FROM: Proceedings of the 1990International Conference on the Learning Sciences; NWU, Evanston, IL; 8/91

Towards Tutor Compilers: Self-Explanatory Simulations as an Enabling Technology

Kenneth D. Forbus
The Institute for the Learning Sciences, Northwestern University
1890 Maple Avenue, Evanston, Illinois 60201

Abstract: This paper discusses *self-explanatory simulations*, an integration of qualitative and quantitative techniques, which we believe will provide the foundation for a new generation of simulation-based tutoring systems. We describe the basic technology and decompose potential instructional uses along three dimensions, *type of explanations, artifact size*, and *processing locus*, indicating what progress seems needed for particular classes of applications.

1. Introductions

Simulation-based training provides a powerful technique for ITS. Examples include SOPHIE (Brown, Burton, and deKleer 1982), STEAMER (Hollan, Hutchins, and Weitzman 1984; Stevens et al. 1981) and RBT (Woolf et al. 1986). Their power stems from the ability of numerical simulation to provide a reasonably accurate depiction of artifact behavior and the use of direct-manipulation interfaces to make the simulation internals inspectable to some degree. But building simulation-based trainers can be very difficult, in part because building numerical simulations themselves is very difficult. In both STEAMER and RBT, pre-existing simulations were used as a starting point. Creating such simulations typically takes many person-years of effort. So while there are many potential applications for simulation-based training, the lack of usable simulators is a serious roadblock.

Even when simulators exist, they are often hard to use or unsuitable for training purposes. Just hooking up an existing simulator can be a herculean chore, requiring equal measures of archeology, operating-systems expertise, and luck. Most numerical simulators suffer the traditional problems of custom software: a lack of documentation and the existence of many implicit modeling assumptions. Intelligent tutoring and training systems should provide explanations, but grafting an explanation system on top of an existing numerical simulator is a difficult, and often impossible, task. In STEAMER, for instance, no qualitative explanations were provided in the main system and only a few modules used qualitative techniques. In part this was due to the primitive state of qualitative physics in the early 80's. But it was also due to the sheer difficulty of "spelunking" through the simulator to figure out just what parts of the steam plant were modeled, and to what degree of fidelity.

This paper explores *self-explanatory simulations* (Forbus and Falkenhainer 1990) as a way around this roadblock. Section 2 describes the basics of self-explanatory simulations. Section 3 examines this technology from the standpoint of building ITS's. We decompose potential systems along three dimensions: their explanation capabilities, the size of artifact involved, and the degree to which on-line processing is required. This decomposition is used to examine what may be practical in the near term versus what requires substantial additional basic research. Finally, Section 4 outlines some further issues and our plans to build a first-pass Tutor Compiler.

2. Self-Explanatory Simulations

The basic idea of self-explanatory simulations is to use the qualitative analysis of a system as a framework to organize a numerical simulation of that system. Consider the traditional *state space* formulation of dynamical systems. In this formulation a system has a set of *state variables* which suffice to completely determine its properties. The state variables for a spring-block oscillator, for instance, might be the position of the block and its velocity. These state variables may in turn determine other properties of the system (e.g., kinetic energy), but all other parameters are functions of these state variables. In Qualitative Process theory (Forbus 1984a), such state variables correspond to *directly influenced* quantities. In QP theory the directly influenced quantities cause in turn changes in other quantities, just as other parameters are determined by state variables in the state-space formulation. In a state-space the state of the system is described by a vector of values for the state parameters; while in qualitative terms it is described by a set of inequalities over the system's variables. Effectively, the qualitative description quantizes state space into regions of "equivalent" (i.e., qualitatively identical) behavior.

A simple system like a spring-block oscillator can be modelled by a single set of equations. But many systems are not so simple. For example, the set of equations relevant for modeling a can of water being heated on a stove is different from the set of equations that model what happens when that water is boiling. Traditional techniques focus on analyzing a given set of equations, and tend to ignore the modeling process itself. Qualitative physics provides tools for formalizing the modeling process, enabling the implicit assumptions of the modeler to be expressed and reasoned about. For example, a spring might behave linearly for small excursions but non-linearly for large displacements. Such changes in the relevant equations are then reflected in the quantization of state-space introduced by the qualitative description.

Already we see one role that qualitative analysis performs in simulation generation: constructing the relevant sets of equations. But there is more to it than that. First, suppose our simulation includes a notion of qualitative state as well as numerical state. This provides the potential for improved *self-monitoring*, i.e., detecting clashes between expectations expressed qualitatively and numerical predictions. Suppose, for instance, that one has modeled a spring-block oscillator in the usual way with the spring providing a restoring force. A naive user might try a spring constant with the wrong sign (corresponding to a "exacerbating force"), and the simulation will detect the inappropriate behavior which results and flag the relevant parameters for correction. Integrating the qualitative state with the numerical state also provides the basis for powerful explanation facilities, since the qualitative state can be used to access the set of equations describing the current behavior, the artifact's causal structure, and the qualitatively distinct behaviors which may occur next. These capabilities are illustrated in Forbus and Falkenhainer (1990).

This self-explanatory simulation concerns a large can containing a small amount of water sitting on a stove. In the dialogue below, the user has run the simulation and is viewing plots of the amounts of water and steam and their temperatures. Questions are posed in a formal query language, and paraphrased here in English. The answers were generated automatically by the simulator's explanation facility. Commentary is in italics.

Q: What is happening at t = 100.0 seconds?
A: Heat is flowing from STOVE to the water in CAN.

Qualitative explanations are available for every time tick in the simulation.

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

The simulator knows that the numerical value it has for this tick is meaningless and does not report it.

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.

Here we see the qualitative and quantitative components interacting: The qualitative aspect identifies this point as interesting, the numerical aspect ascertains its exact time.

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.

The qualitative aspect of the simulation provides alternate possible behaviors.

Figure 1. An example of self-explanatory simulations

The causal structure imposed by the qualitative analysis is more than just a device for qualitative simulation and explanation. It plays a central role in building the numerical aspect of the simulator itself. For example, the order in which parameters are updated can be read directly off the graph of influences in the QP analysis of the system. Also, tests are incorporated to determine when the qualitative state changes, so that different code is used to reflect the different set of governing equations.

To summarize, a self-explanatory simulator combines qualitative and numerical representations to generate predictions and explanations of an artifact's behavior. The numerical component provides precise predictions of behavior, while the qualitative component explains this behavior in terms of the underlying qualitative model and set(s) of equations. (In this respect we differ from the IMTS approach (Towne and Munro 1988), which focuses on crafting simulations with just enough fidelity for providing on-screen animations and training students in troubleshooting.)

Self-explanatory simulators can be built automatically, assuming the following inputs: (1) a *domain theory*, expressed in QP theory, which describes the fundamental physical phenomena of interest in qualitative terms; (2) a *Math Model Library*, which provides fragments of equations for each set of qualitative proportionalities which may constrain a type of quantity in the domain theory; and (3) a *structural description* of the artifact or system to be modeled, including a set of modeling assumptions outlining the kinds of factors to be considered. Given (1) and (3), an *envisionment* is generated for the artifact, representing its quantized state space under the particular modeling assumptions. This envisionment is then analyzed in concert with the Math Model Library to produce a self-explanatory simulation.

3. A Design Space for Simulation-Based Tutors

There are many ways to build and use simulation-based tutors. To clarify the issues involved, we distinguish three dimensions which characterize potential applications. The location of the state of the art along these axes indicates what kinds of systems are (and are not) technologically feasible. We consider each dimension in turn, and then examine some specific potential applications.

Types of Explanations

Instruction can involve a range of explanations, from simple statements about a system's structure to causal explanations to analogies explaining a new phenomenon in terms of the familiar. While there are a host of issues concerning what should be explained (and how and when) which must be tackled by any ITS designer, our focus here is just on what kinds of information are in principle available.

Self-explanatory simulations provide several useful kinds of explanations directly. For any given time, one can ascertain (a) the equations governing the system, (b) the causal story linking the various system parameters, in terms of processes and influences, and (c) the numerical values of the system's continuous properties. Predictions can either be made via additional simulation, or via qualitative analysis. For example, if the state transitions of the envisionment used to generate the simulation are cached as part of the explanation system, alternate behaviors can be easily described (c.f. Figure 1). However, they provide no direct explanation of the teleology of the system or how the system might behave under different modeling assumptions than the simulation was generated for. Nor do they explain the fundamental principles of the domain itself in more detail, nor provide any sort of explanation that is not specific to the particular system the simulator was compiled for. To generate such explanations will require other facilities, built on top of the representations supplied by self-explanatory simulators.

Roughly, additional explanation facilities can be divided into two categories: *canned* and *generative*. Examples of canned explanation facilities include chunks of text describing the system's components, or hypermedia networks. Using hypermedia with self-explanatory simulations is a particularly interesting prospect. Self-explanatory simulations could become a new kind of node in the network, with the terms used in the explanation facility cross-indexed into the rest of the network.

Generative explanation facilities include for example the ability to recognize specific categories of behavior in functional terms. Determining whether or not an oscillation is damped, for instance, requires analyzing the behavior as it unfolds. Another useful generative capacity would be the ability to evaluate a student's analogies, suggesting corrections to avoid misconceptions and to lead to a deeper understanding. Obviously such capabilities would be very useful pedagogically, but currently these are matters of basic research.

Artifact Size

The operation of self-explanatory simulators can be made extremely fast. There is no reason to believe that they cannot asymptotically approach the speed of a traditional mathematical simulator, with clever design. Building them is another story. The current SIMGEN compiler relies on total envisionments as its starting point for generating behaviors. The size of an envisionment tends to be exponential in the size of the system modeled, which means that the current system is suitable only for small-scale examples, and could be applied to medium-scale systems only with substantial computing resources. We are currently exploring several techniques to solve this problem.

Processing Locus

Performing novel reasoning from first principles will always be slower than skill-based behavior. As Anderson (1986) demonstrated, a useful strategy for delivering instruction on affordable machines is to use substantial off-line resources to produce a skilled system that performs in a pre-selected subset of the domain. Self-explanatory simulators can be viewed as generalizing this technique to simulation-based systems. For some instructional tasks off-line reasoning and on-line simulation will suffice. However, there are other tasks which require on-line reasoning. For instance, evaluating a student's design requires the ability to rapidly analyze a new system.

What Can We Do When?

Now that we have some dimensions to characterize simulation-based trainers, we can examine some potential instructional applications for self-explanatory simulations and better understand what is needed to achieve them.

Active illustrations. Simulations which involve a handful of parts that are used to illustrate fundamental physical principles are perhaps the simplest instructional application. Explanations would be in terms of the QP domain theory, the artifact size is small, and the processing locus is totally off-line. Building such systems appears at this point to mainly require some software engineering work, to make the current version of SIMGEN more bulletproof and to make it produce better stand-alone runtime systems.

Subsystem trainers include systems with controls and possible faults, with roughly the complexity of an engineering watch station on a Navy ship or an aircraft subsystem (e.g., an engine or hydraulic system). Building such systems will require extending self-explanatory simulations to use *action-augmented envisionments* (Forbus 1989). This extension seems straightforward, although the inclusion of fault models may require substantially increased compilation time.

Training simulators, such as STEAMER and RBT, involve a substantial (i.e., orders of magnitude) increase in complexity over subsystems trainers. Compiling self-explanatory systems on this scale will require a breakthrough.

Simulated laboratories and construction sets. Learning by building, where students create novel devices by combining stock parts, is pedagogically attractive (c.f. Forbus 1984b). However, they can require substantial on-line processing, since the structure of the artifact cannot be fixed in advance. Again, this will require a breakthrough.

Critics and coaches. Using trainers and construction sets often requires coaching, which ideally should be provided by the program itself. This requires expanding the kinds of knowledge beyond the physical principles encoded in a standard QP domain theory. For instance, analyzing modeling assumptions (Falkenhainer and Forbus 1991) is essential in understanding why perpetual motion machines don't work. Recognizing that a student's refrigerator design doesn't quite work correctly requires teleological reasoning. These issues require more basic research.

4. Discussion

Building ITS's is currently a labor-intensive enterprise, with thousands of hours invested for each hour of instruction delivered. Automation seems crucial. Ultimately, we want a full-fledged *Tutor Compiler*—a system which could produce ITS' given the description of the artifact, its intended users, and target hardware (Forbus 1988). We believe self-explanatory simulators are an important step towards that goal. Currently at ILS we are designing the Mark One Tutor Compiler, which will produce active illustrations and subsystem trainers which can run on small (IBM PS/2) computers. We hope to have this system operational sometime during fall of 1991.

One factor not mentioned in the design space for simulation-based tutors, but which is perhaps the most critical, is the quality and scope of the domain theory. Anderson (1988) points out that developing the domain theory is often about half of the total ITS-building effort. As we learn how to automate ITS construction more, we can only expect this fraction to grow. It will be a happy day when the biggest problem facing the builders of intelligent tutoring systems is teaching a tutor compiler what it should be teaching.

Acknowledgements: The work on self-explanatory simulations is carried out in collaboration with Brian Falkenhainer. The Mark One Tutor Compiler is being built in collaboration with Gregg Collins. Financial support has been provided by NASA Langley Research Center, Xerox, and IBM. The Institute for the Learning Sciences was established in 1989 with the support of Anderson Consulting, part of the Arthur Anderson Worldwide Organization. The Institute receives additional funding from Ameritech (an Institute Partner) and IBM.

References

Anderson, J.R. and Skwarecki, E. (1986). The automated tutoring of introductory programming. *CACM* 29(9):842-849.
Anderson, J.R. (1988). The expert module. In M.C. Polson and J.J. Richardson, *Foundations of intelligent tutoring systems*. New Jersey: Lawrence Erlbaum Associates.
Brown, J.S., Burton, R.R., and de Kleer, J. (1982). Pedagogical, natural language, and knowledge engineering techniques in SOPHIE I, II, and III. In D.H. Sleeman and J.S. Brown (Eds.), *Intelligent tutoring systems*. London: Academic Press.

Falkenhainer, B. and Forbus, K. (1991, Fall). Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, to appear.
Forbus, K. (1984a). Qualitative process theory. *Artificial Intelligence* 24.
Forbus, K. (1984b). An interactive laboratory for teaching control system concepts. BBN Tech Report No. 5511.
Forbus, K. (1988, Fall). Intelligent Computer-aided Engineering. *AI Magazine*.
Forbus, K.D. (1989). Introducing actions into qualitative simulation. *Proceedings of IJCAI-89*.
Forbus, K. and Falkenhainer, B. (1990). Self-Explanatory Simulations: An integration of qualitative and quantitative knowledge. *Proceedings of AAAI-90* Boston, MA.
Hollan, J., Hutchins, E., and Weitzman, L. (1984, Summer). STEAMER: An interactive inspectable simulation-based training system. *AI Magazine*.
Roberts, B. and Forbus, K. (1981). The STEAMER mathematical simulation. BBN Tech Report No. 4625.
Stevens, A., Roberts, B., Stead, L. Forbus, K., Steinberg, C., Smith, B. (1981). STEAMER: Advanced computer-aided instruction in propulsion engineering. BBN Tech report.
Towne, D.M. and Munro, A. (1988). The intelligent maintenance training system. In J. Psotka, D.L. Massey, and S.A. Mutter, *Intelligent tutoring systems lessons learned*. New Jersey: Lawrence Erlbaum Associates.
Woolf, B., Blegen, D., Jansen, J., and Verloop, A. (1986). Teaching a complex industrial process. *AAAI-86*.