Artificial Intelligence 51 (1991) 95–143
Elsevier

Compositional modeling: finding the right model for the job

Brian Falkenhainer

System Sciences Laboratory, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA

Kenneth D. Forbus

The Institute for the Learning Sciences, Northwestern University, 1890 Maple Avenue, Evanston, IL 60201, USA

Abstract

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

To represent an engineer's knowledge will require domain theories that are orders of magnitude larger than today's theories, describe phenomena at several levels of granularity, and incorporate multiple perspectives. To build and use such theories effectively requires strategies for organizing domain models and techniques for determining which subset of knowledge to apply for a given task. This paper describes *compositional modeling*, a technique that addresses these issues. Compositional modeling uses explicit *modeling assumptions* to decompose domain knowledge into semi-independent model fragments, each describing various aspects of objects and physical processes. We describe an implemented algorithm for *model composition*. That is, given a general domain theory, a structural description of a specific system, and a query about the system's behavior, the algorithm composes a model which suffices to answer the query while minimizing extraneous detail. We illustrate the utility of compositional modeling by outlining the organization of a large-scale, multi-grain, multi-perspective model we have built for engineering thermodynamics, and showing how the model composition algorithm can be used to automatically select the appropriate knowledge to answer questions in a tutorial setting.

1. Introduction

Armed with a vast body of knowledge ranging from abstract rules of thumb to precise numerical models, engineers guide and simplify their

analyses through selective attention, approximation, and abstraction. During the course of analysis, they often shift between different perspectives and simplifying assumptions, seeking those views which make needed distinctions most apparent. Selecting the perspective and level of detail appropriate for each task is crucial; analyzing every aspect of an artifact using the most accurate models available is generally prohibitive, even for simple artifacts. For example, answering questions about the throughput of a pump in a power plant requires neither consideration of the plant's turbines and condensers, nor use of a quantum mechanical model of the pump's parts and fluid. Systems designed to automate reasoning about the physical world must also exhibit this kind of focused behavior as the complexity and diversity of their tasks grow. Access to multiple models and the ability to form the most appropriate model for each analysis task is crucial for future intelligent computer-aided design, diagnosis, and tutoring systems.

Accomplishing this requires a view of the modeling process that enables systems to explicitly represent and reason about models' underlying assumptions and areas of applicability. In this paper, we describe *compositional modeling*, a strategy for organizing and reasoning about models of physical phenomena that addresses the following problem: given an artifact description and a query, produce a model of the artifact that is commensurate with the needs of the query.

The first step in accomplishing this is access to multiple, alternative models that differ along a variety of dimensions. This raises the problem of how to represent and organize such models. In the first part of this paper, we address this problem by describing a language for expressing both models of physical systems and the assumptions constraining their use. First, we introduce explicit *simplifying assumptions* to state each model's underlying commitments (e.g., abstraction level, approximations, perspective, and granularity) and the conditions under which they are appropriate. This allows us to determine which models are suitable for a given query and physical setting, and allows mutually inconsistent models to co-exist in the same domain theory. Second, we show how to organize a domain theory into semi-independent modular fragments from which a model of a given scenario is composed. This allows fine-grained control for producing parsimonious answers to questions posed, enables adaptation to a vast space of different scenarios, and allows each fragment to be reused in a variety of settings.

Given a query to answer and a space of models in which to search, composing a good model requires answering some additional difficult questions. Which collection of objects should be considered? At what level of granularity? What phenomena are relevant? From what perspective should the phenomena be described? What approximations are allowable? In the

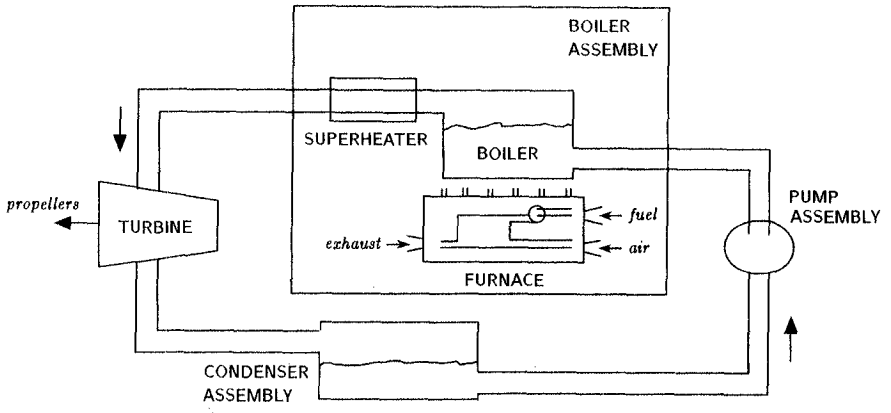


Fig. 1. High-level view of a shipboard steam-powered propulsion plant. The boiler assembly takes in distilled water and fuel and produces superheated steam. The superheated steam enters the turbine, which produces work (i.e., driving the ship's propellers). The steam exhausts from the turbine to the condenser. There it is cooled by circulating sea water and condensed again into liquid, at which point it can be pumped back to the boiler.

second part of this paper, we introduce an implemented technique for composing an appropriate model in response to a given query. It is based on the recognition that the terms in the query provide significant constraint in identifying an appropriate set of modeling assumptions and associated model fragments. For example, given a description of the steam propulsion plant shown in Fig. 1, consider answering the question "how does an increase in the furnace's fuel/air ratio affect the amount of steam flowing in the superheater?" The question alone provides significant clues about the model required: the furnace and superheater, steam flow through the superheater, and the furnace's fuel/air ratio must all be considered. These clues are elaborated to form a complete model by examining the domain theory's dependencies and the artifact's topology.

Finally, composing a useful model is often not sufficient; analysis must often be scoped within an assumed range of behavior. First, parsimonious models do not necessarily produce parsimonious answers. Second, the validity of a model's underlying approximations is generally limited to a restricted range of behavior. We introduce a second class of modeling assumptions called *operating assumptions* which describe the kinds of behavior relevant to a task (e.g., steady-state). This delimits an approximation's validity and can increase the efficiency of simulation by ruling out the consideration of many possibilities.

1.1. The problem

We call the system or situation being modeled the *scenario*, and its model the *scenario model*. One way to construct a scenario model is to directly create a model of that specific situation for a particular purpose. This is a common approach in many engineering domains, where special-purpose simulators are built by hand (typically a FORTRAN program) and the model is encoded as part of the simulator. A change to either the device structure or the task requirements generally requires rewriting the simulator.

Alternatively, we can take an indirect but more robust route—build first a general-purpose *domain theory* that describes a class of related phenomena or systems. A domain theory consists of a set of *model fragments*, each describing some fundamental piece of the domain's physics, such as processes (e.g., liquid flows), devices (e.g., transistors), and objects (e.g., containers). Ideally, a scenario model can then be built by instantiating and composing elements of the domain theory. This too is a common approach in both AI and engineering. Device-centered approaches, both numeric (e.g., SPICE [27,36]) and qualitative [11,54], provide catalogs of devices that can be wired together to build scenario models. Qualitative process theory [16] organizes domain models around quantified descriptions of *processes* and *views* that can be automatically instantiated to form scenario models. QPC was recently developed to provide the same capability for QSIM [7].

Neither approach suggests how to selectively focus attention on a scenario's relevant subset of objects, parameters, and behaviors. Nor do they suggest how alternate models of the same process or device should be organized and automatically chosen for a given task. We define the *model formulation problem* as follows:

Given:

- a *scenario description*: this includes its physical structure S and a (possibly empty) set of statements about its behavior (e.g., initial conditions, steady-state, range limits, etc.);
- a domain theory Th, consisting of a set of *domain model fragments* $\{m_1, \ldots, m_n\}$ and a set of rules constraining their use;
- a *query* about the scenario's behavior.

Produce:

- the most useful, coherent *scenario model* $M = \{m_i\sigma_{i_j}, \ldots, m_k\sigma_{k_l}\}$ for answering the query, where σ_{i_j} is a binding from the variables in m_i to objects in S and each instance $m_i\sigma_{i_j}$ is unique.

Our basic approach is illustrated in Fig. 2. In this approach, the modeling process is a three-stage cycle:

(1) model composition,

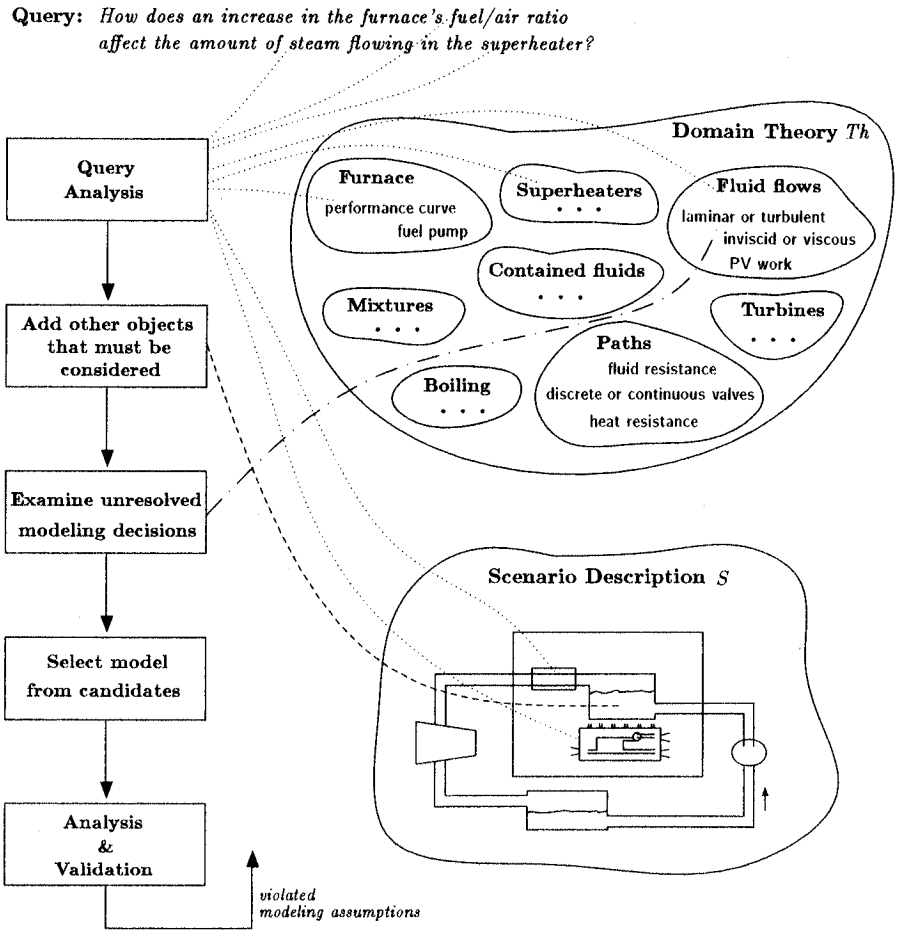


Fig. 2. Overview of the model composition process.

(2) use and validation of the model, and
(3) model revision if necessary.

The domain theory consists of a set of fine-grained model fragments, each explicitly conditioned on the physical setting to which it applies and the modeling assumptions upon which it rests. Model composition begins by matching the terms of the query to their referents in the domain theory, which suggests an initial set of necessary model fragments. This set is then elaborated using the domain's constraints and the scenario's topology. The resulting scenario model should be *most useful* (simplest suitable to intended use) and *coherent* (internally consistent). These criteria are defined further in Section 3. The model's validity may depend on assumptions about

unknown aspects of the system's behavior. Analysis using the model may uncover inconsistencies between the system's assumed and derived behavior which are then used to formulate a more appropriate model.

Model formulation is a difficult problem. Our solution operates under several simplifying presuppositions, which we believe are reasonable for many tasks, including tutoring and many aspects of design analysis and verification (Section 6 discusses how these limitations might be overcome). First, we presume that decomposition of the scenario, if any, is in the form of a strict, structural part-of hierarchy. This, for example, presumes that the interesting boundaries of the scenario can be characterized by a single structural decomposition.

Second, we presume that the needs of the analyst (human or mechanical) can be expressed as a list of requisite modeling terms (generated by the analyst or an interface procedure). That is, the information needed to derive an appropriate scenario model can be gleaned by matching the terms in the query to the terms introduced by the models, without further analysis of the query's implicit information needs.

Third, we presume that for each approximation encoded in a model, there is a set of *a priori* constraints on the behavior that determine when the approximation is valid. These constraints are monitored during analysis to ensure that the error introduced by each approximation is within an acceptable range. For many domains, there is sufficient experience that such limits are known. For example, Reynolds' number is often used in fluid dynamics to delimit when a flow is laminar or turbulent (e.g., "under normal conditions, transition occurs at $Re \approx 2300$ for pipes" [21]). Other examples include using the Mach number to delimit when a gas flow is incompressible and the Biot modulus to delimit when a lumped-parameter analysis of unsteady-state heat conduction is valid [21,52]. The fundamental problem in removing this presumption is that reasoning about an approximation's cost/accuracy tradeoffs can be as expensive as not making the approximation. (See [1,42,51] for some initial work in this area.)

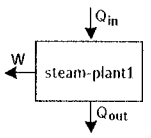
We do not presume that the model fragments use a particular form of mathematics, save that it be "compositional" (Section 2.2). Because one of the original motivations behind qualitative physics was to capture the tacit knowledge behind the use of physical laws, the intellectual roots of our work lies within that community. However, our approach is applicable to quantitative, as well as qualitative representations for continuous systems. In particular, we illustrate our techniques with two classes of examples: ordinary differential equations and QP theory representations.[1]

Even with these simplifications, our compositional modeling algorithm

[1] In this paper, we treat these cases separately. See [19,30,41,43,55] for work on the important problem of integrating qualitative and quantitative representations.

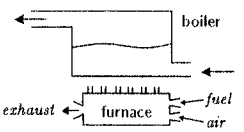
Q: *What affects the efficiency of the plant?*

A: The efficiency of the plant is affected positively by the work rate of the turbine (W). It is also affected negatively by the energy input to the plant (Qin).



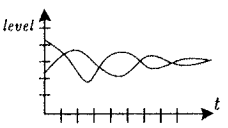
Q: *How does the furnace's fuel/air ratio affect the boiler's steam production?*

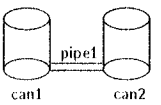
A: When the fuel/air ratio is below peak efficiency, an increase in the fuel/air ratio causes an increase in the boiler's heat rate, an increase in the boiler's heat rate causes an increase in the amount of steam in the boiler.



Q: *The level in container 1 is initially 1.9 meters. The level in container 2 is initially 0.8 meters. Plot the behavior of the containers' levels.*

A:





Because the flow rate is high (Re > 2300), we must assume turbulent flow through pipe1.

Fig. 3. Tutorial question answering about a shipboard steam propulsion plant. Here are some answers generated by an implemented query system using the steam plant model. The questions were formulated in a specialized query language. The explanations are automatically generated by the program.

provides significant leverage. We have been able to analyze a hypothetical steam propulsion plant (Fig. 1) using a large, multi-grain, multi-perspective domain model of engineering thermodynamics. Using the compositional modeling strategy, our system analyzes each query and automatically computes an appropriate set of modeling assumptions. Some questions that can be answered with the model currently are illustrated in Fig. 3. By focusing attention, perspective, and granularity in response to the question posed, each answer took only a few minutes to compute on a Symbolics XL400.

2. Compositional models

What are the characteristics of a good modeling framework? First, the desire to adapt to a wide range of scenarios and tasks in a parsimonious manner argues for finely-tuned control over the dimensions along which a model can vary. Second, the desire to simplify knowledge acquisition, maintenance, and reuse argues for the ability to succinctly state individual

modeling concepts, together with a description of when they apply, as independently as possible. This argues for a fine-grained, modular approach to modeling.

This observation is the heart of the compositional modeling strategy. A domain theory consists of model fragments that can be assembled as needed to form a scenario model under a particular set of modeling assumptions. These fragments need not correspond directly to complete devices or processes; to the extent that individual modeling decisions can be stated independently (e.g., friction, compressibility, etc.), a finer level of decomposition is possible. Of course, the modeling assumptions underlying the model fragments are not always independent. Some assumptions only make sense if others hold, while some sets of modeling assumptions conflict. But as far as possible, the model fragments are conditioned on the minimal set of antecedent assumptions, so that they will be as widely applicable as possible.

This section describes how to organize a domain theory using the compositional modeling strategy. It begins by defining the notion of *model fragment* and describes a language for expressing domain models. Next, we define what it means for a model to be compositional. Finally, we describe techniques for organizing modeling assumptions. Examples from the thermodynamics model are used throughout for illustration.

2.1. Conditionalized model fragments

A set of equations in isolation does not constitute a model, for it implicitly corresponds to some physical setting, a set of assumptions and approximations, and some relevance criteria. To automate model formulation and facilitate model reuse, this context must be made explicit. Therefore, we define a model fragment m to be a quadruple $\langle I, A, O, R \rangle$, where

- I is a set of conditions defining the structural configuration of individuals to which the model applies (e.g., two containers connected by a pipe).
- A is a (possibly empty) set of assumptions concerning the model fragment's relevance to questions of interest (e.g., consider the flow through pipe `p1`) and stating its underlying abstractions and approximations (e.g., frictionless flow).
- O is a (possibly empty) set of operating conditions delimiting the model's behavioral scope. For example, a transistor has several operating modes, each delimited by ranges on parameter values (e.g., $V_{CE} > 0.7$) and characterized by a different device model. O includes inequalities between the individuals' parameters and constants (e.g., *pressure*(`can1`) > *pressure*(`can2`)), conditions on the activity of other

model fragments (e.g., *active*(heat-flow,pi0)), and operating assumptions (e.g., steady-state).

- R is the set of relations imposed by the model fragment. It contains assertions about the individuals in I and constraints on their parameters. These constraints may be either qualitative (e.g., qualitative proportionalities) or quantitative (e.g., ordinary differential equations).

Logically, a model fragment can be encoded as the first-order implication

$$I \wedge A \wedge O \rightarrow R.$$

We find the distinctions highlighted by the definition of model fragment desirable for two reasons. First, it makes distinctions that people tend to express and we have found to be natural in writing a variety of domain theories (i.e., separating the notions of a configuration of objects, their behavior, and the assumptions under which a model of these objects is applied). Second, these distinctions can be utilized to reduce the computational effort of the model composition algorithm, as described in Section 3.

Our current implementation supports a simple modeling language that contains four basic primitives. One is for organizing modeling assumptions and is discussed in Section 2.3.3. Second, ground atomic assertions are made with the form `assert!`. Third, rules (defined by `==>`) are used to state domain rules, part-of and generalization hierarchies, and interactions between modeling assumptions and domain facts. Fourth, domain model fragments are defined using `defModel`, which is of the form

```
(defModel name-form
  Individuals i-spec
  Assumptions list-of-assumptions
  OperatingConditions list-of-operating-conditions
  Relations list-of-consequent-relations)
```

where *name-form* is an expression with variables which provides a term designating an instantiation of this model and *i-spec* is the same set of variables with conditions on their potential bindings.[2] Two classes of mathematical constraints are currently allowed in the relations field:

(1) QP theory constraints (i.e., `I±`, `Qprop±`, `Q=`, `Correspondence`) or

[2]This generalizes the *process* and *view* primitives introduced in QP theory. In particular, a `defModel`'s `Individuals` field corresponds to the union of QP's individuals and preconditions. In [14], we placed modeling assumptions in QP's individuals specification, which was awkward and less efficient. The `OperatingConditions` field is a generalization of QP's quantity conditions which allows symbolic statements about behavior (e.g., steady-state) in addition to the prior inequalities and process activation statements. A `defModel`'s `Relations` are the union of QP relations and influences.

```
(defModel (THERMAL-LIQUID ?CL)
   Individuals ((?cl :conditions (Contained-liquid ?cl)))
   Assumptions ((CONSIDER (thermal-properties ?cl)))
   Relations ((Qprop- (viscosity ?cl) (temperature ?cl :absolute))
              (not (greater-than (A (temperature ?cl :absolute))
                                 (A (temperature (boil ?cl) :absolute))))))

(defModel (CONTAINED-LIQUID-GEOMETRY ?CL)
  Individuals ((?can :conditions (Fluid-container ?can))
               (?cl  :conditions (Contained-liquid ?cl)
                                 (container-of ?cl ?can)
                                 (substance-of ?cl ?sub)))
  Assumptions ((CONSIDER (Geometric-Properties ?can)))
  Relations ((Quantity (level ?cl))
             (= (level ?cl) (/ (* 4 (mass ?cl))
                               (* (density ?sub) PI (expt (diameter ?can) 2))))
             (= (pressure (bottom ?can) :absolute) (* (level ?cl)
                                                      (density ?sub) G))))
```

Fig. 4. Domain model fragments defining a few properties of liquids.

(2) numeric constraints (i.e., algebraic and ordinary differential equations).

An important element of the relations field is the declaration of quantities, which represent continuous parameters of the thing(s) being modeled. The predicate `Quantity` is used to state that an object has a quantity of a particular type, as in

```
Quantity(diameter(can1)).
```

All quantities used in an equation must be defined at the time the equation is stated to hold. Quantity declarations are important because they are the major link for matching models to task requirements. It is generally easy to extract from a query a set of quantities that must be considered for a task to even make sense. And since what quantities are defined depends on what modeling assumptions are in force, this information usually provides the bulk of the suggestions about what modeling assumptions are appropriate.

As an example, Fig. 4 shows two different aspects of a model of liquids. The first model fragment describes the constraints on the parameters of a contained liquid (i.e., a piece of stuff individuated by being in a container) when thermal properties are being considered. The second model fragment describes quantitative relationships that hold when the container's geometric properties are being considered. Importantly, neither statement alone defines "the model" of liquids. Nor in fact do they collectively: Other `defModel` statements provide different aspects of the model of liquids, such as defining the properties under which contained liquids exist, and stating

that when considering the thermal properties of an object it then has the continuous properties `heat` and `temperature`. This fine-grained decomposition of different aspects of a model is a hallmark of compositional modeling. Each model fragment only needs to specify the conditions under which it should be used. Model fragments do not need to (and indeed, should not) attempt to enumerate inappropriate conditions nor allowable combinations of elementary models. This modularity simplifies domain model construction, improves reusability, and enables the system to respond to the specific needs of each task.

2.2. Composable models

A central component of a model fragment is a set of equations specifying constraints between quantities. To be composable, the mathematical representation and inference procedures should support a small set of general modeling operations.

Parameter substitution

Modularity and reusability are supported by stating physical laws in their most general form, using functional parameters when possible. For example, the frictional component of fluid flow (head loss) is computed very differently depending on whether the flow is laminar or turbulent. With head loss as a separate parameter (h_l), the general form of Bernoulli's fluid flow equation can be stated once and used in different contexts.[3]

Shared individuals and parameters

Composing a model of an artifact from models of its components requires a way for the component models to interact. This is typically done through shared parameters and individuals. For example, lumped-element models of electrical circuit components typically communicate via shared nodes (e.g., the voltage on a node in the circuit).

These two are commonplace modeling operations. An additional, powerful source of composability comes from the ability to partially specify equations.

Composable functions

Ideally, elementary models do not require knowledge of the composite environments to which they may be applied (e.g. "no function in structure" [11]). However, some equations cannot be stated in a computable form

[3]This does not imply that actual computation must be performed in this modular manner. A separate compilation step after model formulation may be used to increase computational efficiency.

until a fixed artifact and a fixed set of phenomena are identified. For example, the law "the change in a container's fluid is equal to the sum of all the flows into and out of the container" cannot be stated as a well-formed equation until all flows into and out of the container are known. Still, we would like to state how the flow of liquid out a container's portal affects the amount of liquid in the container *once, as a generally applicable law*, independently of what flows may be occurring in a specific instance.[4] To allow explicit specification of such interactions, we assume a general class of partial constraints called *composable functions*. These are *n-ary* functions of arbitrary n (e.g., summation) that are declared piecemeal through membership assertions. Formulating the aggregate function (and fixing the value of n) in a specific context requires the *closed-world assumption* [40] that all of its elements are known. This can occur once the artifact and phenomena to be analyzed are identified.

We currently use the two composable functions introduced by QP theory [16] which are together known as *influences*. If Q_1 *influences* Q_2, then a change in Q_1 will cause a change in Q_2, all else being equal. The net change of an influenced quantity can be computed once all of its influences are known. If a quantity is *directly influenced*, its derivative equals the sum of all of the direct influences on it. $I+(Q_0,Q_1)$ states that Q_1 directly influences Q_0 positively; $I-(Q_0,Q_1)$ states that Q_1 directly influences Q_0 negatively:

$$\frac{dQ_0}{dt} = \sum_{p|I+(Q_0,Q_p)} Q_p - \sum_{n|I-(Q_0,Q_n)} Q_n.$$

For example, the flow from container c_1 out portal p_1 can be stated as

```
I-(amount(fluid,c1),flow-rate(p1)).
```

Quasi-static assumptions like "the flow in equals the flow out" can be stated separately and explicitly (e.g., `d/dt amount(fluid,c1) = 0`).

The *indirect influence* (also called *qualitative proportionality*) Qprop $\pm(Q_0,Q_1)$ states a much weaker, qualitative relationship. "Q_0 is qualitatively proportional to Q_1" means that there exists a function f such that $Q_0 = f(\ldots,Q_1,\ldots)$ and Q_0 is increasing monotonic in its dependence on Q_1.

2.3. Modeling assumptions

The specification of each domain model fragment includes the modeling assumptions under which it holds (unless it is universally relevant). This provides control over its instantiation and use so that only the relevant

[4] In device-centered models [11], such relationships are not explicit in the representation, but rather are computed once all shared nodes have been identified (i.e., the sum into and out of the node is 0). This makes a quasi-static assumption that is also left implicit.

aspects of a situation are examined. Models whose modeling assumptions contradict knowledge of the scenario or the current focus of attention end up not even being instantiated.

We state modeling assumptions as predicates over specific phenomena or systems, not globally over an entire scenario, so that intricacy or simplification can be introduced selectively where appropriate. For example, we would like to model some flows in a steam plant as laminar and some as turbulent, some containers as having finite capacity and some as being infinite, and some mechanical parts as rigid and some as elastic.

To support this, we assume the objects in the scenario are organized into *systems*. A system is either a primitive object or a named collection of constituent systems. For example, a container is a primitive object, and the boiler assembly is not, since it consists of a furnace, boiler, superheater. The relation `Part-of` holds when one system is part of another. For example, the boiler is *part of* the boiler assembly. The `Part-of` relation is limited to a system and its immediate subsystems (i.e., it is not transitive). The function `components` maps from a system to its set of parts; if s_1 is a primitive object, then `components`(s_1) is the empty set. The transitive closure of `Part-of` is expressed by the relation `Sys-Contains`; system s_1 *contains* system s_2 if either s_2 is a part of s_1 or s_2 is a part of some system s_3 which itself is contained in s_1. We currently assume that systems always form a strict hierarchy. Its root is always a system called `:scenario`, which contains all objects in the scenario.

Conceptually, modeling assumptions are divided into two categories: *simplifying assumptions* (Section 2.3.1) and *operating assumptions* (Section 2.3.2). Organization and control of modeling assumptions are provided by *assumption classes* and domain-specific constraints, which are discussed in Section 2.3.3.

2.3.1. Simplifying assumptions

Simplifying assumptions make explicit a model fragment's underlying approximations, perspectives, and granularity. They provide the majority of the vocabulary for representing the problem solver's decisions about how to model the scenario. We require all simplifying assumptions to take the form

`CONSIDER`(*AsnType*(*system*))

where *AsnType* is a predicate denoting the specific kind of assumption and *system* is the subject of the assumption. Thus

```
CONSIDER(static-friction(tower-of-cards32))
```

enjoins us to think about static friction when reasoning about `tower-of-cards32`.

The collection of CONSIDER assumptions forms the groundwork for any particular analysis. They are classified into three categories:

Ontology assumptions

The first, and perhaps most basic, task in formulating a model is selecting the appropriate method of description. What coordinate system should be used? For example, should the analysis be over time or position? What should be the basic primitives of the analysis—control volumes, particles, lumped elements, etc.? Several distinct varieties of ontologies are used in traditional engineering. For example, *Lagrangian* methods of description (such as particle and rigid-body dynamics) track identifiable elements of mass [21,52]. On the other hand, *Eulerian* methods of description (such as control volume analysis of fluids) track the properties of a flow at specific points over time. Qualitative reasoning has also adopted both methods of description in several forms.

To date, we have made use of four ontologies.

- The *contained stuff* ontology [16,24] is an Eulerian view used to model static and dynamic fluids and their containers. It is most similar to the fluid control volumes of classical thermodynamics. Models using the contained stuff ontology are predicated on ontology assumptions of the form

  CONSIDER(fluid-cs(*system*)).

- The *energy flow* ontology is used to analyze the flow of energy (both heat and work) through a system (cf., energy flow diagrams).
- The *molecular collection* ontology [6] is a Lagrangian view that follows the movement of a localized unit of fluid during flow.
- Finally, *mechanics* is used for the dynamic analysis of mechanisms.

Unlike the other simplifying assumptions, in our approach ontological commitments are fundamental to each analysis and must be global. Thus, any ontological assumptions that are made, and there must be at least one, are applied uniformly to all systems and phenomena under consideration (i.e., they all apply to the system :scenario). When consistent, multiple ontological assumptions may hold. For example, an energy flow analysis often occurs with a mass flow analysis. On the other hand, Eulerian and Lagrangian models of the same phenomenon are always mutually exclusive due to their use of different coordinate systems. However, an analysis in one ontology may make use of results gleaned from a previous analysis using another ontology [6,31,39].

Grain assumptions

Crucial to analyzing large systems is the fact that not all objects in the system need be considered for every analysis task. First, objects outside the current area of concern can simply be ignored. Second, abstractions allow collections of objects to be considered as a single, aggregate entity. A fluid path, for example, can consist of dozens of valves. Yet when reasoning about the system one typically speaks of this path as an entity. Control over which objects to explicitly consider in an analysis is organized around assumptions recognizing their existence, which we call *grain assumptions.*

We use the same `CONSIDER` operator to state grain assumptions. Their syntax is

`CONSIDER(exists(`*system*`))`.

When it holds, the existence of *system* is considered and it is placed within the scope of the current analysis. This means some model for it must be included in any coherent scenario model. For example,

```
CONSIDER(exists(boiler))
```

forces a model of the boiler to be included in an analysis, rather than focusing on some subsystem of it (such as the steam tubes) or treating the entire boiler assembly as a black box. Constraints governing the use of grain assumptions, particularly how considering some objects requires considering others, are described in Section 3.2.

Approximations and abstractions

Approximations are used to construct simplified and (typically) easier to use models at the cost of reducing accuracy. Their utility is in part a function of the scenario's operating conditions, the accuracy requirements of the task, and the computational benefits they afford. Approximations appear in a variety of forms. Some correspond to ignoring influences that are (presumably) insignificant in the scenario under consideration. For example, the *inviscid flow* approximation assumes that the fluid viscosity is zero and thus ignores dissipative effects. While this is fine for many analyses of fluid systems, the assumption is only valid under a limited set of conditions (e.g., low velocity flows). Other approximations of this form include incompressible fluids, inelastic objects, and frictionless motion. Some approximations represent simplifying assumptions about the structure of the environment. For example, our fluid models typically assume level fluid paths. Additionally, a fuel tank can be modeled as an infinite source or as a container with finite capacity limits.

Abstractions reduce the complexity of the modeling language, usually at the cost of reducing the information available (e.g., detail or granularity) and increasing ambiguity, but without reducing accuracy. For example, a fluid

valve can be modeled as either a discrete, on/off switch or as a continuous, variable resistance. Additionally, geometry can often be compiled out of a description, as when reducing a three-dimensional configuration to an equivalent one-dimensional description.

Some simplifying assumptions are difficult to categorize, or have both abstraction and approximation aspects that are difficult to separate. Although the distinction is in principle an important one (see [51] for a good start), our techniques do not require these distinctions. Success depends on the modeler's ability to state within the domain theory when a simplifying assumption is or is not appropriate.

Approximations and abstractions are again represented using CONSIDER. Unlike ontological assumptions and grain assumptions, the constraints on approximation and abstraction assumptions are very domain-specific. For example, one would not wish to uniformly consider all containers as finite or infinite, since one may not be worried about capacity limitations in parts of the system not under direct consideration. Domain-specific constraints between these assumptions must be explicitly encoded either as rules in the domain model or as parts of domain model fragments themselves. For example, in our models it does not make sense to consider the portals that connect a container to a fluid path unless one is willing to consider the geometric properties of the container. The reason is that the only thing which distinguishes a portal is its height, and if geometric properties are ignored, reasoning about portals provides no leverage. This constraint is enforced by making the consideration of portals imply the consideration of geometric properties.

2.3.2. Operating assumptions

Engineers constantly use default assumptions about behavior to manage complexity. For example, when trying to figure out how a system containing a heat exchanger works, engineers tend to assume that the fluid in the hot leg is hotter than the fluid in the cold leg. If the system is operating as intended, making this assumption saves effort because the other two alternatives (i.e., the temperatures being equal or the cold leg temperature being higher than the hot leg temperature) need not be considered. If the system is not operating as intended, then the engineer's predictions will be wrong and the analysis must be re-performed to consider the other alternatives.

Operating assumptions have two roles. First, they focus simulation. For example, if simulation progresses outside the assumed operating range, then it can be halted and the reasons noted. Likewise, qualitative simulation [11,16,28,54] can be constrained by ruling out entire classes of behavior (e.g., assuming all containers are non-empty eliminates the many possible combinations of empty and non-empty containers).

Second, operating assumptions can be tied to approximations by indicating the range of parameter values for which they are valid. For example, the simplifying assumption *laminar flow* is often conditioned on the operating assumption that Reynolds' number is less than 2300. Introductory physics textbooks make their analyses of pendulums tractable by assuming $\sin(x)$ equals x, for suitably small x. Automating model formulation requires providing a language that makes these assumptions explicit, so they may be reasoned about.

Operating assumptions state constraints on possible parameter values, either directly via inequalities or indirectly via symbolic assertions, which in turn define a set of parameter constraints. We presently use three general categories.

Local restrictions on quantity values

These consist of inequalities between quantities and constants. For example,

```
Op-Assumption(greater-than(temp(boiler-steam),
                           temp(condenser-steam)))
```

constrains the temperature of steam in the boiler to be higher than the temperature of steam in the condenser (when the two temperatures are defined).[5]

Operating modes

An operating mode is simply a named collection of local restrictions. For example, the description of a heat exchanger above is an informal specification of the "normal mode" of heat exchangers. It would be stated as

```
Mode(heat-exchanger-1,normal)
```

and then used in conjunction with rules defining the normal mode for a heat exchanger. A steam plant has several operating modes, starting from "cold iron" and ending in "full steam", and each subsystem has modes as well.

[5]We use the `Op-Assumption` form, rather than assuming the inequality directly, so that the assumption is defined even when the quantities are not (i.e., temperature of boiler steam when the boiler is empty).

Steady-state assumptions

These state that all derivatives for some class of parameters is zero. Steady-state assumptions are ubiquitous in engineering analyses. They appear in two forms. First, we use the assumption applied to specific quantity instances as a basic primitive. For example,

```
Steady-State(temperature(boiler-steam))
```

constrains the derivative of `temperature(boiler-steam)` to be zero when it is defined. Second, we have found statements over a system and quantity type to be useful. For example,

```
Steady-State(boiler-assembly,temperature)
```

applies the steady-state assumption to all of the boiler assembly's components for which temperature is defined. This constraint is defined by the rule

```
((steady-state ?system ?q-type)
 (part-of ?component ?system)
  ==>  (steady-state (?q-type ?component)))
```

We presently have no general techniques for mapping from a query to an appropriate set of operating assumptions.[6] Operating assumptions are imposed if they are (1) explicit in the problem statement (i.e., scenario description or query) or (2) required as a by-product of selecting models (e.g., if you want to know about the flow of steam from the boiler to the turbine, the very question presumes that there is steam in both the boiler and turbine and that its conditions are such as to ensure a flow).

2.3.3. Assumption classes

Some collections of assumptions represent natural groupings that can be reasoned about as a whole, such as the alternative ways to model the same aspect of an object or phenomenon. We organize such groupings into mutually exclusive sets called *assumption classes* [2,9]. An assumption class captures one dimension along which a modeling choice must be made. For example, a fluid can be modeled as having zero viscosity (inviscid, frictionless flow), a standard nonzero viscosity, or a non-Newtonian viscosity (e.g., toothpaste). Not all dimensions are relevant in all contexts. Therefore, we also require a condition to state when an assumption class is relevant and a choice must be made along that dimension. For example, the fluid viscosity dimension is only relevant when answering questions about fluid flow. Assumption classes are declared with the form

[6]Stallman and Sussman [46] describe techniques for making behavior assumptions in response to ambiguity encountered during problem solving.

(defAssumptionClass c $(a_1, \ldots, a_n)$)

where condition c is an atomic sentence containing the (possibly empty) set of free variables ν and $(a_1, \ldots, a_n)$ is a mutually exclusive set of atomic sentences (e.g., CONSIDER statements) whose free variables, if any, must be from ν. It is logically equivalent to

$$\forall \nu \; c \rightarrow a_1 \vee \cdots \vee a_n, \qquad \forall a_i, a_j, \; i \neq j, \quad \neg a_i \vee \neg a_j.$$

We say that the assumption class is *active* when c holds. Any scenario model must include exactly one assumption from each active assumption class. Inactive assumption classes are ignored, and none of their constituent assumptions are included. Intuitively, when the class condition holds, one and only one of the assumptions associated with that class must hold in the scenario model. Additional information about the conditions under which each assumption class is most relevant or is inappropriate can then be specified independently via implications whose consequent is c.

In our current scheme, the order of the assumptions $(a_1, \ldots, a_n)$ is important. We assume that models based on assumptions earlier in the list are less costly, in some sense, than models based on assumptions later in the list. This context-independent cost estimate is an oversimplification, but has proved quite useful. The extension to a scheme where a task-specific cost procedure is provided to induce an ordering on assumptions is straightforward. However, specifying a good task-dependent cost procedure is a difficult, open research problem.

As an example, our models of fluid flow include the `fluid-viscosity` assumption class, which controls how the viscosity of a fluid flow process is modeled. Its definition and the condition that it becomes relevant for each fluid flow process under consideration are stated as follows:

```
(defAssumption-class (fluid-viscosity ?pi)
  ((CONSIDER (inviscid ?pi))
   (CONSIDER (viscous ?pi))
   (CONSIDER (non-newtonian ?pi))))

((model-application fluid-flow ?pi)
  ==>  (fluid-viscosity ?pi))
```

Additional constraints can then be placed on these assumptions to ensure that composition of model fragments results in a coherent scenario model. For example, if the question concerns head loss, we should rule out the `inviscid` assumption. Likewise, selection of the `turbulent` assumption (from the `flow-regime` assumption class) is inconsistent with selection of the `inviscid` assumption (from the `fluid-viscosity` assumption class). The use of assumption classes is discussed further in Section 3.3.

3. Model composition

In a rich domain model there are a variety of grain sizes and perspectives. What constraints drive model composition? In the introduction, we stated that a scenario model must satisfy two criteria: *coherent* and *most useful.*

The easiest criterion to guarantee is coherence. By coherent, we mean that modeling decisions are consistent with both the physical and organizing principles of the domain, as defined by the domain theory. An example of violating the domain's physics would be to consider a fluid flow as both turbulent and inviscid (i.e., frictionless). An example of violating domain-specific organizing principles would be to attempt to use an equation concerning head in a model that ignored geometric properties (to calculate head one must have height, which in turn requires modeling geometry at least to some degree).

The more subtle criterion is to maximize utility. Utility is a tradeoff between the (relevant) information acquired and the cost of acquiring it. On the one hand, the scenario model should be sufficient to answer the query; a model of a ship's boiler, no matter how good, is not a reasonable foundation for analyzing its condenser. On the other hand, the model should contain as little extraneous detail as possible and require the least problem-solving effort to achieve the desired answers. For instance, a designer calculating the efficiency of an industrial process probably does not benefit from a detailed analysis of transient behaviors and startup states.

In any responsible modeling strategy sufficiency must dominate. We select the minimal scenario model that satisfies the query's requirements. Sufficiency has two aspects. The first can be characterized as *aboutness* or *relevance*: the scenario model includes all the aspects of the system required to perform the analysis (ontological commitments, parameters of interest, etc.). An example of failure by making an inappropriate ontological commitment would be to attempt to model the steam plant as an abstract heat engine when asked about the mass flows which occur in it. An example of failing to contain the parameters of interest is ignoring thermal properties when a student specifically asks about the thermal effects of boiling. The second aspect of sufficiency is *accuracy*: the scenario model must contain enough information to provide an answer that is accurate enough for the task. An example of inappropriate degree of accuracy is using a qualitative model when a numerical prediction is called for, or using a detailed numerical model when a student only wants to know what parameters can affect something.

To illustrate the basic issues that arise in model composition, we start with a simplified example. Suppose we are given the structural description S and domain theory Th shown in Fig. 5. C(*assumption-type*) represents a modeling assumption; r_i and ac_i indicate rules and assumption classes,

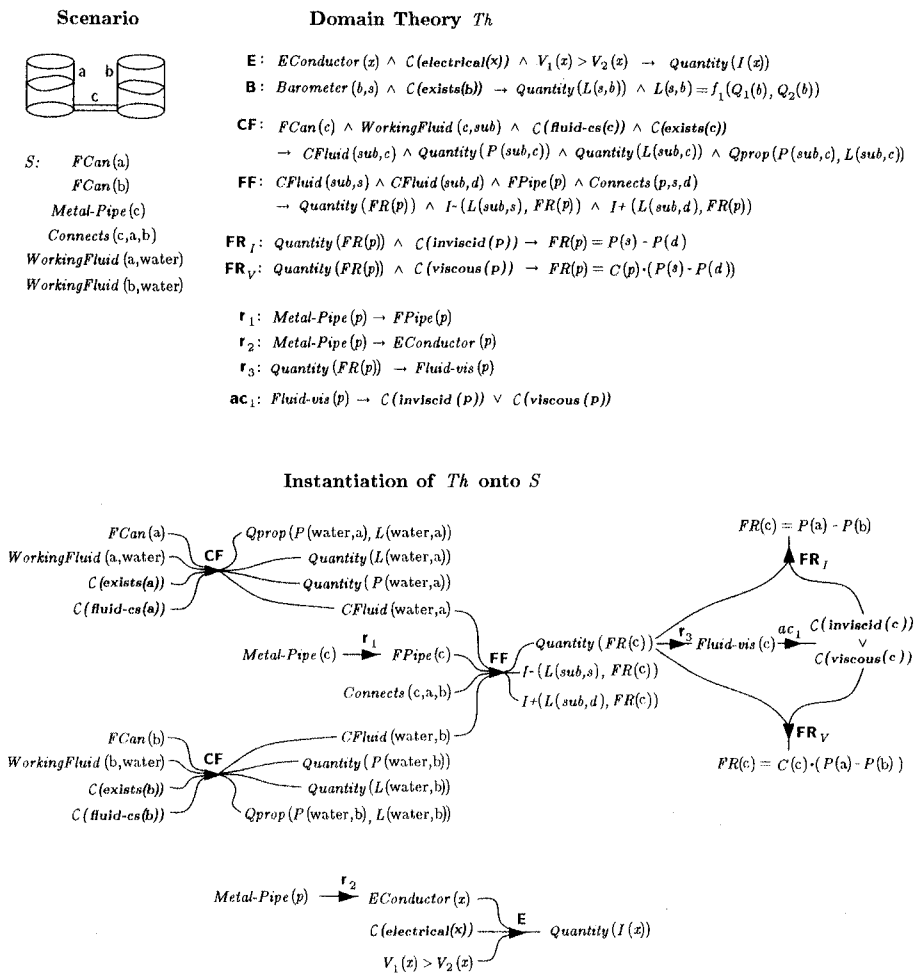


Fig. 5. Simplified view of model composition.

respectively. Suppose as our query we are interested in the behavior of the water levels in the two cans. The input to our algorithm (Section 3.1) is a set of ground expressions (e.g., quantity declarations, relationships, etc.) that must be present in the composed model. For this example, the query would be

$$\{Quantity(L(\text{water}, a)), Quantity(L(\text{water}, b))\}.$$

The structural constraints of the models' individuals specifications and the behavioral constraints of their operating conditions indicate which behaviors

are possible. The query indicates which behaviors are of interest. These various constraints combine to suggest an appropriate model. For example, the query suggests that each can should be modeled as a *fluid container* (as opposed to *heavy object* or *rusting object*). Because there are two contained fluids connected by a pipe, there is also potential for flow. Thus, the metal pipe's behavior qua *fluid path* is relevant. In this example, this occurs as a side effect of the pipe's connection to two fluid containers whose fluids are being considered. Other models that could potentially apply to the pipe, such as being an *electrical conductor*, need never be considered because they do not interact with models supporting the query about fluid levels.

Before describing the basic algorithm, two issues must be examined that are critical to identifying the form of the algorithm's search space and output. The first issue that arises is how to identify candidate models—which model fragments are both applicable to the scenario and describe L(water,a) and L(water,b)? Model fragment ***E*** is irrelevant to the query because it does not cause consideration of any $L(s,c)$. ***B*** is irrelevant to the scenario because *Barometer*(b,s) does not follow from S. ***CF*** appears to be relevant, but we must ensure that its requisite assumptions are consistent with $S \cup Th$. The search for potentially relevant model fragments involves a tradeoff between minimizing the amount of exploration and exhaustively checking all the constraints in an efficient manner to ensure a coherent scenario model. We use the scenario description S as the principle source of focus by searching only in the space of model fragments that are relevant to the scenario. Recall that each model fragment is logically equivalent to

$$I \wedge A \wedge O \rightarrow R.$$

A model fragment may be instantiated for each set of individuals that satisfy I; for each such collection, we say that the model is *applicable* to that collection. If furthermore A is assumed, then we say that the model is *applied* to that collection. For example, a model of a string under tension is not applicable to analyzing a steam propulsion plant, while a model of boiling is applicable but need not be applied. Finally, a model can apply to a scenario yet not be imposing constraints. We say a model is *active* when it is both applied and its operating conditions O hold. The equivalent implication

$$I \rightarrow (A \rightarrow (O \rightarrow R))$$

reflects the ordering of these decisions, as well as their relative variability (the individuals are roughly constant throughout an analysis, while the operating conditions can change on a state-by-state basis). Importantly, for each instantiation of I the remainder of the clause is ground and can be treated propositionally. In the implementation, each `defModel` compiles to a

rule which triggers on the conditions stated in I to produce the substitution σ. The body of the rule then checks the ground assumptions $A\sigma$ and if consistent creates a unique token for the model instance (e.g., MI32) and creates the following ground Horn clauses:

$$I\sigma \wedge A\sigma \rightarrow \textit{Model-Application}(\texttt{MI32}),$$
$$\textit{Model-Application}(\texttt{MI32}) \wedge O\sigma \rightarrow \textit{Active}(\texttt{MI32}),$$
$$\text{and for each } r \in R, \quad \textit{Active}(\texttt{MI32}) \rightarrow r\sigma.$$

This instantiation process (called *scenario expansion* in [17]) converts the quantified domain theory into an equivalent propositional theory that is specific to the scenario S.[7] Search for relevant models can then focus directly on the ground query terms and consistency can be determined in the more tractable propositional theory. The bottom half of Fig. 5 shows the dependency structure produced by instantiating the applicable domain theory for the current example (the *Model-Application* and *Active* nodes have been left out to simplify the figure). From this, it is clear that model fragment ***CF*** introduces quantities $L(\text{water},a)$ and $L(\text{water},b)$.

The second issue that arises concerns the form of the algorithm's output and intermediate results. One approach would be to explicitly reason about combinations of model fragments to ascertain which are consistent and sufficient. However, there can be many combinations, involving many irrelevant model fragments. A better alternative is to reason about combinations of modeling assumptions. There tend to be fewer modeling assumptions than model fragments, and so combinations of assumptions provide a more concise representation. Further, checking consistency requires reasoning about the model fragments' requisite assumptions anyway. The model formulation task is then to select a suitable set of modeling assumptions. We call this set of consistent, ground assumptions the *modeling environment* E. The final output of the algorithm is an E that, together with S and Th, derives a minimal, sufficient scenario model.

For the query about quantities $L(\text{water},a)$ and $L(\text{water},b)$, the modeling environment must be at least

$$E = \{\ldots, C(\textit{exists}(a)), C(\textit{fluid-cs}(a)), C(\textit{exists}(b)), C(\textit{fluid-cs}(b)), \ldots\}.$$

This environment may be incomplete in several ways and therefore must be extended.[8] For example, having decided that fluid flow should be modeled, other decisions about the flow must be made. Rule r_3 implies that

[7] Instantiation and caching of quantified formulae is a common mode of interaction with truth maintenance systems [10,32].

[8] This marks an extension of [14], which assumed that each environment represented a coherent scenario model and thus concluded the model composition process at this stage.

any modeling environment which introduces quantity $FR(c)$ also activates assumption class ac_1 and thus must include the assumption $C(inviscid(c))$ or $C(viscous(c))$. Ultimately, an evaluation criterion must be applied to the set of candidate modeling environments to select the one most useful for answering the query.

The conceptual distinctions and associated algorithms of assumption-based truth maintenance systems (ATMS) [9] apply quite naturally to the model formulation task. Our algorithm exploits the ATMS' ability to easily compare and manipulate sets of assumptions. To briefly review: each statement in the problem solver's database has a corresponding ATMS *node*. Since we use a one-to-one mapping between nodes and statements, we sometimes refer to nodes and statements interchangeably. Dependencies between nodes are indicated by *justifications*, which may be viewed here as Horn clauses. A distinguished subset of the nodes are called *assumptions*. A set of assumptions is called an *environment*. A node is said to hold in an environment if the node can be derived from the environment and the set of justifications. The set of all such environments in which a node holds is characterized by the node's *label*. A label is a set of environments, each consistent (in that it does not imply a contradiction) and minimal (in that there is no other environment in the label which is a subset of it). A node holds in a given environment if some environment in the node's label is a subset of the given environment. Label information is propagated through dependencies by an algorithm described in [9].

Our algorithm uses the labels and dependencies maintained by the ATMS to compute the minimal conjunction of modeling assumptions needed to produce an appropriate scenario model. After using the scenario description to fully instantiate the domain theory, model composition consists of four steps:

(1) *Query analysis.* Identify from the contents of the query a set of objects, quantities and relations of interest. Compute the collection of minimal environments which justify the modeling terms in the query. Every coherent modeling environment must have one of these environments as a subset.

(2) *Object expansion.* Each partial environment identifies a set of objects to consider, but additional objects may need to be considered to capture all relevant interactions. This step uses the part-of hierarchy to find the minimal extension of each partial environment such that a coherent set of objects are considered.

(3) *Candidate completion.* Some choices of simplifying assumptions raise new choices in turn. For example, considering liquid flowing through a pipe requires a decision about whether to model the fluid as compressible or incompressible. This step ensures that every partial

environment is extended to include choices for each assumption class it entails.

(4) *Candidate evaluation and selection.* Finally, each candidate modeling environment is evaluated to select the "best" candidate.

The remainder of this section describes each step in more detail. Steps (2) and (3) are performed for each candidate environment and are described from the perspective of a single environment.

3.1. Query analysis

The form and content of queries for different tasks can vary widely. To remain as task-independent and domain-independent as possible, we restrict ourselves to only very basic information. In particular, we assume that whatever the initial form of the query, it can be decomposed by a *query elaboration procedure* into a set of ground expressions $\mathcal{Q}$, each having referents in the fully instantiated domain model. These ground expressions provide the input for this step of model composition.

The simplest query elaboration procedure is to find all quantities and objects mentioned in a query. Consider again the examples in Fig. 3. The question "What affects the efficiency of the plant", would be transformed into

```
(Quantity (efficiency STEAM-PLANT)).
```

Often simple rules suffice to transform a query into the required form. For example, the question "How many mass flows are there?" refers to the set of entities that represent either liquid flow or gas flow. Fetching such entities from the instantiated domain model is trivial. Some aspects of query elaboration will be task-specific. For example, the question "How does the feed pump's throughput change as the pressure gradient ranges from 500 psi to 1500 psi?" suggests the use of quantitative models, not through the individuals involved (since they have qualitative as well as quantitative models) but by the fact that the question concerns a specific duration, and hence requires the fine resolution of quantitative analysis to provide an answer. The important point for query elaboration is that it must identify which aspects of the query are descriptions that must be supplied by the scenario model.

Given our processed query expressions $\mathcal{Q} = \{e_1, \ldots, e_n\}$, we construct partial, candidate modeling environments as follows. We take as input the ATMS dependency network from the fully instantiated domain theory. This network will have a distinct node for each model term that could appear in the query. Furthermore, notice that each term's label contains the minimal sets of modeling assumptions in which it holds. Let `QUERY` be a new ATMS node. Justify `QUERY` by the conjunction of the expressions in $\mathcal{Q}$ and allow

the ATMS to compute its label. Since the only assumptions in the ATMS database are modeling assumptions, every environment in QUERY's label is a partial, candidate modeling environment. That is, since we have included all potential simplifying assumptions in the instantiated domain model, every minimal consistent combination of these assumptions that together entail the objects and properties mentioned in the query will appear as an environment in the label of the node QUERY. Of course, these environments may not by themselves entail a coherent scenario model. The next two steps extend them into coherent candidate scenario models.

As an example, suppose we are interested in the question "How does the furnace's fuel/air ratio affect the amount of steam flowing in the superheater?". This query is transformed into the set of expressions representing the quantities of interest:

$$\mathcal{Q} = \{\texttt{Quantity(amount-of-in(water,gas,superheater))},\ \texttt{Quantity(FA-ratio(furnace))}\}.$$

The dependencies for the corresponding QUERY node are shown in Fig. 6. Inspection of these dependencies produces the single partial candidate environment:

```
{Consider(exists(furnace)),
 Consider(exists(superheater)),
 Consider(fluid-cs(:scenario))}.
```

This environment provides initial guidance by identifying the simplifying assumptions that enable the minimal set of model fragments supporting the query's expressions. Had the query mentioned other aspects of the boiler assembly, these requirements would have been reflected in the environment. For example, also stating interest in the boiler's water level would add consideration of the boiler's geometry as part of the environment. Asking about how the fuel/air ratio also affects the condenser would add consideration of the condenser. Note that this environment does not represent a coherent scenario model. For example, it considers the existence of the furnace and the superheater, yet fails to include the intervening object, the boiler.

3.2. Object expansion

Each initial environment forces consideration of some set of objects. To ensure coherence we must determine if additional objects are required. For example, if we are thinking about steam flow in the boiler and turbine, then we need to think about the condenser and feed pump, too, so that we will correctly recognize that these flows are part of a closed cycle. On the other hand, if we are only considering possible faults in the furnace, the rest of the steam plant can be ignored. These decisions are governed by the constraints

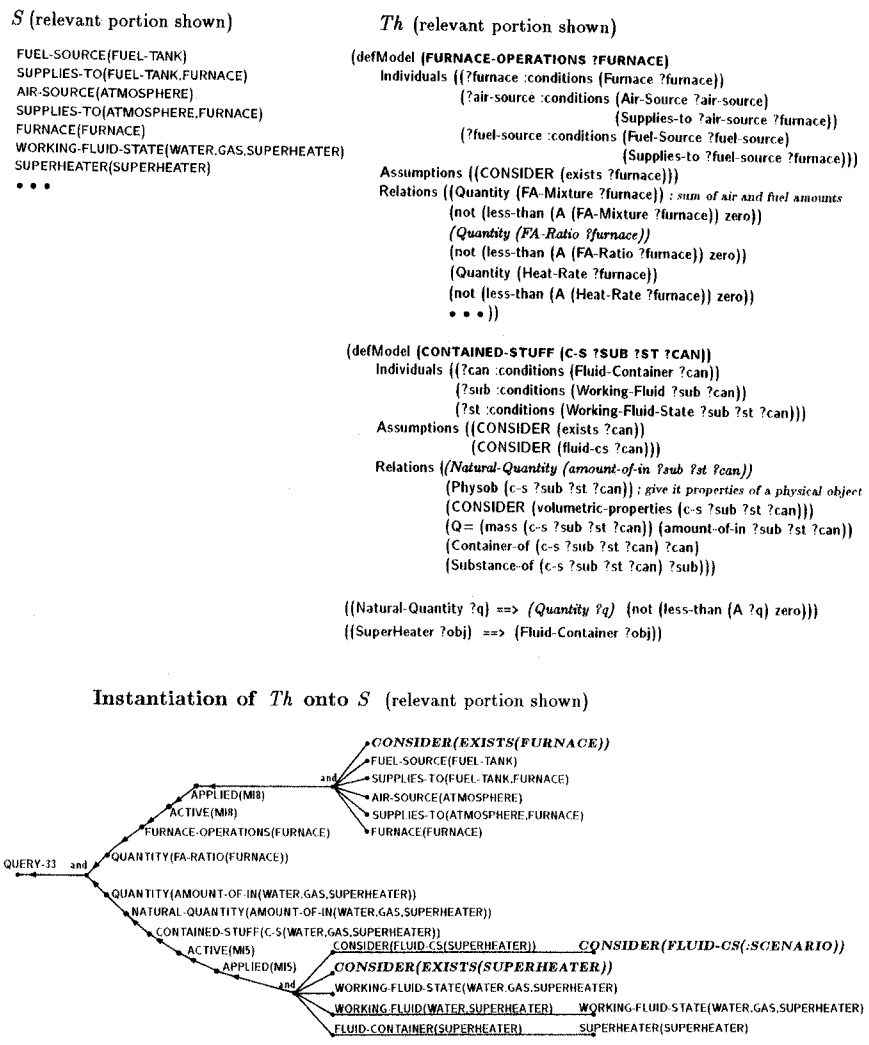


Fig. 6. TMS dependency structure showing the relationship between terms in the models and the modeling assumptions that enable their use. QUERY-33 represents the conjunction of the query expressions `Quantity(amount-of-in(water,gas,superheater))` and `Quantity(FA-ratio(furnace))`.

imposed on grain assumptions. The intuition is that one cannot arbitrarily choose a set of parts to model in isolation. Instead, given interest in some initial set of parts, one has to consider enough of the system so that all of the relationships involving the initial set are included in the model.

In the simplest case, if we consider two objects that are part of the same system, then all of that system's components must be considered:

CONSIDER(exists(s_1)) ∧ CONSIDER(exists(s_2)) ∧ $s_1 \neq s_2$ ∧
Part-of(s_1,s_0) ∧ Part-of(s_2,s_0)
→ CONSIDER(components(s_0)),

CONSIDER(components(s_0)) ∧ Part-of(s_i,s_0)
→ CONSIDER(exists(s_i)).

When objects being considered are part of different systems, a more sophisticated set of constraints are required. We define a *covering system* to be a system that contains all systems of interest. Many covering systems are too large to be useful; :scenario, for example, is always a covering system. Therefore we define a *minimal covering system* to be a covering system that contains no smaller covering system. Specifically, a minimal covering system is the lowest common ancestor in the part-of hierarchy of the systems being considered. Because we assume a strict hierarchy, the minimal covering system will be unique. Given an initial set of grain assumptions, their minimal covering system (MCS) determines what other systems must exist by the following rule:

MCS(s_c) ∧ CONSIDER(exists(s_1)) ∧
Sys-Contains(s_c,s_i) ∧ Sys-Contains(s_i,s_1)
→ CONSIDER(exists(s_i)) ∧ CONSIDER(components(s_i))

If the partial modeling environment only requires a single object, then the minimal covering system will be that object and nothing needs to be added. Otherwise, if s_c is the minimal covering system, then a path of existence assumptions is extended across the systems leading from s_c to the objects mentioned in the initial environment. The modeling environment is then augmented with these new existence assumptions.

Resuming our example, recall that the initial environment explicitly stated a need to consider the furnace and superheater. To identify the complete set of objects required to form a coherent scenario model from this environment, we first determine its minimal covering system. For the furnace and superheater systems, this is the boiler-assembly system. From the grain assumption constraints, the following grain assumptions are then required (Fig. 7):

{Consider(exists(furnace)),
Consider(exists(superheater)),
Consider(exists(boiler))}

The net effect is that the boiler has been added to the model. Note that the steam plant's other primary components, such as the turbine and condenser assemblies, are ignored for this query. Further, additional detail within the selected systems is ignored, such as the furnace's fuel pump and exhaust manifold.

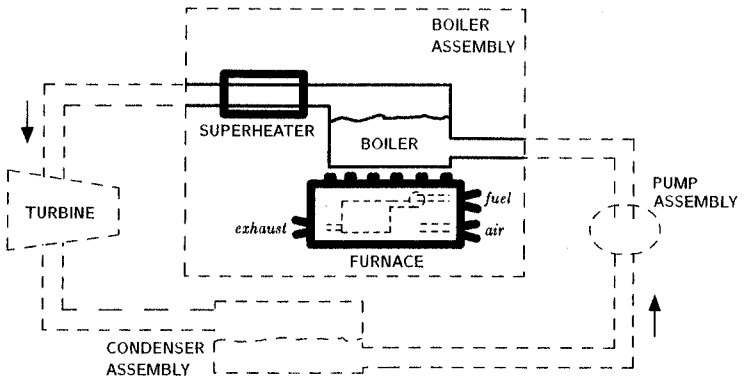


Fig. 7. Extending a partial modeling environment to consider a coherent collection of objects. A partial environment comprised of the furnace and superheater systems must also minimally include the boiler system. Note that the steam plant's other components, as well as the internal structure of the furnace, need not be considered.

3.3. Candidate completion

At this stage the partial modeling environments have grown to include every object that must be included in a coherent domain model. However, this does not necessarily mean that we have chosen exactly how each object and associated phenomena should be modeled. For example, if the query concerned the difference in water pressure between the boiler and the condenser, then the initial environment would consider those two objects from the *fluid-cs* perspective and the object expansion step would extend it to include the feed pump, turbine, and the paths which connect them. But how should the paths be modeled? Should their resistance be taken into account or not? How should flows through the paths be modeled? How about the pump?

The extra information needed to both determine what further kinds of modeling assumptions are required and what the alternatives are is provided by assumption classes (Section 2.3.3). Recall that an assumption class is considered *active* when its activity condition holds (i.e., follows from E, S, Th), and when active, the problem solver must include one of its members in a scenario model. Thus, we extend each partial environment to include one assumption from each assumption class whose activity it entails. This is called the *candidate completion* procedure.

The candidate completion procedure is slightly tricky. For example, choices made for one assumption class can lead to choices for others becoming inconsistent. Obviously, some choices are not constrained, leading to branching and thus a growing number of candidate modeling environ-

```
(defAssumption-class (geometry ?obj)
     ((not (CONSIDER (geometric-properties ?obj)))
      (CONSIDER (geometric-properties ?obj))))

((Fluid-container ?can)  (CONSIDER (exists ?can))  ==>  (Geometry ?can))
```

Fig. 8. Rules defining the geometry assumption class.

ments. Choosing to extend an environment by one assumption may expand its entailments to include yet more assumption classes, which in turn must be searched. This can be cast as a *dynamic constraint satisfaction problem* [33] by viewing class conditions and their associated assumptions sets as active variables and their associated choice sets. We use the ATMS-based algorithm described in [33] to solve it.

The result of this procedure is a set of complete candidate modeling environments. The scenario models represented by these environments are coherent, since they consider all the objects necessary to reason about the queried objects and include a choice for all relevant modeling decisions.

In our continuing example, the developing modeling environment is currently comprised of:

```
{Consider(exists(furnace)),
 Consider(exists(superheater)),
 Consider(exists(boiler)),
 Consider(fluid-cs(:scenario))}
```

This environment activates nine assumption class instances, corresponding to three general assumption class types. The first assumption class, defined in Fig. 8, requires a decision about whether or not to model the geometric properties of each fluid container whose existence is considered. Because there are three "fluid containers" (furnace, boiler, superheater), there are three `geometric-properties` assumption class instances, each conditioned on `Geometry(`*can*`)`. For example, the condition that the boiler is a fluid container and its existence is being considered entails `Geometry(boiler)`, which is the condition that activates the choice set of whether or not to consider the boiler's geometric properties. The second type of assumption class activated concerns the thermal properties of each object whose existence is considered. Again, there are three instances of this class, one for each of the three objects under consideration. Finally, there are three applied fluid flow instances: the liquid input to the boiler, gas from the boiler to the superheater, and gas output of the superheater. Each activates a fluid-viscosity assumption class (defined in Section 2.3.3) applied to the flow

instance (e.g., `MI32`), forcing the decision to model each flow instance as either inviscid or viscous.

There are nine binary assumption classes active for the current, partial modeling environment, leading to 512 consistent extensions for this environment (these assumption classes happen to not interact). In the next section, we describe our metric for evaluating the relative desirability of each choice. This metric is consulted by the candidate completion procedure so that all possibilities need not be generated.

3.4. Candidate evaluation and selection

Given a set of candidate modeling environments, the final step is to apply an evaluation metric to the candidates and select the best one to use in answering the query. What is best often depends on the details of the domain and task—for example, the computational support available to use the model often determines whether one form is better than another. We allow user-supplied evaluation metrics to enable future study of alternate approaches. However, we describe here a simple scheme that is based on crude estimates of model costs and has worked surprisingly well for the examples we have examined.

First, the set of candidates is pruned by retaining only those with the smallest number of objects. For example, if two candidates entail four objects and three candidates entail five objects, those considering five objects would be ignored.

Second, the ordering information in assumption classes is used to estimate overall simplicity. Recall that the choices in each assumption class are ordered in increasing complexity. Consequently, we assign a number to each choice corresponding to its place in its ordering. For each candidate these values are summed, and the candidate whose sum is the smallest is selected. In case of a tie, one of the minimal environments is selected at random.

As an example, assume a domain model containing active assumption classes $A = \{A_1, A_2, A_3\}$ and $B = \{B_1, B_2\}$. If all combinations are consistent, the following candidates are possible:

$\{A_1, B_1\}$		(score = 2)
$\{A_1, B_1\}$		(score = 2)
$\{A_1, B_2\}$	$\{A_2, B_1\}$	(score = 3)
$\{A_2, B_2\}$	$\{A_3, B_1\}$	(score = 4)
$\{A_3, B_2\}$		(score = 5)

Thus, $\{A_1, B_1\}$ would be selected.

The limitations of this scheme are obvious. For example, should $\{A_2, B_2\}$ and $\{A_3, B_1\}$ be considered equivalent? Probably not. But the information

required to distinguish between them (for instance, by ascertaining the relative expense of A_3 and B_2) depends on the specifics of the task and domain, and hence we must defer attempting to provide more guidance here.

Returning to our running example, the following modeling environment is selected:[9]

```
{Consider(exists(superheater)),
 Consider(exists(furnace)),
 Consider(exists(boiler)),
 Consider(fluid-cs(:scenario)),
 Consider(inviscid(boiler-sh)),
 Consider(inviscid(sh-env)),
 Consider(inviscid(env-boiler)),
 ¬Consider(thermal-properties(furnace)),
 ¬Consider(thermal-properties(boiler)),
 ¬Consider(thermal-properties(superheater)),
 ¬Consider(geometric-properties(furnace)),
 ¬Consider(geometric-properties(boiler)),
 ¬Consider(geometric-properties(superheater))}
```

This modeling environment produces the scenario model summarized in Fig. 9. It describes the fluids and processes associated with the three components being considered, as well as their interactions, in isolation from the rest of the steam plant. Although the domain theory contains many other potentially applicable model fragments, the model composition algorithm has allowed us to construct a model that focuses only on relevant aspects.

3.5. *Model use and validation*

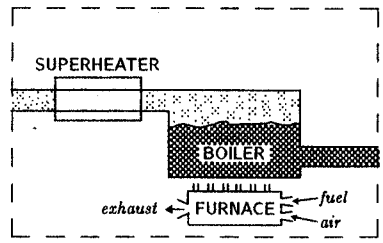
Prior to analysis, the information available about the scenario is incomplete. As a result, the model composition procedure cannot guarantee that the modeling assumptions it selects are valid for the scenario being studied. Here we describe how analysis using a scenario model can detect inappropriate assumptions, thus leading to another, more informed round of model composition.

Simulation, both qualitative and quantitative, has been the principle use of the models we develop. Qualitative simulations are carried out using QPE [17], an envisioner for QP theory. Quantitative simulations are carried out using a fourth-order Runge–Kutta integration algorithm with adaptive step-size control (from [37, Chapter 15]). The equations for the simulator

[9]Throughout this example, env refers to the scenario's physical environment exterior to the minimal covering system.

Contained-Stuff(c-s(water,liquid,boiler))
Contained-Stuff(c-s(water,gas,boiler))
Contained-Stuff(c-s(water,gas,superheater))
Aspatial-contained-liquid(c-s(water,liquid,boiler))
(Qprop *pressure(bottom(boiler))* *volume(c-s(water,liquid,boiler))*)
Non-thermal-gas(c-s(water,gas,boiler))
Non-thermal-gas(c-s(water,gas,superheater)) (Qprop *pressure* *mass*)

Fluid-flow(env-boiler,water,liquid,env,boiler)
Fluid-flow(boiler-sh,water,gas,boiler,superheater)
Fluid-flow(sh-env,water,gas,superheater,env)
Inviscid(env-boiler)
Inviscid(boiler-sh) (Qprop *flow-rate* Δ *pressure*)
Inviscid(sh-env)
Boiling(boiler,water)
Superheater-process(superheater)



Furnace-operations(furnace) (*heat-rate, FA-mixture, FA-ratio, FA-peak-efficiency...*)
Furnace-boiler-connected(boiler,furnace) (*heat-rate(boiler) = heat-rate(furnace)*)
Total-FA-intake(furnace) *total-intake = fuel-intake + air-intake*
FA-ratio(furnace) = fuel-intake / total-intake
etc
Combustion(furnace)
Furnace-running-lean(furnace) (Qprop *heat-rate* *FA-ratio*)
Furnace-running-rich(furnace) (Qprop- *heat-rate* *FA-ratio*)

Fig. 9. Summary of the scenario model composed in response to the query "How does the furnace's fuel/air ratio affect the amount of steam flowing in the superheater?".

are gathered from the scenario model, with some minor processing to get them into the form it expects. Currently we restrict quantitative analysis to concern a single operating region—that is, all equations hold all of the time. This restriction will soon be removed when we add these model composition techniques to SIMGEN [19].

Internal consistency tests have long been used to check the validity of assumptions made during the course of problem solving. How we use them to verify a simulation model depends on whether it is qualitative or quantitative. We consider each in turn.

Given the weak nature of qualitative information, how does one detect an internally inconsistent model? The answer is simple: If the envisionment is empty, then the model is inconsistent because it states that no behavior is possible. For example, imagine a string with one end attached to the ceiling and the other tied to a block. Suppose the block is initially held close to the ceiling, with the string hanging loose, and then released. What happens when the string becomes taut? If the string is assumed inelastic,

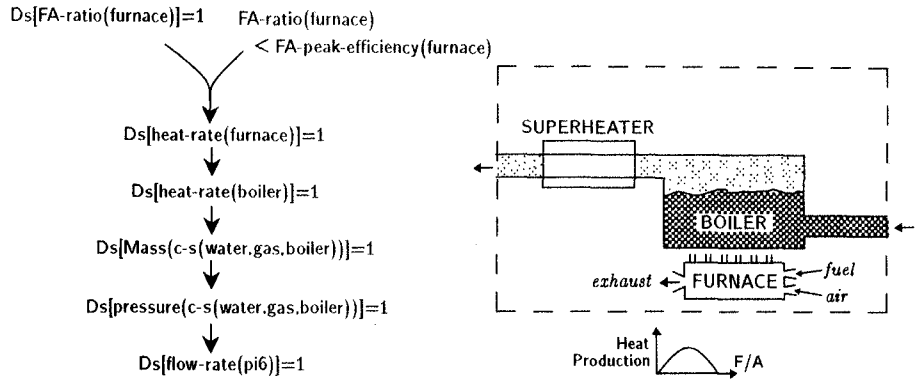


Fig. 10. How does the furnace's fuel/air ratio affect the amount of steam flowing in the superheater?

the block will stop, instantaneously, without any deceleration. This violates continuity, and hence this transition is ruled out. But since the falling block must arrive at the end of its reach (since asymptotic approach is ruled out in QP theory), the entire modeling environment is ruled inconsistent. If, on the other hand, the string is assumed elastic, the block will decelerate over some interval of time and come to rest.

We validate a quantitative model by checking to see if the behavior it predicts violates its assumptions. To do this, we gather the set of *critical inequalities* that represent the operating assumptions required by the model's simplifying assumptions. For example, the incompressible flow assumption requires that the flow's Mach number (ratio of velocity to speed of sound) is less than 0.3. If at any time during a simulation using the incompressible flow assumption its Mach number exceeds 0.3, the modeling environment is deemed inconsistent.

When an inconsistency is discovered, the process is repeated with new, very specific information about inconsistent modeling assumptions (cf. dependency-directed backtracking [46]).

Returning to the scenario model derived in the previous section (Fig. 9), it produces a qualitative envisionment describing the various ways in which the furnace's fuel/air ratio can affect the amount of steam flowing in the superheater.[10] The furnace's heat production is a function of its fuel/air

[10]The query for this example included operating assumptions which constrained this envisionment. These were: the amount of fluid in each container is greater than zero, `pressure(fluid,env) > pressure(fluid,boiler)` (enabling flow), `pressure(steam,boiler) > pressure(steam,superheater)`, `pressure(steam,superheater) > pressure(steam,env)`, and `heat-rate(furnace) > 0`.

ratio—too much air (low F/A) or too much fuel (high F/A) lowers the combustion efficiency. An ideal fuel/air ratio corresponds to a point of peak heat production. For the case in which the furnace has been operating in a suboptimal, low F/A region, Fig. 10 shows the resulting perturbation when the fuel/air ratio is increased. Briefly, an increase in F/A results in increased heat production, which results in increased steam production and increased boiler steam pressure, leading to an increased flow of steam through the superheater. Because no model inconsistencies are detected for this envisionment, the query has been answered and model formulation stops.

4. Tutorial question answering about a steam plant

One of our motivations for building comprehensive, multi-perspective models is to support the construction of intelligent tutoring systems [45,53]. Consequently, we focus on the problem of finding an appropriate model to answer questions in an instructional setting. Our ultimate goal is to develop a generic tutoring system which, given a domain theory and appropriate user interface, could produce reasonable explanations (in the manner of [20]). This task is an excellent one for testing the model composition algorithm, since it would be unreasonable to expect students to know the internals of the domain model or what simplifying assumptions are reasonable. However, we are far from a generic tutoring toolkit. Since we have focused on developing organizing principles for domain models and a method for model composition, everything else is very rudimentary. The natural language explanations in the sections which follow are indeed generated automatically. Queries are expressed as a conjunction of modeling terms or via procedures tuned to specific types of questions that in turn compute a conjunction of modeling terms. For all qualitative queries, we make extensive use of operating assumptions to avoid the expense of unconstrained envisioning. Even with these limitations, we have been able to efficiently use domain theories that are an order of magnitude larger than our previous ones.

As our test domain, we have developed a multi-grain, multi-perspective domain theory for a subset of engineering thermodynamics and use it to answer questions about a shipboard steam-powered propulsion plant. The principles described in this paper have been crucial to managing the model's complexity. At present, our domain model is predominately qualitative, although its quantitative aspects are being extended. It represents three primary areas of coverage (see Fig. 11). One set of elementary models treat the steam plant as a closed, thermodynamic cycle. In this energy flow ontology, the internals of the propulsion cycle need not be considered; heat

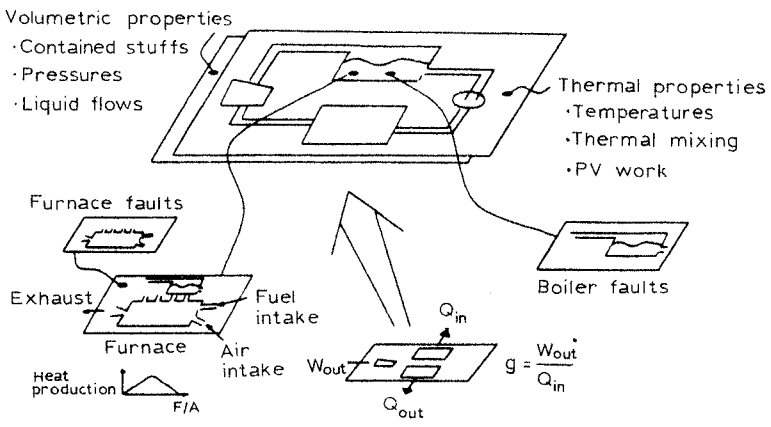


Fig. 11. Differing views of the propulsion plant.

flows into the system and work flows out. It is useful for describing global properties of a thermodynamic system, such as efficiency.

A second set of elementary models focuses more specifically on the boiler assembly and its parts (i.e., the furnace, boiler, and superheater). They represent the furnace explicitly, including the effects of fuel/air ratio on heat production rate and efficiency. Further, they include fault models for the boiler and furnace, such as enabling consideration of the boiler becoming too full or too empty. In operating a plant it is important to keep the water level within a certain range. Too low, and the boiler can melt. Too high, and water droplets are entrained into the superheater. Since steam is moving through the superheater faster than sound, these water droplets can cause tremendous damage.

The largest set of models describe various aspects of the mechanics and thermodynamics of fluids using the contained stuff ontology. These include fluid paths, fluid and heat flows, pumps, containers, and portals.

A major claim of compositional modeling is that it facilitates creating larger domain models. To support this claim requires comparing this model to previously developed models, which can be difficult. One way is to reduce model fragment's to something simpler, such as Horn clauses. If we treat Horn clauses as "axiom-equivalents", then at last measure our domain model corresponded to 1637 axioms. By contrast, our earlier QP models of thermodynamics were around 300 axioms.[11]

[11] Few domain models have been published to date. The best known is Hayes' naive theory of liquids [24]. This theory consisted of 74 axioms, but these were more complex logical forms. Assuming a (generous) multiplier of 3 to translate into Horn clauses, his model of liquids

A good sense of how well the model composition algorithm focuses analysis can be gained by comparing the size of the fully instantiated steam plant model to the size of model created for each query. In fully instantiating the steam plant model 83 model fragments are used. These introduce 89 quantities. As the examples below show, our algorithm provides significant filtering.

4.1. Qualitative questions

Our domain model can be used to answer a variety of questions based on qualitative models of the plant.

4.1.1. Plant efficiency

Q: What affects the efficiency of the plant?
A: `The efficiency of the plant is affected positively by the work rate of the turbine. It is also affected negatively by the energy input to the plant.`
8 quantities, 21 inequalities, 7 model fragments, and 3 situations.

Questions about what affects a parameter are viewed as a request for a qualitative model which includes the qualitative proportionalities that constrain the named parameter. The query given to the model composition algorithm is simply the parameter itself, e.g.,

```
(Quantity (efficiency steam-plant))
```

Thus what one would expect is the simplest model that includes the efficiency of the plant as an explicit parameter.

Plant efficiency is introduced by the closed, thermodynamic cycle model fragment. It is conditioned on two assumptions

```
(CONSIDER (exists ?plant))
```

and

```
(CONSIDER (energy-flow ?plant)).
```

The modeling environment generated by the model composition algorithm is simply these two assumptions applied to the `steam-plant`—they alone produce a coherent scenario model for this query. The closed thermodynamic cycle model in turn forces consideration of three of the steam plant's main components (boiler heat source, compressor heat sink, and turbine work sink), thus creating their respective heat and work rates. This example shows the importance of system boundaries—potentially, almost everything

consisted of roughly 230 axioms. Other models, such as Molle's work on chemical processes [34] appear to be roughly the same size as Hayes' model and our previous models.

in the plant affects efficiency. Yet clearly that is not the level of detail which is appropriate, given a high-level query.

4.1.2. *Finding mass flows*

Q: How many mass flows are there?
A: `There are 4 mass flows:`
`(1) A flow of water from the condenser to the feed pump.`
`(2) A flow of steam from the turbine to the condenser.`
`(3) A flow of steam from the boiler to the turbine.`
`(4) A flow of water from the feed pump to the boiler.`
25 quantities, 83 inequalities, 27 model fragments, and 15 situations.

Questions about the occurrences of processes, whether about a specific process instance or about a set of process instances, are transformed for model composition into the set of terms that name processes satisfying the input pattern. Here, the union of `Liquid-Flow` and `Gas-Flow` processes was sought.

The modeling environment for this query considers the existence of the steam plant's four main components (i.e., boiler assembly, turbine assembly, condenser assembly, and pump assembly) and adopting the contained stuff ontology. All geometry and thermal information was ignored, and all flows were assumed to be inviscid. These are exactly the choices which minimize the amount of extraneous detail, demonstrating that our system can ignore perspectives which do not contribute to answering a given query.

4.1.3. *The smoking furnace*

Q: What is causing black smoke to rise from the furnace?
A: `Black smoke is rising from the furnace because it is not the case that the fuel/air ratio of the furnace is less than the F/A saturation point for the furnace.`
24 quantities, 65 inequalities, 23 model fragments, and 21 situations.

The domain model includes black smoke as an individual view describing one of the possible modes a furnace can be in, due to a fuel/air ratio that is too rich. This is an interesting question for two reasons. First, black smoke means the ship is burning fuel less efficiently than it should, which is bad economically. Second, pouring out black smoke is not a good thing to do if you wish not to be seen. Questions about what is required for an instance of an individual view to hold are treated as questions about instances of processes. That is, the query consists of the term (or terms) designating the view instance.

The modeling environment for this query considers the existence of the furnace in isolation from the rest of the system, using the contained stuff

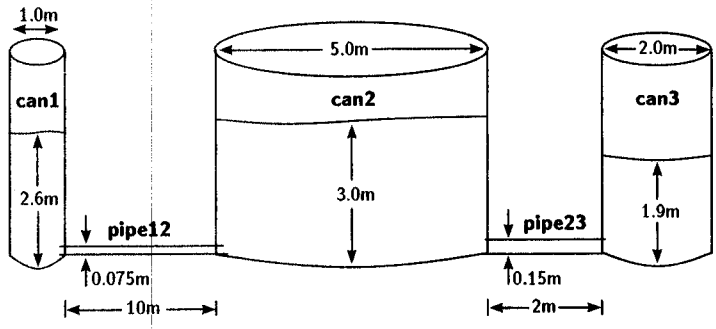


Fig. 12. Two oil supply drums connected to a central reservoir. Find the level of the three containers as a function of time when the system is released from the given initial conditions.

ontology. Since the view in question is in fact a fault, the algorithm included the additional assumption

```
Consider(fault(exhaust-type,furnace)).
```

Normally such fault assumptions would not be included, since they lead to extra complexity in a model. But this example shows that fault assumptions can be included automatically, as necessary, if the domain model is organized properly.

4.2. A quantitative question

Here we describe a quantitative analysis problem concerning a hypothetical set of lubrication tanks and pipes. We are given two oil supply drums connected to a central reservoir, as shown in Fig. 12. The task is to determine the behavior of the oil level in `can1` when the system is released from the initial condition. Our current fluid flow models represent various simplifications of the unsteady Bernoulli equation, which describes incompressible flow along a streamline: [12]

$$\frac{p_1}{\rho} + \frac{\overline{V}_1^2}{2} + gz_1 = \frac{p_2}{\rho} + \frac{\overline{V}_2^2}{2} + gz_2 + h_l + \int_1^2 \frac{\partial V_s}{\partial t}\,\mathrm{d}s$$

where ρ is the density of the fluid, z_i is the height at point i, V_i is the fluid velocity at point i, and h_l is the head loss due to frictional effects

$$h_l = f\frac{L}{D}\frac{\overline{V}^2}{2}.$$

[12]Currently, we use the unsteady Bernoulli equation, which subsumes the steady and unsteady flow cases.

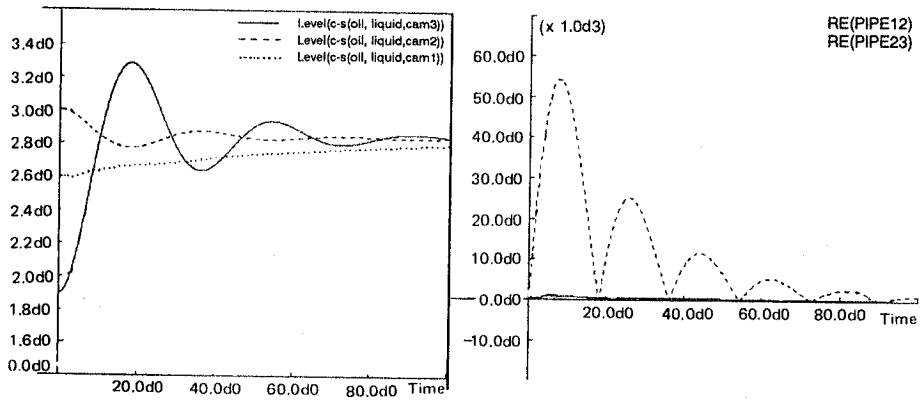


Fig. 13. Behavior predicted by the laminar flow model.

The computation for the friction factor f is dependent on the flow regime (laminar or turbulent), which is normally determined by Reynolds' number ($Re = \rho \overline{V} D/\mu$). For low Reynolds' numbers (low flow rates), the flow is laminar; for high Reynolds' numbers (high flow rates), the flow is turbulent. Although the transition occurs over an interval, flow in pipes is generally taken to be laminar for $Re<2300$.

Based on the quantitative fluid flow model's dependencies, the initial modeling environment for this problem is

```
{Consider(fluid-cs(:scenario)),
 Consider(exists(can1)),
 Consider(geometric-properties(can1))}.
```

The object expansion step extends this to consider can2 and can3. The candidate completion procedure is then left with several final decisions. Given only the initial conditions, there is no information about the flow velocity or Reynolds' number for either pipe. Thus, it selects the simplest options:

```
{Consider(laminar(pipe12)),
 Consider(laminar(pipe23)),
 Consider(incompressible-flow(pipe12)),
 Consider(incompressible-flow(pipe23)).
```

For laminar flow, the friction factor is inversely proportional to Reynolds' number ($f = 64/Re$). The behavior predicted by this model is shown in Fig. 13. The left plot shows the level of each container as a function of time, while the right plot shows the Reynolds number for each flow. At this point, our system inspects the predicted behavior and finds one

modeling environment violation: the Reynolds number for `pipe23` reached 54,000, thus violating the laminar flow assumption for `pipe23`. The model composition procedure is repeated with this added information, producing a new set of flow regime assumptions:

```
{Consider(laminar(pipe12)),
 Consider(turbulent(pipe23))}.
```

This modeling environment models the flow through `pipe12` as laminar and the flow through `pipe23` as turbulent (we use the Moody approximation [22, p. 187] to compute the friction factor for turbulent flow). The new model predicts a lower amplitude oscillation for the flow through `pipe23`, due to the greater dissipative effects of turbulent flow.

Note that because the Reynolds number for `pipe12` never exceeds 1100, the laminar flow assumption for `pipe12` remains consistent. This demonstrates an important attribute of the compositional modeling approach. By representing modeling assumptions as predications over individual objects and phenomena, a scenario model is able to represent the same type of object or phenomenon in different ways, depending on their individual conditions. This ability is crucial for analyzing large systems.

5. Related work

Research on automatically formulating a model in response to a query is relatively new. However, a number of methods have been developed for managing and switching between multiple models. Much of our inspiration comes from Sussman's *slices* notion [47,48], where results from multiple perspectives could be combined in synthesizing engineered systems. In Sussman's system the language for specifying perspectives was domain-dependent (i.e., electronic circuits), and instantiation and use decisions were made by hand. Our techniques aim to automate this decision-making process and should work for any phenomena expressible in our modeling language.

The work closest to our own is the "Graph of Models" (GoM) effort by Addanki, Cremonini and Penberthy [1,2]. In the GoM approach, the space of possible scenario models is represented explicitly as a graph. Each node represents a model of the system being analyzed, and each edge indicates which approximations differ between the models it connects. Given a space of possible models (the GoM) and an observation, the graph of models task is to find the model whose predictions are sufficiently close to the observation, where sufficiently close is evaluated by the user. Search begins with the simplest model, moves to a new model when prediction fails to match observation, and is guided by rules stating each approximation's qualitative effect on the model's predicted behavior.

In some ways, GoM addresses a more restricted version of our model formulation problem. First, the models differ only in the approximations made and other modeling dimensions, such as perspective, granularity, etc., are currently not considered. Second, no query is provided. Thus, the graph of available models is assumed a priori to be both coherent and relevant. Third, approximations are global, rather than applying differently to different subsystems in the model. On the other hand, GoM introduces a general-purpose technique for selecting approximations that does not rest on our presupposition of an explicit set of constraints that delimit when each approximation is valid.

Given domain models of equal detail, GoM incurs an exponential increase in storage over compositional modeling because it must explicitly store all consistent possible combinations of modeling assumptions. This also places a heavier burden on the domain model developer, who must identify consistent combinations of modeling assumptions in advance of reasoning. By contrast, our model composition algorithm constructs such combinations as needed, according to task demands. The builder of a compositional domain model can more easily work on fragments of the model independently. We believe these differences are advantages for compositional modeling. However, the GoM approach makes the important insight that efficiency can be improved by explicitly noting well-understood combinations of domains and tasks. This may be done for compositional models as well, since the grain size of model fragments is up to the domain modeler.

One very interesting aspect of GoM is the ability to reason about how changing approximations qualitatively affects model/observation discrepancies. Weld [51] presents an elegant extension to this for a specialized class of approximation assumptions. Because Weld's approach is domain-independent and does not depend on the presence of an explicitly predefined graph, we believe it could easily be adapted for compositional models. In particular, the set of modeling assumptions that specify a scenario model could be perturbed individually, generating "adjacent" alternatives by repeating our candidate completion procedure.

Davis' system for model-based diagnosis of digital circuits used multiple levels of structural descriptions to control search [8]. Specifically, he used the ability to rule out candidates at an abstract level to reduce an otherwise unmanageable search space, and then moving to more detailed structural levels to make a final diagnosis. In many respects our use of grain assumptions and system distinctions is similar. However, we are able to focus in on individual objects in the system more easily, and do not require a fixed hierarchy of finer-grained models.

The general issue of generating, specifying, and using theories having different abstractions, approximations, and perspectives is of central concern in many areas of research. Some efforts have focused on specifying and making

use of the relationship between a given set of models. For example, different models can be used in a cooperative fashion, such as moving between microscopic and macroscopic ontologies [3,6,31,39] or changing time-scales [23,29]. Fishwick [15] presents a classification of process abstraction methods and associated techniques for mapping from one abstraction level to another. A growing area of interest is the automatic reformulation of a given theory into a (sometimes equivalent) simpler theory via techniques like aggregation, compilation, and re-representation [4,12,49]. Techniques specifically aimed at analyzing physical systems include aggregation of discrete, cyclic events into continuous processes [50], aggregation of variables representing dynamic systems [26], and generating models of differing time-scales [25].

6. Discussion

Capturing the expertise of human engineers will require developing large-scale, multi-grain, multi-perspective models of physical domains and techniques which allow them to be flexibly and efficiently applied to a broad range of tasks. As we attempt to use computers to automate more engineering tasks, problems of effective model organization and task-relevant formulation become paramount. We believe the compositional modeling strategy described in this paper is an important step towards understanding how to build and use the kinds of domain models sought.

By developing compositional domain models, model fragments can be used in many distinct scenarios. Our techniques are domain-independent, only requiring a modeling language rich enough to express a model's dependence on structural setting, modeling assumptions, and operating conditions. Importantly, we showed how much of the burden of developing a model for a specific task could be carried by a simple, ATMS-based algorithm. By predicating pieces of a domain model on explicit modeling assumptions, a relevant model for a given task can be generated automatically, assuming only that one has enough information about what kinds of entities and properties must appear in the desired analysis.

We believe the insulation of the analyst from the details of the domain model our model composition algorithm provides is very important. For tutoring tasks, it is obviously a necessity: If a student's knowledge of the domain were sufficient to select the appropriate simplifying assumptions, there would be little need for the tutor. But we believe even expert engineers will benefit from allowing the model composition algorithm to share the burden of finding the right foundation for an analysis. First, the engineer's task is simplified if she can specify just enough to make her intent clear, and leave the rest to a (mechanical) assistant. Second, in practice, all too often

one finds models where the underlying assumptions made in one part of a model conflict with those made by another part. The automatic selection of a complete set of explicit simplifying assumptions can help ensure that they are used consistently across an analysis.

6.1. Additional research issues

We view this work as an important first step towards understanding and solving the model formulation problem. Progress has required making some strong simplifications and ignoring some important aspects of the problem. In this section, we consider a number of open problems in the current compositional modeling framework.

6.1.1. Decomposing the scenario

Probably the most limiting simplification is the presupposition that any decomposition of the scenario must be taken from a single, strict part-of hierarchy. For many systems this limitation is not severe—for example, part-sharing only becomes evident at low levels of detail in structural descriptions of power plants and chemical plants. However, as the depth of detail in structural descriptions increases, and as domain models include richer functional vocabularies, the problem of choosing the appropriate reference frame will become acute.

One aspect of this problem for models of thermodynamics is the ability to automatically form control volumes (see [44] for some initial work in this area). In dynamics, the concept of *p-component* from QP theory provides a definition of isolation which could potentially help in constructing reasonable system boundaries. In general, determining appropriate system boundaries will require more sophisticated representations of geometry than found in current domain models.

An intermediate step would be to extend the current approach of taking the set of possible groupings as input by allowing multiple, tangled hierarchies. One consequence of moving to a tangled hierarchy is that there need not be a unique minimal covering system. It is possible that our existing model composition algorithm could be extended to tangled hierarchies. This would require (a) a technique for mapping from a query to the types of clusterings needed to answer it and (b) constraints between the paths in the hierarchy to ensure a coherent view of the scenario.

6.1.2. Improving query analysis

Our current query analysis technique is based on the recognition that the terms in the query provide significant constraint in identifying an appropriate set of modeling assumptions and associated model fragments. It is essentially a syntactic approach.

An important next step is the ability to reason about the information requirements of the query and the information provided by the models. Does the model (which syntactically has all the right terms) provide the behavioral distinctions needed to answer the query? Is it possible to determine this *prior* to actually using the model? In the general case, the answer is probably no (e.g., it would be similar to solving all planning problems without search). However, solutions for some restricted classes are probably possible. For example, consider analyzing a spring-block oscillator and asking the question "Will it stop?". Clearly, a model which ignores dissipative effects would not be sufficient.

6.1.3. Estimating cost

Optimally answering a given query requires more sophisticated measures of utility. One aspect is capturing the accuracy requirements of the query and the information loss introduced by simplifying assumptions (discussed partially in the next section). Another aspect is evaluating the computational (and perhaps cognitive) cost of a model. The ordering scheme described in Section 3.4 makes the simplistic assumption that for all assumption classes A and B, the cost of A_1 is the same order of magnitude as B_1, A_2 is the same order of magnitude as B_2, etc. In general, different approximations can have widely different computational consequences (e.g., dropping a term in a polynomial versus enabling a closed-form solution). We do not know of any general mechanism for estimating the cost of a given equational model.

6.1.4. Reasoning about approximations

Approximations introduce error. For some well-understood approximations, their region of validity can be delimited via explicit constraints on parameter values (e.g., Reynolds' number, Biot modulus, Mach number, etc.). This has two limitations. First, such constraints are not known for many cases. Second, these constraints are rules of thumb based on "typical case" task requirements (commonly $\pm 5\%$).

Addanki et al. [1] and Weld [51] present general techniques for selecting suitable approximations via reasoning about qualitative $\{-, 0, +\}$ prediction/observation differences. The utility of focusing on qualitative differences is unclear, particularly given the potential search cost. Falkenhainer and Shirley [42] are currently investigating an alternate approach to this problem. Our central intuitions are that (1) the accuracy requirements of the task must be made explicit and (2) approximation selection is a function of the approximation's cost savings and accuracy loss. This too can be more expensive than simply not using approximations. We are seeking ways to generalize successful problem-solving episodes such that the system's adroitness in using approximations can improve with experience.

In settings where little is known about a model's accuracy or range of applicability, experimental evaluation is required and model formulation blends into theory formation and revision. Methods for repairing a model that diverges from observation are described in [1,13,38,51]. This is an important aspect of the modeling process, and we would ultimately like to integrate such empirical techniques into the compositional modeling framework.

6.1.5. Focused and lazy instantiation

Our current implementation fully instantiates the set of applicable model fragments once for each scenario and then consults the ground dependencies for each query about the scenario. For a large scenario and diverse domain theory, this can result in a huge dependency structure. An alternate approach would be to use the query to guide instantiation and instantiate as little of the domain theory as possible. Additional constraints that we have not yet explored include:

- Determining the ontology and objects required directly from the query and then instantiating only those elements that are consistent with the resulting partial modeling environment. In this manner, a question about control volumes over a selected set of objects can avoid instantiation of Lagrangian views over the entire scenario.
- Enforcing additional control over the rule-instantiation mechanism, such as triggering rules only when their antecedents hold rather than when they are merely present in the database [18].
- Stratifying the rules into self-contained clusters of relevance [35].

6.1.6. Optimizing model composition

Human tutors rarely seem to resort to first-principles reasoning to figure out what perspective to use in answering student questions. Similarly, an engineer who is designing her third distillation plant already has a pretty good idea about the sequence of types of models that will be required. It seems likely that achieving efficient use of compositional models in applications will need both caching of models and acquiring knowledge of common patterns of model usage. Explanation-based learning and analogical reasoning seem to be promising ways to acquire such knowledge.

Acknowledgement

The authors wish to thank Brian Williams, Dan Weld, Mark Stefik, Mark Shirley, Sanjay Mittal, Johan de Kleer and Danny Bobrow for productive discussions about this work. John Collins provided valuable commentary

and technical assistance. Significant portions of our thermodynamics model were developed in collaboration with John Collins [5]. We thank Mark Shirley for providing the Runge-Kutta code, who in turn obtained it from Elisha Sacks.

This research was supported in part by the National Aeronautics and Space Administration, Contract No. NASA NAG-9137, by the Office of Naval Research, Contract No. N00014-85-K-0225, and by an NSF Presidential Young Investigator Award.

References

[1] S. Addanki, R. Cremonini and J.S. Penberthy, Reasoning about assumptions in graphs of models, in: *Proceedings IJCAI-89*, Detroit, MI (1989) 1432–1438.

[2] S. Addanki, R. Cremonini and J.S. Penberthy, Graphs of models, *Artif. Intell.* **51** (1991) 145–177 (this volume).

[3] F.G. Amador and D.S. Weld, Multi-level modeling of populations, in: *Fourth International Workshop on Qualitative Physics*, Lugano, Switzerland (1990) 210–219.

[4] D.P. Benjamin, ed., *Change of Representation and Inductive Bias* (Kluwer, Dordrecht, Netherlands, 1989).

[5] J. Collins, Building qualitative models of thermodynamic processes, in: *Proceedings Workshop on Qualitative Reasoning*, Stanford, CA (1989).

[6] J. Collins and K.D. Forbus, Reasoning about fluids via molecular collections, in: *Proceedings AAAI-87*, Seattle, WA (1987) 590–594.

[7] J.M. Crawford, A. Farquhar and B. Kuipers, QPC: a compiler from physical models into qualitative differential equations, in: *Proceedings AAAI-90*, Boston, MA (1990).

[8] R. Davis, Diagnostic reasoning based on structure and behavior, *Artif. Intell.* **24** (1984) 347–410.

[9] J. de Kleer, An assumption-based TMS, *Artif. Intell.* **28** (2) (1986) 127–162.

[10] J. de Kleer, Problem solving with the ATMS, *Artif. Intell.* **28** (2) (1986) 197–224.

[11] J. de Kleer and J.S. Brown, A qualitative physics based on confluences, *Artif. Intell.* **24** (1984) 7–83.

[12] T. Ellman, R. Keller and J. Mostow, eds., *Working Notes of the Automatic Generation of Approximations and Abstractions Workshop*, Boston, MA (1990).

[13] B. Falkenhainer, Learning from physical analogies: a study in analogy and the explanation process, Ph.D. Thesis, Tech. Report UIUCDCS-R-88-1479, University of Illinois at Urbana-Champaign, IL (1988).

[14] B. Falkenhainer and K.D. Forbus, Setting up large-scale qualitative models, in: *Proceedings AAAI-88*, St. Paul, MN (1988) 301–306

[15] P.A. Fishwick, The role of process abstraction in simulation, *IEEE Trans. Syst. Man Cybern.* **18** (1) (1988) 18–39.

[16] K.D. Forbus, Qualitative process theory, *Artif. Intell.* **24** (1984) 85–168.

[17] K.D. Forbus, The qualitative process engine, in: D.S. Weld and J. de Kleer, eds., *Readings in Qualitative Reasoning about Physical Systems* (Morgan Kaufmann, San Mateo, CA, 1990) 220–235.

[18] K.D. Forbus and J. de Kleer, Focusing the ATMS, in: *Proceedings AAAI-88*, St. Paul, MN (1988) 193–198.

[19] K.D. Forbus and B. Falkenhainer, Self-explanatory simulations: an integration of qualitative and quantitative knowledge, in: *Proceedings AAAI-90*, Boston, MA (1990).

[20] K.D. Forbus and A. Stevens, Using qualitative simulation to generate explanations, in: *Proceedings Third Meeting of the Cognitive Science Society*, Berkeley, CA (1981).

[21] R.W. Fox and A.T. McDonald, *Introduction to Fluid Mechanics* (Wiley, New York, 3rd ed., 1985).

[22] I. Granet, *Fluid Mechanics for Engineering Technology* (Prentice Hall, Englewood Cliffs, NJ, 1989).

[23] W.C. Hamscher, Temporally coarse representation of behavior for model-based troubleshooting of digital circuits, in: *Proceedings IJCAI-89*, Detroit, MI (1989).

[24] P.J. Hayes, Naive physics 1: ontology for liquids, in: J. Hobbs and R. Moore, eds., *Formal Theories of the Commonsense World* (Ablex, Norwood, NJ, 1985).

[25] Y. Iwasaki, Causal ordering in a mixed structure, in: *Proceedings AAAI-88*, St. Paul, MN (1988) 313–318.

[26] Y. Iwasaki and I. Bhandari, Formal basis for commonsense abstraction of dynamic systems, in: *Proceedings AAAI-88*, St. Paul, MN (1988) 307–312.

[27] J. Katzenelson, AEDNET: a simulator for nonlinear networks, *Proc. IEEE* **54** (11) (1966).

[28] B.J. Kuipers, Qualitative simulation, *Artif. Intell.* **29** (1986) 289–338.

[29] B.J. Kuipers, Abstraction by time-scale in qualitative simulation, in: *Proceedings AAAI-87*, Seattle, WA (1987) 621–625.

[30] B.J. Kuipers and D. Berleant, Using incomplete quantitative knowledge in qualitative reasoning, in: *Proceedings AAAI-88*, St. Paul, MN (1988) 324–329.

[31] Z.-Y. Liu and A.M. Farley, Shifting ontological perspectives in reasoning about physical systems, in: *Proceedings AAAI-90*, Boston, MA (1990) 395–400.

[32] D.A. McAllester, An outlook on truth maintenance, MIT AI Memo 551, MIT, Cambridge, MA (1980).

[33] S. Mittal and B. Falkenhainer, Dynamic constraint satisfaction problems, in: *Proceedings AAAI-90*, Boston, MA (1990).

[34] D. Molle, Qualitative simulation of dynamic chemical processes, Tech. Report AI89-107, AI Laboratory, University of Texas at Austin, TX (1989).

[35] K.S. Murray and B.W. Porter, Controlling search for the consequences of new information during knowledge integration, in: *Proceedings Sixth International Workshop on Machine Learning* (1989) 290–295.

[36] L. Nagel, SPICE2: a computer program to simulate semiconductor circuits, Tech. Report UCB ERL-M250, Electronics Research Laboratory, University of California, Berkeley, CA (1975).

[37] W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes* (Cambridge University Press, Cambridge, 1986).

[38] S.A. Rajamoney, Explanation-based theory revision: an approach to the problems of incomplete and incorrect theories, Ph.D. Thesis, Tech. Report UILU-ENG-88-2264, University of Illinois at Urbana-Champaign, IL (1988).

[39] S.A. Rajamoney and S.H. Koo, Qualitative reasoning with microscopic theories, in: *Proceedings AAAI-90*, Boston, MA (1990) 401–406.

[40] R. Reiter, On closed world data bases, in: H. Gallaire and J. Minker, eds., *Logic and Data Bases* (Plenum, New York, 1978).

[41] E. Sacks, Hierarchical reasoning about inequalities, in: *Proceedings AAAI-87*, Seattle, WA (1987) 649–654.

[42] M. Shirley and B. Falkenhainer, Explicit reasoning about accuracy for approximating physical systems, in: *Working Notes of the Automatic Generation of Approximations and Abstractions Workshop* (1990).

[43] R. Simmons, "Commonsense" arithmetic reasoning, in: *Proceedings AAAI-86*, Philadelphia, PA (1986) 118–124.

[44] G.G. Skorstad and K.D. Forbus, Qualitative and quantitative reasoning about thermodynamics, in: *Proceedings Eleventh Annual Conference of the Cognitive Science Society*, Ann Arbor, MI (1989).

[45] D. Sleeman and J.S. Brown, *Intelligent Tutoring Systems* (Academic Press, New York, 1982).

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

[47] G.J. Sussman and G.L. Steele Jr, CONSTRAINTS: a language for expressing almost-hierarchical descriptions, *Artif. Intell.* **14** (1980) 1–39.
[48] G.J. Sussman, SLICES: at the boundary between analysis and synthesis, AI Lab Memo 433, MIT, Cambridge, MA (1977).
[49] J. Van Baalen, ed., *Change of Representation and Problem Reformulation Workshop (Informal Proceedings)*, Menlo Park, CA (1990).
[50] D. Weld, The use of aggregation in causal simulation, *Artif. Intell.* **30** (1986) 1–34.
[51] D. Weld, Approximation reformulation, in: *Proceedings AAAI-90*, Boston, MA (1990).
[52] J.R. Welty, C.E. Wicks and R.E. Wilson, *Fundamentals of Momentum, Heat, and Mass Transfer*, (Wiley, New York, 2nd ed., 1984).
[53] E. Wenger, *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge* (Morgan Kaufmann, Los Altos, CA, 1987).
[54] B.C. Williams, Qualitative analysis of MOS circuits, *Artif. Intell.* **24** (1984) 281–346.
[55] B.C. Williams, MINIMA: a symbolic approach to qualitative algebraic reasoning, in: *Proceedings AAAI-88*, St. Paul, MN (1988) 264–269.