ::= NUMBER

EQUATION ::= (= EXPRESSION EXPRESSION)

ALGEBRAICINFLUENCE ::= (alg-infl QUANTITYPATH EXPRESSION)

DYNAMICINFLUENCE ::= (dyn-infl QUANTITYPATH EXPRESSION)

ASSERTPREDICATION ::= (assert (PREDICATENAME ENTITYPATH+))
;; Persists until explicitly retracted.

RETRACTPREDICATION ::= (retract (PREDICATENAME ENTITYPATH+))

DEFINESLOT ::= (define-slot (SLOTNAME ENTITYPATH))

CREATESLOTFILLER ::= (create-slot-filler UNQUANTIFIEDMFNAME SLOTREFERENCE)

FILLSLOT ::= (fill-slot SLOTREFERENCE ENTITYPATH)

CONSISTENCYTEST ::= (ensure (COMPARATOR EXPRESSION EXPRESSION))
;; Simulation halts if test ever fails.

REPORTQUANTITIES ::= (report QUANTITYPATH+)
;; Simulator returns values of "report" quantities (but saves all quantity values
;; for later explanation).

SETDEFAULTINITIALVALUE ::= (default-initial-value QUANTITYPATH NUMBER)
;; Used to override default initial values of inherited quantities.

```



```

ENTITYPATH ::= ENTITYNAME
              | VARIABLE
              | SLOTREFERENCE
              | INSTANCEOF
              | CREATEANONYMOUSINSTANCE

ENTITYNAME ::= NAME
VARIABLE ::= ?NAME | ?self
;; Using "?self" in the effects of a QuantifiedMF creates its instance entity.
SLOTREFERENCE ::= (SLOTNAME ENTITYPATH)
INSTANCEOF ::= (instance-of QUANTIFIEDMFNAME ENTITYPATH+)
;; Returns the instance of QuantifiedMFName whose args are bound to the EntityPaths.
CREATEANONYMOUSINSTANCE ::= (create-instance UNQUANTIFIEDMFNAME)

QUANTIFIEDMFNAME ::= MFNAME ;; of a QuantifiedMF
SLOTNAME ::= NAME

COMPARATOR ::= < | <= | = | /= | >= | >

EXPRESSION ::= NUMBER
              | QUANTITYPATH
              | (STANDARDFUNCTION EXPRESSION)
              | (UNARYARITHMETICOP EXPRESSION)
              | (BINARYARITHMETICOP EXPRESSION EXPRESSION)
              | (NARYARITHMETICOP EXPRESSION+)

STANDARDFUNCTION ::= expt | exp | log | sqrt | sin | cos | tan
                  | sinh | cosh | tanh | asin | acos | atan
UNARYARITHMETICOP ::= - | /
BINARYARITHMETICOP ::= + | - | * | /
NARYARITHMETICOP ::= + | *

QUANTITYPATH ::= (QUANTITYNAME ENTITYPATH+) | time
QUANTITYNAME ::= NAME

NAME ::= valid LISP identifier not starting with "?"

SCENARIODESCRIPTION ::= { CREATENAMEDINSTANCE
                          | ASSERTFACT
                          | RETRACTFACT
                          | SETCURRENTVALUE }+

CREATENAMEDINSTANCE ::= (create-instance UNQUANTIFIEDMFNAME ENTITYNAME)
ASSERTFACT ::= (assert-fact (PREDICATENAME GROUNDENTITYPATH+))
RETRACTFACT ::= (retract-fact (PREDICATENAME GROUNDENTITYPATH+))
SETCURRENTVALUE ::= (current-value GROUNDQUANTITYPATH NUMBER)

GROUNDENTITYPATH ::= ENTITYPATH ;; that has no variables
GROUNDQUANTITYPATH ::= QUANTITYPATH ;; that has no variables

```