Communicable knowledge in automated system identification*

Reinhard Stolle\(^1\) and Elizabeth Bradley\(^2\)**

\(^1\) Palo Alto Research Center (PARC), Palo Alto, California, USA
\(\text{stolle@parc.com}\)
\(^2\) Department of Computer Science, University of Colorado, Boulder, Colorado, USA
\(\text{lizb@cs.colorado.edu}\)

Abstract. We describe the program PRET, an engineering tool for \textit{non-linear system identification}, which is the task of inferring a (possibly nonlinear) ordinary differential equation model from external observations of a target system’s behavior. PRET has several characteristics in common with programs from the fields of machine learning and computational scientific discovery. However, since PRET is intended to be an engineer’s tool, it makes different choices with regard to the tradeoff between model accuracy and parsimony. The choice of a good model depends on the engineering task at hand, and PRET is designed to let the user communicate the task-specific modeling constraints to the program. PRET’s inputs, its outputs, and its internal knowledge base are instances of communicable knowledge—knowledge that is represented in a form that is meaningful to the domain experts that are the intended users of the program.

1 Introduction

Models of dynamical systems are essential tools in a variety of disciplines ranging from science and engineering to economics and the social sciences (Morrison, 1991). A good model facilitates various types of reasoning about the modeled system, such as prediction of future behavior, explanation of observed behavior, understanding of correlations and influences between variables, and hypothetical reasoning about alternative scenarios.

Building good models is a routine, but difficult, task. The modeler must derive an intensional (and finite) description of the system from extensional (and possibly infinite) observations of its behavior. Traditional examples of such finite descriptions are structural models, reaction pathways, and numeric equations.


**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.
Strictly speaking, every formalization of the properties of a dynamical system constitutes a model of the dynamical system. The spectrum ranges from models that use a language that is very close to the domain of the system to models that use a language that is well-suited to describe the system mathematically. An example of the domain-centered end of this spectrum might be formal instructions on how to build an electrical circuit (e.g., a wiring diagram). These instructions would use terms like resistor, capacitor, inductor, connection, and switch. The other extreme might be differential equations; for example, the ordinary differential equation (ODE)

\[ 1.23\ddot{x} + 3\dot{x} + 46x = 0 \]  

models an electrical circuit consisting of a resistor \((R)\), an inductor \((L)\), and a capacitor \((C)\), but the form of the equation gives no hint of that correspondence.

A practitioner uses his or her domain knowledge to establish the correspondence between the mathematical formulation of the model and its domain-centered interpretation: the interpretation of the variable \(x\)—as current or voltage, respectively—in the above equation governs whether the ODE models a series or parallel circuit. A model for a parallel RLC circuit, for example, is the equation

\[ LC\dddot{v} + \frac{L}{R}\dot{v} + v = 0 \]  

where \(v\) is the voltage variable. Domain-centered models are useful for building physical systems or recognizing the function of existing physical systems, among other things. Mathematical models are useful for the precise simulation, prediction and understanding of dynamical systems. Trained experts routinely use their domain knowledge and expertise to move back and forth between these different types of model during different phases of the reasoning process.

In this chapter, we describe the modeling program PRET, which automatically constructs ODE models for given dynamical systems. The next section relates the task of *modeling in an engineering setting* to other modeling and discovery approaches that make use of related techniques. Sections 3, 4 and 5 describe the program and how it automates the modeling process. PRET uses a generate-and-test paradigm, which is described in Section 3. The “generate” phase of the generate-and-test cycle is described in detail in Easley and Bradley’s chapter of this book. The emphasis of the remainder of this current chapter is then on the “test” phase. In Section 4 we show that PRET’s inputs and internal knowledge base are instances of communicable knowledge. Finally, Section 5 explains how PRET orchestrates its reasoning process, fluidly shifting back and forth between various reasoning modes.

An introductory example of a PRET run is offered in Section 3. This sole purpose of this simple example is to illustrate the basic functionality of PRET and the main ideas behind its design. PRET has been successfully applied on a variety of systems, ranging from textbook problems to difficult real-world applications like vehicle suspensions, water resource systems, and various robotics applications (forced pendulum, radio-controlled car, etc.). Such examples, which
show the power of the program and indicate its intended application space are more complicated; they are better discussed after both phases of the generate-and-test cycle have been described. We refer the reader to Easley and Bradley’s chapter of this volume and to (Bradley et al., 2001).

2 Communicable models for engineering tasks

2.1 Explicit and implicit models

Typically, a modeler builds models out of simple components, assuming that the overall behavior follows from the behavior of the components and their interaction (Falkenhainer & Forbus, 1991). The basic building blocks of models—called model fragments—are usually correspond to well-understood concepts in the modeling domain. For example, in the context of the example illustrated by Eqn. (2), the term \( \frac{I}{R} \) corresponds to the concept “current through a resistor.” Similarly, the composition of models from model fragments corresponds to well-understood principles in the modeling domain. For example, the model of a series circuit that consists of a single loop of components may be composed out of the model fragments for those components according to Kirchhoff’s voltage law: the sum of all voltages in a loop is zero.

We call the type of models that is explained in the previous paragraph explicit models. We use this term in order to emphasize that a model and its fragments explicitly represent entities and concepts that are well-understood in the target domain and that can be reasoned about explicitly using an established body of domain knowledge.

Research on connectionist computing, Bayesian networks, data mining and knowledge discovery has produced different kinds of intensional description of dynamical systems. These new kinds of model use data structures that prevail in the field of Artificial Intelligence (AI), such as decision trees, Bayesian networks, rule sets, or neural networks. We call these descriptions implicit models to emphasize that they are not necessarily compositional and their ingredients do not immediately correspond to concepts and entities with which practitioners of the modeling domain are familiar. The dynamical systems community has also developed a variety of ways to model and predict the dynamics of a low-dimensional system using implicit models (Farmer & Sidorovich, 1987; Casdagli & Eubank, 1992); these methods match up well in practice against traditional statistical and neural-net based techniques (Weigend & Gershenfeld, 1993).

Both implicit and explicit models are very useful, but for different reasons and for different purposes. Implicit models can be extremely powerful tools because they can simulate and predict the behavior of dynamical systems with high

\[1\] The distinction between explicit and implicit models concerns only the result of the modeling process, namely the intensional description of the target system. It is independent of the search method used to find the model. It is possible to construct explicit models using AI search methods like genetic programming or backpropagation, for example. See (Koza et al., 2001; Saito & Nakano, 1997).
accuracy. Furthermore, in many modeling tasks the desired model does not need to resemble the target system structurally. Instead, the modeler merely wants to capture—or replicate—the input/output behavior of the target system. In such cases, implicit models are a very practical choice because they can be learned (or “trained”) from numerical sample data, avoiding a combinatorial search through the space of explicit compositional models. However, since implicit models do not make use of the formalisms of the target domain, they are often less useful for tasks that involve explanations and understanding with respect to the body of knowledge that is familiar to the domain practitioner.

Explicit models, on the other hand, are communicable to domain practitioners; they communicate knowledge about the target system in a form that makes sense in the context of a general body of knowledge about the domain. An explicit model facilitates various types of reasoning about the target system, such as, for example, hypotheses about alternative scenarios. An engineer may recognize a particular term in an ODE model of a mechanical system as a “friction term.” This direct correspondence between a model fragment and real-world knowledge about the phenomenon “friction” allows the engineer to anticipate the effects of changing the friction term by, say, adding a drop of oil to the target system. As another example, consider an ODE model of a robot arm that makes explicit reference to the gravitational constant $g$. Again, the direct correspondence to the real-world phenomenon “gravity” facilitates reasoning about the deployment of the robot in a different gravitational environment—on Mars, for example.

### 2.2 Scientific theories and engineering models

In the AI literature, work on automatically finding a model for a given dynamical system falls under the rubrics of “reasoning about physical systems,” “automated modeling,” “machine learning,” and “scientific discovery.” (See (Langley, 2000) for a survey of approaches to computational scientific discovery.)

The purpose of this chapter is to describe the automated modeling program PRET and to place it in the landscape of related modeling and discovery systems that produce communicable output. PRET—like several other systems that discover communicable models—not only produces a communicable model as its output, but also uses communicable knowledge during the process of computing that output. The advantages and implications of this approach are explained in more detail in the next sections. What makes PRET different and unique is its focus on automated system identification, which is the process of modeling in the context of a particular engineering task. In this section, we examine how this engineering focus distinguishes PRET from programs that discover scientific theories. In particular, we argue that this focus imposes a task-specific tradeoff between parsimony and accuracy on the modeling process. Furthermore, we describe how PRET’s combination of traditional system identification techniques\(^2\)

\(^2\) Perhaps the most important of these techniques—and one that is unique in the AI/modeling literature—is input-output modeling, in which PRET interacts directly and autonomously with its target systems, using sensors and actuators to perform
and AI techniques—especially its qualitative, "abstract-level first" techniques—allows it to find the right balance point with respect to this tradeoff.

In the research areas of Qualitative Reasoning (QR) and Qualitative Physics (QP) (Weld & de Kleer, 1990), a model of a physical system is mainly used as a representation that allows an automated system to reason about the physical system (Forbus, 1984; Kuipers, 1993). QR/QP reasoners are usually concerned with the physical system’s structure, function, or behavior. For example, qualitative simulation (Kuipers, 1986) builds a tree of qualitative descriptions of possible future evolutions of the system. Typically, the system’s structural and functional properties are known, and the task of modeling (Nayak, 1995) consists of finding a formal representation of these properties that is most suitable to the intended reasoning process. Such models frequently highlight qualitative and abstract properties of the system so as to facilitate efficient qualitative inferences. Modeling of systems with known functional and structural properties is generally called clear-box modeling.

The goal of Scientific Discovery (e.g., (Langley et al., 1987)) and System Identification (Ljung, 1987) is to investigate physical systems whose structural, functional properties are not—or are only partially—known. Modeling a target system, then, is the process of inferring an intensional (and finite) description—a model—of the system from extensional (and possibly infinite) observations of its behavior. For example, a typical system identification task is to observe a driven pendulum’s behavior over time and infer from these time series measurements an ordinary differential equation system that accounts for the observed behavior. This process is usually referred to as black-box modeling. It amounts to inverting simulation, which is the process of predicting a system’s behavior over time, given the equations that govern the system’s dynamics.

Whereas the desired model in a system identification task usually takes the form of a set of differential equations, the field of scientific discovery comprises a wider range of tasks with a broader variety of possible models. According to Langley (Langley, 2000), a scientific discovery program typically tries to discover regularities in a space of entities and concepts that has been designed by a human. Such regularities may take the form of qualitative laws, quantitative laws, process models, or structural models (which may even postulate unobserved entities). The discovery of process models amounts to explaining phenomena that involve change over time; it is the kind of scientific discovery that comes closest to system identification. Most of the scientific discovery literature (e.g., (Huang & Żytkow, 1997; Langley et al., 1987; Todorovski & Dzerecki, 1997; Washio et al., 1999; Żytkow, 1999)) revolves around the discovery of natural laws. Predator-prey systems or planetary motion are prominent examples. System identification, on the other hand, is typically performed in an engineering context—building a controller for a robot arm, for example.

For the purposes of this paper, we distinguish between theories and models. Żytkow’s terminology ( Żytkow, 1999) views theories as analytical and models as experiments whose results are useful to the model-building process. See (Easley & Bradley, 1999b).
synthetic products. We prefer to draw the distinction along the generality/specificity axis. A theory and a model are similar in the sense that both are derived from observations of target systems. However, theories aim at a more-general and more-comprehensive description of a wider range of observations. Constructing a theory includes the definition (or postulation) of relevant entities and quantities; the laws of the theory, then, express relationships that hold between these entities and quantities. The developer of a theory tries to achieve a tight correspondence between the postulated structural setup of the entities and the laws that describe the behavior of the entities.

One of the major goals of theory development is an increased understanding of the observed phenomena: the quality of a theory depends not only on how accurately the theory accounts for the observations, but also on how well the theory connects to other theories, on whether it generalizes or concretizes previous theories, and on how widely it is applicable. 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.

Engineering modeling is much less general and much more task-specific. A given domain theory sets up the space of possible quantities of interest. A model, then, is a mathematical account of the behavioral relationships between these quantities. Some or all model fragments may or may not correspond to a structural fragment of the modeled system. For example, one may recognize a particular term as a “friction term” or a “gravity term.” Whether such correspondences exist, however, is of secondary concern. The primary concern is to accurately describe the behavior of the system within a fairly limited context and with a specific task (e.g., controller design) in mind.

2.3 Parsimony in engineering modeling

In the previous paragraphs, we described the distinction between theory development and engineering modeling. As one may expect, the dividing line between these two kinds of observation interpretation is somewhat fuzzy. Even though many engineering models are very specific compared to the scope of a scientific theory, engineers also broaden their exploration beyond a single system, in order, for example, to build a cruise control that works for most cars, not just a particular Audi on a warm day. Furthermore, one might argue that the distinction between “accounting for” and “explaining” an observation is arbitrary. There may not be a big difference between saying “the resistor explains the dissipation of energy” and saying “this term (which corresponds to the resistor) accounts for that behavior (which corresponds to the dissipation of energy).”

By choosing a very specific domain theory and by setting up a specific space of possible model fragments, a human may actually convey substantial information about the target system to the automatic modeling program. As mentioned in Section 3.4, we call this compromise between clear- and black-box modeling grey-box modeling.
Nevertheless, it is important to note that explanation and understanding are stated goals of scientific discovery; in system identification, they are often merely byproducts of the modeling process. Furthermore, scientific discovery approaches often introduce higher-level concepts (e.g., the label “linear friction force” for the mathematical term $c x$) to achieve structural coherence and consistency with background knowledge and/or other theories. Such higher-level concepts are useful in automated system identification as well, but PRET’s approach is to restrict the search space in an effective, efficient grey-box modeling approach. Finally, in system identification, the task at hand provides an objective measure as to when a model is “good enough”—as opposed to a scientific theory, which is always only a step toward a further theory: something that is more general, more widely applicable, more accurate, and/or expressed in more fundamental terms.

The long-term vision in scientific discovery is even more ambitious than the previous paragraph suggests. Rather than “just” manipulating existing concepts, quantities, and entities in order to construct theories, scientific discovery programs may even invent or construct new concepts or entities on the fly. Furthermore, Chapter 10 of (Langley et al., 1987) speculates about the automated “discovery of research problems, the invention of scientific instruments, and the discovery and application of good problem representations.” Such tasks are clearly outside the scope of modeling from an engineering perspective.

This difference in long-term vision between theory developers and model constructors has important consequences concerning the parsimony of the developed theory or constructed model. Both system identification and scientific discovery strive for a simple representation of the target system. However, from a scientific discovery viewpoint, parsimony must be achieved within the constraint that the theory be behaviorally accurate and structurally coherent with background knowledge and other theories, as described in the previous paragraphs. In system identification, however, parsimony is critically important. Modelers work hard to build abstract, minimal models that account for the observations. Typically, the desired model is the one that is just concrete enough to capture the behavior that is relevant for the task at hand.

The remainder of this chapter describes how PRET performs system identification in order to find an ODE model of a given target system. It does so in an engineering context—with all the implications for parsimony and structural accuracy that have been described in this section.

3 Automated modeling with PRET

3.1 ODEs: the formalism of choice for engineering modeling

PRET is an engineering tool for nonlinear system identification, which is the task of inferring a (possibly nonlinear) ODE model from external observations of a target system’s behavior. For several reasons, ODEs are the modeling formalism of choice for many engineering tasks. First, ODEs are explicit and communicable, i.e., they can be interpreted within the context of domain knowledge,
which establishes a structural correspondence between the model fragments and domain-specific phenomena and/or entities. Model builders and users have developed a body of knowledge that uses explicit ODE models as the core language to represent and reason about systems. This body of knowledge contains theorems, techniques and procedures about systems and their corresponding models.

Second, ODEs often mark just the right trade-off between complexity and precision. Partial differential equations, for example, can describe dynamical systems more accurately, but they are vastly more difficult to deal with. Qualitative differential equations (Kuipers, 1986) are an example of the other side of this trade-off point; they can be constructed more easily than ODEs can, but their usefulness for engineering tasks (such as controller design, for example) is also more limited.

Third, there is a large body of ODE theory and associated knowledge that is independent of any particular scientific application domain. Therefore, ODEs and the associated techniques are widely applicable across multiple domains. For example, Eqn. (1) can model both a series and a parallel RLC circuit. Furthermore, the same equation can also model a series or parallel mechanical system consisting of a mass, a spring and a damper. See Easley and Bradley’s chapter of this volume for a more-detailed explanation.

PRET’s inputs are a set of observations of the outputs of the 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 model of the internal dynamics of that system. See Fig. 1 for a block diagram.

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

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 AI techniques around a set of traditional formal engineering methods. Models are represented using a component-based modeling framework (Easley & Bradley, 1999a) 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 (Easley & Bradley, 2000). An input-output modeling subsystem (Easley & Bradley, 1999b) allows PR1T 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 program's entire reasoning process is orchestrated by a special first-order logic inference system, which automatically chooses, invokes, and interprets the results of the techniques that are appropriate for each point in the model-building procedure. This combination of techniques lets PR1T 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.

### 3.2 System identification phases

PR1T's combination of symbolic and numeric techniques reflects the two phases that are interleaved in the general system identification process: 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.

![Fig. 2](image-url) 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. PR1T automates both phases of this process.
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 (Juang, 1994; Ljung, 1987) for some examples. In non-linear 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.

3.3 The generate-and-test paradigm

PRET’s basic paradigm is "generate and test." It 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.

The challenge in the design of the algorithms for PRET’s generate phase was to avoid a combinatorial explosion of the search space. We achieve this objective by allowing the user to provide knowledge about the domain and the target system that may help to limit the space of possible models. The manner in which the generate phase makes use of this kind of communicable knowledge is the topic of Easley and Bradley’s chapter of this volume.

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 PRET’s 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.

The test phase of the generate-and-test cycle is the topic of this chapter. It uses the user’s observations about the target system and PRET’s internal ODE theory in order to rule out bad candidate models quickly. Both the observations and the ODE theory are expressed as communicable knowledge; they make direct use of standard engineering concepts and vocabulary.

3.4 A simple, introductory example

To make these ideas more concrete, this section works through a simple example of a PRET run. This example illustrates how a user specifies the inputs to PRET,
how this information is used in the generate-and-test cycle, and what PRET’s main strategies are to find good models quickly. The purpose of this section is to give the reader an overview of how PRET works and what the main challenges were in the design of PRET’s architecture and set of techniques and tactics. A more precise discussion of how candidate models are generated is presented in Easley and Bradley’s chapter of this book. A more precise discussion of the reasoning that is used to test candidate models against the observations of the target system is presented in Sections 4 and 5 of this current chapter.

Consider the spring/mass system shown at the top right of Fig. 3. The coefficients $m_1$ and $m_2$ represent the two mass elements in the system; the coefficients $m_1$ and $m_2$ represent the two mass elements in the system; the coefficients

![Spring/Mass System Diagram](image)

**Fig. 3.** Modeling a simple spring/mass system using communicable formalisms. The vocabulary and concepts in which the user specifies the modeling problem are drawn from the engineering application domain. 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. The **teletype** font identifies terms that play roles in a user’s interaction with PRET.
\(k_1, k_2, \text{and } k_3\) represent the three spring elements. The state variables \(q_1\) and \(q_2\) measure the positions of the mass elements.

To instruct PRET to build a model of this system, a user would enter the `find-model` call at the left of the figure. This call contains four types of information: the domain, state variables, observations, hypotheses, and specifications.

The domain statement instantiates the relevant domain theory; the next two lines inform PRET that the system has two point-coordinate state variables.\(^4\) 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.\(^5\) The second observation states that the state variable \(q_1\) oscillates.\(^6\) 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\(^7\) ("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% 120 seconds of the system's evolution. Note that PRET uses tolerances (maximal error) as its accuracy criterion, which may seem unorthodox from a, say, machine learning perspective. Again, this choice is rooted in PRET's design rationale as an engineering tool, and it is further explained in Section 6.

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 (Ljung, 1987); 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) from 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 \(k_1q_1 = 0\). The model tester, implemented as a custom first-order logic inference engine (Stolle, 1998), uses

---

\(^4\) As described in (Bradley et al., 2001), 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.

\(^5\) That is, it does not explicitly depend on time.

\(^6\) Oscillation means that the corresponding phase-space trajectory contains a limit cycle (or spiral, in the case of damped oscillation). Again, PRET can infer this sort of qualitative observation from numeric observations of the target system itself; see (Easley & Bradley, this volume).

\(^7\) The functor `deriv` stands for "derivative." Furthermore, expressions are in prefix notation. For example, the expression `(+ r2 (square (deriv <q1>)))` represents the term \(r_2q_1^2\).
a set of general rules about ODE properties to draw inferences from the model and from the observations. In this case, PRET uses its ODE theory to establish a contradiction between the model's order (the model's highest derivative) and its oscillatory behavior. 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 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) (Bradley et al., 1998). The NPER finds no appropriate values for the coefficients \( k_1 \), \( m_1 \), and \( m_2 \), so this candidate model is also ruled out. This, however, is a far more expensive proposition than the simple symbolic contradiction proof for the one-term model above—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 coefficients and integrates the resulting ODE system with fourth-order Runge-Kutta, comparing the result to the numeric time-series observation. The difference between the numerical solution and the observation stays within the specified resolution, so this candidate model is returned as the answer. 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: it confers black-box modeling capabilities on PRET.

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. 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. It is also more realistic, as described in more depth in Easley and Bradley’s chapter of this volume: 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.

Within the space of models that are generated by the grey-box modeling approach described in the previous paragraph, PRET must identify a model that is as parsimonious as possible. At the same time, the chosen model must also meet the accuracy requirements specified by the user. This tradeoff between parsimony and accuracy is driven by the user’s engineering task at hand; it distinguishes engineering modeling from automated scientific discovery, as explained in the previous section.

In our approach, the key to quickly identifying the right model is to classify model and system behavior at the highest possible abstraction level. PRET incorporates several different reasoning techniques that are appropriate in different situations and that operate at different abstraction levels and in different domains. Examples of such techniques are symbolic algebra, qualitative reasoning about symbolic model properties, and phase-portrait analysis. These methods are drawn from the standard repertory of system identification techniques. In fact, PRET’s internal representation—just like its inputs and outputs—uses standard engineering vocabulary, abstracted into first-order predicate logic: the predicates have names like linear-system, damped-oscillation, and divergence, and their meanings are therefore familiar and easily communicable to domain experts. This framework and its associated reasoning modes are discussed in Section 4. Coordinating the invocation and interaction of PRET’s various reasoning modes is a difficult problem. 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. The reasoning control mechanism that lets PRET orchestrate this subtle and complex reasoning process is described in Section 5.

4 Communicable reasoning about dynamical systems

PRET’s inputs and outputs are designed to be in the form of communicable knowledge: the inputs are hypotheses (that is, potential model fragments), observations at various abstraction levels, and specifications concerning the ranges and resolutions of interest. All of these inputs are presented to PRET in a form that directly mirrors the form in which domain experts typically express this
kind of information. PRET’s grey-box modeling approach and its component-based representations allow users to tailor the search space of models by providing knowledge about system components and structures that are typical of the modeling domain. PRET’s outputs are ODEs, which is the model of choice for a wide variety of scientific and engineering tasks. In addition to interacting with the user in a communicable domain-centered language, PRET’s internal reasoning machinery also employs communicable knowledge that makes use of the vocabulary and the concepts of its domain, namely system identification. The communicability of PRET’s internal reasoning is the topic of this section.

As is described in Easley and Bradley’s chapter of this volume, PRET uses component-based representations, user hypotheses, and domain knowledge to generate candidate models of the given target system. Using the reasoning framework described in this section, 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 the following subsections. The advantages of representing PRET’s ODE theory and the associated techniques in the form of communicable knowledge are summarized in Section 4.6.

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 5 briefly describes the reasoning control techniques that allow it to do so.

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, this set of techniques provides PRET with the right tools to quickly test models against the given observations.\(^8\) Parameter estimation and numerical simulation are low-level, computationally expensive methods that

---

\(^8\) See, e.g., the example section of (Bradley et al., 2001).
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.

4.1 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 qual-
itative techniques (de Kleer & Williams, 1991; Faltings & Struss, 1992; Forbus,
1996; Weld & de Kleer, 1990) before advancing to more-expensive semi-numerical
or numerical techniques. Often, only a few steps of inexpensive qualitative rea-
noning suffice to quickly discard a model.

Some of PRET’s qualitative rules, in turn, make use of other tools, e.g., sym-
bolic algebra facilities from the commercial package MAPLE (Char et al., 1991).
For example, PRET’s encoded ODE theory includes the qualitative rule that
nonlinearity is a necessary condition for chaotic behavior:

\[
\text{if } (\text{linear-system}) \quad \text{then } \text{ode is linear}
\]

This lets any linear model be discarded without performing more-complex op-
erations\(^9\) such as, for example, a numerical integration of the ODE. Figure 4
gives some examples of observations and the facts that the logic system infers
from them. These examples highlight an important feature of PRET’s knowl-
edge.

<table>
<thead>
<tr>
<th>Observ. about state var. (x_i)</th>
<th>Implications for model (f(x, t) = 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>autonomous</td>
<td>cannot explicitly contain (t) (i.e., (f(x) = 0))</td>
</tr>
<tr>
<td>chaotic</td>
<td>cannot be linear</td>
</tr>
<tr>
<td>chaotic and autonomous</td>
<td>order (&gt; 2)</td>
</tr>
<tr>
<td>oscillation and autonomous</td>
<td>imaginary part of one pair of roots (&gt; 0)</td>
</tr>
<tr>
<td>linear</td>
<td>should satisfy (\dot{x}_i = 0)</td>
</tr>
<tr>
<td>constant</td>
<td>should satisfy (\dot{x}_i = 0)</td>
</tr>
<tr>
<td>conservative</td>
<td>(\nabla \cdot f = 0)</td>
</tr>
<tr>
<td>damped oscill. and autonomous</td>
<td>(\nabla \cdot f &lt; 0)</td>
</tr>
</tbody>
</table>

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

\(^9\) 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.
base; not only \textsc{pret}'s inputs and outputs but also its internal reasoning rules are communicable to domain experts.

\textsc{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 (Kuipers, 1992). 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.

4.2 Qualitative simulation

After using its qualitative reasoning facilities to the fullest possible extent and before resorting to the numerical level, \textsc{pret} attempts to establish contradictions by reasoning about the states of the physical system (Kuipers, 1992). It does not do full qualitative simulation (Kuipers, 1986); rather, it envisions the state space of all possible combinations of qualitative values of state variables and parameters. Specifically, \textsc{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.

4.3 Constraint reasoning

Often, information \textit{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 a damped-oscillation observation, the coefficients must satisfy the inequalities $4ac > b^2$ and $b/a > 0$.

\textsc{pret} uses expression inference (Sussman & Steele, 1980) to merge and simplify such constraints (Jaffar & Maher, 1994). However, this approach works only for linear and quadratic expressions and some special cases of higher order, but the expressions that arise in model testing can be far more complex. For example, if the candidate model $\ddot{x} + a\dot{x}^4 + b\dot{x}^2 = 0$ is to match an observation that the system is conservative, the coefficients $a$ and $b$ must take on values such that the divergence $-4ax^3 - 2b\dot{x}$ is zero, below a certain resolution threshold,
for the specified range of interest of $x$. We are investigating techniques (e.g., (Faltins & Gelle, 1997)) for reasoning about more-general expressions like this.

4.4 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 (Char et al., 1991) and simple phase-portrait analysis techniques (Bradley, 1995), 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, are described in more detail in (Bradley & Easley, 1998). PRET does not currently reason about topology, but we are investigating how best to do so (Robins et al., 1998; Robins et al., 2000).

4.5 Parameter estimation and numerical simulation

PRET’s final check of any model requires a point-by-point comparison of a numerical integration of that ODE against all numerical observations of the target system. In order to integrate the ODE, however, PRET must first estimate values for any unknown coefficients.

Parameter estimation, the lower box in Figure 2, is a complex nonlinear global optimization problem. PRET’s nonlinear parameter estimation reasoner (NPER) solves this problem using a new, highly effective global optimization method that combines qualitative reasoning and local optimization techniques. Space limitations preclude a thorough discussion of this approach here; see (Bradley et al., 1998) for more details.

4.6 Benefits of the communicable reasoning framework

Representing PRET’s ODE theory and the associated reasoning techniques in a communicable format that resembles a domain expert’s vocabulary and conceptual framework, as described in this section, has several advantages:

1. Formulating the ODE theory is a reasonably straightforward undertaking; the rules in the knowledge base—e.g., the ones shown in Figure 4—resemble the knowledge presented in typical dynamical systems textbooks.
2. Given a trace of $\text{PRET}$’s reasoning, it is easy to understand why a particular candidate model was ruled out.$^{10}$

3. Similarly, it is easy to examine why a particular candidate model (the result of a $\text{PRET}$ run) did not get ruled out. This point is important if $\text{PRET}$ finds an ODE model that cannot be ruled out based on the union of the observations and $\text{PRET}$’s ODE theory, but the model does not match the user’s intuitions about the target system. This means that the model that passes $\text{PRET}$’s validity test does not pass the user’s mental validity test. This situation may arise for two reasons:

(a) Incomplete observations: The user is aware of some aspects of the target system’s behavior that do not match $\text{PRET}$’s model, but the user did not provide the corresponding observations to $\text{PRET}$. See (Bradley et al., 1998) for an example.

(b) Incomplete ODE theory: The user’s knowledge about ODE theory lets him or her rule out $\text{PRET}$’s model, but $\text{PRET}$’s ODE theory does not include that knowledge.

In the first case, the user simply starts another modeling run, this time supplying the additional observations that help refute the model that $\text{PRET}$ has found in the previous run. In the second case, one has to “teach” $\text{PRET}$ some more ODE theory, extending it so as to enable it to prove the contradiction (between the observations and the model) that the user sees and that $\text{PRET}$ does not see.

One of the most important advantages of $\text{PRET}$’s communicable 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 $\text{PRET}$’s ODE theory. Implementing additional rules—similar to the ones shown in Figure 4—is only a matter of a few lines of Scheme code and/or a call to Maple.

4. Similarly to extending the ODE theory, the user may also want to extend $\text{PRET}$’s arsenal of reasoning modes. Such extensions may result in a more-accurate assessment of candidate models and/or increased performance—because they may facilitate high-level, abstract shortcuts for contradiction proofs. Adding a reasoning mode to $\text{PRET}$’s repertoire amounts to writing two or three Horn clauses$^{11}$ that interpret the results of the reasoning mode by specifying the conditions under which those results contradict observations about the target system.

Please see (Stolle, 1998; Stolle & Bradley, 1998) for a complete discussion of $\text{PRET}$’s reasoning modes.

$^{10}$ We are currently working on ways in which such knowledge—essentially a contradiction proof—can be fed back to the generation phase automatically in order to help guide the choice of the next candidate model. Such approaches are often referred to as discrepancy-driven refinement (Addanki et al., 1991).

$^{11}$ This is explained in the next section.
5 Reasoning control in PRET

PRET's challenge in properly orchestrating the reasoning modes described in the previous section was to test models against observations using the cheapest possible reasoning mode and, at the same time, to avoid duplication of effort. In order to accomplish this, the inference engine uses the following techniques.

5.1 Resolution Theorem Proving

The observations and the ODE theory are expressed in the language of generalized Horn clause intuitionistic logic (McCarty, 1988). PRET's inference engine is a resolution-based theorem prover. For every candidate model, this prover combines basic facts about the target system, basic facts about the candidate model, and basic facts and rules from the ODE theory into one set of clauses, and then tries to derive falsum—which represents inconsistency—from that set.

The special formula falsum 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. For a detailed discussion of PRET's logic system see (Stolle, 1998; Stolle & Bradley, 1998; Hogan et al., 1998).

5.2 Declarative Meta Level Control

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. PRET provides meta-level language constructs that allow the implementer of the ODE theory to specify the control strategy that is to be used.

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. For example, PRET's meta control theory prioritizes stability reasoning about the target system depending on whether the system is known to be linear.\footnote{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.} For a discussion of PRET's meta control constructs, see (Beckstein et al., 1996; Hogan et al., 1998).

5.3 Reasoning at Different Abstraction Levels

To every rule, the ODE theory implementer assigns a natural number, indicating its level of abstraction. The inference engine uses less-abstract ODE rules only if the more-abstract rules are insufficient to prove a contradiction.

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. Since abstract reasoning usually involves less detail, this approach leads to short and quick proofs of the *falsum* whenever possible.

### 5.4 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 is too expensive. On the other hand, many intermediate results are very expensive to derive and would have to be redervied multiple times if they were not stored for reuse.

PRET reuses previously derived knowledge in three ways. First, it remembers what it has found out about the physical system across all test phases of individual candidate models. The fact that a time series measured from the physical system contains a limit cycle, for example, can be reused across all candidate models. Second, every time PRET's reasoning proceeds to a less-abstract level, it needs all information that has already been derived at the more-abstract level, so it stores this information rather than rederiving it.\(^\text{13}\) Finally, many of the reasoning modes described in Section 4 use knowledge that has been generated by previous inferences, which may in turn have triggered other reasoning modes. For instance, the NPER relies heavily on qualitative knowledge derived during the structural identification phase in order to avoid local extrema in regression landscapes. To facilitate this, PRET gives these modules access to the set of formulae that have been derived so far.

In summary, PRET's control knowledge is expressed as a declarative meta theory, which makes the formulation of control knowledge convenient, understandable, and extensible. None of the reasoning techniques described in Section 4 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.

### 6 Related modeling approaches

Section 2 described how engineering modeling fits into the more-general landscape of scientific discovery, modeling of physical systems, and machine learning. Sections 3, 4 and 5 have provided an overview of how PRET was designed as a

\(^{13}\) This requires the developer to declare a number of predicates as *relevant* (Beckstein & Tobermann, 1992), which causes all succeeding subgoals with this predicate to be stored for later reuse. See (Hogan et al., 1998) for more discussion of this.
unique tool to meet the particular challenges of modeling in an engineering setting. Given this background, we are now prepared to briefly review some more-closely related work in more detail.

Some of PRET's roots as an engineer's tool can be found in "the dynamicist's workbench" (Abelson et al., 1989; Abelson & Sussman, 1989). Its 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, (Falkenhainer & Forbus, 1991; Forbus, 1984; Nayak, 1995; Addanki et al., 1991)). 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. 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. Capelo et al. (1998), 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, viscoelastic, allows it to model the same problems as in (Capelo et al., 1998).)

PRET takes a strict engineering approach to the questions of accuracy and parsimony. 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 power and responsibility to ensure that the set of observations and specifications presented to PRET reflect the task at hand.

PRET's goal, then, is to construct the simplest model that matches the observed behavior to within the predefined specifications. Because evaluation 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.
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 by hand. PRET can also be used to solve the kinds of cross-validation problems that arise in the machine learning literature; one would simply use it to perform several individual validation runs and then interpret the results.

Like the computational discovery work of Langley et al. (this volume), PRET makes direct contact with the applicable domain theory, and leverages that information in the model-building process. The theory and methods are of course different; PRET's domain is the general mathematics of ODEs rather than the specifics of biological processes. Many of the research issues are similar, though: how best to combine concrete data and abstract models, how to communicate the results effectively to domain experts, etc.

Koza et al. (this volume) use genetic programming to automatically build directed graphs to model a variety of systems. While the goal is similar to PRET's—automatic construction of a mathematical model from observations—the models and techniques for deriving them are very different. PRET's target systems are nonlinear and dynamic, and ODEs are the best way to capture that behavior. Experts have used these kinds of models for many decades, so the associated domain-specific reasoning is fairly well-developed and can be exploited in an automated modeler. PRET takes this approach, rather than relying on general techniques like genetic programming, neural nets, regression rules, etc.

Like (Muggleton, this volume), PRET relies on mathematical logic to capture domain knowledge in a declarative form. Like (Washio & Motoda, this volume), PRET clearly separates domain-specific facts and general knowledge, making the priorities and connections explicit, and expressing each in a manner that is appropriate for their use.

Other automated analysis tools target nonlinear dynamical systems. The spatial aggregation framework of (Zhao et al., this volume) and Yip's KAM tool (Yip, 1991), among others, reason about the state-space geometry of their solutions. PRET's sensor data analysis facilities—see (Easley & Bradley, this volume) and (Bradley & Easley, 1998)—do essentially the same thing, but PRET then goes on to leverage that information to deduce what internal system dynamics produced that state-space geometry. Its ability to solve this kind of inverse problem—deducing a general, nonlinear ODE from partial information about its solutions—is one of PRET's unique strengths.

The branch of scientific discovery/machine learning research that is most closely related to PRET is the work of Todorovski and Džeroski (this volume). This line of work began with LAGRANGE (Džeroski & Todorovski, 1995), which builds ODE and/or algebraic models of dynamical 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 incomplete data and systems that depend in a nonlinear manner on their parameters, whereas LAGRANGE cannot.

LAGRAMGE (Todorovski & Džeroski, 1997), the successor to LAGRANGE, improved upon its predecessor by incorporating the same kinds of optimization
algorithms (e.g., Levenberg-Marquardt) on which PRET's nonlinear parameter estimator is based. This broadened LAGRAMGE's generality (and its search space) to include models that are nonlinear in the state variables and the parameters.

The main difference between PRET and LAGRAMGE lies in how the initial conditions for the optimization are chosen. Simplex-based nonlinear optimization methods are essentially a sophisticated form of hill-climbing, and so initial-condition choice is a key element in their success or failure. PRET core design principle is to leverage all available information about the system and the model insofar as possible, and this plays a particularly important role in parameter estimation. In particular, PRET uses the arsenal of qualitative and quantitative reasoning techniques that have been described in previous sections in order to intelligently choose initial conditions for its nonlinear optimization runs. This not only broadens the class of ODEs for which it attains a successful fit, but also speeds up the fitting process for individual runs.

Because the reasoning involved in PRET's choice of initial conditions is both qualitative and quantitative, and because both the landscapes and methods of the optimization process are nonlinear, it is only possible to prove that the set of models that is accessible to LAGRAMGE is a proper subset of those that are accessible to PRET. (Indeed, any stronger statement would amount to a general solution of the global nonlinear optimization problem.) Very few optimization landscape geometries are forgiving of bad initial-condition choices, however, and so we believe that the difference between the two sets of models—PRET-accessible and LAGRAMGE-accessible—is large. Apart from this difference, PRET and LAGRAMGE are quite similar, though the design choices and implementation details (e.g., knowledge representations, reasoning modes, etc.) are of course different.

7 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.

PRET's inputs, its outputs, and the knowledge used by its internal reasoning machinery are all expressed in a form that makes this knowledge easily communicable to domain experts. The declarative knowledge representation framework described in this chapter 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 for domain experts to understand and use it. This allows scientists and engineers to use PRET as an engineering tool in the context of engineering tasks, communicating with the program using the application domain's vocabulary and conceptual framework.
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 (Bradley et al., 1998; Bradley & Stolle, 1996)), as well as several interesting and difficult real-world examples, such as a well, a shock absorber, and a driven pendulum, which are described in (Bradley et al., 2001), and a commercial radio-controlled car, which is covered in (Bradley et al., 1998). These examples are representative of wide classes of dynamical systems, both linear and nonlinear. The research effort on this project has now turned to the application of this useful problem-solving tool, rather than improvement of its algorithms. Our current task, for instance, is to use Pret to deduce information about paleoclimate dynamics from radioisotope dating data.

Acknowledgements

Apollo Hogan, Brian LaMacchia, Abbie O’Gallagher, Janet Rogers, Ray Spiteri, Tom Wrench, and particularly Matt Easley contributed code and/or ideas to Pret. The authors would like to thank Pat Langley for enlightening discussions.

Parts of this paper are short versions of material previously published in the journal Artificial Intelligence (Bradley et al., 2001) and are republished here with kind permission of the publisher, Reed-Elsevier.

References


