Self-Explanatory Simulations: An integration of qualitative and quantitative knowledge

Kenneth D. Forbus
Qualitative Reasoning Group
Beckman Institute, University of Illinois
405 N. Mathews Street, Urbana IL 61801

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

Abstract

A central goal of qualitative physics is to provide a framework for organizing and using quantitative knowledge. One important use of quantitative knowledge is numerical simulation. While current numerical simulators are powerful, they are often hard to construct, do not reveal the assumptions underlying their construction, and do not produce explanations of the behaviors they predict. This paper shows how to combine qualitative and quantitative models to produce a new class of *self-explanatory simulations* which combine the advantages of both kinds of reasoning. Self-explanatory simulations provide the accuracy of numerical models and the interpretive power of qualitative reasoning. We define what self-explanatory simulations are and show how to construct them automatically. We illustrate their power with some examples generated with an implemented system, SIMGEN. We analyze the limitations of our techniques, and discuss plans for future work.

1 Introduction

A central goal of qualitative physics is to provide a framework for organizing and using quantitative knowledge. One important use of quantitative knowledge is numerical simulation. With recent advances in computational power, numerical simulations are playing an ever increasing role in science and engineering. Yet they have important limitations. Most of today's simulations are built by hand, with the long development time and travails associated with custom software. The physical assumptions underlying the simulation are at best only made explicit in technical reports or documentation, and cannot be accessed by the simulation engine or other reasoning systems using its results. And while numerical simulations are superb at producing sets of numbers representing predictions of system behavior over time, they do not incorporate any mechanism for interpreting their results (save graphics). This paper introduces a new class of *self-explanatory simulations* which integrates methods from qualitative physics to directly address these limitations.

By tightly integrating qualitative knowledge with numerical simulations, we hope to achieve three advantages: *increased automation*, *improved self-monitoring*, and *better explanations*. We describe each in turn.

Increased automation: In most engineering domains numerical simulations are still built by hand. With some exceptions (e.g. SPICE and similar systems for electronic circuits), most simulation tools leave the formulation of physical models to the user. For example, PC-DYSIM [5] supports modeling of dynamic systems [24] by providing a generic simulation engine and standardized graphics routines but lacks a well-tested, standard parts library for building system models. In fact, we have been unable to find any such library describing fluid and thermal systems for engineering thermodynamics. Engineers we talk to agree that such a library would be useful, but they lack ways to organize it. Qualitative physics provides such formalisms for organizing knowledge, so that general model libraries can be built and used by simulation compilers to take on more of the modeling burden.

Improved self-monitoring: One unfortunate consequence of the predominance of hand-crafted simulations is sporadic detection of errors and inconsistencies. Many numerical simulations are designed for a narrow range of behaviors, but often such limitations are only recorded in the mind of the programmer. This can lead to erroneous results for unsuspecting users, such as negative water levels in tanks. By making modeling assumptions explicit, the simulator itself should be able to ensure that its numerical predictions are consistent with the qualitative intuitions.

Better explanations: Computer-based tutors like STEAMER [17; 28] and RBT [31] use a combination of numerical simulators to provide students with a "feel" for a system's dynamics and hand-crafted explanation facilities to tie observed behaviors to principles [16; 11; 31]. We hope to help automate the production of such tutors. Other engineering tasks could benefit from self-explanatory simulations. A designer, for instance, could find what range of parameters leads to the desired set of behaviors, and ascertain what needs to be changed if the desired behaviors are unachievable.

Section 2 describes the structure of self-explanatory simulators and outlines how they can be automatically constructed. Section 3 illustrates these ideas with examples from some simulators generated by SIMGEN, our implemented simulator compiler. Section 4 analyzes our compilation technique. We close with related research and our plans for future work.

2 Self-explanatory Simulations

A self-explanatory simulation integrates qualitative and numerical models to produce accurate predictions and causal explanations of the behavior of continuous phys-

ical systems. Self-explanatory simulators produce numerical simulations of behavior, just as traditional systems do. However, they also can describe what is happening in qualitative terms, and provide a causal explanation of the parameters' behavior at any time during the simulation. In addition, all the modeling assumptions involved in creating the qualitative and numeric models are completely explicit, and hence subject to inspection, review, and revision.

Writing self-explanatory simulators requires knowing a system's qualitatively distinct regions of behavior and the ability to construct mathematical models for each region. These models must then be embedded in a control structure which switches between them as appropriate, and keeps the qualitative and quantitative accounts of behavior in sync to generate useful explanations and detect clashes. We have developed a program which compiles such simulators automatically, to relieve the modeler from this complex chore. Our program, called SIMGEN, takes as inputs (1) a qualitative domain model, (2) a corresponding math-model library, and (3) a specific physical system to model.

Roughly, SIMGEN works like this. The qualitative domain model is used to produce a total envisionment for the physical system. Next the math-model library is used to construct a set of ordinary differential equations for each qualitatively distinct region of behavior identified in the envisionment. A simulation program is written for each set of equations, using the causal account from the qualitative model. The state transitions in the envisionment are used to construct procedures which detect when the set of relevant equations changes. Collectively, these procedures constitute a simulator capable of producing predictions and explanations starting from any valid initial condition of the physical system.

This section describes self-explanatory simulations and how they are built. We begin by examining the domain knowledge required, define an appropriate, integrated notion of state, and describe both their architecture and how they are compiled.

2.1 Integrating qualitative and numerical domain knowledge

For a given physical system and task, a relevant qualitative model can be automatically built from the constructs of a general QP (qualitative process) domain model (c.f. [12; 9; 10]). An envisionment using this model determines the space of possible behaviors to consider. Suppose we design a corresponding quantitative domain model that satisfies the following constraints: (1) All parameters that can change during a behavior must be mentioned in the QP model; (2) all behaviors generated by the numerical model must be predicted and thus explained by the qualitative model (requiring the converse is difficult [22]); and (3) every state-space boundary where the set of governing equations changes is marked by a transition in the qualitative model.[1] In this case the envisionment identifies the set of potentially relevant numerical models, with each qualitative state being governed by a single set of equations. We define a *math model library* to be an association of numerical models to (combinations of) the qualitative proportionalities in a QP domain model. For example, a contained liquid description typically includes the relationship [2]

```
Level(?cl) ∝Q+ Amount-of(?cl)
```

The corresponding numeric entry for cylindrical containers might be (assuming ?cl = (c-s ?sub liquid ?can))

```
(= (A (level (c-s ?sub liquid ?can)))
   (/ (A (amount-of (c-s ?sub liquid ?can)))
      (* PI (A (density ?sub))
         (expt (A (radius ?can)) 2))))
```

where density and radius are numerical constants not appearing in the QP model. Models like these are composed to produce simulation code, as described below.

2.2 State in self-explanatory simulations

Integrating qualitative and quantitative state is a key idea of self-explanatory simulations. For concreteness, consider an envisionment produced by QPE, an envisioner for QP theory [14]. Each qualitative state is defined by a set of assumptions (e.g., an ATMS *environment*), whose consequences are what is true in that state. These assumptions are drawn from classes of statements gleaned from an automatic analysis of the scenario model. For example, if the model for a scenario includes a container can, then the possibility of liquid or gas being in can is important, and hence one of the possible relationships between Amount-of-in(water,gas,can) and ZERO must be included in each qualitative state. Similarly, if a pair of containers is connected by a fluid path, then the possible relationships between their pressures becomes one of the constituents of state, since this information is needed (along with other facts) to ascertain whether or not a liquid flow is occurring between them. In addition to inequalities, other classes of assumptions needed to establish state properties are identified and included as well. For example, an assumption about whether or not the fluid path is blocked is essential to knowing if flow can occur, and hence must be included in a state.

The constituents of a qualitative state are thus a set of propositions, drawn from a set of choices that can be considered the *basis set* for qualitative states. Many (indeed, most) of these propositions are ordinal relationships between continuous parameters. A traditional numerical state, on the other hand, consists of a

[1] Enforcing the converse, that every qualitative transition corresponds to a change in equations, would be useful for minimizing complexity but cannot always be done. Modeling an indicator turning on at 10% below a critical value, for instance, requires a transition without any change of equations.

[2] In QP theory [12], $a \propto_{Q+} b$ is an *indirect influence* and reads "a is qualitatively proportional to b". It indicates a positive monotonic relationship between a and b. $I+(a, b_i)$ is a *direct influence* and indicates that the derivative of a is equal to the sum of all b_i's actively influencing it.

vector of numerical values for the continuous parameters. Call this vector $\mathcal{N}_f$. We define a new notion of state by linking these two notions. First, for each non-ordering proposition class in the basis set, we add to $\mathcal{N}_f$ a boolean variable whose value is *true* or *false* according to whether the corresponding statement is true or false in a given state. For example, `Blocked(Pipe1)` becomes an explicit parameter in $\mathcal{N}_f$. (This extension is common, at least in training simulators.) Second, we define state as a pair $< \mathcal{N}_f, \mathcal{Q}_f >$, where $\mathcal{Q}_f$ ranges over the set of states in the envisionment. A state is *consistent* if and only if the values of $\mathcal{N}_f$ satisfy the propositions of the qualitative state corresponding to the value of $\mathcal{Q}_f$. Otherwise, it is *inconsistent*. Checking the consistency of a state is straightforward. For each non-ordinal proposition in $\mathcal{Q}_f$, check that its corresponding boolean parameter has the appropriate value. For each ordinal relationship in $\mathcal{Q}_f$, check if the same relationship holds between the corresponding numerical parameters.[3]

Determining the components of $\mathcal{N}_f$ for a system requires analyzing the envisionment in concert with the math-model library. A boolean parameter must be included for each class of non-ordering propositions in the basis set. A numeric parameter must be included for each continuous property in the QP model, as well as for each constant introduced by the corresponding numerical models (such as `Density` and `Radius` in the entry for `Level` above).

2.3 The architecture of self-explanatory simulators

A self-explanatory simulation consists of a tightly integrated set of qualitative and numerical representations. Such simulations are generated by self-explanatory simulators, which in turn are constructed by a *simulator compiler*. The compiler takes as input an envisionment of a specific system and a math model library, and produces a simulator, consisting of a set of procedures and datastructures which support prediction and explanation concerning the classes of behaviors described by the envisionment.

There are three crucial components in a self-explanatory simulator: (1) a set of *evolvers*, procedures which specify for each qualitative state how to update its numerical parameters over time; (2) a set of *state transition procedures* (STP's) which detect qualitative changes in state, and (3) an *explanation facility* which uses information from the envisionment to provide causal accounts and characterize possible behaviors. These components interact during simulation as follows. The value of $\mathcal{Q}_f$ is used to fetch the corresponding evolver. The evolver is executed to update $\mathcal{N}_f$. The STP corresponding to $\mathcal{Q}_f$ is fetched and executed to see if a transition has occurred. If it has, then $\mathcal{Q}_f$ is updated to this new state. The cycle repeats until no more simulation is required. The explanation facility can be used during or after simulation to better understand the results.

`SIMGEN` works by first computing the constituents of $\mathcal{N}_f$ and $\mathcal{Q}_f$. Next it writes the evolvers and state transition procedures. Finally, it caches information from the envisionment to support explanation. The rest of this section describes these components in more detail, and how `SIMGEN` builds them.

2.3.1 Evolvers

An evolver is a procedure which, given a state vector and Δt, produces a new state vector representing the evolution of the modeled system over Δt. Roughly, traditional simulations operate by identifying a small set of state variables, estimating derivatives for them, computing their new values, and then calculating new values for any relevant dependent variables. This organization can be easily translated into QP terms. In a QP model, the directly influenced parameters correspond to state variables, since direct influences comprise an integral connection [12]. The indirectly influenced parameters, that is, those linked by some chain of qualitative proportionalities to the directly influenced parameters, form the dependent variables. Any parameters not mentioned in the QP model are constants, and hence cannot change.

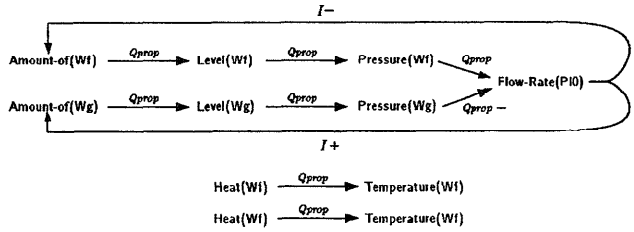
A key problem in writing simulation programs is establishing an order of computation for a given set of equations. In QP theory, a qualitative proportionality represents both a functional and a causal relationship. Since entries in the math-model library correspond to combinations of qualitative proportionalities, we can use the *causal ordering* [18] induced by the influences in the qualitative model to construct an order of computation for any consistent set of numeric equations. In particular, (1) estimate the derivatives of directly influenced parameters (e.g., the state variables), (2) update the amounts (`A`) of directly influenced parameters, (3) recompute amounts of indirectly influenced parameters, and (4) estimate derivatives of indirectly influenced parameters by subtracting old values from new.

The individual updates within steps 1, 2, and 4 can be performed in any order (our current system uses Euler integration for simplicity). The order of computation in step 3 can be determined by a simple search of the influence graph. Notice that QP theory requires the subgraph of qualitative proportionalities to be loop-free in any legal state; feedback is represented by explicit integral connections (e.g., direct influences) only. This has the effect of demanding that any loop contain at least one state variable, a common constraint in numeric simulators. For example, Figure 1 illustrates the graph of influences for a simple two-container liquid flow. Given the current values for the `Amount-Ofs` and `Heats`, the `Levels` are computed next, followed by the `Pressures` and then `Flow-Rate`, while the `Temperatures` can be computed in any order (in this situation). The math-model library must contain at least one model for each consistent combination of qualitative proportionalities.[4]

[3] The finite precision of floating point requires using a "fuzz" parameter to detect equality.

[4] If there is more than one model, currently one is selected

Figure 1: The influence graph for a two-container flow problem. The graph of influences that holds at any moment in a QP description indicates a causal ordering between the parameters it describes. The state variables are given by the direct influences.



The model is tested to ensure that all parameters it mentions have already been computed in the current situation, using the order constraints of the influence graph.

Conceptually, an evolver could be supplied for each qualitative state. However, it is more practical to divide states into equivalence classes, grouping together those governed by a common set of equations and writing only a single evolver for each group. In QP models two states can share an evolver when they have the same set of active processes and views, since the set of qualitative proportionalities for each state is identical, and hence the corresponding equations will be the same.

An important opportunity for self-monitoring occurs when setting up states. The strategy used for writing evolvers is also used to write *initialization routines*, which obtain values for independent variables from the user, calculate dependent parameters and estimate derivatives, and check the resulting state's consistency.

2.3.2 State Transition Procedures

Traditional mathematical formalisms do not provide a comprehensive, formal language for describing when an equation holds. By using QP theory as the basis for a modeling language, such conditions can be stated formally and used in reasoning. Given a particular qualitative state, we can ascertain what conditions must be monitored to detect when a transition occurs, and write STP's that sense such transitions and determine the new qualitative state.

The parameters that must be monitored for each state are determined by analyzing the envisionment's transitions. Recall that *limit hypotheses* indicate possible changes in ordinal relationships [12]. All of the changes possible in the current situation must be monitored. Usually the result of a limit hypothesis is unique, but not always – underconstrained properties of objects coming into existence or actions which cause discontinuous changes [15] can result in multiple next states. In such cases the STP must also perform enough extra tests to discriminate between the possibilities. Importantly, these tests are all inexpensive numerical inequality tests, and typically only a handful are needed for each state, so the overhead of transition finding is quite small.

Handling transitions where a numerical relationship changes to equality requires special care, since the equality may only hold for an instant and hence could be missed. We call this the *numerical transition skip* (NTS) problem. We detect when NTS has occurred by noting when one of the monitored relationships undergoes a discontinuous change (e.g., when $N_1 < N_2$ holds at one tick but $N_1 > N_2$ holds at the next). This causes the runtime system to "roll back" the simulation, performing binary search to find a value for Δt that hits the transition. Once the numerical values at the transition are computed, the simulation proceeds with the original Δt.

STP's also share the burden of self-monitoring. Assuming that the initial state vector is consistent, inconsistencies can only arise when the evolution $\mathcal{N}_f$ "drifts away" from the subspace consistent with $\mathcal{Q}_f$. The procedure outlined above already catches cases where the result of a transition is not a state the envisionment predicted. However, inconsistencies involving unmonitored relationships are not detected by default. What level of error checking is reasonable depends on circumstances. For example, if the domain models are well-tested, only the minimal testing described so far may be needed. When more stringent self-monitoring is required, such as developing a new domain model, the numerical component of state can be re-classified as often as desired. Reclassifying at every clock tick, for instance, ensures that any misalignments between qualitative and quantitative models is caught as early as possible. (Such tests can be made reasonably efficient by using a discrimination tree to peform the classification, but since the overhead is still substantially higher, these extra tests are not performed by default in our implementation.)

2.3.3 Supporting Explanation Generation

Using qualitative models to ground and generate simulation procedures supports a variety of explanation generation tasks. The qualitative model provides a causal account for all changes in every state. The inclusion of $\mathcal{Q}_f$ in the state vector provides access to the appropriate account for any (simulated) time. Furthermore, this causal explanation is not simply a post-hoc reconstruction – given the organization of evolvers above, it is literally the way the simulation of the system is evolved! Similarly, information about classes of possible futures is available through the transitions of the envisionment. At any simulated time, one can find out what events might happen, or could have happened instead with other choices for $\mathcal{N}_f$.

We make two stipulations concerning the run-time system. First, we require that it includes access to information from the envisionment. How much information is needed and how it is accessed varies according to task requirements. If the run-time system cannot be

at random. Clearly, this is an opportunity for a reasoned choice, based on criteria such as desired accuracy and performance requirements.

compute-intensive, the simulation compiler might identify in advance what information is needed about each envisionment state and create a database to serve as a cache for the run-time system. Alternately, relevant portions of an envisionment or history could be generated incrementally, on demand. (Currently we simply include pointers to the envisionment in the simulator itself.)

The second stipulation is that the run-time system must maintain a *registration* [13], which describes the history of the system in terms of occurrences of states of the envisionment. Each episode in this history includes the corresponding state of the envisionment, a numerical value for its temporal beginning, and if it ends, a numerical value for its end along with the limit hypothesis which occurred. (It is straightforward to compute STP's that provide this information.) This history provides the temporal framework required to relate simulated time to a path of qualitative states, and hence provide access to the appropriate qualitative knowledge.

3 Examples

Here we show some interactions with simulators produced by SIMGEN, our compiler of self-explanatory simulations. The QP domain models used are similar to those in [12], but are slightly enhanced to better model the interaction of heat and mass flows [6]. The questions were posed using a formal query language, and the English output was generated automatically by the default explanation facility.

Simple Liquid Flow: The two-container example in Figure 2 shows that the simulation can provide a variety of information about parameters and possible behaviors.

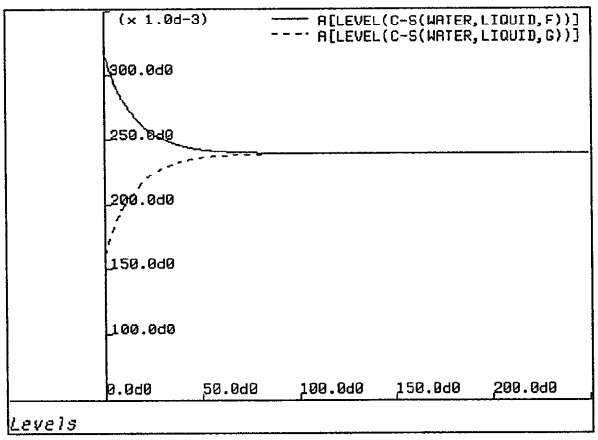
Boiling: In Figure 3, the simulator detects that boiling has begun, and changes evolvers appropriately. Furthermore, it enforces the semantics of existence, by refusing to provide information about properties of objects at times when they don't exist.

Spring/Block Oscillator: As Figure 4 illustrates, SIMGEN is not limited to thermodynamic systems. Here a naive user attempts to provide a negative spring constant (a "perturbing force", rather than a restoring force). This is inconsistent with the user's presumed starting state, and after examining the equations and the assumptions, the user makes a more reasonable choice.

4 Analysis

What are the limits of SIMGEN? Given a QP domain model, a math model library that provides appropriate expressions for each instance of indirectly influenced quantities (e.g., one that follows the causal ordering represented by the graph of influences), SIMGEN can compile a self-explanatory simulator for any system that can be successfully envisioned with that domain model. Issues of scale and numerical stability are important, of course, as noted below. But more fundamentally, when can we have the appropriate domain models?

Figure 2: A simple example of liquid flow. Questions about specific points in time establish the temporal context for subsequent questions.

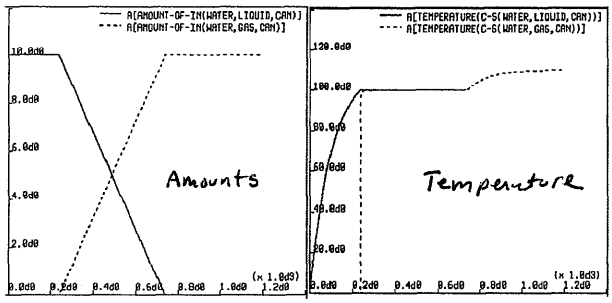



```
Q: What is happening at t = 50.0 seconds?
A: A flow of water from F to G.

Q: What is A[LEVEL(C-S(WATER,LIQUID,F))]?
A:The level of the water in F is 0.24224941.

Q: How is LEVEL(C-S(WATER,LIQUID,F)) changing?
A: The level of the water in F is decreasing.

Q: What affects LEVEL(C-S(WATER,LIQUID,F))?
A:The level of the water in F is changing as a
  function of the amount of the water in F.

Q: What happens next?
A: At 130.5 seconds the pressure of the water
   in F and the pressure of the water in G
   become equal. Then, nothing is happening.

Q: What else might have happened?
A: The transition at 130.5 seconds was inevitable.

Q: What is happening at t = 200.0 seconds?
A: Nothing is happening.

Q: What happens next?
A: This behavior continues forever.
```

Just as no single traditional engineering technique for organizing knowledge has proven adequate for all domains, we should not expect any single ontology to suffice for qualitative physics. [5] The process-centered view works best for domains like thermodynamics [27] and chemistry, where explicit reasoning about processes is crucial. The device-centered view [7; 29] works best when the idealizations of network theory hold. For example, analyzing a voltage divider can give rise to simultaneous equations, which cannot be solved via propagation alone. SIMGEN could not handle this system. It

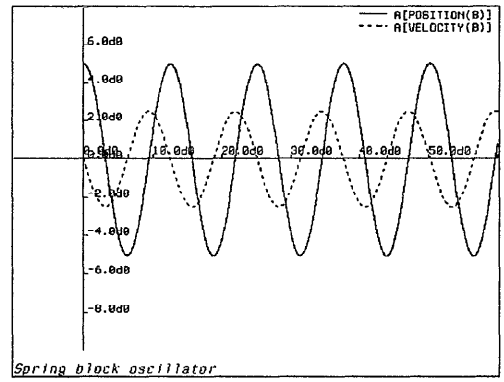
[5] [8] makes a similar point comparing engineering to economics.

Figure 3: Changes in regime and existence are automatically detected.




```
Q: Summarize the behavior
A: The initial state is S3. In S3 there is heat is
   flowing from STOVE to the water in CAN.

   The boiling temperature of WATER and the
   temperature of the water in CAN become equal,
   which leads to S1.  In S1, heat is flowing from
   STOVE to the water in CAN and the water in CAN
   is boiling.

   The amount of steam in CAN becomes positive,
   which leads to S16.  In S16, heat is flowing
   from STOVE to the water in CAN, heat is flowing
   from STOVE to the steam in CAN, and the water
   in CAN is boiling.

   The amount of water in CAN becomes zero,
   which leads to S4. In S4, heat is flowing
   from STOVE to the steam in CAN.

   The temperature of the steam in CAN and the
   temperature of STOVE become equal, which leads
   to S5.  In S5, nothing is happening.

Q: What is happening at t = 100.0 seconds?

A: Heat is flowing from STOVE to the water in CAN.

Q: What is the temperature of the steam?

A: Sorry, the temperature of the steam in CAN
   does not exist at 100.0 seconds.

Q: What happens next?

A: At 238.6 seconds the boiling temperature of WATER
   and the temperature of the water in CAN become
   equal.  Then, heat is flowing from STOVE to the
   water in CAN and the water in CAN is boiling.

Q: What else might have happened?

A: The only other possibility was: The temperature
   of the water in CAN and the temperature of STOVE
   become equal, which leads to S2.  In S2, nothing
   is happening.
```

Figure 4: Clashes between qualitative and quantitative models are detected.




```
Q: Set up a run starting from S5 for 120.0 seconds,
   with dt = 0.001, using these parameters:
      VELOCITY(B) = 0.0
      REST-LENGTH(SPR) = 5.0
      POSITION(B) = 5.0
      MASS(B) = 1.0
      K(SPR) = -0.1
A: The values provided do not match S5.
   The differences are:
   (A (FORCE B)) is greater than zero,
   should be less than. No legal state.

Q: Show the equations for S5.
A: (= (ACC PIO) (/ (FORCE B) (MASS B)))
   (= (FORCE B) (FORCE SPR))
   (= (LENGTH SPR) (+ (POSITION B)
                      (REST-LENGTH SPR)))
   (= (FORCE SPR) (* (- (K SPR))
                     (- (LENGTH SPR)
                        (REST-LENGTH SPR))))
   (= (D (VELOCITY B)) (ACC PIO))

Q: Change K to be 0.1 and simulate.
A: Okay.
```

could handle the analogous fluid system, however. Modeling the nodes of a system as containers, and charge as something that flows between them, provides the basis for a QP-style causal account. Conversely, when the "stuff" flowing in a system has behaviors that cannot be captured in System Dynamics (such as phase changes and chemical properties), the device ontology is unsuitable. **SIMGEN** provides the most leverage where QP theory works best. For domains where time-varying differential equations are only part of the relevant mathematics, such as aerodynamic analyses, stability of structures, and other heavily geometric analyses, **SIMGEN** obviously provides little leverage.

So far we have discovered two kinds of difficulties in building math model libraries. First, we found ourselves using ever more sophisticated qualitative models in order to provide enough functional dependencies to yield reasonable numerical models. Second, many engineer-

ing formulations have evolved under the constraint of simplifying algebraic analyses, rather than supporting causal reasoning. For instance, heat is often not used as an explicit variable in today's formal thermodynamic analyses (those which refer directly to temperature), although textbooks often revert to employing heat in its commonsense usage when discussing difficult points. In any case, we believe the discipline imposed by supporting self-explanatory simulations should be viewed as an invigorating challenge, which will ensure that one's domain models will be both powerful and accurate.

Importantly, while an envisionment is needed to generate self-explanatory simulators, no new qualitative reasoning needs to occur during simulation. The qualitative knowledge is compiled into a set of procedures expressing its implications for the particular system. Given good optimization techniques, it seems self-explanatory simulators could become asymptotically close in speed to the best hand-written numerical simulators, despite their increased transparency and robustness.

5 Related Work

Several recent projects have focused on the relationship between qualitative and quantitative knowledge [23; 30; 27; 32]. None of these efforts focus on automatically constructing numerical simulators or explanation generation. The closest in spirit is [3], which also argues for a unification of qualitative and numerical simulation. We differ in most specifics, however: Berleant augments a QSIM representation with interval values for parameters to restrict behavior generation, while we co-evolve qualitative and numerical states. We generate simulations automatically, whereas QSIM models are hand-crafted, and we also focus on generating causal explanations, while Berleant focuses on constraining a-causal predictions.

Sussman's *Dynamicist's Workbench* project, which uses AI techniques to develop efficient numerical simulations from equational models, shares several of our concerns, including generating efficient code and producing understandable results. Their work complements ours in several ways. They have focused on sophisticated reasoning about numerical techniques [1] maximizing efficient computation [4], including compiling to special-purpose hardware [2]. But while the behaviors of the systems they are analyzing are subtle, they start with a single set of equations which governs the system for all time. By contrast, we have downplayed reasoning about numerical methods in favor of understanding how to automatically generate a system's equations from a physical model, including situations where the relevant set of equations changes over time, and on producing understandable explanations. We believe our work will benefit from their advances in numerical reasoning, while theirs will benefit from our use of qualitative reasoning to guide simulation construction and improved techniques for detecting clashes between qualitative and quantitative models.

Our compilation of qualitative knowledge into simulation procedures finds echos in [21], which describes the compilation of diagnosis and redesign rules from a general-purpose knowledge base of device models. In fact, the KSL group has proposed a *simulation foundry* which could create simulations from a knowledge base of physical models and structural equations [19]. SIMGEN can be viewed in part as an instantiation of this idea, although they did not anticipate our notion of self-explanatory simulations.

6 Discussion

We introduced a new kind of simulation, self-explanatory simulations, which blend qualitative and quantitative knowledge to provide several of the advantages of each. By using qualitative analysis to represent when different sets of equations are appropriate, we gain increased automation. By incorporating knowledge of what behaviors are reasonable into simulation code and co-evolving numerical and qualitative states we achieve improved self-monitoring. And by incorporating "compiled" knowledge from an envisionment, we are able to produce understandable explanations. Importantly, complex, first-principles reasoning can occur off-line – self-explanatory simulations can run at speeds which asymptotically approach standard numerical simulations.

While this method of integrating qualitative and quantitative knowledge is by no means the only one, we believe it is particularly important. For design, it is important to ensure that reasonable parameter values can result in the desired behaviors. For training, the value of numerical simulators is already well-established, but the incorporation of inexpensive explanation facilities can make them even more valuable. In developing models, either qualitative or numerical, of a new phenomena, checking the match between the model and your intuitions is an important task. By formulating intuitions explicitly in the form of qualitative models, self-explanatory simulations can help detect whether or not the behaviors predicted by a numerical model make sense.

6.1 Future Work

This research suggests several new possibilities:

Scaling up: Envisioning is not the only qualitative simulation technique which could support self-explanatory simulations. For example, given an incremental envisioner and simulation compiler, states and simulation procedures could be generated on the fly, just one step ahead of the current qualitative state. Or, if the structure of the system is extremely large and the run-time system must be kept simple, the simulator could be decomposed into subsystems. $\mathcal{Q}_f$ would be a vector of $\mathcal{Q}_f$'s for the subsystems, and the evolvers for the subsystems would be executed in concert to provide the effect of an evolver for the whole system. These and other alternatives need exploration.

Self-explanatory simulations for other kinds of qualitative physics: **SIMGEN** might be adapted to the device ontology. **SIMGEN** relies critically on two features of QP theory: the causal account provides the order of computation for parameters, and the use of quantified descriptions allows explicit modeling assumptions. The causal account in device ontologies is based on an identified input perturbation or signal, which is consistent with the use of specified inputs to drive numerical device-centered simulators such as **SPICE** [20]. It also seems possible to adapt many of the representational techniques of QP theory to give device-centered models the same ability to explicitly encode modeling assumptions. However, we leave such extensions for advocates of this ontology.

Tutor Compilers: Self-explanatory simulators could become a core component in a variety of computer-based tutors. By using **SIMGEN** with action-augmented envisionments [15], it may be possible to automatically construct a class of training simulators (such as **STEAMER**) automatically. By caching envisionment information and "cross-compiling", self-explanatory simulators could be built for delivery on inexpensive target hardware. Actually building a tutor compiler will require a great deal of work, including developing powerful domain models, developing better explanation generation systems, and support software such as graphical systems. However, we are very excited by the possibility of semi-automating the production of computer-based tutors.

7 Acknowledgements

This work was supported by the National Aeronautics and Space Administration Contract No. NAG 9408, and an equipment grant from IBM. We thank Gerald Sussman and Brian Williams for useful comments on this work and its presentation.

References

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

[2] Abelson, H. et. al. Intelligence in Scientific Computing *Communications of the ACM* 32:546-562, 1989.

[3] Berleant, D. A unification of numerical and qualitative model simulation. *Proceedings of the 1989 Workshop on Model-Based Reasoning at IJCAI-89.*

[4] Berlin, A. A Compilation strategy for numerical programs based on partial evaluation MIT AI Lab Technical Report No. 1144, July, 1989.

[5] Christensen, P, Kofoed, J. and Larsen, N. PC-DYSIM - A program package for simulation of continuous dynamic processes: User's manual Technical Report Riso-I-342, Riso National Laboratory, DK-4000 Roskilde, Denmark, February, 1988

[6] Collins, J and Forbus, K. Building qualitative models of thermodynamic processes. Technical report in progress.

[7] deKleer, J. and Brown, J. S. A qualitative physics based on confluences. *Artificial Intelligence*, 24:7–83, 1984.

[8] deKleer, J. and Brown, J. S. Theories of causal ordering *Artificial Intelligence*, 29:33-62 1986.

[9] Falkenhainer, B and Forbus, K. D. Setting up large-scale qualitative models. In *Proceedings of AAAI-88.*

[10] Falkenhainer, B. and Forbus, K. D. Compositional Modeling: Finding the right model for the job *Submitted for publication*, February, 1990.

[11] Forbus, K. An interactive laboratory for teaching control system concepts BBN Tech Report No. 5511, 1984.

[12] Forbus, K. D. Qualitative process theory. *Artificial Intelligence*, 24, 1984.

[13] Forbus, K. The logic of occurrence. *Proceedings of IJCAI-87.*

[14] Forbus, K. D. The qualitative process engine in *Readings in Qualitative Reasoning about Physical Systems*, Weld, D. and de Kleer, J. (Eds.), 1989.

[15] Forbus, K. D. Introducing actions into qualitative simulation. *Proceedings of IJCAI-89.*

[16] Forbus, K. and Stevens, A. Using qualitative simulation to generate explanations. *Proceedings of the Third Annual Conference of the Cognitive Science Society*, 1981.

[17] Hollan, J, Hutchins, E, and Weitzman, L. STEAMER: An interactive inspectable simulation-based training system. *AI Magazine*, Summer 1984.

[18] Iwasaki, I. and Simon, H. "Causality in device behavior", *Artificial Intelligence*, **29**, 1986.

[19] Iwasaki, Y., Doshi, K.,Gruber, T.,Keller, R., and Low, C. M. Equation model generation: Where do equations come from? *Proceedings of the 1989 Workshop on Model-Based Reasoning at IJCAI-89.*

[20] Katzenelson, J. AEDNET: A simulator for nonlinear networks *Proceedings of the IEEE*, Vol 54, No. 11, November, 1966.

[21] Keller, R., Baudin, C.,Iwasaki, Y., Nayak, P. and Tanaka, K. Compiling special-purpose rules for general-purpose device models Stanford Knowledge Systems Laboratory Report No. KSL 89-50, June, 1989.

[22] Kuipers, B. Qualitative simulation. *Artificial Intelligence*, 29:289–338, 1986.

[23] Kuipers, B. and Berleant, D. Using incomplete quantitative knowledge in qualitative reasoning Proceedings of AAAI-88, St. Paul, Minnesota, 1988.

[24] Larsen, N. Simulation model of a PWR power plant Technical Report Riso-M-2640, Riso National Laboratory, DK-4000 Roskilde, Denmark, March, 1987

[25] Roberts, B. and Forbus, K. The STEAMER mathematical simulation. BBN Tech Report No. 4625, 1981.

[26] Shearer, J., Murphy, A. and Richardson, H. *Introduction to System Dynamics*, Addison-Wesley, 1971.

[27] Skorstad, G. G and Forbus, K. D. Qualitative and quantitative reasoning about thermodynamics. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, Ann Arbor, MI, August 1989.

[28] Stevens, A., Roberts, B., Stead, L. Forbus, K., Steinberg, C., Smith, B. "STEAMER: Advanced computer-aided instruction in propulsion engineering", BBN Technical report, July, 1981

[29] Williams, B. Qualitative analysis of MOS circuits. *Artificial Intelligence*, 24:281–346, 1984.

[30] Williams, B. MINIMA. A symbolic approach to qualitative algebraic reasoning. *Proceedings of AAAI-88.*

[31] Woolf, B., Blegen, D., Jansen, J., and Verloop, A. Teaching a complex industrial process *AAAI-86.*

[32] Yip, K. Generating global behaviors using deep knowledge of local dynamics *Proceedings of AAAI-88.*