Reasoning about Nonlinear System Identification

Elizabeth Bradley [a,1], Matthew Easley [a,1], Reinhard Stolle [b,2]

[a] *Department of Computer Science,*
University of Colorado,
Boulder, CO 80309-0430
{lizb,easley}@cs.colorado.edu

[b] *Xerox PARC*
3333 Coyote Hill Road
Palo Alto, California 94304
rstolle@parc.xerox.com

Technical Report CU-CS-894-99

In review, *Artificial Intelligence*

Abstract

System identification is the process of deducing a mathematical model of the internal dynamics of a black-box system from observations of its outputs. The computer program PRET automates this process by building a layer of artificial intelligence (AI) techniques around a set of traditional formal engineering methods. PRET takes a generate-and-test approach, using a small, powerful *meta-domain theory* that tailors the space of candidate models to the problem at hand. It then tests these models against the known behavior of the target system using a large set of more-general mathematical rules. The complex interplay of heterogeneous reasoning modes that is involved in this process is orchestrated by a special first-order logic system that uses static abstraction levels, dynamic declarative meta control, and a simple form of truth maintenance in order to test models quickly and cheaply. Unlike other modeling tools—most of which use libraries to model small, well-posed problems in limited domains and rely on their users to supply detailed descriptions of the target system—PRET works with nonlinear systems in multiple domains and interacts directly with the real world via sensors and actuators. This approach has met with success in a variety of simulated and real applications, ranging from textbook systems to real-world engineering problems.

Key words: Automated Model Building, System Identification, Qualitative Reasoning/Physics, Knowledge Representation/Reasoning Framework, Input-Output Modeling

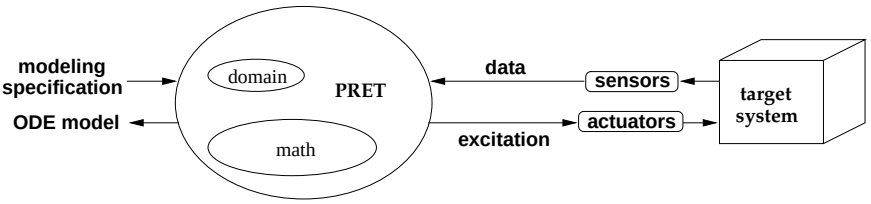

Fig. 1. PRET combines AI and formal engineering techniques to build ODE models of nonlinear dynamical systems. It uses domain-specific knowledge to build models and an encoded ODE theory to test them, and it interacts directly and autonomously with target systems using sensors and actuators.

1 Introduction

One of the most powerful analysis and design tools in existence—and often one of the most difficult to create—is a good model. Modeling is an essential first step in a variety of engineering problems. Faced with the task of designing a controller for a robot arm, for instance, a mechanical engineer performs a few simple experiments on the system, observes the resulting behavior, makes some informed guesses about what model fragments could account for that behavior, and then combines those terms into a model and checks it against the physical system. This model then becomes the mathematical core of the controller. Accuracy is not the only requirement; for efficiency reasons, engineers work hard to construct minimal models—those that ignore unimportant details and capture only the behavior that is important for the task at hand. 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"[63].

The computer program PRET, the topic of this paper, formalizes these intuitions and automates a coherent and useful part of this art. PRET is an automated tool for nonlinear system identification. Its inputs are a set of observations of the outputs of a black-box system, some optional hypotheses about the physics involved, and a set of tolerances within which a successful model must match the observations; its output is an ordinary differential equation (ODE) model of the internal dynamics of that system. See Figure 1 for a block diagram. PRET uses a small, powerful domain theory to build models and a larger, more-general mathematical theory to test them. It is de-

[1] Supported by NSF NYI #CCR-9357740, ONR #N00014-96-1-0720, and a Packard Fellowship in Science and Engineering from the David and Lucile Packard Foundation.

[2] Research performed while a research assistant at the University of Colorado at Boulder and during a postdoctoral fellowship at Stanford University funded by the German Academic Exchange Service (DAAD) "Gemeinsames Hochschulsonderprogramm III von Bund und Ländern."

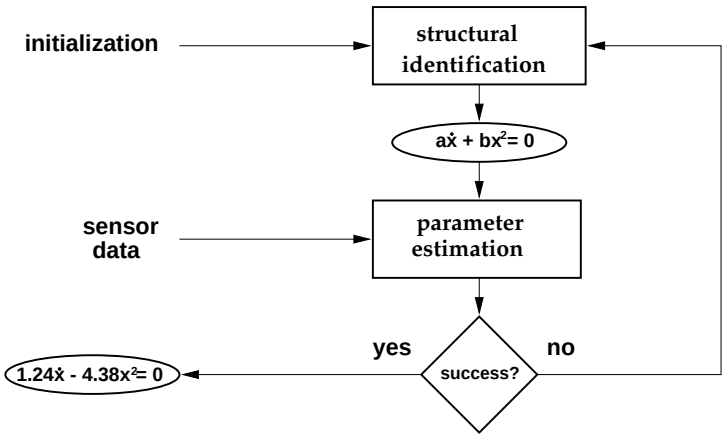

Fig. 2. The system identification (SID) process. Structural identification yields the general form of the model; in parameter estimation, values for the unknown coefficients in that model are determined. PRET automates both phases of this process.

signed to work in any domain that admits ODE models; adding a new domain is simply a matter of coding one or two simple domain rules. Its architecture wraps a layer of artificial intelligence (AI) techniques around a set of traditional formal engineering methods. This AI layer incorporates a variety of reasoning modes: qualitative reasoning, qualitative simulation, numerical simulation, geometric reasoning, constraint propagation, resolution, reasoning with abstraction levels, declarative meta control, and a simple form of truth maintenance. Models are represented using a component-based modeling framework, which accommodates different domains, adapts smoothly to varying amounts of domain knowledge, and allows expert users to create new domains easily. An input-output modeling subsystem allows PRET to observe target systems actively, manipulating actuators and reading sensors to perform experiments whose results augment its knowledge in a manner that is useful to the modeling problem that it is trying to solve. The entire reasoning process is orchestrated by a special first-order logic inference system, which automatically chooses, invokes, and interprets the results of the techniques that are appropriate for each point in the model-building procedure. This combination of techniques lets PRET shift fluidly back and forth between domain-specific reasoning, general mathematics, and actual physical experiments in order to navigate efficiently through an exponential search space of possible models.

In general, system identification proceeds in two interleaved phases: first, *structural identification*, in which the form of the differential equation is determined, and then *parameter estimation*, in which values for the coefficients are obtained. If structural identification produces an incorrect ODE model, no coefficient values can make its solutions match the sensor data. In this event, the structural identification process must be repeated—often using information about why the previous attempt failed—until the process converges to a solution, as shown diagrammatically in Figure 2. In linear physical systems,

structural identification and parameter estimation are fairly well understood. The difficulties—and the subtleties employed by practitioners—arise where noisy or incomplete data are involved, or where efficiency is an issue. See [55,60] for some examples. In *nonlinear* systems, however, both procedures are vastly more difficult—the type of material that is covered only in the last few pages of standard textbooks.

Unlike system identification software used in the control theory community, PRET is not just an automated parameter estimator; rather, it uses sophisticated reasoning techniques to automate the structural phase of model building as well. The basic paradigm is "generate and test." PRET first uses a small, powerful domain theory—the upper ellipse in Figure 1—to assemble combinations of user-specified and automatically generated ODE fragments into a candidate model. In a mechanics problem, for instance, the generate phase uses Newton's laws to combine force terms; in electronics, it uses Kirchhoff's laws to sum voltages in a loop or currents in a cutset. In order to test a candidate model, PRET performs a series of inferences about the model and the observations that the model is to match. This process is guided by two important assumptions: that abstract reasoning should be chosen over lower-level techniques, and that any model that cannot be proved wrong is right. PRET's inference engine uses an encoded mathematical theory (the lower ellipse in Figure 1) to search for contradictions in the sets of facts inferred from the model and from the observations. An ODE that is linear, for instance, cannot account for chaotic behavior; such a model should fail the test if the target system has been observed to be chaotic. Furthermore, establishing whether an ODE is linear is a matter of simple symbolic algebra, so the inference engine should not resort to a numerical integration to establish this contradiction. Like the domain theory, PRET's ODE theory is designed to be easily augmentable by an expert user.

To make these ideas more concrete, consider the spring/mass system shown at the right-hand side of Figure 3. To instruct PRET to build a model of this system, a user would enter the `find-model` call at the left of the figure. (PRET also has a GUI that leads users through this interaction without subjecting them to this syntax.) The `domain` statement instantiates the relevant domain theory; the next two lines inform PRET that the system has two point-coordinate state variables[3]. `Observations` are measured automatically by sensors and/or interpreted by the user; they may be symbolic or numeric and can take on a variety of formats and degrees of precision. For example, the first observation in Figure 3 informs PRET that the system to be mod-

[3] As described later in this paper, PRET uses a variety of techniques to infer this kind of information from the target system itself; to keep this example simple, we bypass those facilities by giving it the information up front.

```
(find-model
  (domain mechanics)
  (state-variables (<q1> <point-coordinate>) (<q2> <point-coordinate>))
  (observations
    (autonomous)
    (oscillation <q1>)
    (oscillation <q2>)
    (numeric (<time> <q1> <q2>)
             ((0 .1 .1) (.1 .109 .110)...)))
  (hypotheses
    (<force> (* k1 <q1>))
    (<force> (* k2 (- <q1> <q2>)))
    (<force> (* k3 <q2>))
    (<force> (* m1 (deriv (deriv <q1>))))
    (<force> (* m2 (deriv (deriv <q2>))))
    (<force> (* r1 (deriv <q1>)))
    (<force> (* r2 (square (deriv <q1>))))
    (<force> (* r3 (deriv <q2>)))
    (<force> (* r4 (square (deriv <q2>)))))
  (specifications
    (<q1> relative-resolution 1e-2 (-infinity infinity))
    (<time> absolute-resolution 1e-6 (0 120))))
```

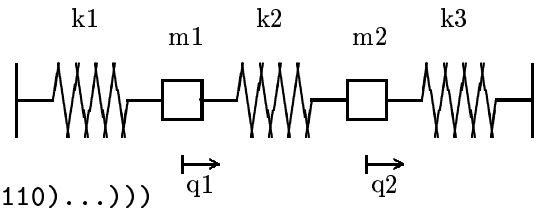

Fig. 3. Modeling a simple spring/mass system. In this example call to PRET, the user first sets up the problem, then makes five observations about the position coordinates q_1 and q_2, hypothesizes nine different force terms, and finally specifies resolution and range criteria that a successful model must satisfy. Angle brackets (e.g., `<time>`) identify state variables and other special keywords that play roles in PRET's use of its domain theory.

eled is autonomous[4]; the second states that the state variable q_1 oscillates. `Numeric` observations are physical measurements made directly on the system. An optional list of `hypotheses` about the physics involved—e.g., a set of ODE terms ("model fragments") that describe different kinds of friction—may be supplied as part of the `find-model` call; 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. The ones at the end of Figure 3, for instance, require any successful model to match q_1 to within 1% and one microsecond over the first 120 seconds of the system's evolution. It should be noted that this spring/mass example is representative neither of PRET's power nor of its intended applications. Linear systems of this type are very easy to model[5,60]; no engineer would use a software tool to do generate-and-test and guided search on such an easy problem. We chose this simple system to make this presentation brief and clear.

To construct a model from the information in this `find-model` call, PRET uses

[4] That is, it does not explicitly depend on time.

the `mechanics` domain rule `(point-sum <force> 0)` that is encoded in its knowledge base to combine hypotheses into an ODE. (We will use `teletype` font in the body of this paper to identify terms that play roles in a user's interaction with PRET.) In the absence of any domain knowledge—omitted here, again, to keep this example short and clear—PRET simply selects the first hypothesis, producing the ODE $k_1 q_1 = 0$. This candidate model then passes to the test phase for comparison against the observations. The model tester, implemented as a custom first-order logic inference engine[77], uses a set of general rules about ODE properties to draw inferences from the model and from the observations. In this case, a SCHEME[5] function called on the ODE $k_1 q_1 = 0$ establishes the fact `(order <q1> 0)`, which expresses that the highest derivative of q_1 in this model is zero. Reasoning from this fact and the `(oscillation <q1>)` observation in the `find-model` call, PRET uses the following two rules from its ODE theory to establish a contradiction:

```
(<- (not-oscillation (var StateVar))
    ((linear-system)
     (autonomous (var StateVar))
     (order (var StateVar) (var N))
     (< (var N) 2)))

(<- (falsum)
    ((oscillation (var StateVar))
     (not-oscillation (var StateVar))))
```

Rules are represented declaratively using a logic-based formalism; each implication is a generalized Horn clause[9], written—following SCHEME convention—in prefix notation. A clause `(<-` *head body*`)` has the usual meaning: the *head* is implied by the conjunction of the formulae in the *body*; `falsum` is the formula that represents inconsistency. The first rule expresses that a state variable x_i of a linear system does not oscillate if its order (i.e., the highest derivative that appears in the model) is less than two and the physical system is autonomous; the second simply states that no state variable can be oscillatory and non-oscillatory at the same time. The way PRET handles this first candidate model demonstrates the power of its abstract-reasoning-first approach; only a few steps of inexpensive qualitative reasoning suffice to let it quickly discard the model.

PRET tries all combinations of `<force>` hypotheses at single point coordinates, but all these models are ruled out for qualitative 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

$$k_1 q_1 + m_1 \ddot{q}_1 = 0$$

[5] PRET is written in SCHEME[69].

$$m_2\ddot{q}_2 = 0$$

PRET cannot discard this model by purely qualitative means, so it invokes its nonlinear parameter estimation reasoner (NPER), which uses knowledge derived in the structural identification phase to guide the parameter estimation process (e.g., choosing good approximate initial values and thereby avoiding local minima in regression landscapes)[16]. The NPER finds no appropriate values for the coefficients k_1, m_1, and m_2 such that any ODE solution matches the numeric time series, so this candidate model is also ruled out. This, however, is a far more expensive proposition than the simple contradiction proof of the fact `(order <q1> 0)`—roughly five minutes of CPU time, as compared to a fraction of a second—which is exactly why PRET's inference guidance system is set up to use the NPER only as a last resort, after all of the more-abstract reasoning tools in its arsenal have failed to establish a contradiction.

After having discarded a variety of unsuccessful candidate models via similar procedures, PRET eventually tries the model

$$k_1 q_1 + k_2(q_1 - q_2) + m_1\ddot{q}_1 = 0$$
$$k_3 q_2 + k_2(q_1 - q_2) + m_2\ddot{q}_2 = 0$$

Again, it calls the NPER, 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. The difference between the integration and the observation stays within the specified resolution, so the numeric comparison yields no contradiction and this candidate model, together with its parameter values, is returned as the answer[6]. If the list of user-supplied hypotheses is exhausted before a successful model is found, PRET generates hypotheses automatically using power-series expansions on the state variables—the standard engineering fallback in this kind of situation.

The technical challenge of this model-building process is efficiency; the search space is huge, and so PRET must choose promising model components, combine them intelligently into candidate models, and identify contradictions as quickly and simply as possible. Simple hypothesis enumeration would create a combinatorial explosion. This profoundly influenced the design goals for both phases of the model-building process. In particular, PRET's *generate* phase must exploit all available domain-specific knowledge insofar as possible. A modeling domain that is too small may omit a key model; an overly general domain has a prohibitively large search space. Appropriate use of domain knowledge lets PRET tailor the search space to the problem by classifying model and system behavior at the highest possible abstraction level. As demonstrated in the example above, high-level techniques like symbolic algebra can be used

[6] If more than one adequate model exists, PRET returns the first one it encounters.

to remove huge branches from the search space; knowledge that the target system oscillates, for instance, lets PRET quickly rule out any autonomous linear ODE model of order less than two. In other situations, pruning a leaf off the tree of possible models can be extremely expensive (e.g., estimating parameter values for a nonlinear ODE prior to a final corroborative simulation/comparison run). Efficient search, then, requires rapid, accurate selection of the appropriate reasoning mode—a difficult, dynamic problem that depends on how much PRET knows about the target system at any given stage of the model-building process. Judicious use of domain-specific knowledge is also important to speeding the model *testing* phase. Some analysis methods—such as creep tests in viscoelastic systems, for example—are extremely powerful, but only apply in specific domains. Other methods, such as phase-portrait analysis, apply to all dynamical systems, but are more general and arguably less powerful. To effectively build and test models of nonlinear systems, PRET must determine which methods are appropriate to a given situation, invoke and coordinate them, and interpret their results.

Orchestrating this subtle and complex reasoning process is a difficult problem. PRET's solution rests on carefully crafted knowledge representation frameworks, described in the following section, that allow for an elegant formalization of the essential building blocks of an engineer's knowledge and reasoning, and powerful automated reasoning machinery, described in Section 3, that uses the formalized knowledge to reason flexibly about a variety of modeling problems. The input-output modeling techniques described in Section 4—also omitted from the simplified example in Figure 3—allow PRET to autonomously explore the relationship between the inputs and outputs of a target system, and to reason about multiple behavioral regimes. Working in concert, these methods allow PRET to construct accurate, parsimonious models of the internal dynamics of nonlinear systems in any domain that admits ODE models, ranging from toy problems like Figure 3 to difficult real-world applications. Section 5 covers three practical engineering examples—a vehicle suspension, a water resource system, and a parametrically forced pendulum—in some detail, and summarizes results on several other applications, including a radio-controlled (R/C) car. In all of these cases, good models are crucial not only to the understanding of the physics of the system, but also to the process of engineering design. The core of a controller designed to direct the behavior of an R/C car, for instance, is an ODE model of the device—information that is absent from a Radio Shack spec sheet. Similarly, decision support for water resource systems depends critically on knowledge about how changes (e.g., rainfall) propagate through the system, which is most effectively captured by an ODE model, and the driven pendulum is the basic mechanical element in many modern robotics systems. As an AI tool that automates the model building process, PRET has many possible implications for and roles in the practice of science and engineering: as a means of corroborating and/or evaluating 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 work described in this paper generally falls into the active AI research area of automated model building and reasoning about physical systems. See, for example, [3,35,38,67,85,86]. Because of the highly interdisciplinary nature of the contents of this article, there are also important relations to several other fields and disciplines. We have chosen to distribute the related work discussion among the appropriate subsections, rather than gather it into a separate section.

2 Representations for Model Building

A central problem in any automated modeling task is that the size of the search space is exponential in the number of model fragments unless severe restrictions are placed on the model-building process. Ideally, one would like to build *black-box* models using general reasoning techniques that applied to *any* system and did not require any *domain knowledge* about the system under examination. The combinatorics of the generate phase make this paradigm unrealistic. Most AI modeling work has taken a *clear-box* modeling approach, in which one knows almost everything about what one is trying to model. This is unrealistic for engineering practice. A good compromise is *gray-box* modeling, where partial information about the internals of the box—e.g., whether the system is electronic or viscoelastic—is used to prune the search space down to a reasonable size. The key to making gray-box modeling of nonlinear dynamical systems practical is a flexible knowledge representation scheme that adapts to the problem at hand. Domain-dependent knowledge can drastically reduce the search-space size, but its applicability is fundamentally limited. The challenge in balancing these influences is to be able to determine, at every point in the reasoning procedure, what knowledge is applicable and useful.

Our solution, termed *component-based modeling* or CBM, combines a representation that allows for different levels of domain knowledge, a set of reasoning techniques appropriate to each level, and a control strategy that invokes the right technique at the right time. In particular, we combine ideas from *generalized physical networks*[73], a meta-level representation of idealized two-terminal elements, with traditional compositional model building[35] and qualitative reasoning[84]. The intent is to span the spectrum between highly specific frameworks that work well in a single, limited domain (e.g., a spring/dashpot vocabulary for modeling simple mechanical systems) and abstract frameworks that rely heavily upon general mathematical formalisms at the expense of huge search space sizes (e.g., [17]).

In the late 1950s and early 1960s, inspired by the realization that the principles underlying Newton's third law and Kirchhoff's current law were identical [7], researchers began combining multi-port methods from a number of engineering fields into a generalized engineering domain with prototypical components[68]. The basis of this *generalized physical networks* (GPN) paradigm is that the behavior of an ideal two-terminal element—the "component"—may be described by a mathematical relationship between two dependent variables: generalized flow and generalized effort, where *flow(t) * effort(t) = power(t)*. This pair of variables manifests differently in each domain: (*flow*, *effort*) is (*current*, *voltage*) in an electrical domain and (*force*, *velocity*) in a mechanical domain. In bond graphs[56], another generalized representation paradigm that has seen some use in the AI modeling literature, velocity is a flow variable and force is an effort variable. The only difference between GPNs and bond graphs is a frame-of-reference shift. While bond graphs are a good alternative to generalized physical networks—especially if causality issues are a concern—converting them into ODE models is difficult, which makes them less useful for the kinds of complex nonlinear modeling tasks that we address in this paper.

Table 1
Example component representations.

Component:	Proportional	Differentiating	Integrating
General	$e = Bf$	$f = C\frac{de}{dt}$	$e = A\frac{df}{dt}$
Electrical	$v = Ri$	$i = C\frac{dv}{dt}$	$v = L\frac{di}{dt}$
Mechanical	$v = Bf$	$f = M\frac{dv}{dt}$	$v = K\frac{df}{dt}$

The GPN representation has three important advantages for model building. Firstly, its two-port nature makes it easy to incorporate sensors and actuators as integral parts of a model. For example, a sinusoidal current source often has an associated impedance that creates a loading effect on the rest of the circuit. With a network approach, these effects naturally become part of the model, just as they do in real systems. Secondly, GPNs bring out similarities between components and properties in different domains. Electrical resistors ($v = iR$) and mechanical dampers ($v = fB$) are physically analogous; both dissipate energy in a manner that is proportional to the operative state variable. Both of these components can be represented by a single GPN component that incorporates a *proportional* relationship between the flow and effort variables. Two other useful GPN components instantiate *integrating* and *differentiating* relationships; the representation also allows for flow and effort source components, as shown in Table 1. See [56] or [73] for additional domains and

[7] Summation of {forces, currents} at a point is zero, respectively; both are manifestations of the conservation of energy.

components. Thirdly, the GPN representation makes it very easy to incorporate varying amounts and levels of information. This is closely related to its ability to capture behavioral analogs. Both of the networks in Figure 4, for example, can be modeled by a series integrating/proportional/differentiating GPN; knowledge that the system is electronic or mechanical would let one refine the model accordingly (to a series RLC circuit or mass-spring-damper system, respectively). The available domain knowledge, then, can be viewed as a lens that expands upon the internals of some GPN components, selectively sharpening the model *in appropriate and useful ways*.

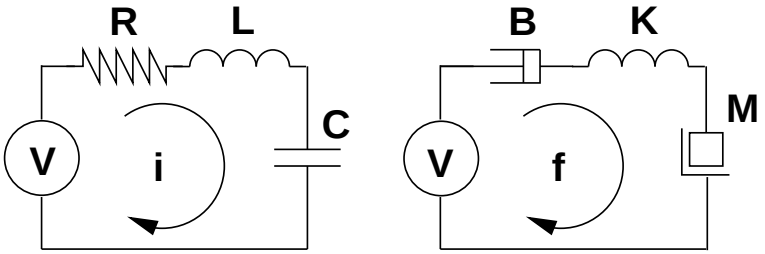

Fig. 4. Two systems that are described by the same GPN model: (a) a series RLC circuit and (b) a series mass-spring-damper system. **V** is a voltage source in (a) and a velocity source in (b).

PRET currently incorporates five specific GPN-based modeling domains: `mechanics`, `viscoelastics`, `linear-electronics`, `linear-rotational`, and `linear-mechanics`. Domains are constructed by domain experts, stored in the domain-theory knowledge base, and instantiated by the `domain` line of the `find-model` call. Each consists of a set of component primitives and a framework for connecting those components into a model. The basic `linear-electronics` domain, for example, was built by an electrical engineer; it comprises the components {`linear-resistor, linear-capacitor`}, the standard parallel and series connectors, and some codified notions of model equivalence (e.g., Thévenin). Specification of state variables for different domains—type, frames of reference, etc.—is a nontrivial design issue. In the `mechanics` domain, a body-centered inertial reference frame is assumed, together with coordinates that follow the formulation of classical mechanics[44], which assigns one coordinate to each degree of freedom, thereby allowing all equations to be written without vectors. The representation described in this section is designed to handle the coordinate issues associated with the remaining domains. Finally, these modeling domains are dynamic: if a domain does not contain a successful model, it automatically expands to include additional components and connections. This procedure is also described in the following section.

If a user wants to apply PRET to a system that does not fall in an existing domain, he or she can either build a new domain from scratch—a matter of making a list of components and connectors—or use one of PRET's

meta-domains: general frameworks that arrange hypotheses into candidate models by relying on modeling techniques that transcend individual application domains. The `xmission-line` meta-domain, for instance, generalizes the notion of building models using an iterative pattern, similar to a standard model of a transmission line, which is useful in modeling distributed parameter systems. The `linear-plus` meta-domain takes advantage of fundamental linear-systems properties that allow the linear and nonlinear components to be treated separately under certain circumstances, which dramatically reduces the model search space. Both can be used directly or customized for a specific application domain, as demonstrated in Section 5. We chose this particular pair of meta-domains as a good initial set because they cover such a wide variety of engineering domains. We are exploring other possibilities, especially for the purposes of modeling nonlinear networks.

Choosing a modeling domain for a given problem is not trivial, but it is not a difficult task for the practicing engineers who are PRET's target audience. Such a user would first look through the existing domains to see if one matched his or her problem. If none were appropriate, he or she would choose a meta-domain based upon the general properties of the modeling task. If there is a close match between the physical system's components and the model's components (i.e., it is a lumped parameter system), then `linear-plus` is appropriate; `xmission-line` is better suited to modeling distributed parameter systems. There is significant overlap between the various domains and meta-domains; a linear electronic circuit can be modeled using the specific `linear-electronics` domain, the `xmission-line` meta-domain, or the `linear-plus` meta-domain. In all three cases, PRET will produce the same model, but the amount of effort involved will be very different. The advantage of the `linear-electronics` domain is its specialized, built-in knowledge about linear electrical circuits, and the effect of this knowledge is to focus the search. A capacitor in parallel with two resistors, for instance, is equivalent to a single resistor in parallel with that capacitor. The `linear-electronics` domain "knows" this, allowing it to avoid duplication of effort; the two meta-domains do not. Perhaps most important of all, their generality and overlap make the meta-domains particularly helpful if one does not know exactly what kind of system one is dealing with, which is not an uncommon situation in engineering practice.

There are a variety of ways to use generalized physical networks to help automate PRET's structural identification phase. One could create a library of GPN components and test each possible combination until a valid model is found. This method is obviously impractical, as simple enumeration creates an exponential search space—a severe problem if the component library is large, as must be the case if one is attempting to model nonlinear systems [8]. A more-

[8] Nonlinear terms are somewhat idiosyncratic, and each would have to be supplied

intelligent method is to use a hierarchy of domain-dependent and -independent knowledge to direct the search, as described next.

2.2 The CBM Paradigm: Reasoning

The GPN representation is an effective basis for dynamic modeling domains whose complexity naturally rises and falls according to the available information about the target system. A general domain—e.g., the set of all dynamical systems—has a complex search space; a specific domain like the set of conservative mechanical systems has a much smaller one. The challenge in reasoning about GPN models is to tailor the reasoning to the knowledge level in such a way as to prune the search space to the minimum. Organizing domains into a hierarchy of generality—as shown in Figure 5—is not enough; what is needed is a hierarchical set of analysis tools, as well as a means for assessing the situation and choosing which tool is appropriate.

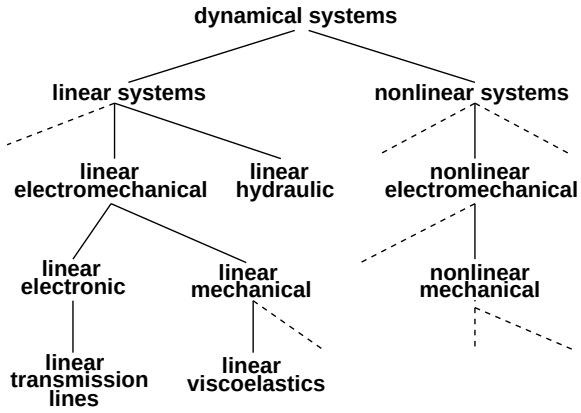

Fig. 5. A hierarchy of model building domains.

Focused, appropriate analysis is critical to the efficiency of the automated model building process. As demonstrated in the spring/mass example of Figure 3, invalid models can often be ruled out using purely qualitative information, rather than expensive point-by-point numerical comparisons. The challenge in designing the component-based modeling paradigm was to come up with a framework that supported this kind of reasoning. The key idea is that different analysis techniques are appropriate in different domains, and our solution combines a structured hierarchy of analysis tools, part of which is shown in Table 2, with a scheme that lets the GPN component type and domain knowledge dictate which tools to use. *Cell dynamics* is a geometric reasoning technique that classifies a phase portrait qualitatively using simple

as a separate library entry. This issue has not arisen in previous work on GPNs because their use has been generally confined to linear systems.

discretized heuristics. *Delay-coordinate embedding* lets one infer the dimension and topology of the internal system dynamics from a time series measured by a *single* output sensor. Nonlinear time-series analysis is a blanket term for classification that follows the {`attractors, bifurcations, ...`} ontology of nonlinear dynamics. Linear systems analysis refers to the techniques taught to undergraduate engineers (pole-zero diagrams, step response, etc.). Analysis tools for restricted linear systems—e.g., creep testing—are highly domain-specific. Tools at any level of the table apply at all lower levels as well. See Section 4.1 for further details.

Table 2
Component type and domain knowledge dictate what analysis tools PRET should use to build and test models. Tools higher in this table are more general but their results can be less powerful.

System Type	**Analysis Tools**
Nonlinear	Cell dynamics[53]
	Delay-coordinate embedding[74]
	Nonlinear time series analysis[14,80]
Linear	Linear system analysis[70]
Restricted linear	Domain dependent (e.g., [21])

Reasoning about model-building proceeds in the obvious manner, given this hierarchy: if no domain knowledge about the target system is available (i.e., the true "black box" situation), then models are constructed using general reasoning techniques and analysis tools that apply to *all* ODEs—those in the top line of Table 2. This highly general approach is computationally expensive but universally applicable. If the system is known to be linear, the extensive and powerful repertoire of linear analysis tools developed over the last several decades makes the model building and testing tasks far less imposing. Moreover, system inputs (drive terms) in linear systems appear verbatim in the resulting system ODE, which makes input/output analysis much easier, as described in Section 5. In more-restricted domains, analysis tools are even more specific and powerful. In viscoelastics, for example, three qualitative properties of a "strain test" reduce the search space of possible models to linear[21].

Given all of this machinery, PRET's generate phase builds models as follows. First, a candidate GPN model is constructed using the basic components and connectors of a particular domain. This process is guided by the analysis tools in Table 2. If the system is nonlinear, for example, the cohort of nonlinear tools is applied to the sensor data to determine the dimension d of the dynamics; this fact allows PRET to automatically disregard all models of order $< d$. Other nonlinear analysis techniques yield similar search-space reductions. If the system is linear, many more tools apply; these tools are cheaper and more powerful than the nonlinear tools, and so the CBM framework guides

PRET to use the former before the latter. Knowledge that the target system is oscillating, for example, not only constrains any autonomous linear model to be of least second order, but also implies some constraints on its coefficients; this reasoning is purely symbolic and hence very inexpensive.

The generate and test phases are interleaved, as shown in Figure 2: if the generate phase's first-cut search space does not contain a model that matches the observed system behavior, the GPN modeling domain dynamically expands to include more-esoteric components. As described in Section 2.1, for example, the `linear-electronics` domain begins with the components {`linear-resistor`, `linear-capacitor`}. If all models in this search space are rejected, PRET automatically expands the domain to include the component type `linear-inductor`. The intuition captured by this notion of "layered" domains is that inductors are much less common, in practical engineering systems, than capacitors. Expanding domains beyond linear components is more difficult, since the number of possible component types increases dramatically and any ordering scheme necessarily becomes somewhat *ad hoc*. In many engineering domains, however, there exist well-defined categorization schemes that help codify this procedure. Tribology texts, for example[9], specify different kinds of friction for different kinds of ball bearings, as well as some notions about which of those are common and should be tried first, and which are rare and esoteric[46]. The CBM paradigm is designed to let an expert user—a "domain builder"—encode this kind of information quickly and easily, and to let PRET exploit that knowledge insofar as possible during the model-building process. Because a domain expert should not be required to have a detailed understanding of the internal structure of the program or a working knowledge of Scheme, PRET provides a simple construct for specifying the structure of this information: a natural number that prioritizes possible components. The meta-domains also simplify this process; customizing a domain can be as simple as adding a few components to a meta-domain. The `viscoelastics` domain, for instance, is an instantiation of the `xmission-line` meta-domain with a proportional and a differentiating element in series[21]. See Section 5 for more examples.

Once a GPN network is generated, the final step in the model-building process is to convert it into ODE format. This conversion step, which rests upon powerful network-theoretic principles that have been in the engineering vernacular for many decades, is fairly easy to automate. In particular, PRET uses loop and node equations to convert a GPN network with unspecified component values into an ODE (also with unspecified component values). The details of the conversion algorithm are described elsewhere[30].

[9] Tribology is the science of surface contact.

The component-based modeling paradigm's hierarchy of qualitative and quantitative reasoning tools, which relate observed physical behavior and model form, coupled with its generalized physical network-based representation, provide the flexibility required for gray-box modeling of nonlinear dynamical systems. This type of reasoning, wherein the modeling tool has only partial knowledge of the internals of the target system, accurately reflects the abstraction levels and reasoning processes used effectively by engineers during the system identification procedure. The GPN representation is powerful enough to describe a wide range of systems. It naturally captures similarities (e.g., between mechanical damping and electronic resistance) and it adapts smoothly to different levels of domain knowledge: the same GPN model can be abstract and general or highly specific, depending on how much one knows about the system. This reduces PRET's search space by allowing it to work with GPN components—and the corresponding abstract, qualitative reasoning techniques—as long as possible: up until the point when it converts the GPN into a set of ODEs. Coupled with the domain and meta-domain facilities described in Section 2.2, the CBM paradigm makes creating a domain simple: an expert need only specify the prototypical components and connections. Optionally, he or she can also specify efficient model construction techniques and data analysis tools for different situations, and prescribe a set of rules to help identify the correct model quickly. This layered domain/meta-domain structure makes it easy to apply PRET to new problems (e.g., the human ear, which we are currently modeling with the `xmission-line` meta-domain). The CBM paradigm also helps naïve users in that it allows hypotheses to take the form that they do in engineering and physical sciences textbooks. This makes interacting with PRET very natural; the user only needs to know the domain and its components, not the physics of their function, interaction, and composition. Finally, the two-port nature of the GPN representation lets PRET incorporate sensors and actuators as integral parts of the model—an essential part of its solution to the input/output modeling, as described in Section 4.

The novelty and utility of component-based modeling lie in its use of meta-level, two-terminal components for automated modeling of nonlinear systems. Previous work in the AI community using meta-level components similar to GPNs has typically been restricted to reasoning about causality[82] and modeling hybrid systems[64]. Amsterdam's work on automated model construction in multiple physical domains[4] is an exception to this, but it is limited to linear systems of order two or less. Capelo *et al.*[21] build ODE models of linear viscoelastic systems by evaluating time series data using qualitative reasoning techniques. Although these goals are similar to PRET's, the library of possible models is more restricted: it involves only two component types (linear springs and dashpots). CBM in PRET is much more general; not only does it use a

large and rich set of meta-level components for automated model building, but it also supports easy user customization of these components and domains.

3 Orchestrating Reasoning about Models

As described in the previous section, PRET uses component-based representations, user hypotheses, and domain knowledge to generate candidate models of a given target system. In this section, we describe the reasoning framework in which PRET tests such a model against observations of the target system. Like a human expert, PRET makes use of a variety of reasoning techniques at various abstraction levels during the course of this process, ranging from detailed numerical simulation to high-level symbolic reasoning. These modes and their interactions are described in Section 3.1. The challenge in designing PRET's model tester was to work out a formalism that met two requirements: first, it had to facilitate easy formulation of the various reasoning techniques; second, it had to allow PRET to reason about which techniques are appropriate in which situations. In particular, reasoning about both physical systems and candidate models should take place at an abstract level first and resort to more-detailed reasoning later and only if necessary. To accomplish this, PRET judges models according to the opportunistic paradigm "valid if not proven invalid:" if a model is bad, there must be a reason for it. Or, conversely, if there is no reason to discard a model, it is a valid model. PRET's central task, then, is to quickly find inconsistencies between a candidate model and the target system. Section 3.2 describes the reasoning control techniques that allow it to do so.

3.1 Reasoning Modes

PRET's test phase uses six different classes of techniques in order to test a candidate model against a set of observations of a target system:

- qualitative reasoning,
- qualitative simulation,
- constraint reasoning,
- geometric reasoning,
- parameter estimation and
- numerical simulation.

In our experience (see Section 5), this set of techniques, described in the following five subsections, provides PRET with the right tools to quickly test models against the given observations. Parameter estimation and numerical

simulation are low-level, computationally expensive methods that ensure that no incorrect model passes the test. Intelligent use of the other, more-abstract techniques in this arsenal allows PRET to avoid these costly low-level techniques in most cases; most candidate models can be discarded by purely qualitative techniques or by semi-numerical techniques in conjunction with constraint reasoning. Section 3.2 describes how PRET uses meta control techniques to exploit this by deciding which of these modes is most appropriate for each part of the model test phase.

3.1.1 Qualitative Reasoning

Reasoning about abstract features of a physical system or a candidate model is typically faster than reasoning about their detailed properties. Because of this, PRET uses a "high-level first" strategy: it tries to rule out models by purely *qualitative* techniques[28,37,40,84] before advancing to more-expensive semi-numerical or numerical techniques. Often, only a few steps of inexpensive qualitative reasoning (QR) suffice to quickly discard a model. Some of PRET's qualitative rules, in turn, make use of other tools, e.g., symbolic algebra facilities from the commercial package MAPLE[23]. For example, PRET's encoded ODE theory includes the qualitative rule that nonlinearity is a necessary condition for chaotic behavior:

```
(<- (falsum)
    ((linear-system)      ;; ode is linear
     (chaotic)))          ;; target system is chaotic
```

This lets any linear model be discarded without performing more-complex operations[10] such as, for example, a numerical integration of the ODE. PRET's QR facilities are not only important for accelerating the search for inconsistencies between the physical system and the model; they also allow the user to express incomplete information[58]. For example, the user might not know the exact value of a friction coefficient, but he or she might know that it is constant and positive. This is useful not only in isolation, but in conjunction with the constraint reasoning mode, as described later in this section.

3.1.2 Qualitative Simulation

After using its qualitative reasoning facilities to the fullest possible extent and before resorting to the numerical level, PRET attempts to establish contradictions by reasoning about the states of the physical system[58]. It does not do

[10] Determining whether or not an ODE is linear involves calculation of the Jacobian, which is a simple symbolic operation that PRET accomplishes via a single call to MAPLE, as shown on page 22.

full qualitative simulation[57]; rather, it envisions the state space of all possible combinations of qualitative values of state variables and parameters. Specifically, PRET's qualitative envisioning module constrains the possible ranges of parameters in the candidate model. If the constraints become inconsistent—i.e., the range of a parameter becomes the empty set—the model is ruled out. Currently, the qualitative states contain only sign information $(-, 0, +)$. For example, for the model $ax + by = 0$, the state $(x, y) = (+, +)$ constrains (a, b) to the possibilities $(+, -)$ or $(0, 0)$ or $(-, +)$. This strategy is faster than full qualitative simulation, but it is also less accurate; it may let invalid models pass the test, but these models will later be ruled out by the numeric simulator. However, for the models that do fail the qualitative envisioning test, this test is much cheaper than a numeric simulation and point-by-point comparison would be.

3.1.3 Constraint Reasoning

Often, information *between* the purely qualitative and the purely numeric levels is also available. If a linear system oscillates, for example, the imaginary parts of at least one pair of the roots of its model's characteristic polynomial must be nonzero. If the oscillation is damped, the real parts of those roots must also be negative. Thus, if the model $a\ddot{x} + b\dot{x} + cx = 0$ is to match an `damped-oscillation` observation, the coefficients must satisfy the inequalities $4ac > b^2$ and $b/a > 0$. PRET uses expression inference[81] to merge and simplify such constraints[54]. However, this approach works only for linear and quadratic expressions and some special cases of higher order, but the expressions that arise in model testing can be far more complex. For example, if the candidate model $\ddot{x} + a\dot{x}^4 + b\dot{x}^2 = 0$ is to match an observation that the system is conservative, the coefficients a and b must take on values such that the divergence $-4a\dot{x}^3 - 2b\dot{x}$ is zero, below a certain resolution threshold, for the specified range of interest of x. We are investigating techniques (e.g., [36]) for reasoning about more-general expressions like this.

3.1.4 Geometric Reasoning

Other qualitative forms of information that are useful in reasoning about models are the geometry and topology of a system's behavior, as plotted in the time or frequency domain, state space, etc. A bend of a certain angle in the frequency response, for instance, indicates that the ODE has a root at that frequency, which implies algebraic inequalities on coefficients, much like the facts inferred from the `damped-oscillation` above; asymptotes in the time domain have well-known implications for system stability, and state-space trajectories that cross imply that an axis is missing from that space. In order to incorporate this type of reasoning, PRET processes the `numeric` observations—curve

fitting, recognition of linear regions and asymptotes, and so on—using MAPLE functions[23] and simple phase-portrait analysis techniques[13], producing the type of abstract information that its inference engine can leverage to avoid expensive numerical checks. These methods, which are used primarily in the analysis of sensor data[15], are described in more detail in Sections 2.2 and 4. PRET does not currently reason about topology, but we are investigating how best to do so[71,72].

3.1.5 Parameter Estimation and Numerical Simulation

PRET's final check of any model requires a point-by-point comparison of a numerical integration of that ODE against all numerical observations of the target system. In order to integrate the ODE, however, PRET must first estimate values for any unknown coefficients. Parameter estimation, the lower box in Figure 2, is a complex nonlinear global optimization problem[51,83]. Given an ODE with unknown coefficients and a possibly noisy time-series observation of some small subset of its state variables, a parameter estimator must find values for the unknown parameters of the ODE [11] . In linear physical systems, this procedure is fairly well understood. In *nonlinear* systems, however, it is vastly more difficult; linear signal processing methods do not apply, so PRET must fall back on regression, and nonlinear regression landscapes typically exhibit local extrema that can trap numerical methods.

PRET's nonlinear parameter estimation reasoner (NPER) solves this problem using a new, highly effective global optimization method that combines qualitative reasoning (QR) and local optimization techniques[16]. The method that lies at the core of the NPER is ODRPACK[11,12], a robust nonlinear least-squares solver. Around this core is built a layer of QR techniques that allow PRET to automatically interact with and exploit ODRPACK's unique and powerful features. Using qualitative observations—provided by the user or inferred from other observations via any of the reasoning modes described in the previous subsections—the NPER's QR layer can, for instance, intelligently choose starting values for the unknown coefficients, helping ODRPACK avoid local extrema in the parameter landscape. QR can be used to determine cutoff frequencies for filtering algorithms, so noise can be removed without disturbing the data's structure. The NPER also uses QR to interpret ODRPACK's results on an abstract level—quickly and yet correctly. This demonstrates the importance and power of interaction among the various reasoning modes. A qualitative result in the NPER about the sign of a parameter or a constraint on a product (e.g., $b^2 > 4ac$) can be used by the constraint reasoner, and vice versa. The truth maintenance facilities of PRET's logic inference system,

[11] Note that this is not simply a curve-fitting problem; it actually amounts to inverting methods like Runge-Kutta.

described in the following section, were designed to facilitate exactly this type of interaction.

3.2 Control of Reasoning

A model should be ruled inconsistent if its mathematical properties conflict with any known observation of the target system. The reasoning modes described in the previous section play different roles in the search for inconsistency; PRET's challenge in orchestrating them properly was to test models against observations using the cheapest possible reasoning mode and, at the same time, avoid duplication of effort. In order to accomplish this, the inference engine uses the following techniques to represent and reason with knowledge about the target system and about candidate models, and to guide the reasoning process by choosing and invoking the appropriate modes and interpreting their results:

- SLD-based resolution,
- declarative dynamic meta-level control,
- a hierarchy of abstraction levels, and
- a simple form of truth maintenance.

3.2.1 SLD-based Resolution

As is common for AI programs, PRET's knowledge representation and reasoning formalism is declarative: both object-level and meta-level knowledge are represented as first-order logical formulae. The language in which observations and the ODE theory are expressed is that of generalized Horn clause intuitionistic logic (GHCIL)[62]. Roughly, GHCIL clauses are Horn clauses that also allow embedded implications in their bodies. The special atomic formula `falsum`—which means inconsistency—may only appear as the head of a clause. Such clauses are often called *integrity constraints*; they express fundamental reasons for inconsistencies, e.g., that a system cannot be oscillating *and* non-oscillating at the same time (cf., the second rule on Page 6). PRET's search for an inconsistency between the observations and a particular candidate model amounts to an attempt to prove `falsum`. A model is ruled out if and only if a contradiction exists between a mathematical property of the physical system (e.g., the `(oscillation <x>)` observation of Figure 3) and a mathematical property of the model (e.g., the fact `(no-oscillation <x>)`, derived via the ODE theory from a root-locus analysis of some candidate model). Therefore, proving `falsum` is the critical mechanism in the model test procedure: if PRET can derive `falsum` from the union of the observations and facts about a candidate model, then that model is ruled out. This concept of

negation as inconsistency[41] is the only form of negation in our paradigm. Negation as failure, which is the standard form of negation in PROLOG[76], is particularly undesirable for our purposes. Since we do not require the user to supply all possible [12] observations, the absence of knowledge cannot be used to generate new knowledge.

PRET's inference engine is an SLD resolution[61]-based theorem prover. For every candidate model, this prover combines basic facts about the target system, basic facts about the candidate model, and basic facts and rules from the ODE theory into one set of clauses, and then tries to derive `falsum` from that set. The basic facts about the system are the `observations` from the `find-model` call. The ODE theory comprises several dozen rules like those on Pages 6 and 18. The basic facts about the current model are obtained by a collection of SCHEME "model observer" functions that identify mathematical properties of the model, e.g., that it is linear, of third order, etc. The following model observer function, for example, called on a candidate model `the-model`, makes a call to the MAPLE `jacobian` function, and returns the fact `(linear-system)` if no system state variable appears in the resulting matrix.

```
(define (linear-system? the-model state-variables)
  (let ((model-jacobian (jacobian the-model)))
    (not (any-variable-in-matrix? state-variables model-jacobian))))
```

Together with the ODE theory, these model observer functions, which are typically non-logic-based, implement the basic operations found in any differential equations text. Because the inference engine cannot invoke other functions directly, however, the implementer of the ODE theory must use the special logical predicate `scheme-eval` to call upon them. Whenever the inference engine attempts to prove a goal with this predicate, the corresponding function is invoked automatically. The `scheme-eval` predicate also provides the link to all modules that implement other non-logical reasoning modes, such as the constraint reasoner and the parameter estimator. For a more detailed discussion of PRET's logic system, including examples of how `scheme-eval` is used in the ODE theory, see [49,77–79].

One of the most important advantages of this declarative reasoning framework is its modularity and extensibility: a mathematics expert can easily modify and extend PRET's ODE theory. This framework was intentionally designed so that working with it does not require knowledge of any of the inner workings of the program. Adding a reasoning mode to PRET's repertoire amounts to writing two or three Horn clauses, similar to the rules on Pages 6 and 18. These Horn clauses interpret the results of the reasoning mode by specifying the conditions under which those results contradict observations about the

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

target system. If a model observer function that evaluates the corresponding mathematical property of a model does not exist, one would also have to write a short SCHEME/MAPLE function like `linear-system?`, above.

Declarative formalisms like the one described in this section are widely used by AI systems. However, PRET uses declarative techniques not only for the representation of knowledge about dynamical systems and their models, but also for the representation of strategies that specify under which conditions the inference engine should focus its attention on particular pieces or types of knowledge. This is the topic of the next two subsections.

3.2.2 Declarative Meta Level Control

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 provides meta-level language constructs that allow the implementer of the ODE theory to specify the *control strategy* that is to be used to accomplish this. The notion of controlling resolution in a declarative manner originated in the late 1970s[26,42,43]. More recently, implemented logic programming languages (e.g., [7,45,48]) and planning systems (e.g., [6,22]) have been influenced by these ideas. The declarative specification of query optimization techniques for relational databases[24] can also be seen as a form of declarative meta control. The intuition behind PRET's declarative control constructs is, again, that the search should be guided toward a cheap and quick proof of a contradiction. As an example, consider the following (simplified and schematized) excerpt from the knowledge base.

$$
\begin{aligned}
\mathit{stable} &\leftarrow \mathit{linear}, \mathit{all\_roots\_in\_left\_half\_plane}.\\
\mathit{stable} &\leftarrow \mathit{nonlinear}, \mathit{stable\_in\_all\_basins}.\\
\mathit{hot}(L) &\leftarrow \mathit{linear}, \mathit{goal}(L, \mathit{stable}).
\end{aligned}
$$

A linear dynamical system has a unique equilibrium point, and the stability of that point—and therefore of the system as a whole—can be determined by examining the system's eigenvalues, which is a simple symbolic manipulation of the coefficients of the equation. *Nonlinear* systems, on the other hand, can have arbitrary numbers of equilibrium sets, which can be expensive to find and evaluate. Thus, if a system is known to be linear, its *overall* stability is easy to establish, whereas evaluating the stability of a nonlinear system is far more complicated and expensive. PRET's meta control predicates are intended to allow the crafter of the knowledge base to prioritize checks in ways that are appropriate to situations like this. Besides the predicate `hot`, the selection of the next literal may also be specified using the predicates `before` and `notready`. The order in which clauses are used to resolve a chosen literal is

specified using the meta control predicate `clauseorder`. For a more detailed discussion of PRET's meta control constructs, see [49].

3.2.3 Reasoning at Different Abstraction Levels

To every rule, the ODE theory implementer assigns a natural number, indicating its level of abstraction: the lower the *abstraction level number*, the more abstract the rule. These abstraction levels reflect the anticipated expense of the reasoning involved in a given rule; they express *static* control knowledge of the type: "In general, try to build proofs involving qualitative properties before building proofs involving numeric properties." The meta predicates described in Section 3.2.2, in contrast, specify *dynamic* control—i.e., how to build one particular proof in one particular situation. The implementation of this scheme is straightforward; the inference engine proceeds to a higher abstraction level number only if the attempt to prove the `falsum` with ODE rules with lower abstraction level numbers fails. (This means that bad choices for abstraction levels affect only speed, and not results.) For example, the `scheme-eval` rule that triggers numerical integration has a higher abstraction level number than the `scheme-eval` rule that calls the qualitative simulation. This static abstraction level hierarchy facilitates strategies that cannot be expressed by the dynamic meta-level predicates alone: whereas the dynamic control rules impose an *order* on the subgoals and clauses of *one* particular (but complete) proof, the abstraction levels allow PRET to *omit* less-abstract parts of the ODE theory altogether. The omitted parts are considered only in a subsequent (less-abstract) proof attempt if the more-abstract proof attempt fails. Therefore, PRET tries to build entire proofs on a more-abstract level before even considering less-abstract rules. Since abstract reasoning usually involves less detail, this approach leads to short and quick proofs of the `falsum` whenever possible.

3.2.4 Storing and Reusing Intermediate Results

In order to avoid duplication of effort, PRET stores formulae that have been expensive to derive and that are likely to be useful again later in the reasoning process. Engineering a framework that lets PRET store just the right type and amount of knowledge is a surprisingly tricky endeavor. On the one hand, remembering every formula that has ever been derived (e.g., in an ATMS[27]) is too expensive, especially since variables that range over real numbers have prohibitively many potential instantiations[13]. On the other hand, many intermediate results are very expensive to derive and would have to be rederived

[13] Everett and Forbus [34] have shown that freeing facts for garbage collection can often solve that problem. As an alternative solution, we are investigating the notion of "sparse truth maintenance."

multiple times if they were not stored for reuse. PRET reuses previously derived knowledge in three ways. First, knowledge about the physical system is global, whereas knowledge about a candidate model is local to that model. Because of this, knowledge that is independent of the current candidate model can be reused throughout the run. The fact that a time series measured from the physical system contains a limit cycle, for example, can be reused across all candidate models, but the fact that the current candidate model is of second order must be thrown out when a new candidate model is considered.

Knowledge is also reused within the process of reasoning about one particular model. Every time the reasoning proceeds to a less-abstract level, PRET needs all information that has already been derived at the more-abstract level, so it stores this information rather than rederiving it. PRET currently relies on the ODE theory implementer to help identify what kinds of information fall into this category. This involves declaring a number of predicates as *relevant*[8], which causes all succeeding subgoals with this predicate to be stored for later reuse [14] . Currently, PRET recognizes special cases and generalizations of previously proved formulae, but it maintains no contexts or labels[27] for intermediate results. Unfortunately we do not have a clear-cut heuristic as to which predicates should be declared *relevant*. Ultimately this is a matter of experimentation and experience. Good candidates for relevancy, however, are formulae that are expensive to derive or likely to be useful in multiple reasoning contexts. For example, the formula `(linear-system)`, which states that the current candidate model is a linear ODE, is established at a high abstraction level by a `scheme-eval` goal. If the abstract proof of the `falsum` fails, subsequent (less-abstract) proofs would evaluate the same `scheme-eval` goal again if PRET had not stored the `(linear-system)` fact for reuse. It is important to note that inappropriate declarations of relevance, like badly chosen static abstraction levels, do not lead to *incorrect* reasoning—only to *slow* reasoning.

Finally, many of the reasoning modes described in Section 3.1 use knowledge that has been generated by previous inferences, which may in turn have triggered other reasoning modes. As described at the end of Section 3.1.5, for instance, the nonlinear parameter estimation reasoner relies heavily on qualitative knowledge derived during the structural identification phase in order to avoid local extrema in regression landscapes. To facilitate this, PRET gives these modules access to the set of formulae that have been derived so far. For example, the parameter estimator may use a constraint (e.g., $4ac > b^2$) that has been derived by a (symbolic) root-locus analysis.

[14] The set of previously derived relevant formulae is currently implemented as a hash table.

The declarative framework described in Sections 3.1 and 3.2 allows knowledge about dynamical systems and their models to be represented in a highly effective manner. Since PRET keeps its operational semantics equivalent to its declarative semantics and uses a simple and clear modeling paradigm, it is extremely easy—even for non-programmers—to understand and use it. PRET's control knowledge works with that declarative knowledge about systems and models in order to test the latter against the former quickly and cheaply. This control knowledge is expressed as a declarative meta theory, which makes the formulation of control knowledge convenient, understandable, and extensible. This framework maintains correctness and completeness while guiding the model testing process towards quick and cheap contradiction proofs. This particular way of controlling reasoning and its instantiation to a combination of tactics—designed for and focused upon a particular problem domain—constitute one of the novel contributions of the work described in this article. None of the reasoning techniques described in Section 3.1 is new; expert engineers routinely use them when modeling dynamical systems, and versions of most have been used in at least one automated modeling tool. The set of techniques used by PRET's inference engine, the multimodal reasoning framework that integrates them, and the system architecture that lets PRET decide which one is appropriate in which situation, make the approach taken here novel and powerful.

4 Input-Output Modeling

Dynamical systems used in engineering applications are rarely passive. Rather, they have both inputs and outputs, and the relationship between the two is a critical feature of the system's behavior—and thus an important part of its model. Moreover, many dynamical systems have multiple behavioral regimes, and any successful model builder must be able to reason about this property. This can be a daunting task, even for human experts; selecting and exploring appropriate ranges of state variables, parameter values, etc., is a subtle and difficult problem that has received much attention in the qualitative reasoning community[2,13,87,88]. Manipulation of actuators and sensors so as to effect this kind of exploration is another difficult problem; determining what experiments are possible from a given initial condition with a given actuator configuration is the control-theoretic problem known as *controllability* or *reachability*. For nonlinear systems, this is an open problem; the automatic control community has developed some partial solutions for limited classes of systems[75], but we are not aware of any systematic schemes for truly automatic experiment planning, execution, and interpretation in nonlinear sys-

tems. Lastly, the results of input-output analysis must be consolidated with any existing knowledge before being used in the model-building process.

PRET's input/output modeling facilities solve these problems using a new knowledge representation and reasoning framework called *qualitative bifurcation analysis* or QBA. This framework, which is designed to support reasoning about input/output effects and the existence of multiple behavioral regimes, is based on ideas from hybrid systems, nonlinear dynamics, and computer vision. Its representation is a new construct termed the *qualitative state/parameter (QS/P)* space, which combines information about the behavior of a complex system and the effects of its control parameters (inputs) upon its behavior. QBA's reasoning procedures emulate a classic technique from nonlinear dynamics known as *bifurcation analysis*, wherein a human expert changes a control parameter, classifies the resulting behavior, determines the regime boundaries, and groups similar behaviors into equivalence classes. Putting these ideas into physical practice requires yet another reasoning layer, which translates abstract concepts about experiments, such as "measure the step response," into low-level commands that manipulate actuators and sensors in appropriate ways. PRET uses the knowledge that results from the QBA procedure in both its generate and test phases, iterating the analysis/knowledge consolidation steps if necessary until it finds one model (or set of models) that accurately describes the target system in all specified operating regimes.

4.1 The QBA Paradigm: Representation

The model-building procedures described in Sections 2 and 3 are purely passive: given a static set of observations made in a single operating regime, they produce a single ODE model that accounts for that behavior. In order to reason about the relationships between inputs and outputs, PRET needed to combine these facilities with a representation that could handle *multiple* sets of observations about a given system. Our solution, termed the *qualitative state/parameter space*, is specifically designed to classify system behavior and to support reasoning about different regimes thereof. For linear systems, there are a variety of well-known and well-understood tools for doing this (e.g., step response). Almost all of these analytical tools, however, are useless in nonlinear problems. Because of this, nonlinear dynamicists typically reason about attractors in the state space, and how the topology of those attractors changes ("bifurcates") when the system parameters are varied. This is the basis of the QBA framework.

One of the goals of the qualitative reasoning (QR) community[40] is to abstract specific instances of behavior into more-general descriptions of a system. Reasoning about time-series voltage signals from two slightly different

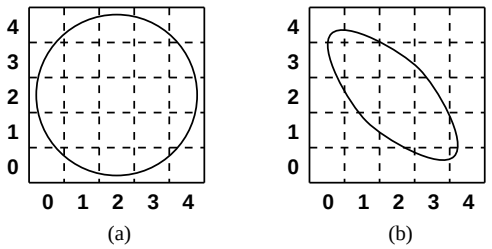

Fig. 6. Identifying a *period-one limit cycle* using the cell-dynamics method.

RLC circuits, for instance, requires detailed examination of the envelopes and phase of two decaying sinusoids. The state-space representation, which suppresses the time variable and plots voltage versus its time derivative, brings out the similarity between these two behaviors in a very clear way; all damped oscillations in linear systems, for instance, manifest on a state-space plot as similar decaying spirals. A *discretized* version of the state-space representation can effectively abstract away even more of the low-level details about the dynamics of a system while preserving its important qualitative and invariant properties. The cell dynamics formalism[52,53], for instance, discretizes a set of n-dimensional state vectors onto an n-dimensional mesh of uniform boxes or *cells*. The circular state-space trajectory in Figure 6(a), for example—a sequence of two-vectors of floating-point numbers—can be represented as the following *cell sequence*

$$[...(0,0)(1,0)(2,0)(3,0)(4,0)(4,1)(4,2)(4,3)...]$$

Because multiple trajectory points are mapped into each cell, this discretized representation of the dynamics is significantly more compact than the original series of floating-point numbers and therefore much easier to work with. Using this representation, the dynamics of a trajectory can be quickly and qualitatively classified using simple geometric heuristics—in this case as a *limit cycle*. PRET's intelligent sensor data analysis procedures use this type of discretized geometric reasoning to "distill" out the qualitative features of a given state-space portrait, allowing it to reason about these features at a much higher (and cheaper) abstraction level. These automated phase-portrait analysis techniques, which combine ideas from dynamical systems, discrete mathematics, and artificial intelligence, are covered in more detail in [15].

Raising the abstraction level of the analysis of individual sensor data sets, however, is only a very small part of the power of qualitative analysis of state-space portraits. Dynamical systems can be extremely complicated. Attempting to understand one by analyzing a single behavior instance—e.g., system evolution from *one* initial condition at *one* parameter value, like Figure 6(a)—is generally inadequate. Rather, one must vary a system's inputs and study the change in the response. Even in one-parameter systems, however, this procedure can be difficult; as the parameter is varied, the behavior may vary

smoothly in some ranges and then change abruptly at certain critical parameter values. A thorough representation of this behavior, then, requires a "stack" of phase portraits: at least one for each interesting and/or distinct range of parameter values. Constructing such a stack requires automatic recognition of these regimes, and the cell dynamics representation makes this easy. Figure 6(b), for example, shows another period-one limit cycle—one with different geometry but identical topology. The key concept here is that a set of geometrically different and yet qualitatively similar trajectories—an "equivalence class" with respect to some important dynamical property—can be classified as a single coherent group of state-space portraits.

Consider, for example, the driven pendulum system described by the ODE

$$\ddot{\theta}(t) + a_2\dot{\theta}(t) + a_1 \sin\theta(t) = d_1 \sin\alpha t$$

with drive amplitude d_1 and drive frequency α. a_1 and a_2 are physical constants of the system representing the effects of the pendulum's mass length, and damping factor, as well as gravity; the state variables of this system are θ and $\omega = \dot{\theta}$. In many experimental setups, the drive amplitude and/or frequency are controllable: these are the "control parameter" inputs to the system. The behavior of this apparently simple device is really quite complicated and interesting. For low drive frequencies, it has a single stable fixed point; as the drive frequency is raised, the attractor undergoes a series of bifurcations between chaotic and periodic behavior[29]. These bifurcations do not, however, necessarily cause the attractor to *move*. That is, the qualitative behavior of the system changes and the operating regime (in state space) does not. Traditional *bifurcation analysis* of this system would involve constructing phase portraits of the system, like the ones shown in Figure 6, at closely spaced control parameter values across some interesting range. Traditional AI/hybrid

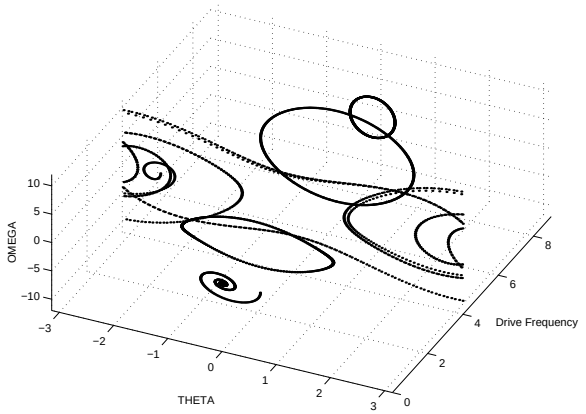

Fig. 7. The state/parameter (S/P) space portrait of the driven pendulum: a parameterized collection of phase portraits of the device at various drive frequencies. Each (θ, ω) slice of this *S/P-space portrait* is a standard state-space plot at one parameter value.

representations[18,65] do not handle this smoothly, as the operating regimes involved are not distinct. If, however, one adds an axis to the space, most of these problems vanish. Figure 7 describes the behavior of the driven pendulum in this new *state/parameter space* (S/P space) representation. Each θ, ω slice of this plot is a state-space portrait, and the control parameter varies along the Drive Frequency axis.

Our final step in the development of the representation for the qualitative bifurcation analysis framework is to combine the state/parameter space idea pictured in Figure 7 with the qualitative abstraction of the cell dynamics of Figure 6, producing the *qualitative state/parameter space* (QS/P space) representation. A QS/P-space portrait of the driven pendulum is shown in Figure 8. This representation is similar to the state/parameter space portrait in Figure 7, but it groups qualitatively similar behaviors into equivalence classes, and then uses those groupings to define the boundaries of qualitatively distinct regions. The QS/P space is an extremely effective way to capture information

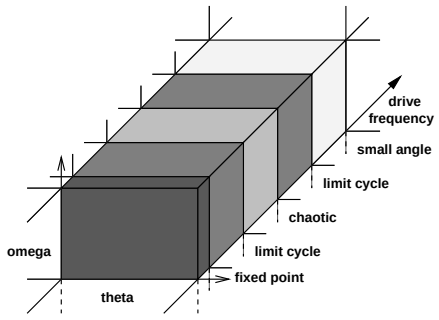

Fig. 8. The *qualitative state/parameter-space* (QS/P-space) portrait of the driven pendulum. In this abstraction of the state/parameter space, qualitatively similar behaviors are grouped into equivalence classes and those groupings are used to define the boundaries of qualitatively distinct regions of state/parameter space.

about actuator signals, sensor data, and different behavioral regions in a single compact representation. It lets the model builder leverage the knowledge that its regions—e.g., the five slabs in Figure 8—all describe the behavior of the *same system*, at different parameter values. This is very useful in reducing the complexity of the model generation and test phases. The QS/P-space representation also contributes to automatic experiment planning and execution. Among other things, it allows PRET to reason effectively about test inputs; a good test input excites the behavior in a useful but not overwhelming way, and choosing such an input is nontrivial. The following section elaborates on these ideas.

The "input" part of PRET's input-output reasoning takes place in the *intelligent sensor data analyzer*[15]. This subsystem first reconstructs any hidden dynamics from the sensor data. This step is necessary because fully *observable* systems, in which all of a system's state variables can be measured, are rare in normal engineering practice. Often, some of the state variables are either physically inaccessible or cannot be measured with available sensors. This is control theory's *observer* problem: the task of inferring the internal state of a system solely from observations of its outputs. (There has been some work in the AI/QR community on this topic; see [39].) Delay-coordinate embedding[1], PRET's solution to this problem, creates an m-dimensional *reconstruction-space* vector from m time-delayed samples of data from a single sensor. The central idea is that the reconstruction-space dynamics and the true (unobserved) state-space dynamics are topologically identical. This provides a partial solution to the observer problem, as a state-space portrait reconstructed from a single sensor is qualitatively identical to the true multidimensional dynamics of the system [15]. Given a reconstructed state-space portrait of the system's dynamics, the intelligent sensor data analyzer's second phase uses geometric reasoning to parse the trajectories into transients and attractors, and then distills out the qualitative properties of the latter—e.g., `limit-cycle`, `fixed-point`, `chaotic`, etc.—using the cell dynamics method described in the previous section.

Reasoning about actuators, which takes place in PRET's *intelligent actuator controller*[31], is much more difficult. The problem lies in the inherent difference between passive and active modeling. It is easy to recognize damped oscillations in sensor data without knowing anything about the system or the sensor, but using an actuator requires a lot of knowledge about both. Different actuators have different characteristics (range, resolution, response time, etc.); consider the difference between the time constants involved in turning off a burner on a gas versus an electric stove, and what happens if the cook is unaware of this difference. The effect of an actuator on a system also depends intimately on how the two are connected. For example, a DC motor may be viewed as a voltage-to-RPM converter with a linear range and a saturation limit. How that motor affects a driven pendulum depends not only on those characteristics, but also on the linkage between the two devices, (e.g., a direct rotary drive configuration versus a slot/cam-follower setup). To execute experiments successfully, PRET must model these kinds of properties and effects. It does so by exploiting the two-port nature of the GPN representation that was presented in Section 2.1: the effects of an actuator simply become part of the model via additional modeling components. Note that introduction of an

[15] This assumes that each sensor measures a single state variable.

actuator necessarily makes the system nonautonomous, which means that the model must include ODE terms that explicitly contain the variable `<time>`. Moreover, the explicit form of the actuator's effect on the system may not be known exactly; even if PRET transmits a known sinusoidal voltage to a particular motor, for instance, that motor may respond in a nonlinear manner (e.g., saturating above some threshold voltage). PRET's framework handles these kinds of problems automatically; the only difference between modeling drive effects and modeling internal physics is that the ODE terms that describe the former are functions of time as well as of the state variables.

Qualitative bifurcation analysis requires interleaved input and output reasoning, in which PRET uses its sensors and actuators to probe the system at a variety of control parameter values to find interesting behaviors and identify boundaries between different regimes. Currently, the user must specify the applicable range for each parameter, in the form of a `specification`. The QBA reasoner simply scans each of these ranges in turn, classifying the results as described above; it then zeroes in on the bifurcations using a simple bisecting search. These bifurcations are the dividing lines between regimes in the QS/P-space portrait of the system, and the qualitative classifications are the labels for the regimes. The results of the QBA process are twofold: a QS/P-space representation of the system dynamics—like the one shown in Figure 8—and a set of qualitative observations similar to those a human engineer would make about the system, such as "When the control parameter ρ is in the range [1.2, 5.6], the state variable x_1 oscillates." This qualitative information is useful in that it raises the abstraction level of PRET's reasoning about models, in the manner described at length in Section 3. The information captured in the QS/P-space portrait is also used by PRET's generate phase to reason about candidate models. For example, a model that is valid in one regime may be valid in other regimes that have the same qualitative behavior. And even when the qualitative behavior is different, continuity suggests that a neighboring regime's model is a reasonable starting point. (See Section 5 for discussion of some related caveats.) Coupled with the ideas described in Section 2, this kind of reasoning lets the generate phase avoid combinatorial explosions in the search space.

Apart from the identification of regime boundaries, QBA is a relatively mechanical procedure, and we are currently working on making it more intelligent. Specifically, it would be useful to focus the experiments using the knowledge that PRET has about the target system (and perhaps the candidate model), rather than simply doing a simple scan of each control parameter range. This will require determining what experiments are possible, and which of those are useful to the task at hand. As shown in Figure 9, these sets of experiments may or may not overlap. The *utility* of a particular experiment depends on what PRET knows and what it is trying to establish. It would not be useful, for instance, to duplicate existing knowledge, but utility-related

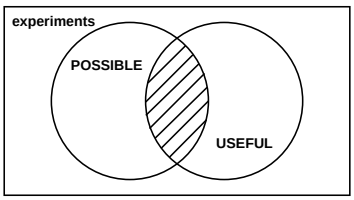

Fig. 9. Effective planning, execution, and interpretation of experiments will require PRET to reason about which ones are *useful* and *possible*.

reasoning is not always so simple. If PRET were trying to build a full-range model of a driven pendulum, but all of its observations concerned small-angle motions (where the dynamics looks like a simple harmonic oscillator), it might be useful to investigate initial conditions with larger angles. Experiment utility also depends on the domain. Impulse response is an extremely useful analytic tool if the system is known to be linear; if one can confine the problem to an even narrower domain, the available tools are even more powerful, as described in conjunction with Table 1. The GPN-based representation described in Section 2.1 will play a critical to the solution to this problem, as it is specifically designed to incorporate and exploit different levels of knowledge. The declarative meta control constructs described in Section 3.2 can be used to guide PRET's input/output exploration in a dynamic fashion that adapts to its evolving knowledge about the system (cf., the example on page 23). Determining what experiments are *possible* is even more challenging; to do so, PRET must reason about what state-space points are reachable from a given initial condition with a given actuator configuration—which requires solving the controllability/reachability problem. In the pendulum modeling scenario, for instance, it might be extremely useful to find out what happens when the device is balanced at the inverted point, but getting the system to that point is a significant real-time control problem in and of itself. Solving problems like this requires reasoning about the structure and function of the system, the actuators, and the combination of the two, given only partial information about each. The GPN-based representation, again, will be critical to this solution, as it allows PRET to model the actuator/system combination explicitly. This type of reasoning—even more so than that involved in determining utility—depends on how much PRET knows about the domain of the target system.

4.3 Summary: Automating Input-Output Modeling

The goal of input-output modeling is to apply a test input to a system, analyze the results, and learn something useful from the cause/effect pair. Automating this process is worthwhile for a variety of reasons. It makes PRET's expert knowledge useful to novices, allows it to corroborate and verify an expert's results, and it represents an important step towards the type of fully autonomous

operation that is required for many real-world AI applications (e.g., [50,66]). Automating the input/output analysis procedure is hard and interesting, from both AI and engineering standpoints. In particular, planning, executing, and interpreting experiments requires some fairly difficult reasoning about what experiments are useful and possible. Research on these general classes of reasoning problems is ongoing in the control theory/operations research and AI communities [16], and many of the specific techniques used in the design and implementation of the QBA framework have appeared in one or both of these contexts. The new idea here is the notion of working out a systematic scheme for truly automatic experiment planning, execution, and interpretation *for the purposes of modeling nonlinear systems*.

Our solution to the problem of reasoning about sensors, actuators, and how to use them to perform experiments that are useful to the modeling process rests on a new hybrid construct, the *state/parameter (S/P)* space, which combines information about the behavior of a complex system and the effects of the control parameter upon the behavior. Via a reasoning procedure termed qualitative bifurcation analysis, PRET decomposes the S/P space into discrete regions, each associated with an equivalence class of dynamical behavior, derived using geometric reasoning, to produce a *qualitative state/parameter space* (QS/P space) portrait of the system's dynamics. In this representation, each trajectory is effectively equivalent, in a well-known sense, to all the other trajectories in the same region, which allows PRET to describe the behavior of a multiregime system in a significantly simpler way, thereby streamlining the analysis and easing the computational burden. Finally, PRET also exploits the behavioral similarity captured by the QS/P space, together with continuity along its parameter axis, in order to assist the generate phase.

5 Examples

The representation, reasoning, and input-output techniques described in this paper have enabled PRET to build models of a variety of engineering problems, ranging from the simple spring/mass exercise of Figure 3 to a radio-controlled car in a deployed robotics system. This section presents three examples. The first two highlight the general execution of a PRET call and the model generation representations of Section 2.1—especially the notions of domain and meta-domain. The first of these two is a traditional automotive engineering task: modeling a vehicle's suspension. The second, drawn from a water resource domain, shows how pumping water into an aquifer affects wells that are attached to that aquifer. The third example, a parametrically driven pen-

[16] under the rubrics of "experimental design" and "reasoning about action," respectively

dulum, highlights PRET's reasoning techniques and its input-output modeling facilities.

5.1 Modeling a Shock Absorber

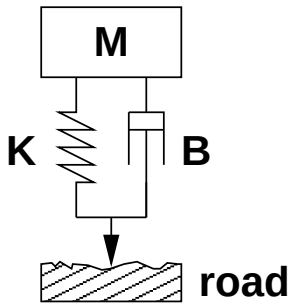

Fig. 10. A one-degree-of-freedom quarter-vehicle model that includes a shock absorber—a viscous damping element, B—connected in parallel with a spring, K, and the loading effects of the car, M.

Hydraulic shock absorbers, common in modern commercial vehicles, are complex nonlinear devices whose behavior depends upon the amplitude and frequency of the imposed motion. Accurate mathematical models of this behavior are key to realistic vehicle simulations and active-suspension controllers, among other things. Shock-absorber models normally come in two forms: either as a stand-alone shock absorber—typically just a connected spring and damper element—or as part of a quarter-vehicle model, where the loading effects of one quarter of the vehicle are included; see Figure 10. The effects of different shock absorbers in one- and two-degree-of-freedom quarter-vehicle models may be found in [59]; Besigner *et al.*[10] summarize the behavior of five different damper model variants, such as "no spring," "velocity-dependent damping," or "linear spring." Note the similarity to the component-based modeling terminology of Section 2. This is exactly the kind of expert reasoning that motivated PRET's domain knowledge framework, and these kinds of similarities make it easy for engineers to use PRET on real problems.

To illustrate its operation, we will give a brief narrative description of PRET's actions as it models a hydraulic shock absorber. As shown in the `find-model` call fragment in Figure 12, the user initializes PRET in a manner similar to the spring/mass example of Figure 3, but with a different domain (`linear-mechanics`), different hypotheses (a spring force that obeys a cubic version of Hooke's law), a drive term that represents a constant normal force on the suspension, and a noisy time-series measurement of the deflection, shown graphically in Figure 11(a).

The `linear-mechanics` domain has three default components: linear spring, mass, and damping forces. Because these components are built into the do-

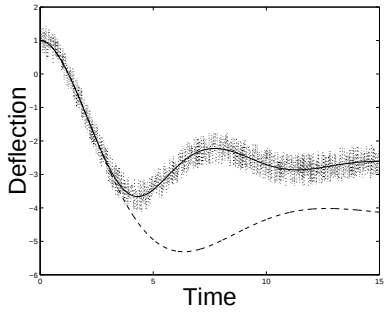

Fig. 11. Step responses of (a) a hydraulic shock absorber (dotted) (b) an unsuccessful candidate model with a linear spring (dashed) and (c) PRET's final result, which incorporates a cubic spring (solid). The deflection sensor is quite noisy; part of PRET's task is to avoid overfitting this sensor trace.

```
(find-model
  (domain linear-mechanics)
  (state-variables (<x> (integral <v>)))
  (observations
    (numeric (<time> <x>)
             ((0 1.3) (0.1 1.2) ...))
  (hypotheses
             (<force> (* k (cube (integral <v>)))))
  (drive (<force> d))
  (specifications
    (<x> absolute-resolution 1.2 (0 15))))
```

Fig. 12. A `find-model` call for the shock absorber. The state variable `<v>` is velocity (i.e. $v = \frac{dx}{dt}$, where $x =$ `<x>` is the deflection from the equilibrium position). The hypothesis represents a cubic spring force: $F = kx^3$. By default, the `linear-mechanics` domain includes linear spring, mass, and damping components, so PRET's user need not specify them explicitly. The `numeric` observation is the noisy dotted time series shown in the previous figure.

main, the user need not enter them explicitly in the `find-model` call; PRET's model generator will automatically use these three hypotheses along with those specified in the user's `hypotheses` entry. One of the challenges here is for PRET to construct a minimal model: one that avoids overfitting the noisy data. The specification concerning the resolution of the state variable `<x>` plays a key role in this, allowing the user to explicitly prescribe how closely the model must match the numeric observation. PRET uses this information to set up the cell size for its geometric reasoning stage, which automatically filters out smaller fluctuations. In the case of Figure 11(a), this means that the small, high-frequency oscillation about the mean is disregarded and state variable `<x>` is judged to be undergoing a damped oscillation to a fixed point. From these facts, the inference engine described in Section 3 deduces (among other things) that the order of any linear model must be at least two. As in the spring/mass example of Section 1, this qualitative information lets PRET immediately rule out the first dozen or so candidate models. Proceeding to

slightly more complex ODEs, PRET generates the model $a\ddot{x} + b\dot{x} + cx + d = 0$ (where x = <x>), which is made up of three domain hypotheses[17] and the user's drive hypothesis. None of the qualitative rules in the ODE theory allows this model to be ruled out, so PRET is forced to use its parameter estimator, numerical integration, and a point-by-point comparison between this model—Figure 11(b)—and the `numeric` observation—Figure 11(c)—to establish a contradiction and rule out this ODE. After discarding a variety of other unsuccessful candidate models by various means, PRET eventually generates and tests the ODE $a\ddot{x}+b\dot{x}+cx+kx^3+d = 0$. This model passes all qualitative and quantitative checks, and so is returned as PRET's output:

```
 ... no refutation ...
(model ((= (+ (* (const a) (deriv (deriv <x>)))
              (* (const b) (deriv <x>))
              (* (const c) <x>)
              (* (const k) (cube <x>))
              (const d)) 0)
        ((a 1.000) (b 0.500) (c 0.308) (k 0.0245) (d 1.304))))
```

The crafter of PRET's knowledge bases can implement the `linear-mechanics` domain in several ways. The current instantiation uses the `linear-plus` metadomain and adds several domain-specific components: `linear-mass`, `linear-damping` and `linear-spring`. These are just the generalized components `linear-differentiating`, `linear-proportional` and `linear-integrating`, renamed in a manner that makes their meaning obvious to someone who would be using the `linear-mechanics` domain. Jargon matching is only a small part of the power of domain customization, however; as described in Section 2, domain knowledge allows PRET to selectively sharpen its knowledge in appropriate ways. In this example, because PRET knows that the system is linear and mechanical, the general component `linear-integrating` takes on the more-specific meaning associated with `linear-spring`—e.g., the knowledge that mechanical springs often have appreciable mass and internal friction that cannot be neglected.

Implementing the `linear-mechanics` domain using the `linear-plus` metadomain has another very important advantage for problems like this, which have a few drive terms and a few nonlinear terms. `linear-plus` separates components into a linear and a nonlinear set, as shown in Figure 13, in order to exploit two fundamental properties of linear systems: (1) there are a polynomial number of unique nth-order linear ODEs[20], and (2) linear system inputs (drive terms) appear verbatim in the resulting ODE system. The first

[17] `linear-mass`, which is `(<force> (* a (deriv <v>)))`; `linear-damping`, which is `(<force> (* b <v>))`; and `linear-spring`, which is `(<force> (* c (integral <v>)))`.

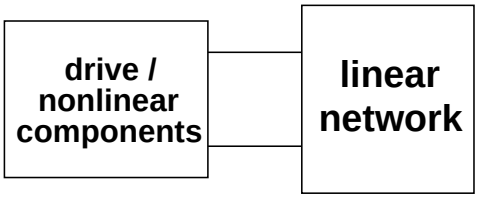

Fig. 13. PRET's `linear-plus` meta-domain is designed for systems that are basically linear, but that incorporate a few nonlinear terms and a few drive terms. Separating the linear and nonlinear/drive parts of the model reduces an exponential search space to polynomial and streamlines handling of drive terms.

of these properties effectively converts an exponential search space to polynomial; functionally equivalent linear networks reduce to the same Laplace transform transfer function, which allows PRET to identify and rule out any ODEs that are equivalent to models that have already failed the test. The second property allows this meta-domain (and thus any specific domain constructed upon it) to handle a limited number of nonlinear terms[18] by treating them as system inputs. As long as the number of nonlinear hypotheses remains small, the search space of possible models remains tractable under this assumption. See [32] for further details.

PRET's user could also skip the `linear-mechanics` domain and use the `linear-plus` meta-domain directly for this problem, simply by specifying a few extra terms in the `hypotheses` line of the `find-model` call (e.g., `(<force> (* c (integral <v>)))` and so on). The only difference between doing this and using the `linear-plus` meta-domain with extra components would manifest in PRET's run time, as it would no longer be able to use domain knowledge to streamline the generate and test phases. Indeed, one could even omit *all* hypotheses, since PRET automatically performs power-series expansions if it runs out of user and domain hypotheses, but that would increase the run time even further. The best course of action is to use as much domain knowledge as possible, and PRET's layered domain/meta-domain framework is designed to make it easy to do so.

5.2 Modeling a Well/Aquifer System

Water resource systems are made up of streams, dams, reservoirs, wells, aquifers, etc. In order to design, build, and/or manage these systems, engineers must model the relationships between the inputs (e.g., rainfall), the state variables (e.g., reservoir levels), and the outputs (e.g., the flow to some farmer's irrigation ditch). To do this in a truly accurate fashion requires partial differential equations because the physics of fluids involves multiple independent

[18] thus the name linear-*plus*.

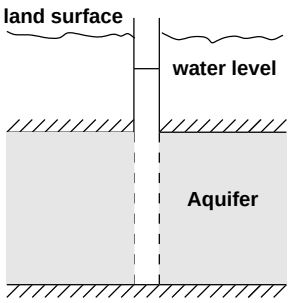

Fig. 14. An idealized representation of an open well penetrating an artesian aquifer. If the pressure in the aquifer fluctuates (e.g., sinusoidally), the water level in the well will move in response.

variables—not just time—and involves an infinite number of state variables. Partial differential equations (PDEs) are extremely hard to work with, however, so the state of the art in the water resource engineering field falls far short of that. Most existing water resource applications, such as river-dam or well-water management systems, use rule-based or statistical models. ODE models, which capture the dynamics more accurately than statistical or rule-based models but are not as difficult to handle as PDEs, are a good compromise between these two extremes, and the water resource community has begun to take this approach[19,25]. In this section, we use PRET to duplicate some of these research results and model the effects of sinusoidal pressure fluctuation in an aquifer on the level of water in a well that penetrates that aquifer, as shown in Figure 14. This example is a particularly good demonstration of the power of generalized components: it shows how GPNs allow PRET to model a variety of systems using the same underlying representation. This is especially useful when none of PRET's existing domains matches a user's application area. The well/aquifer example also demonstrates how domain knowledge and the structure inherited from the meta-domain let the model generator build interesting systems without creating an overwhelming number of models, as well as how GPNs let PRET model the load effects easily and naturally.

The first step in describing this modeling problem to PRET is to specify a domain. Because there is no built-in "water resource" domain, the user has to choose (and perhaps customize) one of the meta-domains. For this problem, the choice is obvious, as the `xmission-line` meta-domain is specifically designed for this kind of *distributed* physics, which turns up in fluid flows, vibrating strings, gas acoustics, thermal conduction and diffusion, etc. This meta-domain is designed to serve as a bridge between two very different paradigms. The GPN components of Section 2.1 represent prototypical *lumped* elements, each of which models a single physical component, whereas a system like a transmission line or a guitar string can be thought of as an infinite number of small elements (the basis of a PDE model). Using the former to model the latter requires an incremental approach. In particular, one can approximate a spatiotemporally distributed system using an iterative structure with a large

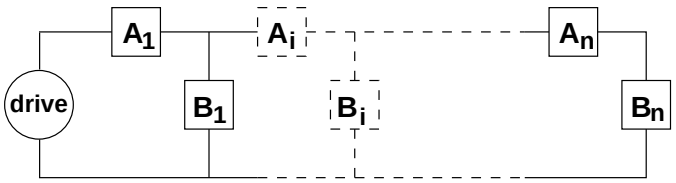

Fig. 15. The `xmission-line` meta-domain allows PRET to use its lumped-element GPN components to model spatially distributed systems like transmission lines, vibrating strings, and so on. The basic paradigm is an iterative structure with a variable number of sections, each of which has the same topology—a series element A_i and a parallel element B_i. The number of sections, each of which models a small piece of the continuum physics, rises with the precision of the model.

number of identical lumped sections. Figure 15 shows a diagram of this: a generalized iterative two-port network with n uniform sections, each of which has a serial (A_i) and a parallel (B_i) element. Each of these elements can contain one or more GPN components; they may also be "null" (essentially a short for the A_i and an open for the B_i).

The motivation for this meta-domain was the basic engineering treatment of an electrical transmission line, wherein typical electrical parameters, such as resistance or inductance, are given in per-unit-length form. Note that the topology of the sections is fixed in this metaphor: all the A_i contain the same network of GPN components, as do all the B_i; coefficient values within the individual elements can, of course, vary. If the internal structure of the elements is known *a priori*, the user would specify a single option for each of the A_i and B_i in the `hypotheses` argument to the `find-model` call[19], and the search space is $O(n)$, where n is the number of sections that are ultimately required to build an adequate model. (In the viscoelastic domain[21], for example, the A_i are null and the B_i are made up of an integrating and a proportional GPN—which correspond to a linear spring and a linear dashpot, respectively—connected in series.) If the structure within a section is not known, the user would suggest several hypotheses; the `xmission-line` meta-domain would then try out various combinations of those components in the A_i and B_i, iterating each combination out to a predetermined depth[20]. Although this does give an exponential search space—$O(2^d n)$, if there are d possible components and n required sections—d is almost always three or less in engineering practice. (If the application really demands more than three or four components, it would be best to build a new application-specific domain, based on those components and incorporating knowledge about how to efficiently combine them, as described in Section 2.1.)

[19] Doing so requires using an extended version of the `find-model` syntax, which is covered elsewhere[33].

[20] currently set at five; we are investigating other values, as well as intelligent adaptation of that limit.

As in the shock absorber example, PRET's user can either customize the meta-domain or use it directly. For the well/aquifer problem, this customization would consist of renaming the general components `linear-differentiating`, `-integrating`, and `-proportional` to match the standard domain vocabulary; differentiating and proportional effects, in particular, simulate radial flow in an aquifer, and water mass is treated as integrating. (These concepts and equivalences are a routine part of a water-resource practitioner's textbook knowledge.) The domain could be further customized based upon knowledge of the aquifer's forcing function; if the water's velocity changes slowly, for instance, inertial effects can normally be ignored, which reduces the size of the search space. For the purposes of demonstrating how one uses a PRET meta-domain directly, however, we omit this customization in this example, so the `find-model` call of Figure 16 simply instantiates the `xmission-line` meta-domain directly.

```
(find-model
  (domain xmission-line)
  (state-variables (<well-flow> <flow>))
  (observations
    (not-constant <well-flow>)
    (not-constant (deriv <well-flow>))
    (numeric (<time> <well-flow> (deriv <well-flow>))
             ((0 4.0 0.5) (0.1 4.25 0.6) ...)))
  (hypotheses
    (<effort> (* c (integral <flow>)))
    (<effort> (* l (deriv <flow>)))
    (<effort> (* r <flow>)))
  (drive (<effort> (* da (sin (* df <time>))))))
```

Fig. 16. A `find model` call fragment for the well/aquifer example of Figure 14, illustrating how one uses the meta-domain `xmission-line` directly.

This call differs from the previous examples in a variety of ways. This meta-domain has no built-in components; it only provides the template of an iterative network structure. PRET must therefore rely solely on user-specified hypotheses to build models[21]. The state variables bear domain-independent names like `<effort>` and `<flow>`, rather than domain-specific ones like `<voltage>` and `<force>`. The `xmission-line` meta-domain uses these hypotheses as the constituents of the serial and parallel elements (A_i and B_i, respectively), dynamically creating instances of each kind of state variable as A_i/B_i segments are added to the model. Since numeric observations describe specific state variables (e.g., the numerical observation of `<well-flow>` in the `find-model` call

[21] In other domains, PRET uses power-series expansions if it runs out of user hypotheses. Since the basic paradigm in `xmission-line` is essentially a spatial expansion, power-series expansions would be a duplication of effort.

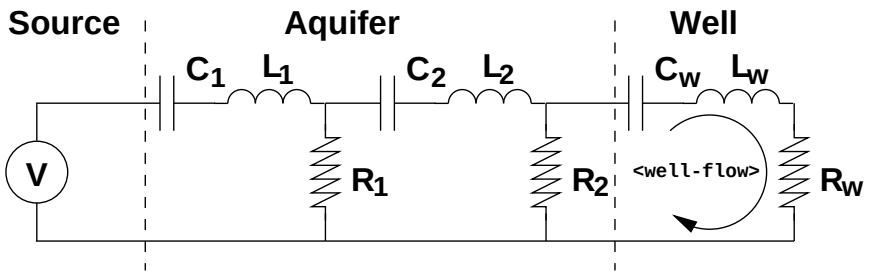

Fig. 17. A network that models the well/aquifer system of Figure 14. The drive, $V = d_a \sin d_f t$, simulates a sinusoidal pressure fluctuation in the aquifer. Note how the GPN components let the load (the well) be incorporated naturally into the model.

of Figure 16), their identifiers are pre-specified in the `state-variables` line of the call. Finally, unlike the shock absorber and spring/mass systems, the well/aquifer includes a *nonautonomous* drive term: an ODE fragment that has an explicit time dependence.

As before, PRET automatically searches the space of possible models, using the `xmission-line` meta-domain template to build models and the qualitative and quantitative techniques of Section 3 to test them, until a successful model is found. The first series of candidate models is based on a network topology where the A_i and B_i contain a single integrating component and a single differentiating component, respectively [22]. PRET first tries a one-section network of this form, and then adds identical sections to this network up to five, but none of these models passes the test. In the second series of candidate models, the A_i are integrating GPNs and the B_i are proportional GPNs, and so on. After ruling out all models whose series and parallel elements contain single GPN components, PRET then moves to more complicated topologies. The first such series, where A_1 is a single integrating GPN component and B_1 is a network containing parallel differentiating and proportional components; the third element of this series, shown in Figure 17, passes all qualitative and quantitative checks. A perfect model of a spatiotemporally distributed system, of course, requires an infinite number of discrete sections, but one can construct approximations using only a few sections, and the fidelity of the match rises with the number of sections. This dovetails neatly with PRET's `specifications`, which prescribe the required resolution of the model: PRET simply keeps adding sections until the model matches the observations [23]. In this case, it used two `xmission-line` sections to model the aquifer and one to model the well. This automatic incorporation of the well as an integral part of the model—the rightmost RLC loop—is an important feature, as loading

[22] That is, the first and second hypotheses. All hypotheses are expressed here with respect to `<flow>` state variables.

[23] This is exactly the notion of an ODE truncation of a PDE, which PRET uses the `xmission-line` meta domain to capture.

effects play critical roles in engineering analysis and design. (An audio amplifier, for example, will behave very differently if one short-circuits its speaker outputs, and the two-terminal GPN load model would factor in those effects automatically.) Component-based modeling also facilitates a simple and natural treatment of the drive term, which is attached directly to the iterative part of the network. With this approach, an actuator itself, with its various nonlinear and non-ideal properties, is represented directly as part of the network; its effects automatically become part of the model, just as they do in real systems. Finally, like `linear-plus`, the `xmission-line` meta-domain lets PRET avoid duplication of effort. Connecting arbitrary components in parallel and series creates an exponential number of candidate models, many of which are mathematically equivalent (cf., Thévenin and Norton equivalents, in network theory). The `xmission-line` meta-domain avoids this duplication by first limiting the number of possible component combinations in the initial network model and then incrementing this structure to a limited depth before attempting another initial network.

5.3 Modeling a Driven Pendulum

The driven pendulum introduced in Section 4 is a prototypical example in dynamical systems and control theory. It is also widely useful in mechanical engineering in general and robotics in particular, as well as in a variety of other fields, and accurate models of its dynamics are essential to all of these applications. Our experimental setup consists of a 15cm aluminum arm ("bob") rotating on a ball bearing under the influence of gravity, together with an actuator (a DC motor) that can impart a sinusoidal torque to that bob and a sensor (an optical encoder) that measures its angular position. As mentioned before, the behavior of this device is complex and interesting: for low drive frequencies, it has a single stable fixed point, but as the drive frequency is raised, the attractor undergoes a series of bifurcations. In the sensor data, this manifests as interleaved chaotic and periodic regimes. In this section, we show how PRET uses noisy, incomplete sensor data from this device to build accurate ODE models for each of these behavioral regimes, and how it unifies those models into a single ODE.

PRET's `linear-rotational` domain—which is based upon the `linear-plus` meta-domain—is ideal for systems like this, so the `find-model` call in Figure 18 begins by specifying that domain. Recognizing that the domain is rotational and one of the important forces (gravity) is not, the user then offers a hypothesis that captures the notion of circular-to-linear projection: $F = a_1 \sin\theta$. Furthermore, the device is driven, but the `linear-rotational` domain does not include nonautonomous drive terms, so the user suggests a parametric forcing term: $F = d_1 \sin \alpha t$. As in the case of the shock absorber

```
(find-model
  (domain linear-rotational)
  ...
  (hypotheses
    (<force> (* a1 (sin <theta>))))
  (drive (<force> (* d1 (sin (* alpha <time>)))))
  (observations
    (numeric (<time> <theta>)
             (data-acquisition (eval *acq-handle*) alpha)))
  (specifications
    (<theta> absolute-resolution 0.0125)
    (control-parameter (alpha absolute-resolution 1.0 (0.0 10.0))) ))
```

Fig. 18. Using PRET to model the driven pendulum: `<theta>` is the bob angle; `alpha` (the drive frequency) is the control parameter.

example, PRET will automatically make use of the built-in domain hypotheses; in this case, those include terms like linear friction and inertia.

Unlike the `find-model` calls for the other examples in this paper, however, the user does not specify any observations directly. Rather, she or he uses an incantation that instructs PRET's data-acquisition system to gather data directly from the system, using an actuator that controls the drive frequency `alpha` and a sensor that gathers angle (`<theta>`) versus time information. In order to support reasoning about experiments, the `specifications` take on additional roles and meanings in this example; rather than simply specifying the accuracy of the desired model, they also convey some of the physical limitations of the sensors and actuators. In this case, for example, the optical encoder that measures `<theta>` has a resolution of $\pm$ 0.7 degrees. The motor that drives the pendulum has a fixed amplitude; its frequency range and resolution are 0-10 radians per second and 1 radian per second, respectively. This information alone, however, is not enough to let PRET execute experiments automatically. Every sensor and actuator is different—whether it is digital or analog, voltage- or current-activated, involves frequency or phase, etc. Because this kind of information is all but impossible to deduce automatically, PRET requires its user to give a minimal description of the operative sensors and actuators, in the form of a short SCHEME procedure (in this case called `*acq-handle*`) that the user defines in the environment from which he or she calls PRET. The function, which is used to gather data from the target system for specific settings of the parameter `alpha`, is passed to PRET in the `find-model` call via the keyword `eval`, which causes the variable `*acq-handle*` to be evaluated in the calling environment. In this case, `*acq-handle*` has one argument—the control parameter `alpha`—and it returns a time series `<theta>` vs. `<time>`:

```
(define *acq-handle*
  (let ((length 100)          ;;; run 100 seconds for each alpha
        (time-step 0.1)       ;;; sample at 10 Hz
        (type 'DC-voltage))   ;;; sensor type
    (lambda (alpha)
      (let ((v-control alpha))
        (run-DAQ length time-step type v-control)))))
```

The function `run-DAQ`, which is part of PRET's knowledge base, manages the data-acquisition (DAQ) system that communicates with the actuator and sensor. Its first three arguments specify the length and sample rate of the sensor trace[24], together with the sensor type. (There are currently four types of sensors—`AC-voltage`, `DC-voltage`, `4-wire-resistance`, and `4-wire-temperature`—and four types of actuators: DC and AC voltage and current sources.) `run-DAQ` first sends a control voltage (`v-control`) out to the physical device via the DAQ system's digital-to-analog converter; this voltage is connected to the frequency-control input of the motor that drives the pendulum base. `run-DAQ` then uses the DAQ's multimeter board to gather a 100-second long time series of the voltage from the bob angle sensor, measured every tenth of a second. This driver function operates via system calls to the Standard Instrument Control Library (SICL)[47], which contains high-level procedures that control the data-acquisition hardware. PRET will eventually incorporate a variety of such driver functions, one for each useful actuator/sensor combination that is supported by the DAQ.

PRET begins by performing a qualitative bifurcation analysis of the device's behavior, as described in Section 4.1. It first sets the actuator to the lower end of its range (0.0) and uses the sensor to gather a series of time-stamped samples of the state variable `<theta>`. (Note that the `*acq-handle*` definition above specifies the input and output parameters of the data acquisition process completely, except for the drive voltage, which remains controllable by PRET.) In order to apply its phase-portrait analysis tools to these `<theta>` versus time data, PRET must first reconstruct the hidden dynamics from the single measured variable. This is nontrivial, since the dimension of the dynamics (the total number of state variables) is unknown. The ideal driven pendulum has three state variables—angle, angular velocity, and time—but a real device may be shaking the table on which it is mounted, expanding or contracting in response to room temperature, etc., and these effects may be large enough to factor into the behavior—and thus the model. In a real-world modeling problem, one can rarely measure (or even *know*) all the state

[24] Typically, the user will set the `time-step` of this sensor trace to be equal to the time resolution in the `specifications` section of the `find-model` call. However, this need not necessarily be the case. Whereas the `time-step` determines the sample rate at which data is gathered, the `resolution` specifies how closely the final ODE model must match the data.

variables; usually, one has access to a single imprecise, noisy sensor. PRET solves this problem using delay-coordinate embedding, as described in Section 4.2. It then uses cell dynamics and geometric reasoning to distill qualitative information (e.g., `chaotic`, `limit-cycle`) from the reconstructed portrait, as described in conjunction with Figure 6. Since the embedding theorems guarantee that the reconstructed trajectory is topologically equivalent to the true state-space dynamics, this qualitative classification also holds for the underlying (unobserved) phase portrait. PRET then repeats the reconstruction/classification procedure at the top of the `alpha` range (10.0) and in the middle (5.0). If the behavior in neighboring portraits is qualitatively different (i.e., the attractor has undergone a bifurcation), PRET uses a simple bisecting search algorithm to find the bifurcation point, down to the precision specified in the `(control-parameter (alpha absolute-resolution ...))` statement[25]. Repeating this exploration procedure across the `alpha` range, PRET eventually produces the QS/P-space portrait shown in Figure 8.

The next task is to build an ODE model for each of the five regimes in Figure 8. As in the spring/mass, shock absorber, and aquifer/well examples, PRET goes through all candidate models, trying simple ones first and more-complicated ones later, always taking advantage of the abstract-reasoning-first paradigm to discard bad models quickly. The qualitative information distilled out of the sensor data during the QBA procedure plays a particularly important role in this process; without it, PRET would be forced to rely on manipulations of the low-level data in order to build and test models, which is an extremely expensive proposition. For example, the first step in the delay-coordinate embedding procedure estimates the dimension of the system dynamics; this symbolic fact allows the model tester to immediately rule out all first order models. Because the formulae that encode the information returned by the geometric reasoning processes reside at high abstraction levels, the logic system can preferentially use this kind of abstract information to establish contradictions between model and observations quickly and cheaply.

The results of invoking PRET in each individual regime of Figure 8 are shown in Table 3. Note that four of these five ODEs are different, but all five are, in reality, instances of a single ODE that accounts for the physical behavior across the whole parameter range. PRET's goal is to find that globally valid model, so it must *unify* these ODEs. Unification is reasonably straightforward if it is correctly interleaved with the model-building process; even more importantly, this interleaving vastly reduces the complexity of the generate phase. For the

[25] This algorithm can miss bifurcations altogether if two or more of them occur *between* parameter slices, cancelling out each others' effects. It is, however, a good compromise; bifurcations represent deep changes in the dynamics, and each one leaves a highly individual signature in the behavior—signatures that are unlikely to cancel out.

Table 3
Models of the driven pendulum in different behavioral regimes.

Drive Frequency	**ODE**	**Description**
None	$\ddot{\theta}(t) = -a_2\dot{\theta}(t) - a_1 \sin\theta(t)$	damped oscillator
Low	$\ddot{\theta}(t) = -a_1 \sin\theta(t)$	nonlinear solution
Medium	$\ddot{\theta}(t) + a_2\dot{\theta}(t) + a_1 \sin\theta(t) = d_1 \sin\alpha t$	"true" (full) solution
High	$\ddot{\theta}(t) = -a_1 \sin\theta(t)$	nonlinear solution
Very High	$\ddot{\theta}(t) = -a_1\theta(t)$	linear (small angle) solution

first region of the QS/P-space portrait, where the drive is off (`alpha` = 0), PRET builds the model $\ddot{\theta}(t) = -a_2\dot{\theta}(t) - a_1 \sin\theta(t)$. When the drive is turned on, PRET reasons that a neighboring regime's model is a good starting point, and so it tries the same model, but finds that it is no longer valid. This forces a new model search, yielding the model $\ddot{\theta}(t) = -a_1 \sin\theta(t)$. (PRET is not currently equipped to perform model *refinement*: that is, it cannot *remove* terms from a model, which would be a clear win here. We are working on this.) It then tries to reconcile the two models, applying both of them in both regimes. Since $\ddot{\theta}(t) = -a_1 \sin\theta(t)$ is a special case of $\ddot{\theta}(t) = -a_2\dot{\theta}(t) - a_1 \sin\theta(t)$, the former holds in only one of the two regimes, whereas the latter holds in both, so PRET discards the $\ddot{\theta}(t) = -a_1 \sin\theta(t)$ model and goes on to the next slab of the QS/P-space portrait, repeating the model building/unification process. The notion of using the neighboring regime's model as a starting point proves to be particularly valuable in the medium-drive regime: PRET simply has to add an extra term to the evolving model[26]. Once PRET finds a single model that accounts for all observed behavior in all regimes across the range of interest, its task is complete. Such a model may not, of course, exist; a system may be governed by completely different physics in different regimes, and no single ODE may be able to account for this kind of behavior. In this case, the models in the different regimes would be mutually exclusive, and PRET would be unable to unify them into a single ODE, and so it would simply return the list of regimes, models, and transitions. This is exactly the form of a traditional hybrid model[18] of a multi-regime system.

[26] This is by no means a general, mathematically justifiable result; neighboring regimes in nonlinear systems can have very different models. Moreover, similarity of qualitative behavior of solutions to two ODEs does not formally imply anything about the similarity of those ODEs. Both approximations seem to work fairly well in practice, however: the amount of time gained when they *do* hold far outweighs the time required for PRET's inference engine to catch them when they do not.

6 Conclusion

PRET is designed to produce the type of formal engineering models that a human expert would create—quickly and automatically. Unlike existing system identification tools, PRET is not just a fancy parameter estimator; rather, it uses sophisticated knowledge representation and reasoning techniques to automate the structural identification phase of model building as well. Unlike existing AI tools, PRET takes an *active* approach to the task of modeling complex, nonlinear systems, using techniques drawn from engineering, dynamical systems, and control theory to explore their behavior directly, via sensors and actuators.

The challenges involved in automating the system identification task are significant, especially because PRET is designed for use on high-dimensional, nonlinear systems in any domain that admits ordinary differential equations. The nonlinear control-theoretic issues involved in reasoning about dynamical systems—particularly those involving the planning and execution of experiments—routinely stymie human experts. PRET's solution is based on two paradigms: generate-and-test and input/output modeling. The generate phase rests on a flexible, powerful representation—the generalized physical network—and a layered hierarchy of knowledge bases that we term *domains* and *metadomains*. This framework lets PRET effectively model systems in a wide variety of application areas and also adapt smoothly to different levels of domain knowledge. The test phase is based on a first-order logic inference system that incorporates a variety of heterogeneous reasoning modes to check models against observations. The inference system orchestrates the selection, invocation, and interaction of these reasoning modes using declaratively represented knowledge about dynamical systems—together with knowledge about *how to reason about* dynamical systems—in order to test candidate models as cheaply as possible. Input/output techniques—automatic planning, execution, and interpretation of experiments—make the modeling process interactive. This is an extremely difficult problem. In practice, one can rarely measure (or even *know*) all the state variables of a system; usually, one has access to a few imprecise, noisy sensors. PRET works with imprecision by representing and reasoning with it explicitly, deals with noise by QR-guided filtering in its nonlinear parameter estimation reasoner, reconstructs any unmeasured dynamics using delay-coordinate embedding, and identifies different behavioral regimes and the connection between inputs and outputs using cell dynamics, bifurcation analysis, and a new representation called the qualitative state/parameter space.

PRET has successfully constructed models of a dozen or so textbook problems (Rössler, Lorenz, simple pendulum, pendulum on a spring, predator-prey, Chua's circuit, etc.; see [16,17,33]), as well as several interesting and difficult

real-world examples, such as the well, shock absorber, and driven pendulum in the previous section, and a commercial radio-controlled car, which is covered in [16]. These examples are representative of wide classes of mechanical systems, both linear and nonlinear. PRET's model of the radio-controlled car was particularly interesting; it not only fit the experimental data, but actually enabled the project analysts to identify what was wrong with their mental models of the system. Specifically, PRET's model matched the observations *but not their intuition*, and the disparities led them to understand the system dynamics better. We use this anecdote to emphasize that PRET is an engineer's tool, not a scientific discovery system. Its goal is not to infer physics that the user left implicit, but rather to construct the simplest model that accounts for the observed behavior of a high-dimensional black-box system.

This notion of a *minimal* model represents a very different philosophy from traditional AI work in this area. PRET does not attempt to exceed the range and resolution specifications that are prescribed by its user: a loose `specification` for a particular state variable, for instance, implies that an exact fit of that state variable is not important to the user, so PRET will not add terms to the ODE in order to model small fluctuations in that variable. Conversely, a single out-of-range data point will cause a candidate model to fail PRET's test. These are not unwelcome side effects of the finite resolution; they are intentional and useful by-products of the abstraction level of the modeling process. A single outlying data point may appear benign if one reasons only about variances and means, but engineers care deeply about such single-point failures (such as the temperature dependence of O-ring behavior in space shuttle boosters), and a tool designed to support such reasoning must reflect those constraints.

Combining the notion of model minimality with automatic planning, execution, and interpretation of experiments is a nontrivial problem. Manipulating actuators and sensors in order to augment a model-builder's knowledge in useful ways is made difficult not only by the control-theoretic issues that are buried in the physics, but also by the meta-knowledge constraints that are inherent in the word "useful." Corroborating, verifying, and even filling in a human expert's knowledge is a worthwhile goal, but it can conflict with model minimality. Automating the input/output modeling process is an important step for autonomous situations, for nonexpert users, and so on, so sensible solutions to this conundrum are one of the current foci of our research.

Acknowledgements

Apollo Hogan, Brian LaMacchia, Abbie O'Gallagher, Janet Rogers, Ray Spiteri, and Tom Wrensch also contributed code and/or ideas to PRET.

References

[1] H. Abarbanel. *Analysis of Observed Chaotic Data.* Springer, 1995.

[2] H. Abelson. The Bifurcation Interpreter: A step towards the automatic analysis of dynamical systems. *International Journal of Computers and Mathematics with Applications*, 20:13, 1990.

[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. Astrom and P. Eykhoff. System identification — A survey. *Automatica*, 7:123–167, 1971.

[6] A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, Y. Sun, and D. Weld. UCPOP user's manual (version 4.0). Technical Report 93-09-06d, Department of Computer Science and Engineering, University of Washington, November 1995.

[7] C. Beckstein, R. Stolle, and G. Tobermann. Meta-programming for generalized Horn clause logic. In *Fifth International Workshop on Metaprogramming and Metareasoning in Logic (META-96)*, pages 27–42, 1996. Bonn, Germany.

[8] C. Beckstein and G. Tobermann. Evolutionary logic programming with RISC. In *Fourth International Workshop on Logic Programming Environments*, pages 16–21, November 1992. Washington, D.C. Technical Report TR 92-143, Center for Automation and Intelligent Systems Research at Case Western Reserve University.

[9] C. Beckstein and G. Tobermann. Algorithmic debugging and hypothetical reasoning. *Journal of Automated Software Engineering*, 4:151–178, 1997.

[10] F. Besinger, D. Cebon, and D. Cole. Damper models for heavy vehicle ride dynamics. *Vehicle System Dynamics*, 24:35–64, 1997.

[11] P. Boggs, R. Byrd, J. Rogers, and R. Schnabel. User's reference guide for ODRPACK — software for weighted orthogonal distance regression. Technical Report 4103, National Institute of Standards and Technology, Gaithersburg, MD, 20899, 1991.

[12] P. Boggs, R. Byrd, and R. Schnabel. A stable and efficient algorithm for nonlinear orthogonal distance regression. *SIAM Journal of Scientific and Statistical Computing*, 8(6):1052–1078, 1987.

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

[14] E. Bradley. Time-series analysis. In M. Berthold and D. Hand, editors, *Intelligent Data Analysis: An Introduction.* Springer, 1999.

[15] E. Bradley and M. Easley. Reasoning about sensor data for automated system identification. *Intelligent Data Analysis*, 2(2):123–138, 1998.

[16] E. Bradley, A. O'Gallagher, and J. Rogers. Global solutions for nonlinear systems using qualitative reasoning. *Annals of Mathematics and Artificial Intelligence*, 23:211–228, 1998.

[17] E. Bradley and R. Stolle. Automatic construction of accurate models of physical systems. *Annals of Mathematics and Artificial Intelligence*, 17:1–28, 1996.

[18] M. Branicky, V. Borkar, and S. Mitter. A unified framework for hybrid control. In *Proceedings of the 33rd IEEE Conference on Decision & Control*, pages 4228–4234, December 1994. Lake Buena Vista, FL.

[19] J. Bredehoeft, H. Cooper, and I. Papadopulos. Inertial and storage effects in well-aquifer systems. *Water Resource Research*, 2(4):697–707, 1966.

[20] W. Brogan. *Modern Control Theory.* Prentice-Hall, New Jersey, 3rd edition, 1991.

[21] A. Capelo, L. Ironi, and S. Tentoni. Automated mathematical modeling from experimental data: An application to material science. *IEEE Transactions on Systems, Man and Cybernetics - C*, 28:356–370, 1998.

[22] J. Carbonell, J. Blythe, O. Etzioni, Y. Gil, R. Joseph, D. Kahn, C. Knoblock, S. Minton, A. Pérez, S. Reilly, M. Veloso, and X. Wang. PRODIGY 4.0: The manual and tutorial. Technical Report CMU-CS-92-150, School of Computer Science, Carnegie Mellon University, June 1992.

[23] 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, 1991.

[24] M. Cherniack and S. Zdonik. Changing the rules: Transformations for rule-based optimizers. In *ACM SIGMOD International Conference on Management of Data*, June 1998. Seattle, WA.

[25] D. Chin. *Water-Resource Engineering.* Prentice Hall, New Jersey, 2000.

[26] R. Davis. Meta-rules: Reasoning about control. *Artificial Intelligence*, 15(3), 1980.

[27] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28(2):127–162, 1986.

[28] J. de Kleer and B. C. Williams, editors. *Artificial Intelligence*, volume 51. Elsevier Science, Amsterdam, 1991. Special Volume on Qualitative Reasoning About Physical Systems II.

[29] D. D'Humieres, M. Beasley, B. Huberman, and A. Libchaber. Chaotic states and routes to chaos in the forced pendulum. *Physical Review A*, 26:3483–3496, 1982.

[30] M. Easley and E. Bradley. Generalized physical networks for model building. In *Proceedings International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.

[31] M. Easley and E. Bradley. Reasoning about input-output modeling of dynamical systems. In *Proceedings of the Third International Symposium on Intelligent Data Analysis (IDA-99)*, Lecture Notes in Computer Science 1642, pages 343–355. Springer, 1999. Amsterdam, The Netherlands.

[32] M. Easley and E. Bradley. Meta-domains for automated model building. Technical Report CU-CS-898-00, University of Colorado at Boulder, 2000. In review for AAAI-00.

[33] M. Easley and E. Bradley. Meta-domains for automated system identification. Technical Report CU-CS-904-00, University of Colorado at Boulder, 2000. In review for ANNIE-00.

[34] J. Everett and K. Forbus. Scaling up logic-based truth maintenance systems via fact garbage collection. In *Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 614–620, 1996.

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

[36] B. Faltings and E. Gelle. Local consistency for ternary numeric constraints. In *Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 392–397, August 1997. Nagoya, Japan.

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

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

[39] K. Forbus. Interpreting observations of physical systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(3):350–359, 1987.

[40] K. Forbus. Qualitative reasoning. In A. Tucker, editor, *CRC Computer Science and Engineering Handbook*. CRC Press, Boca Raton, FL, 1997.

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

[42] H. Gallaire and C. Lasserre. Controlling knowledge deduction in a declarative approach. In *Proc. IJCAI-79*, pages S–1 – S–6, 1979.

[43] H. Gallaire and C. Lasserre. Metalevel control for logic programs. In K. L. Clark and S. A. Tärnlund, editors, *Logic Programming*, pages 173–185. Academic Press, London, 1982.

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

[45] B. Grosof. Courteous logic programs: Prioritized conflict handling for rules. Technical Report RC 20836, IBM Research, December 1997.

[46] J. Halling, editor. *Principles of Tribology*. MacMillan, 1978.

[47] Hewlett-Packard. *Standard Instrument Control Library Reference Manual*, 1996.

[48] P. Hill and J. Lloyd. *The Gödel Programming Language*. MIT Press, Cambridge, MA, 1994.

[49] A. Hogan, R. Stolle, and E. Bradley. Putting declarative meta control to work. Technical Report CU-CS-856-98, University of Colorado at Boulder, April 1998. In review for IEEE Transactions on Systems, Man, and Cybernetics.

[50] D. Hong, S. Velinsky, and X. Feng. Verification of a wheeled mobile robot dynamic model and control ramifications. *Dynamic Systems, Measurement, and Control*, 131(1):58–63, 1999.

[51] R. Horst, P. Pardalos, and N. Thoai. *Introduction to Global Optimization*. Kluwer, 1987. *Nonconvex Optimization and its Applications*, Volume 3.

[52] C. Hsu. A theory of cell-to-cell mapping dynamical systems. *Journal of Applied Mechanics*, 47:931–939, 1980.

[53] C. Hsu. *Cell-to-Cell Mapping*. Springer-Verlag, New York, 1987.

[54] J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 20:503–581, 1994.

[55] J.-N. Juang. *Applied System Identification*. Prentice Hall, Englewood Cliffs, N.J., 1994.

[56] D. Karnopp, D. Margolis, and R. Rosenberg. *System Dynamics: A Unified Approach*. Wiley, New York, second edition, 1990.

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

[58] B. J. Kuipers. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. Addison-Wesley, Reading, MA, 1992.

[59] R. Langlois and R. Anderson. Preview control algorithms for the active suspension of an off-road vehicle. *Vehicle System Dynamics*, 24:65–97, 1997.

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

[61] J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 2nd extended edition, 1987.

[62] L. McCarty. Clausal intuitionistic logic I. Fixed-point semantics. *The Journal of Logic Programming*, 5:1–31, 1988.

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

[64] P. Mosterman and G. Biswas. Formal specifications for hybrid dynamical systems. In *IJCAI-97*, 1997.

[65] P. J. Mosterman and G. Biswas. A formal hybrid modeling scheme for handling discontinuities in physical system models. In *Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 985–990, 1996.

[66] N. Muscettola, P. Nayak, B. Pell, and B. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103:5–48, 1998.

[67] P. P. Nayak. *Automated Modeling of Physical Systems*, volume 1003 of *Lecture Notes in Computer Science*. Springer, Berlin, 1995. Revised version of Ph.D. thesis, Stanford University, 1992.

[68] H. Paynter. *Analysis and Design of Engineering Systems*. MIT Press, Cambridge, MA, 1961.

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

[70] J. Reid. *Linear System Fundamentals*. McGraw-Hill, 1983.

[71] V. Robins, J. Meiss, and E. Bradley. Computing connectedness: An exercise in computational topology. *Nonlinearity*, 11:913–922, 1998.

[72] V. Robins, J. Meiss, and E. Bradley. Computing connectedness: Disconnectedness and discreteness. *Physica D*, 139:276–300, 2000.

[73] R. Sanford. *Physical Networks*. Prentice-Hall, 1965.

[74] T. Sauer, J. Yorke, and M. Casdagli. Embedology. *Journal of Statistical Physics*, 65:579–616, 1991.

[75] E. D. Sontag. *Mathematical Control Theory*. Springer, 1998.

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

[77] R. Stolle. *Integrated Multimodal Reasoning for Modeling of Physical Systems*. PhD thesis, University of Colorado, August 1998.

[78] R. Stolle and E. Bradley. A customized logic paradigm for reasoning about models. In Y. Iwasaki and A. Farquhar, editors, *Tenth International Workshop on Qualitative Reasoning (QR-96)*, 1996. Stanford Sierra Camp, CA. AAAI Technical Report WS-96-01.

[79] R. Stolle and E. Bradley. Multimodal reasoning for automatic model construction. In *Proceedings of the 15th National Conference on Artifical Intelligence (AAAI-98)*, pages 181–188, July 1998. Madison, WI.

[80] S. Strogatz. *Nonlinear Dynamics and Chaos*. Addison-Wesley, Reading, MA, 1994.

[81] G. Sussman and G. Steele. CONSTRAINTS—a language for expressing almost hierarchical descriptions. *Artificial Intelligence*, 14:1–39, 1980.

[82] J. Top and H. Akkermans. Computational and physical causality. In *IJCAI-91*, 1991.

[83] A. Torn and A. Zilinskas. *Global Optimization*. Springer, 1995. Lecture Notes in Computer Science, number 350.

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

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

[86] B. C. Williams and W. Millar. Decompositional, model-based learning and its analogy to diagnosis. In *Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998. Madison, Wisconsin.

[87] K. Yip. *KAM: A System for Intelligently Guiding Numerical Experimentation by Computer*. Artificial Intelligence Series. MIT Press, 1991.

[88] F. Zhao. Computational dynamics: Modeling and visualizing trajectory flows in phase space. *Annals of Mathematics and Artificial Intelligence*, 8:285–300, 1993.