Artificial Intelligence 133 (2001) 139–188

Artificial Intelligence

www.elsevier.com/locate/artint

Reasoning about nonlinear system identification

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

[a] *Department of Computer Science, University of Colorado, Campus Box 430, Boulder, CO 80309-0430, USA*

[b] *Rockwell Science Center, 444 High Street, Suite 400, Palo Alto, CA 94301, USA*

[c] *Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA*

Received 19 May 2000; received in revised form 4 May 2001

Abstract

System identification is the process of deducing a mathematical model of the internal dynamics of a 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.

Keywords: Automated model building; System identification; Qualitative reasoning; Qualitative physics; Knowledge representation framework; Reasoning framework; Input-output modeling

* Corresponding author.

E-mail addresses: lizb@cs.colorado.edu (E. Bradley), me@rpal.rockwell.com (M. Easley), rstolle@parc.xerox.com (R. Stolle).

[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".

PII: S0004-3702(01)00143-6

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" [69].

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 target 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 Fig. 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 designed 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 that accommodates different domains, adapts smoothly to varying amounts of domain knowledge, and allows expert users to create model-building frameworks for new application 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

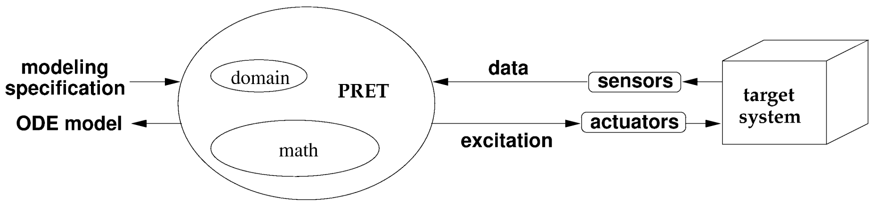


Fig. 1. PRET combines AI and formal engineering techniques to build ODE models of nonlinear dynamical systems. It uses domain-specific knowledge to build models and an encoded ODE theory to test them, and it interacts directly and autonomously with target systems using sensors and actuators.

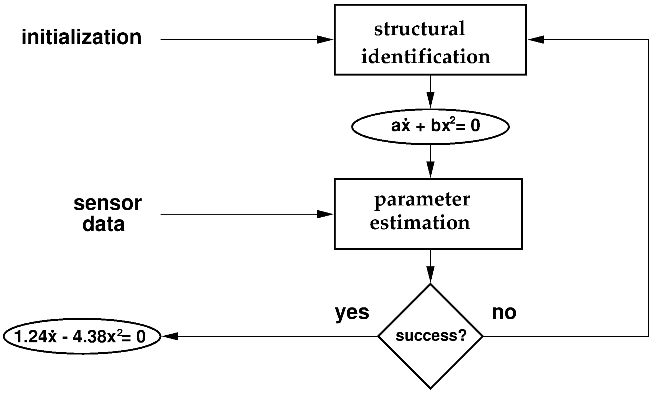


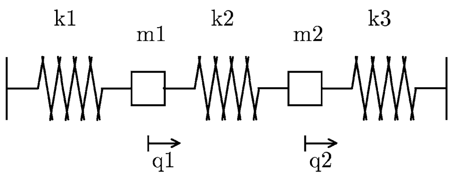
Fig. 2. The system identification (SID) process. Structural identification yields the general form of the model; in parameter estimation, values for the unknown coefficients in that model are determined. PRET automates both phases of this process.

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 Fig. 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 [59,66] 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 its encoded domain theory—the upper ellipse in Fig. 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 Fig. 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 extended by an expert user.

To make these ideas more concrete, consider the spring/mass system shown at the top right of Fig. 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




```
(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. We use `teletype` font in the body of this paper to identify terms that play roles in a user's interaction with PRET.

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 Fig. 3 informs PRET that the system to be modeled 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 Fig. 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 [5,66]; 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 the `mechanics` domain rule `(point-sum <force> 0)` that is encoded in its knowledge base to combine hypotheses into an ODE. 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 [83], 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)))                              (1)

(<- (falsum)
    ((oscillation (var StateVar))
     (not-oscillation (var StateVar))))           (2)
```

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

[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.

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

[5] PRET is written in SCHEME [75].

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

$$k_1 q_1 + m_1 \ddot{q}_1 = 0,$$
$$m_2 \ddot{q}_2 = 0.$$

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 solution of this ODE 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 Taylor-series expansions on the state variables—the standard engineering fallback in this kind of situation. This simple solution actually has a far deeper and more important advantage as well, as discussed later in this paper: it confers black-box modeling capabilities on PRET.

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

The technical challenge of this model-building process is efficiency; the search space is huge—particularly if one resorts to Taylor expansions—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. By specifying the modeling domain, the user helps PRET identify what the possible or typical "ingredients" of the target system's ODE are likely to be, thereby narrowing down the search space of candidate models. This "grey-box" modeling approach differs from traditional "black-box" modeling, where the model must be inferred only from external observations of the target system's behavior. (PRET can actually do both, as mentioned in the previous paragraph.) It is also more realistic, as described in more depth in Section 2: the engineers who are PRET's target audience do not operate in a complete vacuum, and its ability to leverage the kinds of domain knowledge that such users typically bring to a modeling problem lets PRET tailor the search space to the problem at hand.

The key to our approach is to classify 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 to remove large 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 single 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 a 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 this 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 Fig. 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 Fig. 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 process of building models of systems like this, 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.

In the AI literature, work on automatically finding a model for a given dynamic system falls under the rubrics of "reasoning about physical systems", "automated modeling", "machine learning", and "scientific discovery". The techniques presented in this paper resemble ideas from all of these research areas. PRET's representational scheme and its reasoning about candidate models build on a large body of work in automated model building and reasoning about physical systems (see, for example, [3,36,39,73,93,94]). In particular, our emphasis on qualitative reasoning and qualitative representations and their integration with numerical information and techniques falls largely into the category of "qualitative physics" (e.g., [92]). The project in this branch of the literature that is most closely related to PRET is the QR-based viscoelastic system modeling tool developed by Capelo et al. [21], which also builds ODE models from time-series data. PRET is more general; it handles linear *and nonlinear* systems in a variety of domains using a richer set of model fragments that is designed to be adaptable. (Indeed, one of PRET's implemented modeling domains, `viscoelastics`, allows it to model the same problems as in [21].)

The problem of modeling dynamic systems has also been examined from the perspectives of machine learning and scientific discovery (e.g., [57,63,88,91,97]). Target systems for automated modeling tools range from general natural phenomena and systems (e.g., gravity, planetary motion) to specific natural systems (e.g., predator–prey systems) to engineered systems (e.g., a radio-controlled car) to isolated behavioral episodes of engineered systems (the radio-controlled car's drive across the hallway last Wednesday). Correspondingly, the spectrum of possible models of such target systems ranges from scientific theories (which may even postulate newly discovered entities) to natural laws to equation systems whose abstraction level is determined by task-driven engineering requirements. Scientific discovery systems, as implied by their name, have traditionally emphasized the discovery of scientific theories or entities. Therefore, research in scientific discovery must address the question about whether the discovered theory accurately models the target system (e.g., nature), or whether it just happens to match the observations that were presented to the discovery program. Likewise, machine learning systems routinely use validation techniques (such as cross-validation) in order to ensure the "accuracy" of the learned model. PRET takes a strict engineering approach to the question of accuracy. Its goal is to find an ODE system that serves as a useful model of the target system *in the context of engineering tasks*, such as controller design. PRET's notion of "accuracy" is relative only to the given observations: it finds an ODE system that matches the observations to within the user-specified precision, and does not try to second-guess these specifications or the user's choice of observations. It is the user's responsibility to ensure that the set of observations presented to PRET and the supplied specifications reflect the engineering task at hand. It is, of course, possible to use PRET as a scientific discovery tool by supplying several sets of observations to it in separate runs and then unifying the results. PRET can also be used to solve the kinds of cross-validation problems that arise in the ML literature: one would simply use it to perform several individual validation runs and then interpret the results.

The project in the scientific discovery/machine learning branch of the literature that is most closely related to PRET is LAGRANGE [30], which builds ODE models of completely observed linear systems by applying regression techniques to time-series data. PRET and LAGRANGE can model problems of similar complexity; they differ in that PRET can handle *nonlinear* systems and *incomplete* data, while LAGRANGE cannot. This is reflected in their internal complexity as well. Since linear models admit linear regression, which is much easier than PRET's vastly more computationally expensive nonlinear parameter estimation, LAGRANGE's model tester is simple, fast, and cheap, and so its generator can afford to create a much larger number of models. PRET's search space is not only much larger, but much more expensive to navigate, hence the varied arsenal of techniques described in the rest of this paper.

Because of the highly interdisciplinary nature of the contents of this article, there are 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 *grey-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 grey-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.

Table 1
Example component representations

Component	Proportional	Differentiating	Integrating
General	$e = Bf$	$f = C\frac{\mathrm{d}e}{\mathrm{d}t}$	$e = A\frac{\mathrm{d}f}{\mathrm{d}t}$
Electrical	$v = Ri$	$i = C\frac{\mathrm{d}v}{\mathrm{d}t}$	$v = L\frac{\mathrm{d}i}{\mathrm{d}t}$
Mechanical	$v = Bf$	$f = M\frac{\mathrm{d}v}{\mathrm{d}t}$	$v = K\frac{\mathrm{d}f}{\mathrm{d}t}$

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 [79], a meta-level representation of idealized two-terminal elements, with traditional compositional model building [36] and qualitative reasoning [92]. 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]).

2.1. The CBM paradigm: Representation

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 [74]. The basis of this *generalized physical networks* (GPN) paradigm is that the behavior of an ideal two-terminal element—a "component"—may be described by a mathematical relationship between two dependent variables: generalized flow and generalized effort, where $\mathit{flow}(t) * \mathit{effort}(t) = \mathit{power}(t)$. 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 [60], 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. 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 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

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

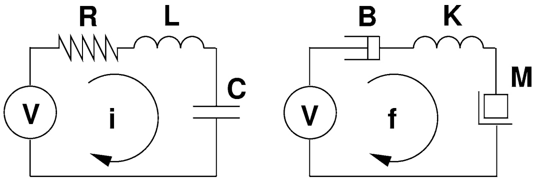


Fig. 4. Two systems that are described by the same GPN model: (a) a series RLC circuit and (b) a series damper-spring-mass system. **V** is a voltage source in (a) and a velocity source in (b).

($v = iR$) and mechanical dampers ($v = fB$) are physically analogous; both dissipate energy in a manner that is proportional to the operative state variable, and so both 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 [60] or [79] for additional domains and 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 Fig. 4, for example, can be modeled by a series proportional/integrating/differentiating GPN; knowledge that the system is electronic or mechanical would let one refine the model accordingly (to a series RLC circuit or damper-spring-mass, 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*.

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 in 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 [46], 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 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-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

[8] Nonlinear terms are somewhat idiosyncratic, and each would have to be supplied as a separate library entry. This issue has not arisen in previous work on GPNs because they have been applied mainly to linear problems.

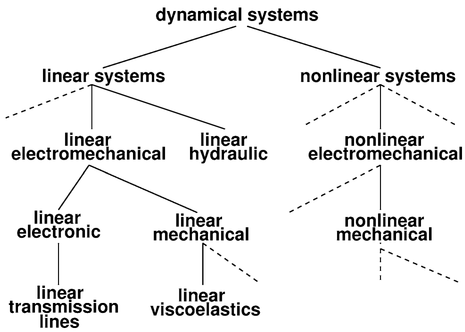


Fig. 5. A hierarchy of model building domains.

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 Fig. 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.

Focused, appropriate analysis is critical to the efficiency of the automated model building process. As demonstrated in the spring/mass example of Fig. 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 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.

Reasoning about model-building proceeds in the obvious manner dictated by 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. (Combined with the Taylor-series hypothesis generation, this layer makes PRET a capable, albeit slow, black-box modeling tool.) 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

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 [56] Delay-coordinate embedding [80] Nonlinear time series analysis [14,86]
Linear	Linear system analysis [76]
Restricted linear	Domain dependent (e.g., [21])

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 proceeds as follows. First, a candidate model is constructed from the list of hypotheses—in the form of GPN components—that are built into the domain and/or supplied by the user. These component hypotheses are combined into models using the rules of the operative domain: e.g., Newton's Third Law for `mechanics` and Kirchhoff's Voltage Law for `electrical-xmission-line`. Note that this entails selecting and using the proper *connectors* as well, as model topology is highly domain-specific. Special keywords within the hypotheses (e.g., `<force>` or `<voltage>`) serve as links to these domain rules. Equally important, these keywords (and the declarative nature of PRET's overall knowledge representation scheme; see Section 3) make the flow of the reasoning easily understandable to application engineers, allowing them to modify or augment the domain rules. The model generator selects hypotheses via simple enumeration on the list of built-in and user-supplied hypotheses, beginning with all possible one-term models, then all possible two-term models, and so on. If no model in this sequence is consistent with the observations, it uses a variety of power-series expansions to automatically generate new ODE terms, adds them to the end of the hypothesis list, and continues the process.

This model testing process is guided by the hierarchy in Fig. 5 and 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. These rules, too, are represented declaratively, using keywords that follow the language of mathematics texts (e.g., `deriv`, `jacobian`, etc.), which allows experts to customize the ODE theory as well.

The generate and test phases are interleaved, as shown in Fig. 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, the CBM framework 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 [48]. 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 leverage that knowledge in 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, CBM provides a simple construct for specifying the structure of this information: a natural number that prioritizes possible components. The meta-domain facilities also simplify this process; building a customized 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 an integrating element in series. See Section 5 for more examples.

The primary disadvantage of the CBM paradigm—and of energy-based modeling approaches in general—is that their assumptions constrain the types of problems that can be modeled. Nonphysical systems like currency exchanges, for instance, do not necessarily obey conservation of energy, and so their dynamics cannot be described by GPNs. There has been some recent work that extends energy-based modeling paradigms for problems that may not obey the traditional (effort-flow) relationships. Thermal systems, for instance, seem to have a direct electrical analogy: temperature to voltage and heat flow to current. However, the product of temperature and heat flow is not power; rather, heat flow itself *is* power. One thermal modeling tool [60] works around this by treating temperature as an effort variable and heat flow as a flow variable, but this requires *ad hoc* techniques to couple a thermal subsystem to a more traditional subsystem (e.g., electrical or mechanical) where the traditional power relationship holds. Another interesting approach, termed *Forrester System Dynamics* (FSD) [42,49], has met with success in diverse areas where no power relationship exists, such as biology, epidemiology, sociology, economics, and strategic management. FSD models consist of a network of components that model the flow or accumulation of *other* important conserved quantities (e.g., pollutants in a river, ecological populations, or buyers in a market). This is a natural extension to our framework, and we are currently working out how to incorporate a more-general notion of conservation into our current implementation, which would greatly expand the class of applications that

[9] Tribology is the science of surface contact.

PRET can handle. This extension should be fairly straightforward; indeed, LeFèvre [65] showed that FSD models are a subset of bond graphs that use only capacitance, modulated flow sources, and generalized Kirchhoff's voltage and current laws.

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 mechanical details of the conversion algorithm are described in [32].

2.3. Component-based modeling: Summary

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 grey-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 problem, 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 [89] and modeling hybrid systems [70]. 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], as described in the previous section, 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 linear dashpots). PRET's CBM framework is much more general; not only does it use a large and rich set of meta-level components for automated model building of linear and nonlinear systems, 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 the list above allows PRET to avoid these costly low-level techniques insofar as possible; 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 [28,38,41, 92] 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
```
(3)

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 [62]. 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 [62]. It does not do full qualitative simulation [61]; 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.

[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 in (4).

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 [87] to merge and simplify such constraints [58]. However, this approach works only for linear and quadratic expressions and some special cases of higher order, and 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., [37]) 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 [77,78].

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 Fig. 2, is a complex nonlinear global optimization problem [54,90]. Given an ODE with unknown coefficients and a possibly noisy time-series observation of some 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

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

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. Space limitations preclude a thorough discussion of this approach here; what follows is only a brief overview. Please see [16] for more details. The basic idea is to use QR to do the abstract, broad-brush reasoning part of the optimization problem and then to focus in using more-precise numerical methods. 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, 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—illustrated in Fig. 6—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) [68]. Roughly, GHCIL clauses are Horn clauses that also allow embedded implications in their bodies.

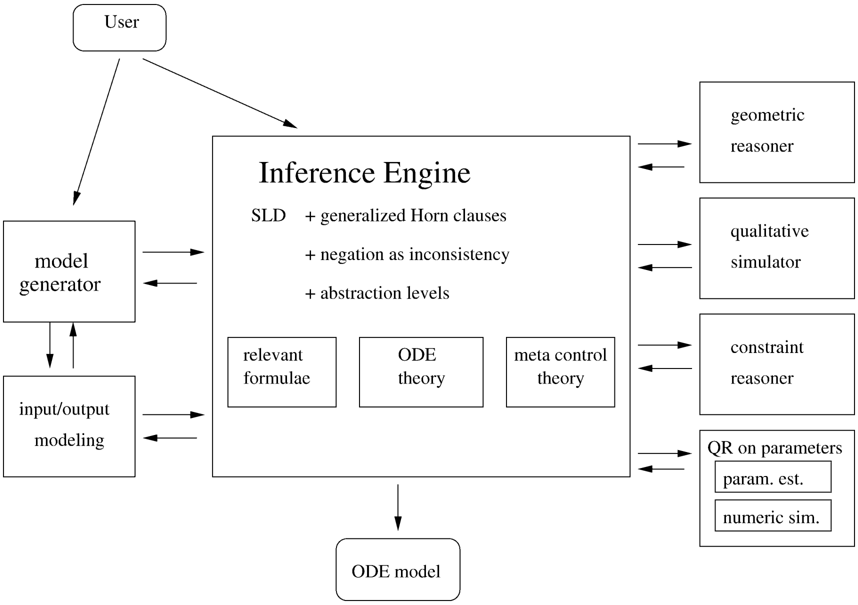


Fig. 6. PRET's architecture: an SLD-based inference engine performs deductions in generalized Horn clause logic with negation as inconsistency, operates relative to various abstraction levels, is controlled by a meta theory, stores intermediate results as relevant formulae, and interacts with the model generator of Section 2 and the input/output modeling subsystem of Section 4.

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. rule (2)). 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 Fig. 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* [43] is the only form of negation in our paradigm. Negation as failure, which is the standard form of negation in PROLOG [82], 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-based theorem prover [67]. 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,

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

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 rules (1), (2), and (3). 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))))                    (4)
```

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 [52,83–85].

One of the most important advantages of this declarative reasoning framework is its modularity and extensibility. It was intentionally designed so that working with it does not require knowledge of any of the inner workings of the program, which allows mathematics experts to easily modify and extend PRET's ODE theory. Adding a reasoning mode to PRET's repertoire, for example, amounts to writing two or three Horn clauses, similar to rules (1), (2), and (3). These Horn clauses interpret the results of the reasoning mode by specifying the conditions under which those results contradict observations about the target system. If a model observer function that evaluates the corresponding mathematical property of a model does not exist, the expert would also have to write a short SCHEME/MAPLE function like `linear-system?`, above. The existing ODE theory provides ample models for these kinds of functions.

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 [26,44,45]. More recently, implemented logic programming languages (e.g., [7,47,51]) 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.

$$\begin{aligned} \mathit{stable} &\leftarrow \mathit{linear}, \mathit{all\_roots\_in\_left\_half\_plane}. \\ \mathit{stable} &\leftarrow \mathit{nonlinear}, \mathit{stable\_in\_all\_basins}. \\ \mathit{hot}(L) &\leftarrow \mathit{linear}, \mathit{goal}(L, \mathit{stable}). \end{aligned} \tag{5}$$

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 [7,52].

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 correctness or completeness.) 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 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

[13] Everett and Forbus [35] 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”.

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

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—e.g., a constraint (e.g., $4ac > b^2$) that has been derived by a (symbolic) root-locus analysis—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.

3.3. Reasoning about models: Summary

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 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 [2,13,95,96]. 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 [81], but we are not aware of any systematic schemes for truly automatic experiment planning, execution, and interpretation in nonlinear systems. Lastly, the results of input-output analysis must be consolidated with any existing knowledge before they can be 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, such as step and frequency 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 community is to abstract specific instances of behavior into more-general descriptions of a system. Reasoning about time-series voltage signals from two slightly different 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

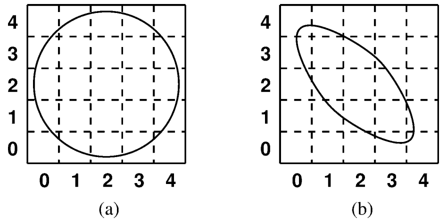


Fig. 7. Identifying a *period-one limit cycle* using the cell-dynamics method.

[55,56] discretizes a set of n-dimensional state vectors onto an n-dimensional mesh of uniform boxes or *cells*. The circular state-space trajectory in Fig. 7(a), for example—a sequence of two-vectors of floating-point numbers—can be represented as the following *cell sequence*

$$[\ldots(0,0)(1,0)(2,0)(3,0)(4,0)(4,1)(4,2)(4,3)\ldots]$$

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 AI, 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 Fig. 7(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. Fig. 7(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

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

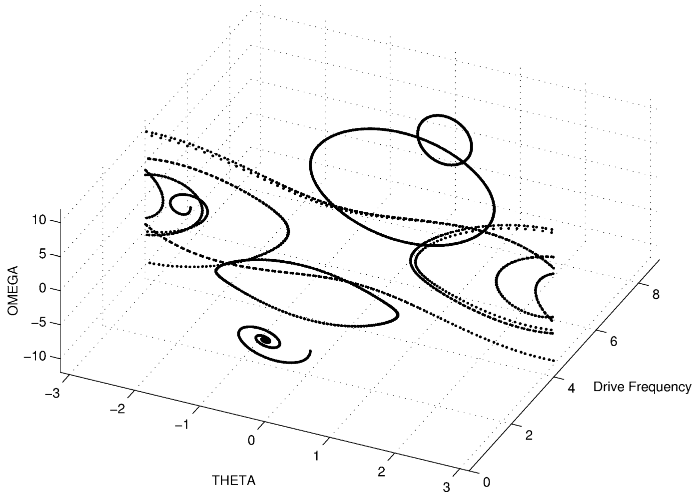


Fig. 8. The state/parameter (S/P) space portrait of the driven pendulum: a parameterized collection of phase portraits of the device at various drive frequencies. Each (θ, ω) slice of this *S/P-space portrait* is a standard state-space plot at one parameter value.

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 Fig. 7, at closely spaced control parameter values across some interesting range. Traditional AI/hybrid representations [18,71] 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. Fig. 8 describes the behavior of the driven pendulum in this *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 Fig. 8 with the qualitative abstraction of the cell dynamics of Fig. 7, producing the *qualitative state/parameter space* (QS/P space) representation. A QS/P-space portrait of the driven pendulum is shown in Fig. 9. This representation is similar to the state/parameter space portrait in Fig. 8, 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 about actuator signals,

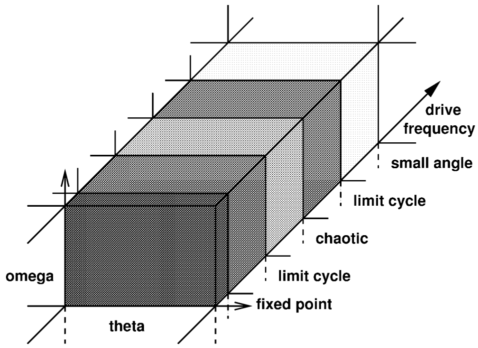


Fig. 9. The *qualitative state/parameter-space* (QS/P-space) portrait of the driven pendulum. In this abstraction of the state/parameter space, qualitatively similar behaviors are grouped into equivalence classes and those groupings are used to define the boundaries of qualitatively distinct regions of state/parameter space.

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 Fig. 9—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 all of these ideas.

4.2. The QBA paradigm: Reasoning

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 [40].) Delay-coordinate embedding [1,80], 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. 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* [33], 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 introduced in Section 2.1: the effects of an actuator simply become part of the model via additional modeling components. Note that introduction of an 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 Fig. 9—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 of the associated caveats.) Coupled with the ideas described in Section 2, this kind of reasoning lets the generate phase avoid combinatorial explosions in the search space.

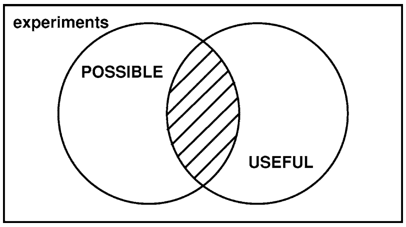


Fig. 10. Effective planning, execution, and interpretation of experiments will require PRET to reason about which ones are *useful* and *possible*.

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 Fig. 10, 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 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 2. The GPN-based representation described in Section 2.1 will play a critical role in 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. example (5)). 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 represents an important step towards the type of fully autonomous operation that is required for many real-world AI applications (e.g., [53,72]). 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, [15] 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 solutions to the problems of reasoning about sensors, actuators, and how to use them to perform experiments that are useful to the modeling process are embodied in PRET's ISAAC (intelligent sensor analysis and actuator control) module. ISAAC rests on a 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 its behavior. Via a reasoning procedure termed qualitative bifurcation analysis, ISAAC uses geometric reasoning to decompose the S/P space into discrete regions, each associated with an equivalence class of dynamical behavior, 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 ISAAC to describe the behavior of a multiregime system in a significantly simpler way, thereby streamlining the analysis and easing the computational burden. Finally, ISAAC 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 Fig. 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 modeling domains and meta-domains. The first of these two is a traditional automotive engineering task: a vehicle's suspension. The second, drawn from a water resource domain, shows how pumping water into an aquifer affects wells that are attached

[15] Under the rubrics of "experimental design" and "reasoning about action", respectively.

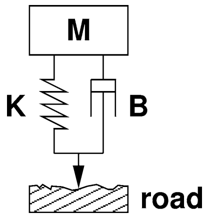


Fig. 11. A one-degree-of-freedom quarter-vehicle model that includes a shock absorber—a viscous damping element, B—connected in parallel with a spring, K, and the loading effects of the car, M.

to that aquifer. The third example, a parametrically driven pendulum, highlights PRET's reasoning techniques and its input-output modeling facilities.

5.1. Modeling a shock absorber

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 device—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 Fig. 11. The effects of different shock absorbers in one- and two-degree-of-freedom quarter-vehicle models may be found in [64]; 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 Fig. 12, the user initializes PRET in a manner similar to the spring/mass example of Fig. 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 Fig. 13. The `linear-mechanics` domain has three default components: linear spring, mass, and damping forces. Because these components are built into the domain, 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 in this problem is 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 Fig. 13, this means that the small, high-frequency oscillation about the local mean is disregarded and state

```
(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., $v = dx/dt$, 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 following figure.

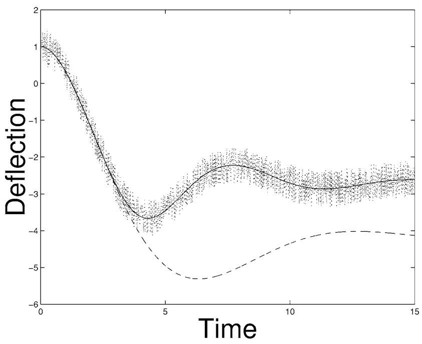


Fig. 13. Step responses of a hydraulic shock absorber (dotted) an unsuccessful candidate model with a linear spring (dashed) and PRET's final result, which incorporates a cubic spring (solid). The deflection sensor is quite noisy; part of PRET's task is to avoid overfitting this sensor trace.

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 [16] 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—the dashed trace in Fig. 13—and the noisy `numeric` observation to establish a contradiction and rule out

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

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` meta-domain 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` meta-domain 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 Fig. 14, 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 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 [17] 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 [31] 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

[17] Thus the name linear-*plus*.

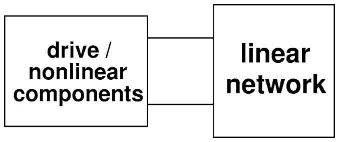


Fig. 14. PRET's linear-plus meta-domain is designed for systems that are basically linear, but that incorporate a few nonlinear terms and a few drive terms. Separating the linear and nonlinear/drive parts of the model reduces an exponential search space to polynomial and streamlines handling of drive terms.

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 is an increase 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 domain and user hypotheses, but that would increase the search space and the run time even further. (This is the true "black box" situation.) The best course of action is to exploit all available domain knowledge to reduce the search space, 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 variables—not just time—and 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 recently 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 Fig. 15. 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 exactly 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

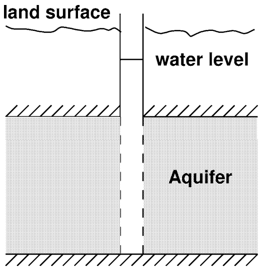


Fig. 15. An idealized representation of an open well penetrating an artesian aquifer. If the pressure in the aquifer fluctuates, the water level in the well will move in response.

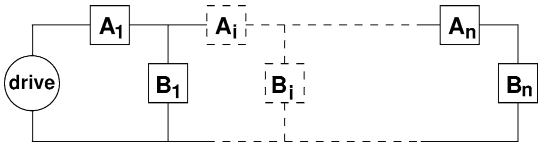


Fig. 16. The `xmission-line` meta-domain allows PRET to use its lumped-element GPN components to model spatially distributed systems like transmission lines, vibrating strings, and so on. The basic paradigm is an iterative structure with a variable number of sections, each of which has the same topology—a 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.

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. 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 number of identical lumped sections. Fig. 16 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 the `xmission-line` 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 may vary. If the user knows the internal structure of the elements, he or she would specify a single option for each of the A_i and B_i in the `hypotheses` argument to the `find-model` call, [18] and the search space is $O(n)$, where n is the number of sections that are ultimately required to build a successful model. If the internal element structure is known *a priori* across a given application area, that information can be used to construct a new modeling domain. PRET's `viscoelastics` domain, for instance, is an instantiation of the `xmission-line` meta-domain where 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. The second layer of components, which PRET uses if it needs to expand the domain dynamically, incorporates dashpots and parallel dashpot/spring combinations as well. Together, these layers allow PRET to model the same problems as in [21]. 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. [19] 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.

As in the shock absorber example, PRET's user can either customize the meta-domain for the well/aquifer problem or use it directly. In this case, the 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 meta-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 all customization in this example, so the `find-model` call of Fig. 17 simply instantiates the `xmission-line` meta-domain.

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. [20] 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`

[18] Doing so requires using an extended version of the `find-model` syntax, which is covered elsewhere [34].

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

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

```
(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. 17. A `find model` call fragment for the well/aquifer example of Fig. 15, illustrating how one uses the `xmission-line` meta-domain directly.

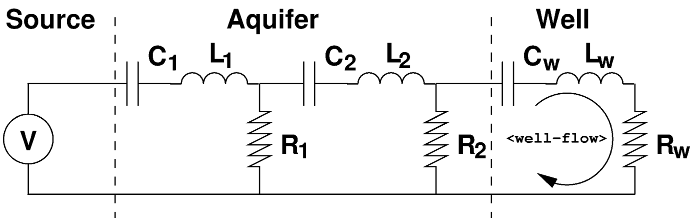


Fig. 18. PRET's model of the well/aquifer system of Fig. 15. The drive, $V = d_a \sin d_f t$, simulates a sinusoidal pressure fluctuation in the aquifer. Note how the GPN components let the load (the well) be incorporated naturally into the model.

call of Fig. 17), 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 sinusoidal 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.[21] 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. All members of this series fail the test as well. After ruling out all models whose series and parallel elements contain single GPN components, PRET then moves to more complicated

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

topologies. The first such series has A_i as a serial network containing a differentiating and an integrating GPN component and B_i as a single proportional component; the third element of this series, shown in Fig. 18, passes all qualitative and quantitative checks. In PRET's internal format, this model is expressed as follows:

```
(= (+ (∗ (deriv (deriv <i1>)) l1)
      (∗ da (sin (∗ df <time>)))
      (∗ c1 <i1>) (∗ r1 (deriv <i1>))
      (- (∗ r1 (deriv <i2>)))) 0)
(= (+ (∗ (deriv (deriv <i2>)) l2) (∗ r1 (deriv <i2>))
      (- (∗ r1 (deriv <i1>))) (∗ c2 <i2>)
      (∗ r2 (deriv <i2>))
      (- (∗ r2 (deriv <iW>)))) 0)
(= (+ (∗ (deriv (deriv <iW>)) lW) (∗ r2 (deriv <iW>))
      (- (∗ r2 (deriv <i2>))) (∗ cW <iW>)
      (∗ rW (deriv <iW>))) 0)
```

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 `specification`s, which prescribe the required resolution of the model: The CBM framework simply keeps adding sections until the model matches the observations.[22] 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 loop in Fig. 15—is an important feature, as loading effects play critical roles in engineering analysis and design. The behavior of an audio amplifier, for example, will change radically 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

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

```
(find-model
  (domain linear-rotational)
  ...
  (hypotheses
    (<force> (∗ a (sin <theta>))))
  (drive (<force> (∗ d (sin (∗ alpha <time>)))))
  (observations
    (numeric (<time> <theta>)
             (data-acquisition (eval ∗acq-handle∗) alpha)))
  (specifications
    (<theta> absolute-resolution 0.0125)
    (control-parameter (alpha absolute-resolution 1.5 (6.0 0.0)))))
```

Fig. 19. Using PRET to model the driven pendulum: `<theta>` is the bob angle; `alpha` (the drive frequency) is the control parameter.

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 15 cm 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 Fig. 19 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: $F = a\sin\theta$. Furthermore, the device is driven, but the `linear-rotational` domain does not include nonautonomous drive terms, so the user suggests a parametric forcing term: $F = d\sin\alpha t$. As in the case of the shock absorber 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, the user does not specify any observations directly. Rather, she or he uses an incantation that instructs ISAAC[23] to gather data directly from the target 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 prescribing 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

[23] The subsystem that embodies the input-output modeling solutions described in Section 4.

`<theta>` has a resolution of ± 0.7 degrees. The motor that drives the pendulum has a fixed amplitude; its frequency range and resolution are 0–6 radians per second and 1.5 radians 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 ISAAC'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. As instructed by `*acq-handle*`, `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. The associated driver function operates via system calls to the Standard Instrument Control Library (SICL) [50], 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 building an ODE model for the first time series. As before, the model generator proposes simple models first and more-complicated ones later, and the model

[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 is not an absolute requirement. 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.

tester checks these models individually, 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, the rest of 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.

Once PRET accepts a model for the initial time series, ISAAC repeats the reconstruction/classification procedure at the next `alpha` value. The ∗`acq-handle`∗ uses the `absolute-resolution` range and step specifications in the `find-model` call to increment the control parameter `alpha`. If the behavior of the resulting time series is qualitatively the same, ISAAC conjectures that the physical system is in the same regime of the qualitative state/parameter space. Reasoning from this conjecture, the model generator proposes the model that it constructed for the previous `alpha` value. If, however, the portrait is qualitatively different (e.g., the attractor has undergone a bifurcation), ISAAC assumes that the previously identified model does not hold, and so the model search process for this particular time series must begin anew. PRET continues looping in this fashion until all of the `alpha` values have been investigated. If it identifies different models for adjacent regimes, it attempts to unify the two in a pair-wise fashion—by applying the second model to the first model's regime and vice versa. Since coefficient values may vary between neighboring domains even if the same ODE holds in both, this generally requires another round of parameter estimation on each model. Whichever one is applicable in both regimes is accepted as the unifying model. Such a model may not, of course, exist; a system may be governed by completely different physics in different regimes, and thus no single ODE may be able to account for its behavior. In such a case, the different regime models would be mutually exclusive and the unification process would be unable to create a single ODE. If this happens, our solution is to simply return the list of regimes, models, and transitions, which is exactly the form of a traditional hybrid model [18] of a multi-regime system.

For the driven pendulum, this procedure plays out as follows:

- ISAAC begins its exploration of the system by generating a time series with the drive frequency α set to 6.0 radians per second, and its sensor analysis tools identify the resulting pendulum phase portrait as a `limit-cycle` response. The model generator proposes a number of first-order models, which the model tester eliminates using qualitative and quantitative means. The model generator next proposes a second-order model using a single built-in domain hypothesis that represents a proportional relationship between force and angular displacement: $\ddot{\theta} = a_1\theta$. The model tester accepts this model with the parameter value $a_1 = -35.99$.
- ISAAC continues its exploration at the next control parameter value ($\alpha = 4.5$) and again identifies the time series as a `limit-cycle`. Since the neighboring phase portraits are qualitatively similar, ISAAC reasons that the adjacent regime's model is a good starting point, and so it signals the model generator accordingly. In this case,

however, this conjecture is not borne out. Though the phase portraits are qualitatively similar, the arc of the pendulum arm is much larger at slower drive frequencies. This *quantitative* difference actually necessitates a *qualitative* change in the ODE model. In particular, a pendulum may be modeled as a simple harmonic oscillator if the angle is small ($\theta \approx \sin\theta$). At larger angles, however, this approximation no longer holds, so the simple harmonic oscillator ($\ddot{\theta} = a_1\theta$) model from the neighboring regime fails the test, forcing PRET to begin the model search anew. (This is an example of what we call an "incorrect" ODE: one whose solutions cannot match the observed time series for *any* coefficient values.) PRET continues generating and testing as before, eventually finding the nonlinear model $\ddot{\theta} = a\sin\theta$ with $a = -36.10$. The next step is to reconcile the two models, applying both of them in both regimes. Since $\ddot{\theta} = a\sin\theta$ subsumes $\ddot{\theta} = a_1\theta$, PRET discards the latter.

- ISAAC continues the QBA procedure by repeating the reconstruction/classification procedure at $\alpha = 3.0$, finds that the pendulum has bifurcated into a new behavior regime (in this case, chaotic), [25] and again restarts the model search. After generating and testing several models, PRET finds the ODE model $\ddot{\theta} = a_2\dot{\theta} + a\sin\theta + d\sin\alpha t$, which contains another built-in `linear-rotational` domain hypothesis that relates force and angular *velocity*, with coefficient values $a_2 = -1.65$, $a = -24.81$, $d = 20.72$, and $\alpha = 3.02$. (This is actually the globally valid model, but PRET cannot know this and must treat it as any other.) Note that the α value in this model is not exactly 3.000. This comes about because PRET's parameter estimator adjusts the model coefficients to optimize its fit to the observed time series—an intentional design feature that lets it ignore any noise that may be present in its boundary conditions. Lastly, PRET notes that this more-complex model subsumes the neighboring regime's model ($\ddot{\theta} = a\sin\theta$), and so accepts the former in both regimes; at the same time, it performs a bisecting search to find the precise α value for the boundary between them.
- At the next `alpha` value, 1.5, ISAAC identifies the phase portrait as a `limit-cycle`. Since this behavior has been observed before, ISAAC has examples from which to work. In particular, it proposes the most recent model that was valid in a limit-cycle regime—in this case, the globally valid model. The model tester quickly accepts $\ddot{\theta} = a_2\dot{\theta} + a\sin\theta + d\sin\alpha t$ for this regime as well, with the coefficient values $a_2 = -1.66$, $a = -24.49$, $d = 20.833$, and $\alpha = 1.50$.
- At the final control parameter value, $\alpha = 0.0$, ISAAC identifies the phase portrait as a `damped-oscillation`. Since PRET has not yet modeled any such behavior, ISAAC has no prior model to suggest, and the model generator must start anew. After rejecting a few models, the model tester accepts $\ddot{\theta} = a_2\dot{\theta} + a\sin\theta$ with $a_2 = -1.59$ and $a = -24.38$. A final unification step accepts the globally valid for this regime as well.

[25] This algorithm may miss bifurcations altogether if two or more of them occur *between* parameter slices, canceling 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	$\ddot{\theta}(t) = a_2\dot{\theta}(t) + a\sin\theta(t)$	Damped oscillator
Low	$\ddot{\theta}(t) = a_2\dot{\theta}(t) + a\sin\theta(t) + d_1\sin\alpha t$	Complex oscillator
Medium	$\ddot{\theta}(t) = a_2\dot{\theta}(t) + a\sin\theta(t) + d_1\sin\alpha t$	Chaotic behavior
High	$\ddot{\theta}(t) = a\sin\theta(t)$	Sinusoidal oscillator
Very High	$\ddot{\theta}(t) = a_1\theta(t)$	Small angle (linear) model

In the absence of unification, PRET would return different ODE models, listed in Table 3, for the five different regimes that ISAAC identifies in this system. Because it interleaves the unification process with the model-building process, however, PRET is able to unify these models in a straightforward and efficient manner. Even more important, this interleaving can vastly reduce the complexity of the generate phase. One problem with the pair-wise unification procedure, however, is that it can fail to unify a set of models even if it includes a single globally valid one. For example, assume model "A" is discovered first, followed by a different model "B" and that these two models cannot be unified. If a third and globally unifying model "C" is found next, the pair-wise procedure would not correctly unify the three models into one. A better unification procedure that solves the pair-wise deficiency would be to apply all valid models to every time series. We choose not to do this because it is expensive—$O(n^2)$, where n is the number of time series—and not absolutely necessary; practicing engineers are quite capable of deciding which models in a list subsume one another if PRET misses a few equivalences.

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. Unlike any existing software tools, PRET works with high-dimensional, nonlinear, grey-box systems: the kinds of hard problems with which engineers are faced on a daily basis.

The challenges involved in automating nonlinear system identification are significant, especially because PRET is designed to be easily extensible to any problem domain that admits ordinary differential equations. The control-theoretic issues involved in reasoning about nonlinear 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 representations that we term *domains* and *meta-domains*. 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 uses a variety of heterogeneous reasoning modes to check models against observations. This 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 a particularly difficult problem. In practice, one can rarely measure (or even *know*) all the state variables of a system; usually, one has access to imprecise, noisy sensors attached to some subset of its outputs. 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.

Theoretically, PRET can model *any* system that admits an ODE model—even in the most severe black-box situation, where it knows nothing whatsoever about that system. [26] Because the `xmission-line` meta-domain allows PRET to use its lumped-element GPN components to model spatially distributed systems, it can even model systems that technically require PDE models. In practice, however, the size of the associated search space and the available computer power limit PRET's range. The representation and reasoning tactics described in this paper mitigate this by intelligently streamlining the model-building and -testing processes. Thanks to these tactics, PRET has been able to successfully construct models of a dozen or so textbook problems (Rössler, Lorenz, simple pendulum, pendulum on a spring, etc.; see [16,17,34]), 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 dynamical 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.

This anecdote brings out an important point: PRET is intended to be an engineer's tool, and that goal dictated a specific set of design choices. From an engineering standpoint, a successful model balances accuracy and parsimony. Accordingly, PRET's goal is not to infer physics that the user left implicit, but rather to construct the simplest model that matches the observed behavior to within the predefined `specifications`. Because evaluation

[26] This is due largely to its Taylor-series model-generation facilities and the lowest-level layer of model-testing tools in Table 2.

criteria are always domain-specific, we believe that modeling tools should let their domain-expert users dictate them, and not simply build in an arbitrary set of thresholds and percentages. The notion of a *minimal* model that is tightly (some might say myopically) guided by its user's specifications represents a very different philosophy from traditional AI work in this area. Unlike some scientific discovery systems, PRET makes no attempt to exceed the range and resolution specifications that are prescribed by its user: a loose `specification` for a particular state variable, for instance, is taken as an explicit statement 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.

Achieving model parsimony and accuracy in the face of incomplete, heterogeneous knowledge and an exponential search space is a nontrivial problem. Automatic planning, execution, and interpretation of experiments can aid in its solution, but the associated implementation issues are nontrivial. 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 problem requirements. Corroborating, verifying, and even filling in a human expert's knowledge is a worthwhile goal, but it can conflict with model minimality. Nonetheless, automating the input/output modeling process is an important step for autonomous situations, for nonexpert users, etc., so sensible ways around 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 contributed code and/or ideas to PRET.

References

[1] H. Abarbanel, Analysis of Observed Chaotic Data, Springer, Berlin, 1995.
[2] H. Abelson, The bifurcation interpreter: A step towards the automatic analysis of dynamical systems, Internat. J. Comput. Math. Appl. 20 (1990) 13.
[3] S. Addanki, R. Cremonini, J.S. Penberthy, Graphs of models, Artificial Intelligence 51 (1991) 145–178.
[4] J. Amsterdam, Automated qualitative modeling of dynamic physical systems, Ph.D. Thesis, MIT, Cambridge, MA, 1992.
[5] K. Astrom, P. Eykhoff, System identification—A survey, Automatica 7 (1971) 123–167.
[6] A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, Y. Sun, D. Weld, UCPOP User's Manual (version 4.0), Technical Report 93-09-06d, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 1995.
[7] C. Beckstein, R. Stolle, G. Tobermann, Meta-programming for generalized Horn clause logic, in: Proc. 5th International Workshop on Metaprogramming, Metareasoning in Logic (META-96), Bonn, Germany, 1996, pp. 27–42.

[8] C. Beckstein, G. Tobermann, Evolutionary logic programming with RISC, in: Proc. 4th International Workshop on Logic Programming Environments, Washington, DC, Technical Report TR 92-143, Center for Automation and Intelligent Systems Research at Case Western Reserve University, Cleveland, OH, November 1992, pp. 16–21.
[9] C. Beckstein, G. Tobermann, Algorithmic debugging and hypothetical reasoning, J. Automat. Software Engineering 4 (1997) 151–178.
[10] F. Besinger, D. Cebon, D. Cole, Damper models for heavy vehicle ride dynamics, Vehicle System Dynamics 24 (1997) 35–64.
[11] P. Boggs, R. Byrd, J. Rogers, 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, R. Schnabel, A stable and efficient algorithm for nonlinear orthogonal distance regression, SIAM J. Sci. Statist. Comput. 8 (6) (1987) 1052–1078.
[13] E. Bradley, Autonomous exploration and control of chaotic systems, Cybernetics and Systems 26 (1995) 299–319.
[14] E. Bradley, Time-series analysis, in: M. Berthold, D. Hand (Eds.), Intelligent Data Analysis: An Introduction, Springer, Berlin, 2000.
[15] E. Bradley, M. Easley, Reasoning about sensor data for automated system identification, Intelligent Data Analysis 2 (2) (1998) 123–138.
[16] E. Bradley, A. O'Gallagher, J. Rogers, Global solutions for nonlinear systems using qualitative reasoning, Ann. Math. Artificial Intelligence 23 (1998) 211–228.
[17] E. Bradley, R. Stolle, Automatic construction of accurate models of physical systems, Ann. Math. Artificial Intelligence 17 (1996) 1–28.
[18] M. Branicky, V. Borkar, S. Mitter, A unified framework for hybrid control, in: Proc. 33rd IEEE Conference on Decision & Control, Lake Buena Vista, FL, 1994, pp. 4228–4234.
[19] J. Bredehoeft, H. Cooper, I. Papadopulos, Inertial and storage effects in well-aquifer systems, Water Resource Research 2 (4) (1966) 697–707.
[20] W. Brogan, Modern Control Theory, 3rd edn., Prentice-Hall, Englewood Cliffs, NJ, 1991.
[21] A. Capelo, L. Ironi, S. Tentoni, Automated mathematical modeling from experimental data: An application to material science, IEEE Trans. Systems Man Cybernet. 28 (1998) 356–370.
[22] J. Carbonell, J. Blythe, O. Etzioni, Y. Gil, R. Joseph, D. Kahn, C. Knoblock, S. Minton, A. Pérez, S. Reilly, M. Veloso, X. Wang, PRODIGY 4.0: The manual and tutorial, Technical Report CMU-CS-92-150, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1992.
[23] B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, S.M. Watt, Maple V Language Reference Manual, Springer, Berlin, 1991.
[24] M. Cherniack, S. Zdonik, Changing the rules: Transformations for rule-based optimizers, in: Proc. ACM SIGMOD International Conference on Management of Data, Seattle, WA, 1998.
[25] D. Chin, Water-Resource Engineering, Prentice Hall, Englewood Cliffs, NJ, 2000.
[26] R. Davis, Meta-rules: Reasoning about control, Artificial Intelligence 15 (3) (1980) 179–222.
[27] J. de Kleer, An assumption-based TMS, Artificial Intelligence 28 (2) (1986) 127–162.
[28] J. de Kleer, B.C. Williams (Eds.), Artificial Intelligence, Vol. 51, Elsevier Science, Amsterdam, 1991, Special Volume on Qualitative Reasoning about Physical Systems II.
[29] D. D'Humieres, M. Beasley, B. Huberman, A. Libchaber, Chaotic states and routes to chaos in the forced pendulum, Phys. Rev. A 26 (1982) 3483–3496.
[30] S. Džeroski, L. Todorovski, Discovering dynamics: From inductive logic programming to machine discovery, J. Intelligent Inform. Systems 4 (1995) 89–108.
[31] M. Easley, Automating input-output modeling of dynamic physical systems, Ph.D. Thesis, University of Colorado at Boulder, 2000.
[32] M. Easley, E. Bradley, Generalized physical networks for model building, in: Proc. IJCAI-99, Stockholm, Sweden, 1999, pp. 1047–1052.
[33] M. Easley, E. Bradley, Reasoning about input-output modeling of dynamical systems, in: Proc. 3rd International Symposium on Intelligent Data Analysis (IDA-99), Amsterdam, Springer, Berlin, 1999, pp. 343–355.

[34] M. Easley, E. Bradley, Meta-domains for automated system identification, in: C.H. Dagli, M. Akay, O.K. Ersoy, Ferná, A. Smith (Eds.), Proc. Smart Engineering System Design (ANNIE 00), ASME Press, Fairfield, NJ, 2000, pp. 165–170.
[35] J. Everett, K. Forbus, Scaling up logic-based truth maintenance systems via fact garbage collection, in: Proc. AAAI-96, Portland, OR, 1996, pp. 614–620.
[36] B. Falkenhainer, K. Forbus, Compositional modeling: Finding the right model for the job, Artificial Intelligence 51 (1991) 95–143.
[37] B. Faltings, E. Gelle, Local consistency for ternary numeric constraints, in: Proc. IJCAI-97, Nagoya, Japan, 1997, pp. 392–397.
[38] B. Faltings, P. Struss (Eds.), Recent Advances in Qualitative Physics, MIT Press, Cambridge, MA, 1992.
[39] K. Forbus, Qualitative process theory, Artificial Intelligence 24 (1984) 85–168.
[40] K. Forbus, Interpreting observations of physical systems, IEEE Trans. Systems Man Cybernet. 17 (3) (1987) 350–359.
[41] K. Forbus, Qualitative reasoning, in: A. Tucker (Ed.), CRC Computer Science and Engineering Handbook, CRC Press, Boca Raton, FL, 1997.
[42] J. Forrester, World Dynamics, Wright Allen Press, New York, 1971.
[43] D.M. Gabbay, M.J. Sergot, Negation as inconsistency I, J. Logic Programming 3 (1) (1986) 1–36.
[44] H. Gallaire, C. Lasserre, Controlling knowledge deduction in a declarative approach, in: Proc. IJCAI-79, Tokyo, Japan, 1979, pp. S–1–S–6.
[45] H. Gallaire, C. Lasserre, Metalevel control for logic programs, in: K.L. Clark, S.A. Tärnlund (Eds.), Logic Programming, Academic Press, London, 1982, pp. 173–185.
[46] H. Goldstein, Classical Mechanics, Addison Wesley, Reading, MA, 1980.
[47] B. Grosof, Courteous logic programs: Prioritized conflict handling for rules, Technical Report RC 20836, IBM Research, 1997.
[48] J. Halling (Ed.), Principles of Tribology, MacMillan, 1978.
[49] B. Hannon, M. Ruth, Dynamic Modeling, Springer, New York, 1995.
[50] Hewlett-Packard, Standard Instrument Control Library Reference Manual, 1996.
[51] P. Hill, J. Lloyd, The Gödel Programming Language, MIT Press, Cambridge, MA, 1994.
[52] A. Hogan, R. Stolle, E. Bradley, Putting declarative meta control to work. Technical Report CU-CS-856-98, University of Colorado at Boulder, 1998.
[53] D. Hong, S. Velinsky, X. Feng, Verification of a wheeled mobile robot dynamic model and control ramifications, Dynamic Systems, Measurement, and Control 131 (1) (1999) 58–63.
[54] R. Horst, P. Pardalos, N. Thoai, Introduction to Global Optimization, Nonconvex Optimization and its Applications, Vol. 3, Kluwer, Dordrecht, 1987.
[55] C. Hsu, A theory of cell-to-cell mapping dynamical systems, J. Appl. Mech. 47 (1980) 931–939.
[56] C. Hsu, Cell-to-Cell Mapping, Springer, New York, 1987.
[57] K.-M. Huang, J.M. Zytkow, Discovering empirical equations from robot-collected data, in: Z. Ras, A. Skowron (Eds.), Foundations of Intelligent Systems, Lecture Notes in Computer Science, Vol. 1325, Springer, Berlin, 1997, pp. 287–297. Proceedings of ISMIS-97, Charlotte, NC, October 1997.
[58] J. Jaffar, M. Maher, Constraint logic programming: A survey, J. Logic Programming 20 (1994) 503–581.
[59] J.-N. Juang, Applied System Identification, Prentice Hall, Englewood Cliffs, NJ, 1994.
[60] D. Karnopp, D. Margolis, R. Rosenberg, System Dynamics: A Unified Approach, 2nd edn., Wiley, New York, 1990.
[61] B.J. Kuipers, Qualitative simulation, Artificial Intelligence 29 (3) (1986) 289–338.
[62] B.J. Kuipers, Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge, Addison-Wesley, Reading, MA, 1992.
[63] P. Langley, H.A. Simon, G.L. Bradshaw, J.M. Zytkow (Eds.), Scientific Discovery: Computational Explorations of the Creative Process, MIT Press, Cambridge, MA, 1987.
[64] R. Langlois, R. Anderson, Preview control algorithms for the active suspension of an off-road vehicle, Vehicle System Dynamics 24 (1997) 65–97.
[65] J. LeFèvre, Reactive system dynamics: An extension of forrester's system dynamics using bond graph-like notations, in: Bond Graph Modeling and Simulations, ICBGM '97, Conference Proceedings, Phoenix, AZ, 1997, pp. 149–155.
[66] L. Ljung (Ed.), System Identification: Theory for the User, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[67] J.W. Lloyd, Foundations of Logic Programming, 2nd extended edn., Springer, Berlin, 1987.
[68] L. McCarty, Clausal intuitionistic logic I. Fixed-point semantics, J. Logic Programming 5 (1988) 1–31.
[69] F. Morrison, The Art of Modeling Dynamic Systems, Wiley, New York, 1991.
[70] P. Mosterman, G. Biswas, Formal specifications for hybrid dynamical systems, in: Proc. IJCAI-97, Nagoya, Japan, 1997.
[71] P.J. Mosterman, G. Biswas, A formal hybrid modeling scheme for handling discontinuities in physical system models, in: Proc. AAAI-96, Portland, OR, 1996, pp. 985–990.
[72] N. Muscettola, P. Nayak, B. Pell, B. Williams, Remote agent: To boldly go where no AI system has gone before, Artificial Intelligence 103 (1998) 5–48.
[73] P.P. Nayak, Automated Modeling of Physical Systems, Lecture Notes in Computer Science, Vol. 1003, Springer, Berlin, 1995. Revised version of Ph.D. Thesis, Stanford University, 1992.
[74] H. Paynter, Analysis and Design of Engineering Systems, MIT Press, Cambridge, MA, 1961.
[75] J. Rees, W. Clinger, The revised^3 report on the algorithmic language Scheme, ACM SIGPLAN Notices 21 (1986) 37.
[76] J. Reid, Linear System Fundamentals, McGraw-Hill, New York, 1983.
[77] V. Robins, J. Meiss, E. Bradley, Computing connectedness: An exercise in computational topology, Nonlinearity 11 (1998) 913–922.
[78] V. Robins, J. Meiss, E. Bradley, Computing connectedness: Disconnectedness and discreteness, Physica D 139 (2000) 276–300.
[79] R. Sanford, Physical Networks, Prentice-Hall, Englewood Cliffs, NJ, 1965.
[80] T. Sauer, J. Yorke, M. Casdagli, Embedology, J. Statist. Phys. 65 (1991) 579–616.
[81] E.D. Sontag, Mathematical Control Theory, Springer, Berlin, 1998.
[82] L. Sterling, E. Shapiro, The Art of PROLOG, MIT Press, Cambridge, MA, 1986.
[83] R. Stolle, Integrated multimodal reasoning for modeling of physical systems, Ph.D. Thesis, University of Colorado, 1998. To appear in Lecture Notes in Computer Science, Springer, Berlin.
[84] R. Stolle, E. Bradley, A customized logic paradigm for reasoning about models, in: Y. Iwasaki, A. Farquhar (Eds.), Proc. 10th International Workshop on Qualitative Reasoning (QR-96), Stanford Sierra Camp, CA, AAAI Technical Report WS-96-01, 1996.
[85] R. Stolle, E. Bradley, Multimodal reasoning for automatic model construction, in: Proc. AAAI-98, Madison, WI, 1998, pp. 181–188.
[86] S. Strogatz, Nonlinear Dynamics and Chaos, Addison-Wesley, Reading, MA, 1994.
[87] G. Sussman, G. Steele, CONSTRAINTS—A language for expressing almost hierarchical descriptions, Artificial Intelligence 14 (1980) 1–39.
[88] L. Todorovski, S. Džeroski, Declarative bias in equation discovery, in: Proc. 14th International Conference on Machine Learning (ICML-97), San Francisco, CA, Morgan Kaufmann, San Mateo, CA, 1997, pp. 376–384.
[89] J. Top, H. Akkermans, Computational and physical causality, in: Proc. IJACI-91, Sydney, Australia, 1991.
[90] A. Torn, A. Zilinskas, Global Optimization, Lecture Notes in Computer Science, Vol. 350, Springer, Berlin, 1995.
[91] T. Washio, H. Motoda, N. Yuji, Discovering admissible model equations from observed data based on scale-types and identity constraints, in: Proc. IJCAI-99, Stockholm, Sweden, 1999, pp. 772–779.
[92] D. Weld, J. de Kleer (Eds.), Readings in Qualitative Reasoning about Physical Systems, Morgan Kaufmann, San Mateo, CA, 1990.
[93] D.S. Weld, Reasoning about model accuracy, Artificial Intelligence 56 (1992) 255–300.
[94] B.C. Williams, W. Millar, Decompositional, model-based learning and its analogy to diagnosis, in: Proc. AAAI-98, Madison, WI, 1998.
[95] K. Yip, KAM: A System for Intelligently Guiding Numerical Experimentation by Computer, Artificial Intelligence Series, MIT Press, Cambridge, MA, 1991.
[96] F. Zhao, Computational dynamics: Modeling and visualizing trajectory flows in phase space, Ann. Math. Artificial Intelligence 8 (1993) 285–300.
[97] J.M. Zytkow, Model construction: Elements of a computational mechanism, in: Proc. Conference on Creativity, Edinburgh, 1999.