Automatic Construction of Accurate Models of Physical Systems

Elizabeth Bradley and Reinhard Stolle[1]

University of Colorado
Department of Computer Science
Boulder, CO 80309-0430
[lizb,stolle]@cs.colorado.edu

Annals of Mathematics of Artificial Intelligence, **17**:1–28 (1996).

Abstract

This paper describes an implemented computer program called PRET that automates the process of system identification: given hypotheses, observations, and specifications, it constructs an ordinary differential equation model of a target system with no other inputs or intervention from its user. The core of the program is a set of traditional system identification (SID) methods. A layer of artificial intelligence (AI) techniques built around this core automates the high-level stages of the identification process that are normally performed by a human expert. The AI layer accomplishes this by selecting and applying appropriate methods from the SID library and performing qualitative, symbolic, algebraic, and geometric reasoning on the user's inputs. For each supported domain (e.g., mechanics), the program uses a few powerful encoded rules (e.g., $\Sigma F = 0$) to combine hypotheses into models. A custom logic engine checks models against observations, using a set of encoded domain-independent mathematical rules to infer facts about both, modulo the resolution inherent in the specifications, and then searching for contradictions. The design of the next generation of this program is also described in this paper. In it, discrepancies between sets of facts will be used to guide the removal of unnecessary terms from a model. Power-series techniques will be exploited to synthesize new terms from scratch if the user's hypotheses are inadequate, and sensors and actuators will allow the tool to take an *input-output* approach to modeling real physical systems.

1 Introduction

Traditional system identification addresses the task of inferring a mathematical model of a system from observations of that system. A controls engineer might perform this task, in its most basic form, by choosing a power series and matching its coefficients against the numerical observations via some sort of regression[40]. This paper describes a computer program called PRET that automates the system identification process, at several levels, by building an artificial intelligence (AI) layer on top of a set of traditional system identification techniques. This AI layer automates the high-level stages of the identification process that

[1]This research was supported by two NSF grants: National Young Investigator award #CCR-9357740 and #MIP-9403223

are normally performed by a human expert. Qualitative, symbolic, and geometric reasoning are used to perform *structural identification* — the choice of the power series made by the engineer in the example above. This layer also automates another subtle and difficult part of the process: the choice and application of the appropriate lower-level method for each stage of the process.

PRET works with ordinary differential equation (ODE) models, linear or nonlinear, in one variable or many. Its implementation is a hybridization of traditional numerical analysis methods, such as simulation and nonlinear regression, with logic programming, computer vision techniques, and qualitative reasoning. The input consists of specific information about an individual system, in three forms:

- the user's hypotheses about the physics involved
- observations, interpreted and described by the user, symbolically or graphically, in varying formats and degrees of precision
- physical measurements made directly and automatically on the system

To construct an ODE model from this information, PRET combines powerful mathematical formalisms, such as the link between the divergence of an ODE and the friction of the system that it describes, with domain-specific notions — such as force balances in mechanical systems — to allow the types of "custom-generated approximations[53]" that are lacking in existing AI modeling programs. Two sets of rules, both of which may easily be changed or augmented by the user, play very different roles in the model-building task. Domain-specific rules are used to combine hypotheses into models — a nontrivial task in a system with more than one degree of freedom, or a system in which physical effects couple to one another — while general rules about ODE properties are used by a custom deduction engine to infer facts from models and from observations. Both model- and observation-based inferences are governed by specifications, which prescribe the resolution for quantities of interest. Any contradictions between the set of facts inferred from the observations and the set of facts inferred from a candidate model cause that model to be ruled out, in which case PRET tries a new combination of hypotheses. The first noncontradictory model in this sequence is returned as the answer.

Acting upon simple mechanical examples like the parametrically driven pendulum, the current version of the program can efficiently perform these tasks and construct accurate ODE models. Of course, a model of a driven pendulum is not the research goal here — physicists and engineers have spent centuries constructing and refining such models. This is simply an example — one that was chosen because it is instantly recognizable and intuitively obvious to the reader. In spite of its simplicity, this example is interesting from an AI standpoint, as it demonstrates effective automated reasoning, even if only on the level performed by a 17th-century physicist. One of the ultimate goals of this research is to produce a tool that can construct a model of a black-box system using only information from its ports. Textbook examples like the pendulum are critical to such an endeavor, as one must, for obvious reasons, verify the tool's performance on such exercises before trusting the results that it produces when presented with difficult open problems.

The following subsection is a brief review of AI and AI-specific perspectives on modeling research, coupled with a description of where this work fits within that context. The next section presents an overview of PRET's function, illustrates its input syntax using a simple example, and discusses some of the more important implications of that syntax. Section 3 outlines how the program uses that information, together with its encoded knowledge, to build an ODE model and closes with some discussion of related work, both in AI and other fields. The final section wraps up the example, gives a status report, discusses future directions, and summarizes some of the most important issues of the research.

1.1 Artificial Intelligence and Modeling

Research in AI has two major goals: the understanding of the mechanisms that make human intelligence possible[15] and the construction of intelligent artifacts[28]. Both ends of this spectrum — analytic and synthetic AI — depend on each other: a theory of human intelligence may be verified by the intelligent behavior of an artifact that instantiates the theory. Conversely, an engineer who builds an intelligent system may obtain useful ideas by observing human experts.

Intelligent behavior requires that the world knowledge that is relevant to the task at hand be available, along with the means to reason about it. The construction of an intelligent computer program requires a framework in which both knowledge and reasoning can be *formalized*. This formal system should be small and neat enough to be understandable and easy to maintain, and yet powerful enough to allow its users to think and formulate in the language and concepts of the application domain. An adequate representation formalism allows a natural formulation of the problem that is to be solved. Therefore, AI programmers typically try to represent knowledge declaratively rather than operationally: one formulates *facts* rather than *instructions*. Ideally, the declarative representation of the problem is executable. That means, it is not just *part of* the intelligent artifact but it *is* the artifact.

Modeling physical systems is an ideal application for these ideas and techniques. One of the most powerful analysis tools in existence — and often one of the most difficult to create — is a good model. Expert model-builders typically construct hierarchies of successively subtler representations that capture the salient features of a physical system, each incorporating more physics than the last. At each level in the hierarchy, the modeler assesses what properties and perspectives are important and uses approximations and abstractions to focus the model accordingly. The subtlety of the reasoning skills involved in this process, together with the intricacy of the interplay between them, has led many of its practitioners to classify modeling as "intuitive" and "an art[41]." Any tool that effectively automated a coherent and useful part of this art would be of obvious practical importance in science and engineering: as a corroborator of existing models and designs, as a medium within which to instruct newcomers, and as an intelligent assistant, whose aid allows more time and creative thought to be devoted to other demanding tasks.

The computer program PRET described in this paper is exactly such an automatic modeler. This work falls on the "intelligent artifact" end of the AI spectrum — its focus

is not to construct a cognitive model of the thought process of a physicist or 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. However, as outlined above, learning from a physicist's techniques is a fruitful approach.

PRET's techniques fall mostly in the category of *qualitative physics* (QP) or *qualitative reasoning* (QR)[7, 9, 21, 25, 54]. Like AI in general, qualitative reasoning about physics spans a whole spectrum, from modeling how humans reason about their physical environment[33] to engineering artifacts that can reason about physics[49]. Its main goals are the prediction of behavior, analysis, design, control, monitoring, and fault diagnosis. Many QR programs, particularly the ones that perform monitoring[23] or diagnosis[20], infer the behavior of a physical system from its structure or vice versa[16]. What distinguishes QP from other formalisms that represent physics knowledge, such as differential equations, is the abstraction to a qualitative level. For example, so-called *landmarks* divide the continuum of real numbers into a finite number of intervals. Typically, landmarks are critical values of quantities that describe the physical system. The behavior of the physical system — the progression of the values of relevant quantities — is described as a discrete sequence of states and state transitions. *States* describe situations, such as "$x = 0$" or "y is positive" or "$z = l_1$" where l_1 is a landmark. *State transitions* describe changing values, e.g., "x is monotonically increasing."

Many well-developed formalisms to represent and reason about mathematical, quantitative, and numerical knowledge exist. The goal of QP, however, is to formalize and automate conceptual, abstract, and qualitative reasoning in the physics domain. This qualitative kind of knowledge and reasoning requires a completely different set of primitives, such as the states and state transitions described above, or the facts about ordinary differential equations that are PRET's primitives. Adequate combinations of well-chosen primitives are a primary research goal in problems like this, as they enable a computer program to handle unforeseen situations. The ability to rearrange thought primitives in order to solve new problems is a crucial part of what makes a good scientist or engineer — or an intelligent artifact that performs the same tasks.

In general, *modeling*[24, 38] underlies most of the approaches to reasoning about physical systems. Strictly speaking, every formalization of the properties of a physical system constitutes a *model* of the physical system. The spectrum ranges from models that use a language that is very close to the physics of the system to models that use a language that is well-suited to describe the system mathematically. An example of the physics end of this spectrum might be formal instructions how to build a pendulum. These instructions would use terms like rod, bob, and bearing. The other extreme might be differential equations that use terms like $mg \sin\theta$. In any case, a modeler — human or not — builds the model out of simple components, assuming that the overall behavior follows from the behavior of the components and their interaction. Examples of QR modeling systems include the ENVISION system[18], which reasons about the components of the system and their interaction, as well as QPT (Qualitative Process Theory)[26] and its successor QPE (Qualitative Process Engine)[27], which emphasize the notion of causality. This is a very useful approach if the goal of the program is to *explain* certain phenomena in

physical terms[34]. QSIM (Qualitative SIMulation)[35] simulates the behavior of a physical system qualitatively. The description of the physical system, called a *qualitative differential equation* (QDE), uses mathematical language rather than terms from physics, namely the progression of relevant quantities (functions) and constraints on relations between these functions. The following example, drawn directly from a recent and thorough text on this topic[37], is a QDE fragment that relates the amounts of fluids in two connected containers, A and B, and the flow rates in the pipe between them, while constraining the total amount of fluid to remain constant:

```
...
((minus flowAB -flowAB))
((d/dt amtB flowAB))
((d/dt amtA -flowAB))
((add amtA amtB total))
((constant total))
...
```

From this information and some initial conditions[2], QSIM generates qualitative descriptions of every possible outcome, which it presents on graphs whose breakpoints are the landmarks described above. One current thrust of research by this group targets the integration of quantitative and qualitative information[6, 55]. Some other useful QR references are: [42] on varying resolution, [45] on order-of-magnitude reasoning, and [51] on mathematical aspects. A few useful general AI references are [1, 57].

On the spectrum from physics language to mathematics language, QPT resides on the physics end, QSIM on the mathematics end, and PRET somewhere in between. Its inputs — primarily observations and hypotheses about the physical system — are partially in the terms of physics. This approach allows the user to state the problem in his or her domain language. Also, PRET's reasoning uses concepts from physics, allowing it to rule out bad candidate models by high-level abstract reasoning. The rules about how hypotheses are combined into ODEs reflect laws of physics, such as $F = ma$. The decision about what to try next if some candidate model fails will also use physics concepts, e.g, "try quadratic friction instead of linear friction." However, PRET's output — the model of the physical system that it constructs — is purely mathematical: an ODE.

In summary, PRET makes heavy use of QR's notions of landmarks, qualitative vocabulary, and qualitative behavior. As do most QR systems, it exploits symbolic reasoning and reason-maintenance techniques. The structure of the physical system is described by notions like *point*, *loop*, etc., but there is no elaborate scenario description (e.g., a description that uses language that is highly specific to, say, fluids in containers). The program does not reason about causal relationships between behaviors of physical system components, nor does it try to *explain* observed phenomena. PRET takes a minimalist approach: find a simple model that is consistent with the observed behavior of the physical system.

[2] plus a few other details about variable names, ranges, and so on

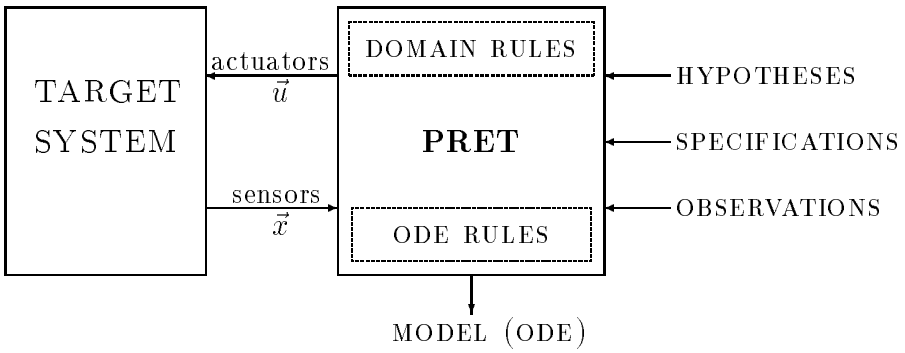


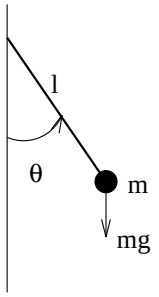
Figure 1: Structure and application

2 An Overview and an Example

The PRET modeling program, represented by the right-hand block in figure 1, constructs an ODE model of a target system based on information entered by a user; if the model is to be based on direct observations of a physical system, the program will obtain additional information via sensors, actuators, a hardware I/O channel, and data acquisition software. The output is an ordinary differential equation, of the form $f(\vec{x}, t) = 0$, whose behavior matches the observations to within the prescribed resolution. PRET builds this model in several stages, first using the encoded domain rules to combine hypotheses into candidate models and then checking those models against the observations modulo the precision inherent in the specifications. This check is performed by a first-order logic system that uses the current status and a set of encoded ODE rules to infer facts from the models and from the physical system. Any conflict between these two sets of facts, as mentioned in the previous section, is grounds for dismissal of the model under examination. The current version of the program implements this general framework, incorporating a basic set of domain and ODE rules that allow it to build models of simple mechanical systems. It is written in Scheme[46], uses symbolic algebra facilities from the commercial package Maple[13], calls upon the public-domain package ODRPACK[10] for parameter estimation, and will soon incorporate qualitative simulation[35]. Ultimately, PRET will refine and simplify models using a collection of techniques that are outlined in the later sections of this paper to add and remove terms from the ODE.

Figure 2 shows an example of how one might instruct PRET to construct a model of the damped pendulum. In this example, the user first sets up the problem, then hypothesizes four different force terms, makes five observations about the bob angle θ, and finally specifies resolutions and ranges for two important variables. It is important to note, as alluded to in the introduction, that this simple example is not by any means a true test of PRET's power.

The initial problem setup requires several steps. The first line of the `find-model` call




```
(find-model
        (domain mechanics)
        (state-variables <theta>)
        (point-coordinate <theta>)
        (hypotheses
             (<force> (* (constant A1) (deriv (deriv <theta>))))
             (<force> (* (constant A2) (sin <theta>)))
             (<force> (* (constant A3) (deriv <theta>)))
             (<force> (* (constant A4) (square (deriv <theta>)))))
        (observations
             (autonomous <theta>)
             (linear (<time> <theta>) (1 0) (range <theta> -.05 .05))
             (damped-oscillation <theta>)
             (asymptote (eqn 0)
                        (at <time> *infinity*)
                        (range <theta> 0 *infinity*))
             (numeric (<time> <theta>) ((0 .1234) (.1 .1003) ...  )))
        (specifications
             (resolution <time> absolute 1e-6 (0 120))
             (resolution <theta> absolute 1e-3 (0 (* 2 pi)))))
```

Figure 2: Instructing PRET to model the damped pendulum

specifies the domain and causes the program to instantiate the associated rules — here, the force-balance rule `(point-sum <force> 0)`. The next two lines identify `<theta>` as a state variable that is a `coordinate` associated with a `point` (the bob). Other types of coordinates — e.g., *flow coordinates* like currents in a circuit or *bulk coordinates* that quantify turbulence in a volume of liquid — will be necessary in other domains.

Identifying and classifying the critical state variables of a system is a non-trivial task. We do not plan to automate this process, but we do offer the user some assistance. PRET works most efficiently if all of the important state variables are identified in the `state-variables` statement, but it will ultimately incorporate power-series techniques, addressed in section 3.4, that will allow it to construct a model, in some cases, *even if the user omits important state variables. Redundant* state variables — ones that play no role in the model — increase the size of the search space but otherwise present no problems.

Hypotheses are ODE fragments expressed in terms of elements of the observation vector ($\vec{x}$ in figure 1) — which are typically system state variables[3] — and special keywords that provide the link to the domain and ODE rules. In this example, the keyword `<force>` is the link to the rule `(point-sum <force> 0)`[4]. The four `<force>` hypotheses shown in figure 2 are the obvious ones found in any freshman mechanics text: $F = ma$, circular-to-linear projection, and two simple forms of friction. A PRET call on an electronics application would use the keywords `<current>` and `<voltage>` and the associated domain rules `(point-sum <current> 0)` and `(loop-sum <voltage> 0)`. The `<time>` keyword is common to all domains. Manipulation of these constructs places some subtle and interesting requirements on both the internal and external representations of `coordinates`, as the program must be able to infer connections, loops, cutsets, etc., from the information in the input call. Note that the concepts of loop and point sums are not only appropriate for these examples, but also generalizable well beyond mechanics or electronics. Amsterdam[4] uses an energy-based modeling approach that generalizes these notions even further, into effort coordinates and flow coordinates. We have chosen a more-specific semantics in order to make the syntax more comprehensible to the domain experts who will be PRET's end users. Finally, extending the program to other paradigms (e.g., `volume-change`, etc.) is easy: one need only think of a new keyword and write a new rule that uses it, which is typically a matter of two or three lines of Scheme code, as shown in section 3.2.

Multiple hypotheses about a single effect may — *and should* — exist; the program will automatically determine which one(s) are appropriate. Some other modeling programs, e.g., Forbus and Falkenhainer's *compositional modeling*[24], define *groupings* of terms in such situations, such as a set of hypotheses about friction. Mutual exclusivity constraints are then imposed within each group, allowing the program to choose only one of the members, and thereby greatly reducing the complexity of the process. We have chosen not to follow this paradigm, for two reasons: (1) to minimize the restrictions on the models that the program can choose and (2) to minimize the high-level conceptual processing required of

[3]Not all state variables may be observable; this issue is discussed in section 3.5.

[4]A body-centered inertial reference frame is assumed here, together with coordinates that follow the formulation of classical mechanics[31], which assigns one coordinate to each degree of freedom, thereby allowing all equations to be written without vectors.

the user. We *do* allow users to specify that two terms must appear together — one simply lists both in a single hypothesis. This also brings up another interesting issue: the presence of a particular term can carry implications about others (e.g., centripetal and coriolis forces, which have distinctive forms and which appear together in rotating frames). These cases, which depend intimately on domain knowledge, will be exploited in the model refinement process that is the next step in this research project.

An *observation* describes the behavior of a single element of the observation vector, either in the time domain or in any state-space projection[5]. Unlike hypotheses, observations may not conflict. They have two potential sources — the user and the sensors — and may be *descriptive*, *graphical*, or *numeric*. The former use special *descriptive observation keywords*, the second are sketches drawn on a computer screen with a mouse, and the third simply specify data points. Descriptive observation keywords, such as `damped-oscillation`, `linear with [slope, intercept]`, `chaotic`, `autonomous`, etc., resemble terms in qualitative physics(QP)[54]. These terms play a key role in PRET's reasoning process: the logic engine uses the encoded ODE rules to infer facts from them in order to corroborate or rule out candidate models. For example, if the system is observed to be autonomous, the logic system infers that the variable `<time>` cannot appear explicitly in the ODE model. The processing of numeric observations, on the other hand, proceeds at a much lower reasoning level, forcing point-by-point comparisons with numerical integrations of ODEs. We have added a qualitative layer on top of this that uses curve fitting, asymptote recognition, phase-portrait analysis techniques drawn from computer vision[12], etc., to distill high-level descriptive information out of the raw numerical data. This information is then used much as descriptive observation keywords are, greatly raising the level of the reasoning process. The user's sketched graphical observations will undergo exactly the same type of preprocessing and distillation. Observations from the hardware I/O channel will be treated much like numeric observations, but at a higher confidence level. Finally, observations of any form must encode the range in which they are valid; the endpoints of these ranges are akin to QP's *landmarks*.

Observations guide the modeling process in a fundamental way. A model constructed by a human expert matches — *minimally* — a particular set of observations; the model builder does no more work than 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. The automatic modeler described here does exactly the same thing, expending the least possible effort to corroborate models and observations by reasoning at the highest possible level at all times and using the most qualitative form of the information involved.

These issues have important implications for the treatment of the unknown coefficients `A1, A2`, etc. in figure 2 — ones that are closely related to the issues addressed in *constraint logic*[14]. A descriptive observation often places only qualitative restrictions or bounds on these values, whereas matching a model against a numeric observation usually requires computation of exact coefficient values out to the number of significant figures dictated by the specifications. Moreover, a single observation, qualitative or quantitative, can contain information about many different variables. Such an observation would significantly

[5] PRET will perform no frequency-domain reasoning.

sharpen the model by forcing the evaluation of several coefficients in several different parts of the differential equation. For all these reasons, there is no good way to determine in advance how the number and type of undetermined coefficients, state variables, hypotheses, and observations will affect the form of the model. A final, more-obvious implication of the role of observations in the modeling process is that neither a program nor a human expert can be expected to construct a model of a system if no observations about that system are given.

A *specification* concerns any function of any number of observation vector elements; it prescribes the range of interest and the resolution limits for that quantity and specifies whether the latter is absolute or relative. The `resolution` statements in figure 2, for instance, instruct the modeler to impose microsecond and milliradian resolution over 120 seconds of system evolution. A successful model must match all observations to within these bounds, so they strongly influence many forms of processing in all stages of the program. This has a variety of implications, the most interesting of which stem from the translation between quantitative and qualitative representations; a nanosecond glitch, for instance, should be invisible in a millisecond-resolution run. These issues have been discussed extensively in the QP literature[8] and are reflected in obvious ways throughout PRET's code. Both range and resolution effects can alter behavior classification: a tiny chaotic attractor can be checked through as a fixed point, and a globally nonlinear ODE can be returned as the model for a linear system — if it has a linear region of the right width in the right place. This is not an unwelcome side effect of finite resolution. It is an intentional and useful by-product of the abstraction level of the modeling process. It is important to note that specifications implicitly govern the level of abstraction that the modeler enforces: sharpening the resolution will typically force the modeler to account for lower-level effects and add terms to the ODE, given a fixed set of observations.

This set of inputs was consciously chosen to mimic the information that an expert designer uses when he or she constructs a model. The intention was to make PRET interface smoothly with human skills, reasoning, and communication patterns. These choices, and their justification, are specific to this particular project. The debate about whether or not computer problem-solving processes should in general emulate their human equivalents has a long and somewhat contentious history, to which we plan no contribution.

3 How PRET Works

To build an appropriate ODE model from the information described in the previous section, PRET calls upon its encoded knowledge via a custom first-order logic system. As mentioned before, this knowledge base includes general knowledge about ODEs — how to recognize and locate attractors; that the divergence of a dissipative system is negative; that the entries in the Jacobian of a linear system are constants; etc. — and domain-specific knowledge like force or energy balances. Both general ODE rules and domain-specific rules may easily be changed or augmented by the user; several dozen of the former and half a dozen of the latter will be hard-coded in each supported domain. These rules govern

how hypotheses and models are combined, tested, ruled out, augmented, and simplified. Throughout the process, reasoning proceeds at the highest level possible; symbolic and qualitative techniques are attempted first and numeric ones are used only as a last resort. Finally, the modeling process will ultimately be an active one, since the program has been designed to use actuators and sensors to manipulate the system and test the evolving model.

A set of candidates for the program's first attempt at a model is constructed by mapping the domain rules onto the hypotheses — e.g., using `(point-sum <force> 0)` to combine `<force> =` $A_1\ddot{\theta}$ and `<force> =` $A_2 \sin\theta$ into the model $A_1\ddot{\theta} + A_2 \sin\theta = 0$. Most of the resulting candidate models can quickly be ruled out using symbolic techniques; for example, if a state variable is known not to be constant, the model must include its derivative. The simplest remaining model — the *base model* — is then checked against the specifications and observations. If this check fails, the program *refines* the ODE, using the domain rules to introduce another hypothesis from the user's list. The check is then repeated and the program loops until either a successful model is found or the possibilities are exhausted. As a last resort, if no combination of user hypotheses yields an adequate model, the refiner will ultimately use power-series expansion methods to synthesize terms from scratch. Successful models that emerge from this process will eventually undergo a round of simplification, wherein terms are removed and the resulting ODE is rechecked, to remove superfluous terms.

This control flow design contains a compromise. The simplify/refine loop will allow the program to move sideways through the search tree of models, recovering from bad choices and making globally good moves that require descent into *one* local minimum of the search landscape. We could also have chosen to loop more than once, which would increase the width of the program's lateral reach through the search space and allow it to find the *provably minimal model* in that space. However, such a search would have the standard complexity problems[56] and one of the goals of this project is to produce a "good enough" answer in minimal time.

The four main steps in the process described in the previous paragraphs — and the mechanics and implications of failure in each — are described in the next four subsections. The following subsection describes the design and use of hardware interface and the mechanics of input-output modeling, and is followed by a brief discussion of related work.

3.1 Generating the First Model

The base model generator uses domain rules to combine hypotheses and ascertains which of those combinations (models) are minimally consistent with the user's observations. The intent is to produce a preliminary solution to an exponentially complex problem very quickly; to effect this, the base-model generator uses a subset of the full set of inference rules called upon by the consistency checker and does very little reasoning about what terms to try next. When the answer that it produces is incorrect — a not-unlikely occurrence because of its quick-and-dirty techniques — the refiner and simplifier will act as a safety net, as described in the penultimate paragraph of the previous section.

The hypothesis list is first sorted using a simple-minded complexity metric, discussed in section 3.3. The base-model generator then constructs a one-term model from the first hypothesis in the list and checks it against the observations. The checker tries to find contradictions between properties of the model and properties of the system to be modeled. At the beginning of this process, the only known properties of the target system are the user's observations. The inference mechanism deduces additional properties of both the model and the target system. This deduction process will eventually be governed by a set of control rules in order to find existing contradictions quickly. The syntax in which properties and rules are represented is similar to first-order logic; our "descriptive observation keywords" correspond to logic's "predicates." For example, from the observation `(damped-oscillation <theta>)` in figure 2, it will be inferred that the model must be of at least second order. The first candidate model — the one-term ODE $A_2 \sin\theta = 0$ constructed by `(point-sum <force> 0)` from the second hypothesis in the figure — does not meet this requirement, so the program establishes a contradiction and the check fails. Note that PRET accomplished this using only high-level, symbolic reasoning, much as a human expert would, dismissing this model "by inspection."

The process is repeated using the next entry in the sorted hypothesis list, and so on. If none of the one-term models passes, the base model generator then starts testing two-term combinations. Note that *combination* need not imply *sum*; the form of the combination operator is implicit in the operative domain rule. Note, too, that the hypothesis combination mechanism is obvious and trivial here, but it rapidly becomes complicated and subtle in an even slightly more complex system, such as a double pendulum, which has two degrees of freedom and four coupled state variables.

If all two-term models fail, the program proceeds to three-term models, and so on. The first ODE in this succession whose properties do not conflict with the facts inferred from the observations by the operative set of inference rules is then passed to the full check/refine/simplify loop, whose elements are described in the next three sections. The refiner will improve upon this simple, mechanical progression through the hypothesis list by reasoning qualitatively about what the model-observation disparities imply about what kinds of hypotheses to try next; see section 3.4 for more discussion of this issue.

3.2 Checking Models Against Observations

The consistency checker carefully compares the behavior of an ODE to a set of observations, using the specifications as guidelines for how closely to enforce the match. The core of this part of the program is the same custom logic inference system used in the base model generator to infer knowledge about the model and knowledge about the physical system via application of rules to the ODE and to the observations. The checker, however, uses a much larger set of inference rules. Any contradiction between the two sets of inferred facts, as before, causes the model to fail the check. Like the other parts of the program, the consistency checker uses high-level reasoning first, attempting to establish a contradiction using only qualitative and symbolic methods, and performs blind numerical simulations and comparisons — which are absent from the base model generator's rule set — only as

a last resort.

The program's knowledge about the model's coefficients evolves in an observation-dependent fashion as the ODE moves through the phases of the consistency check. If all of the facts inferred by the logic system are coefficient-independent — e.g., the "model must be at least second order" implication of the `(damped-oscillation <theta>)` observation in figure 2 — none of the A_i will be determined in the inference process, so the returned model strongly resembles a QDE (see the QSIM example in section 1.1). In a point-by-point comparison with a numeric observation, on the other hand, the ODE coefficients must be computed before the model's behavior can be simulated. Thus, quantitative observations "drive" the ODE model away from the imprecise, QDE-type end of the spectrum of precision and abstraction and towards the exact, fully specified ODE end.

The problem of determining values for the coefficients, known as *parameter estimation*, is the topic of a rich body of literature[48] and the focus of many sophisticated global optimization techniques. We solve it using the ODRPACK package[10], based on orthogonal distance regression and developed at NIST. Given a data set and an ODE with unknown coefficients, ODRPACK computes values for the coefficients and returns a least-squares error measurement for the fit. Currently, we pass the returned coefficients directly to a numerical integrator, so if ODRPACK's fit is bad, the point-by-point comparison will fail. A more intelligent interpretation of the least-squares error metric is almost certainly possible, but we have not yet determined how to automate it effectively. One *should* be able to do some useful qualitative reasoning with this information, since it typically differs by 4 to 5 orders of magnitude between ODRPACK runs on good and bad models. However, it is a strong function of the length of the data set, the nonlinearity of the ODE, and a variety of other factors, so its use is more complicated than it might appear. Incidentally, we cannot just use the least-squares sum to corroborate a model against a numeric observation, as there is no way to instruct ODRPACK to filter its computations through specific ranges or resolutions.

Many observations fall between the numeric and symbolic extremes, causing the inferred facts to contain bounds or qualitative restrictions on algebraic combinations of coefficients. One of the program's most powerful and complex rule groupings is a good example of this: if a system oscillates, the imaginary parts of at least one pair of its roots must be nonzero. A *damped* oscillation further implies that the real parts of all roots must be less than zero, and so on. 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 preceding discussion is actually somewhat simplified; the concept of a "root" of a nonlinear equation, even over a limited, linear range, requires some extra care in interpretation and use. Note, finally, that many different forms of facts can be inferred from a single observation, ranging from the purely symbolic ("at least second order") to partially ($4A_1A_3 > A_2^2$) or fully quantitative ($A_1 = 1.235764$).

Much like a human modeler, the checker takes a minimalist approach: if there is no indication that the model is wrong, we assume it is right[6]. In the inference process used by the

[6] This resembles the *closed world assumption* that underlies many logic systems.

checker, knowledge about the physical system and knowledge about the candidate model is represented declaratively using first-order logic. New knowledge is deduced via application of rules to previously established facts. The check fails if and only if a contradiction has been established. An example of a rule whose application leads to a contradiction is

```
(<- falsum ((oscillation (var X))
            (scheme-eval (all-roots-real? (var X) current-model))))
```

This rule can be read as "it is contradictory that X oscillates and all roots of the model relative to X are real." The general form of these rules is (`<-` *head body*) where the body is a list of goals. Logical variables are identified through the keyword `var` and instantiated during the inference process by usual unification[7]. The special variable `current-model` is instantiated with the model that is currently being checked; in this way, the current model can be passed as an argument to Scheme functions that are invoked by the special predicate `scheme-eval`. A succeeding `scheme-eval` goal returns the corresponding unifiers. An ordinary goal succeeds if it can be unified with a previously established fact. The current incarnation of the program employs breadth-first forward reasoning. We are currently investigating the advantages and disadvantages of forward versus backward reasoning for our specific task. Note that all knowledge inferred about a particular model is relative to that model and is therefore local to the particular checker call. The knowledge inferred about the physical system, however, does not depend on the model whose check caused the knowledge to be inferred. Thus, facts about the physical system can be globally reused.

The set of inference rules that deduce facts from the descriptive keywords of qualitative observations is similar to the rule set used by the base model generator — in fact, the latter is a subset of the former. Table 1 gives some examples of descriptive observations and the facts that the logic system infers from them. This is only a small sampling of observations and inferences. Many other possibilities exist; because of the logic system's structure, implementing each is only a matter of a few lines of Scheme code and/or a call to Maple. For instance, regions where the behavior is observed to match a particular polynomial can be checked using the terms of a Taylor series constructed symbolically by Maple's `taylor` function. Singularities and symmetries are not only obvious to users, but also extremely powerful sources of purely symbolic comparisons. The classical mechanics formulation adopted here makes reasoning about symmetries straightforward: if the *conjugate momentum* that is associated with a coordinate — easily derived from an ODE with simple algebra — is zero, the system is symmetric with regard to that coordinate. A simple ODE rule exploits this property to corroborate model and observation symmetries at a purely symbolic level. The QSIM program[35, 37] mentioned in section 1.1 performs qualitative simulation of *qualitative differential equations* (QDEs); given an ODE↔QDE translator, currently under development, QSIM could be used to establish qualitative facts about ODE models.

While observations represent explicit information about properties of the physical system, an ODE does not contain explicit information about properties of the model — rather,

[7]Since this prototype of the program emphasizes correctness over efficiency, the unification algorithm includes the occurs check[50].

Observation about state variable x	**Implications for the model $f(x,t)=0$**
autonomous	cannot explicitly contain t (i.e., $f(x)=0$)
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 $\ddot{x}=0$
constant	should satisfy $\dot{x}=0$
conservative	$\nabla \cdot f = 0$
damped oscillation and autonomous	$\nabla \cdot f < 0$
growing oscillation and autonomous	$\nabla \cdot f > 0$

Table 1: Some observations and the corresponding inferences drawn by the logic system

ODE property	**Method**	**Consequence**
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

Table 2: Observer functions that infer facts from models

the ODE *is* the model. Observations act as a basic set of facts about the physical system from which further knowledge can be inferred. In a corresponding manner, a collection of Scheme functions — e.g., `all-roots-real?` — "observes" the model, yielding a basic set of facts about it, like the ones listed in the right-hand column of table 1. This collection of Scheme functions essentially implements the basic operations found in any mathematics text; some representative examples are shown in table 2. PRET's framework makes implementing the functions in these tables extremely simple. The rule that a linear model cannot account for chaotic behavior, for instance, requires only the following three statements:

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

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

(<- (linear (var X)) ((scheme-eval (linear-system? (var X)
                                                   current-model))))
```

We also would have to implement a Scheme function `linear-system?` that checks if an ODE (the current model passed as an argument) is linear in a certain variable — a simple symbol check on the result of a Maple `jacobian` call.

A consistency check of a model against a numeric observation proceeds at a much lower — and more expensive — reasoning level than a descriptive observation check. Such an operation typically entails a comparison of a numerical integration of the ODE, performed here with fourth-order Runge-Kutta[44], and a set of point-by-point observations. As mentioned above, this comparison is preceded by a orthogonal distance regression fit of model coefficients to numeric data, performed by the ODRPACK global optimization package. The integrator time step is chosen smaller than the spacing of the numeric data in order to make the comparison accurate; if the time step in the real data varies, integrated points are connected with splines. Proceeding directly to an expensive point-by-point comparison, however, is not the most intelligent approach, particularly since qualitative information can be distilled out of numeric data. This suggests a preprocessing step, in which qualitative features of the data in a numeric observation are extracted once, at the beginning of the run, and then used in the checks of every model in the progression. This tactic allows the logic system to establish some contradictions without performing numerical integrations of candidate models, even if all the observations are numeric. To test this idea, we have implemented geometric recognition of high-level behavior patterns like chaos, thresholds, oscillations, etc., using computer vision techniques that superimpose a variable-resolution grid over the system's state space, discretize the trajectories into lists of grid squares, and analyze the patterns in those lists[11]. This ties in well to the input format of section 2: specifications and observations can be used to set up the geometry of the grid, assuring adequate resolution in the results, attained with the minimum computation.

Graphical observations fall between descriptive and numeric observations in terms of the precision and processing required. The geometric reasoning and classification techniques described in the previous paragraph will be used to distill out high-level information; we also plan to use Maple's symbolic algebra, interpolation, and curve-fitting facilities to recognize bounds, asymptotes, polynomial approximations, etc., in the user's sketches. The logic system will treat the qualitative results yielded by these techniques exactly like any other descriptive observation keyword. Given its inherent imprecision, trying to distill any quantitative — or even semi-quantitative — information from a sketch makes no sense.

If any model fails a check, a failure report is generated; this report contains the model, the violated observation(s), and the proof tree constructed by the logic system in establishing the contradiction. These discrepancies are a rich source of information about the process, both preceding and following the failure. They could be used by the other program modules — a form of *discrepancy-driven refinement*[3, 53] — in a variety of ways: to back intelligently out of blind alleys, to avoid duplicating previously performed checks, or to pick up at some appropriate midpoint in the event of a restart. As a start, we plan to use qualitative and symbolic reasoning to interpret the discrepancy data; this could, for instance, suggest what terms from the hypothesis list to try next: if the cause of failure was a contradiction between a linear model and some observed complex behavior in the system, one might not wish to try yet another linear model. To avoid duplication and facilitate

escape from blind alleys, we ultimately plan to add the model's full genealogy to the failure report — all branches in the search tree of candidate models that were traversed during the construction, simplification, and refinement processes.

3.3 Simplification: Reducing a Model to Lowest Order

The purpose of the simplifier is to remove unnecessary terms from models. It will first identify a candidate term, delete it, and then invoke the consistency checker on the remaining ODE. If the check fails, the simplifier will replace that term and remove another. This process will be repeated until some $(n-1)$-term "sub-model" of the n-term model passes the check, or until all $(n-1)$-models have been tested unsuccessfully. In the latter case, the model is deemed minimal and is passed on as a final result. If any sub-model *does* pass the check, the entire process will be repeated, testing all $(n-2)$-term subsets of the original ODE, and so on. A subtle issue is involved in this progression: sometimes adding or removing *pairs* of terms (or larger groupings) works where successive single-term removal does not: again, consider coriolis and centripetal forces. We plan to investigate this and tailor the design accordingly.

The interesting reasoning in the simplifier concerns the following task: given an ODE and a set of observations that it satisfies, choose the term that is most likely to be superfluous. Like the consistency checker, the simplifier will take a top-down, symbolic-first approach to this task. It will use many of the same techniques as do the base model generator and the checker. For instance, if a system is observed to be chaotic, the model must be nonlinear, and so at least some of the entries in the system's Jacobian matrix must be functions of the state variables. Symbolic computation of that matrix will show which terms cannot be removed without making those entries constants. Symbolic algebra could also be used to identify terms whose removal would destroy matches to observed symmetries. For example, if angular momentum is conserved around a particular axis, the conjugate momentum p_ϕ associated with the coordinate ϕ that defines motion around that axis must be zero. If the user has defined such a coordinate — which, having observed the symmetry, s/he would almost certainly have done — p_ϕ is an extremely simple algebraic function of the ODE, and it would be obvious what terms must be present in the model to make it equal to zero. Since so much of this reasoning is so similar to that performed by the checker, much of the inference system will probably be shared between the two program modules.

If high-level symbolic and qualitative reasoning cannot identify a good candidate for removal, the simplifier will then turn to less-intelligent techniques, such as removing the "simplest" terms, as defined in the following paragraph, or terms whose coefficients are much smaller than the others in the model. These choices are somewhat *ad hoc*; it is not at all clear that we will gain much from either one, beyond a defined order of attack. Asymptotic order-of-magnitude reasoning[58] could be used to formalize the process of identifying terms with small coefficients; another approach might be to give the user some leverage by weighting hypotheses according to order of entry. Each of these tactics has advantages and disadvantages; each shifts a different amount of the burden of rigor from

the simplifier to the checker.

Ordering terms or hypotheses logically is difficult because no satisfying simplicity metric exists. The modeling literature contains no good solutions to this problem, and we have no better suggestions. The metric that we use is based on number and complexity of terms and derivatives: $x+3 \triangleright x$; $x^n \triangleright x^{n-1}$; $\dot{x} \triangleright x$; etc., where $\triangleright$ denotes "more complex than." This is unsatisfying for a variety of reasons, not the least of which is how to resolve ties between equations like $x + x^2$ and $x^3 + 4$. One of the seminal papers in qualitative modeling uses a similar metric[24] and gives similar disclaimers about its unsatisfying nature. A more-recent paper[43] defines model simplicity heuristically, terming a model that is "more approximate" or that "uses fewer phenomena" as more simple, but does not suggest how to solve the problem of how many hypotheses are associated with a phenomenon.

Finally, discrepancy-driven reasoning can aid in the simplification process. If a sub-model fails the check, the cause-of-failure report returned by the checker — specifically, the proof tree that contains the reasoning that led up to the contradiction — can potentially help the simplifier decide which term to try removing next. For example, if removing an everywhere-positive term like x^2 caused the model to fail to match some observation that required the divergence of the ODE to be zero, one might not want to remove an x^4 term next. The time-saving mechanism here is reuse. Removing other terms in the model may cause similar proof trees to be built and any section that is identical to that created by the previous removal of another term can be reused rather than reconstructed. Reason maintenance techniques[17, 22, 28] will be used to perform this type of book-keeping and manipulation of proof trees.

3.4 Refinement: Adding Terms to an Inadequate Model

The refiner will use ODE and domain rules, observations, and hypotheses to add terms to a model, keeping track of previous attempts in order to avoid duplication of effort. It will first draw upon the user's hypotheses, then synthesize ODE terms from scratch using power-series expansions if no successful hypothesis-based model can be found. These power-series methods are a powerful safety net: a one- or two-term expansion in θ and $\dot{\theta}$ would reproduce every form of friction found in freshman physics texts — and a few more besides. Moreover, these methods will actually allow the program to *create new state variables* — an important feature if the observation vector is smaller than the true state vector.

The refiner's first task will be to prune the sorted hypothesis list, removing any entries that were used in the base model. The first hypothesis in the sorted list will then be added to the base model and the checker called on the new ODE. If that model fails, the refiner will remove the previously introduced term and successively try the rest of the hypotheses in the ordering, one at a time. If no one-term addition causes the model to match the observations, the refiner will then try *pairs* of terms, and so on. This process is exponentially complex, but in a fairly small number, since the user is unlikely to enter more than a half a dozen hypotheses per coordinate. More importantly, most of these combinations will be ruled out with quick symbolic checks. The first successful model will be returned as the refiner's result. If the refiner exhausts the list of hypotheses before

finding an adequate model, the power series methods outlined in the previous paragraph will be invoked. These expansions are subject to a user-specified limit p; if no match has been established after the refiner tries the p^{th} term in the expansion, the program will admit failure and request additional hypotheses from the user.

Simplicity metric difficulties, discussed at the end of the previous section, also affect the design of the refiner. The metric proposed here may not be rigorous, but it at least provides a starting point. A possibly useful alternative would be, again, to use the entry order verbatim. A half-dozen informal interviews have suggested that most users do indeed list hypotheses in order of perceived simplicity, so this may not be a bad idea. A much more intelligent approach would be to choose the hypothesis to be added according to its behavior, using symbolic and numeric techniques similar to those discussed in the previous section, and to use the failure report as an additional source of pointers. For instance, if a model failed a check because its order was too low, a constant hypothesis should be lower on the list of things to try than a term containing a fifth derivative. Note that such a hypothesis should *not* be removed from the list altogether; it may well appear in the ultimate model, along with other terms that were added first. An even more intriguing idea would be to use domain knowledge, such as the link between coriolis and centripetal forces, to go directly to the right hypothesis in the user's list (or even to compensate for a user's oversight and introduce it, should it be missing).

The two automatic term-synthesis techniques proposed here are derived from power-series expansions. *Canonical perturbation theory*[31, chapter 11] creates new parameters and *Frobenius's method*[41, page 187] creates new state variables, each via expansion in the appropriate quantity. One can also synthesize new state variables using delay coordinates[32], but doing so would vastly complicate the symbolic algebra that plays such a critical role in PRET. Because the lower-order terms of power-series expansions are likely to match (up to a coefficient) simple physics hypotheses, the modeler will use simple symbolic techniques to avoid duplication of effort, eliminating any power series expansion terms that also appear in a user hypothesis. Note that the intelligent term-addition technique suggested at the end of the previous paragraph is based in domain physics, whereas these power-series expansions are mathematically general and purely mechanical methods.

The combination of the machinery described in this and the preceding three subsections will allow the program to preferentially check the user's hypotheses in some intelligent order, then synthesize and explore new state variables and parameters if necessary, possibly concocting model terms that do not resemble any of the given hypotheses. PRET will even be able to construct models in the absence of *any* hypotheses — if the underlying physics admits a power-series description. The ODE and domain rules, together with symbolic reasoning and qualitative behavior descriptions, will allow this tool to reason at a very high level indeed. For example, if a system was observed to be chaotic, but only one state variable was specified, the power series methods would be invoked automatically to synthesize a new state variable before the program even attempted to build a base model. This automatic modeling tool will be able to adapt the scope of the model on the fly, inferring "state variables" that are internal (e.g., voltages inside a black box) or that were omitted by oversight (e.g., low-amplitude, low-frequency vibration of a bench by a lab's

HVAC system). These quantities are diffeomorphically related to the true state variables of the system, so models based upon them can be used to draw some rigorous conclusions. Again, formal observer theory is not a goal of this project, so PRET will not be able to solve this problem in all situations.

3.5 Incorporating Physical Measurements

An important feature of the physical link between modeler and system is that data will be able to flow across it in both directions, making the modeling process an *active* one — in all parts of the procedure described in the previous four subsections. This has a number of important implications. Among other things, PRET could autonomously exercise the target system in order to verify or augment the observations in the `find-model` call, possibly constructing and using observations that transcend its user's knowledge of physics. For instance, the boundaries of the basins of attraction of the ODE and the target system could be compared using different-energy kicks, even if the user knew nothing about dynamics and basin structure. Obviously, this presents some dangers: if the program is free to use the sensors, the actuators, and the full breadth of its own (significant) knowledge base, the only controls on the sophistication and intricacy of the resulting model are the resolution and the expansion limit p. To address this problem, we have introduced another parameter, `max-level`, that gives the user explicit control over the number of terms that PRET may use in the model.

This approach also has some serious and imposing limitations: observability, reachability, and controllability. The components of the *observation vector* $\vec{x}$ in figure 1 may not represent unique functions of elements of the internal system state. Moreover, measuring any particular quantity may not be possible — because it is physically inaccessible or because no appropriate sensor exists. Finally, some state-space regions may not be reachable from the existing initial condition, which will limit PRET's ability to test out its hypotheses. We do not plan to attempt any sort of general solution and write code that instantiates the vast number of ideas in the controls literature that address these problems; we simply plan to identify the resultant limitations and work around them. Even if the possibilities are limited, any physical corroboration is better than none; moreover, even identifying the limitations will require some subtle and interesting manipulation of the actuators and interpretation of the sensor data.

On the whole, physical measurements will be treated exactly like numeric observations, from the program's point of view, with one important distinction: in the event of a conflict, the former will be trusted over the latter. Measurements will be processed and translated into the syntax of descriptive user observations and then used in exactly the same ways — as targets for symbolic comparisons. The measurement-processing algorithms that extract this qualitative information from the sensor data will follow the same approach to behavior recognition outlined near the end of section 3.2.

With the hardware data channel in place, the program could potentially be used not only to model a physical system, but also to debug designs, or even to validate and verify devices ostensibly constructed according to a known design — a procedure known in the

AI literature as *model-based diagnosis*, e.g., [20, 23]. For example, if a particular 2.5KΩ, 1/4 Watt resistor burned up every time a device was turned on, one could place probes across its terminals, set up an observation that specifies "voltage over 25 volts," and observe discrepancies between the resulting model and what was intended. Needless to say, one would want to isolate the sensors from potential damage during such an experiment.

3.6 Relationships to Related Work

Modeling research spans many fields, from the cognitive science-related branch of AI[39] through operations research[29], dynamic systems[30], and control theory[5] to qualitative reasoning (QR)[8]. The research described in this paper falls most naturally under the general AI rubric, but has strong ties to the system identification branch of control theory as well — in fact, it can accurately be described as an AI-assisted approach to system identification. Section 1.1 presented a broad introduction to AI and QR modeling research; readers of that section may wish to skip the next paragraph.

Some of the earliest QR/modeling work[24] builds upon a fixed base of hypotheses, instantiates only those that are appropriate to answer a given query, and chooses between them with a truth maintenance system. The *graph-of-models* approach[2] is similar in many regards to [24], but represents the space of possible models as a directed graph of models where edges between nodes (models) are approximations[8]. Another approach[53] adapts models to problems using *model sensitivity analysis*, which formalizes and exploits the effects of parameter changes in the construction of the model. The combinatorial explosion involved in limit checking (e.g., the pendulum's asymptote to $\theta = 0$) can be mitigated[36] by decomposing and abstracting time into a hierarchy of scales. Rules for determining the behavior of a composite device can be derived from the models of its constituent components[18]. This is an extremely active research area; many good papers by many other groups, as well as many other papers by the cited groups, have appeared in the past five years. The state of the art in this field is particularly well-summarized in a review article by Weld[53], which is also the source of much of the terminology used here. Concepts common to this paper and the bulk of the QR/modeling literature include avoidance of unnecessary terms, model refinement driven by failure at a "lower" modeling level, and reasoning that proceeds at as a high level of abstraction as possible.

PRET solves the same problems as traditional system identification[40], but techniques in the latter tend to be purely mechanical and numerical. [47], for example, does a Volterra-series expansion in the input variable to obtain a set of terms (much like those concocted by the second refiner phase, after the user's hypotheses have been exhausted), then matches coefficients using regression. Commercial programs like Diffeq automate the lowest levels of the system identification procedure, allowing users to interactively modify parameters to obtain an optimal solution. The goal of the research described here — and the most important difference between PRET and programs like Diffeq — is *the automation of that interaction*. An important and difficult part of this task is the automation of the decision

[8]The propositional reasoning in [24] also uses a digraph, but it is used implicitly and constructed somewhat differently.

about when to use which techniques and how to set them up. For instance, PRET automatically sets up a complex call to a parameter estimation package to find values for the unknown coefficients, specifying parameters, initial conditions, tolerances, timesteps, etc. that are appropriate for each individual situation. Such decisions, often the most subtle parts of the problem-solving process, are unthinkingly natural to a human expert.

4 Status and Discussion

This paper describes results of the first stages of a highly ambitious project and proposes some solutions that will be required for the next stages of this task. To date, we have built a functional and representative subset of the program — hence the mix of verb tenses and section lengths herein. The current version of PRET incorporates a few instances of each technique, providing a quick check of the whole symbolic/numeric paradigm and the overall control flow (hierarchy of hypotheses, domain rules and ODE rules; control flow between modules, and so on). Most importantly, it has allowed us to test and refine the syntax and the use of the various types of user inputs.

This first version allows one-term hypotheses involving a single state variable and the keywords `<force>` and `<time>`, sorts them according to the simplistic metric proposed in section 3.3, incorporates one rule (`(point-sum <force> 0)`) in one domain (`mechanics`), has a reduced vocabulary of a dozen or so qualitative terms (e.g., `chaotic, linear, autonomous, numeric`) and a correspondingly small repertoire of symbolic and numeric techniques that infer facts from these properties, and has been tested on two-dimensional systems, such as the pendulum modeling call of figure 2.

To illustrate PRET's operation, we will give a narrative description of the program's actions as it executes the `find-model` call given in figure 3, a version of figure 2 that has been simplified to streamline presentation. Once again, note that the pendulum is used here because it is a clear and obvious example; 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.

Preprocessing of the numeric observation shows that `<theta>` is not constant[9]. The base-model generator then uses the force-balance rule to map the first hypothesis into a one-term model. This model, $A_2 \sin\theta = 0$, does not contain the derivative of θ, and so is immediately ruled out via contradiction with the `not-constant` fact inferred from the numeric observation. PRET's second attempt, $A_1\ddot{\theta} = 0$, does pass the `not-constant` consistency check because its order is high enough. The check of this model then proceeds to the numeric phase. The parameter estimator is invoked to compute the coefficients, but the model is so far off that ODRPACK is unsuccessful, causing the check to fail. The program then attempts the two-hypothesis model $A_1\ddot{\theta} + A_2 \sin\theta = 0$. This model, like $A_1\ddot{\theta} = 0$, passes the `not-constant` check because it contains a second derivative of θ. Unlike the simpler model, however, it is good enough to let the parameter estimator match

[9] This can be established with the first two data points: $0.1234 - 0.1003 > 1e-3$.

```
(find-model
        (domain mechanics)
        (state-variables <theta>)
        (point-coordinate <theta>)
        (hypotheses
             (<force> (* (constant A1) (deriv (deriv <theta>))))
             (<force> (* (constant A2) (sin <theta>)))
        (observations
             (autonomous <theta>)
             (numeric (<time> <theta>) ((0 .1234) (.1 .1003) ...  )))
        (specifications
             (resolution <time> absolute 1e-6 (0 120))
             (resolution <theta> absolute 1e-3 (0 (* 2 pi)))))
```

Figure 3: A simple modeling run on the damped pendulum: the invocation

its coefficients against the data. Using these coefficients, fourth-order Runge-Kutta, divided differences for the initial conditions, and splines to interpolate between data points, PRET compares the numerical solution of the ODE to the numeric observation in the `find-model` call. This comparison succeeds, and the model and its coefficients are returned as the final result:

```
((model ((= (+ (* (constant A2) (sin <theta>))
               (* (constant A1) (deriv (deriv <theta>))))
               0)))
        (((constant A1) 2.) ((constant A2) 3.)))
```

The returned value also contains some other information, not shown here, concerning inference depths, number of rule applications, and abstraction level, plus a list of all known facts about the model — which, since there was no contradiction, we assume are also facts about the system.

Note that the `autonomous` observation never comes into play here because none of the hypotheses are functions of time. Had the user entered a term like

```
(<force> (* (constant A6) (sin (* 1.5 <time>)))),
```

PRET would have constructed and tested nonautonomous models like $A_1\ddot{\theta} + A_6 \sin 1.5t = 0$. Facts inferred from such models would conflict with facts inferred from the `autonomous` observation. Since these inferences would require only symbolic manipulation, eliminating those models would be an inexpensive operation — much less so than the parameter estimation, numerical integration, and so on that might be required in the absence of that observation.

Had figure 3 included a `(damped-oscillation <theta>)` observation, as in figure 2, PRET would have inferred, among other things, that the model must be of at least second

order. Like the `not-constant` fact inferred from the numeric observation, this would cause a contradiction that ruled out the model $A_2 \sin\theta = 0$. There are two important issues here: (1) inferring two facts where one would be enough to establish a contradiction is wasted effort and (2) the processing of the `damped-oscillation` is much less computationally expensive. PRET currently applies *all* of the inference rules and *then* looks for contradictions. We plan to address the two issues mentioned above by implementing a *control rule* layer that will guide the reasoning more intelligently — e.g., applying the least-expensive rules first and looking for contradictions at several midpoints in the process.

Finally, we have also experimented with adding white noise to the numeric data and changing the resolution in the specifications, with predictable results: the program still finds the right model if the added noise is small compared to the resolution, and fails when it is not.

PRET does not currently simplify models if they contain too many terms, nor does it refine them in any of the intelligent ways suggested in section 3.4; these tasks are two of our current research objectives. We are currently refitting PRET to use the proof trees maintained by the logic engine (which document all of the reasoning that has been performed) and discrepancy-driven reasoning to decide which terms to add or remove from the ODE. We are also working on power-series techniques to synthesize hypotheses automatically — an idea that is ubiquitous in system identification but completely absent from the AI modeling literature. Finally, we are exploring analysis and control techniques to be used to actively and realistically exploit sensors and actuators to allow a true *input-output* approach to modeling[10].

Longer-term goals are to expand PRET's capability to handle more-complex and less well-specified systems, in several domains (mechanics, electronics, etc.), described by sketchier observations, using a much richer rule set and multiple, heterogeneous keywords. We expect these tasks to be harder and more interesting than those described in the previous paragraph, particularly since our aim is emphatically not to build a tool that is tuned for one particular application domain.

5 Summary

The work described in this paper constitutes a new approach to the old problem of system identification — one that calls upon artificial intelligence techniques, like logic programming and qualitative, symbolic, algebraic, and geometric reasoning, to automate some of the high-level reasoning involved in the system identification process — choices, judgements, and tasks that are normally performed by an expert control engineer. PRET, the computer program that embodies these ideas, uses general mathematical theory as a foundation, adds concise and powerful domain-specific rules, and funnels user-specified hypotheses through general ODE theory and domain-specific rules via a custom first-order logic system to gen-

[10] This is also relatively common in system identification but rare in AI. There has been much recent interest in diagnosis and testing in the *general* AI literature[19], but the modeling subset of that community has only just begun to use those ideas[52].

erate "appropriate" models — ones that are well-matched to the task at hand. It exploits intelligent, high-level techniques like symbolic manipulation from the outset, carrying them as far as possible through each phase of the work, and using them to make the type of quick, overall assessment that a human expert performs in the first stages of model-building. The program then resorts to lower-level, less-intuitive methods to complete the analysis and synthesis processes. The chosen set of inputs and the way that they are used closely resembles the process one finds documented on any designer's scratch paper: parts of equations, rough sketches, scratched-out forays up analytical blind alleys, and an overall progression of ideas and abstractions from simpler to more complex.

What distinguishes this work from existing system identification tools is the amount of the problem-solving process that is automated. The program described here uses many of the same mathematical, mechanical, and numerical techniques used in traditional system identification, but contains an additional reasoning layer that automates many of the higher-level tasks normally performed by its human practitioners — importantly, the choice of which particular mechanical technique to use in a given phase of the identification process.

What distinguishes this work from traditional AI research is its use of quantitative methods, engineering constraints, and real-world properties. Its mixture of exact and approximate techniques and precise and heuristic knowledge may appear inelegant, but it is very powerful. Another distinguishing characteristic is the ultimate goal of the research: to help engineers model real systems, not to solve toy problems. Obviously, this will be difficult to attain, and progress must necessarily begin with solutions of known, textbook-style problems.

The ultimate target applications of this modeling tool lie in areas where the basic physics is not well-understood, where the systems are hard to observe, or where the mathematics is very complicated. Ideally, this program would be able to build a model of a black-box circuit, given access only to a single port[11]. The power-series methods of section 3.4 would be critical to this, as they provide the mechanism whereby internal state variables are inferred. Another optimistic example might be turbulent flow, where the underlying physics is based on partial differential equations (PDEs) and the difficult part of the modeling task often consists of finding the right ODE truncation of the underlying PDE-governed behavior. Here, the modeling tool's contribution would be to combine mathematics, observed physical phenomena, approximation, and abstraction to construct and test a large number of models much more rapidly than an unassisted human user could, perhaps even reasoning at a level beyond the user's knowledge of mathematics, physics, or observer theory. The structure of the program — syntax, control flow, ODE/domain rule mechanics, term-synthesis methods, actuator/sensor use, etc. — was designed to facilitate this, and the examples presented here are small but useful steps in the right direction.

Acknowledgements

The authors would like to thank Janet Rogers of NIST for making ODRPACK available and for instructions on its use.

[11]Of course, the model could only account for behavior that is "visible" from that port.

Biographies

Elizabeth Bradley received the S.B., S.M., and Ph.D. degrees from the Massachusetts Institute of Technology in 1983, 1986, and 1992, respectively, including a one-year leave of absence to compete in the 1988 Olympic Games. She joined the Department of Computer Science at the University of Colorado at Boulder in January of 1993 as an Assistant Professor, with a joint appointment in Electrical and Computer Engineering, and is also affiliated with the Program in Applied Mathematics. Her research interests are nonlinear dynamics and chaos, scientific computation and AI, network theory and circuit design, and classical mechanics. She is a member of Eta Kappa Nu, Tau Beta Pi, and Sigma Xi, and the recipient of a 1993-1998 National Young Investigator award and a 1995-2000 Packard Fellowship.

Reinhard Stolle is a PhD candidate and research assistant at the University of Colorado. He received the M.S. degree from the University of Colorado in 1993 and the Vordiplom from the Universität Erlangen-Nürnberg, Germany, in 1990. He was the recipient of a 1992/1993 Fulbright travel grant.

References

[1] H. Abelson, M. Eisenberg, M. Halfant, J. Katzenelson, G. J. Sussman, and K. Yip. Intelligence in scientific computing. *Communications of the ACM*, June 1989.

[2] S. Addanki, R. Cremonini, and J. S. Penberthy. Reasoning about assumptions in graphs of models. In *Proceedings IJCAI-89*, 1989. Detroit, MI.

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

[4] J. Amsterdam. *Automated Qualitative Modeling of Dynamic Physical Systems.* PhD thesis, MIT, 1992.

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

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

[7] D. G. Bobrow. Qualitative reasoning about physical systems: An introduction. *Artificial Intelligence*, 24:1–5, 1984.

[8] D. G. Bobrow, editor. *Qualitative Reasoning about Physical Systems.* MIT Press, Cambridge MA, 1985.

[9] D. G. Bobrow, editor. *Artificial Intelligence in Perspective.* MIT Press, Cambridge MA, 1993.

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

[11] E. Bradley. *Taming Chaotic Circuits*. PhD thesis, MIT, September 1992.

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

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

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

[15] A. Collins and E. E. Smith, editors. *Readings in Cognitive Science*. Morgan Kaufmann, San Mateo CA, 1988.

[16] R. Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24, 1984.

[17] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:2, 1986.

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

[19] J. de Kleer, A. Mackworth, and R. Reiter. Characterizing diagnoses. In *Proceedings AAAI-90*, 1990.

[20] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1), 1987.

[21] J. de Kleer and B. C. Williams, editors. *Artificial Intelligence*, volume 51. 1991.

[22] J. A. Doyle. A truth maintenance system. *Artificial Intelligence*, 12, 1979.

[23] D. Dvorak and B. J. Kuipers. Model-based monitoring of dynamic systems. In *Proceedings IJCAI-89*, 1989.

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

[25] B. Faltings and P. Struss, editors. *Recent Advances in Qualitative Physics*. MIT Press, Cambridge MA, 1992.

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

[27] K. D. Forbus. The qualitative process engine. In D. S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning About Physical Systems*. Morgan Kaufmann, San Mateo CA, 1990. University of Illinois Department of Computer Science Technical Report 1986.

[28] K. D. Forbus and J. de Kleer. *Building Problem Solvers.* MIT Press, Cambridge, MA, 1993.

[29] S. I. Gass, editor. *Operations research, mathematics and models.* American Mathematical Society, Providence, R.I., 1981.

[30] N. S. Gershenfeld and A. S. Weigend. The future of time series. In *Time Series Prediction: Forecasting the Future and Understanding the Past.* Santa Fe Institute Studies in the Sciences of Complexity, Santa Fe, NM, 1993.

[31] H. Goldstein. *Classical Mechanics.* Addison Wesley, Reading MA, 1980.

[32] G. Gouesbet and J. Maquet. Construction of phenomenological models from numerical scalar time series. *Physica D*, 58:202–215, 1992.

[33] P. Hayes. The second naive physics manifesto. In J. Hobbs and R. Moore, editors, *Formal Theories of the Commonsense World*, pages 1–36. Ablex, Norwood, 1985.

[34] Y. Iwasaki and H. A. Simon. Causality in device behavior. *Artificial Intelligence*, 29:3–32, July 1986.

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

[36] B. J. Kuipers. Abstraction by time scale in qualitative simulation. In *Proceedings AAAI-87*, pages 621–625, 1987. Seattle, WA.

[37] B. J. Kuipers. *Qualitative Reasoning: Modeling and Simulation with incomplete knowledge.* Addison-Wesley, Reading, MA, 1992.

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

[39] P. Langley, H. A. Simon, G. L. Bradshaw, and J. M. Zytkow, editors. *Scientific Discovery: Computational Explorations of the Creative Process.* MIT Press, Cambridge, MA, 1987.

[40] L. Ljung, editor. *System Identification: Theory for the User.* Prentice-Hall, Englewood Cliffs, N.J., 1987.

[41] F. Morrison. *The Art of Modeling Dynamic Systems.* Wiley, New York, 1991.

[42] S. Murthy. Qualitative reasoning at multiple resolutions. In *Proceedings AAAI-88*, 1988.

[43] P. Nayak. Causal approximations. In *Proceedings AAAI-92*, 1992.

[44] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, Cambridge U.K., 1988.

[45] O. Raiman. Order of magnitude reasoning. *Artificial Intelligence*, 51, 1991.

[46] J. Rees and W. Clinger. The revised^3 report on the algorithmic language Scheme. *ACM SIGPLAN Notices*, 21:37, 1986.

[47] M. Schetzen. A theory of non-linear system identification. *International Journal of Control*, 20:577–592, 1974.

[48] H. W. Sorenson. *Kalman Filtering: Theory and Application*. IEEE Press, 1985.

[49] R. M. Stallman and G. J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135–196, 1977.

[50] L. Sterling and E. Shapiro. *The Art of PROLOG*. MIT Press, Cambridge, MA, 1986.

[51] P. Struss. Mathematical aspects of qualitative reasoning. *International Journal for Artificial Intelligence in Engineering*, 3(3), 1988.

[52] P. Struss. Testing physical systems. In *Proceedings AAAI-94*, 1994.

[53] D. S. Weld. Reasoning about model accuracy. *Artificial Intelligence*, 56:255–300, 1992.

[54] D. S. Weld and J. de Kleer, editors. *Readings in Qualitative Reasoning About Physical Systems*. Morgan Kaufmann, San Mateo CA, 1990.

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

[56] P. H. Winston. *Artificial Intelligence*. Addison Wesley, Redwood City CA, 1992. Third Edition.

[57] K. Yip. Understanding complex dynamics by visual and symbolic reasoning. *Artificial Intelligence*, 51, 1991.

[58] K. Yip. Model simplification by asymptotic order of magnitude reasoning. In *Proceedings AAAI-93*, pages 634–640, 1993.