Task-Driven Model Abstraction

Daniel S. Weld
Department of Computer Science and Engineering, FR-35
University of Washington
Seattle, WA 98195
Phone: (206) 543-9196
Internet: WELD@CS.WASHINGTON.EDU

Sanjaya Addanki
IBM T.J. Watson Research
Yorktown Heights, NY 10598

Date: February 21, 1990

ABSTRACT

Since algorithms for tasks such as design, diagnosis, and analysis become intractable when manipulating large models, we explore techniques for automatically abstracting a concrete model using the current task for guidance. First we define the *downward-solution* and *downward-failure properties* on abstractions of artifact models. Then we use these tools to analyze the abstractions used by some existing programs for design and analysis. Finally, we describe how these properties can be used to direct the selection of useful abstractions for the task of analysis.

1 Introduction

It is commonly recognized that a central problem in automated reasoning about physical systems is that the complexity of reasoning increases drastically (often exponentially) with the size of the model of the system in question. We believe that computers can be programmed to solve the complexity problem in the same way that human engineers cope with complexity — by introducing simplifying assumptions.

This proposal is no longer surprising. For example, explanation based learning can be seen as a technique for creating a specialized domain model that efficiently supports a restricted set of queries [Van Harmelen and Bundy, 1988], and the benefits of this approach have been clearly demonstrated [Minton, 1988]. Recently a number of researchers in qualitative physics have described systems that reason with abstract models to great advantage [Kuipers, 1987, Falkenhainer and Forbus, 1988, Addanki *et al.*, 1989, Weld, 1989]. However, most of the modeling work in qualitative physics assumes that a hierarchy of models is given to the program as input.

In this paper we describe first steps towards eliminating the need for prespecified abstract models. We catalog a set of abstraction operators which a reasoning program can use to customize simpler models for the task at hand. Then we analyze the behavior of abstractions in terms of the DOWNWARD-SOLUTION and DOWNWARD-FAILURE PROPERTIES. Next we illustrate how existing programs for hierarchical design and diagnosis can be viewed in this framework. And finally we explain how these properties can be used to direct the dynamic selection of useful abstractions for the task of analysis.

The important attributes of our approach are as follows:

- **Task Driven.** Our approach uses the current reasoning task and the particular problem at hand to drive the process of model creation.
- **Qualitative / Quantitative.** Our techniques are independent of representation.
- **Abstraction Guarantees.** Our theory provides a reasoner with a promising about the utility of its abstraction by specifying when an abstract solution can be specialized to the concrete level.

Section 2 makes clear the relationship between a task and a model, then presents a set of model abstraction operators. Section 3 defines the downward-solution and downward-failure properties and uses them to analyze some existing programs for design and diagnosis. Section 4 shows how these properties can be used to guide the selection of model abstraction operators for the task of analysis. Finally, section 5 concludes the paper.

2 Abstraction Operators

Since the notion of abstraction is most clear (and has been most thoroughly studied) in the context of search, we take state-space search as our foundation and generalize to more structured domains. The key step in this generalization is determining the nature of the models used by each task. Basing ourselves in search, we consider the tasks of design, diagnosis, and analysis.

2.1 Abstraction in Planning and Search

The use of an abstract representation for simplifying or speeding reasoning has been best studied in the context of search and planning. ABSTRIPS [Sacerdoti, 1974] solved planning problems abstractly by eliminating low priority preconditions and then refining the solution by gradually reintroducing detail. [Gaschnig, 1979] defined an abstract space as an edge supergraph of the concrete space and showed that the cost of solving the abstract problem provided an admissable heuristic for solving the concrete problem. More recently, [Mostow and Prieditis, 1989] implemented this approach and generalized the abstraction transformations to include node merging as well as the addition of edges in the abstract graph. [Korf, 1987] has shown that the use of a hierarchy of abstraction spaces can transform an exponential time problem into a linear time problem. We seek to extend these results to types of reasoning less commonly formalized as search: design, diagnosis, and especially analysis.

2.2 Models for Different Tasks

Although the tasks of design, diagnosis, and analysis can all be formalized as search, this reduction does not necessarilly give insight into why abstract models are powerful for these tasks. The key issue in understanding hierarchical design, diagnosis and analysis is the notion of *model.*

The objective of design is to generate a description of an artifact that satisfies the design specifications. Assuming a lumped element (system dynamics) framework [Shearer *et al.*, 1971], a given design might be described by a set of *components* with connections between named ports. Thus a model of a design would be a specification of the components and their connections. We are interested in abstraction-based design systems that use multiple spaces of models (e.g., in one space all transistors might be represented as discrete switches while in another they might be represented as nonlinear systems). Regardless of the model space in question, the same

problem solving operators would be used.

The objective of diagnosis is to explain why a particular artifact is malfunctioning. Assuming that the artifact can be thought of as a number of connected components, standard diagnosis algorithms use a model of the components to predict behavior based on the assumption that various components are functioning properly. A conflict between predicted and observed behavior allows the diagnoser to narrow the set of candidate faults. Hierarchical diagnosis programs such as XDE [Hamscher, 1988] use a series of abstract models to predict less detailed (and hence less costly to manipulate) behavioral descriptions for comparison with observation, but the same conflict recognition, candidate generation, and probe selection operations are used regardless.

Although there are many types of analysis (e.g., stability analysis, estimation of reliability or manufacturing cost, etc.), in this paper we take the objective of analysis to be the prediction of an artifact's time-varying behavior. In particular, we consider in some detail techniques for answering questions concerning the value of a parameter at a point in time. In their simplest form, these questions reduce to the evaluation of an inequality, for example, "Is the maximum torque on beam three greater than that required to break the beam?" To answer such an analysis question, one needs a model of the artifact that allows the prediction of how external inputs to the system (say forces) are distributed between components in the artifact. As is shown below, we have much to say about the use of abstract system models for the task of analysis.

2.3 Abstraction Operators

As we have seen, very similar component models are used in design, diagnosis, and analysis. In this section we catalog some of the ways one may transform a model into a new model that is more abstract.

- **Abstract Parameter Values.** One technique for creating a more abstract model is to relax real-valued parameters and use a qualitative representation instead. There is a natural flow from the real numbers to interval arithmetic to sign algebra [Murthy, 1988].

- **Abstract Component Constraints.** Typically, component models are built out of constraint equations. If the component models include dynamic equations, then there is a natural progression from ordinary differential equations (ODEs), to piecewise linear differential equations, to qualitative differential equations (QDEs), to a simple (signless) record of what parameters affect what other parameters.

By relaxing these constraints, a more abstract model is achieved. In addition, constraint equations can often be abstracted without changing their classification: for example, by eliminating insignificant terms. One important type of constraint abstraction is the introduction of an approximation reformulation [Weld, 1989]; this leads to an idealization of the original equation.

- **Temporal Abstraction.** By viewing behavior at a more coarse grain size, an abstract model is created. To this end, several operators have been advanced. Aggregation [Weld, 1986] disregards the intermediate changes in a recurring cycle of processes by determining the net effects of one iteration. XDE's class of 'change abstractions' ignores the particular values of a parameter over an interval, instead it computes simply whether or not the value changed [Hamscher, 1988]. XDE's class of 'sequence abstractions' describes the sequence of values taken by a parameter in an interval, but ignores the duration of each value.

- **Structural Consolidation.** By merging several components and considering them as one, a more abstract model is created. Typically this operator is followed by an abstraction of the resulting component constraints.

2.4 Notation

Applying one or more of these operators to a model results in a more abstract model. Since we are interested in the relation between a CONCRETE model and its ABSTRACT model, we consider the reformulation function that maps between concrete and abstract models, and we use the symbol Ψ_M to denote this abstraction reformulation. Typically an abstraction reformulation is a noninvertible surjection (from the space of concrete models onto the space of abstract models). We use Ψ_M^{-1} to denote the corresponding mapping from an abstract model to its preimage under Ψ_M (the set of concrete models that Ψ_M maps to the abstract model).

Determining whether a particular model satisfies the goals of a task requires comparing the behavior predicted by the model with some desired behavioral specification. Since the legal behavioral constraints vary between spaces of models, our abstraction reformulations must map concrete behavioral constraints to abstract ones. We use the Ψ_B and Ψ_B^{-1} functions to denote these mappings. Finally, we note that since the domains of Ψ_M and Ψ_B are disjoint, we can define a total reformulation function Ψ which has the union of the two domains as its domain. Similarly, we use Ψ^{-1} to map from abstract models and behavior constraints back to concrete ones.

The next section explains which abstraction reformulations are appropriate for which tasks and why.

3 Task-Driven Reformulation

We seek to explore the properties of the various model abstraction operators and their relation to the tasks of design, diagnosis, and analysis. We take as our starting point the following definition (adapted from [Tenenberg, 1989]).

Definition 1 *An abstraction reformulation Ψ has the* UPWARD-SOLUTION PROPERTY *iff $\Psi_M(\pi_c)$ is an abstract model satisfying abstract behavioral constraints $\Psi_B(\rho_c)$ for all concrete models π_c satisfying concrete constraints ρ_c (figure 1). Similarly, an abstraction reformulation Ψ has the* DOWNWARD-SOLUTION PROPERTY *iff for all concrete constraints ρ_c any abstract model π_a satisfying $\Psi_B(\rho_c)$ can be specialized to a concrete model satisfying ρ_c. I.e., there exists a $\pi_c \in \Psi_M^{-1}(\pi_a)$ such that π_c satsifies ρ_c.*

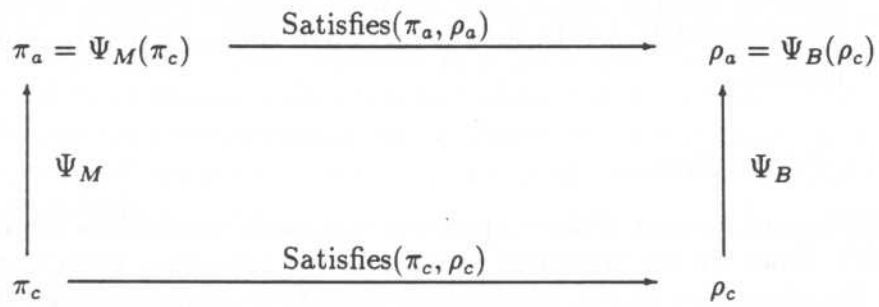

Figure 1: The Upward-Solution Property

We note that [Tenenberg, 1989] observed that that the upward-solution property is independent from the downward-solution property (neither implies the other). We now extend these properties.

Definition 2 *An abstraction reformulation Ψ has the* UPWARD-FAILURE PROPERTY *iff the fact that there is no model satisfying concrete constraints ρ_c implies that there is no abstract model satsifying $\Psi_B(\rho_c)$. Ψ has the* DOWNWARD-FAILURE PROPERTY *iff the fact that there is no model satifying abstract constraints ρ_a implies that forall $\rho_c \in \Psi_B^{-1}(\rho_a)$ there is no concrete model satisfying ρ_c.*

The following relationships are straightforward, but important.

Proposition 1 *An abstraction reformulation Ψ has the upward-solution property iff it has the downward-failure property; Ψ has the downward-solution property iff it has the upward-failure property.*

To make these notions concrete, consider the classic reformulation of the mutilated-checkerboard problem. It is clear that a checkerboard missing two diagonally opposed corners cannot be tiled by similarly-sized dominoes because each domino obscures both a red and black square and the checkerboard no longer has the same number of squares of each color. The abstraction of a tiling operation into an operation that decrements the number of remaining squares of each color has the downward-failure property and the upward-solution property. For an example of a useful reformulation with the other two properties, see section 3.1.

While these properties are clearly deserving of further exploration, we choose instead to use them in the context of specific tasks. In the next two sections we analyze some existing approaches to hierarchical design and diagnosis in terms of these properties. Then in section 4 we consider the task of analysis more carefully.

3.1 Design

In this section we analyze two existing design systems that use hierarchical representations, VEXED [Steinberg, 1987] and Ibis [Williams, 1989], in terms of our framework. Although both programs first search for satisfactory designs in an abstract space and then consider more concrete representations, analysis shows that the reformulations they use are very different. This is because VEXED embodies a theory of top-down refinement of library designs while Ibis implements a theory of innovative design.

VEXED views design as "top-down refinement plus constraint propagation." The system searches[1] for an abstract design that satisfies the desired constraints, and then specializes that model with a series of refinement rules. Since each refinement rule specifies "legal, correct implementations" of the abstract description, it is clear that VEXED's refinement hierarchy satisfies the downward-solution property. However VEXED does not have the upward-solution property since there may be legal designs that cannot be generated by the refinement rules. This abstraction hierarchy is an excellent choice for a system that does library-based design.

Ibis [Williams, 1989] takes quite a different approach to design. Given a design specification, Ibis first searches an abstract space, one of "interactions," to see if there is a way to connect changes in one parameter to changes

[1] VEXED's search was manually controlled.

in another. At this initial stage in design, Ibis ignores whether the causal connections are of the correct magnitude or even of the correct sign; the space of interactions represents the most abstract constraint abstraction operator listed in section 2.3. Once a promising path is discovered through the interaction space, Ibis attempts to refine the design by substituting a more concrete model of the primitive components and using the MINIMA algebraic reasoner [Williams, 1988] to see if the resulting design satisfies the concrete constraints. Since it is impossible for one quantity to influence another in a particular direction and with a particular magnitude unless there is some path of interaction between the two quantities, Ibis' representational abstraction satisfies the downward-failure property and hence the upward-solution property. But since some paths of interaction may act in the direction opposite to what was specified, Ibis does not satisfy the downward-solution property. Since Ibis is intended as a theory of innovative design this conservative approach (no concrete solutions are missed by searching in the abstract space) is appropriate.[2]

3.2 Diagnosis

We now analyze the hierarchical representations in the XDE hardware troubleshooter [Hamscher, 1988]. Recall that a key step in model-based diagnosis is conflict recognition — given a set of components,[3] determine if the artifact's observed behavior contradicts the assumption that the components in the set are functioning properly. If there is a conflict, then candidate hypotheses are created and a probe point selected for measurement to discriminate among the candidates. Conflict recognition can can be expressed in a form similar to that used in our analysis of diagnosis, since the observed misbehavior forms a kind of behavioral constraint. When the predicted behavior is inconsistent with measurements (i.e. a conflict is found), we will say that we have a "solution." Our interest is in whether these "solutions" map between representations. In other words, if an inconsistency is found in the abstract space, is it guaranteed that the inconsistency will persist when it is refined? Similarly, if no inconsistency is found in the abstract space, is it guaranteed that all components in the set are functioning properly at the concrete level?

XDE uses a hierarchical functional model of the circuit to be diagnosed and starts diagnosis with an abstract description, refining components only

[2]Our discussion of Ibis is necessarily cursory; for example, [Williams, 1989] describes a compact representation for the space of interactions which makes it a logical choice as an abstraction space.

[3]An 'environment' in the terminology of GDE [de Kleer and Williams, 1987].

as necessary to narrow the set of candidate faults. For example, an abstract model of a microprocessor might distinguish only two unfaulted states: stopped and running. The advantage of this representation is that behaviors can be quickly checked during conflict recognition. If the processor has been reset and should be running, but the oscilloscope shows that the program counter is not changing, then a conflict has been found. Because the behavioral abstractions used by XDE are sound (though incomplete), it has the downward-solution property. If an abstract model of a component allows a conflict to be recognized, then at least one fault is in the component. However, the fact that the microprocessor appears to be running, doesn't mean it is running correctly. Hence XDE's abstract representations do not have the downward-failure property.

4 Directing Reformulation for Analysis

We now show how a program could use the downward-solution and downward-failure properties to *guide* the introduction of simplifying assumptions for the task of analysis. The result is a framework that allows a program to ignore detail in a model, selectively. By using the particular question under consideration to determine which components to abstract, the program will reason efficiently yet still be sure of its results.

As mentioned above, we can take the goal of analysis to determine the truth value associated with an inequality statement about the value of one or more parameters at a given time. For example, in the system of figure 2, we might wish to ask "If force f is applied to the end of bar A, will beam C break?" Naturally, this can be translated into a question about an inequality: $\mathcal{S} > \mathcal{B}c$. "Is the shear force on beam C greater than the force required to break such a beam?"

The job for an intelligent analyst, then, is to determine the truth value of this inequality. But to calculate precisely the shear force requires a detailed model of beams, bending, joint friction, the action of spring S, and the flow of air around the sides of piston P, hence abstract models are preferable if they can be used. In this section, we show how the abstraction properties defined above enable the selection of simplifying assumptions while guaranteeing a valid answer.

The fundamental insight is simple. We say that the reformulation to an abstract space has the downward-solution property if whenever the abstract calculation of shear force $\mathcal{S}_a$ is greater than the abstract approximation of the force required to break the beam $\mathcal{B}_a$, then that inequality holds in the concrete model as well. Thus we say that the downward-solution property

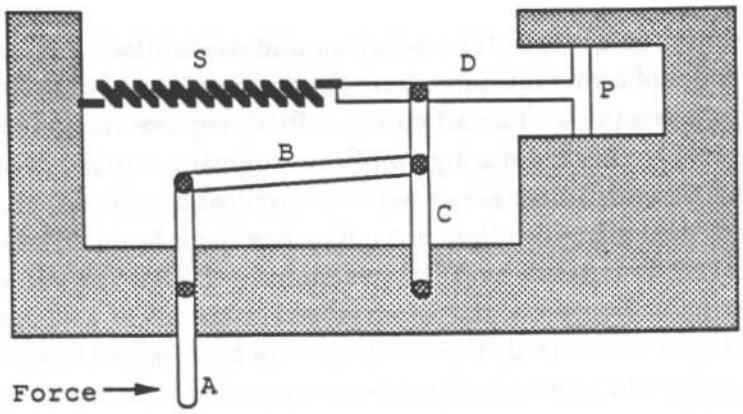

Figure 2: Will Beam C Break?

holds if

$$\mathcal{S}_a > \mathcal{B}_a \Rightarrow \mathcal{S} > \mathcal{B}$$

and the reformulation has the downward-failure property if

$$\mathcal{S}_a \leq \mathcal{B}_a \Rightarrow \mathcal{S} \leq \mathcal{B}$$

In other words, if an analysis in an abstract model with the downward-solution property suggests that the beam will break, then we can be sure that it will. Similarly, if analysis in an abstract model with the downward-failure property claims that the beam will not break, then we can be sure that it will not. An answer that does not meet these criteria admits doubt.

Thus a powerful reasoning strategy follows a kind of "devils advocate," and attempts both to prove and disprove this inequality by simplifying with problem reformulations that have either the downward-solution or downward-failure property. For example, one could try to prove that the force *will not* break beam C (in figure 2) by modeling spring S and piston P as infinitely stiff, thus considering D's position fixed. If beam C doesn't break if D is fixed, then it certainly won't break if there is give and play in D. Thus this abstraction satisfies the downward-failure property. Alternatively, one could try to show that the force *will* break beam C by choosing abstractions that underestimate the applied force; this would amount to a reformulation with the downward-solution property.

A key question, of course, is which abstraction operators lead to the downward-solution property and which engender downward-failure. This determination cannot be made except in the context of the particular artifact under analysis. Depending on the system at hand, an abstraction operator

may lead to an over- or an underestimate of various quantities. This suggests rethinking the doctrine of CLASS-WIDE ASSUMPTIONS [de Kleer and Brown, 1984] which states that all components in a given class should be modeled at the same level of abstraction. leading to the policy that if you model one joint as frictionless, you should use that model for all joints. But notice that this can result in an abstraction with *neither* the downward-solution nor the downward-failure property. While modeling joints A/B and B/C as frictionless increases the shear force on C, removing friction from C/D decreases the shear force.

5 Conclusions

We are encouraged by the flurry of results on automated model management [Kuipers, 1987, Falkenhainer and Forbus, 1988, Addanki *et al.*, 1989, Weld, 1989] since we believe that enabling the intelligent introduction of simplifying assumptions is the key step in building problem solvers that can deal with detailed models of complex physical systems. However, most existing modeling work in qualitative physics suffers from two major limitations.

- Programs do not generate their own abstractions — they require as input a graph of models [Penberthy, 1987] or its equivalent.
- Programs do not make sufficient use of information about the current task and question when selecting models.

In this paper we have presented a framework for analyzing task-driven reformulation. Abstraction operators extend a concrete model into a space of possible abstractions; the trick is to find a member of the abstraction space which is useful for the problem at hand. By extending the notion of the downward-solution property to continuous, physical domains, and by introducing the concept of the downward-failure property, we have provided a powerful tool for analyzing reformulation for the tasks of design, diagnosis, and analysis.

5.1 Related Work

Several other recent pieces of research have also investigated automatic abstraction of dynamic systems. Aggregation [Weld, 1986] recognizes repeating cycles and constructs a continuous process summary of the effect of each iteration. By performing transition analysis on the abstract description, simulation efficiency is greatly increased. Unfortunately, this abstraction technique is limited to cases with repeating processes.

Recently, a more formal model for the aggregation of dynamic structure was presented [Iwasaki and Bhandari, 1988]. This approach defines an aggregate variable for each subsystem of a nearly decomposable system and rewrites the equations in terms of the aggregate variables. While this approach has a solid mathematical foundation, the restructuring is not task driven.

5.2 Future Work

Much remains to be done. The space of reformulations generated by our abstraction operators is huge, yet we have not suggested heuristics for guiding search. For the task of analysis, we have yet to provide techniques for classifying the properties of abstraction operators as they affect a given problem. We continue to workon these questions and suspect that qualitative solutions are possible and promising. In particular, we believe that INTER-MODEL COMPARATIVE ANALYSIS [Weld, 1989] (based on approximation reformulations [Weld, 1990]) may enable abstraction operator classification, at least for the domain of analysis. For example, to see if abstracting away joint friction for joint A/B preserves the downward-solution property, one must determine whether this assumption will increase or decrease the shear force. This is precisely the task of inter-model comparative analysis.

ACKNOWLEDGEMENTS

Many thanks for conversations with Mark Shirley which started this line of thinking and conversations with Armand Prieditis which suggested new directions. This paper was improved by comments from Franz Amador, Tony Barrett, Steve Hanks and Scott Penberthy. The heroic efforts of Eric Lundberg saved (a version of) this paper from the wreckage of a dead magnetic disk, scant hours before the AAAI deadline.

This research was funded in part by National Science Foundation Grants IRI-8902010 and IRI-8957302, and a donation from the Xerox corporation.

References

[Addanki *et al.*, 1989] S. Addanki, R. Cremonini, and J. S. Penberthy. Reasoning about Assumptions in Graphs of Models. In *Proceedings of IJCAI-89*, August 1989.

[de Kleer and Brown, 1984] J. de Kleer and J. Brown. A Qualitative Physics Based on Confluences. *Artificial Intelligence*, 24, December 1984.

[de Kleer and Williams, 1987] J. de Kleer and B. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32, April 1987.

[Falkenhainer and Forbus, 1988] B. Falkenhainer and K. Forbus. Setting up Large Scale Qualitative Models. In *Proceedings of AAAI-88*, August 1988.

[Gaschnig, 1979] J. Gaschnig. A Problem Similarity Approach to Devising Heuristics: First Results. In *Proceedings of IJCAI-79*, 1979.

[Hamscher, 1988] W. Hamscher. Model-Based Troubleshooting of Digital Systems. AI-TR- 1074, MIT AI Lab, August 1988.

[Iwasaki and Bhandari, 1988] Y. Iwasaki and I. Bhandari. Formal Basis for Commonsense Abstraction of Dynamic Systems. In *Proceedings of AAAI-88*, pages 307–312, August 1988.

[Korf, 1987] R. Korf. Planning as Search: A Quantitative Approach. *Artificial Intelligence*, 33(1), September 1987.

[Kuipers, 1987] B. Kuipers. Abstraction by Time-Scale in Qualitative Simulation. In *Proceedings of AAAI-87*, July 1987.

[Minton, 1988] S. Minton. Quantitative Results Concerning the Utility of Explanation-Based Learning. In *Proceedings of AAAI-88*, pages 564–569, August 1988.

[Mostow and Prieditis, 1989] J. Mostow and A. Prieditis. Discovering Admissable Heuristics by Abstracting and Optimizing: A Transformational Approach. In *Proceedings IJCAI-89*, pages 701–707, August 1989.

[Murthy, 1988] S. Murthy. Qualitative Reasoning at Multiple Resolutions. In *Proceedings of AAAI-88*, pages 296–300, August 1988.

[Penberthy, 1987] J.S. Penberthy. Incremental Analysis and the Graph of Models: A First Step towards Analysis in the Plumber's World. MS Thesis, MIT Laboratory for Computer Science, January 1987.

[Sacerdoti, 1974] E. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence*, 5:115–135, 1974.

[Shearer *et al.*, 1971] J. Shearer, A. Murphy, and Richardson H. *Introduction to System Dynamics.* Addison-Wesley Publishing Company, Reading, MA, 1971.

[Steinberg, 1987] L. Steinberg. Design as Refinement Plus COnstraint Propagation: The VEXED Experience. In *Proceedings of AAAI-87*, pages 830–835, August 1987.

[Tenenberg, 1989] J. Tenenberg. Inheritance in Automated Planning. In *Proceedings of the International Conference on Knowledge Representation*, pages 475–485, May 1989.

[Van Harmelen and Bundy, 1988]
F. Van Harmelen and A. Bundy. Explanation-Based Generalisation = Partial Evaluation. *Artificial Intelligence*, 36(3), October 1988.

[Weld, 1986] D. Weld. The Use of Aggregation in Causal Simulation. *Artificial Intelligence*, 30(1), October 1986.

[Weld, 1989] D. Weld. Automated Model Switching: Discrepancy Driven Selection of Approximation Reformulations. Technical Report 89-08-01, University of Washington, Department of Computer Science and Engineering, October 1989.

[Weld, 1990] D. Weld. Approximation Reformulations. In *Submitted to AAAI-90*, Perhaps to appear 1990.

[Williams, 1988] B. Williams. MINIMA: A Symbolic Approach to Qualitative Algebraic Reasoning. In *Proceedings of AAAI-88*, August 1988.

[Williams, 1989] B. Williams. Invention from First Principles via Topologies of Interaction. PhD Thesis, MIT Artifical Intelligence Lab, June 1989.