A Customized Logic Paradigm for Reasoning about Models

Reinhard Stolle and Elizabeth Bradley*
University of Colorado at Boulder
Department of Computer Science
Boulder, Colorado 80309-0430
{stolle,lizb}@cs.colorado.edu

Abstract

Modeling is the process of constructing a model of a target system that is suitable for a given task. Typically, in the hierarchy from more-abstract to less-abstract models, the model of choice is the one that is just detailed enough to account for the properties and perspectives of interest for the task at hand. The main goal of the work described here was to design and implement a knowledge representation framework that allows a computer program to reason about physical systems and candidate models (ordinary differential equations, specifically) in such a way as to find the right model at the right abstraction level as quickly as possible.

A key observation about the modeling process is the following. *Not only is the resulting model the least complex of all possible ones, but also the reasoning during model construction takes place at the highest possible level at any time.* Because of this, the knowledge representation framework was designed to allow easy formulation of knowledge and meta knowledge relative to various abstraction levels.

Candidate models are constructed via simple, powerful domain rules. The customized knowledge representation framework is then used to generate new knowledge about the physical system and new knowledge about the candidate model. A candidate model is valid if the facts about the system that is to be modeled are consistent with the facts about the candidate model. Any inconsistency is a reason to discard the candidate model.

The implemented framework is the core of PRET, a program — currently under development — that automates the modeling process.

Introduction

Models are powerful tools that are used to understand physical systems. Abstract models are simple: they account for major properties of the physical system. Less-abstract models are more complicated, allowing them to capture the features of the physical system more accurately and in more detail. The level of abstraction that a model-builder chooses within this hierarchy is typically just detailed enough to account for the properties and perspectives that are useful for the task at hand. This paper describes a framework, based on a logic paradigm, that facilitates reasoning about properties of physical systems and candidate models in such a way as to quickly find the right model in the abstraction hierarchy.

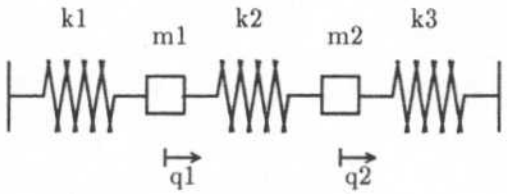

Figure 1: "Three Springs" example.

The task of finding a model that matches the observed behavior of a target system is often called *system identification* (Astrom & Eykhoff 1971; Ljung 1987). The first stage of this task, *structural identification*, identifies the form of the model or skeleton of the equation, such as $a\ddot{\theta} + b \sin\theta = 0$ for a simple pendulum. In the second system identification stage, *parameter estimation*, the parameter values a and b are determined.

PRET is a program that is an attempt to automate a useful part of the system identification process; its goal is to find a system of ordinary differential equations (ODEs) that models a given physical system. Inputs are *observations* about that system, user-supplied *hypotheses* about the desired model, and *specifications*. Observations are measured automatically by sensors or interpreted by the user, symbolically or numerically, in varying formats and degrees of precision. Hypotheses about the physics involved are supplied by the user; these may conflict and need not be mutually exclusive, whereas observations are always held to be true. Finally, specifications indicate the quantities of interest and their resolutions. Figure 1 shows a physical system that consists of two masses and three springs and

*Supported by NSF NYI #CCR-9357740, NSF MIP #9403223, and a Packard Fellowship in Science and Engineering from the David and Lucile Packard Foundation.

```
(find-model
  (domain mechanics)
  (state-variables <q1> <q2>)
  (point-coordinates <q1> <q2>)
  (hypotheses
    (<force> (* m1 (deriv (deriv <q1>))))
    (<force> (* m2 (deriv (deriv <q2>))))
    (<force> (* k1 <q1>))
    (<force> (* k2 (- <q1> <q2>)))
    (<force> (* k3 <q2>))
    (<force> (* r1 (deriv <q1>)))
    (<force> (* r2 (square (deriv <q1>))))
    (<force> (* r3 (deriv <q2>)))
    (<force> (* r4 (square (deriv <q2>)))))
  (observations
    (autonomous <q1>)
    (autonomous <q2>)
    (damped-oscillation <q1>)
    (damped-oscillation <q2>)
    (numeric
      (<time> <q1> <q2>)
      ((0 .1 .1) (.1 .1099 .1103) ... )))
  (specifications
    (resolution <time> absolute 1e-6 (0 120))
    (resolution <q1> absolute 1e-3 (0 1))
    (resolution <q2> absolute 1e-3 (0 1))))
```

Figure 2: Instructing PRET to model the physical system of Figure 1.

Figure 2 exemplifies how one might instruct PRET to construct a model of that system. In this example, the user first sets up the problem, then hypothesizes nine different force terms, makes five observations about the position coordinates q_1 and q_2, and finally specifies resolutions and ranges for three important variables. It is important to note that this simple example is not by any means a true indication of PRET's power.

Candidate models are generated by combining the user's hypotheses into ODEs. These candidate models are checked against the observations about the physical system. PRET incorporates two types of knowledge:

- *ODE rules* that formulate mathematical precepts that are true for all ODEs, such as the definition of an equilibrium point, and
- *domain rules* that apply in individual domains, such as Kirchhoff's voltage law for electronic circuits.

Domain rules and ODE rules play different roles; domain rules are used to combine hypotheses into models, while ODE rules are used to infer facts from models and from observations.

The high-level control flow of PRET can be described as a variant of "generate-and-test." While many modeling programs derive the model from the observations about the physical system, our approach generates candidate models independently of the observations. It is only in the "test" part of the "generate-and-test" cycle where the observed behavior of the physical system currently plays a role.[1] The behavior of each candidate model is compared to the observed behavior of the physical system; discrepancies between a model and the observations cause that model to be ruled out. The first model in this generate-and-test sequence that is consistent with the observations is currently returned as the result.

This paper focuses on the "test" part of the "generate-and-test" cycle, which checks generated candidate models against the observations about the physical system. We describe the basic paradigm of the reasoning framework used in this "model checker" and discuss how this framework can be expanded to incorporate additional reasoning mechanisms.

The next section outlines the major features of the modeling program PRET. For a more elaborate discussion consult (Bradley & Stolle 1996). Then we describe how the paradigm "valid if not proven invalid" is able to combine several heterogeneous mechanisms into a powerful framework that is well suited for reasoning about models of physical systems. After describing the current status of the reasoning framework, we illustrate the reasoning process with an example. Finally, we briefly review related literature and summarize the paper.

Structure of PRET

Generating Candidate Models

PRET generates candidate models by combining user hypotheses into ODEs of the form $f(\vec{x}, t) = 0$. Hypotheses are ODE fragments that contain special keywords that provide links to the domain and ODE rules. In the example shown in Figure 2, PRET uses the domain rule `(point-sum <force> 0)` about the keyword `<force>` to generate all different combinations of the nine force hypotheses. The current implementation first tries models that consist of only one term. If all one-term models fail, two-term models are tried, and so on.[2]

A human expert quickly assesses the important features of a physical system and translates these into model fragments. This is akin to the "generate" phase of PRET's generate-and-test cycle. Currently, there is no high-level reasoning involved in this process: models are simply generated in strict order of increasing complexity as defined by a simple metric: the complexity of a model fragment is the number of operators in its symbolic representation. This order is a handle that can be used to make the program smarter and we are investigating how best to use it.

[1] Future incarnations of the program will simplify and/or refine the model returned by the generate-and-test cycle. The simplifier and refiner modules will also be guided by the observations.

[2] We are working on more sophisticated reasoning about what models to try next.

If the system has more than one degree of freedom — as is the case in the example in Figure 2 — every point coordinate (degree of freedom) is a potential source of a force-balance equation, but all coordinates may not play roles in the model (e.g., if k_1 is so large to effectively fix q_1). To account for this possibility, PRET first tries to use only one point coordinate. If all models that result from single-point force balances fail, the program generates candidate models by using two force-balance equations, and so on.

If PRET cannot find an appropriate model in this sequence of hypothesis combinations, it admits failure and asks the user to supply additional hypotheses. We are developing power-series techniques for automatic term synthesis to be invoked if the user's list of hypotheses is exhausted before a valid model has been found.

Note that it is easy to extend the program to other domains and paradigms (e.g., fluid mechanics, `volume-change`; electronics, `loop-sum`; etc.). One need only think of a new keyword and code a new rule that uses it, much as `(point-sum <force> 0)` couples to hypotheses that include the `<force>` keyword. Finally, the concepts of loop and point sums are not only appropriate for this example, but also generalizable well beyond mechanics.

The next subsection gives an overview of how a candidate model is checked against the set of observations about the physical system. The rest of the paper then describes this reasoning process in more detail.

Checking Models Against Observations

Observations about a target system have two potential sources — the user and the sensors — and may be *descriptive*, *graphical*, or *numeric*. The former use special descriptive keywords, the second are sketches drawn on a computer screen with a mouse, and the third simply specify data points. Only the first and the last are currently implemented. Keywords in descriptive observations (`damped-oscillation`, `linear with [slope, intercept]`, `chaotic`, `autonomous`, etc.) resemble terms in qualitative physics.

PRET's goal is to find a model that accounts for all given observations. The test part of the generate-and-test loop around which the program is built uses a logic-based knowledge representation framework to infer facts about the physical system and about the current candidate model. Some of the ODE rules that play roles in the inference process trigger calls to a number of Scheme[3] functions that identify mathematical properties of the current candidate model, e.g., "the ODE is linear in x." Some of these functions in turn make use of other tools. The current implementation of PRET uses symbolic algebra facilities from the commercial package Maple (Char *et al.* 1991), calls upon the public-domain package ODRPACK (Boggs *et al.* 1989) for parameter estimation, and will soon incorporate qualitative simulation (Kuipers 1986).

The collection of Scheme "model observer" functions that identifies mathematical properties of the current candidate model essentially implements the basic operations found in any mathematics text; some representative examples are shown in Figure 3. Mathematical properties of both the model and the physical system are represented declaratively. Henceforth, the two sets of properties are called *knowledge* about the model and the physical system, respectively. A logic system expands both sets of properties, inferring new properties from known ones by applying domain-independent ODE rules; for example, if the system is known to be autonomous, its model cannot explicitly contain the variable `<time>`. If the knowledge about the ODE model does not conflict with the knowledge about the physical system, the model passes the check. Otherwise, it fails.

Figure 4 gives some examples of descriptive observations and the facts that the logic system infers from them. This is only a small sampling of rules. Many other possibilities exist; because of the logic system's structure, implementing them is only a matter of a few lines of Scheme code and/or a call to Maple. This is one of the advantages of the declarative reasoning framework: an expert user of PRET can easily extend the ODE theory represented in the knowledge base and specialize it for typical applications.

In order to keep the architecture of the program modular, in order to keep the paradigm uniform, and in order to not replicate information, we intend to keep the process that generates candidate models as simple as possible and incorporate all of the knowledge that makes the modeling process intelligent in the process that checks candidate models against observations. An example of an intelligent reasoning policy is the following: if it is possible to rule out a model by purely *qualitative* techniques, one should not even attempt to match the model to the observations *numerically*.

A model constructed by a human expert matches, *minimally*, a particular set of observations; the model builder does no more work than is necessary to effect the match, and does not try to anticipate extensions or further developments until forced to do so by model failure or requirement escalation. PRET does exactly the same thing, expending the least possible effort to corroborate models and observations by using the highest possible level of information and reasoning at all times.

Verification of a *valid* candidate model with respect to a numeric observation requires point-by-point comparison with a numeric integration of the ODE. However, *invalid* models can often be ruled out without performing such an expensive numeric comparison. Many inappropriate models can be discarded by reasoning about purely qualitative information that can be inferred from numeric information. In order to

[3] PRET is written in Scheme.

ODE property	Method	Consequences
linearity	Maple/`jacobian`	model is linear model is nonlinear
time dependence	Scheme/symbolic	model is function of time model is not a function of time
divergence	Maple/mixed	algebraic equation for divergence
order	Scheme/symbolic	order of model is n
roots	Maple/`eigenvals`	algebraic equation for roots

Figure 3: Some observer functions that infer facts from models.

Observation about state variable x_i	Implications for the model $f(\vec{x},t)=0$
autonomous	cannot explicitly contain t (i.e., $f(\vec{x})=0$)
chaotic	cannot be linear
chaotic and autonomous	order >2
oscillation and autonomous	imaginary part of one pair of roots >0
oscillation and autonomous	order ≥ 2
linear	should satisfy $\ddot{x}_i=0$
constant	should satisfy $\dot{x}_i=0$
conservative	$\nabla \cdot f=0$
damped oscillation and autonomous	$\nabla \cdot f<0$
growing oscillation and autonomous	$\nabla \cdot f>0$

Figure 4: Some observations and the corresponding inferences drawn by the logic system.

achieve this behavior, PRET preprocesses the observations — curve fitting, recognition of linear regions, and so on — using Maple functions and simple phase-portrait analysis techniques (Bradley 1995), both of which yield high-level results that can then be used much as qualitative observations.

The next section describes, in more detail, the paradigm that underlies the reasoning framework that is used to check candidate models against observations.

The Reasoning Paradigm: Valid if not Proven Invalid

A *programming style* is the way one states the problem that a program is supposed to solve, the way one thinks about it, and the way one formulates the solution. Programming styles are tailored to certain application domains and are usually enhanced by programming languages that support that style. AI researchers often go one step further and design frameworks (*paradigms*) that are tailored to certain classes of problems. The more theoretical and the more general a paradigm is, the more uniform, clean and understandable it is. The closer a paradigm is to an actual problem, on the other hand, the bigger, less uniform, and less clean it is. As an example, consider the difference between pure first-order logic and the programming language PROLOG; the "predicates" *assert* and *retract*, for example, and most importantly the *cut* are supposed to expedite programming, but they certainly have no equivalent in first-order logic.

Applications require compromises and exceptions that complicate system implementation and maintenance. This is especially true in our case, where the application integrates many techniques at many abstraction levels. For our application, pure first-order logic is too neat because abstraction levels are hard to model in first-order logic. Formulae are either true or false in pure logic, but in our application facts can be true with respect to one abstraction level, but false with respect to another. On the other hand, we needed some sort of declarative knowledge representation framework because PRET's reasoning skills are too complex to be controlled operationally. One of the goals of the work described in this paper was to find the right balance of uniformity and practicability of the paradigm.

PRET incorporates two types of knowledge. For each of these types, we had to find a suitable knowledge representation framework and a useful control regime that processes the represented knowledge. The first type of knowledge is the set of domain rules that combine hypotheses into candidate models. These rules, such as `(point-sum <force> 0)`, are used in the model generation process, which is not the main focus of this paper. Here we concentrate on the second type of knowledge, which is used to check models against observations. The next few paragraphs give an overview of the asssociated knowledge representation framework and control regime.

Examples of the second type of knowledge are the

ODE rules

```
(2 (<- (constant (var Variable))
       ((degrees-of-freedom 1)
        (autonomous (var Variable))
        (order (var Variable) 0))))
(1 (<- falsum
       ((constant (var Variable))
        (not-constant (var Variable))))).
```

A rule consists of a number and an implication. The number indicates the level of abstraction of the rule: the lower the *abstraction level number*, the more abstract the rule. The implication is a Horn clause of the form (<- *head body*). The first example above has abstraction level two and expresses that a state variable x_i is constant if the physical system has one degree of freedom, the system is autonomous with respect to x_i, and x_i does not appear in the ODE (i.e., its order is zero). The second example simply states that no state variable can be constant and non-constant at the same time.

ODE rules encoded in this form are used when candidate models are checked against observations according to the paradigm "Valid if not Proven Invalid." This paradigm corresponds to the minimalistic approach a human modeler takes: if there is no indication that the model is wrong, we assume it is right. This resembles the *closed world assumption* that underlies many logic systems. Thus, the program returns a model that cannot be proved invalid. Models are ruled out if and only if a contradiction exists between a mathematical property of the physical system (e.g., (**oscillation <x>**)) and a mathematical property of the model (e.g., (**no-oscillation <x>**)).

Figure 5 illustrates the reasoning process involved in establishing contradictions between a candidate model and the physical system. The observations about the target system form the initial knowledge about the system to be modeled. Additional knowledge is generated from these observations by applying ODE rules (like those in Figure 4) to known properties. A similar "knowledge growth" process is also performed for the model. Various Scheme functions (see Figure 3) are applied to the model, yielding an initial set of properties. These model properties play the role of "observations" about the behavior of the model, which are then compared to the behavior of the target system. As with the body of knowledge inferred from the observations, additional knowledge is obtained by applying ODE rules to these model properties.

The current implementation uses breadth-first forward reasoning with a user-specified maximum inference depth. As previously mentioned, our goal was to let the program reason at the highest possible abstraction level at all times; typically, the higher the level of reasoning, the faster it performs the check. In order to accomplish this, dominant mathematical properties must be used first, more subtle features later. The abstraction level numbers are used to approximate this "high-level-first" control flow. The reasoning proceeds to a less-abstract level only if the inference engine has not been able to prove a contradiction using only the rules that are more abstract.

In this abstraction hierarchy, ODE rules that concern numeric information are considered "concrete" whereas ODE rules that deal with qualitative information are considered "abstract." We are working on an intermediate level that uses constraint logic programming techniques (Cohen 1990) in order to reason about ranges (instead of symbols or single numbers). For example, if a physical system oscillates, the imaginary parts of at least one pair of its roots must be nonzero. Thus, if the model $A_1\ddot{x} + A_2\dot{x} + A_3x = 0$ is to match an **oscillation** observation, the coefficients have to satisfy the inequality $4A_1A_3 > A_2^2$.

The syntax in which properties and rules are represented is similar to first-order logic — the "descriptive keywords" in the observations are first-order logic predicates.[4] A mathematical property of the physical system or the candidate model is a list that represents an atomic formula (*predicate* arg_1 arg_2 ... arg_n).

The special atomic formula **falsum** may only appear as the head of an implication. Such an *integrity constraint* (<- **falsum** ($goal_1$ $goal_2$ ... $goal_m$)) asserts that the conjunction of $goal_1, \ldots, goal_m$ is inconsistent. This concept of *negation as inconsistency* (Gabbay & Sergot 1986) is the only form of negation in our paradigm. Negation as failure, for example, which is the standard form of negation in PROLOG, is particularly undesirable for our purposes. Since we do not require the user to supply all possible[5] observations, the absence of knowledge cannot be used to generate new knowledge.

Variables in the formulae have the form (**var** *symbol*); they are (implicitly) universally quantified. A new property can be inferred by a rule if the body of the rule can be instantiated with known properties. Since this prototype of the program emphasizes correctness over efficiency, the unification algorithm includes the occurs check.

Note that knowledge about the physical system is global, whereas knowledge about a candidate model is local to that model. Every time a model is ruled out because of a contradiction and a new candidate model is chosen, the Scheme functions that determine the initial model knowledge must be applied anew. In order to facilitate reuse of the knowledge that has been inferred about the physical system, every formula carries a tag that is either **from-observations** or **from-both**.

[4] We could have decided to implement the reasoning framework as a meta interpreter in PROLOG, for example. However, because of PRET's heterogeneous nature the choice of any programming language would have advantages and disadvantages. We chose to implement the reasoning framework in Scheme in order to facilitate interaction with other parts of PRET.

[5] Here, *possible* means *expressible with the implemented observation vocabulary.*

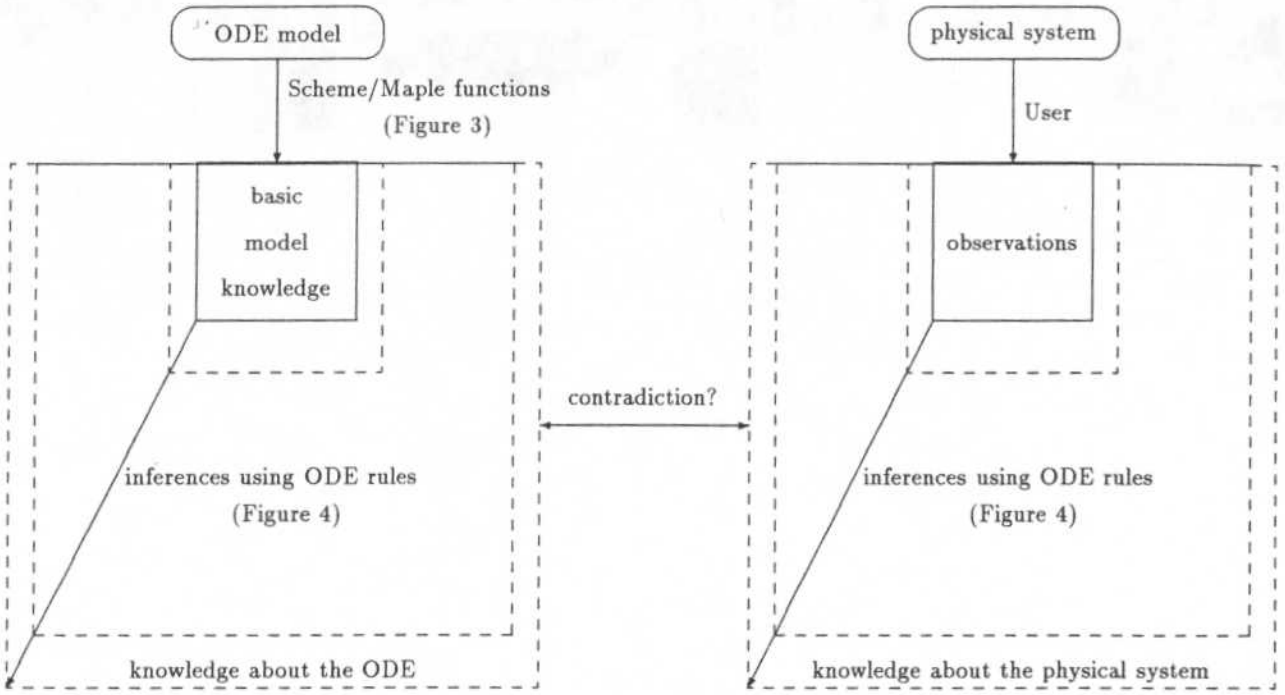

Figure 5: A candidate model is valid if there is no contradiction between the knowledge about the model and the knowledge about the physical system.

The former indicates that the formula is independent of the current candidate model — it is a mathematical property of the target system and thus can be reused. The latter tag indicates that knowledge about the current candidate model has been used during the derivation of the formula. These formulae cannot be reused. More precisely,

- observations about the physical system are tagged `from-observations`;
- the set of initial mathematical properties of the current candidate model that are established by various Scheme functions are tagged `from-both`;
- an instantiation of *head* that is derived by the implication (`<-` *head* (*goal*$_1$ ... *goal*$_m$)) is tagged `from-observations` if all used instantiations of the goals *goal*$_1$, ..., *goal*$_m$ are tagged `from-observations`; otherwise, it is tagged `from-both`.

To see how the Scheme "model observer" functions (Figure 3) provide links between the ODE model and the logic inference system, consider the following clauses

```
(<- (no-oscillation (var X))
    ((scheme-eval (all-roots-real?
                   (var X) current-model))))
(<- (linear (var X))
    ((scheme-eval (linear-system?
                   (var X) current-model))))
```

The special constant `current-model` refers to the current candidate model. This allows the current model to be passed as an argument to Scheme functions that are invoked by the special predicate `scheme-eval`. A `scheme-eval` goal fails if the corresponding Scheme function call returns the symbol `fail`. Otherwise, it returns a list of substitutions for the variables in the goal. In a manner analogous to `current-model`, the special constants `current-state-variables` and `current-specifications` refer to the list of state variables and the list of specifications, respectively.[6]

As an example, consider an ODE model that is linear in the state variable `<q>`. Application of the Scheme function `linear-system?` will establish the model property

```
(from-both (linear <q>)).
```

If there is an observation about the physical system that implies the fact

```
(from-observations (chaotic <q>)),
```

application of the rule that captures the mathematical relationship between nonlinearity and chaos:

```
(<- (non-linear (var X))
    ((chaotic (var X))))
```

will yield the physical system property:

```
(from-observations (non-linear <q>)).
```

[6] In the current implementation the values of `current-state-variables` and `current-specifications` do not change. Therefore, the prefix "current-" may appear to be misleading. However, later versions of PRET will use techniques that can discover new state variables or create additional specifications "on the fly."

Then, by applying the rule

```
(<- falsum
    ((non-linear (var X)) (linear (var X))))
```

the inference system will detect an inconsistency in the union of the set of properties of the physical system and the set of properties of the candidate model. In this case, the derived **falsum** is tagged **from-both**. In the case where the **falsum** can be derived from observations alone, PRET aborts and informs the user that the supplied observations are inconsistent.[7]

Currently, the search for a contradiction performs breadth-first forward reasoning: the knowledge about both the physical system and the candidate model grows step by step. First, only the most abstract ODE rules (abstraction level number 1) are used. In each step, all ODE rules of this abstraction level are applied *once* to all known properties. This step is repeated until the **falsum** has been derived or the maximum inference depth has been reached. If this depth has been reached without derivation of the **falsum**, the inference engine increments the abstraction level number and repeats the process. If this algorithm reaches the maximum inference depth of the least abstract level, the candidate model passes the check; otherwise, it fails.

Breadth-first forward reasoning was chosen for two primary reasons. First, it avoids infinite search paths. Second — and of arguable importance — forward reasoning seems to emulate the reasoning of a human expert more closely than does backward reasoning. The next version of the inference system will use SLD resolution, a form of backward reasoning that has the advantage of being goal-directed. SLD resolution theorem provers act on a "current set of literals." The *control strategy* of a SLD resolution theorem prover is defined by the function that selects the literal that is resolved and by the function that chooses the resolving clause. PRET will provide meta-level language constructs that allow the implementer of the ODE theory knowledge base to specify the control strategy that is to be used (Gallaire & Lasserre 1979; 1982; Beckstein, Stolle, & Tobermann 1995), such as the formula (**ruleorder** H (N_1 ... N_n)) which specifies the order in which ODE rules must be applied to resolve a certain subgoal of the current goal. Here, H specifies the head of rules for which this control rule applies and N_i are names of ODE rules. For example, a useful meta-level control rule might specify that an ODE rule that symbolically determines the order of a model must be applied before an ODE rule that triggers the numeric point-to-point comparison of a numeric observation and a numerical integration of the candidate ODE. An example of a construct that allows specification of the order in which subgoals are resolved is (**before** G_1 G_2) where G_i are subgoals. For example, a control rule can specify that the inference engine must attempt proofs for formulae with the predicate **order** before it tries to prove formulae with the predicate **oscillation**. The intuition here is, again, that the search should be guided towards a cheap and quick proof of a contradiction. Therefore, control predicates will typically be used to force cheap tests to be performed before expensive tests and test that are likely to lead to a contradiction to be performed before others. Finding such good heuristics is a non-trivial task.

Abstraction levels statically express control information. An ODE rule with an abstraction level number n is used before ODE rules with abstraction level numbers that are greater than n. The theorem prover proceeds to a higher abstraction level number only if the attempt to prove the **falsum** with ODE rules with lower abstraction level numbers fails. We are working on replacing the abstraction level number by a more meaningful *qualitative categorization*. Given that the underlying inference system (resolution with negation as inconsistency) is logically complete,[8] and given that the implementation of control rules preserves completeness, the resulting reasoning process is complete relative to the set of ODE rules that are in use. However, reasoning performed at abstraction level number n is typically incomplete with respect to an abstraction level number greater than n. That means that a model that fails at a detailed level might pass the check at a more abstract level. This is exactly the intended behavior of the program.

As described earlier, facts that are tagged **from-observations** can be reused when another model is tried. The standard logic programming tool for reuse of knowledge that has already been proved is called *truth maintenance* or *reason maintenance*. We are incorporating an assumption-based truth maintenance system (ATMS) (de Kleer 1986) in order to generalize the tagging approach. Rejecting a candidate model and considering another candidate model then simply amounts to a context switch of the ATMS. Furthermore, truth maintenance systems can generate *explanations* for facts that can be derived in the current context. In our application, an explanation for the formula **falsum** can be interpreted as the reason for failure of the current candidate model. This explanation can be passed to the candidate model generator, which may then use this information in order to decide what model to try next.

An Example

Figure 6 shows a sample call of PRET on the "Three

[7] It is also possible that the set of ODE rules is inconsistent. However, PRET assumes that this is not the case because an inconsistent ODE theory is useless.

[8] SLD resolution is not necessarily *operationally* complete. Logical completeness means that every logical consequence of the logic program can be proved under *some* control regime. Operational completeness means that every logical consequence of the logic program can be proved under the control regime *that is currently in use*.

```
(find-model
  (state-variables <q1> <q2>)
  (point-coordinates <q1> <q2>)
  (hypotheses
    (<force> (* k1 <q1>))
    (<force> (* k2 (- <q1> <q2>)))
    (<force> (* k3 <q2>))
    (<force> (* m1 (deriv (deriv <q1>))))
    (<force> (* m2 (deriv (deriv <q2>)))))
  (observations
    (autonomous <q1>)
    (autonomous <q2>)
    (oscillation <q1>)
    (oscillation <q2>)
    (numeric (<time> <q1> <q2>) (eval *data*)))
  (specifications
    (resolution <q1> absolute 0.1)
    (resolution <q2> absolute 0.1)))
```

Figure 6: A sample call of PRET on the "Three Springs" problem.

Springs" problem of Figure 1. The keyword `eval` in the numeric observation causes the variable `*data*` to be evaluated in the calling environment. Bound to this variable is a time series that was generated by Runge-Kutta integration of the system

$$\begin{aligned} \dot{q_1} &= p_1 \\ \dot{q_2} &= p_2 \\ \dot{p_1} &= -0.1\,q_1 - 0.2\,(q_1 - q_2) \\ \dot{p_2} &= 0.2\,(q_1 - q_2) - 0.3\,q_2. \end{aligned}$$

Figure 7 illustrates the resulting modeling process. PRET examines combinations of hypotheses in order of increasing complexity. The first candidate model tried is $k_1 q_1 = 0$. A Scheme function called on the ODE establishes the fact `(order <q1> 0)` which expresses that the order of the highest derivative of q_1 in this model is zero. This fact conflicts with facts inferred from the observation `(oscillation <q1>)`, so this model is ruled out. The way PRET handles this first candidate model demonstrates the advantage of its abstract-reasoning-first approach; only a few steps of inexpensive qualitative reasoning suffice to quickly discard the model.

PRET tries all combinations of `<force>` hypotheses at single point coordinates, but all these models are ruled out for qualitative or numeric reasons. It then proceeds with ODE systems that consist of *two* force balances — one for each point coordinate. One example of a candidate model of this type is

$$\begin{aligned} k_1 q_1 + m_1 \ddot{q_1} &= 0 \\ m_2 \ddot{q_2} &= 0 \end{aligned}$$

None of the implemented rules discards this model by purely qualitative means. PRET transforms these two second-order ODEs into a system of four first-order ODEs and calls the ODRPACK parameter estimator on the system. The parameter estimator finds no appropriate values for the coefficients k_1, m_1, and m_2 such that the ODE solution matches the numeric time series. Therefore, this candidate model is also ruled out.

After having discarded a variety of unsuccessful candidate models in a similar manner, PRET tries the model

$$\begin{aligned} k_1 q_1 + k_2 (q_1 - q_2) + m_1 \ddot{q_1} &= 0 \\ k_3 q_2 + k_2 (q_1 - q_2) + m_2 \ddot{q_2} &= 0 \end{aligned}$$

Again, it transforms the ODEs and calls the parameter estimator, this time successfully. It then substitutes the returned parameter values for the constants k_1, k_2, k_3, m_1, and m_2 and integrates the resulting ODE system with fourth-order Runge-Kutta, comparing the result to the numeric time-series observation.[9] The difference between the integration and the observation stays within the specified resolution, so the numeric comparison yields no contradiction and this candidate model is returned as the answer.

We have chosen this simple, clear, and obvious example only to illustrate PRET's operation. Linear systems of this type are easy to model; no engineer would use a software tool to do generate-and-test model generation and guided search to find an ODE model of a system so simple and well-understood. This example is representative neither of the power of the program nor of its intended target applications; PRET's true targets are nonlinear, high-dimensional black-box systems. Modeling these types of systems is where PRET's mixture of exact and approximate techniques, quantitative and qualitative reasoning, and precise and heuristic knowledge will become truly powerful.

Related Work

PRET's techniques fall mostly in the category of *qualitative physics* or *qualitative reasoning* (QR) (Weld & de Kleer 1990; de Kleer & Williams 1991; Faltings & Struss 1992). In general, *modeling* (Falkenhainer & Forbus 1991; Kuipers 1993) underlies most approaches to reasoning about physical systems. Strictly speaking, every formalization of the properties of a physical system constitutes a *model* of that system. The spectrum ranges from models and tools that use a language that is very close to the physics of the system (e.g., QPT/QPE (Forbus 1984; 1990)) to models that use a language that is well suited to describe the system mathematically (e.g., ODEs). QSIM (Kuipers 1986) is a qualitative realization of the mathematics end of this spectrum. PRET resides somewhere in the middle. Its inputs are partially expressed in terms of physics and its reasoning uses concepts from physics. However, its output — the model of the physical system that it constructs — is purely mathematical: an ODE.

[9] The discrete data points that are generated by the numeric integration are connected by splines.

```
Trying model (model ((= (+ (* k1 <q1>)) 0))).
Contradiction established by rule (<- falsum ((oscillation (var x)) (no-oscillation (var x)))).
[...]
Trying model (model ((= (+ (* k1 <q1>) (* m1 (deriv (deriv <q1>)))) 0)
                     (= (+ (* m2 (deriv (deriv <q2>)))) 0))).
Checking first-order system
((= (deriv d<q1>) (/ (- 0 (* (const k1) <q1>)) (const m1)))
 (= (deriv <q1>) d<q1>)
 (= (deriv d<q2>) (/ (+ 0) (const m2)))
 (= (deriv <q2>) d<q2>)) numerically.
Could not find coefficients for system.
Contradiction established by rule (<- falsum ((numeric-comparison (fail (var reason)))))
[...]
Trying model (model ((= (+ (* k1 <q1>) (* k2 (- <q1> <q2>)) (* m1 (deriv (deriv <q1>)))) 0)
                     (= (+ (* k3 <q2>) (* k2 (- <q1> <q2>)) (* m2 (deriv (deriv <q2>)))) 0))).
Checking first-order system
((= (deriv d<q1>) (/ (- 0 (* (const k1) <q1>) (* (const k2) (- <q1> <q2>))) (const m1)))
 (= (deriv <q1>) d<q1>)
 (= (deriv d<q2>) (/ (- 0 (* (const k3) <q2>) (* (const k2) (- <q1> <q2>))) (const m2)))
 (= (deriv <q2>) d<q2>)) numerically.
((model ((= (+ (* k1 <q1>) (* k2 (- <q1> <q2>)) (* m1 (deriv (deriv <q1>)))) 0)
         (= (+ (* k3 <q2>) (* k2 (- <q1> <q2>)) (* m2 (deriv (deriv <q2>)))) 0)))
 (((const k1) .10000007) ((const k2) .20000001) ((const k3) -.29999989)
  ((const m1) 1.)        ((const m2) -1.)))
```

Figure 7: The transcript generated by the sample call of Figure 6.

PRET aims to integrate quantitative and qualitative information. Many good papers have reported work in this extremely active research area among which are (Forbus & Falkenhainer 1990; Williams 1991; Berleant & Kuipers 1992; Farquhar & Brajnik 1994; Gao & Durrant-Whyte 1994; Zhao 1994; Kleiber & Kulpa 1995; Vescovi, Farquhar, & Iwasaki 1995).

Truth maintenance systems were already being used in some of the earliest QR/modeling work (Falkenhainer & Forbus 1991). Their purpose there was slightly different, however, from their function in our approach. Context switches in (Falkenhainer & Forbus 1991) amount to choices of model fragments, whereas context switches in our approach occur when a new candidate model is checked. The *graph-of-models* approach (Addanki, Cremonini, & Penberthy 1991) is similar in many regards to (Falkenhainer & Forbus 1991), but represents the space of possible models as a directed graph of models where edges between nodes (models) are approximations.

Using reasons for failure to improve the model is called *discrepancy-driven refinement*. Algorithmic debugging of logic programs (Shapiro 1983) detects discrepancies between a logic program and its intended interpretation. Recently, model-based diagnosis has been based on these ideas (Console, Friedrich, & Dupré 1993).

Summary

PRET is a program that automates a useful part of the system identification process: modeling physical systems with ordinary differential equations. Our focus is not to construct a cognitive model of the thought process of an engineer when he or she builds a model of a physical system, but rather to build a useful tool that obtains the same result as a human expert would.

In this paper, we described PRET's reasoning framework, which uses ODE rules to check candidate models against observations about a physical system. This framework is well suited to the representation of knowledge about physical systems and their ODE models with respect to various abstraction levels, and it allows smooth transition between these abstraction levels. The framework also supports heterogeneity: various very different existing system identification tools and techniques have been smoothly integrated.

PRET's use of qualitative techniques proves particularly powerful in the structural stage of the system identification process. Parameter estimation required integration of lower-level numerical techniques which also raised some interesting AI issues: how to integrate these techniques, how to interact with them, and how to interpret their results on the structural level. Like any numerical integrator, Runge-Kutta has advantages and disadvantages in particular problems. Automating the choice of a numerical integrator and its parameters raises similar AI issues as the integration of the parameter estimator. Our answers to these questions will be the topic of a later paper.

Experimentation with the current implementation of the reasoning framework — which includes about a dozen ODE theory rules — shows that the paradigm "valid if not proven invalid" is well suited for the modeling task.

Acknowledgements

We thank Janet Rogers of NIST for making ODRPACK available and for instructions on its use. We would also like to thank the referees for helpful comments.

References

Addanki, S.; Cremonini, R.; and Penberthy, J. S. 1991. Graphs of models. *Artificial Intelligence* 51:145–178.

Astrom, K. J., and Eykhoff, P. 1971. System identification — a survey. *Automatica* 7:123–167.

Beckstein, C.; Stolle, R.; and Tobermann, G. 1995. Declarative meta level control for logic programs. In *Proceedings of the 1. Russian-German Symposium on Intelligent Information Technologies in Decision Making.*

Berleant, D., and Kuipers, B. J. 1992. Combined qualitative and numerical simulation with Q3. In Faltings, B., and Struss, P., eds., *Recent Advances in Qualitative Physics.* Cambridge MA: MIT Press.

Boggs, P. T.; Donaldson, J. R.; Byrd, R. H.; and Schnabel, R. B. 1989. Algorithm 676 — ODRPACK: Software for orthogonal distance regression. *ACM Transactions on Mathematical Software* 15:348–364.

Bradley, E., and Stolle, R. 1996. Automatic construction of accurate models of physical systems. *Annals of Mathematics and Artificial Intelligence.* Forthcoming.

Bradley, E. 1995. Autonomous exploration and control of chaotic systems. *Cybernetics and Systems* 26:299–319.

Char, B. W.; Geddes, K. O.; Gonnet, G. H.; Leong, B. L.; Monagan, M. B.; and Watt, S. M. 1991. *Maple V Language Reference Manual.* New York: Springer.

Cohen, J. 1990. Constraint logic programming languages. *Communications of the ACM* 33(7):52–68.

Console, L.; Friedrich, G.; and Dupré, D. T. 1993. Model-based diagnosis meets error diagnosis in logic programs. In *Proceedings IJCAI-93,* 1494–1499.

de Kleer, J., and Williams, B. C., eds. 1991. *Artificial Intelligence,* volume 51. Special Volume on Qualitative Reasoning About Physical Systems II.

de Kleer, J. 1986. An assumption-based TMS. *Artificial Intelligence* 28(2).

Falkenhainer, B., and Forbus, K. D. 1991. Compositional modeling: Finding the right model for the job. *Artificial Intelligence* 51:95–143.

Faltings, B., and Struss, P., eds. 1992. *Recent Advances in Qualitative Physics.* Cambridge MA: MIT Press.

Farquhar, A., and Brajnik, G. 1994. A semi-quantitative physics compiler. In *Proceedings of the International Workshop on Qualitative Reasoning about Physical Systems.* Nara, Japan.

Forbus, K. D., and Falkenhainer, B. 1990. Self-explanatory simulations: an integration of qualitative and quantitative knowledge. In *Proceedings AAAI-90,* 380–387.

Forbus, K. D. 1984. Qualitative process theory. *Artificial Intelligence* 24:85–168.

Forbus, K. D. 1990. The qualitative process engine. In Weld, D. S., and de Kleer, J., eds., *Readings in Qualitative Reasoning About Physical Systems.* San Mateo CA: Morgan Kaufmann.

Gabbay, D. M., and Sergot, M. J. 1986. Negation as inconsistency I. *The Journal of Logic Programming* 3(1):1–36.

Gallaire, H., and Lasserre, C. 1979. Controlling knowledge deduction in a declarative approach. In *Proceedings IJCAI-79,* S1–S6.

Gallaire, H., and Lasserre, C. 1982. Metalevel control for logic programs. In Clark, K. L., and Tärnlund, S. A., eds., *Logic Programming.* London: Academic Press.

Gao, Y., and Durrant-Whyte, H. F. 1994. Integrating qualitative simulation for numerical data fusion methods. In *Proceedings of the International Workshop on Qualitative Reasoning about Physical Systems.* Nara, Japan.

Kleiber, M., and Kulpa, Z. 1995. Computer-assisted hybrid reasoning in simulation and analysis of physical systems. *Computer Assisted Mechanics and Engineerind Sciences* 2(3):165–186.

Kuipers, B. J. 1986. Qualitative simulation. *Artificial Intelligence* 29(3):289–338.

Kuipers, B. J. 1993. Reasoning with qualitative models. *Artificial Intelligence* 59:125–132.

Ljung, L., ed. 1987. *System Identification; Theory for the User.* Englewood Cliffs, N.J.: Prentice-Hall.

Shapiro, E. 1983. *Algorithmic Program Debugging.* Cambridge, MA: MIT Press.

Vescovi, M.; Farquhar, A.; and Iwasaki, Y. 1995. Numerical interval simulation: Combined qualitative and quantitative simulation to bound behaviors of non-monotonic systems. In *Proceedings of the International Workshop on Qualitative Reasoning.* Amsterdam, Netherlands.

Weld, D. S., and de Kleer, J., eds. 1990. *Readings in Qualitative Reasoning About Physical Systems.* San Mateo CA: Morgan Kaufmann.

Williams, B. C. 1991. A theory of interactions: Unifying qualitative and quantitative algebraic reasoning. *Artificial Intelligence* 51.

Zhao, F. 1994. Intelligent computing about complex dynamical systems. In *Proceedings of the International Workshop on Qualitative Reasoning about Physical Systems.* Nara, Japan.