Automated Qualitative Modeling of Dynamic Physical Systems

Jonathan Amsterdam

January 1993

This work has been supported in part by research grant no. R01 LM 04493 and by Medical Informatics training grant no. T 15 LM 07092 from the National Library of Medicine.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014–89–J–3202.

This report is a revised version of my Ph.D. thesis.

Abstract

This report describes MM, a computer program that can model a variety of mechanical and fluid systems. It addresses several issues: What is the appropriate input to the modeling process? How should the search for models be organized? What evidence can be brought to bear to constrain the task?

MM takes as input both a description of the structure of the system to be modeled and a trace of the system's behavior. The structure can be represented by 2D line drawings augmented with some additional information, or by component types and their interconnections, or both. The behavior trace provides the program with qualitative information about the behavior that the user wishes to capture with a model.

MM organizes its search for models using a generalized, energy-based modeling framework and represents its models using bond graphs, a notation optimized for that framework. The program searches the system structure for plausible locations of primitive elements, connects the elements together via pathways that conduct power, and verifies the resulting model by comparing its predictions with the user-supplied behavior.

MM attempts to analyze models by using some general facts about systems and their behavior to relate qualitative properties of the desired behavior to qualitative properties of the model. These facts enable a more focussed search for models than would be obtained by mere comparison of desired and predicted behaviors. When these facts do not apply, MM uses *behavior-constrained qualitative simulation,* a modified version of standard qualitative simulation, to verify candidate models efficiently. If multiple candidate models are found, MM can design experiments to distinguish among them.

To my parents

Acknowledgments

I would first like to thank my advisor, Peter Szolovits, for continual faith, generosity, kindness, support, constructive criticism and gentle prodding. My readers, Patrick Winston and Neville Hogan, have also been extremely kind and gracious. All three committee members provided substantive comments that have improved the content and presentation of this thesis. Neville Hogan taught me nearly everything I know about modeling and was instrumental in debugging the engineering side of the thesis. Peter Breedveld, Michael Caine, David Clemens, William Durfee, Derek Rowell, Warren Seering, Karl Ulrich, and John Wyatt also contributed to my engineering and modeling education.

My life over the past several years has been enriched by many wonderful friends, including Scott Brewer, Miriam Cardozo, Jennifer Duncan, Oren Etzioni, Grace Freedman, Michael Kashket, Rachel Lefkowitz, Liora Minkin, Melanie Mitchell, Barbara Moore, Larry Penn, Paulo Pereira, Paul Resnick, Naomi Ribner, Ruth Schonfeld, Rebecca Simmons, Cindy Wible, and especially David Clemens, Nomi Harris, Michele Popper, Karen Sarachik, and Rachel Thorburn. I apologize to anyone I may have omitted.

My squash partners, Lijian Chen, David Clemens, Robert Givan, David Jacobs, Partha Niyogi and David Sturman, kept me running three hours a week so I could sit the rest of the time.

My officemates, Sajit Rao and Ruth Schonfeld, have borne much with no complaints. At last the day of their deliverance is at hand.

In my secret life as a thespian I have been aided and abetted by mentors Alan Brody, Michael Ouellette and Janet Sonenberg, and inspired by colleagues too numerous to mention. Special thanks go to Rachel Thorburn for starting me out and to Jennifer Duncan for keeping me in. David Thorburn also deserves thanks for his literary encouragement, literate criticism, and warm friendship.

I thank my relatives, especially my sister, Valerie, her husband Kevin, and my grandparents Charles and Molly, for their unconditional support and affection.

I am especially grateful to Valerie Leiter for accompanying me through my final, arduous year with compassion, encouragement and love.

Finally I thank my parents, Daniel and Carol, who have made this and everything else possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is about the *automatic qualitative modeling* of *dynamic physical systems*. Modeling, for the purposes of this thesis, is the activity of creating a mathematical system whose behavior is analogous to some other object or system. In this thesis the models will in general be sets of differential equations with time the independent variable; this is the essence of the meaning of *dynamic*. The equations will be models of *physical* systems—mechanical and fluid devices like motors, pumps, a U-shaped fluid-filled tube, and so on—rather than, say, economic or biological systems. The modeling process discussed here is *qualitative* in that it makes no attempt to determine particular numerical parameters of the model in order to make the model conform exactly with the modeled system's behavior; instead, the models provide a rough approximation, a first cut. (The notion of a qualitative model is familiar to workers in Artificial Intelligence (AI) and will be made more precise in Section 3.2.) Finally, this thesis is about the *automation* of the modeling task. It describes a working computer program, MM,[1] that is capable of modeling some simple physical systems.

I was motivated to write MM by two observations. The first occurred while taking a mechanical engineering course on modeling. I noticed that many students had trouble with modeling even simple systems. Sometimes the difficulty was that the student's model was incapable of describing even the gross behavior of system being modeled. More subtle was the tendency to over-model—to create a more complex model than necessary. Experienced modelers tended not to have these problems. I began to wonder whether a program could model simple physical systems while avoiding these basic errors.

My second observation arose out of reading the automated modeling literature. Work in automated modeling seemed to beg certain modeling questions by providing programs with system descriptions that already embodied important modeling choices. I wished to explore whether automated modeling was feasible when the input was in a somewhat more "raw" form, in which little besides the geometry of the

[1] "Machine Modeler" is one plausible expansion of this acronym.

system was presented to the program.

So in writing MM my goal was to uncover the principles that make it possible to model physical systems, particularly when the program's input embodied few or no modeling decisions.

I believe I have achieved this goal to a large extent. Here, in brief, are the principles that describe my approach:

- *Modeling is a generate-and-test process.* Like many complex tasks, modeling consists of conjecturing candidates, then analyzing them to see if they are appropriate. MM's architecture mirrors this breakdown: the program proposes candidate models based on heuristic rules, then passes the candidates to an analysis phase.

- *Multiple structural representations are useful.* Describing a system in geometric terms helps avoid usurping modeling decisions, but sometimes it makes more sense to describe a part of a system by using a component name, such as "motor." MM accepts both geometric and component descriptions, and allows the two representations to be combined.

- *Modeling is aided by knowing both structure and behavior.* Modeling a physical system is difficult or impossible without some idea of the geometrical or component-and-connection structure of the system; and model analysis requires some standard against which to evaluate candidate models. MM's rules use the structure to provide hints about plausible model elements, and MM's analysis phase compares the model against a given behavior trace to determine if the model's behavior matches the system's and, in many cases, to diagnose the model's flaws if the behaviors disagree.

- *The energy-based approach aids automated modeling.* MM employs ideas from system dynamics that treat systems as collections of elements that exchange, store and transform energy. These ideas have long been used by human modelers; MM's success demonstrates that they are useful for machine modelers as well.

1.1 Why Modeling is Important

Modeling is one of the most central and challenging tasks in engineering and science. It comes into play in all of the following activities:

Understanding. Scientists are continually trying to model systems that they do not understand, or understand only imperfectly. A mathematical model of a physical system can by itself provide insight about the system, and can focus the search for other phenomena that also provide insight.

Prediction. If a model of a system can be simulated faster than the system itself evolves, then the model can be used to predict the future behavior of the system. This can be useful for warning (e.g. earthquakes), investment (e.g. the stock market) or for controlling the system to keep its behavior within pre-determined limits.

Design. Building a new device of more than trivial complexity requires modeling, in order to ensure that the design will provide the intended behavior at reasonable efficiency and cost.

Diagnosis. Determining what is wrong with a broken system is greatly facilitated by having a model of the properly functioning system, and also by having models of the system in various failure modes.

Testing. A model can aid in designing a comprehensive test suite for a device.

This thesis is concerned with modeling for engineering activities, rather than the kind of modeling in which science engages. All the systems[2] MM can model are well-understood scientifically; MM is not a scientific discovery program along the lines of, say, BACON [20], which could also be said to be constructing models of systems. Thus the first goal, understanding, does not motivate the sort of modeling MM does. It is still a non-trivial problem to construct a good model for a system even when all of the system's phenomena are well-understood scientifically. The focus is on choosing a small and appropriate set of known phenomena to describe the system, rather than on discovering new scientific laws.

1.2 Why Modeling is Hard

One might argue that modeling is not all that difficult. One might use as a supporting example the system of Figure 1-1, which is a simple electrical circuit. Modeling this system, it could be argued, is quite simple: just write down the equations governing the resistor, capacitor and voltage source—a matter of table lookup—add Kirchoff's voltage and current laws, and solve for the desired output. This is a process most undergraduates master quickly. More relevantly, it is patently easy to automate: collecting the equations is trivial, and they can be handed off to one of several high-quality symbolic mathematics programs for solution.

I agree that going from schematic to model—set of equations—is easy. It is so trivial, in fact, that I would not call it modeling. The schematic already *is* the model; it already is, for all intents and purposes, a mathematical description of a system. In fact, a schematic-like notation called a *bond graph* is the output, not the input, to MM. (I explain bond graphs in Chapter 2.)

[2]The word "system" in this thesis always refers to some external device; the word is never used to refer to the MM program or any other piece of software.

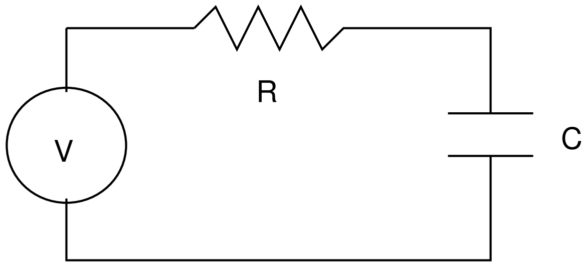


Figure 1-1: An electrical circuit

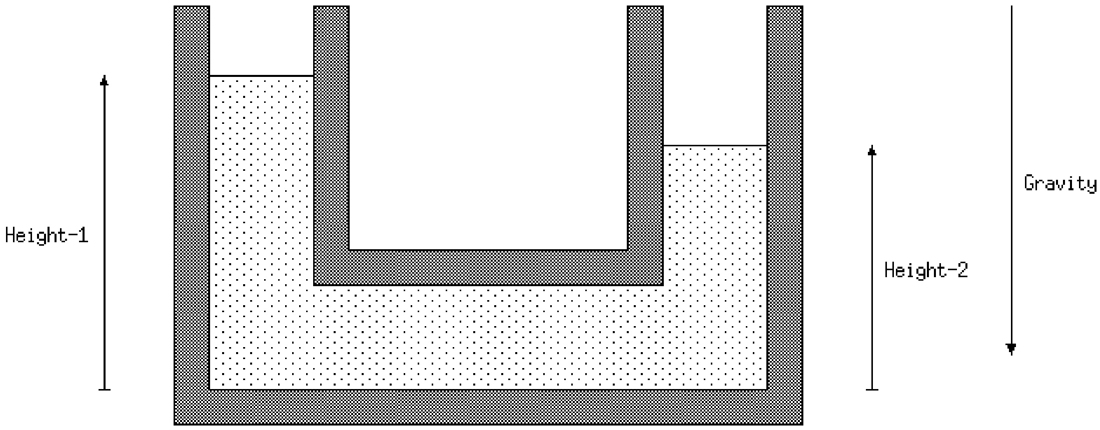


Figure 1-2: A U-tube

$$
\begin{aligned}
h_1 &= m_1(v_1) \\
h_2 &= m_2(v_2) \\
p_1 &= m_3(v_1) \\
p_2 &= m_4(v_2) \\
f &= m_5(p_1 - p_2) \\
\dot{v_1} &= -f \\
\dot{v_2} &= f
\end{aligned}
$$

Figure 1-3: "Standard" model for the U-tube

For a better example of what makes modeling difficult, consider the U-tube of Figure 1-2. The figure depicts a U-shaped tube of some solid material, like glass or plastic, partially filled with fluid. The figure is an actual rendering of the input to MM. If some excess fluid is quickly added to the left side of the tube, the amount of fluid in the left side will decrease and the amount in the right will increase until both sides are at the same height. (Imagine that the fluid is molasses, so that there is no oscillation.)

The "standard" model for this system, given in many papers on qualitative reasoning, is shown in Figure 1-3. In the equations, the h_i and v_i are the fluid heights and volumes in the two sides of the U-tube, the p_i are the pressures at the sides' bottoms, f is the rate of fluid flow through the tube, and the m_i are monotonically increasing functions. The model is "correct" in the sense that it predicts the described behavior of the system. Indeed, I would argue that the model is more than merely correct; it is in fact a good model, because it contains only what is necessary to explain the behavior.

There is a simple, straightforward explanation for how one might arrive at the above model. Begin with the facts that height is proportional to volume in a container, and the volume of fluid in a container is proportional to the pressure at the bottom of the container. This gives us the first four equations, where the containers are the two sides of the U-tube. Now note that in any tube or pipe carrying fluid, there is some resistance to fluid flow, such that the higher the difference in pressure, the faster the fluid moves. The fifth equation represents this resistance in the horizontal section of the U-tube. The remaining two equations merely express the definition that fluid flow rate is the time derivative of volume, and provide a consistent sign convention for the direction of flow.

This explanation seems plausible, but on closer inspection it contains a number of puzzling gaps. It begins by talking about properties of fluid in a container, then applies these properties to each side of the U-tube. But why is a side of the U-tube a container? Why couldn't we take as a container, say, the leftmost vertical inch of

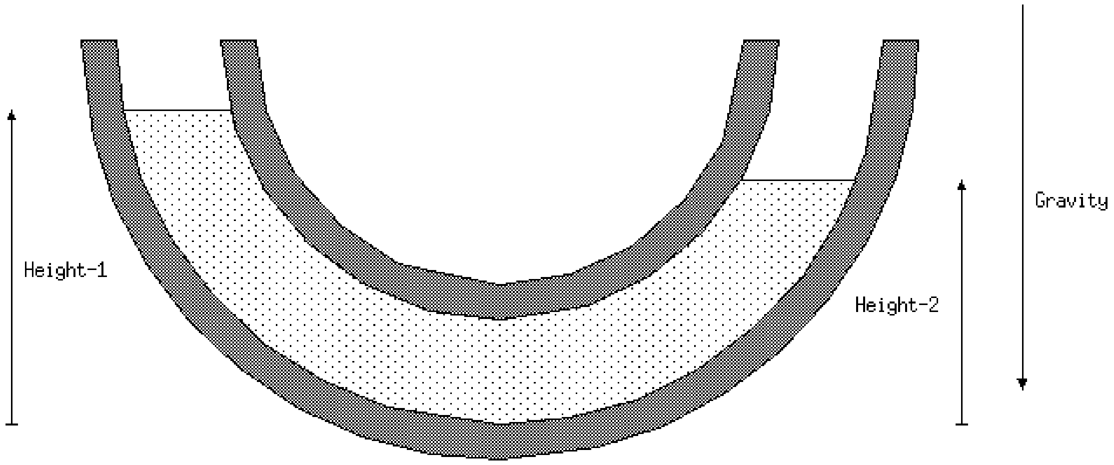


Figure 1-4: A curved U-tube

the left side of the U-tube, or the left side plus one inch of the horizontal section? After all, the same properties hold for those regions of space. For that matter, the horizontal section itself could be a container.

The discussion of resistance is puzzling as well. For if "any tube or pipe" has resistance, then the sides of the U-tube should have resistance just as does the horizontal section. Of course, the model neglects these resistances, just as it quite properly neglects many other effects; but why then does it include the resistance of the horizontal section? If one is going to neglect resistance, why not do a thorough job of it? Perhaps, you say, the resistance of the sides is lower than the resistance of the horizontal section. But assume it isn't; I claim that the above model is *still* a reasonable one.

But the most dramatic failure of the explanation appears when we note that the U-tube of Figure 1-4 is validly described by *the same model,* even though there is patently no principled way to divide the U-tube up into horizontal and vertical parts.

This example demonstrates the *lumping* problem: how should a spatially continuous system be divided into discrete lumps? Lumping is one of the essential abstraction techniques of physical systems modeling. When it succeeds, the system can be described by a set of ordinary differential equations. When lumping is not possible, or not sufficiently accurate, the modeler must enter the much more complex world of partial differential equations.

One of the goals of this thesis is to demonstrate that it is possible to solve the lumping problem, if only partly, by means of a computer program. One of the crucial observations enabling a solution to the lumping problem is that *lump boundaries are largely arbitrary.* Many different choices for lumps will produce qualitatively equivalent models. (A precise definition of qualitative equivalence is provided later.) The key to solving the lumping problem is avoid fastidiousness in choosing lumps; a modeler must be willing to make arbitrary decisions, up to a point.

1.3 A Glimpse of MM in Action

To preview Chapter 5, here is a summary of how MM solves the U-tube problem. Keep in mind that this is a greatly simplified description; many aspects of the process have been abstracted or simply ignored. All of the magic is dispelled in chapters 3 through 5.

MM begins with a description of the U-tube's structure in the form of a set of two-dimensional polygons annotated with information about the composition of the object. For the U-tube, there are three polygons: the two walls and the fluid inside. MM is also given a trace of the behavior of the system. The U-tube system is described as having no input and two outputs, the heights of the fluid in each side of the tube. MM is told that the height of fluid in the left side of the tube decreases, and that in the right side increases, until both come to rest at some intermediate value. The behavior trace is not a plot of numerical values, but is instead a qualitative description, with pretty much the same information as is contained in the previous sentence. The complete input to MM is given in Chapter 3.

MM first tries to interpret the input and output variables. MM has a general typology of quantities based on energy domain (hydraulic, electrical, and so on) and the generalized power variables *effort* and *flow*.[3] MM attempts to fit the input and output variables into this framework in order to facilitate model construction and analysis. For the U-tube, MM determines that the height of the fluid is the integral of a flow variable, fluid flow rate.

Once the variables are interpreted, MM analyzes the resulting model. MM's analysis portion consists of several special-purpose rules based on general properties of physical systems, as well as a general-purpose qualitative simulator (based on QSIM [19]) when the special-purpose rules fail. In this case, one of the special-purpose rules is triggered; it recognizes that the given behavior of the system implies the presence of a resistance, and since the model contains no resistance, it is labeled as defective.[4] In fact, MM does not simply give a Pass/Fail grade to its models; rather, it attempts to classify their problems in order to focus the search for improvements. In this case, as you might guess, the model is labelled as lacking a necessary resistance.

The model is then enqueued for subsequent processing by MM's conjecture phase, which now is charged with looking for plausible locations for resistance. No geometrical feature suggesting resistance is present, so MM arbitrarily chooses regions of the

[3] Chapter 2 explains these concepts in detail.

[4] Chapter 4 describes the analysis portion of MM, including the exact content of and justification for this resistance rule.

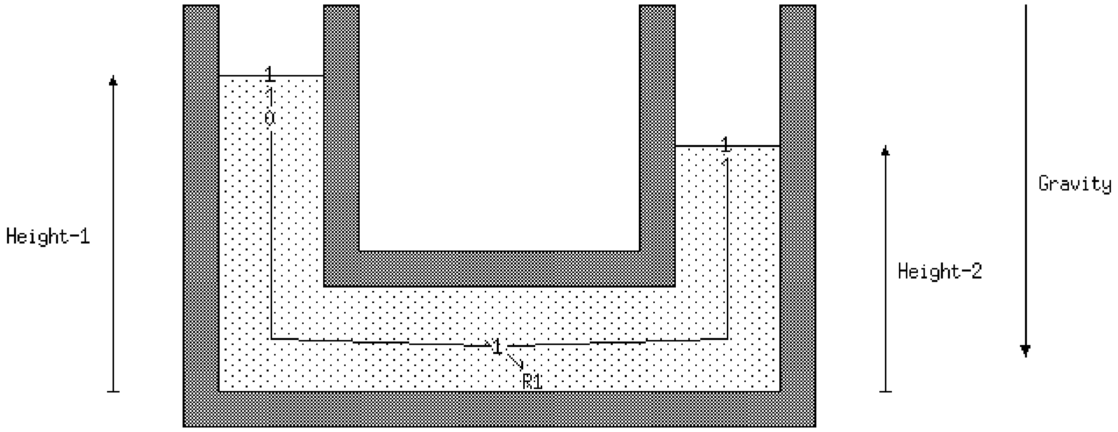


Figure 1-5: U-tube with resistance

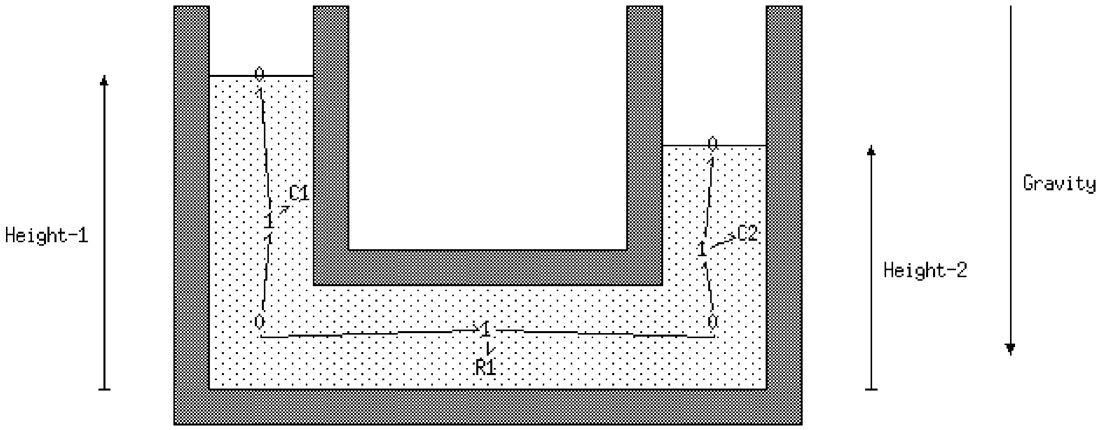


Figure 1-6: U-tube with superimposed bond graph

system. One of these regions is the horizontal portion of the U-tube, which gives rise to the model of Figure 1-5, shown superimposed on the U-tube.

The analysis phase now determines that the model's order[5] is too low to account for the given behavior. Rules are activated which look for the presence of effects that might raise the model's order. For the U-tube, the most important effect is *capacitance,* a proportionality between an effort (e.g. pressure) and the integral of a flow (e.g. position or height). Since choice of lumps is to some extent arbitrary, MM simply looks for rectangles that could represent regions of the system exhibiting capacitance. MM contains a rule for fluid capacitance that says to look for rectangles that contain fluid in the presence of gravity. In the case of the U-tube, the relevant rectangles are the sides of the U-tube. MM attempts to construct the largest viable rectangle, so the entire sides of the rectilinear U-tube are treated as a capacitance region. For the curved U-tube, MM's rectangles begin at the top and extend an arbitrary amount along the tube. Figure 1-6 shows the resulting model. (The bond graph notation will be demystified in Chapter 2.) The equations for this model are identical to those of Figure 1-3. MM also arrives at this model when presented with the curved U-tube.

1.4 Roadmap

The goals of this introductory chapter were to present the problem MM is trying to solve, and to provide a glimpse into its operation. The next chapter supplies a compact but thorough introduction to energy-based modeling and to the bond graph formalism used by MM. Chapter 3 covers the internals of the MM program, except for the analysis phase, which is discussed in Chapter 4. Chapter 5 explores four example systems, explaining in detail how MM models them. Chapter 6 reviews related work and concludes. An appendix lists all of MM's rules.

[5]Section 4.1 defines "order" and explains how it is used by MM.

Chapter 2

Energy-Based Modeling

This chapter explains *energy-based modeling,* the theoretical foundation underlying MM's knowledge of physics. Energy-based modeling is a framework for modeling physical systems. It is closely related to system dynamics, a modeling paradigm that first became popular in the middle of this century and continues in popularity today. The chapter also introduces the *bond graph* notation used throughout the thesis.

Modeling begins by drawing a boundary between a *system* and its *environment.* Where the boundary is drawn depends on the needs and desires of the modeler. The basic idea behind energy-based modeling is to track the flow of energy between a system and its environment, and between the parts of a system. A physical system can be viewed as a collection of processes that transform, store, supply or dissipate energy. Any physical system can be described as a connected group of such processes. Since the concept of energy, and its instantaneous rate of change power, unites all of physics, energy-based modeling provides a uniform framework for describing a very great variety of physical systems.

The first important observation of an energy-based approach to modeling is that in any physical domain, power can be represented as the product of two real-valued variables. In the electrical domain, these variables are commonly taken to be voltage and current. In the domain of mechanical translational motion, the variables are force and velocity. For any sort of energy domain, there are always two variables whose product is power. And luckily, for most common domains, there is a familiar choice of variables.

Since we are trying to capture a wide variety of physical domains in a single framework, it will be necessary to use a terminology that is divorced from any particular domain. We would not want to use "voltage" and "current" to name the two power variables in general, because those terms connote the electrical domain. Instead, we will call these variables *effort* and *flow.* The power variables for the energy domains handled by MM are given in table 2.1. The mathematics is indifferent to which variable is called the effort and which the flow, but there are well-entrenched conventions for all the listed domains and several others as well.

Domain	*Effort variable*	*Flow Variable*
Electrical	voltage	current
Mechanical Translation	force	velocity
Mechanical Rotation	torque	angular velocity
Hydraulic	pressure	flow rate

Table 2.1: Effort and flow variables for energy domains

The next move is to assume that the energy of a system can be parceled up. First, we distinguish between the energy contained within the system, and the energy of the environment. We assume that the system and its environment exchange energy through a fixed number of *ports;* we further assume that energy transactions within the system can be localized to a finite number of regions which are themselves connected by a finite set of ports. This is the *lumping assumption.* With this assumption in place, we can now ask about the kinds of regions that exist, and the different ways they can be interconnected. Let us call both of these notions—the energy regions, and their methods of interconnection—*elements,* for reasons that will soon become clear.

2.1 The Elements of Modeling

A priori, one might imagine that the catalogue of elements is limitless. But in fact there are very few distinct elements of any physical interest. Two elements suffice for describing system-environment interactions; three elements cover the kinds of energy regions; two elements deal with transformation of energy; and two more elements are concerned with the partitioning of energy. These nine elements (and their multi-variable generalizations) are sufficient for describing the dynamic behavior of an enormous variety of physical systems, from U-tubes to jet airplanes. The phenomena capable of description range from Newton's laws to the unusual behavior of supercooled helium [3].

2.1.1 Resistors

Let us begin the list of elements with the *resistor.* A resistor element represents an object or process that dissipates energy, i.e. that transfers energy from the system to the environment. Typical examples are an electrical resistor and a mechanical dashpot. All such dissipation processes relate an effort quantity to a flow quantity. (The relationship established by an element is called the *characteristic* or *constitutive relation* of the element.) For instance, an ordinary electrical resistor obeys Ohm's law: voltage (effort) equals current (flow) times resistance. A dashpot obeys a similar law for friction, in which force (effort) is proportional to velocity (flow).

An electrical resistor and a dashpot are both *passive* elements; they always dissipate power. The characteristic of a passive resistor—its effort vs. flow plot—increases monotonically and passes through the origin; it lies entirely in the first and third quadrants. Power, the product of effort and flow, is always positive, meaning the net power flow is *into* the device—it is removing, not supplying, energy.

Some resistors, such as an amplifier, appear to supply power when they are displaced from equilibrium. Such a resistor is called *locally active.* Part of its characteristic has negative slope.

MM assumes that *all resistors are passive.* MM does not assume that the resistor is linear (i.e. that its characteristic is a linear function). For a physical justification of MM's assumptions, see [2].

2.1.2 Energy Storage

In many energy domains, energy is manifest in two forms: "potential" and "kinetic." Usually, kinetic energy involves movement and potential some sort of location in a potential field, but this need not always be the case. Two elements, one for each energy form, represent the storage of energy.

A *capacitor* stores potential energy. A spring is a mechanical capacitor; when compressed, it is storing an amount of energy related to its displacement (the difference between its rest length and its current length). A linear spring obeys Hooke's law, applying a force linearly proportional to its displacement. A typical nonlinear spring will still exert a force that increases monotonically with the displacement from its rest state, although non-monotonic springs can be constructed. In general, a capacitor relates an effort to a *generalized displacement* (the integral of a flow). In the mechanical case, force is the effort and displacement is the flow integral. An electrical capacitor is also, as you might guess, a capacitor; a linear capacitor obeys $i = C\frac{dv}{dt}$, which is just the derivative of the conventional capacitor relation: current (flow) is proportional to the change in voltage (derivative of effort). A capacitor is passive by definition. MM makes the assumption that capacitors have monotonically increasing characteristics.

If you were to interchange effort and flow in the relation for a capacitor—a process known as *dualizing*—you would obtain the second energy storage element, the *inertia.*[1] An inertia stores kinetic energy. It relates a flow to a *generalized momentum* (the integral of an effort).

A typical inertia is an object undergoing translational motion. Such an object obeys $F = ma$, Newton's third law.[2] Armed with our energy-based modeling frame-

---

[1] I did not mention duality when talking about resistors because it isn't very interesting: the dual of a resistor is a resistor.

[2] No great difficulty ensues in the theory if we add the relativistic correction to this formula, but since it plays no role in the systems MM models (and in most engineered systems), I ignore it.

work, we can now see this law for what it is: nothing more than the derivative of an inertia relationship, relating the derivative of a flow (acceleration) to an effort (force). An electrical inductor is also an inertia; it relates a voltage (effort) to a change in current (flow derivative).

Because the relations we have been talking about—Newton's law and the laws for electrical capacitors and inductors—are linear, we can switch freely between a relation and its derivative and integral. In general, however, we cannot. It is important to keep in mind that the defining relationships are between effort and flow integral (for capacitors), and flow and effort integral (for inertias).

MM makes the same assumptions about the characteristics of inertias as it does for capacitors: they are monotonically increasing and pass through the origin. A justification for the monotonicity assumption can be found in [2].

2.1.3 Sources

Two elements called *sources* represent the flow of energy into the system from the environment. The *effort source* is used to model effort inputs, and the *flow source* is for flow inputs. A battery could be modeled as an effort source, since it supplies a voltage (effort) independent of current (up to some limit). Another familiar effort source is gravity.

The characteristic of a source is not a relationship between effort and flow or their derivatives; rather, it is a relationship between time (or another independent variable) and a quantity. An effort source represents some function from time to effort, and similarly for a flow source. A source asserts nothing about the other quantity. For instance, nothing can be concluded about the flow into or out of an effort source without looking at the rest of the model.

What if the actual input to the system is not an effort or flow? As long as the input can be related to effort or flow by some function (possibly including integration or differentiation), we can still use an effort or flow source, whose characteristic—a plot of effort or flow over time—is the original input suitably transformed. For instance, if the input is a linearly increasing displacement, then it can be modeled as a constant flow (velocity) source.

2.1.4 Bond Graphs

We have seen enough elements to describe some simple systems, but we need a notation to do so. This section introduces *bond graphs,* a notation developed to support energy-based modeling.

In bond graphs, elements are written as one- or two-letter symbols. The symbols for the elements discussed so far are: R for resistor, C for capacitor, I for inertia, Se for effort source and Sf for flow source. Bond-graph elements are connected by *bonds,* drawn as lines. A bond represents two quantities, an effort and a flow. Each

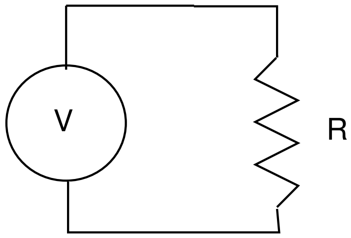


Figure 2-1: Circuit corresponding to bond graph Se—R

element can be connected to a certain number of bonds, much as chemical elements have valences that limit their connectivity. In bond graphs, these connections are called *ports.* All of the elements discussed so far are *one-ports;* they can be connected to exactly one bond.

Here is a very simple bond graph:

Se —— R

It describes a system consisting of an effort source connected to a resistor. The bond asserts two equalities. First, it says that the source's effort equals the resistor's effort. Second, it says that the source's flow equals the resistor's flow, though this isn't very interesting in this case, because an effort source's flow is arbitrary.

The effort along the bond is determined by the effort source; the flow depends on the resistor's characteristic. Figure 2-1 shows an equivalent circuit schematic. Note that the schematic symbol for a resistor has two connection points, but the bond graph symbol has only one. That is because a single port or bond represents two quantities, an effort (the voltage across the resistor) and a flow (the current through the resistor).

In order to turn the bond graph into a set of equations, we need one more piece of information: a sign convention. Which direction of power flow should be positive? Since resistors in general dissipate energy, the convention is to take positive power as flowing *into* the resistor. (The same convention applies to C and I elements.) The sign convention is indicated by adding a half-arrow to the bond, like so:

$\mathrm{Se} \rightharpoondown \mathrm{R}$

Now we can turn this bond graph into an equation. Use $Se(t)$ for the function of time represented by the effort source, and assume a linear resistor with characteristic $e = Rf$, where e is effort and f flow. Then we can write the equation for the flow along the bond as

$$f = Se(t)/R$$

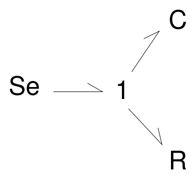
We get this by equating the e of the resistor's characteristic with the effort source's function, since they share the same bond.

With only one-ports at our disposal, we cannot create very complex systems. We need a way to connect several bonds together. Junctions provide this ability.

2.1.5 Junctions

Two elements, called *junctions,* exist solely to provide ways to connect bonds together. The junctions are *power-continuous:* the total power into a junction is always zero. Each junction relates the bonds connected to it in two ways, one for the bonds' efforts and one for their flows. The *1-junction* asserts that all bonds connected to it have the same flow, and that the bonds' efforts sum to zero. Its dual is the *0-junction,* which asserts a common effort and zero total flow. When summing a bond's effort or flow, the value is negated if the bond's sign arrow points away from the junction.

As a simple example, consider the bond graph



This bond graph is equivalent to the schematic of Figure 1-1 in Chapter 1. The 1-junction here says two things: first, the flows for the three one-ports are identical (the sign arrows don't matter); second, the efforts along the three bonds sum to zero. Using Se for the source's effort, e_C for the capacitor's and e_R for the resistor's, and observing the sign convention,

$$Se + -e_C + -e_R = 0,$$

or

$$Se = e_C + e_R$$

The other equations for this bond graph, assuming a linear capacitor and resistor, are

$$f_C = C\dot{e}_C$$
$$f_R = Re_R$$

From these equations we can derive the expression for a typical first-order linear system with damping,

$$Se = e_C + \frac{1}{RC}\dot{e}_C$$

It should be evident from the above that a bond graph corresponds straightforwardly to a set of equations—the equations obtained by writing down the relation or relations asserted by each element and bond. However, solving this set of equations, or even manipulating them into a useful form—say, a form suitable for computer simulation—is another matter. This process may involve considerable difficulties, even open research issues. But that is not the concern of modeling. The output of the modeling process is just a set of equations describing a system, and you should be convinced that a bond graph is essentially equivalent to such an equation set.

Junctions tend to be the sticking point in most people's mastery of bond graphs. Unfortunately they are not well-named, but the following mnemonic may help: a 1-junction asserts common flow, and both *one* and *flow* have one syllable; a 0-junction asserts common effort, and both *zero* and *effort* have two syllables. Another useful rule, for simple cases at least, is that 0-junctions correspond to parallel connections and 1-junctions correspond to series connections when the bond graph is translated to a schematic.

Readers familiar with the idea of a *constraint network* might find the following observation helpful. Take a circuit schematic and turn it into a constraint network, where the constraint boxes are the components' characteristics and Kirchoff's voltage and current laws (KVL and KCL). The resulting network will be isomorphic to a bond graph, where the component boxes correspond to one-ports, the KVL boxes are 0-junctions and the KCL boxes are 1-junctions.

Since a 1-junction asserts a common flow along all its bonds, the junction element itself can be used to represent a point where flow is measured. Similarly, a 0-junction can be used as a point where effort is measured. MM uses junctions in these ways to interpret input-output variables. The details are given in Chapter 5.

2.1.6 Transformers and Gyrators

Two other elements round out the energy-based modeling collection. They are typically used to represent the conversion of energy from one form or domain to another. Consider a hydraulic piston. The piston converts energy between the mechanical and hydraulic energy domains: When a force is applied to the shaft, a pressure is exerted at the fluid aperture, and vice versa. Similarly, the velocity of the shaft is proportional to the flow rate of the fluid.

Since the piston relates one effort (force) to another effort (pressure), and one flow (velocity) to another flow (fluid flow rate), it can be modeled by a *transformer.* The transformer represents a lossless energy conversion where efforts and flows are each proportional. It is written as TF in bond graphs, and it has two ports.

A *gyrator* is just like a transformer, except that the effort in one domain is proportional to the *flow* in the other, and vice versa. A good example is an ordinary DC motor: the voltage (effort) is proportional to the motor shaft angular velocity (flow), and the current (flow) is proportional to the torque (effort).

You might be wondering whether there are any transformer-like elements in which, say, efforts are proportional to each other but flows are identical, or some such combination. The answer is that there are not. The transformer and the gyrator exhaust the possibilities for *lossless, storage-free* energy transfer. This is easy to show. We consider only the transformer, since the gyrator is similar. Since a transformer is lossless and storage-free, the instantaneous power at one of its ports equals that at the other. We can write the power at one port as $e_1 f_1$ and at the other as $e_2 f_2$. (Recall that power is just the product of effort and flow.) We wish to equate these. Say the constant of proportionality between efforts is m (assuming a linear transformer for simplicity). That is, $e_2 = me_1$. So we have

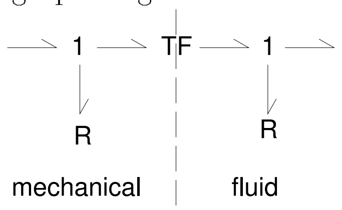
$$e_1 f_1 = e_2 f_2 = (me_1) f_2,$$

or

$$f_1 = m f_2$$

So not only are the flows proportional as well, but the constant of proportionality is $1/m$.

Of course, no real-world transformer or gyrator is perfect. The transformer and gyrator elements, like the other elements, isolate one aspect of energy behavior. The elements can be combined to produce more realistic behavior. For instance, if one wanted to model a hydraulic piston with power losses on both the mechanical and fluid sides, the following bond graph fragment would be appropriate:



(The dotted line and energy-domain labels are merely for exposition.) The resistors model the losses. They are connected to the central "spine" of the model by 1-junctions because we are assuming there is no leakage, so the entire flow is subject to resistance.

2.2 Pros and Cons of Energy-based Modeling

This chapter has provided an overview of energy-based modeling, the modeling framework used by MM. In this final section, I consider the advantages and disadvantages of the approach for automated modeling.

The chief disadvantage of energy-based modeling is that it puts constraints on the kinds of systems that can be modeled easily. Non-physical systems, like an economy, may have parts that do not conform to any element or collection of elements, because such systems do not have to obey any of the energy laws dictated by thermodynamics. Limiting models to the basics of energy-based modeling which I have described here further limits the set of modelable systems. For example, I have not mentioned the possibility of a resistor's characteristic varying over time, although such resistors do exist and can be accommodated in the energy-based modeling framework. Also omitted from the above discussion, and from MM's repertoire of modeling techniques, are such concepts as energy fields and information-only connections (so-called "active bonds"). The curious should consult [18] to learn about these topics, but they are not necessary for understanding the systems in this thesis. Their omission does, however, limit MM's scope.

The other side of this lack of expressiveness is a reduction in the search space. This is just the classic tradeoff between expressiveness and speed: by using a restricted representation, MM needs to examine fewer model candidates to find a successful model for those systems within its purview. But the advantage is more than just mathematical; we are not merely throwing out some scope to gain pruning power. The energy-based modeling representation embodies constraints that all physical systems must observe, and we can make use of these constraints to provide an additional boost to the search process. MM's analysis phase in particular makes use of these constraints.

Another disadvantage of the energy-based approach is that it can be difficult to express some models, even when they are models of physical systems. If we were to allow our elements to implement arbitrary functions between arbitrary quantities, then some modeling jobs might be easier. For example, say that part of the system being modeled is a "black box" whose behavior we have observed empirically. We have determined that the black box implements a relationship between the integral of a force and a displacement. If we were using energy-based modeling, we would be forced to construct some multi-element model for this black box, because no single element implements a relation between those variables. But if we were using an ad-hoc technique, we could simply use the empirical relationship directly in the model, without analyzing it further.

Again, there is another side to this disadvantage. All of the energy-based modeling elements are physically realizable, and a syntactically valid bond graph is *guaranteed* to meet the constraints of thermodynamics. For example, if a bond graph is constructed from the elements described here (namely, passive ones), then it can never put more energy into the environment than it removes. If an ad-hoc approach is used, no guarantees can be made about the energy behavior of the model, and opportunities for error are rife. The point about physical constraint made above can be made here as well: energy-based modeling elements force the constraints of physics to be observed, ensuring models that are at least physically plausible and aiding their analysis.

There are additional advantages to the energy-based modeling approach. By confining our vocabulary to a small set of primitive elements, we can simplify the modeling process. MM works by looking for features that suggest the presence of elements; fewer elements mean fewer feature-detectors. And since the elements' behaviors are independent of energy domain, the analysis phase is greatly simplified: the same analysis can apply to a resistor-capacitor circuit and a mass-spring system. Finally, carving up the world into energy domains allows us to organize our knowledge of physics concisely, and without redundancy. The methods and elements of energy-based modeling form a principled middle-ground between so-called first-principles reasoning and purely domain-dependent reasoning. They provide a physically motivated framework in which to place domain-dependent information.

Chapter 3

The MM Program

This chapter and the next describe the essential workings of the MM program. This chapter discusses most of the program, including input and output representations, the main loop and the rule interpreter. The following chapter covers the analysis phase. Descriptions of specific rules and of some other parts of the program, such as the experiment design facility, are omitted; they are best understood in context, and will be covered as they come up in the discussion of examples in Chapter 5.

3.1 Input Representations

One of the thorniest problems for research in automated modeling is choosing appropriate inputs for the modeling program. There are two difficulties here. First, if the work is intended (as is this thesis) to demonstrate the computer's ability to do modeling, then the program's input should not prejudge any modeling issues. We could come closest to this ideal if we hooked up a vision system to the program and pointed the camera at the system of interest, but this is not feasible at present. However, we must avoid erring in the other direction, in which we give the program a circuit schematic and it produces a set of equations; as I argued in the introduction, this process is not modeling.

The second difficulty involved in choosing appropriate inputs for the automated modeler is how to provide sufficient constraints on the modeling process to make modeling feasible. An infinite number of models can be constructed from the physical structure of the system; some information must be provided to enable a choice among them.

I have solutions for both of these problems, though admittedly neither is fully satisfactory. In order to avoid begging at least some crucial modeling decisions, namely those involved in lumping, MM accepts a description of the system's structure in the form of a *two-dimensional annotated geometric sketch.* (One can also describe a structure to MM using the traditional topological, or component-and-connection, representation.) To provide a constraint on modeling, the structural input must be accompanied by a *qualitative trace of behavior* which both identifies the system's input/output variables and provides an example of the system's dynamic behavior. It is up to the user to identify which variables are inputs and which are outputs.

3.1.1 Geometric Structure Representation

MM's geometric representation consists of two-dimensional polygons, each of which is annotated with information that cannot be conveyed by a simple line drawing. The decision to confine the representation to two dimensions was a pragmatic one: reasoning about three-dimensional space is vastly more complicated, and it was felt that the added complexity would not contribute significantly to the goals of this work.

To describe the structure of a system to MM, the user must divide it into a set of *parts.* The shape of a part is described by a polygon. In addition, each part must have a single *composition,* which must be one of the following:

rigid A solid, rigid object.

flexible A solid, non-rigid object that can deform and has elastic properties.

fluid A region filled with fluid.

Aside from being physically connected, describable by a single polygon, and having a single composition, there are no restrictions on what can be a part. The term "part" is something of a misnomer, since an MM part can consist of a portion of an off-the-shelf component, or several components.

It is crucial to note that the user's division of the system into parts does *not* in and of itself usurp modeling decisions. The part division of a structure is an input convenience only; it is ignored by the program. Internally, MM converts the part descriptions into a set of convex polygons of uniform composition called *segments.* Part boundaries play no role in MM's processing.

Unlike the choice of parts, the choice of composition for each part is a modeling decision that must be made by the user. Whether an object is best described as rigid, flexible or fluid depends on many factors, such as the temperature, the magnitude of the forces involved, and the time scale of interest. Depending on the conditions, a steel beam can be any of the three. Since MM knows about none of these quantitative factors, the user must make the choice.

In addition to its shape and composition, a part can be marked as *fixed.* A fixed part is immobile; it is not subject to any forces. Fluid parts cannot be fixed. One would mark a part fixed if, for example, it rests on a table and so is not subject to gravity.

A final annotation for parts is designed to overcome in some measure the limitations of a two-dimensional representation. Each part may have a list of parts to which it is connected. The connection is assumed to be in a plane other than that of the drawing, and is assumed to be rigid.

Figure 3-1 shows the structural description for the U-tube. This is the actual, unedited input to MM. (Section 5.1 describes how MM processes this input.) The first part of the description is a list of variables whose sole purpose is to facilitate cosmetic changes to the structure, so that it looks nicer on the screen. The second component of the description is a list of parts. Notice that in the U-tube structure, the outer wall of the U-tube is marked as being fixed, and the inner wall is connected to the outer wall. These two assertions ensure that the walls of the U-tube will not move when subject to gravity.

In addition to a list of parts, a structure representation can specify two other aspects of the system, both of which are present in the U-tube. A list of *position quantities* is used to relate the system's inputs and outputs, as described in the behavior, to the structure. Each position quantity has a name, which must agree with the name of some variable in the behavior. To indicate what the quantity represents, a vector and a line segment are given. (For `height-1`, the line segment is the first one of the part named `water`; for `height-2`, it is `water`'s fourth.) A change in the line segment along the direction of the vector is a change of the quantity. The `text-location` of a position quantity is solely cosmetic.

Finally, a structural description may specify a set of *external forces.* These act as (possibly implicit) inputs to the system. For the U-tube, gravity is the only external force. The vector specifies its direction, and the `acting-on` field lists the parts subject to the force, or contains the special symbol `:all` if the entire system is subject to the force, as is the case with gravity.

Most of the information in a structural description can be rendered on the screen. Figure 3-2 is a display of the U-tube's structural description. A part's composition is indicated by filling the polygon with a particular pattern.

3.1.2 Component Representation

It is not always desirable to describe a system in purely geometric terms. Providing the modeler, automated or otherwise, with descriptive labels for parts of the system is often a very useful expedient. An experienced person encountering a motor will quickly recognize it as a motor, either from geometric cues that are too complex for researchers in modeling to worry about programming (such recognition is a research problem in machine vision), or by reading the text engraved on the device. In this case it seems reasonable simply to tell the automated modeling program that the device is a motor.

MM has a component-based representation in which the user can describe a system as a set of components, each of which has a type hinting at its function and a set of *ports* through which it can be connected to other components.

Component types are first declared with a `defcomponent` form. Here is the description of an ordinary DC motor:

```
(defcomponent motor
```

```
(defstructure U-tube
  (vars (wall-width 10)    (start-x 120)
        (start-y 40)       (grav-x 360)
        (water-width 40)   (outer-wall-height 110))
  (parts
   (outer-tube-wall (composition rigid)
     (shape (begin-polygon start-x start-y)
    (go-right wall-width) (go-down outer-wall-height)
    (go-right 160) (go-up outer-wall-height)
    (go-right wall-width)
    (go-down (+ outer-wall-height wall-width)) (go-left 180)
    (go-up (+ outer-wall-height wall-width)) (end-polygon))
     (fixed))
   (water (composition fluid)
     (shape (begin-polygon (+ start-x wall-width) (+ start-y 20))
    (go-right 30) (go-down 60) (go-right 100)
    (go-up 40) (go-right 30) (go-down 70)
    (go-left 160) (go-up 90) (end-polygon)))
   (inner-tube-wall (composition rigid)
     (shape (begin-polygon (+ start-x wall-width water-width) start-y)
    (go-down 70) (go-right 80) (go-up 70)
    (go-right wall-width) (go-down 80) (go-left 100)
    (go-up 80) (go-right wall-width) (end-polygon))
     (connected-to outer-tube-wall)))
  (position-quantities
   (height-1 (fixed-point 100 (+ start-y outer-wall-height))
             (changing-point 100 (+ start-y 20))
             (linked-to water 0)
             (text-location :left))
   (height-2 (fixed-point 320 (+ start-y outer-wall-height))
             (changing-point 320 (+ start-y 40))
             (linked-to water 4)))
  (external-forces
   (gravity (vector grav-x start-y grav-x (+ start-y 100))
            (acting-on :all))))
```

Figure 3-1: U-tube structure

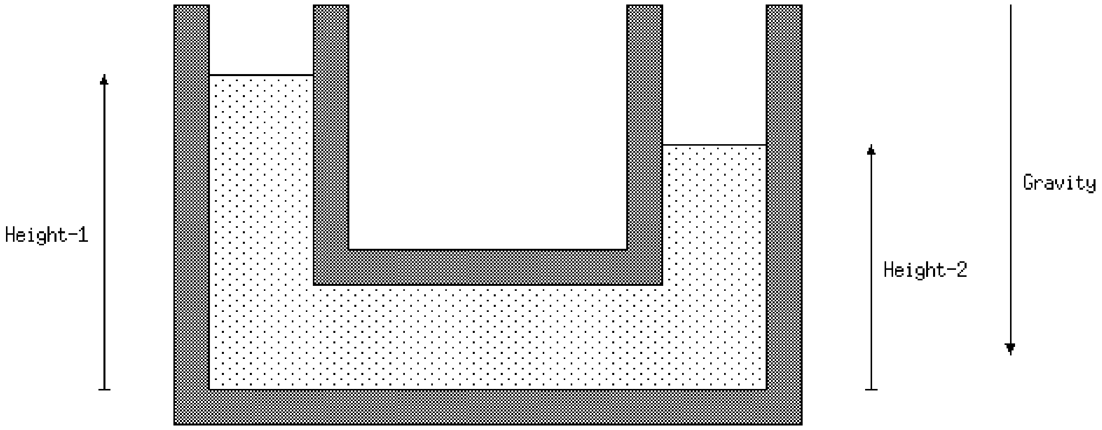


Figure 3-2: Rendering of the U-tube structural description

```
(ports (leads electrical)
       (shaft rotational))
(bare-model
 (GY leads shaft)))
```

Each `port` form contains a name and the energy domain for the port. The name is used to describe connections, and the energy domain is used both to type-check a system representation (connected ports must have the same domain) and as information for recognition rules.

The *bare model* of a component is the simplest model that still implements the basic functionality of the component. The bare model is expressed as a list whose first element is a bond graph element and whose remaining elements are either port names or other lists of the same form. This simple notation allows the description of any bond graph that is a tree, which suffices for most components, including all components considered in this thesis. The form in the motor example corresponds to the partial bond graph

leads — GY — shaft

which, as I discussed in Chapter 2, captures the essential behavior of a DC motor: voltage is proportional to shaft velocity, and current is proportional to torque. A more accurate model of a motor can be obtained by adding resistive and dynamic effects; MM's rules add resistors, capacitors and inertias to components' bare models in order to construct a model satisfying the given system behavior.

Figure 3-3 shows the actual input to MM for a system consisting of a DC motor attached to a flywheel. (The system is discussed further in Section 5.3.) The flywheel's component description is

```
(defstructure motor-flywheel
    (vars (motor-x 300)    (motor-y 200)
          (motor-len 90)   (fly-len 60)
          (battery-len 60) (horiz-space 50))
    (components
     (Motor motor
         (shape (begin-polygon motor-x motor-y)
                (go-right motor-len)
                (go-down motor-len)
                (go-left motor-len)
                (go-up motor-len)
                (end-polygon)))
     (Flywheel flywheel
        (shape (begin-polygon (+ motor-x motor-len horiz-space) motor-y)
               (go-right fly-len)
               (go-down motor-len)
               (go-left fly-len)
               (go-up motor-len)
               (end-polygon)))
    (Battery battery
       (shape (begin-polygon (- motor-x battery-len horiz-space) motor-y)
              (go-right battery-len)
              (go-down motor-len)
              (go-left battery-len)
              (go-up motor-len)
              (end-polygon))))
    (connections
     ((Motor shaft) (Flywheel shaft))
     ((Battery leads)  (Motor leads))))
```

Figure 3-3: Motor-Flywheel structure

```
(defcomponent flywheel
  (ports (shaft rotational))
  (bare-model
   (I shaft)))
```

which states that the flywheel has one port, its shaft, and is an inertance. The `components` part of the structure description provides the name of each component, its type, and its shape, which is given only so that the program can produce a nice drawing of the system. The `connections` specify which ports are connected.

The component representation obviously usurps some modeling decisions, forcing the user rather than the program to decide that a particular piece of the system exhibits a certain kind of behavior. This problem is mitigated somewhat by the fact that bare models are extremely simplified; the more interesting decisions about how to realistically augment the bare models are still left to the program.

One of MM's novel features is the ability to combine both types of structure representation, the geometric and the component. One or more contiguous line segments of a geometric part can be declared a port, and any port so declared can be connected to other ports, either of components or of other geometric parts. This allows different pieces of a complex system to be described in whichever way is most convenient. The table-bed system in Section 5.4 provides an example.

3.1.3 Behavior Representation

In addition to a description of the system's structure, input to MM must include a qualitative trace of the system's behavior. The behavior trace declares which variables of the system are to be taken as inputs and outputs, and describes in qualitative terms one possible dynamic behavior of these variables. Figure 3-4 shows the input to MM for one behavior of the U-tube. The behavior represents the situation where initially the fluid in one side is higher than the other, and over time the fluid level in one side decreases and the other increases, until eventually both sides reach the same height.

The behavior of each variable is a function from time to a magnitude. In a typical quantitative plot, the times and magnitudes would be real numbers. MM uses a qualitative description in which both time and magnitude are discretized. The representation is common in the Qualitative Reasoning community and is closest to, in fact nearly identical to, that of QSIM [19]. Time is represented as an ordered list of points, beginning with 0 and ending with the special time "point" `inf`. It is not implied that the time points are equally spaced. Magnitude is also represented using an ordered list of symbolic values called a *quantity space.* The quantity space for the U-tube behavior is given by the `qspace` form. It ranges from `minf` (negative infinity) to `inf` (positive infinity). These two values, and the value 0, are recognized specially by the program, but other symbols in the quantity space have no intrinsic meaning. In particular, it cannot be concluded from the quantity space alone that the value `-x` is the negative of the value `x`; all that the quantity space implies is that the first is

```
(behavior
   (qspace  minf bottom -x 0 x inf)
   (input nil)
   (output height-1 position ((fixed bottom))
     (0 x)
     ((0 inf) (0 x) dec)
     (inf 0))
   (output height-2 position ((fixed bottom))
     (0 -x)
     ((0 inf) (-x 0) inc)
     (inf 0))))
```

Figure 3-4: Non-oscillating behavior of the U-tube

finite, negative, and greater than `bottom`, and the second is finite and positive. The quantity space symbols are called *landmarks.*

The remaining parts of the behavior describe the variables involved and give their individual behaviors. There are no explicit inputs to the system (the effect of gravity is handled automatically by the program, so there is no need to make it an input) but there are two outputs, the heights of the fluid in each side of the tube. Each output is named; note that the names correspond to the names of the position quantities in the structural description, allowing MM to associate the two. The type of each output variable is given as well. Here both variables represent positions (displacements), but MM supports many other types; essentially all the common names for effort, flow and their integrals and derivatives in the four energy domains listed in Table 2.1. The complete list is given in Table 3.1.

Specifying the outputs of the system is, of course, a significant modeling decision. However, it is one that the user must make; the program cannot determine which quantities are of interest from the structure alone. For all the program knows, the user may be planning to drop the U-tube from a tall building. A good model for this purpose would be rather different from the "obvious" model.

Forcing the user to specify the inputs to the system is somewhat more suspect. Ideally, the program should be able to determine the relevant inputs itself.

A more subtle modeling decision is also being foisted on the user, however: the type of the variable. Although it might seem innocuous to interpret a height as a position, it is often a complicated matter to determine the type of certain inputs. Some sources of power, for instance, can be viewed as either effort or flow sources when the loads they are subject to are well within their operating range. The distinction is important, but difficult to determine *a priori;* typically the best answer can only be determined by seeing which choice makes the most sense in a model. MM does not attempt to do this sort of reasoning, leaving the decision to the user.

Name	*Effort/Flow Relationship*	*Energy Domain*
acceleration	flow derivative	translational, hydraulic
velocity	flow	translational, hydraulic
speed	flow	translational, hydraulic
position	flow integral	translational, hydraulic
pressure	effort	hydraulic
flow-rate	flow	hydraulic
volume	flow integral	hydraulic
momentum	effort integral	translational, hydraulic
force	effort	translational
voltage	effort	electrical
current	flow	electrical
torque	effort	rotational
angle	flow integral	rotational
angular-velocity	flow	rotational
angular-acceleration	flow derivative	rotational
angular-momentum	effort integral	rotational

Table 3.1: Variable types supported by MM

To continue with the format of the behavior representation: the next part of each variable description is used to associate values in the quantity space with points in the structural description. For the U-tube the form is `((fixed bottom))`, which asserts that the value `bottom` is associated with the fixed point of the position quantity as given in the structural description. This is the top edge of the U-tube's outer wall. In other words, it states that `bottom` is the value used to describe the situation where no fluid is left in the side of the U-tube.

The last part of each variable's description is the behavior trace itself. Each part of the trace specifies the time, the magnitude of the variable at that time, and optionally the sign of the variable's derivative. Intervals of time or magnitude are indicated by enclosing in parentheses two adjacent times, or two landmark values that are adjacent in the quantity space. Derivative signs are written as `inc`, `dec` and `const`, for positive, negative and zero.

Here is an interpretation for each of the three forms in `height-1`'s behavior. The behavior of `height-2` is similar.

`(0 -x)`	At time 0, The fluid height is at the value `-x`. No information about its direction of change is given.
`((0 inf) (-x 0) inc)`	Between times 0 and infinity, the height rises from `-x` to `0`.
`(inf 0)`	At time infinity, the height is at zero.

Qualitative vs. Quantitative

The behavior trace is an important part of MM's input, because it is essentially the only test for whether a model is an acceptable representation of a system. It is difficult to see how it can be omitted from an automated modeler of the sort I am describing, one concerned with the dynamics of physical systems. However, the decision to use a qualitative instead of a quantitative representation does require some justification.

A quantitative behavior trace—a numerical plot of a variable's value over time—carries considerably more information than a qualitative plot, and thus would constrain the choice of model more strongly. Not only the approximate shape, but also the actual magnitude of the model's predicated behavior would have to match the supplied behavior in order for the model to be considered acceptable. But this additional pruning power comes at a cost in complexity. First, there is the issue of noise. A simulation would be noise-free, but a series of measurements would certainly be noisy. One would have to tackle the difficult issue of determining when simulation and measurement match. The second burden on the automated modeler is the need to do parameter fitting. MM is good at determining the overall *structure* of a model, but does not attempt to determine numerical values for the model's parameters, or even the precise functional form of the elements' constitutive relations. (The next section defines MM's output more precisely.) Adding such a component to MM would be a significant undertaking with little research payoff, since the areas of function- and parameter-fitting have been and are being studied extensively in AI (e.g. BACON [20] and its ilk), statistics, and system identification [21]. MM can be viewed as a first pass. Its output, a set of reasonable model structures, can be used by parameter-fitting techniques.

An intermediate tack would be to accept a quantitative description of behavior, but mechanically convert it to a qualitative one. I also rejected this approach, on similar grounds: it would raise a host of difficulties, the problem of noise preeminent among them, which have only an indirect connection to the central research issues of this work.

3.2 Output Representation

As stated in Chapter 2, MM uses bond graphs to represent the models it generates. This section explains more precisely the actual content of the bond graphs generated by MM.

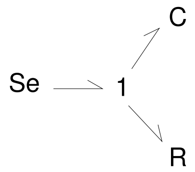
MM-generated bond graphs are qualitative, in the sense that no values are given for any of the parameters of the model; furthermore, the constitutive relations of all one-ports (resistances, capacitances and inertances) are not specified except for the assertions that they are monotonically increasing and pass through the origin.

To be more precise about the last point, let me define the relation $\mathrm{M}_0^+(f,g)$. Two differentiable functions $f(x)$ and $g(x)$ are M_0^+ iff they intersect at the origin and there is some continuous function $H(x)$ such that $H'(x) > 0$ and $f(x) = H(g(x))$. It follows from this definition that the derivatives of f and g always have the same sign. The intuition is that f and g are monotonically related: whenever one increases or decreases, the other moves in the same direction. Note that the M_0^+ relation is reflexive, symmetric and transitive.

In this thesis, the functions are always functions of time, and we use just the function name, writing e_R, say, for $e_R(t)$, the function from time to resistance R's effort. Since a resistance relates effort and flow, we can express the fact that the resistance R's constitutive relation is monotonically increasing by saying $\mathrm{M}_0^+(e_R, f_R)$. Since the derivatives of e_R and f_R always have the same sign, whenever e_R increases so does f_R, and this implies a monotonically increasing effort-flow graph. $\mathrm{M}_0^+(e_R, f_R)$ also states that there is zero flow at zero effort.

The assertions for the other two one-ports are similar. A capacitance relates an effort to the integral of a flow, for which the letter q is used by convention, so for each capacitance C in a bond graph MM asserts $\mathrm{M}_0^+(e_C, q_C)$. Dually, MM asserts $\mathrm{M}_0^+(f_I, p_I)$ for each inertance I, where p is the variable used to signify the integral of effort.

As an example, consider the bond graph



which we saw in Chapter 2. According to MM, this bond graph is equivalent to the following set of relations:

$$\begin{aligned} Se &= e_C + e_R \\ f_R &= f_C \\ &\mathrm{M}_0^+(e_C, q_C) \\ &\mathrm{M}_0^+(e_R, f_R) \\ f_C &= \dot{q_C} \end{aligned}$$

The first two equations come from the meaning of the 1-junction; the next two are the constitutive relations of the one-ports; and the last states that f_C is the time derivative of q_C.

Readers familiar with the Qualitative Reasoning literature may recognize that the above is a set of *qualitative differential equations.* In fact, the equations are already in the form required by the qualitative simulation program QSIM [19]. This is no accident; MM uses QSIM to help analyze its models. An important fact is that a bond graph using the features discussed in Chapter 2 can always be represented as a set of qualitative differential equations suitable for QSIM when the one-ports' constitutive relations are M_0^+. The only additional information necessary to perform a QSIM simulation on the above equations is the value of Se. Since sources correspond to inputs, this value must be supplied by the user in the description of the system's behavior. MM's use of QSIM is discussed in greater depth in Section 4.3.1.

3.3 The Main Loop

At its highest level, MM performs *generate-and-test.* It repeatedly proposes a candidate model, analyzes it by comparing its predicted behavior against the system behavior given as input by the user, and rejects it if the behaviors fail to match. Model proposal is done by a set of rules that key off the information given in the structural description. After a rule fires, adding one or more elements to a model, the model is passed to the analysis phase, which either declares that it matches the system behavior and thus is adequate, or tries to classify its inadequacies into one of several problem categories. An inadequate model is not rejected outright; instead, it is put on a queue of potential models for later consideration. The rules are divided into groups based on the categories used in the analysis, so when the model is dequeued for subsequent processing, only those rules relevant to its inadequacies are given a chance to run.

Figure 3-5 presents the algorithm more formally. Initially, the model queue contains only one model. If the system's structure description is entirely geometric, this initial model is empty. If the structure description is expressed using the component representation, then the bare models of the specified components are composed, and the resulting bond graph becomes the initial model. The initial model is given the problem `uninterpreted`, meaning that the inputs and outputs of the system's behavior have not yet been mapped to parts of the model (this is the first step in modeling). Each time through the loop, a model is dequeued, and each rule that is eligible to fire is invoked on a fresh copy of the model, one rule firing per copy. A rule typically modifies the copy by adding elements. Each resulting model is analyzed, and if it is faulty its problems are classified and it is enqueued at the end of the model queue. By placing the rejected model at the end of the queue, MM implements a breadth-first search through the space of models. This practically ensures that MM will always find the simplest adequate model, i.e. the model with the fewest elements. A depth-first search would run the risk of developing an elaborate, clumsy but adequate model when a simpler one would have done as well.

Rejected models are not enqueued if they are equivalent to a model already on

Construct initial model and place on queue
While queue is not empty
    Dequeue first model on queue
    If it has no problems, output it and stop
    else if no rule applies to the model, discard it
    else for each rule that applies,
        Run the rule on a copy of the model
        Analyze the resulting model
        Enqueue the model

Figure 3-5: Top-level Algorithm for MM

the queue. Two models are *equivalent* if they have the same elements in the same positions. This is a stronger notion than bond graph isomorphism, which is essentially graph isomorphism. The difference comes from the fact that the model elements must be in the same position in the structure to count as matching. For instance, consider two models for the U-tube, one of which consists solely of a capacitance in the right side of the tube, the other of which contains only a capacitance on the left side. The bond graphs are isomorphic, but the models are not equivalent because the capacitances are in different places. The check for model equivalence is not expensive, and can save considerable wasted effort on those occasions when MM generates equivalent models in different ways.

The size of MM's model search space is difficult to measure as a function of the input size, which corresponds roughly to the total number of line-segments in the structural description. The difficulty arises because MM is flexible about how it lumps regions of space. However, it is easy to show that the number of bond graph models with k elements is exponential in k, and if we make the plausible assumption that MM will generate a number of elements that is polynomial in the size of the input, we can place the size of the search space at $O(2^{p(n)})$. The important point is that it is exponential: there are a lot of models.

3.4 The Rule Interpreter

Each rule consists of two parts: a *test* and an *action*. When a rule is applied, the test is evaluated and if it succeeds, the action is evaluated. Here is a typical rule, expressed in a format that I will use throughout the thesis:

position→flow-integ/fluid (*a position is the integral of a flow*)

IF	there is a position quantity whose changing point is linked to a fluid segment
THEN	associate a 1-junction with that segment
AND	consider the quantity to be the integral of the junction's flow.

The actual Lisp code for the rule looks like this:

```
(defrule position->flow-integ/fluid (uninterpreted)
  (test (and (position-quantity ?pq)
     (let ?edge (pq-link-edge ?pq))
     (let ?seg (segment-containing ?edge))
     (eq (composition ?seg) 'fluid)))
  (action
      (interpret-position-quantity/fluid ?pq ?edge)))
```

The rule's name is followed by a list of problem categories for which it may be useful; in this case, the rule is used to propose interpretations in the model for variables in the given system behavior. The test specifies that the rule can be used on a position quantity (recall that position quantities are specified in the structural description). The remaining forms in the test extract the edge and segment corresponding to the position quantity and ensure that the segment is composed of fluid. The action part of the rule is a call to a Lisp function that implements the actions described in the above English paraphrase of the rule.

A rule's action is Lisp code that is evaluated normally, but its test is treated specially by MM. A test form is actually a pattern for a special interpreter called the rule interpreter. Rather than returning a single value, as the Lisp interpreter would, the rule interpreter returns a list of values for the variables (symbols beginning with a question mark) in the form. In the example, `position-quantity` returns a list of bindings of `?pq` to each of the external forces given in the structural description. (A binding is a pairing-up of a variable, in this case `?pq`, to something else, in this case a data structure representing a position quantity.) Other parts of the pattern may add new variables to this list, or prune some values. For instance, the fourth form in the test, beginning with `eq`, is a piece of Lisp code that retains only those values of `?seg` that have a composition of fluid. In the event that the list of values is empty, the test part fails and the rule's action is not executed.

Each time a rule is invoked, it is run on only one of its set of possible bindings. Each subsequent invocation uses the next binding in the list. The effect is that all of the rule's possible invocations actually occur, each one with a fresh copy of the current model. Each of the other rules that can run on a model are also invoked on copies of that model. Thus the overall result of running all rules on a model is a set of copies of that model, each modified by exactly one rule with one binding for that rule's variables. Each of these new models, if it is faulty, is later processed again by the rules and further augmented. Bookkeeping code assures that a rule will be run at most once on a model with a given set of bindings.

As an example, say that a rule for adding resistance to a model identified three distinct places in the given system structure where a resistance might be added. Each place would be the value of a variable in the rule's test part. (MM represents places with geometric constructs like points or polygons). The first time the rule was invoked on a model M, it would create three new models, each one having a single resistance in one of the three places. Let us call these models M.1, M.2 and M.3. If M.1 does not pass the analysis phase, it will eventually be dequeued again, and if the same rule applies, then copies of M.1 which we can call M.1.2 and M.1.3 will be generated. (The rule interpreter ensures that M.1.1 cannot happen.) After another trip through the queue, M.1.2.3, a model with all three resistances, will have been created.

This mechanism can lead to the same model being created multiple times: M.1.2 and M1.3 will both lead to the creation of M.1.2.3. This wasteful duplication is mitigated somewhat by the check for model equivalence before enqueuing, discussed above.

Chapter 4

Analysis

The purpose of MM's analysis phase is to compare candidate models against the user-supplied information about the system begin modeled. The analysis phase determines whether or not the model's predicted behavior matches the system behavior trace given as input.

The simplest and most straightforward way to accomplish this would be to simulate the model and compare the result with the given behavior. MM's analysis phase is interesting because it can do better than that: by using theorems and heuristics about the behavior of physical systems, it can weed out many models without performing a simulation, and more importantly can provide specific suggestions for modifying the model to remedy its flaws.

MM classifies model problems into one of four categories, each of which triggers a different set of rules.

`uninterpreted` The model does not contain analogues for one or more of the input-output variables specified in the behavior description of the system to be modeled. This triggers rules designed to interpret quantities.

`resistor-needed` The model lacks a resistor, but the behavior trace suggests that one must be present. As you might guess, this triggers rules that add resistors.

`order-too-low` The model's *order* (defined below) is lower than the given behavior trace warrants. This triggers rules that add energy-storage elements (capacitors and inertias) to the model.

`wrong-behavior` The model's predicted behavior disagrees with the given behavior trace. A catch-all used when none of the other categories can be shown to apply. All rules except those used for `uninterpreted` are triggered.

The analysis involves the following tests, performed in the order given, on the candidate model and the given behavior trace of the system to be modeled.

Check for uninterpreted variables. If there are variables in the given behavior trace that have not been interpreted in the model, the model is labeled `uninterpreted`. This step is straightforward and is not described in more detail.

Determine whether resistance is necessary. If the model contains no resistors, and if the behavior trace suggests that a resistor is necessary, the model is labeled `resistor-needed`. See Section 4.2 for more detail.

Determine order. The model's order is determined and the minimum order needed to produce the behavior trace is estimated. If the model's order is too low, it is labeled `order-too-low`. This step is explained more fully in Section 4.1.

Simulate. If the model passes all of the previous tests, a qualitative simulation of the model is carried out and the results compared to the behavior trace. If they disagree, the model is labeled with `wrong-behavior`. See Section 4.3.

Simulation is a last resort for two reasons. First, the other tests allow a more specific determination of the model's problems. Second, the tests are in some ways more powerful than simulation; they can reject models that would pass the simulation test. An example of this second point can be found in Section 5.1.

4.1 Order Determination

A fundamental assumption of the kind of models used in this thesis is that at any point in time, a *state*—a finite set of values—suffices to describe all that is relevant about the model. The entire future and past of the model's behavior can be predicted from its state.

The *order* of a model is the number of values needed to describe its state. Order is a natural measure of the complexity of a model. It is also closely connected with the makeup of the model itself, when the model is expressed in an energy-based modeling framework. It turns out that the order of a model is equal to the number of independent energy-storage elements (capacitors and inertias) it contains. An element is independent if its behavior is not completely determined by other elements in the model.

Determining a model's order is not difficult, and is described in the next section. A more interesting computation is determining from a behavior what the minimum possible order of a model generating that behavior could be. If the behavior trace is quantitative and the model is assumed linear, then there is a suite of techniques for determining the order in general [21]. For qualitative plots and qualitative models, I have been able to make a precise determination only for low-order systems. Given a qualitative behavior trace, it is usually possible to determine whether the model could be zeroth-order (that is, containing only algebraic, not differential, equations), first-order, or of second or higher order. The distinctions are crude, but crucial. For the purposes of naive physical reasoning about dynamics, they are probably all that is required; all the lay concepts of dynamic behavior, such as delay, oscillation and overshoot, are exhibited by second-order systems. Zeroth-order systems can be viewed as fundamentally different from dynamic systems—they are in general much simpler to understand and analyze, requiring only algebra, not calculus—so it is advantageous to use them when possible. It is probably impossible to extract information about higher-order behavior from a purely qualitative plot of a system's time behavior, so modeling programs like MM that reason from qualitative data alone may be confined to low-order models. Finally, models of order two or less are thoroughly understood by both mathematicians and engineers, so a modeling program that interacts with humans should place a premium on constructing such models.

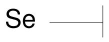
The next sections explain how to determine the order of a bond graph model and of a qualitative behavior trace.

4.1.1 Causality Assignment

Only the energy-storage elements—capacitors and inertias—contribute to the order of a model. In general, each energy-storage element contributes one state variable. However, it is possible (and not uncommon) to have an energy-storage element that is dependent on other parts of the model. In other words, the putative state variable of the element can be rewritten in terms of the other state variables.

The order of a bond graph—the minimum number of state variables needed to capture the model's behavior—is the number of *independent* energy storage elements. The method for determining the order is called *causality assignment* and is well-known in the bond graph literature (see, e.g. [18]). It involves labeling each bond in the bond graph with a *causal stroke* which describes from which direction effort and flow are imposed on the bond. A bond with a causal stroke looks like this:

The stroke on the right side of the bond indicates that effort is imposed from the left, flow from the right. The idea is best explained by example. Consider the bond graph fragment



Since an effort source determines the effort along its bond, the causal stroke is placed to reflect that fact. A flow source, which determines the flow along its bond, always has the causal stroke placed near it:

Now consider the junctions. In a 1-junction (common flow), flow along any one bond determines the flow along all bonds, and at least one bond must determine the junction's flow. It follows that exactly one of the bonds impinging on a 1-junction imposes flow on the junction, i.e. has its causal stroke on the side away from the junction. All other bonds have causal strokes facing the junction. The 0-junction is,

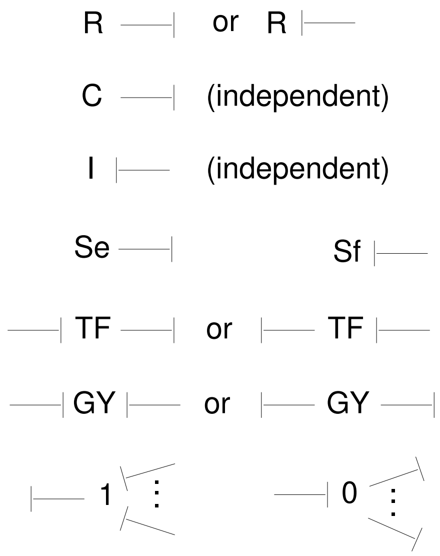


Figure 4-1: Constraints on causal strokes

as always, the dual: all but one bond must have its causal stroke facing away from the junction.

The transformer and gyrator also place constraints on causal strokes, which follow from their constitutive relations. For example, the effort on one side of the transformer determines the effort on the other side, so if one of a transformer's bond has its causal stroke adjacent to the transformer, the other bond must have its stroke facing away from the transformer. Analysis of the gyrator is similar.

The case of energy-storage elements is a little different. A capacitor can either impose effort (causal stroke away from the element) or have effort imposed upon it; but the two situations are not symmetric. A capacitor with effort imposed on it is dependent and its state variable can be rewritten in terms of the other state variables of the model. The inertia, as usual, is dual: it is dependent when flow is imposed on it.

These causal constraints are summarized graphically in Figure 4-1. A resistor with a strictly monotonic effort-flow relation imposes no constraints.

To determine the order of a bond graph model, it is first necessary to label every bond with a causal stroke. This can be done by a straightforward algorithm that repeatedly propagates all causality constraints imposed by the elements, then assigns causality to some remaining bond arbitrarily, backtracking when contradictions are encountered. Initially all energy-storage elements are made to appear independent, and this is changed only when forced by a contradiction. When all bonds have been consistently labeled, the number of independent energy storage elements provides an upper bound on the model order. Additional causality-assignment steps (described in [28], but not implemented in MM) can be used to ferret out additional dependencies.

4.1.2 Order from Behavior

Two tests are done on the qualitative behavior trace given to MM to see if it could be generated by a low-order model. The first determines whether differential equations are required to produce the behavior; that is, whether a zeroth-order model will suffice. The second test is a heuristic that can often determine whether the behavior can be captured by a first-order model.

Zero-order Models

Before discussing the zeroth-order test, it is worth understanding just what it means for a model to be zeroth-order. A zeroth-order model has no differential equations, hence no state, hence no memory. Does this mean there is no change? The question is subtler than it seems.

Of course, the system will change if its input is changing. If you rigidly move a block of wood from place to place on a table top, then the position of the block changes even though an accurate model of the system—a flow source connected to an inertia—is zeroth-order.

But now consider a block at rest in the middle of a table, with no movement. Is the system changing? No—unless you take as an output the time integral of the block's distance from the table's edge. This output is a linearly increasing function of time.

The moral of the story is that *change is not a mathematical notion, but a physical one.* It is impossible to tell merely from the plots of a system's inputs and outputs whether it exhibits change or state, that is, whether it can be described by a zeroth-order model. A fundamental piece of information is missing: the types of the input-output variables. This is one very important reason why these types must be given by the user.

We are now in a position to define zeroth-order more precisely. We will say that *adjusting* a variable is integrating or differentiating it sufficiently to make it an effort or flow. For example, to adjust a position, differentiate it once to make it a velocity, which is a flow. Two variables exhibit a *zeroth-order relationship* if, when adjusted, they are functionally related; in other words, a plot of the time behavior of one against that of the other yields a function. The adjustment takes care of the problem raised above, where variables of certain types can give rise to seeming change when there is none.

A qualitative behavior trace—a set of input and output variables, their types, and a qualitative time behavior for each variable—can be implemented by a zeroth-order model if each pair of input and output variables is in a zeroth-order relationship.

The Zeroth-order Test

MM's zeroth-order test could mimic the above definitions, first differentiating or integrating each input-output variable as necessary and then plotting the adjusted variables against each other to see if the relationship is a function. (This was, in fact, the original implementation.) But there are problems. Differentiating a qualitative behavior trace loses a great deal of information about the function; taking a second derivative is even worse. Treating qualitative differentiation as a function from behaviors to behaviors is thus unwise. A better approach is to treat it as a constraint between two behaviors. The property of having a functional relationship can also be represented as a constraint. The zeroth-order test can then be implemented as a test to see if a set of constraints can be satisfied.

Since MM's bond-graph elements all have M_0^+ characteristics, as discussed in Section 3.2, it is overwhelmingly likely that bond graphs constructed by MM will implement functions are M_0^+ or M_0^-. (The difference between two M_0^+ functions can be arbitrary, but MM is not likely to construct a model that contains such a difference.) Hence the constraint that two variables x and y have a functional relationship can be expressed as $\mathrm{M}_0^+(x, y)$ or $\mathrm{M}_0^-(x, y)$. Say that the two variables that we wish to test for having a zeroth-order relationship are a position p and a force F. The position needs to be differentiated once to obtain a flow; the force is already adjusted, since it is an effort. There are two sets of constraints:

$$\frac{dp}{dt} = \dot{p}$$
$$\mathrm{M}_0^+(\dot{p}, F)$$

and

$$\frac{dp}{dt} = \dot{p}$$
$$\mathrm{M}_0^-(\dot{p}, F)$$

In general, the constraint sets can always be written to contain only differentiation, M_0^+ and M_0^- constraints.

These constraints are suitable for use by QSIM, a qualitative simulation program. MM effectively simulates each of the two sets of constraints and sees if any of the simulated behaviors agree with the given behavior for the two variables in question. Actually, MM does something more efficient: during the simulation, it prunes behaviors that diverge from the given behavior. The modified QSIM algorithm that accomplishes this is detailed in Section 4.3.

MM performs the zeroth-order test for each pairing of an input with an output variable. If all pairings are zeroth-order, then MM concludes that the entire behavior trace could be generated by a zeroth-order model.

The First-order Test

To determine whether a behavior trace could be captured by a first-order model or whether it requires a model of order two or greater, MM employs the following heuristic: *The outputs of a first-order system with constant or monotonic inputs will not oscillate (i.e. will not change direction).* This includes the case of no input (a constant zero input). As with the zeroth-order test, the inputs and outputs must be adjusted to be efforts or flows by integration or differentiation.

For a first-order model with no input or constant input, this is a fact. It can be easily shown by considering the phase space of the model and recalling that phase-space trajectories cannot intersect (except to form a cycle). The phase space of a first-order model is one-dimensional, so no change in direction of the model's state is possible without intersection. Hence the output variables must all be monotonic. When we consider models with monotonic but not constant input, the principle is merely a heuristic. It is justified by the fact that many first-order models exhibit *tracking* behavior: they follow their input. Thus a monotonic input will engender a monotonic output.

While the zeroth-order test works for any input, the first-order test can only be applied if all of the system's inputs are constant or monotonic. If this is the case, and if all the outputs are constant or monotonic, then MM concludes that a first-order model could be responsible for the given behavior trace.

As with the zeroth-order test, MM constructs sets of constraints and uses qualitative simulation to implement the test. DERIV constraints are used to adjust the variables, and two new unary constraints, which I call NSM^+ and NSM^-, are used to express the constraint on the adjusted output variable. A function is NSM^+ if it is Non-Strictly Monotonically increasing; that is, if its derivative is never negative. Similarly, a function is NSM^- if its derivative is never positive.

4.2 Resistance Analysis

MM also uses a heuristic to determine whether the behavior trace suggests that the model requires a resistor. Of course, all real physical systems have resistance—that is, all dissipate energy—but sometimes one can neglect losses and obtain a simpler model.

MM decides that a behavior trace must be modeled with a resistor when the behavior has unchanging inputs (after adjustment) and the system moves to an equilibrium point; that is, when there is a time after which no (adjusted) outputs change.

The intuition behind MM's heuristic is that all systems with losses eventually "wind down." The intuition is formalized and proved in Duffin's theorem [8], which says (roughly) that if given no input or a constant input, any system consisting of passive components and at least one resistor will have constant efforts and flows in the steady state. In other words, the system cannot oscillate indefinitely.

Duffin's theorem guarantees that if a system has resistance, it will reach a constant steady state. The converse, unfortunately, is not true. It is possible for a lossless system to move from some non-equilibrium state to an equilibrium point, a point where none of the system's parameters are changing. But a more typical behavior for a lossless system is to oscillate or move indefinitely if perturbed. So MM's resistance test, though a heuristic, is a reasonable one.

4.3 Simulation

When a model passes the above tests—when its order is high enough and it contains a resistor if necessary—or when the tests are not applicable, MM resorts to comparing the simulated behavior of the model with the given behavior. Before doing so, however, MM conducts two preliminary tests.

First, MM compares the candidate model to the models it has previously created to see if they are behaviorally equivalent. The test for behavioral equivalence is not the same as the equivalence test discussed in Section 3.3. That test was stronger, because it not only required bond graph isomorphism but also insisted that corresponding elements be at the same place in the system structure. The test for behavioral equivalence requires only bond graph isomorphism. If a behaviorally equivalent model is found, its analysis result is used, saving a simulation. (The equivalence test could be done right at the beginning of the analysis phase, but the order and resistance tests are so fast that the time saved would be negligible.)

If no behaviorally equivalent model exists, MM confirms that the model asserts some relationship between the inputs and outputs. It does this by ensuring there is a path in the *interaction graph* between each output and some input. The interaction graph is simply the graph arising from the connectivity of the model's equations. Each node in the graph is a variable, and an arc represents the fact that some equation or constraint relates the variables. MM performs this step because the qualitative simulator will happily simulate a model in which the outputs are unconstrained. Such a model predicts every behavior, and so agrees with any given behavior. Ensuring that each output is constrained eliminates this problem in practice.

Unlike quantitative simulation, qualitative simulation is inherently ambiguous: a model can give rise to more than one behavior even when begun from the same initial conditions. What is worse, some physically impossible behaviors may be generated by the simulation, although every physically possible behavior will also be represented.

The first of these problems, ambiguity, can result in intractable computation. It is possible to generate a very large number of behaviors from even a simple model. Early attempts to simulate a model and then compare the result with the user-supplied behavior trace often failed because the simulation could not proceed far enough due to resource limitations. The problem was solved by modifying QSIM, the qualitative simulation program, to interleave simulation and pruning of incorrect behaviors. The modification does not alter the inherent intractability of QSIM, but for practical purposes it makes it vastly more efficient.

The second problem with qualitative simulation, the so-called spurious behavior problem, is not so easy to solve. One partial solution is to use additional equations which are mathematically redundant, but which nonetheless provide additional information to the simulator. For instance, a qualitative simulation of an undamped harmonic oscillator will give rise to the behavior in which oscillations have the same amplitude—the correct behavior—as well as behaviors where the amplitude increases, decreases, or varies. By incorporating the equation asserting conservation of energy into the model, only the correct behavior remains. This technique is not always successful, however, and there is no general solution to the spurious behavior problem. MM simply neglects the difficulty: if *any* of the model's simulated behaviors agrees with the given behavior, the model is deemed correct.

In the remainder of this section, I explain the simulation test in more detail. First I give an overview of the QSIM program. Then I describe QSIM-CHECK, a version of QSIM incorporating modifications that improve its performance for the task of checking a model against a given behavior.

4.3.1 QSIM

The input to QSIM is a set of *qualitative constraints* that hold between a set of parameters, which are functions of time. These constraints are summarized in Table 4.1. QSIM also possesses constraints that are weaker than M_0^+ and M_0^- in that they do not assert that the variables agree at zero.

Section 3.2 explained briefly how a bond graph model can be translated into a set of these constraints. Table 4.2 lists the constraints generated for each bond-graph element. The actual constraints for junctions depend on the orientation of power half-arrows on the bonds; the examples in the table assume that bonds 1 and 2 point into the junction and bond 3 points out. For junctions with more than three bonds, MM creates additional parameters and uses multiple ADD constraints, and possibly MINUS constraints as well.

Information about interpreted input-output variables is also turned into constraints. Input constraints are associated with effort and flow sources; output variables are generally associated with junctions. DERIV constraints may be associated with the variables in order to do adjustment (see Section 4.1.2). Output variables are not otherwise constrained. The INDEPENDENT constraint is used for constant inputs. The set of constraints is passed through an algebraic simplifier which eliminates equalities and redundant constraints.

In QSIM, each parameter has its own quantity space (MM simplifies this, insisting on a single quantity space for all parameters). As discussed in Section 3.1.3, a quantity space is a linearly ordered set of symbolic values called landmarks. A *qualitative magnitude* is either a landmark, representing a single (but unknown) numerical value, or it is an ordered pair of landmarks, representing some numerical value between the

DERIV(x, y)	$\frac{dx}{dt} = y$
ADD(x, y, z)	$x + y = z$
MULT(x, y, z)	$xy = z$
MINUS(x, y)	$x = -y$
INDEPENDENT(x)	x is constant
$\mathrm{M}_0^+(x, y)$	see Section 3.2
$\mathrm{M}_0^-(x, y)$	see Section 3.2

Table 4.1: QSIM's constraints and their meanings

R	$\mathrm{M}_0^+(e, f)$	
C	DERIV(q, f),	$\mathrm{M}_0^+(q, e)$
I	DERIV(p, e),	$\mathrm{M}_0^+(p, f)$
1	$f_1 = f_2 = f_3$,	ADD(e_1, e_2, e_3)
0	$e_1 = e_2 = e_3$,	ADD(f_1, f_2, f_3)
TF	$\mathrm{M}_0^+(f_1, f_2)$,	$\mathrm{M}_0^+(e_1, e_2)$
GY	$\mathrm{M}_0^+(f_1, e_2)$,	$\mathrm{M}_0^+(e_1, f_2)$

Table 4.2: QSIM constraints generated for bond-graph elements. The variables e, f, e_1, f_1, etc. refer to the effort and flow on the bond(s) of the element.

two. A *qualitative value* for a parameter consists of a qualitative magnitude for that parameter, and information about the sign of the parameter's derivative—whether the parameter is increasing, decreasing, or constant. A parameter with a quantity space consisting of three values a, b and c could take on any one of fifteen qualitative values: each of the five qualitative magnitudes a, (a, b), b, (b, c) and c could be coupled with any of the three derivative signs. Since derivatives are an integral (no pun intended) part of the representation, it follows that QSIM assumes that all parameters are differentiable, and hence continuous.

Just as the notion of *state* is crucial to numerical simulators, so the notion of *qualitative state* is central to QSIM. In a numerical simulator, a state is a vector of real numbers, one for each state variable of the model. A state of a QSIM model is a vector of qualitative values, one for each parameter.

A QSIM simulation begins with an initial state, the state of the system at a point in time arbitrarily called T0. Since QSIM has no numerical information, it cannot determine the actual duration of events in the simulation, so it simply labels time with consecutive integers. This does not imply that the durations between time-points are equal.

QSIM now tries to determine what could happen "next"; in this case, "next" means over the interval of time from T0 to T1. QSIM determines all the possible *successor states* of the initial state; that is, all the states the model could be in immediately after T0. To do this, QSIM first goes through each parameter, generating all the possible qualitative values that might immediately follow the current one in time, assuming that the parameter represents a continuous function. For instance, imagine some parameter has the qualitative value $\langle a, inc\rangle$ at T0, meaning that its qualitative magnitude is a and it is increasing. Then its only possible value for the interval between T0 and T1 is $\langle(a, b), inc\rangle$, because the positive derivative means the parameter's value must increase.

QSIM is designed so that no parameter moves more than one position along its quantity space in the course of any one time interval. If a parameter moves between a and c, it will do so over two intervals; in the first it will move to b.

After it enumerates the possible successors of each parameter, QSIM combines these values into a set of potential successor states by taking their cross-product, then filters these states using the constraints supplied with the model to arrive at a final set of successor states. (Actually, it interleaves filtering with state construction for efficiency.) For instance, if some potential state has a value of $\langle a, inc\rangle$ for parameter x and a value of $\langle b, dec\rangle$ for parameter y, and one of the constraints is $\mathrm{M}_0^+(x, y)$, then the state is not a possible successor state because the M_0^+ constraint implies that the parameters' derivatives must have the same sign.

After discarding states in this way, QSIM arrives at a set of successors of the initial state. Ideally there is just one successor, but because of ambiguity there may be several. The process is repeated for each of these states, but with a slight difference: now QSIM tries to determine what will happen when moving from the interval of time between T0 and T1, to the time point T1. The transitions for the individual parameters are slightly different when moving from an interval to a point.

Readers familiar with QSIM will note that I have not described QSIM's "landmark generation" feature. This feature is not used by MM.

QSIM can be viewed as conducting a breadth-first search through a graph whose nodes are qualitative states and whose arcs are determined by the successor relationship between states. QSIM maintains a queue of states which at the outset contains only the initial state. It repeatedly removes the next state from the queue, finds its successors, and places them at the end of the queue. When it is finished, QSIM has constructed a tree, or more precisely a DAG (directed acyclic graph) of states, whose root is the initial state. Every path from the initial state to a leaf state is a possible behavior of the model (modulo the spurious behavior problem).

The simulation can terminate in one of two ways. First, QSIM checks each state it generates to see if all its derivatives are zero (indicating that all successor states will be identical to this one) or if the state is identical to one that occurred previously in this behavior (indicating a cycle). In either case, QSIM no longer generates successors for that state. If all states meet one of these conditions, the queue empties and the simulation stops.

Because QSIM can take a great deal of time and generate a very large number of states (exponential in the number of parameters and the size of the quantity spaces), the algorithm can also be made to terminate after a fixed number of states have been generated.

4.3.2 QSIM-CHECK

Although QSIM's most well-known limitation is the spurious behavior problem, it has two others which are of concern to the implementation of MM. First is its exponential behavior. If the threshold on the number of states generated is set to one hundred, running QSIM on a fairly simple second-order model of an oscillator will take over a minute and will result in twenty or so behaviors, none of which exhibit more than a couple of oscillations. The program's performance degrades exponentially, so doubling the state threshold will double the time taken but will not produce behaviors that have significantly more oscillations. If MM were to use QSIM directly, and the user were to supply a system behavior that lasted for five oscillations, MM's time and space requirements would be prohibitive.

QSIM's second problem is that it works only for models with constant or no input. There is no easy way to specify a changing input to a QSIM model.

QSIM-CHECK solves both of these problems. It has proven to be an efficient method for testing a model against a behavior with arbitrary inputs.

The QSIM-CHECK algorithm is a modification of QSIM that performs simulation and comparison with a given behavior in tandem. Like QSIM, QSIM-CHECK takes as input a set of parameters and constraints between them, and an initial state. It also takes a *history* as input. A history is just what I have been calling a behavior throughout this thesis—it is a list of parameters and how they change qualitatively over time—but the change in terminology will make this section clearer. The history may include both input and output parameters. The parameters in the history should include some, but need not include all, of the model's parameters.

QSIM-CHECK behaves much like QSIM. The crucial difference is that as each successor state is generated, QSIM-CHECK determines if it matches the history. If not, the state is discarded. QSIM-CHECK stops immediately and indicates success if the state matches the end of the history. The result is a qualitative simulation where the only behaviors being simulated are those that could ultimately fully match the history.

QSIM-CHECK incorporates a second modification to QSIM as well: it performs best-first instead of breadth-first search. The goodness heuristic is explained following the discussion of the behavior comparison step.

Pruning States Using Behavior

QSIM-CHECK compares each newly constructed state against the history and discards it if it fails to match. This extra pruning step is not as straightforward as it might appear at first glance. The difficulty is that there might be additional states in the simulation that are not represented in the history.

Say that part of the history is the behavior of some parameter P, which begins at zero and increases steadily until it reaches some value:

Behavior 1

T0	a	const
(T0 T1)	(a b)	inc
T1	b	const

A simulated behavior that is identical to this will of course match it. But the following behavior should match as well:

Behavior 2

T0	a	const
(T0 T1)	(a b)	inc
T1	(a b)	inc
(T1 T2)	(a b)	inc
T2	b	const

Here, parameter P is also beginning at zero and increasing; the only difference is in the time scale. Such a behavior for P might result because other parameters of the model changed between T0 and T2.

In the discussion of QSIM it was mentioned that no qualitative magnitude ever spans more than two landmarks. However, the user who describes a behavior to MM is not restricted by this. So Behavior 1 would also be a valid input if the quantity space were $a, a2, b$. In that case, another matching behavior is

	Behavior 3	
T0	a	const
(T0 T1)	(a a2)	inc
T1	a2	inc
(T1 T2)	(a2 b)	inc
T2	b	const

In this behavior, P is again rising from a to b, but the rise is shown in two stages.

So far, we have been assuming that Behavior 1 is the history and the others are the simulation results. But the opposite is also possible: the history might be Behavior 2, for instance.

It should be clear from the examples that to compare behaviors, it does not suffice to check that magnitudes and derivatives are equal at equal times. Instead, QSIM-CHECK distinguishes between the *simulation time* and the *history time* of a state. Simulation time is the time that QSIM assigns to a state. A state's history time is the time in the history in which the state belongs. For example, in Behaviors 2 and 3, all the states with derivative `inc` have a history time of (T0 T1), because they are part of the single `inc` state in the history (Behavior 1).

Given a state, QSIM-CHECK has two jobs: first, determine the history time of the state; then determine whether the state matches the history at that time. In fact, a single state might match several times in the history, so each state has a set of history times.

The sole history time for the first state is T0. To determine any other state S's history times, QSIM-CHECK first looks at the history times of the S's predecessor, P. Any one of those history times is potentially a history time of S, because it is possible that two adjacent states share the same history time. The other potential history times for S are the times immediately following P's history times, since the transition from P to S might represent a transition in the history. For each potential history time, S is compared with the history at that time, and only those history times for which S matches the history are retained.

If a state represents an interval of time and one of its candidate history times is a point, we can eliminate that history time from consideration, because an interval state cannot represent a point in time. The other three combinations are possible, however; in particular, a point state can be part of an interval history time. For these cases, it is necessary to compare the values of the parameters in the state and the history. Both the derivative and magnitude of the qualitative values are compared. The derivatives must always agree exactly. If the time is a point, then equality of magnitudes is the right comparison. But if the time is an interval and the history's magnitude is an interval as well, then it is too strict to insist that the simulation's magnitude is the same interval, because the simulation might not yet have traversed the entire interval. Consider the third example above: there, the first simulation interval only reaches the value $a2$, even though the corresponding history interval ranges from a to b. The appropriate test is not equality, but subset: if the simulation's magnitude is a subset of the history's, then the state matches. The subset test is obviously necessary. It is also sufficient, because all behaviors are continuous and all derivatives and time-point magnitudes match exactly. Hence it is not possible for a sequence of simulation states to cover only a portion of a history state's interval: the equality checks at the interval's endpoints will ensure that the complete interval is traversed.

Best-First Search

QSIM-CHECK's second improvement on QSIM is to use best-first instead of depth-first search. If a successor state was not pruned by the check against the history, it will have been assigned one or more history times. Instead of placing the state at the end of the queue, it is placed just before the first state on the queue whose latest history time is less than its own latest history time. In other words, the queue is kept sorted in history-time order, from latest to earliest. The effect is that QSIM-CHECK follows the most promising, i.e. the most quickly advancing, behaviors first. In practice, this is significantly faster than breadth-first search. It is possible that the best-first algorithm will get bogged down on a behavior that, though more advanced than any other, does not make progress quickly, or at all. (That is, its history time does not advance.) However, as long as QSIM's landmark-generation feature is turned off, there is no danger of an infinite loop. This is prevented by QSIM's cycle check, which discards a state if it is equivalent to an earlier one in the same behavior. Since there are only a finite number of landmarks, there are only a finite number of qualitative values, hence a finite number of states, so eventually every behavior will cycle if it is not discarded for some other reason. Thus QSIM-CHECK is guaranteed to eventually either produce a matching behavior, or terminate with the conclusion that no behaviors match the history.

Additional Improvements

Further improvements to QSIM-CHECK are possible. The first step of QSIM-CHECK, as with QSIM, is the creation of all possible initial states consistent with the history's parameters. This is at present the slowest part of the algorithm in cases where the number of parameters in the history is a small subset of the total number of parameters in the model. One improvement would involve creating one initial state at a time and testing it before creating the next. Another, more far-reaching change would be to implement a lazy evaluation scheme, where a state's parameters would be determined only when necessary. This might in some cases avoid the creation of very large numbers of states, which is the current bottleneck in the program.

Related Work

Behavior-constrained qualitative simulation is closely related to the problem of *measurement interpretation,* in which one is given a set of measurements over time and a model and the goal is to determine what model states the measurements correspond to. If there is no sequence of model states corresponding to the given measurements, one can conclude that the model cannot predict the measurements. This is just the test that QSIM-CHECK implements.

Ken Forbus first raised the measurement interpretation problem and provided a solution for his Qualitative Process (QP) Theory representation [13]. Dennis DeCoste extended and improved Forbus's work in his DATMI program [5]. DATMI deals with a larger problem than QSIM-CHECK, that of interpreting possibly noisy or incomplete numeric data. The early stages of the program are devoted to transforming this data into a qualitative history like that used by QSIM-CHECK, and other parts of the program try to deal with issues like sensor error that I have finessed. At their core, however, DATMI and QSIM-CHECK perform the same task: both attempt to find paths of states that match a qualitative history. DATMI differs from QSIM-CHECK in that it first constructs an *envisionment* of the model, a graph containing all possible model states where an arc connects two states if one could be a successor of the other. Once it has constructed the envisionment, DATMI is fairly efficient in finding paths through it that correspond to interpretations of the measurements. However, the number of envisionment states could be exponential in the number of model parameters. QSIM-CHECK takes an incremental approach, creating states only when necessary.

DATMI's approach makes sense if the same model will be used to interpret multiple measurements, for then the cost of constructing the envisionment is amortized over multiple runs. For the applications in this thesis, however, a model is tested only once, so the incremental approach would seem to be preferable.

Another, somewhat more subtle point concerns the meaning of QSIM's M^+ and M^- constraints (and the more specific M_0^+ and M_0^- constraints used in this thesis), versus the qualitative proportionalities used in QP theory and assumed in DATMI. A QSIM "M" constraint asserts the existence of a function relating the two parameters; a qualitative proportionality does not. This permits QSIM to draw an inference that a QP reasoner cannot. Consider two parameters P and Q related by one of the "M' constraints, and say that at some time point P takes on the value a and Q takes on the value b. Now since they are functionally related, we can conclude that, at any time, whenever P is a then Q must be b, and vice versa. The values a and b are called *corresponding values.*

QSIM and QSIM-CHECK incorporate this inference by accumulating sets of corresponding values as a simulation proceeds. If at time T1 a state S is generated where P is a and Q is b, then S's successor's record this fact, and any states that arise from S in which P is a but Q is not b, or Q is b but P is not a, can be pruned. Note that this implies that the parameter values of a state do not completely determine the information that state contains; it also matters which states have preceded it in the simulation. DATMI does not seem to be equipped to deal with this extra information, and it is not clear how much it would cost in efficiency to add it. By contrast, QSIM-

CHECK, because it is built on top of QSIM, automatically uses corresponding-values information.

Chapter 5

Examples

This chapter describes how MM processes four systems: a U-tube, a fluid piston, an electric motor attached to a flywheel, and a table-bed positioning system.

Recall that MM processes models in breadth-first order in order to generate simpler models first. However, because of this, it is sometimes confusing to watch its progress. MM may modify a model in a way that brings it closer to being correct, but then it defers the model, placing it on the back of its processing queue, and proceeds to work on what may be a less satisfactory model. To spare the reader this confusion, I present the examples in a more natural order, in which a model and its successor are presented consecutively. It should be understood that wherever I have written "next, MM does this..." I really mean "the next time MM processes this model."

5.1 The U-tube

Figure 3-1 presented the structural description of the U-tube system; Figure 5-1 shows a rendering of that description. Figure 3-4 displayed a behavior trace for the system in which the fluid levels in the two sides of the tube begin at different points, and over time one increases and the other decreases until they are equal.

MM begins by constructing a model that contains no elements and is labeled with the problem `uninterpreted`. MM's interpretation rules are triggered, one of which is:

position→flow-integ/fluid (*a position is the integral of a flow*)

IF	there is a position quantity whose changing point is linked to a fluid segment
THEN	associate a 1-junction with that segment
AND	consider the quantity to be the integral of the junction's flow.

After MM interprets both fluid heights with this rule, the model looks like Figure 5-2. The model no longer has the problem `uninterpreted`. In order to continue its analysis, MM must form a connected bond graph from the two isolated 1-junctions

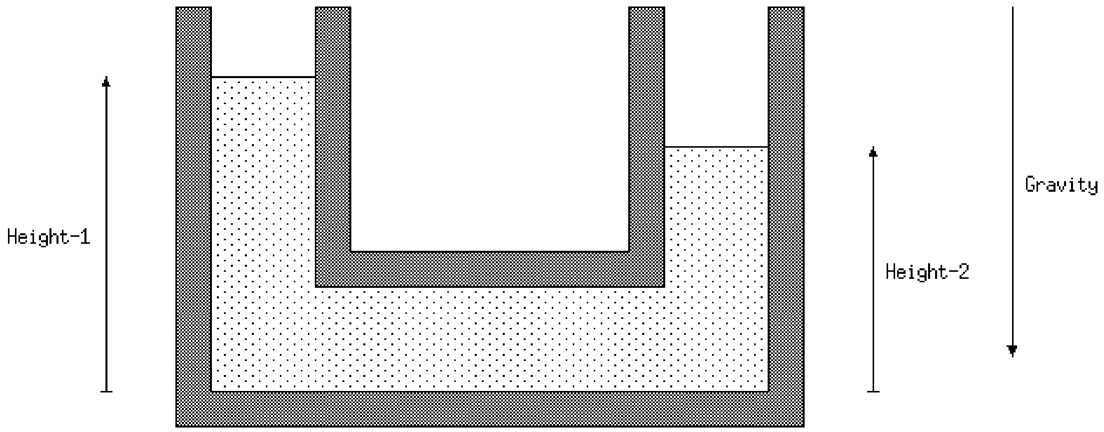


Figure 5-1: Structure of U-tube

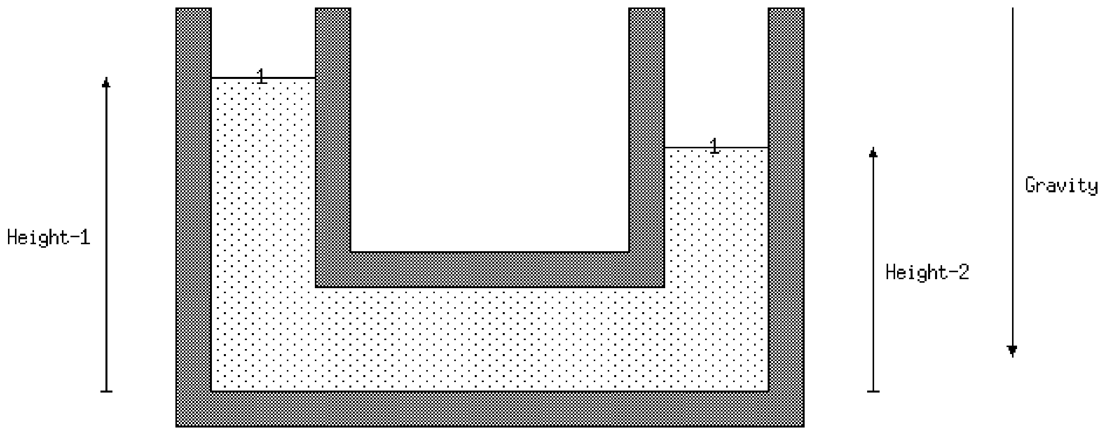


Figure 5-2: U-tube model with interpreted outputs

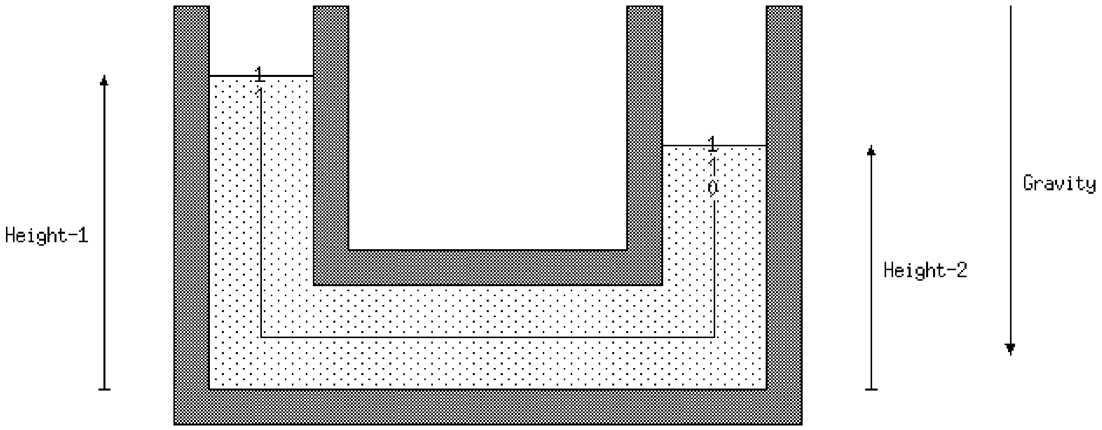


Figure 5-3: The previous U-tube model, connected

Figure 5-4: Two conventions for fluid flow

of the model. This connection process is not straightforward in general and will be discussed fully in the context of a more interesting example, in Section 5.2. Roughly, MM tries to find a path through the structural description from one group of connected nodes to another, preferring to stay in segments of the same composition. In this case, the obvious all-fluid path is found.

However, there is an important twist, as Figure 5-3 shows: a 0-junction has been inserted, inserting a negation into the equations for the model. A direct connection between the two 1-junctions would result in the equation $f_1 = f_2$, where the f_i are the two flows. But the model of Figure 5-3 has the equation $f_1 = -f_2$. The reason is that MM adopts a particular convention regarding the measurement of fluid flow.

Consider a slug of fluid in a pipe or container (see Figure 5-4). There are two views of this slug, corresponding to two conventions for representing its velocity. In one, we treat the slug as a unit, and we speak of the velocity of the entire slug. If we decide that a rightward velocity is positive, then the velocity at each edge of the slug is positive when to the right, as the solid arrows in the figure indicate.

However, when we are interested in the individual velocities of each edge of fluid, it is natural to represent a positive velocity as one that expands the fluid region. This is the U-tube case: the two outputs are the velocities of the fluid edges, and we think of positive as "up." For the slug in the figure, the left edge would represent positive as to the left, and the right edge, to the right. The dashed arrows show this case.

MM is aware of these two conventions and tries to pick the appropriate one. When it creates a 1-junction that represents fluid flow, it determines whether that junction represents the changing velocity of a fluid edge, or the flow of fluid through a region of space, and chooses the sign convention appropriately. In the U-tube case, MM recognizes that the former convention applies, and so negates the sign of one of the flows. The result is a model that, though inaccurate, at least gets the sign of flow correct for the two sides of the U-tube.

5.1.1 Analysis

MM now analyzes the model. First, MM observes that the given behavior requires a resistor, since the system moves to an equilibrium state; and because the model has no resistor, it is labeled with the problem `needs-resistor`. MM also determines that the given behavior cannot be zeroth-order, because the system has a constant zero input (recall that a behavior with no input is translated to one with a constant zero input), but a changing adjusted output: the fluid velocity changes from its initial value of zero to some non-zero value (positive on one side of the tube, negative on the other). Since the model is zeroth-order (it has no energy-storage elements), it is also labeled with the problem `order-too-low`.

The simulation step is not carried out, because the model failed the other tests. However, it is worth observing what would have happened if there had been a simulation. The model's only useful constraint is that $f_1 = -f_2$, that is, the fluid velocities in the two sides of the U-tube are opposite. The other constraint is that the input is zero and constant, but this plays no role in the simulation, since the input is not connected to any outputs. A QSIM-CHECK simulation would have confirmed that this constraint is consistent with the given behavior. Indeed, it would be consistent with *any* behavior of the U-tube, given the assumption (implicit in our modeling technique) that the fluid is incompressible. For then any movement on one side of the tube would be mirrored instantaneously and exactly by a movement on the other side.

Although the model that asserts only $f_1 = -f_2$ is consistent with all behaviors, our intuition is that it is not a good model. The reason is that it fails to account for the reasons that the fluid moved. If we were observing a chunk of matter traveling through the void of space at a constant velocity, then we might not be able to say anything much about its behavior. But in the case of the U-tube, it behooves us to explain how the fluid moves in the absence of any apparent input.

This explanation for the inadequacy of the $f_1 = -f_2$ model also explains why the order-determination heuristics are in a sense more powerful than the simulation test: they take both input and output into account, regardless of any established connection between the two. Hence the zeroth-order test recognizes that the input is constant while the output is not, while the simulation effectively disregards the input because the model fails to include it.

5.1.2 Adding Resistance

Having concluded that the model of Figure 5-3 is inadequate, MM next tries to improve it. It begins by invoking rules that add resistors, since `needs-resistor` is one of the model's problems.

Three rules are relevant. One looks for structural evidence of resistance, namely, a narrowing of the walls enclosing the fluid. Since the fluid region in this is example is of uniform thickness, this rule is not activated. Instead, two other rules are used which simply insert resistors into the model, taking care not to introduce redundancies. These rules are:

resistor-1j/fluid (*add a fluid resistor to a 1-junction*)

IF	there is a 1-junction in the model with no resistors attached
THEN	add a resistor to the 1-junction.

resistor-new/fluid (*add a fluid resistor in a region with no 1-junction*)

IF	there is a contiguous region of fluid with no 1-junction
THEN	place a 1-junction in that region and note that the 1-junction represents the region
AND	add a resistor to the 1-junction.

The rules are distinguished only by whether they use an existing 1-junction or create a new one. The second rule, applied to the horizontal portion of the U-tube, yields the model of Figure 5-5.

5.1.3 Adding Capacitance

The model of Figure 5-5 no longer has the problem `needs-resistor`, but its order is still too low. Therefore, MM next invokes rules that add energy-storage elements, since doing so tends to increase the order of a model.

For the U-tube, the important energy-storage element is the capacitor. Each side of the tube can be viewed as a fluid capacitor, because there is a monotonic relationship between a generalized displacement—the volume of fluid in the side—and an effort—the pressure at the bottom of the side.

The ideal rule for fluid capacitors would be of the form "look for fluid regions whose volume changes correspond to pressure changes." However, such a rule cannot be effectively implemented as it stands, because there would seem to be no general way to determine the correspondence from the sort of geometric information MM has

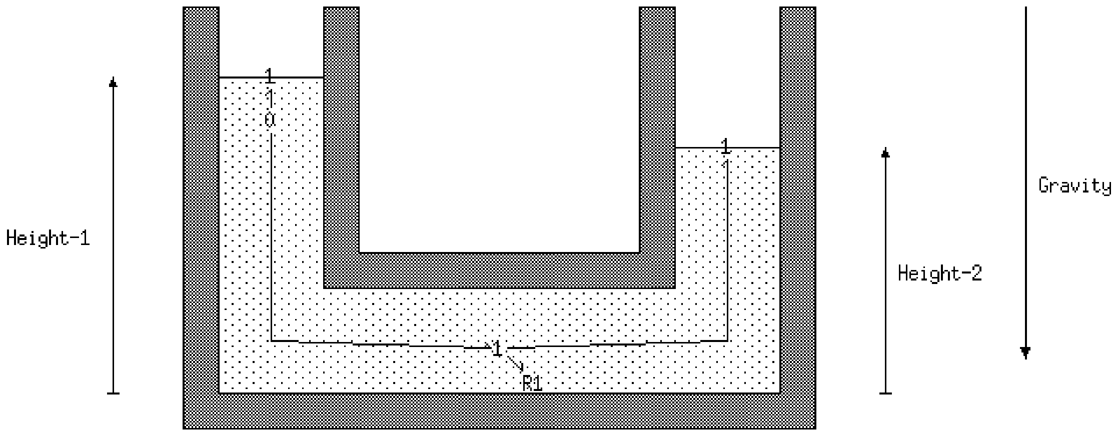


Figure 5-5: U-tube model with resistor

available. Instead, MM looks for geometric clues that point to the likely presence of a fluid capacitor.

MM uses two heuristic rules to look for fluid capacitors. The first is:

capacitor-gravity/fluid (*Propose a capacitor in a partially contained fluid region that admits volume change and that is subject to gravity*)

IF	there is a gravitational force F
AND	there is a region R containing fluid,
AND	R is partly bounded by solid on the bottom,
THEN	add a capacitor to the model and set its region equal to R.

This rule captures a common sort of fluid capacitor, in which gravity is responsible for the monotonic relationship between volume and pressure. A fluid-filled tank is an example, as is a side of the U-tube. The requirement that the fluid region is bounded by solid captures the fact that a capacitor exists in the presence of gravity only when the fluid is contained. The boundary on the bottom need not be horizontal (i.e. perpendicular to the gravitational force) nor need it extend for the complete width of the region. We will see this situation in Section 5.1.5.

MM's second fluid capacitor rule is:

capacitor-movement/fluid (*Propose a capacitor in a partially contained fluid region into which a solid can move*)

IF	there is a solid region S
AND	S is free to move into and away from a fluid region R
AND	R is partly bounded by solid opposite and adjacent to S
THEN	add a capacitor to the model and set its region equal to R.

This rule can recognize a variety of situations that tend to result in a fluid ca-

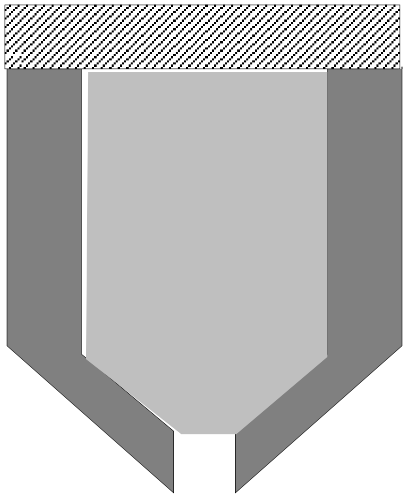


Figure 5-6: A fluid bladder

pacitor. One example is the fluid bladder of Figure 5-6, in which a contained fluid is bordered above by a flexible cap, which can flex into the fluid region (MM's algorithm for determining the movement directions of parts of the system are discussed in Section 5.2.1.) The flexible cap exerts a force on the fluid that increases when the volume of fluid increases, so the pressure at the base of the bladder is monotonically related to the volume of fluid. Another case of this sort of capacitor is a flexible-walled pipe or line; we will see this rule used for this situation in Section 5.4.

These two rules recognize many of the typical occurrences of fluid capacitors in mechanical-hydraulic systems, but I do not claim that they are exhaustive. For instance, one reason for inserting a capacitor into a fluid region is to model the compressibility of the fluid. The above rules do not recognize this case.

Nor are the rules foolproof: it is possible for them to suggest a capacitor in a fluid region that would not, in fact, exhibit capacitance. For instance, consider a solid, unfixed plug resting in a hole in the side wall of a fluid container. If the plug were to be pushed towards the fluid, the conditions for the second fluid capacitance rule would be satisfied, even though this situation does not result in capacitance.

These rules, like MM's other rules, are intended merely as heuristics. They use geometry to suggest spatial regions where effects might plausibly occur. No rule is completely general, but each rule covers a significant class of phenomena. Moreover, adding a new rule to MM is relatively easy and does not require the modification of existing code, so when new situations are encountered MM can be modified to recognize them.

MM implements the fluid capacitor rules in a similar way. For the gravity rule, MM determines the direction of gravity in the structural description (if it is given), and collects all line-segments of the structure that are not parallel to it. (These line-segments are the boundaries of the convex polygons MM constructed when it

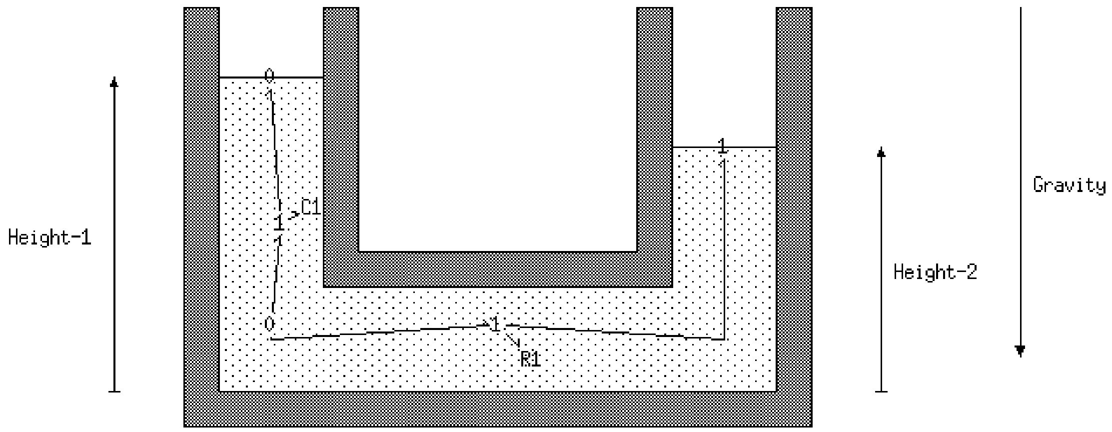


Figure 5-7: U-tube model with resistor and capacitor

divided the structural description into segments.) The second capacitor rule uses the movement directions of solid polygons (determined by the algorithm of Section 5.2.1) to find sets of line-segments. After this point, both rules work as follows.

For each line-segment, the composition on each side of the line-segment is checked, and the line-segment is retained if and only if the lower or farther side (as determined by the direction of gravity, or the direction of movement into the fluid) is fluid, and the other is not. Given such a line-segment, a polygon is formed by extending the line-segment's endpoints through the fluid until the edge of the fluid region is reached, and connecting the resulting points. Each of the candidate line-segments results in one polygon, and each polygon represents a potential fluid capacitor.

There is much that is arbitrary about this construction, but as I argued in Chapter 1, some amount of arbitrariness is inevitable in lumping. I chose the above construction because it is simple and it gives good results. Often, there is no more complex construction that can do any better: as we saw in Chapter 1 and will see again in Section 5.1.5, the curved U-tube admits of no principled choice for region of fluid capacitor.

When MM chooses a region to represent a capacitor, it does more than add a C element. It actually adds four elements: in addition to the capacitor, it adds two 0-junctions to represent the pressures at each end of the rectangle, and a 1-junction for the fluid flow through the rectangle. The extra junctions facilitate the addition of other elements to the model and the joining of model pieces into a single bond graph. The model with a capacitor added to the left side of the U-tube is shown in Figure 5-7.

This model is actually sufficient to yield the desired behavior; it turns out that the other capacitor is not necessary for the first-order case. Normally MM would stop at

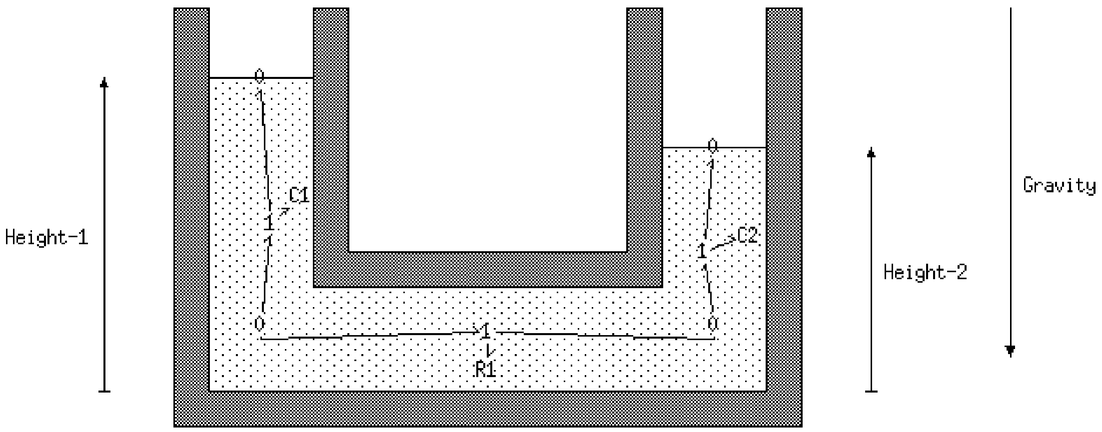


Figure 5-8: Complete U-tube model

this point. If it is allowed to continue running, it will eventually arrive at the model of Figure 5-8, which reflects the "standard," intuitive model in which each side of the U-tube is an independent capacitor.

5.1.4 Second-Order Behavior

In a more common U-tube behavior, the fluid oscillates after being disturbed, and eventually settles. The behavior of Figure 5-9 shows the behavior input given to MM for this case. It describes an "overshoot" behavior, where the fluid dips below its final position and then returns to that position.

This behavior requires a model of at least second order, a fact MM determines when the behavior fails both the zeroth-order and first-order tests. MM proceeds much as before, except now the first-order models of Figures 5-7 and 5-8 are inadequate. MM proceeds to add an inertia to the model of Figure 5-8 to arrive at Figure 5-10, which does adequately capture the second-order behavior. (Adding an inertia is only one of the modifications attempted by MM. It will also add capacitors in an attempt to raise the model's order.)

5.1.5 The Curved U-tube

MM processes the curved U-tube of Figure 5-11 very much like the original U-tube. The fluid capacitor rule is robust enough to work even in the presence of the curved sides. Figure 5-12 shows the final first-order model of the curved U-tube, with the capacitors' regions outlined. As I have argued earlier, this choice of region is arbitrary,

```
(behavior
 (qspace  minf bottom -x 0 x inf)
 (output height-1 position ((fixed bottom))
         (0 x)
         ((0 1) (-x x) dec)
         (1 -x)
         ((1 inf) (-x 0) inc)
         (inf 0))
 (output height-2 position ((fixed bottom))
         (0 -x)
         ((0 1) (-x x) inc)
         (1 x)
         ((1 inf) (0 x) dec)
         (inf 0)))
```

Figure 5-9: Second-order behavior for the U-tube

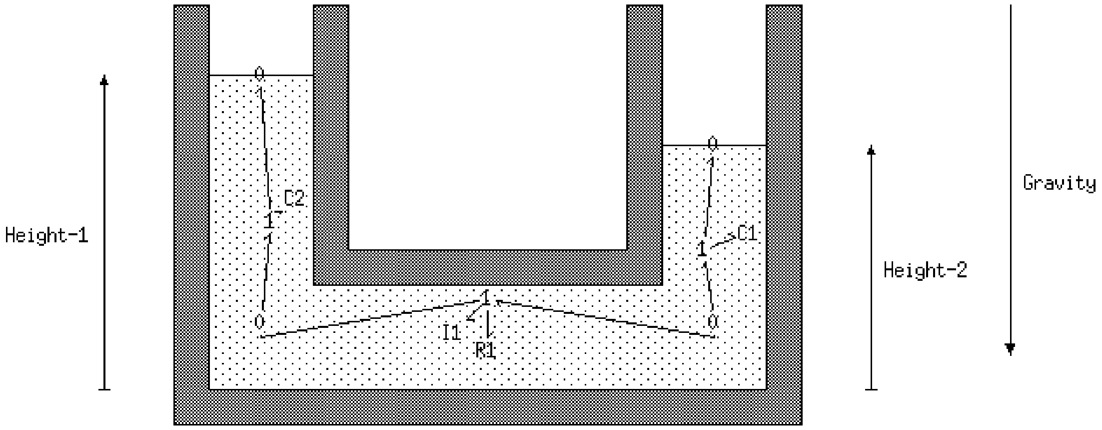


Figure 5-10: Model for U-tube with second-order behavior

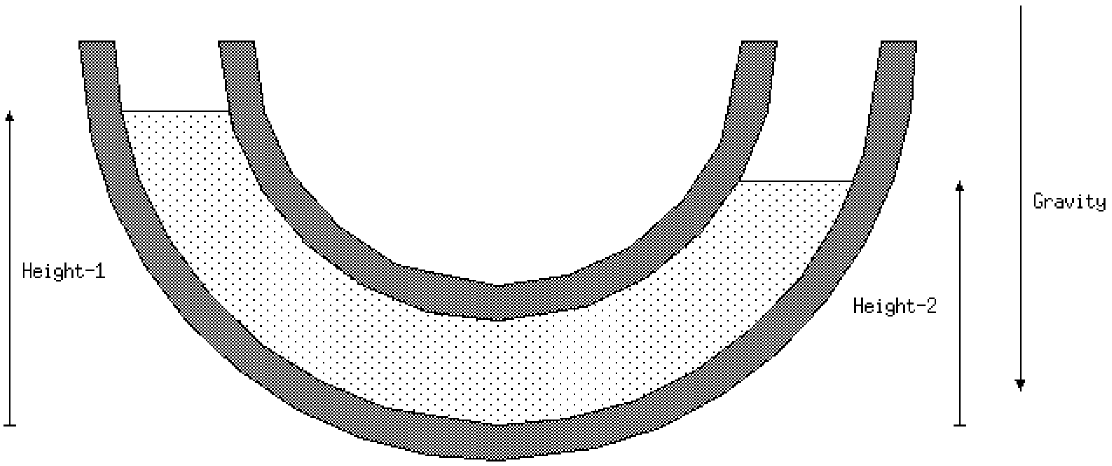


Figure 5-11: Curved U-tube

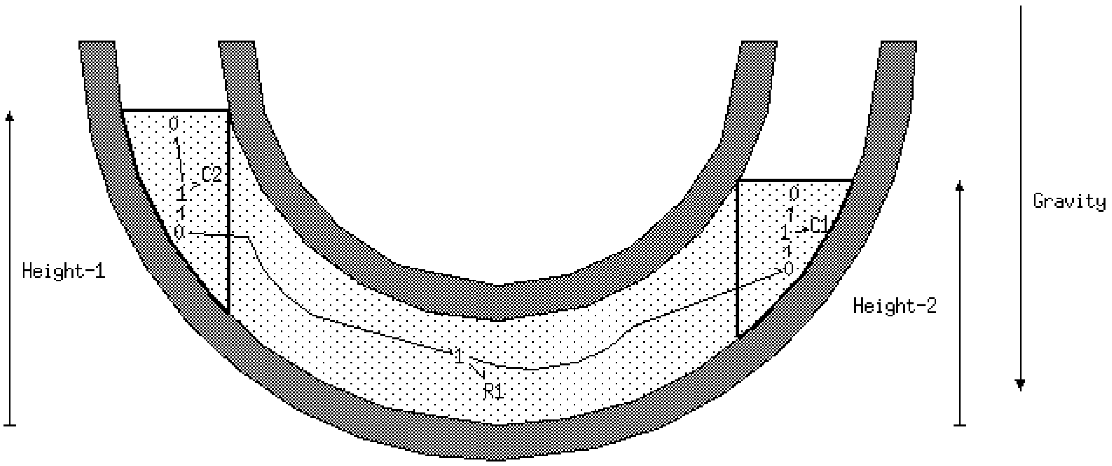


Figure 5-12: First-order model of curved U-tube with regions of capacitance

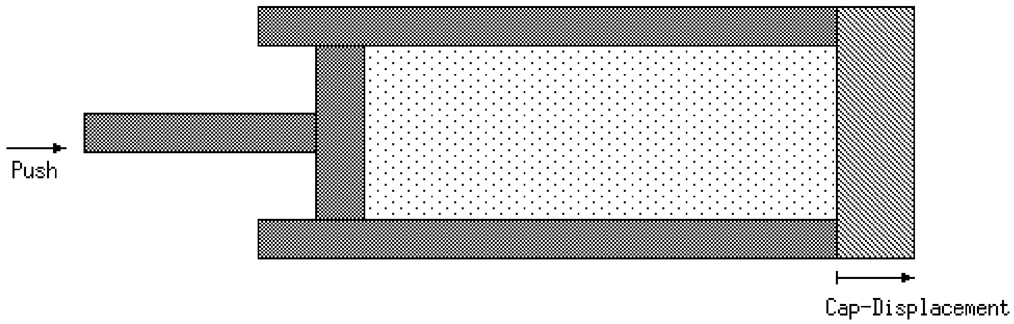


Figure 5-13: A hydraulic piston

```
(behavior
   (qspace minf 0 c f inf)
   (input push force ()
          ((0 inf) f const))
   (output cap-displacement position ((fixed 0))
          ((0 inf) c const)))
```

Figure 5-14: Zeroth-order behavior of the piston

but no worse than any other for the purposes of obtaining a qualitative model.

5.2 The Hydraulic Piston

Figure 5-13 shows a hydraulic piston acting on a fluid inside a tube. The right end of the tube is sealed with a flexible cap. (As you can see from the figure, MM renders flexible parts with a pattern that differs from those for rigid and fluid parts.) The piston walls and the cap are all fixed in place. When the piston is pushed as indicated in the figure, the cap bulges outward.

In addition to the presence of flexible parts, which we have not seen before, this system also presents a more interesting challenge for the problem of connecting separated pieces of bond graph into a single, complete model.

In the first behavior we will consider, the piston is depressed instantaneously and held at a constant position, and the cap displacement occurs instantaneously. (See Figure 5-14.) This behavior can be explained by a zeroth-order model, since there is no lag in the cap's response to the piston.

As always, MM first attempts to interpret the input-output variables it is given. Figure 5-15 shows this interpretation. The force on the piston is interpreted as an ef-

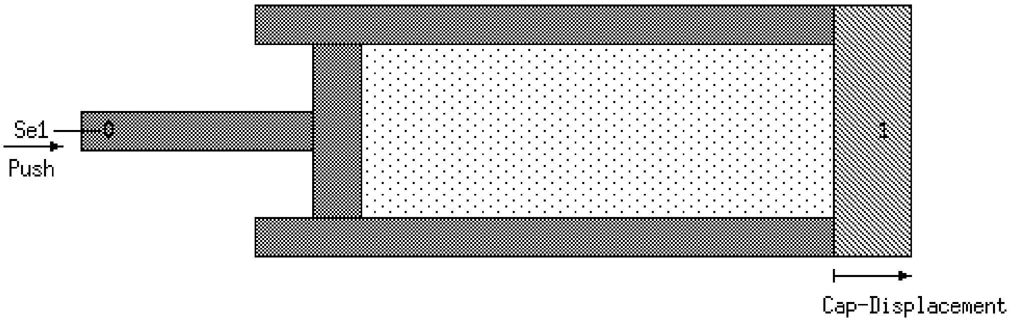


Figure 5-15: Piston with interpreted I/O variables

fort source (connected to a 0-junction), and the displacement of the cap is interpreted as the integral of a flow (the cap's velocity), indicated by a 1-junction.

5.2.1 Connecting Bond Graph Fragments

To analyze the model of Figure 5-15, MM must construct a connected bond graph from the two fragments. In earlier discussions I have glossed over this step, but it is not a mere detail. Here, I explain MM's connection algorithm in depth.

MM's structure recognition rules look for regions of interesting behavior in a model. Once a set of interesting regions has been found, the rest of the structure is, by implication, uninteresting. But this does not mean it is irrelevant. Though there may be no dynamic behavior associated with uninteresting regions of the structure, some of these regions might still be involved in the energy flow of the model: they may serve as conduits of power that link the dynamically interesting parts of the model. That is essentially what we are saying when we place a bond through a region of space: we are asserting that power flows through the region.

The model connection problem, then, can be stated as follows: given two regions in the structure—the interesting regions—determine the paths between them (if any) through which power flows. The model can be completely connected by successively connecting pairs of interesting regions.

My general approach to finding these *power paths* uses the idea of flow. The direction of flow—whether it be mass flow, current flow, heat flow, etc.—is (by convention) the direction in which power flows. Since MM deals only with translational and fluid systems, it deals only with mass flow.

The method requires that the system structure be divided into convex polygons of uniform composition. This step is a by-product of MM's division of the structure into segments, since segments are just groups of adjacent convex polygons of the same composition.

The approach proceeds by determining, for each polygon, in which directions the polygon (or more precisely, the substance contained in it) can move. Here are the rules MM uses to perform this step (recall that there are three compositions: rigid, flexible and fluid).

1. If the polygon is rigid and fixed, it does not move.

2. If the polygon is solid (either rigid or flexible) and is unfixed, it can move in the direction(s) of any applied forces.

3. If the polygon is either flexible, or rigid and unfixed, and an adjacent polygon (of any composition) moves into it, then it moves in the same direction. This applies to fixed flexible polygons, where the movement is deformation.

4. If the polygon is flexible and fixed, and moves in some direction, it also moves in the opposite direction. This captures the elastic property of flexible objects.

5. If the polygon is fluid, flow can occur through any edge that does not border a fixed, rigid object. Note that flow can occur through an edge bordering flexible material, even if it is fixed; this corresponds to a deformation of the flexible substance.

These rules embody a rather coarse and in some cases inaccurate view of flow. For instance, without quantitative information rule 2 has no way to resolve the impinging forces to determine a single, unique direction of motion. This rule, therefore, only gives accurate results in the case of a single impinging force. The rules completely ignore kinematic constraints; in general, MM does not handle kinematics. They also ignore rotation.

The algorithm does perform acceptably in the domain for which MM is specialized: mechanical-hydraulic systems with trivial or no kinematics, where the parts are in contact. Rule 5 accurately predicts possible fluid flow directions, and the other rules handle interactions of solid parts that correspond to one-dimensional translation without rotation.

Once the movement or flow directions of each polygon have been determined, a graph is constructed. The nodes of the graph are polygons, and an directed arc joins two polygons if the first would move into the second. To determine whether one polygon would move into another, MM translates the first polygon by a small amount in its allowed movement directions and tests for overlap with the second. It is this step of the algorithm that imposes the assumption that objects be touching, or nearly touching.

When the graph is complete, determination of power paths has been reduced to graph search: the power paths between two polygons are the paths in the graph between the polygons. The graph for the piston is shown in Figure 5-16. The arrows in the picture are the arcs of the graph.

This graph is derived as follows. The initial, external force (not shown) acts against the piston shaft. Since the shaft is not fixed, it can potentially move in the

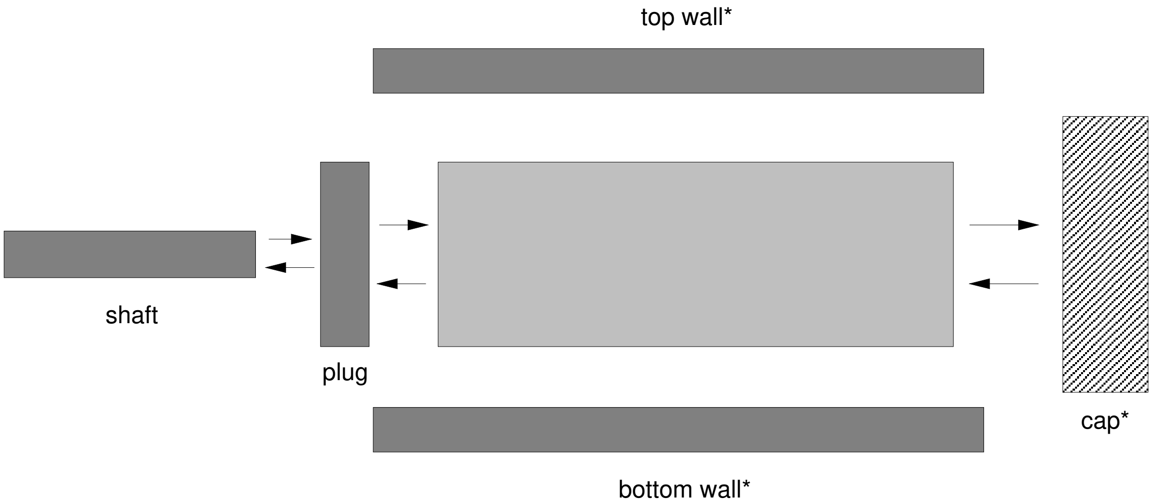


Figure 5-16: Flow graph for the piston. Starred polygons are fixed in place.

force's direction (rule 2). The movement of the shaft impinges on the plug, so the plug also can move in the same direction (rule 2). The fluid polygon can potentially move either to the left, into the unfixed plug, or to the right, deforming the flexible cap (rule 5), which moves left into the fluid (rule 4). The piston walls are fixed, so the fluid cannot move in those directions. The fixed polygons cannot move at all (rule 1).

Once it has constructed the flow graph, MM uses it to compute power paths between polygons that contain bond graph pieces. If no path is found, MM does not connect the pieces. When there is more than one path, MM first eliminates paths with cycles, then connects the pieces along all remaining paths.

Avoiding cycles is a good idea, since they are generally artifacts resulting from the overgeneral treatment of motion. For instance, in the piston flow graph of Figure 5-16, there is a cycle between the plug and fluid polygon which should rightly be ignored. Sometimes, however, meaningful power paths will contain cycles. Consider a device in which an object's horizontal movement results, through a series of interactions, in that same object being moved vertically, perhaps interacting with some other object. That series of motions appears as a cycle in the flow graph, but it is a useful one because the object's directions of motion differ. The flow graph is not refined enough to capture this distinction.

Once a power path has been found, it is not sufficient to simply draw a bond along the path that connects the polygons. If the power path crosses energy domains, transformer or gyrator bond graph elements must be inserted at these junctions to mark the fact. Currently, MM assumes that the transformer is the correct element to insert. This assumption is often correct when dealing with the change from mechanical to fluid domains in which the fluid is contained, for then a change in force on the mechanical side will tend to cause a corresponding pressure change in the contained

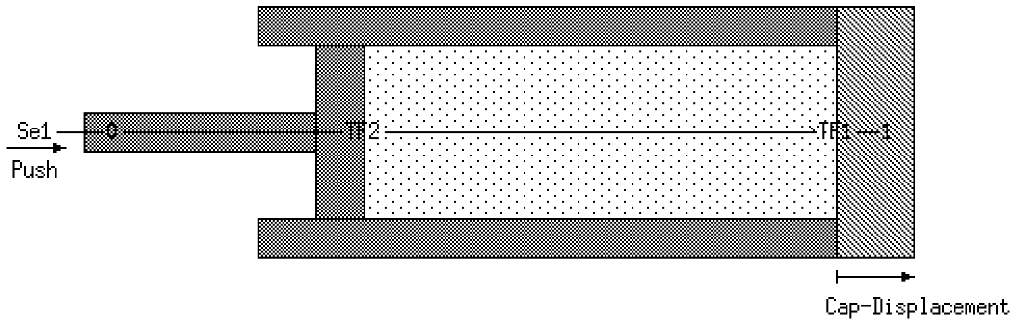


Figure 5-17: Interpreted piston model, connected

fluid, and this is a transformer relationship. However, in other cases, even mechanical-fluid cases, a gyrator is the correct element (e.g. a turbine). MM currently does not have sufficient geometrical or physical knowledge to distinguish these cases. This is an area for future work.

Note that a transformer has no qualitative effect on the model, since it says only that one effort is proportional to another, and likewise for flows. However, MM would be remiss if it did not include them in the model, for they do describe a crucial piece of the model, namely a change in energy domain.

Figure 5-17 shows the connected version of the model in Figure 5-15, with transformers. This model is zeroth-order, but is rejected because MM observes that there is no relationship between the input effort and the output flow. The mere presence of a bond does not guarantee such a relationship. Recall that a bond represents two variables, an effort and flow, but does not place any constraint between them. The bond graph of Figure 5-17 contains no elements which relate the effort and flow, and MM checks for this before performing a qualitative simulation. If it did not, the qualitative simulation would say that the model agrees with the given behavior; because the model's output is unconstrained, it will agree with *any* behavior.

5.2.2 The Zeroth-order Model

One of next models that MM constructs, shown in Figure 5-18, is a correct one. MM constructs it using the following structure recognition rule:

capacitor-force/trans (*Propose a capacitor in a flexible region that can move in opposition to a force*)

IF	there is a force F
AND	there is a rectangle R containing flexible material such that the material can expand or contract in the direction of F,
THEN	add a capacitor to the model and set its region equal to the flexible portion of R.

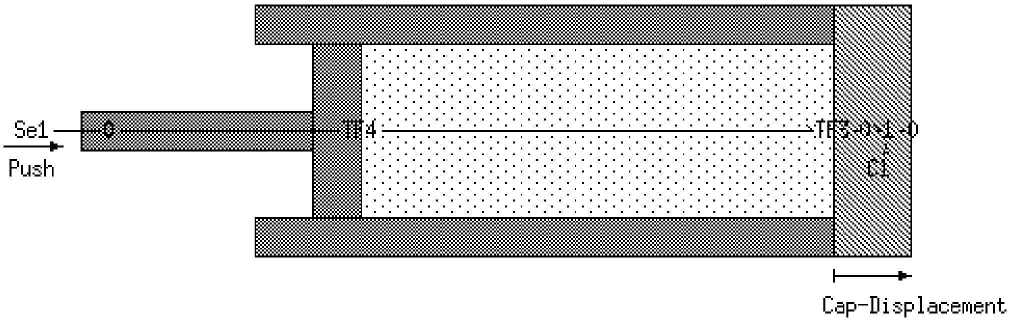


Figure 5-18: Zeroth-order piston model with capacitor

```
(behavior
   (qspace minf 0 c f inf)
   (input push force ()
  ((0 inf) f const))
   (output cap-displacement position ((fixed 0))
   (0 0)
   ((0 inf) (0 c) inc)
   (inf c const)))
```

Figure 5-19: First-order behavior of the piston

Essentially, the rule recognizes the potential for spring-like behavior.

The resulting model is zeroth-order despite the presence of a capacitor, because the capacitor is dependent. The model asserts that the cap position is a function of the force in the piston, which is consistent with the behavior.

5.2.3 The First-order Model

When MM is given the behavior of Figure 5-19, in which there is a delay before the cap displacement reaches its maximum value, MM proceeds as before except that now a first-order model is required. To obtain this model from Figure 5-18, it is only necessary to add a resistor. One such model, in which the resistance is attributed to the fluid, is shown in Figure 5-20.

5.3 The Motor and Flywheel

The motor-flywheel system serves two purposes. It is our first example of a system that uses a component, rather than geometric, representation for structure. It also

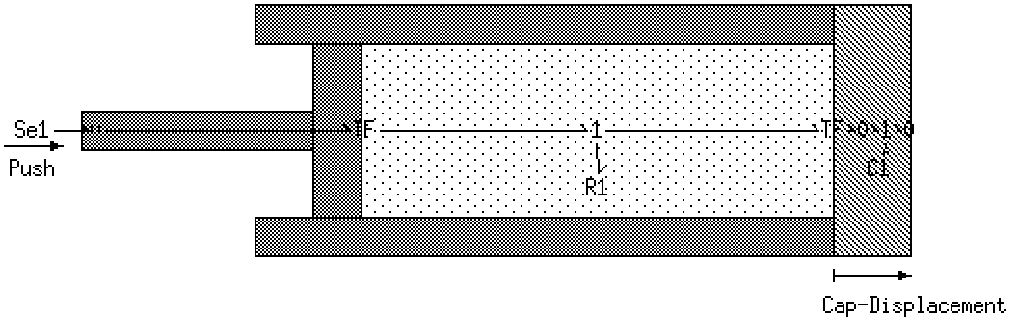


Figure 5-20: First-order piston model

```
(defstructure motor-flywheel
    (components
     (Motor motor)
     (Flywheel flywheel)
     (Battery battery))
    (connections
     ((Motor shaft) (Flywheel shaft))
     ((Battery leads)  (Motor leads))))
```

Figure 5-21: Motor-flywheel structure (shape information elided)

provides a good example for a discussion of MM's experiment generation facility.

The motor-flywheel system is quite simple: it consists of an ordinary motor whose shaft has a flywheel at its end, connected to an electrical power source. The input to MM for this system consists solely of component information and is shown in Figure 5-21.

MM's component descriptions for the three components used in the system are given in Figure 5-22. A motor has two ports, an electrical (leads) and a rotational (shaft). It is modeled initially as a gyrator. A flywheel is initially modeled as a rotary inertia, and a battery as an effort source.

When it processes a system described with components, MM begins by composing the bare models of the components to form an initial model. For the motor-flywheel system, this initial bond graph is shown in Figure 5-23. Recall that the shapes in the figure are specified by the user for cosmetic purposes only.

A first-order behavior for the motor-flywheel is shown in Figure 5-24. A constant battery voltage causes the flywheel's angular velocity to ramp up from zero to some constant. The initial model, however, predicts that the flywheel's velocity will increase forever, so the model must be modified. The problems with the initial model are both

```
(defcomponent motor
  (ports (leads electrical)
         (shaft rotational))
  (bare-model
   (GY leads shaft)))


(defcomponent flywheel
  (ports (shaft rotational))
  (bare-model
   (I shaft)))


(defcomponent battery
  (ports (leads electrical))
  (bare-model
   (Se leads)))
```

Figure 5-22: Component descriptions used in the motor-flywheel system

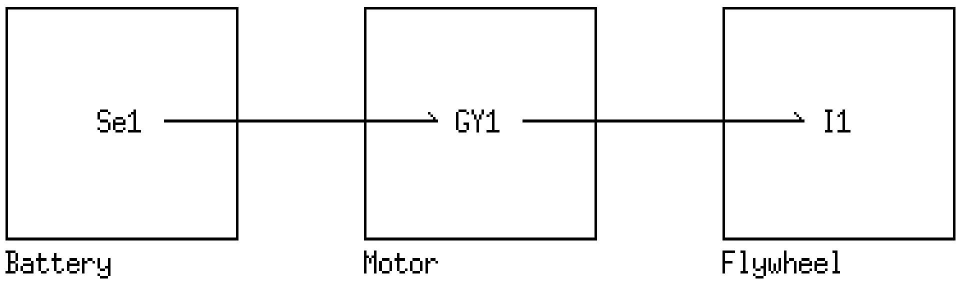


Figure 5-23: Initial model for motor-flywheel system

```
(behavior
   (qspace minf 0 +12v c inf)
   (input (Battery volts) voltage ()
          ((0 inf) +12v const))
   (output (Flywheel speed) angular-velocity ()
           (0 0)
           ((0 inf) (0 c) inc)
           (inf c const))))
```

Figure 5-24: First-order behavior of the motor-flywheel

that its order is too low (the single inertia is dependent) and that it needs a resistor.

Once the initial model is constructed, MM follows the same procedure with a component system as it does with a geometric system. However, the rules it uses to add bond graph elements to component models differ from the structure recognition rules employed for a system described geometrically. Since there is no information available about the system aside from the component types and their connections, MM's component rules simply add bond graph elements to component bare models, taking care not to create trivially redundant models. For instance, a resistor will not be added to a 1-junction if there is already a resistor attached to the junction, because two resistors with monotonic characteristics that are attached to the same 1-junction are equivalent to a single monotonic resistor. Facts like this one are built in to the procedure for generating non-redundant models. The procedure takes into account the possibility that more than one element of a given type may be added to a component model without creating redundancies. For instance, the motor model can have a resistor on either or both sides of the gyrator.

There are four rules for elaborating component models. One adds a resistor at a 1-junction (and if necessary, adds the 1-junction as well). A second does the same for inertias. The third rule, for capacitors, is identical except for the use of 0-junctions instead of 1-junctions. Capacitors typically occur on 0-junctions because they involve a split in flow. For instance, in a fluid capacitor such as a tank with a hole at the bottom, some of fluid flow passes through the tank while the rest remains inside the tank. The fourth rule adds a resistor at a 0-junction, which models leakage.

These four rules do not by any means capture all the possibilities. While it would be easy to add other component-elaboration rules to MM, some additions would be unwise and others superfluous. Adding source elements would effectively sanction hidden inputs to the system, enormously reducing the constraints on possible model behaviors and making analysis all but impossible. Adding transformers is superfluous because they contribute nothing to the qualitative behavior of the model. Gyrators are also superfluous if the bond graph is a tree, because a gyrator can be eliminated by dualization of the bond graph on one side of the gyrator. All the bond graphs

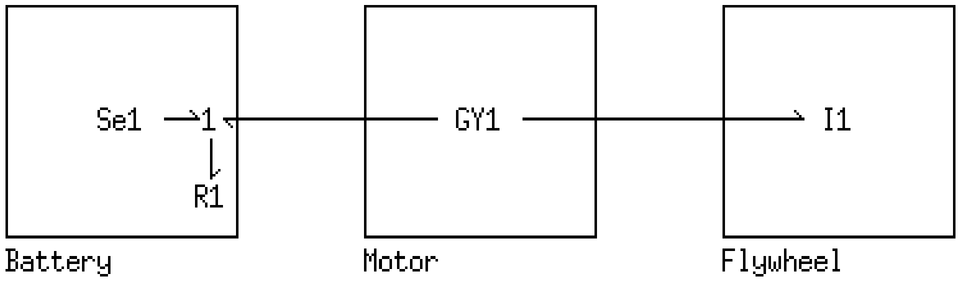


Figure 5-25: One model for the motor-flywheel system

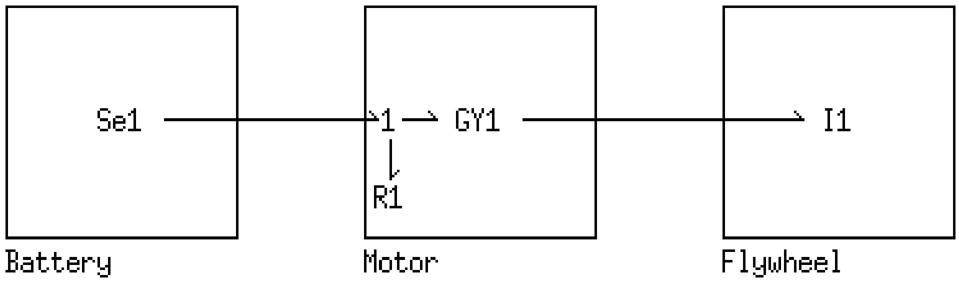


Figure 5-26: A second model for the motor-flywheel system

constructed by MM's component elaboration rules are trees, because the rules always add new elements to a single element on the existing graph. 0—I and 1—C rules are also plausible. Adding these rules would not prevent MM from discovering any models it presently discovers; it would simply increase the time MM takes to do so.

Rather than provide MM with new component-elaboration rules, future work should be directed towards adding information to the built-in component models in order to allow MM to intelligently choose how to elaborate each component model.

Since one problem of the initial motor-flywheel model is that it needs a resistor, MM invokes its resistor-adding rule. There are four potential locations for a resistor: in the battery, in the flywheel, and on either side of the motor's gyrator. Resistors to the right of the gyrator (the mechanical part of the system) still leave a zeroth-order model,[1] but resistors to the left result in a first-order model that does in fact predict

[1]The alert reader may find this counterintuitive. It implies, for instance, that the friction between the motor shaft and housing cannot by itself result in the gradual increase and tapering off of the flywheel's velocity, no matter how great that friction is. The problem is that by modeling the battery

the desired behavior. There are two such correct models, one with the resistor in the battery and one in the motor. They are shown in Figures 5-25 and 5-26. MM cannot distinguish between them with the information it has, but it can do so by obtaining additional measurements. That is the topic of the next section.

5.3.1 Experiment Design

If we have multiple models for a system, all of which correctly predict the behavior of the system, and we wish to distinguish the candidate models, one choice would be to flesh them out fully by assigning quantitative values to the model parameters, based on material properties or component specifications, and to compare the models' predictions with the behavior quantitatively. The other choice, and the one I will pursue here, is to obtain additional measurements of the system and use them to distinguish among the models. The aim, then, is to design experiments to distinguish among competing models.

Since measurements tend to be expensive or time-consuming compared to calculation, the goal of experiment design is to choose those measurements that will provide the most benefit for the least cost. I assume, for simplicity, that all measurements are equally costly. The notion of benefit I will use favors those measurements whose outcomes can eliminate the greatest number of models. My experiment design method chooses the measurement that maximizes the expected number of eliminated models.

More formally, say we have a set of models M and a set of measurable variables V, both finite. Each model m predicts a certain outcome or outcomes for each variable v; we write this set of outcomes as $m(v)$. We will assume this set is finite. Each variable v, then, has a finite set of possible outcomes $o_1, o_2, \ldots, o_{n_v}$. We will write the probability of an outcome's occurrence as $P(v, o_i)$.

For each outcome, some of the models in M agree with the outcome and some do not; the latter are the ones that would be eliminated if the variable exhibited that outcome. That is, given a variable v, an outcome o and a model m, m is eliminated if

$$o \notin m(v)$$

Let $E(v, o)$ be the number of models eliminated when variable v has outcome o. Then the expected number of eliminated models for a variable v is

$$EE(v) = \sum_{i=1}^{n_v} P(v, o_i) E(v, o_i)$$

as an effort source, we are assuming that it can supply enough current to overcome any mechanical friction. Only by modeling the inherent electrical resistance can the proper effects of mechanical resistance be taken into account.

If we assume that the outcomes are equiprobable, this reduces to

$$EE(v) = \frac{\sum_{i=1}^{n_v} E(v, o_i)}{n_v}$$

The best variable to measure is the one with the largest value of EE.

This measure of variable quality is simpler than, but similar to, the expected information gain of the variable. The information-theoretic approach would allow for incorporation of prior probabilities on the models. Both approaches have the property that they will prefer variables that tend eliminate *all* candidate models. This is desirable if one is willing to go back to the drawing board and obtain better models; it is a drawback if one wants the best of the current set of models.

I implement this theory of experiment design in MM as follows. First, I assume that the only measurable quantities are the efforts and flows between components. In other words, you cannot open up a component and take a measurement inside it; nor can you measure anything but an effort or flow. To compute $E(v, o)$, MM simulates each model in M—the candidate models—and keeps track of the values of these measurable variables. It is not necessary to perform multiple simulations per model, because the simulator keeps track of all variables "in parallel," as it were. Furthermore, since we are only interested in model behaviors that conform to our original given behavior, MM can use QSIM-CHECK to perform the simulation. The only change to the QSIM-CHECK algorithm described in Section 4.3.2 is that the simulation proceeds even if a matching behavior is found, rather than halting immediately with success.

Once all simulations have been completed, it is a simple matter to compute $EE(v)$ for each variable. n_v is determined by counting the number of distinct behaviors of the variable over all simulations, and $E(v, o)$ is determined by counting the models that don't include o in their simulation.

After MM conducts the above analysis, it selects the best variables and prints a summary of its results. The output for the motor-flywheel case is given in Figure 5-27. Note that in addition to listing the best variables to measure, MM also lists those that are useless. The output suggests measuring the voltage between the battery and motor. In the case where the resistor is inside the battery (model `BG-450`, corresponding to Figure 5-25), this voltage will "ramp up" to a constant value. When the battery is a pure effort source (model `BG-447`, corresponding to Figure 5-26), the voltage is constant throughout.

5.4 The Table Bed System

Figure 5-28 shows a table-bed positioning system, such as might be used in a manufacturing plant. It consists of an electric motor coupled to a rack-and-pinion via a hydraulic linkage that includes a positive-displacement pump, a valve, a filter, and a

```
There are 4 measurable variables.
3 of these provide no information.  They are:
The current between the leads of motor and the leads of battery
The torque between the shaft of motor and the shaft of flywheel
The angular-velocity between the shaft of motor and the shaft of
  flywheel

The best variables to measure are the following:
The voltage between the leads of motor and the leads of battery:
                Behavior                                 Possible Models
((T0 0 INC) ((T0 T1) (0 C) INC) (T1 C STD))       BG-450
((T0 +12V STD) ((T0 T1) +12V STD) (T1 +12V STD))   BG-447
```

Figure 5-27: Output of MM's experiment design facility

hydraulic motor.

This is the largest example that MM runs on. It demonstrates that MM can be used successfully on systems of moderate complexity, and it also illustrates how mixed component-geometric structural descriptions are handled.

Figure 5-29 shows MM's rendering of its input for this system. Note that a part that was implicit in the sketch of Figure 5-28, namely the flexible hydraulic line that connects the pump to the valve, is represented explicitly as an object which consists of three geometric parts: the top and bottom walls, which are of flexible composition and are fixed in place, and the central fluid region. The left and right line-segments of the fluid region are designated as hydraulic ports, and connections are specified which join the pump's outlet to the left port and the right port to the valve's inlet.

The initial bond graph for the table-bed system is shown in Figure 5-30. Explanations for some of the bare models follow:

- The power source is modeled as a constant electrical effort (i.e. voltage) source.
- The pump transforms rotational to hydraulic energy. In a positive-displacement pump, the kind modeled, the angular velocity of the shaft is proportional to the velocity of fluid through the pump; hence a transformer is used.
- The bare model for the valve specifies only that it divides the fluid flow into two parts, one that delivers power to the motor, and one that serves as a shunt. The adjustable nature of the valve is not captured.
- The filter is modeled only as a connection between its inlet and outlet (a 1-junction with two bonds has no effect whatever; it is present only to simplify the component-model composition process). The 0-junction to the right of the filter is inserted by MM to indicate that two paths are joining at that point.

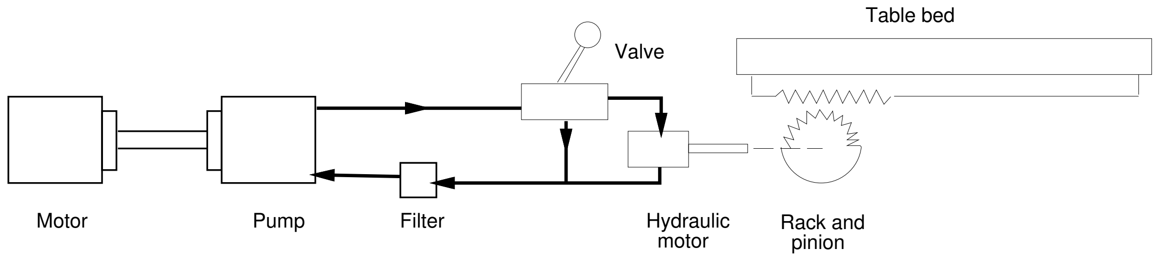


Figure 5-28: A table-bed positioning system (after [18])

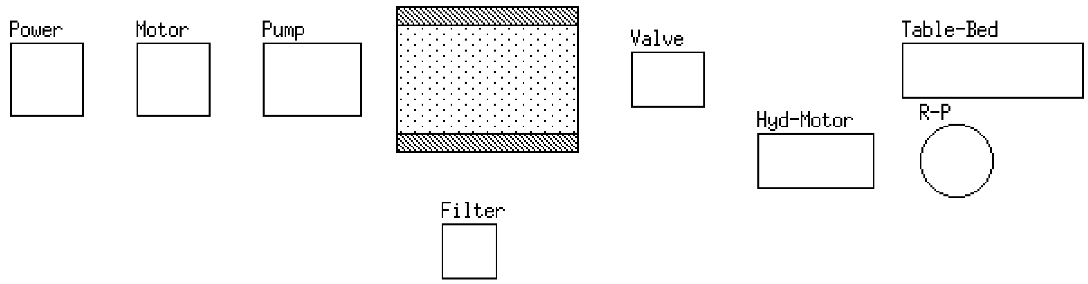


Figure 5-29: MM's version of the table bed system

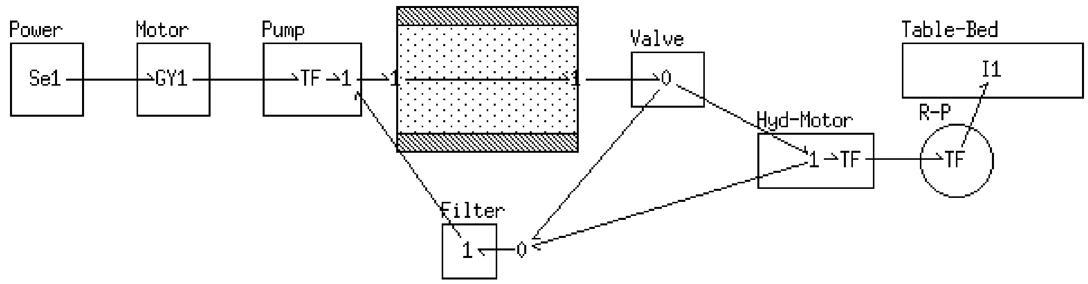


Figure 5-30: Initial bond graph for the table-bed system

```
(behavior
   (qspace minf 0 c1 c2 c3 inf)
   (input (Power volts) voltage ()
          (0 c1 const)
          ((0 inf) c1 const)
          (inf c1 const))
   (output (Table-Bed speed) velocity ()
           (0 0)
           ((0 1) (0 c3) inc)
           (1 c3 const)
           ((1 inf) (c2 c3) dec)
           (inf c2 const))))
```

Figure 5-31: Second-order behavior of the table-bed

- The hydraulic motor is the opposite of the pump: it transforms fluid flow into the rotation of a shaft.
- The rack-and-pinion transforms rotational motion to translational motion.
- The table-bed, like the flywheel, is simply an inertia.
- Geometric parts have no associated bare models. When geometric parts are mixed with components, MM simply connects the ports of the part to each other (inserting 0-junctions where necessary to indicate split flow, as with the filter). The 1-junctions at the port boundaries are useful markers and, like the filter's 1-junction, have no other effect.

This model is zeroth-order. When presented with a zeroth-order behavior, MM need do no additional work. When presented with a second-order behavior like that shown in Figure 5-31, which shows the table-bed's velocity overshooting before settling down to match the constant input voltage, MM determines that the initial model has two problems: it requires a resistor, and it must be at least second-order.

MM begins by adding a resistor. MM tries all possible locations for adding a resistor, of which there are many, but we will consider the case where the resistor is associated with the power source.

To raise the order of the model, MM must insert an energy-storage element. Because a geometric description provides some clues about what behavior a part may exhibit over and above the bare-model behavior, whereas component descriptions do not, MM's geometric rules take precedence over its component rules. Thus one of the first models MM constructs is that of Figure 5-32. The capacitor-movement/fluid rule we have seen before in Section 5.1.3 is responsible for inserting the capacitor into

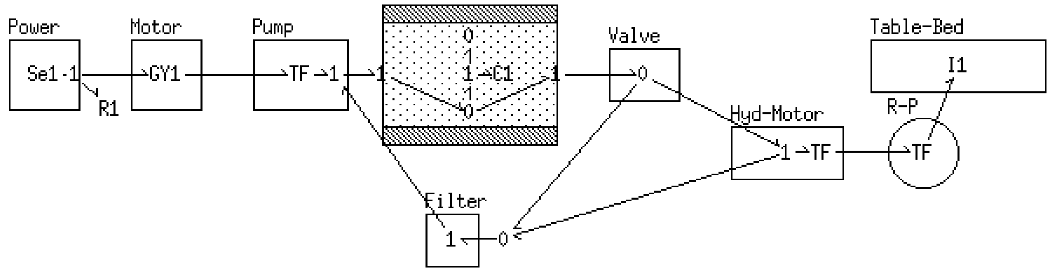


Figure 5-32: Table-bed model with resistor and fluid capacitor

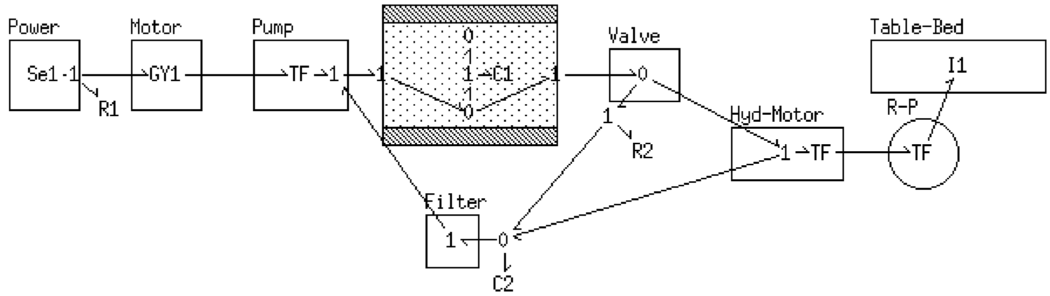


Figure 5-33: Correct table-bed model

the hydraulic line. The rule fires because MM determines, from the movement rules detailed in Section 5.2.1, that the flexible upper wall can move into the fluid.

Although this model contains a resistance and is second-order, it does not correctly predict the given behavior. A correct model is shown in Figure 5-33. A second resistance is necessary to avoid a hydraulic short circuit; this resistance corresponds to positioning the valve so that some of the fluid flows into the hydraulic motor. Without the resistance, all the fluid would flow through the valve shunt. The second capacitor is necessary because the volume of fluid in the hydraulic part of the system is constant, so when the first capacitor (the fluid line) expands or contracts, some other part of the hydraulic system must take up the slack. There are several possible locations for the second capacitor besides the one shown. Adding a resistor to a 0-junction, to model leakage of hydraulic fluid, would also have been possible.

Although MM can generate models through Figure 5-32 on its own, the remainder of the example proved too much for it. In order to complete the run in a reasonable amount of time, it was necessary to hand-guide MM towards the model of Figure 5-33 by pruning models I knew would not lead to the solution. This interference did not and could not result in MM creating any models that it would not have created automatically; it simply prevented MM from wasting its (and my) time.

The table-bed system is a dramatic illustration of the utility of MM's analysis heuristics in avoiding expensive simulation. The order and resistance heuristics take a few seconds on the table-bed models, but simulation, even with QSIM-CHECK, takes many minutes. Most of the time is in constructing the many initial states that can result from the partial description of the initial state given by the input and output variables. For models as complex as that of Figure 5-33, the number of initial states proved to great for my computer, and I was forced to verify the correctness of the model by hand.

What has been gained by representing a hydraulic line geometrically? First, we have been compelled to represent it in some fashion. Note that if we drew up a component model directly from Figure 5-28, it would have been all too easy to omit the lines and simply assert that the pump and valve were connected. (Indeed, representing one line geometrically does not grant us immunity from this problem: there are other hydraulic lines that we have neglected.) The second advantage of providing a geometric model for the line is that we have helped focus MM's search. Geometric descriptions are richer in behavioral clues than component ones, so MM uses geometry first before resorting to the relatively blind strategy of adding elements wherever one is possible. Of course, this does not imply that the behaviors of geometric parts are always more important than those of components. It may be, for instance, that the springiness of the flexible hydraulic line is not the major contributor to the system's capacitance. But at least behaviors derived from geometry have a more palpable justification than those derived from elaborating component bare models.

Chapter 6

Conclusion

This final chapter includes a review of the automated modeling literature, a summary of MM's contributions, and a discussion of directions for future work.

6.1 Related Work

Automated modeling has only recently gained the momentum needed to make it an acknowledged subfield of Artificial Intelligence. However, work akin to automated modeling has been going on in several related fields, and I first present a brief survey of this work. The second part of this literature review focuses on the more recent collection of papers which are closest to this thesis in goals and approach.

6.1.1 Other Fields

The origins of work in automated modeling can be traced to work in solving textbook physics problems (e.g. [25, 6, 4, 15]). Modeling the problem is a vital part of solving it, as emphasized particularly in [15]. The textbook domain has three properties that make it quite different from the sort of work I have described in this thesis. First, there is often a natural-language component to textbook problem-solving programs. Second, such programs have access only to the information given in the problem; there is no behavioral data against which models can be checked. Third, because there is no other information about the system, special conventions are involved in understanding textbook problems. For instance, the problem is expected to have all the information needed to solve it.

Work in automated scientific discovery (e.g. [20]) is akin to automated modeling in that the goal is to come up with a useful, accurate description of some aspect of the world. The difference is that work in scientific discovery has generally assumed a minimal background theory which cannot, of itself, explain the phenomena presented to the program, whereas in automated modeling it is assumed that the available knowledge suffices for constructing a correct model. In other words, a scientific discovery program is supposed to develop a scientific theory (or a piece of one), while an automated modeler is supposed to apply established scientific theories to the understanding of systems whose physical interactions are unexceptional.

The techniques of *system identification* [21] are designed to solve a particular, mathematically tractable piece of the automated modeling problem. System identification uses a system's behavior to determine the numerical parameters for a given model. When the model is linear, this problem is well-understood and many good techniques are available, but there are few general purpose methods for non-linear systems. MM's task can be viewed as complementary to system identification: after MM constructs a qualitatively accurate model for a system, system identification techniques can be used to determine the model's parameters.

6.1.2 Automated Modeling

It is only in the past five years that work on the problem of automating model construction for physical systems has begun in earnest. Here I discuss some of the key examples of this work.

There has been some important work on building kinematic models of systems [11, 14, 17]. In the latter two cases, this has been combined with some dynamics. I do not discuss this work in detail because the issues involved in kinematic modeling are quite different from those for dynamics.

Richard Doyle's JACK program [7] took a qualitative behavioral trace of a system as input, along with limited structural information, and hypothesized models for the internals of the system. Doyle's work differs from mine in several ways. I assume that more structural information is given about the system; Doyle viewed his system largely as a "black box." The qualitative properties of the input that we use also differ considerably: in my case, they are based on theorems of system theory, which are physically accurate; Doyle used different properties, motivated by his representation. Doyle's representation of devices is richer than mine in that it considers discontinuities and certain energy domains (e.g. light) which mine omits; on the other hand, Doyle's representations of many components are physically inaccurate. JACK's representation is weakened considerably by this fact. For instance, JACK cannot represent acceleration. Many important physical effects, like the oscillation caused by the presence of capacitance and inertance in a system, cannot be represented in Doyle's framework.

Ulrich's program [30] produces models that satisfy a given relationship between input and output. For instance, his system may be asked to produce an accelerometer, i.e. a device that takes acceleration to displacement. The first part of Ulrich's program enumerates bond graphs that satisfy the behavior; the second part instantiates these bond graphs in a crude way, using one component per bond graph element, and then performs *function sharing* to simplify the system by combining multiple functions into a single component.

In a way, Ulrich's function-sharing technique is the inverse of what MM does in the case when a component description is given as input. My program begins with very simple models of individual components, then adds functions to these components in order to match the input behavior; Ulrich begins with the diverse functions and tries to make a single component from them. For instance, MM begins with a model of a pipe that has no functions except to transmit fluid, and then adds resistance and capacitance effects to match the data. Ulrich's function-sharing method would begin with resistance and capacitance functions, and perhaps come up with a flexible, rough-walled pipe to embody those functions.

Bruce Wilson and Jeffrey Stein [35] have recently written a program that generates low-order linear models for a class of mechanical systems. Wilson and Stein consider specifically the problem of when to treat a shaft or other rigid part as having compliance (mechanical capacitance). Wilson and Stein's inputs are a component-like structural description of the system, and a Frequency Range of Interest (FROI). The program attempts to construct the lowest-order model it can (that is, to add the fewest compliances it can) while including all frequency modes in the FROI. (Wilson and Stein do not consider adding inertias because part of their problem requirement is that all the inertias of the system be modeled at the outset.) For each type of part, Wilson and Stein have written a model generator that enumerates ever more complex models for the part. A shaft, for instance, may consist solely of an inertia, or it may consist of a single compliance with an inertia, or it may consist of several repeated compliance-inertia groups. The generative nature of this model construction process makes it quite similar to MM. Wilson and Stein are also the only researchers mentioned here who take into account the fact that a single component may have an unbounded number of models. One problem with this work is that it applies only to linear systems, because only with linear systems is the issue of frequency response well-understood.

In Brian Williams' work on interaction-based invention [32, 33, 34] models which meet a behavior specification are constructed by generating paths through the graph of possible component interactions (i.e. equations) that connect the input and output variables. (Actually, Williams seems to use something closer to causal ordering (see below) to relate variables.) Williams' association between components and models is one-to-one: for each component in the hydraulic domain he explored—pipes, tanks, valves, etc.—there is a single model. For instance, Williams models a pipe with Ohm's law, i.e. as having a fixed, linear resistance. In [34] he shows how to construct an abstracted model that has only those interactions relevant for answering a particular query. In a sense, then, Williams' model-construction procedure begins with a "most complex" system model and simplifies it, as opposed to MM's approach of building up from simpler models. The difficulty with Williams' approach is that there may not be a most complex model of a component or device. As we saw above in the discussion of Wilson and Stein's work, a shaft can be modeled by an infinite set of ever more complex models.

One recently influential idea in automated modeling has been the *graph of models*. Penberthy [26] developed the idea in the context of modeling for design. The graph of models has been used in the PROMPT program [22, 1]. The graph of models is a graph whose nodes are models and whose arcs are labeled with the assumptions that must be added or removed to switch from one model to another.

The idea behind the graph of models is to use the simplest possible model until it proves inadequate. MM follows the same maxim: I begin with a simple model and elaborate it until it is adequate. The difference is that MM uses a *generative* theory of models—energy-based modeling—to construct the graphs of models dynamically. As an example, consider Figure 6-1, which shows a graph of models for a pipe. There are eight possible models, corresponding to every subset of the three elements of resistor, capacitor and inertia. A program using the graph of models approach would explicitly represent all eight models for the pipe, but in MM all but the simplest model for each component is implicit. The program may end up with any model in any graph, but it does this by composing individual elements in response to experimental results or to geometrical clues.

Dan Weld's work on automated model-switching [31], uses a graph-of-models approach. Given a system and a model of that system, Weld uses discrepancies between predicted and observed behavior to suggest a better model. Weld does not create the new model dynamically; instead, all models are stored in a graph of models. Each model in Weld's graph is a model of the *entire* system. It seems that it would be possible, however, to use Weld's technique on component models as well.

It was realized that having a graph of models for an entire system would require a prohibitive amount of space, and even a graph-of-models approach applied to each component would be inefficient and redundant. More sophisticated modeling representations were developed, notably the Compositional Modeling approach of Falkenhainer and Forbus [9, 10].

Falkenhainer and Forbus address the problem of how to construct a model that is adequate for answering a particular query without being overly detailed or expensive to use. The program is given a query (typically asking how, or whether, one quantity influences another) as input, as well as a component-like representation of the system structure. It has access to a large library of model fragments, each of which describes the behavior of a particular device or process (used in the sense of QP theory [12]). The quantities mentioned in the query are used to help select a set of model fragments.

Falkenhainer and Forbus' model fragments are carefully designed to be as modular and reusable as possible. It is possible to instantiate multiple models for a single object, so that both the resistance and inertance of a pipe could be considered together. Unlike a graph-of-models approach, the resistance and inertance effects would be described by separate model fragments, so all eight pipe models could be expressed with only three model fragments.

Falkenhainer and Forbus pay particular attention to the variety of assumptions and conditions on the modeling task, using them as constraints to ignore irrelevant

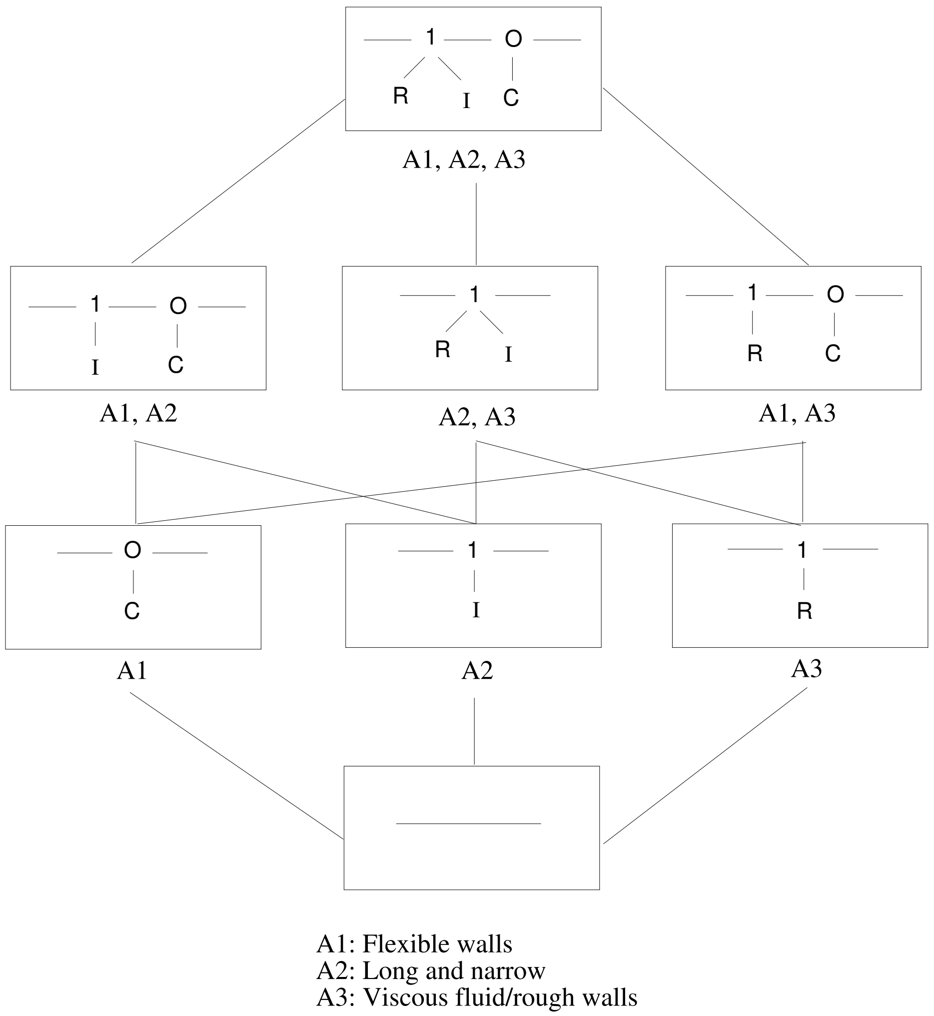


Figure 6-1: Graph of Models for a pipe

parts of their model fragment library and as guides to providing adequate models. For instance, each model fragment contains its operating conditions—the scope of its valid behavior. One important notion is that of the *assumption class,* an ordered list of exclusive modeling choices for a single dimension. For instance, the fluid-viscosity assumption class includes inviscid, viscous and non-newtonian assumptions, in that order. Model construction will begin with the simplest assumption—inviscid flow (no resistance)—and switch to others only when required by some other aspect of the modeling process, or when simulation of a completed model shows that an assumption is violated. These modeling assumptions are not global; different objects in the system can be subject to different assumptions.

The attention to explicit representation of assumptions points up a weakness in MM: most of its assumptions are implicit, in the sense that they are nowhere represented in the program itself.

Although it is an improvement over a graph-of-models approach, the model fragment library still has two weaknesses. First, it fails to capture some commonality in the behavior of physical systems. There is no single model fragment for capacitance across energy domains, as there is a single capacitor element in the energy-based modeling approach. And second, there is no way to represent the potentially infinite number of models that some devices possess, like the shaft discussed earlier.

Although Falkenhainer and Forbus use geometric information in the construction of their models (e.g. some models allude to the diameter of objects), their input is definitely component-like, not geometric. Hence they beg some modeling issues that MM does not.

Jeff Rickel [27] is in the process of extending the compositional modeling approach to the domain of plant physiology. Some of Rickel's improvements include paying attention to the time scales at which model fragments are valid in order to produce smaller and simpler models, and relaxing a restriction which required that all model fragments fit into a single partonomic hierarchy. In addition, Rickel uses Williams' idea of interaction to help determine which model fragments are relevant to a query. Like Williams, he looks for a connected path of interactions between input and output quantities.

MM does not use interaction paths to construct models, but it does use a similar idea during the analysis phase, when it verifies that paths through the model's equations exist for all pairs of input/output variables. Those paths are identical to interaction paths.

One problem with using interaction paths for model construction is that they provide no guide to choosing dynamic effects and thus will tend to give rise to zeroth-order models. Consider the motor-flywheel system of Section 5.3. The bare model for that system includes an interaction path between the input and output, but it cannot account for the system's first-order behavior, or for the fact that that behavior requires a resistor.

In general, techniques that are insensitive to dynamic behavior, like interaction paths, will tend to produce models with overly simple dynamic behavior, if the model fragment library includes simple model fragments. One can avoid the problem by omitting simple models from the library, but then one is caught on the other horn of the dilemma: the generated models will tend to be too complex. The solution is to include simple models, but to augment an interaction-path approach with one that is sensitive to dynamic behavior. MM's method is to augment the simplest model until it matches a given behavior. Rickel's time-scale technique, and Falkenhainer and Forbus' idea of remodeling when a simulation reveals that previous modeling assumptions have been violated, are two other methods sensitive to dynamic behavior.

The work of Pandurang Nayak [24, 23] also fits into a compositional modeling framework. Nayak's model fragments (which he calls *context-dependent behaviors*) are similar to those of Falkenhainer and Forbus. Model fragments are related by an approximation relation, which is domain-dependent (in general, the equations do not determine whether one model fragment approximates another). The model fragment library must contain approximation assertions.

The goal of Nayak's modeling procedure is to construct the simplest model that relates an input to an output variable, where "simpler" means, roughly, more approximate. To determine whether a model relates two quantities, Nayak computes the *causal ordering* of the model's parameters. Causal ordering is a method developed by Simon and extended by Iwasaki [16] that extracts causal relationships from inherently acausal equations. It treats exogenous parameters as uncaused, and asserts that one parameter causes another if the first determines the value of the second. The causal ordering of a model's parameters is a directed graph where the nodes are parameters and the links represent causation. To determine whether one parameter causes another, it suffices to find a path in graph connecting the two parameters.

The causal ordering graph is similar to, but not the same as, the interaction graph, because interactions are undirected. The causal ordering process essentially imposes a direction on each arc of the interaction graph.

Causal ordering is probably a better choice for answering the question of whether one quantity can influence another, but only under the additional assumption that exogenous variables cannot be changed by the system. Consider two quantities that are both influenced by a single exogenous variable; e.g., the flows in two pipes that are regulated by the same valve, whose position is exogenous. Neither flow can influence the other, since both are determined by the valve. The two flows are connected in the undirected interaction graph, but not in the directed causal ordering graph.

The input to Nayak's program is a component-like description of a device and a pair of input-output variables. Nayak shows that the decision version of this problem (is there a model of the device that can relate the variables?) is NP-complete, so it is likely that any algorithm that attempts to construct a simplest model is intractable.

Nayak goes on to define a special kind of approximation, called a *causal approximation.* If one model fragment is a causal approximation of another, then replacing the less detailed with the more detailed fragment in a model will only add paths to the causal ordering graph. Thus if all approximations in the model fragment library are causal approximations, the causal ordering graph for a simpler model is always a subset of that for a more complex model. This monotonicity property enables a polynomial-time algorithm for finding the simplest model. Nayak's algorithm composes the most detailed model fragments for each component of the device in question, then walks down the approximation lattice step by step, ensuring at each step that the resulting model satisfies the query and stopping as soon as it fails to do so.

The full definition of causal approximation is complicated, but one special case is when the approximation can be derived by setting an exogenous parameter of an equation to zero or infinity; these are what Dan Weld [31] calls *fitting approximations.* All of the approximation relationships between my models are of this form; e.g. removing a resistor element from a model is equivalent to setting the resistance to zero. Thus it should be possible to modify Nayak's polynomial-time algorithm to work with a system-dynamics representation.

The essential differences between Nayak's work and this thesis have all been considered above, in the discussions of the work of Williams, Falkenhainer and Forbus, and Rickel. These differences are: use of a component representation for input, instead of a geometric or mixed representation; redundancy and inexpressiveness of the model fragment representation; concentration on input-output connectedness rather than dynamic behavior; and the assumption that there is a most complex model for a device. These last two points are related: arbitrarily complex models of the sort described above for a shaft do not change the interaction graph for a model; they affect only transient dynamic behavior.

6.2 MM's Contributions

MM differs from previous work in automated modeling along several dimensions. In each of these dimensions, MM's success marks an important contribution to the field of automated modeling. In this section, I discuss MM's contributions; in the following section, I bound these contributions by discussing its limitations.

- *Use of geometry.* Although geometry has been used as an input representation for kinematics modeling programs, it has not been used for dynamics. MM is able to construct models for many fluid-mechanical systems from two-dimensional geometric information, along with minimal additional structural information such as general material composition. MM is able to perform this task because in many cases, geometric features strongly suggest the presence of certain phenomena.

- *Combining geometrical and component system descriptions.* MM is unique in that it can represent the structure system using a mixture of component and geometrical representations. This allows the user to present each piece of the system to the program in the clearest, most natural and least question-begging way.

- *Concern with dynamic behavior.* MM is unique among current automated modeling programs in that it attempts to match, at least qualitatively, the dynamic (time-varying) behavior of the systems it models. Current work focuses on the problem of ensuring merely that the model relates the desired inputs and outputs. As I argued above, the techniques used for this, such as interaction paths and the causal ordering, tend to produce zeroth-order models.

- *Generative model construction.* MM is almost unique ([35] is an exception) in that it does not have a fixed, finite set of models for each component in its library. Instead, MM generates models dynamically, and can generate an infinite number of models for a single component.

- *Use of system dynamics.* Despite the fact that system dynamics has been an important modeling framework for over forty years, and bond graphs have been used extensively for more than half that time, few workers in automated modeling have used these methods. To my knowledge, this is the first attempt at fully integrating the system dynamics perspective with AI techniques like qualitative reasoning, for the purpose of doing automated modeling. The energy-based modeling approach promoted by system dynamics isolates a small set of primitive elements that apply across energy domains, and restricts the form that models can take. This facilitated the writing of MM's rules and made it possible to deploy powerful analysis techniques like order determination.

  Moreover, because it is an implemented and cleanly structured program that builds models using a system dynamics approach, MM can be viewed as a precise codification of a portion of that approach.

- *Determining model problems without simulation.* Though many techniques for revising a model based on its predictions have been proposed in the automated modeling community [31, 29, 10], these techniques all involve simulation. MM is unique in that it can analyze a behavior and a model and, in some cases, determine that the model is flawed without conducting a simulation. It can determine that the model's order is too low to account for the given behavior, or that the model requires a resistor. Moreover, these two problems map well to the addition of particular modeling elements. Both MM's ability to make these judgments about model suitability, and the correspondence between model problems and their repairs, are benefits of the system dynamics approach.

- *Behavior-constrained qualitative simulation.* When simulation is necessary, MM uses a new and efficient qualitative simulation technique, embodied in the QSIM-CHECK program, that constrains a simulation to match a given behavior.

6.3 Limitations and Future Work

Despite its abilities, MM has a number of limitations. Some of these are relatively minor and could be removed without serious damage to the overall framework of the program. Other limitations are more profound.

6.3.1 Expressiveness of the Modeling Language

MM is limited in the kind of models it can create. It assumes that the systems are lumpable, i.e. that they do not require partial differential equations to be adequately modeled. MM works only in the fluid, mechanical translation, and mechanical rotational energy domains. MM also assumes its systems are passive, continuous, have no relevant kinematic behavior and have only one-dimensional dynamics. Fluids are assumed incompressible. Leakage is assumed not to occur. The modeling elements MM uses cannot have time-varying characteristics, and cannot be modulated by external signals (this problem showed up in the poor representation of the bare model for the hydraulic valve, which should contain modulated elements). Furthermore, MM assumes that the equations of the model do not change over time, or to put it another way, that the structure of the system does not change.

Some of these limitations could be removed without great difficulty. MM could be extended to include thermal, electric and electromagnetic domains. MM could be incorporated into a program that determined structural changes and re-invoked MM for each distinct system structure. The incompressibility and no-leakage assumptions could be removed by the addition of suitable rules.

Some of these built-in assumptions, while they could be removed in principle, enormously expand the search space of models. For instance, allowing modulated elements, i.e. signals, into the class of models means that a resistor's characteristic could vary arbitrarily over time in response to other quantities in the system or to exogenous factors. This fact would make model analysis extremely difficult.

6.3.2 Quantitative Information

My decision to focus on qualitative behavior and qualitative properties of systems had the advantage of keeping the research scope tractable, and it does provide a considerable generality: when MM models a U-tube, for instance, it is really modeling an infinite set of systems of different shapes and sizes. However, I now consider the use of qualitative information alone unacceptably restrictive.

Adding quantitative information would have a number of benefits. If the input behavior to MM were an actual (or imagined) quantitative trace of the system's behavior, then it would contain much richer clues as to the nature of the model needed to represent it. We might be able to glean information about the model order for orders higher than two, for instance. If the system were excited with a periodic input and the input were of a different frequency, we could conclude that a linear model would be inadequate. (A linear model has the property that its output has the same frequency as its input.) We could also get a sense of which aspects of the behavior were significant and which were not. If the user supplied a range of times of interest to the program (e.g. from one millisecond to 10 seconds), then behavior that fell outside these bounds (like a rapid, initial overshoot) could be ignored, enabling a simpler model to be constructed. Currently, the user must effectively make a modeling decision when encoding the behavior of the system qualitatively. The great challenge in using a quantitative behavior representation is noise: it can be difficult to distinguish noise from actual system behavior.

Another use for quantitative information lies in providing MM with material properties of the objects in the system. If MM knew the materials and dimensions of the U-tube, it would have better information for making decisions about whether to model a wall as rigid or flexible, or a fluid as having resistance. Replacing the three composition types `solid`, `flexible` and `fluid` with quantitative information would also cut down on the modeling decisions the user must make.

MM's analysis phase would be facilitated by quantitative models. Quantitative models, though less general than qualitative ones, are easier and faster to simulate. They are also deterministic. In addition to making simulation intractable, non-determinism also gives rise to the fact that qualitative simulation produces physically impossible behaviors. This is a fundamental flaw in MM's analysis phase: MM has no way of knowing whether a model predicts a given behavior because it truly gives rise to that behavior, or because the qualitative simulator generated a spurious prediction. Another, related weakness is that an overly general model may be created: for instance, a model that predicts every behavior will certainly predict the given one. This problem is largely mitigated in MM because its most common cause, lack of interaction between the input and output, is checked for explicitly before simulation. However, other, more subtle cases of overgeneral models can arise, and deterministic models would completely eliminate the problem.

The presence of quantitative information opens the door to an important line of work that has been neglected in the qualitative reasoning literature: determining when linearity is a valid assumption. The advantages of working with a linear model cannot be overestimated: the potential behaviors are few and easy to predict, the rich and powerful mathematical theory of the frequency domain becomes available, system identification is greatly facilitated, and high-quality control systems are easy to design. But linear models are not always appropriate. One exciting area for future work is developing techniques for recognizing, from the system and the goals of the model, whether a linear model will do the job.

6.3.3 Geometry

MM uses only 2D geometry. This was a decision made to facilitate research and is not an inherent limitation of the ideas. Since reasoning in three dimensions tends to get complex quickly, a reasonable intermediate approach is to use a $2\frac{1}{2}$ representation, in which each part is described as a set of 2D slices. Increasing the dimension of the representation should probably go hand-in-hand with increasing the expressible dimensions of the modeling language. For instance, 3D reasoning about rotational motion would require MM to treat rotational inertia as a 3-by-3 matrix rather than a scalar.

Another improvement to MM's geometric reasoning abilities would be to incorporate some basic kinematic analysis, for instance, determining the axes of motion of parts directly from the geometry. Joskowicz and Sacks perform this computation [17].

6.3.4 Explicit Assumptions

One severe deficiency of MM, in comparison with other recent work in automated modeling, is its inability to reason about its own assumptions and retract them when necessary. MM already carries out the retraction process in a sense: whenever it adds, say, capacitance to a solid object, it is retracting the assumption of rigidity; or when it adds resistance to a fluid region, it can be viewed as retracting the assumption of inviscid flow. But not all assumptions correspond to the addition or removal of a single bond graph element. For instance, if fluid density varies greatly then the simple treatment of fluid used by MM is inadequate and a more detailed approach that takes density into account is required. One challenge is to give MM the means to reason about when these shifts in modeling detail are required.

6.3.5 Component Models

MM's generative notion of component models can profitably be combined with the compositional modeling framework. The organizing principles of system dynamics can lead to a highly factored and modular model library. For those components that have an unbounded number of models, the models all have a common form, so it should be possible to generate them automatically from a single, concise description.

6.3.6 Control Structure

MM's simple control structure is a weak point. Breadth-first search will indeed produce simpler models before more complex ones, but will do so very slowly. A more aggressive strategy, coupled perhaps with a mechanism for *removing* elements from a model, could perhaps produce the same results more quickly. One such aggressive search strategy is suggested by MM's two-part analysis phase. A single model could be elaborated until it successfully passed the low-order and needs-resistor tests, and only after it failed in simulation would it be put aside temporarily to consider another model. Because the special-case tests are fairly quick, this control structure would provide rapid solutions for simple problems. A similar strategy would use best-first search and order models by the type of problems they have, preferring those that did not have the `wrong-behavior` problem. Another search strategy would pursue a single model for some number of modifications, giving up on it when it exceeded some complexity threshold.

As the search strategy becomes more aggressive, a problem we could call "garden-pathing" becomes more acute. It is possible that a model is generated which is adequate, but is more complex than necessary; the search procedure has been led down a garden path. To cope with the difficulty, an element-removal phase could be added to the modeling process. After a model is deemed successful, elements would be removed from it and the resulting, simpler model would be retested to see if it, too, was adequate. This two-part strategy might be able to generate complex models more quickly than MM's plodding, breadth-first approach.

6.3.7 Knowledge

MM could incorporate a great deal more knowledge about physics, engineering and the modeling process. I have already discussed the advantages that would accrue to providing quantitative information. Here I want to consider some other possibilities.

Exploiting the Bond Graph

MM could more effectively exploit the information contained in a model's bond graph. Instead of using the overall model order in its low-order tests, MM should pay more attention to the particular variables designated as outputs. It may be, for instance, that a variable's behavior is zeroth-order—directly constrained by the inputs—even though the model containing it is second-order. It should be possible to use bond graph causality techniques to make this more fine-grained order determination.

Causality can also help in determining how to modify a model to correct its faults. For instance, if a model's order is too low because one or more of its energy-storage elements is dependent, the bond graph causality assignment can be used to isolate parts of the model that can be augmented in order to break the dependency. If the order is too low because of a lack of energy-storage elements, causality information can again be used to isolate parts of the bond graph where inserting an element would result in its being dependent.

One rather simple idea that could be used to cut down on MM's search space is that of *equivalence regions* of a bond graph. Consider a 1-junction J and all its bonds. The junction and bonds form a 1-equivalence region, meaning that an element connected to a 1-junction would result in the same overall behavior regardless of where in the equivalence region it were connected. For instance, adding a resistor directly to J results in the same model as splitting one of J's bonds and adding a new 1-junction with a resistor attached. Thus only one of the possibilities needs to be tried. MM does something like this in its analysis phase when it tests whether the candidate model is behaviorally equivalent to previous models by testing for bond graph isomorphism: the isomorphism routine performs graph simplifications that exploit equivalence regions. But MM would do better to never construct the equivalent models in the first place.

Teleological Information

Currently, MM will hypothesize leakage at each possible point in the table-bed model. If MM could realize, or be informed, that the hydraulics were designed to avoid or minimize leakage, it could avoid generating a great many models. (Of course, if one were using MM for diagnosing a faulty system, one would not want to do this, since hydraulic leakage might well be the fault.) More generally, MM incorporates no knowledge about the purpose of the systems it is modeling, and therefore can make no assumptions about what sorts of phenomena might plausibly be excluded. Note that MM already possesses teleological information about component types, in that a component's bare model provides the ideal behavior of the component. For instance, an ideal DC motor is simply a gyrator. However, MM lacks information about the system being modeled, whether that is specific information about a part of the system (e.g. "this particular DC motor is so good that you can ignore power loss") or global information about the whole system (e.g. "no leakage"). Representing and using this information is not difficult when it corresponds directly to particular rules or classes of rules, as do the examples I have given here. But how to deal with other sorts of teleological information (e.g. "this system was designed to respond rapidly") is an open research issue.

Appendix A

Rules

A.1 Interpretation Rules

Interpretation rules were written only when needed by the examples. It is quite easy to add new interpretation rules.

position→flow-integ/fluid (*A position is the integral of a flow*)

IF	there is a position quantity whose changing point is linked to a fluid segment
THEN	associate a 1-junction with that segment
AND	consider the quantity to be the integral of the junction's flow.

position→flow-integ/trans (*A position is the integral of a flow*)

IF	there is a position quantity whose changing point is linked to a rigid or flexible segment
THEN	associate a 1-junction with that segment
AND	consider the quantity to be the integral of the junction's flow.

position→flow-integ/trans (*An applied force is an effort source*)

IF	there is an external force that acts on a segment
THEN	associate a 0-junction with that segment
AND	consider the force's magnitude to be the junction's effort.

A.2 Geometric Rules

These rules examine the geometric parts of the structural description.

A.2.1 Resistors

resistance-narrow/fluid (*Add a fluid resistor to a narrow fluid region*)

IF	there is a fluid segment S
AND	the average width of S is significantly less than that of the adjacent fluid segments
THEN	add a resistor to the model and set its region equal to S.

resistor-1j/fluid (*Add a fluid resistor to a 1-junction*)

IF	there is a 1-junction in the model with no resistors attached
THEN	add a resistor to the 1-junction.

resistor-new/fluid (*Add a fluid resistor in a region with no 1-junction*)

IF	there is a contiguous region of fluid with no 1-junction
THEN	place a 1-junction in that region and note that the 1-junction represents the region
AND	add a resistor to the 1-junction.

A.2.2 Capacitors

capacitor-gravity/fluid (*Propose a capacitor in a partially contained fluid region that admits volume change and that is subject to gravity*)

IF	there is a gravitational force F
AND	there is a region R containing fluid,
AND	R is partly bounded by solid on the bottom,
THEN	add a capacitor to the model and set its region equal to R.

capacitor-movement/fluid (*Propose a capacitor in a partially contained fluid region into which a solid can move*)

IF	there is a solid region S
AND	S is free to move into and away from a fluid region R
AND	R is partly bounded by solid opposite and adjacent to S
THEN	add a capacitor to the model and set its region equal to R.

capacitor-force/trans (*Propose a capacitor in a flexible region that can move in opposition to a force*)

IF	there is a force F
AND	there is a rectangle R containing flexible material such that the material can expand or contract in the direction of F,
THEN	add a capacitor to the model and set its region equal to the flexible portion of R.

A.2.3 Inertias

inertia-1j/fluid (*Add a fluid inertia at a 1-junction*)

IF	there is a 1-junction in the model with no inertias attached
THEN	add an inertia to the 1-junction.

inertia-new/fluid (*Add a fluid inertia in a region with no 1-junction*)

IF	there is a contiguous region of fluid with no 1-junction
THEN	place a 1-junction in that region and note that the 1-junction represents the region
AND	add an inertia to the 1-junction.

inertia/trans (*Propose an inertia in a solid object in response to a force.*)

IF	there is a rigid or flexible segment S
AND	S is not fixed
AND	S contains no inertias
AND	there is a force acting on S
THEN	add an inertia to S.

A.3 Component Rules

MM's component rules are all extremely simple. There is a single rule for interpreting component quantities, which determines the interpretation from the component definitions. Four other rules add components and junctions to an existing bond graph by looking for *element addition points,* places in the bond graph where new elements could be placed irredundantly. (E.g. placing an element on a junction that already contains an element of that type is redundant.) One of the four component addition rules is given below; the other three are similar.

component-add-resistance (*Add a resistance to a component model*)

IF	there is a component C
AND	there is an element addition point for a 1-junction and a resistor
THEN	add the resistor at the element addition point.

Bibliography

[1] Sanjaya Addanki, Roberto Cremonini, and J. Scott Penberthy. Reasoning about assumptions in graphs of models. In *Proceedings of the IJCAI*, 1989.

[2] J. J. Beaman and R. C. Rosenberg. Constitutive and modulation structure in bond graph modeling. *Journal of Dynamic Systems, Measurement, and Control*, 110:395–402, 1988.

[3] Peter C. Breedveld. *Physical Systems Theory in terms of Bond Graphs.* PhD thesis, Twente University, 1984.

[4] Alan Bundy. Will it reach the top? Prediction in the mechanics world. *Artificial Intelligence*, 10:111–122, 1978.

[5] Dennis DeCoste. Dynamic across-time measurement interpretation. *Artificial Intelligence*, 51:273–341, 1991.

[6] Johan deKleer. Qualitative and quantitative knowledge in classical mechanics. Technical Report 352, MIT AI Lab, 1975.

[7] Richard Doyle. Hypothesizing device mechanisms: Opening up the black box. Technical Report 1047, MIT AI Lab, June 1988.

[8] R. J. Duffin. Impossible behavior of nonlinear networks. *Journal of Applied Physics*, 26(5):603–605, May 1955.

[9] Brian Falkenhainer and Kenneth D. Forbus. Setting up large-scale qualitative models. In *Proceedings of the AAAI*, 1988.

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

[11] Boi Faltings. Qualitative kinematics in mechanisms. In Daniel S. Weld and Johan de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems.* Morgan Kaufmann, San Mateo, CA, 1990.

[12] Ken Forbus. Qualitative process theory. *Artificial Intelligence*, 24, 1984.

[13] Kenneth D. Forbus. Interpreting observations of physical systems. In Daniel S. Weld and Johan de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, San Mateo, CA, 1990.

[14] Andrew Gelsey. Automated reasoning about machine geometry and kinematics. In Daniel S. Weld and Johan de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, San Mateo, CA, 1990.

[15] Choon P. Goh. Model selection for solving kinematic problems. Technical Report 1257, MIT AI Lab, August 1990.

[16] Yumi Iwasaki and Herbert A. Simon. Causality in device behavior. *Artificial Intelligence*, 29, 1986.

[17] Leo Joskowicz and Elisha Sacks. Computational kinematics. *Artificial Intelligence*, 51:381–416, 1986.

[18] Dean Karnopp and Ronald Rosenberg. *System Dynamics: A Unified Approach*. John Wiley & Sons, New York, 1975.

[19] Benjamin Kuipers. Qualitative simulation. In Daniel S. Weld and Johan de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, San Mateo, CA, 1990.

[20] Pat Langley, Herbert A. Simon, Gary L. Bradshaw, and Jan M. Zytkow. *Scientific Discovery: Computational explorations of the creative processes*. MIT Press, Cambridge, MA, 1987.

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

[22] Seshashayee S. Murthy and Sanjaya Addanki. Prompt: An innovative design tool. In *Proceedings of the AAAI*, 1987.

[23] P. Pandurang Nayak. Causal aprroximations. In *Proceedings of the AAAI*, 1992.

[24] P. Pandurang Nayak, Leo Joskowicz, and Sanjaya Addanki. Automated model selection using context-dependent behaviors. In Benjamin Kuipers, editor, *Fifth International Workshop on Qualitative Reasoning about Physical Systems*, Austin, Texas, 1991.

[25] Gordon Novak. Representations of knowledge in a program for solving physics problems. In *Proceedings of the IJCAI*, 1977.

[26] J. Scott Penberthy. Incremental analysis and the graph of models: A first step towards analysis in the plumber's world. Master's thesis, MIT, 1985.

[27] Jeff Rickel. Automated modeling for answering prediction questions: Exploiting interaction paths. Technical Report AI92-178, University of Texas at Austin, 1992.

[28] R. C. Rosenberg and J. Beaman. Clarifying energy storage field structure in dynamic systems. In *Proceedings of the American Control Conference*, 1987.

[29] Mark Shirley and Brian Falkenhainer. Explicit reasoning about accuracy for approximating physical systems. In *Workshop on the Automatic Generation of Approximations and Abstractions*, 1990.

[30] Karl Ulrich. Computation and pre-parametric design. Technical Report 1043, MIT AI Lab, September 1988.

[31] Daniel S. Weld. Automated model switching: Discrepancy driven selection of approximation reformulations. Technical Report 89-08-01, University of Washington, Seattle, October 1989.

[32] Brian Williams. *Interaction-based Invention.* PhD thesis, MIT, 1990.

[33] Brian Williams. Interaction-based invention: Designing novel devices from first principles. In *Proceedings of the AAAI*, 1990.

[34] Brian C. Williams. Capturing how things work: Constructing critical abstractions of local interactions. In *Workshop on the Automatic Generation of Approximations and Abstractions*, 1990.

[35] Bruce H. Wilson and Jeffrey L. Stein. An algorithm for obtaining minimum-order models of distributed and discrete systems. In Brian Falkenhainer and Jeffrey L. Stein, editors, *Automated Modeling Workshop of the Winter Annual Meeting of the American Society of Mechanical Engineers (ASME WAM), DSC-Vol. 41*, 1992.