Thoughts on a Practical Theory of Reformulation for Reasoning about Physical Systems

Berthe Y. Choueiry, Sheila McIlraith[*], Yumi Iwasaki
Tony Loeser, Todd Neller, Robert S. Engelmore, and Richard Fikes
Knowledge Systems Laboratory
Stanford University
Stanford, CA 94305-9020
Email: [choueiry|sam|iwasaki|loeser|neller|rse|fikes]@ksl.stanford.edu

Abstract

In this paper, we propose a practical framework for characterizing, evaluating and selecting reformulation techniques for reasoning about physical systems, with the long-term goal of automating the selection and application of these techniques. We view reformulation as a mapping from one encoding of a problem to another. A problem-solving task is in turn accomplished by the application of a sequence of reformulations to an initial problem encoding to produce a final encoding that addresses the task. Our framework provides the terminology to specify the conditions under which a particular reformulation technique is applicable, the cost associated with performing the reformulation, and the effects of the reformulation with respect to the problem encoding. As such it provides the vocabulary to characterize the selection of a sequence of reformulation techniques as a planning problem. Our framework is sufficiently flexible to accommodate previously proposed properties and metrics for reformulation. We have used the framework to characterize a variety of reformulation techniques, three of which are presented in this paper.

1 Introduction

Reformulation plays an important role in various intellectual activities and is ubiquitous in reasoning about physical systems. Reformulation improves the effectiveness of a mental or computational problem-solving process by recasting a problem into a new one that is tailored to a given task. The selection of reformulation techniques must be carried out relative to a problem-solving task. In this paper we examine the role of reformulation in reasoning about physical systems, and provide a practical framework for evaluating various reformulation techniques applicable to this class of problems.

Informally, we define reformulation to be a *mapping* from one encoding of a problem to another. A problem-solving task is accomplished by the application of a select sequence of reformulations to an initial problem encoding to produce a final encoding that addresses the task. We use the term reformulation to subsume the notions of *abstraction* and *approximation*, thereby avoiding any lexical implication that the mapping generalizes or simplifies the domain theory. Given an encoding of a problem and a reasoning task, one may choose to reformulate for any of the following reasons:

1. *Engine-driven problem re-encoding:* enabling the use of a particular reasoning engine by satisfying its input requirements, either because no engine exists to address the initial problem, in order to improve the performance of problem-solving, or to reduce the cost of the reasoning.
2. *Cognitive insight:* improving the user's understanding of the problem or solution space.

In order to develop a framework with sufficient detail to compare reformulation techniques, we focus on a specific class of problems, namely reasoning about physical systems. We require that the behavior of the physical system be expressible as a set of lumped-parameter hybrid (continuous and discrete) models, containing algebraic or differential equations, but problem solving need not be restricted to a direct manipulation of equations. Finally, we require that the task be motivated by a specific query, thus constraining the computational machinery necessary to carry it out.

The long-term goal of our research is to develop an automatic task-driven capability that selects and applies appropriate reformulation techniques in the course of modeling and analyzing the behavior of physical systems. The contribution of this paper is a practical framework for evaluating specific reformulation techniques with respect to the restricted class of problems we described above. The motivation for developing such a framework came from the observation that much of the previous work on reformulation (including abstraction and approximation) was either too specific or too general to be of practical use in developing automated reformulation mechanisms. Our framework

[*] and Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304.

Working notes of the Symposium on Abstraction, Reformulation and Approximation (SARA'98). A previous version of this paper appears in the working notes of the Twelfth International Workshop on Qualitative Reasoning (QR'98).

provides a significant step towards this long-term goal by defining general criteria for understanding the properties of various reformulation techniques.

The paper is organized as follows. Section 2 introduces our conception of the processing stages involved in reasoning about physical systems, setting the context for our reformulation framework. Section 3 introduces the framework itself and presents a set of evaluators for assessing the effects of reformulation. Section 4 applies the framework to three examples of reformulation techniques previously described in the literature. Section 5 discusses related work, and Section 6 concludes with a brief summary and outlines directions for future research.

2 Reasoning about Physical Systems

In this section we describe the various processes that may be executed in reasoning about physical systems. We illustrate these processes in terms of several examples drawn from the literature. Starting with a description of the task of interest, we perceive the entire endeavor as a progression through the three *processing stages* illustrated in Fig. 1, namely: the model, the equation, and the solution processing stages. Identifying these stages has proved instructive in distinguishing and situating the various reformulation techniques useful for reasoning about physical systems.

Task description. Reasoning about physical systems begins with a *task description* specified by the following four elements: the domain theory, the scenario, the query, and the modeling assumptions. The domain theory is a corpus of knowledge corresponding to heterogeneous, possibly redundant or contradictory descriptions of a physical system, including logical statements, symbolic equations, numeric parameters. The scenario is a description of a particular problem instance (*e.g.*, a set of system components and their physical structure, and the initial conditions of the system). The query is an explicit specification of the user's question (*e.g.*, variables, their quantitative or qualitative values, direction of change at specific time points). Finally, the modeling assumptions include assumptions that the problem solver may make in order to broadly delimit the scope of the answer to the query (*e.g.*, the temporal and physical extent of its coverage, granularity).

Model processing. Given a task description, the *model building* process assembles the relevant aspects of the domain theory to produce a model, which is an instantiation of a subset of the domain theory that is both consistent and sufficient to address the query. A model at this point often consists of knowledge of the physical structure (components and their topology, for example) as well as knowledge of the relevant physical phenomena (including the conditions under which they are active), in contrast to the purely mathematical model of the following stage. A typical example of a model building process is compositional modeling as in [4; 15; 12] and in the modeling algorithm of TRIPEL [19].

This process can be followed by a *model reformulation* process. Model reformulation may involve structural consolidation [24], time scale selection [19], or simplification [16] or expansion of a model through addition, deletion, or replacement of a description of components or phenomena.

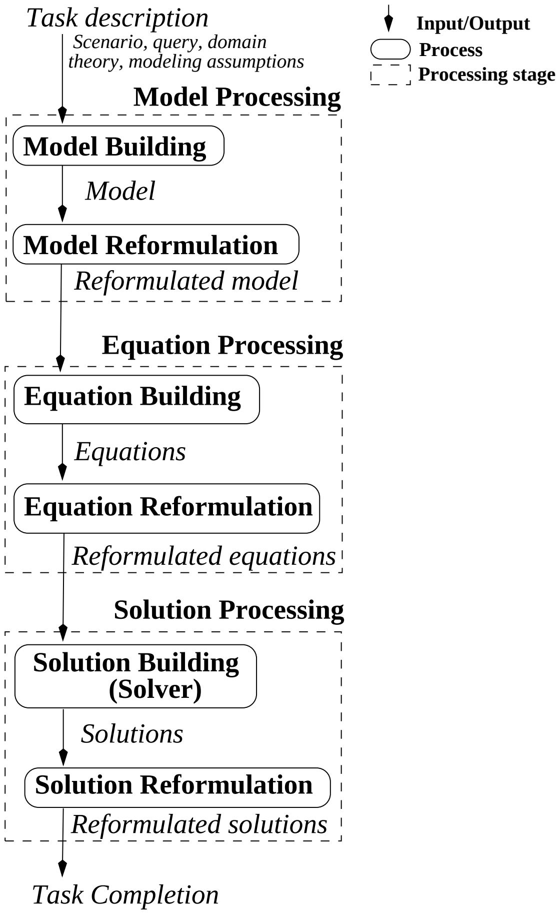

Figure 1: *Reasoning about physical systems.* Stages and their corresponding processes.

Equation processing. *Equation building* produces an equation model[1] either directly from a task description or through reformulation of a model into a set of mathematical equations describing the behavior of the system. For example, the Qualitative Physics Compiler [5] converts a model expressed in QP Theory [6] into a set of qualitative differential

[1]We distinguish between non-equational models and equational ones in order to capture the various possibilities for manipulating models reported to date in the literature for reasoning about physical systems.

equations.

An *equation reformulation* process may then be carried out. An equation reformulation is primarily motivated by a desire to transform the present equation model into a form that is amenable to a particular problem-solving engine. Many reformulations operate at this stage, such as mapping polar to Cartesian coordinates, mapping a time domain to a frequency domain, reformulating ordinary differential equations as qualitative differential equations, dropping insignificant terms, linearizing, and aggregating nearly decomposable systems.

Solution processing. The *solution building* process is often a problem-solving engine acting on either the model (*e.g.*, QPE [7]) or the equations (*e.g.*, QSIM [11] and Mathlab®) to produce one or more solutions to the query.

A *solution reformulation* process may subsequently be performed to enhance cognitive insight. Examples of such reformulations are summarization [14] and explanation by generation of active documentation [9]. Solution reformulation may also be applied for engine-driven problem re-encoding. For example, Clancy and Kuipers [1] interleave a QSIM simulation with the aggregation of partial solutions corresponding to chatter in a qualitative simulation. In so doing, they significantly improve the overall performance of QSIM on their problem.

Note that reasoning need not necessarily transition through every processing stage, nor through every process within a stage. For example, a particular problem-solving task may go from task description directly to the equation processing stage, or from the model processing stage directly to the solution processing stage. Moreover, a problem-solving task may loop any number of times through one or more of the individual processes. In the example of solution reformulation reported above, Clancy and Kuipers [1] loop over the solution building and solution reformulation processes. During the simulation of the behavior of a dynamical system, the operating conditions may change as a result of system dynamics. The model used for simulation must then be updated to comply with the new assumptions. In such situations, the reasoning process may involve a cycle of model processing, equation processing and solution processing as the system sequentially transitions from one discrete state to another.

In Section 4, we provide one example of reformulation at each of the model, equation, and solution processing stages.

3 Proposed framework

In this section we introduce a framework and a terminology for characterizing and evaluating reformulation techniques for reasoning about physical systems. Section 3.1 introduces the components of the framework. Section 3.2 shows how selection of a sequence of reformulations can be reduced to a planning task. Finally, Section 3.3 introduces the attributes necessary to characterize a reformulation technique, so that selection can be performed.

3.1 Components of the framework

Reformulation is a mapping of an original problem into a new problem, as shown in Fig. 2.

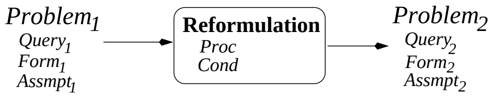

Figure 2: *Reformulation.*

We distinguish two primary components, the *problem* and the *reformulation*, and a composite component, the *strategy*, obtained from composing the former two.

3.1.1 Problem

We define a problem as a three-tuple: $\texttt{Problem} = \langle \texttt{Query}, \texttt{Form}, \texttt{Assmptn} \rangle$. `Query` specifies the question that the user is trying to answer. `Form` denotes the formulation, *i.e.* the conceptualization of the domain. Finally, `Assmptn` designates the conditions under which the formulation is valid, *e.g.* the domain of applicability and the temporal granularity. In Fig. 3 the problem P_1 is represented as a node.

Figure 3: *Problem,* P_1.

3.1.2 Reformulation

A reformulation technique is applied to an original problem $\texttt{Problem}_1 = \langle \texttt{Query}_1, \texttt{Form}_1, \texttt{Assmptn}_1 \rangle$, to produce the reformulated problem, $\texttt{Problem}_2$. We describe the reformulation technique as a tuple: $\texttt{Reformulation} = \langle \texttt{Cond}, \texttt{Proc} \rangle$. `Cond` denotes the applicability conditions and `Proc` denotes the procedure that maps the original problem into a reformulated one. `Cond` is a set of conditions that must be satisfied by $\texttt{Problem}_1$ for the reformulation method to be technically applicable. It must be noted that `Cond` is a *necessary* condition for the applicability of `Proc`. `Proc` is a computable procedure that realizes a mapping. In Fig. 4 the reformulation R_a is illustrated as a transition between nodes representing two problems P_1 and P_2.

As mentioned in the introduction, the decision to perform a reformulation may be motivated by the availability of a suitable problem-solving engine and its

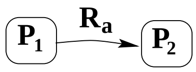

Figure 4: *Applying reformulation R_a to an initial problem to produce a reformulated problem.*

performance for solving a problem[2]. A solution engine is applied to a problem to produce a result as an answer to the query. We use the term "engine" broadly to include anything from an algorithm, to a special-purpose simulation program, to a general-purpose solution package such as Mathematica®. There could be multiple engines at one's disposal to solve the original or reformulated problems. Alternately, there may be none, when the problem is too difficult.

It follows from our definition of reformulation that a *solution engine is nothing but a reformulation* that partially or completely answers a query. More specifically, the information necessary to answer the query exists implicitly in the problem encoding (*i.e.*, formulation, query, and assumptions). A solution engine merely manipulates the problem to make this implicit answer explicit. As a consequence, all subsequent discussion of the general notion of a reformulation procedure also pertains to what has traditionally been called a problem-solving engine or solution engine.

3.1.3 Strategy

A reformulation is simply a mapping from one problem encoding to another. Thus, a reformulation can be understood as a step *towards* providing an answer to the query. A sequence of reformulations, which may include one or more engines, constitutes one *strategy* for addressing a task. Execution of a strategy constitutes problem solving.

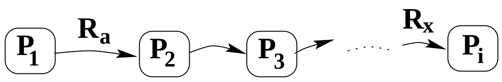

Figure 5: *Strategy.*

We define a strategy S_i to be a sequence of reformulations, $[R_a \dots R_x]$ that is applied to an original problem P_1. The path $[P_1 R_a P_2 \dots R_x P_i]$ in Fig. 5 is an example of the execution of such a strategy. Any subsequence, S_k, of S_i, starting at P_1 and stopping at any intermediary problem P_k, between P_1 and P_i, is also a strategy, and is called a sub-strategy of S_i.

3.2 Reasoning as plan execution

We perceive reasoning about physical systems to proceed according to processes identified in Fig. 1. According to this figure, the content of the initial input, *i.e.* the task description, is gradually modified by a combination of any number of processes culminating in an answer to the query. Given our definition of reformulation, any of these processes is a reformulation. The stage of processing distinguishes whether the reformulation is applied to a collection of model fragments, an equational model, or to one or more solutions. Reasoning about physical systems is thus a successive application of reformulation procedures that transforms an initial problem encoding.

Problem solving involves the successive application of reformulation procedures to an initial problem encoding to produce a final problem encoding. Clearly there may be multiple sequences of reformulations that may be applied to address the problem solving task, as illustrated in the figure below. Identifying such sequences of reformulation procedures can be viewed as a planning problem in which the *states* are problem encodings, the *transition* (or actions) are reformulations, and the *plans* are strategies. Hence, the task of reasoning about physical systems becomes an execution of the selected plan.

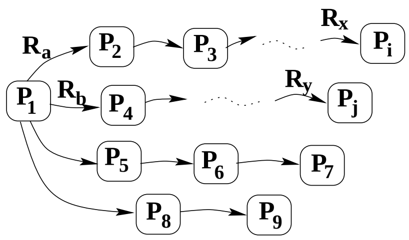

Figure 6: *Reasoning about physical systems is a plan execution.*

In Fig. 6 we illustrate a tree of four alternative strategies. In practice, as for planning, resources may be limited, and one may want to associate a utility or objective function to the problem solving task. We expect the user to provide the *goal* of the problem-solving task in terms of a *goal test* and of an *objective function* that specifies the importance of some desired features of the problem and the resources available. In this context, selecting an optimal plan or strategy becomes a multi-criteria optimization problem.

3.3 Evaluating and comparing components

To articulate the goal driving this planning process, we identify features of a problem, of a reformulation, and of the application of a reformulation to a problem that are relevant to selection of reformulation procedures. These features are divided into sets, relative to the components of our framework. The sets of features are incrementally augmented and refined as one explores, defines, and proposes new reformulation procedures. There are three main categories of sets, depending on whether they evaluate a component of the framework

[2] In this paper, we do not address reformulations that apply to the engine itself, as proposed in [8], because such reformulations do not seem to arise in the class of problems we address.

(called *evaluators*, denoted Evals), assess the change due to a reformulation (called *change indicators*, denoted Diffs), or compare components obtained by distinct strategies (called *comparators*, denoted Compars). We first introduce these sets, then we discuss each of them in further details for each of the components to which they apply.

Evaluators. For comparing the reformulation techniques described in the literature, and for assessing and comparing reformulation techniques in the context of problem solving, we must first define measures or evaluators that designate relevant characteristics of problem and the reformulation or their inter-relationships. We distinguish three sets of such evaluators, $\texttt{Evals}_{\texttt{prob}}$, $\texttt{Evals}_{\texttt{reform}}$, and $\texttt{Evals}_{\texttt{strat}}$ corresponding to evaluators that assess aspects of a problem, a reformulation technique, and a strategy respectively.

Change indicators. For tracking the evolution of a problem-solving task along successive reformulations, it is important to be able to characterize the changes that occur in the problem as the result of reformulation. One way to capture these changes is to measure the difference in the values returned by the evaluators before and after one or more reformulations. Another way is to measure changes between the application of different strategies to the same initial problem encoding. The values returned by change indicators are not necessarily quantitative; they could be qualitative or logical, but must at least capture some notion of change or evolution. We identify two sets of change indicators, $\texttt{Diffs}_{\texttt{prob}}$, and $\texttt{Diffs}_{\texttt{strat}}$.

Comparators. An essential aspect of our framework is the ability to articulate the relative merits of alternative strategies for problem-solving by comparing between their components. We introduce two sets of comparators, $\texttt{Compars}_{\texttt{prob}}$, and $\texttt{Compars}_{\texttt{strat}}$.

3.3.1 Problem

Below we introduce the terminology for characterizing and comparing problems. We illustrate this vocabulary in Fig. 7.

1. $\texttt{Evals}_{\texttt{prob}}(P_i)$ denotes the set of evaluators for assessing some aspects of a problem P_i, including the quality of the answer 'contained' in P_i.

   In the most general case, the elements in this set can be defined with respect to any of the three elements of the problem, *i.e.* the formulation, query, or assumptions. In examining a wide collection of reformulation techniques, we have found that the query and assumptions often remain unchanged before and after reformulation, and that most evaluators are functions applied exclusively to the formulation. Counterexamples do exist and will be discussed in a forthcoming technical report.

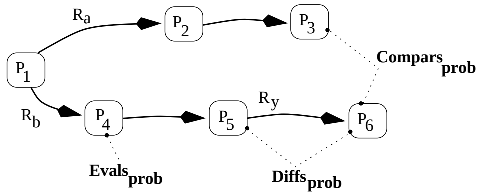

Figure 7: *Evaluating and comparing problems.*

   These evaluators usually provide a quantitative assessment of some aspect of the formulation (*e.g.*, size) or of its logical properties (*e.g.*, provability and refutation). They can also address qualitative, less quantifiable, aspects of the formulation (*e.g.*, expressive power). For a system of equations, an example of a quantitative evaluator is the number of equations or variables, or the number of terms per equation; an example of a qualitative evaluator is adherence of the equations to some canonical form. Other evaluators of the formulation that appear in the literature include scope [23] (which is the range of phenomena that it can describe), expressiveness, syntactic form, simplicity, generality, relevance, absence of irrelevant information, and language restriction to familiar terms [22]. It is important to define an evaluator in sufficient detail. In the case of simplicity, for example, we must define the specifics of how it is measured (*e.g.*, the number of variables/equations in a equation set, or the number of components in a model).

   Some of the evaluators in $\texttt{Evals}_{\texttt{prob}}$ are dedicated to assessing the result to the query as it is made explicit in Form. Examples of such evaluators are the soundness of the result, and its precision. These are typically the evaluators to use in the test that determines whether the goal of the planning process is achieved.

2. $\texttt{Diffs}_{\texttt{prob}}(P_i, P_j)$ denotes a set of effects of the reformulation, thus measuring a change in some feature of the problem. Any element in this set measures the change between the corresponding elements in $\texttt{Evals}_{\texttt{prob}}(P_i)$ and $\texttt{Evals}_{\texttt{prob}}(P_j)$, such that P_i and P_j are situated along the same strategy S_k. When P_i and P_j are adjacent in S_k, $\texttt{Diffs}_{\texttt{prob}}(P_i, P_j)$ indicates the effects of applying a reformulation to P_i. When P_i and P_j are not adjacent in S_k, it indicates the effects of the application of a sequence of reformulations.

   One possible effect of reformulation on the problem is to improve cognitive insight; this is common at the solution reformulation stage, see Fig. 1. For example, if the original formulation is too complex for a user to understand, reformulation may produce a description better suited to human understanding. Other examples of effects on the formulation are the following properties theorem increasing/decreasing/constant, upward/downward solution [21], upward/downward-failure [24], ordered monotonicity [10], and safety [2].

Similarly to the case of $\texttt{Evals}_{\texttt{prob}}(P_i)$, some elements of $\texttt{Diffs}_{\texttt{prob}}(P_i, P_j)$ are dedicated to assessing the change of some features of the result. An example of such an effect is a 10% loss in the accuracy of the result.

The change between two problems, P_i and P_j, reflects the effect of a reformulation (alternatively, a sequence of reformulations) on P_i. This outcome can also be predicted from considering the mathematical properties of the reformulation itself when applied to P_i (alternatively, the composition of the properties of the sequence of reformulations). There are, thus, two redundant ways of expressing this change, either as $\texttt{Diffs}_{\texttt{prob}}(P_i, P_j)$ or as $\texttt{Effects}_{\texttt{prob}}(R_k, P_i)$, which simply captures the effects of applying the reformulation R_k to P_i. For instance, Struss [20] considers reformulations procedures, called representational transformations, that are surjective, and not injective; Giunchiglia and Walsh [8] study procedures that are computable surjective total functions between two formal systems. The relationship between $\texttt{Effects}_{\texttt{prob}}$, $\texttt{Evals}_{\texttt{prob}}$ and $\texttt{Diffs}_{\texttt{prob}}$, can be described as follows:

$$\begin{aligned}\texttt{Effects}_{\texttt{prob}}(R_k, P_i) &= f(\texttt{Evals}_{\texttt{prob}}(P_j)) \qquad (1)\\ &= g(\texttt{Evals}_{\texttt{prob}}(P_j), \texttt{Diffs}_{\texttt{prob}}(P_i, P_j))\end{aligned}$$

where f and g are functions to be defined. For instance, one may want to state that a reformulation R_k doubles the size of P_i, thus specifying $\texttt{Effects}_{\texttt{prob}}(R_k, P_i)$, or that the size of P_j is twice that of P_i by taking the ratio of the elements 'size' in $\texttt{Evals}_{\texttt{prob}}(P_j)$ and $\texttt{Evals}_{\texttt{prob}}(P_i)$, referring thus to an element of $\texttt{Diffs}_{\texttt{prob}}(P_i, P_j)$. We choose to include both representations and to not arbitrarily favor one possibility over the other.

3. $\texttt{Compars}_{\texttt{prob}}(P_i, P_j)$ denotes a set of effects of two *distinct* strategies S_k and S_l applied to a given problem by measuring a change in some features between P_i and P_j resulting from applying S_k and S_l to the problem. It measures the relative merits, with respect to the problem, of two alternative reasoning strategies.

   Similarly to $\texttt{Evals}_{\texttt{prob}}$ and $\texttt{Diffs}_{\texttt{prob}}$, $\texttt{Compars}_{\texttt{prob}}$ encompass elements dedicated to comparing some features of the results in the problems obtained by the two alternative strategies.

3.3.2 Reformulation

For the reformulation, we introduce $\texttt{Evals}_{\texttt{ref}}$ a set of evaluators for assessing the reformulation, *i.e.* the conditions and the procedure.

This set is somehow complex. It contains evaluators that describe the reformulation technique in *absolute* terms (*e.g.*, the size of the code, the programming language it is written in, the price of a commercial software, the human effort required to exploit it, and perhaps whether it requires a special hardware or a human expert).

Moreover, $\texttt{Evals}_{\texttt{ref}}$ includes functions that assess the behavior of the reformulation technique *relative* to a given problem, for instance, its time complexity when applied to the problem. An example of an evaluator that applies to the conditions of the reformulation, $\texttt{Cond}$, is the tractability of verifying them. A typical and important evaluator is the computational complexity of the procedure, $\texttt{Proc}$, with respect to the original problem, denoted $\texttt{Complexity}(\texttt{Proc}, \texttt{Problem}_1)$.

3.3.3 Strategy

Below we introduce the evaluators, change indicators and comparators applicable to strategies, while illustrating them in Fig. 8.

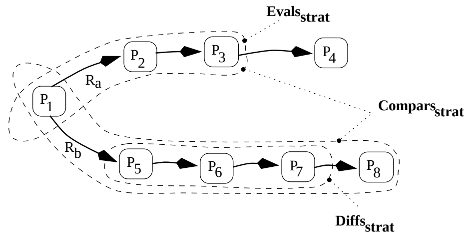

Figure 8: *Evaluating and comparing strategies.*

1. $\texttt{Evals}_{\texttt{strat}}(S_i)$ denotes a set of evaluators for assessing some aspects of a problem-solving strategy S_i. An element in this set is obtained by considering, over a given path, some combination of the values of an element $\texttt{Evals}_{\texttt{ref}}$, which measure features of the reformulation technique as introduced above. En example of such an evaluator is the cost of the strategy, assessed as the sum of the costs of applying the reformulations to the corresponding problem and the absolute cost of the procedures (*e.g.*, commercial price).

   These are typically the evaluators to use in the objective function that expresses the preferences and manages the resources of the planning process.

2. $\texttt{Diffs}_{\texttt{strat}}(S_i, S_j)$ denotes a set of the effects of *extending* a strategy S_i by one or more reformulation steps into a strategy S_j. Any element in this set measures the change between the corresponding elements in $\texttt{Evals}_{\texttt{strat}}(S_i)$ and $\texttt{Evals}_{\texttt{strat}}(S_j)$, such that S_i is a sub-strategy of S_j. Examples of such elements are increase in cost, loss of time, and consumption of available resources.

3. $\texttt{Compars}_{\texttt{strat}}(S_i, S_j)$ denotes the set of effects of two *distinct* reasoning strategies on an original problem by measuring a change of some feature in $\texttt{Evals}_{\texttt{strat}}(S_i)$ and $\texttt{Evals}_{\texttt{strat}}(S_j)$, thus yielding an assessment of the relative merits of the strategies. An element in this set is obtained by measuring the

difference, or ratio, of an element of $\texttt{Evals}_{\texttt{strat}}$ for each path.

Traditionally, a reformulation is said to be *cost-wise* beneficial when the cost of reformulating the problem and that of solving the reformulated problem do not exceed the cost of solving the original problem. This is typically an element of $\texttt{Compars}_{\texttt{strat}}$.

Since one of the goals of reformulation is to improve overall problem-solving performance, the reformulation procedure itself should not significantly add to the computational cost. However, sometimes there is no solution engine applicable to the original problem, and consequently any amount of effort to reformulate to make it solvable is justifiable. A *cost-intensive* reformulation may also be justified when it is performed off-line to improve runtime performance of a system.

3.3.4 Remark

Observe that the evaluators for the problem, reformulation, and strategy are not necessarily independent. For example, simplification of a set of equations often reduces the size of the formulation (measured by $\texttt{Evals}_{\texttt{prob}}$) and reduces the cost of the reformulation (measured by $\texttt{Evals}_{\texttt{ref}}$ and consequently by $\texttt{Evals}_{\texttt{strat}}$), at the expense of also reducing the precision of the result (measured again by $\texttt{Evals}_{\texttt{prob}}$).

4 Illustrative examples

In this section, we examine three reformulation techniques described in the literature from the perspective of our framework for reformulation. These techniques are representative of the types of reformulation that can occur at the three stages of reasoning about physical systems. Each of the reviewed examples consists of a summary of the reformulation technique and the desiderata, followed by a characterization of the reformulation procedure in terms of our framework. Figures 9, 10, and 11 illustrate the most critical aspects of the process. Due to space limitations, effects are mostly summarized in prose rather than exact and detailed definitions of relevant evaluators. Moreover, the authors do not provide a comparison of their techniques with other procedures, possibly because none exists. Thus, we will not discuss comparators of problems and strategies in the context of these examples.

4.1 Model reformulation: Simplification

Nayak and Joskowicz [16] propose a model reformulation technique that simplifies a compositional model of a device, while maintaining its ability to provide a causal explanation of the expected behavior of the device. The primary objective of their work is to perform efficient compositional modeling for generating parsimonious causal explanations of the functioning of a device. They provide tools for model-fragment library indexing and selection to support the construction of device models. The reformulation procedure is applied to the model thus built in order to simplify it. We focus here on this simplification process.

Given a device description, the expected behavior of a device, and the above mentioned tools, the authors provide a model building algorithm that composes an initial *adequate* model of the device. This initial model is adequate in that it explains the expected behavior, includes significant phenomena, and excludes insignificant and irrelevant phenomena. However, it may not be as *parsimonious* as it could be; that is, it may be possible to further approximate the model fragments, according to the causal approximations defined in the library, while maintaining the structural and behavioral constraints of the device, and the ability of the model to explain the expected behavior. The reformulation procedure transforms the composed adequate model into one that is both adequate and parsimonious. This procedure is predicated on the fact that all model fragment approximations provided in the library are causal and acyclic.

The reformulation procedure is portrayed in our framework as follows, see Fig. 9. Generating causal ex-

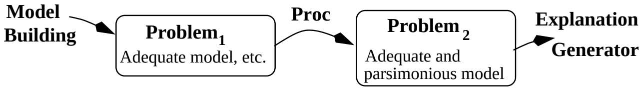

Figure 9: *Model simplification.*

planations is the stated problem-solving objective, but the authors do not propose a specific problem solver. As a consequence, we do not evaluate this model reformulation in the context of a larger strategy that includes the generation of causal explanations, and we restrict our evaluation to only one transition, corresponding to the model simplification. As a result, we will not discuss evaluators that apply to the strategy, but only to the problem and reformulation. Nevertheless, the authors observe that their notion of simplicity "does not guarantee that a simpler model will be more efficient to simulate or will produce simpler causal explanations than more complex ones." Thus, their choice for simplicity strongly affects the quality of the explanation generated by the overall strategy. The metric for evaluating the reformulation procedure is parsimony of the problem formulation.

Characteristics of the problem:

$\texttt{Form}_1$ comprises:

- an adequate model of a device that consists of ordinary differential equations, algebraic equations and qualitative equations,
- the expected behavior of the device,
- structural and behavioral constraints, and
- a library of model fragment approximations, in

which: (1) models [3] are restricted to time-varying, equilibrium lumped parameter ones, (2) approximations are causal and (3) the approximation relation is acyclic.

$\mathtt{Form_2}$: A *parsimonious* adequate model of a device and the above-mentioned library.

$\mathtt{Query_1} = \mathtt{Query_2}$: Generate a causal explanation of the specified expected behavior.

$\mathtt{Assmptn_1} = \mathtt{Assmptn_2}$: Whatever assumptions underlying the library of model fragments.

Characteristics of the reformulation:

`Proc`: Input to the reformulation procedure is an adequate compositional model that may contain unnecessary model fragments and may not be as approximated as possible.

The reformulation procedure exploits two operators: (1) Replacement of a model fragment by one of its immediate approximations, as defined in the model fragment library; and (2) Removal of an unnecessary model fragment. The first operator is applied repeatedly ensuring that the resultant model can explain the expected behavior. This is achieved by an order of magnitude reasoner. The second operator is then applied, again ensuring that the expected behavior can be explained and that all the structural and behavioral coherence constraints are satisfied. Note that the reformulation procedure generates one simplest adequate model. More than one may exist but the procedure stops after finding the first.

`Cond`: The approximations in the model fragment library must be causal and the approximation relation acyclic.

Evaluators and effects:

$\mathtt{Evals_{prob}}$: The problem is evaluated with respect to the parsimony of the problem formulation. The effect of reformulation on the problem, $\mathtt{Effects_{prob}}$, is the simplest compositional device model that will explain the expected behavior.

$\mathtt{Evals_{reform}}$: The reformulation is evaluated with respect to the complexity of the reformulation procedure relative to the problem encoding. With respect to this problem and reformulation, $\mathtt{Complexity(Proc, P_1)}$ is assessed to be tractable, provided the order of magnitude reasoning used to verify that the simplified model still explains the expected behavior is approximated to be polynomial. The reason for the tractability of the procedure is as follows. Because of the compositional modeling paradigm and the provision of causal approximations, the reformulation algorithm need not consider all combinations of model fragments during simplification. It considers each model fragment independently, and replaces it by one of its immediate simplifications according to the causal approximation relation. The algorithm computes the simplest adequate model (where simplest means that no model fragment can be replaced by a simpler one that satisfies the expected behavior), and it stops after finding the first adequate simplest model.

[3] Although models may use ordinary differential equations and qualitative equations, their behavior is studied only at equilibrium. Relaxing this constraint would require to expand the definition of causal relations.

4.2 Equation reformulation and solution building: linearization and stability

This example illustrates a problem-solving strategy that exploits linearization to determine the stability of a particular class of nonlinear systems. The problem-solving strategy consists of two sequential reformulations—an equation reformulation $\mathtt{R_a}$ which maps problem encoding $\mathtt{P_1}$ into problem encoding $\mathtt{P_2}$, and a problem-solver reformulation $\mathtt{R_b}$ which maps $\mathtt{P_2}$ into $\mathtt{P_3}$.

Linearization is a strategy commonly used to evaluate the stability of a nonlinear system near one of its equilibrium points. The reformulation is used to facilitate the inference of this stability property: it replaces the analysis of the stability of the nonlinear system by that of the linear system, derived by linearization of the equations of the nonlinear system. The stability of the resulting linear system is determined by the location of the eigenvalues of the system matrix in the complex plane. The main rationale for exploiting this strategy is that "for small deviations from the equilibrium point, the performance of the system is approximately governed by the linear terms. These terms dominate and thus determine stability–provided that the linear terms do not vanish" [13]. If this is not the case, a separate analysis is required.

In general, no problem-solver reformulation is available to directly determine the stability of nonlinear systems[4]. Hence, the motivation for performing this sequence of reformulations is engine-driven problem re-encoding. The metric for evaluating the strategy is with respect to the 'answerability' of the stability question. The strategy is deemed to be successful if an answer to the stability question can always be given and a stability region near an equilibrium point estimated. The reformulation procedures are portrayed in our framework as follows, see Fig. 10.

Characteristics of the problem:

$\mathtt{Form_1}$ comprises:

[4] Unless one is given a Liapunov function that can be used to prove stability within a region containing the equilibrium point.

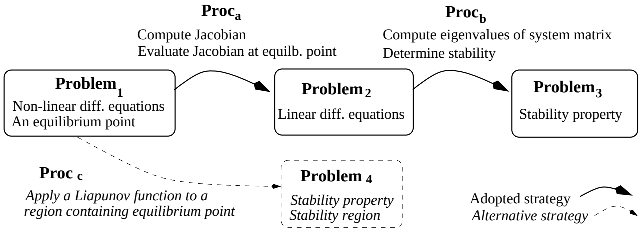

Figure 10: *Stability by linearization.*

- a set of nonlinear time-invariant[5] differential equations $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$ of n variables, and
- an equilibrium point for the nonlinear system.

$\mathtt{Form}_2$ is a set of time-invariant linear differential equations of n variables that approximate $\mathtt{Form}_1$ at an equilibrium point.

The system matrix of the linearized system is the Jacobian of the nonlinear system computed at the equilibrium point. It has the same number of variables and equations as $\mathtt{Form}_1$.

$\mathtt{Form}_3$ is $\mathtt{Form}_2$ plus the result, *i.e.* one symbol of $\{\text{stable}, \text{unstable}, \text{unknown}\}$.

$\mathtt{Query}_1 = \mathtt{Query}_2 = \mathtt{Query}_3$**:** Determine the stability properties near an equilibrium point.

$\mathtt{Assmptn}_1 = \mathtt{Assmptn}_2 = \mathtt{Assmptn}_3$**:** None.

Characteristics of the reformulations:

$\mathtt{Proc}_\mathtt{a}$ comprises the following steps: (1) Computation of the Jacobian of the nonlinear system. (2) Evaluation of the Jacobian at the equilibrium point.

$\mathtt{Cond}_\mathtt{a}$**:** Input must conform to $\mathtt{Form}_1$.

$\mathtt{Proc}_\mathtt{b}$ determine the stability near a specific equilibrium point, the eigenvalues, λ_i, of the system matrix of the linearized system are determined. The stability of the original system is inferred from that of the linearized version as follows:

- If at least one eigenvalue is found to be in the right-hand side of the complex plane ($\exists i,\ \mathrm{Re}(\lambda_i) > 0$), the nonlinear system is unstable.
- If all eigenvalues are in the left-hand side of the complex plane ($\forall i, \mathrm{Re}(\lambda_i) < 0$), the nonlinear system is stable.
- If all eigenvalues are in the left-hand side of the complex plane, but at least one has a zero real value ($\exists i,\ \mathrm{Re}(\lambda_i) = 0$), then no conclusions can be drawn for the stability of the nonlinear system, and one must analyze the higher order terms of the function $\mathbf{f}$.

[5]The dynamic behavior of a continuous system of n variables is described by a set of differential equations of the following general form: $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t)$. The system is said to be *time-invariant* when the functions $\mathbf{f}$ do not depend explicitly on time, *i.e.* $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$.

$\mathtt{Cond}_\mathtt{b}$**:** The problem must be formulated as described in $\mathtt{Form}_2$.

Evaluators and effects:

$\mathtt{Evals}_\mathtt{prob}(P_2)$**:** Following equation reformulation, the problem is evaluated with respect to the syntactic form of the problem formulation. The effects of the equation reformulation on the problem, $\mathtt{Effects}_\mathtt{prob}(P_2)$, are that the set of equations is now linear. The system matrix of the linearized system is the Jacobian of the nonlinear system computed at the equilibrium point.

$\mathtt{Evals}_\mathtt{reform}(R_a)$**:** The reformulation is evaluated with respect to the complexity of the reformulation procedure relative to the problem encoding, $\mathtt{Complexity}(\mathtt{Proc}_\mathtt{a}, \mathtt{P}_1)$, which is $O(n^3)$.

$\mathtt{Evals}_\mathtt{prob}(P_3)$**:** Following solution building, the problem is evaluated with respect to whether the stability of the system near an equilibrium point is established, refuted, or remains undetermined. This will depend on the problem. This technique cannot specify the boundaries of the stability region near the equilibrium point. Moreover, sometimes P_3 is not conclusive and one must analyze the effects of neglected higher-order terms.

$\mathtt{Evals}_\mathtt{reform}(R_b)$**:** The solution building procedure computes the eigenvalues of a system matrix and it is evaluated with respect to complexity, $\mathtt{Complexity}(\mathtt{Proc}_\mathtt{b}, \mathtt{P}_2)$, which is $O(n^3)$.

4.3 Solution reformulation: Behavior abstraction for explanation

In [14], Mallory *et al.* propose to summarize the results of the qualitative simulation of a physical system in order to help users recognize "basic patterns of behavior." Their goal is to support human understanding of the solution space. This reformulation is a typical instance of solution reformulation.

Given the user's query and the complete behavior tree of a simulation, generated by QSIM [11], the reformulation procedure summarizes the behavior of the system by generating a behavior graph that retains only those aspects of the behavior tree relevant to the query. The procedure examines the labels of the nodes in the original tree, discards irrelevant information from the labels, and merges adjacent nodes according to a well-defined strategy. The task of inducing patterns of behaviors and producing a higher-level description of the resulting graph is currently entrusted to the user, but the authors plan to extend their work in this direction.

The motivation here is to enhance cognitive insight into the solution space, *i.e.* the formulation. The quality of the formulation is measured by (1) the size of the behavior graph and its tractability with respect to manipulation and understanding by a human user; (2) a user's subjective opinion of the quality of the summary provided by the behavior graph; and (3) soundness and completeness of the behavior graph with respect to the original behavior tree, defined as follows. Soundness: any reformulated behavior corresponds to at least one original behavior. Completeness: all original behaviors are represented in the abstract graph. The reformulation procedure is portrayed in our framework as follows, see Fig. 11.

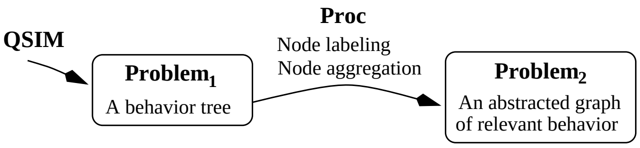

Figure 11: *Solution reformulation.*

Characteristics of the problem:

$\texttt{Form}_1$: A behavior tree representing the qualitative simulation of the behavior of a physical system. Each node of the tree represents a qualitative state of the system.

$\texttt{Form}_2$: A graph whose nodes are either nodes or aggregates of two or more nodes of the original tree.

$\texttt{Query}_1 = \texttt{Query}_2$: Summarize the behavior of a specified subset of quantities/variables in terms of a specified subset of their so-called "methods" (*e.g.*, qualitative values and direction of change).

$\texttt{Assmptn}_1 = \texttt{Assmptn}_2$: None.

Characteristics of reformulation:

`Proc` comprises the following steps: (1) Label each node in the tree with the values and methods of the specified variables. (2) Generate a graph by aggregating nodes that have the same labels and satisfy some adjacency conditions. The authors provide a definition for the adjacency of nodes that guarantees that all significant behaviors in the original behavior tree are reproduced in the behavior graph.

`Cond`: Input must conform to $\texttt{Form}_1$.

Evaluators and effects:

$\texttt{Evals}_{\texttt{prob}}$: The problem is evaluated relative to the solution formulation. As explained above, the following are evaluators for the problem: `Size`, `Understandability`, `Soundness`, and `Completeness`.

The following general observations can be made relative to various differences and effects, $\texttt{Effects}_{\texttt{prob}}$ and $\texttt{Diffs}_{\texttt{prob}}$ on the evaluators. With respect to `Size`, the size of the abstracted-behavior graph is smaller than or equal to that of the original one. With respect to `Soundness` and `Completeness`, the abstracted behavior graph is guaranteed to keep only those states pertinent to the query. and the authors provide a proof of the soundness and correctness of the reformulation procedure. Finally, as an observation with respect to `Understandability`, the authors report that the user may have to experiment with different specifications of the query in order to achieve a satisfactory summary of the behavior of interest.

$\texttt{Evals}_{\texttt{reform}}$: The reformulation is evaluated with respect to the complexity of the solution reformulation procedure relative to the problem encoding. With respect to this problem and reformulation, $\texttt{Complexity}(\texttt{Proc}, \texttt{P}_1)$ is polynomial.

4.4 Discussion

The three examples reported above illustrate the significance of many of the terms introduced by our framework for specifying and evaluating reformulations; more examples will be reported in a forthcoming technical report that illustrate, among other things, the significance of the attribute `Assmptn`.

When reviewing various reformulations techniques from the literature, we encountered some difficulties that we report and analyze here.

It was difficult to distinguish between what should go into the attribute `Form` of a problem encoding from what should appear in `Assmpt`. We decided eventually to include in `Form` the syntax of the formulation as well as the assumptions that are directly apparent from examining `Form` (*e.g.*, in the case of ordinary differential equations, with no gradient terms, we state in `Form` that the model is one of time-varying lumped parameters). Further we chose to use `Assmpt` to express all those conditions under which `Form` is valid but that are not expressed in some fashion in `Form` (*e.g.*, the time scale under which the model is valid). The same difficulty arose for deciding on the content of the attribute `Cond` of the reformulation technique. We elected to include in `Cond` the most general conditions that are, in our understanding, required for applying the procedure, regardless of the particular context in which the procedure is presented in the literature.

Another aspect that was a source of animated debate during our study was the distinction between reformulation techniques and problem-solving engines. Both are computational procedures, but have traditionally been treated as distinct. As we argue in Section 3.1.2, our definition of reformulation as a mapping between two problem encodings does not distinguish between engines and reformulations. Interestingly, this proved to be conceptually elegant, and also convenient for the purpose of presenting a problem-solving process as the execution of a sequence of reformulation techniques. It remains to examine whether more discriminating definitions of reformulations and solution engines are necessary, to identify what these definitions might be, and when the distinction is necessary.

More globally, our framework proved to be convenient for expressing simultaneously all the relevant aspects of a reformulation process, both quantitatively and qualitatively, whereas previous theories addressed either logical properties or computational aspects (see Section 5). Finally, an important feature of our framework is that it allows one to explicitly state the goal of problem solving as well as preferences among alternative strategies, which enables the selection and evaluation of reformulation techniques with respect to this goal. This aspect, although it was previously acknowledged ([24; 3], see also Section 5), has not to date been resolved.

5 Related work

Various theories of reformulation including abstraction and approximation have been proposed in the literature. Some of these theories provide an encompassing high-level characterization. Others restrict their scope to some specific aspect (*e.g.*, cost or faithfulness of results). These theories proved to be essential to our understanding of reformulation, but we found them to be of limited practical use in automating the selection and application of reformulation techniques.

Giunchiglia and Walsh [8] introduce a general theory of abstraction. They introduce a general characterization of reformulation and its properties. Both Cremonini *et al.* [2] and Nayak and Levy [17] explore abstraction theories that are restricted to logical systems and to abstraction techniques that preserve consistency and correctness of proofs. None of these theories make extensive analysis of complexity issues, nor do they provide the terminology for quantitatively evaluating the effects of reformulation. In contrast, the body of research on approximations in the computational complexity community [18], provides rigorous evaluation criteria with respect to cost while neglecting to address issues of expressiveness of representations, which are fundamental in artificial intelligence.

In [24], Weld and Addanki take a task-driven approach to reformulation and adopt Tenenberg's vocabulary [21] for describing the effects of the reformulation on the formulation, only. In [3], Davis studies approximation and abstraction and focuses on the practical application of reformulation techniques applied to reasoning about solid object kinematics. Davis too stresses that the selection of the reformulation technique must be task-driven and in order to satisfy some well-defined criteria. Neither works, however, provides a general framework for reformulation, or identifies attributes for describing and evaluating reformulation techniques.

As a final note, multiple perspectives are commonly sought in automated reasoning to improve performance of the reasoning. We view generation of such perspectives as a reformulation only when the mapping from one perspective to another is well articulated.

6 Conclusions

In this paper we provide a practical framework for characterizing, evaluating, and selecting reformulation techniques, with the long-term goal of automating their selection and application in the context of reasoning about physical systems. While the focus of our research has been on reasoning about physical systems, and hence all our examples are drawn from this domain, the framework developed appears to be applicable to a broad range of tasks and domains.

We identify the three stages of reasoning about physical systems at which interesting reformulations may be performed. However, we do not require that the reasoning transition through all stages, or that it do so sequentially. Our study uncovered two simple, not yet articulated, observations. First, solving engines can naturally be cast as reformulations, eliminating the implicit and poorly defined distinction between reformulations and solving engines. Second, the task of selecting a sequence of reformulations to achieve the goal of problem solving can be cast as a planning problem. Hence, reasoning about physical systems can be reduced to execution of the selected plan.

Our framework provides the terminology to specify and assess the three components in a complex reformulation process (namely, the problem, the reformulation, and the strategy) by providing evaluators of the properties of these components, and comparators of their relative merits. We believe that our framework is sufficiently flexible to accommodate previously proposed properties and metrics for reformulation. We have also collected a variety of evaluators reported in the literature, and structured them according to our framework. The evaluators discussed here are not intended to be exhaustive and will certainly need to be augmented when the framework is extended to comprise reformulation of solution engines or to apply in other types of problem domains.

In an effort to evaluate this framework, we have used it to characterize numerous reformulation techniques, three of which were presented in this paper. This process was not straightforward, especially since we were initially introducing a conceptual distinction between reformulations and solving engines. The current framework is the result of countless iterations over the analysis of the examples, and our perception of problem-solving goals and strategies. Further evaluations still need to be carried out, which we intend to report in a forthcoming technical report.

There are numerous avenues for future work: (1) extend our framework to encompass engine reformulation; (2) assess the usefulness of our framework for tasks other than query answering (*e.g.*, design) and disciplines other than automated reasoning (*e.g.*, cognitive modeling); and (3) study, in more detail and more formally, the composition and inverse mapping of reformulations.

Acknowledgments

The authors are grateful to William Buchanan for various discussions, to Lee S. Brownston for proofreading an early version of this document, and to AAAI anonymous reviewers for constructive comments. B. Y. Choueiry is supported by a fellowship for advanced researchers from the Swiss National Science Foundation. S. McIlraith was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by Xerox Palo Alto Research Center.

References

[1] Daniel J. Clancy and Benjamin Kuipers. Static and Dynamic Abstraction Solves the Problem of Chatter in Qualitative Simulation. In *Proc. of AAAI-97*, pages 118–125, Providence, Rhode Island, 1997.

[2] Roberto Cremonini, Kim Marriott, and Harald Søndergaard. A General Theory of Abstraction. In *Proceedings of the 4th Australian Joint Conference on Artificial Intelligence*, pages 121–134, Australia, 1990.

[3] Ernest Davis. Approximation and Abstraction in Solid Object Kinematics. Technical Report TR706, New York University, New York, NY, 1995.

[4] Brian Falkenhainer and Kenneth D. Forbus. Compositional Modeling: Finding the Right Model for the Job. *Artificial Intelligence*, 51:95–143, 1991.

[5] Adam Farquhar. A Qualitative Physics Compiler. In *Proc. of AAAI-94*, pages 1168–1174, Seattle, WA, 1994.

[6] Kenneth D. Forbus. Qualitatite Process Theory. *Artificial Intelligence*, 24:85–168, 1984.

[7] Kenneth D. Forbus. The Qualitative Process Engine. In Daniel S. Weld and Johan de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 220–235. Morgan Kaufmann, San Mateo, CA, 1990.

[8] Fausto Giunchiglia and Toby Walsh. A Theory of Abstraction. *Artificial Intelligence*, 57:323–389, 1992.

[9] Tom R. Gruber and Pierre O. Gautier. Machine-generated Explanations of Engineering Models: a Compositional Modeling Approach. In *Proc. of the 13 th IJCAI*, pages 1502–1508, Chambéry, France, 1993.

[10] Craig A. Knoblock, Josh D. Tenenberg, and Qiang Yang. Characterizing Abstraction Hierarchies for Planning. In *Proc. of AAAI-91*, pages 692–697, Anaheim, CA, 1991.

[11] Benjamin Kuipers. Qualitative Simulation. *Artificial Intelligence*, 29:289–338, 1986.

[12] Alon Y. Levy, Yumi Iwasaki, and Richard Fikes. Automated Model Selection for Simulation Based on Relevance Reasoning. *Artificial Intelligence*, 96:351–394, 1997.

[13] David G. Luenberger. *Introduction to Dynamic Systems : theory, models, and applications*, chapter Analysis of Nonlinear Systems. Wiley, New York, 1979.

[14] Richard S. Mallory, Bruce W. Porter, and Benjamin J. Kuipers. Comprehending Complex Behavior Graphs through Abstractions. In *Tenth International Workshop on Qualitative Physics. AAAI Technical Report WS-96-01*, pages 137–146, Fallen Leaf Lake, CA, 1996.

[15] P. Pandurang Nayak. Causal approximations. *Artificial Intelligence*, 70:277–334, 1994.

[16] P. Pandurang Nayak and Leo Joskowicz. Efficient Compositional Modeling for Generating Causal Explanations. *Artificial Intelligence*, 83:193–227, 1996.

[17] P. Pandurang Nayak and Alon Y. Levy. A Semantic Theory of Abstractions. In *Proc. of the 14 th IJCAI*, pages 196–203, Montreal, Canada, 1995.

[18] Christos H. Papadimitriou. *Computational Complexity*, chapter Approximation and Complexity, pages 299–328. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.

[19] Jeff Rickel and Bruce Porter. Automated Modeling for Answering Prediction Questions: Selecting the Time Scale and System Boundary. In *Proc. of AAAI-94*, pages 1191–1198, Seattle, WA, 1994.

[20] Peter Struss. On Temporal Abstraction in Qualitative Reasoning (A Preliminary report). In *Proceedings of the Seventh International Workshop on Qualitative Reasoning about Physical Systems*, pages 219–227, Orcas Island, Wa, 1993.

[21] Josh D. Tenenberg. Inheritance in Automated Planning. In *First International Conference on Knowledge Representation and Reasoning*, pages 475–485, Toronto, Canada, 1989.

[22] Daniel S. Weld. Exaggeration. *Artificial Intelligence*, 43:311–368, 1990.

[23] Daniel S. Weld. Reasoning about Model Accuracy. *Artificial Intelligence*, 56:255–300, 1992.

[24] Daniel S. Weld and Sanjaya Addanki. Task-Driven Model Abstraction. In *4th International Workshop on Qualitative Physics*, pages 16–30, Lugano, Switzerland, 1990.