Automated Modeling of Physical Systems in the Presence of Incomplete Knowledge[1]

Adam Farquhar

October 27, 1993

[1]The author's current address is: Knowledge Systems Lab, Stanford University, 701 Welch Rd., Bldg. C, Palo Alto, CA 94303. Email may be sent to: Adam_Farquhar@ksl.stanford.edu. This work has taken place in the Qualitative Reasoning Group at the Artificial Intelligence Laboratory, The University of Texas at Austin. Research of the Qualitative Reasoning Group is supported in part by NSF grants IRI-8905494 and IRI-8904454, by NASA grant NAG 2-507, by the Texas Advanced Research Program under grant no. 003658175, and by the Jet Propulsion Laboratory.

Abstract

This dissertation presents an approach to automated reasoning about physical systems in the presence of *incomplete knowledge* which supports formal analysis, proof of guarantees, has been fully implemented, and applied to substantial domain modeling problems. Predicting and reasoning about the behavior of physical systems is a difficult and important task that is essential to everyday commonsense reasoning and to complex engineering tasks such as design, monitoring, control, or diagnosis.

A capability for automated modeling and simulation requires

- expressiveness to represent incomplete knowledge,
- algorithms to draw useful inferences about non-trivial systems, and
- precise semantics to support meaningful guarantees of correctness.

In order to clarify the structure of the knowledge required for reasoning about the behavior of physical systems, we distinguish between the *model building* task which builds a model to describe the system, and the *simulation* task which uses the model to generate a description of the possible behaviors of the system.

This dissertation describes QPC, an implemented approach to reasoning about physical systems that builds on the expressiveness of Qualitative Process Theory [Forbus, 1986] and the mathematical rigor of the QSIM qualitative simulation algorithm [Kuipers, 1986].

The semantics of QPC's modeling language are grounded in the mathematics of ordinary differential equations and their solutions. This formalization enables the statement and proof of QPC's correctness. If the domain theory is adequate and the initial description of the system is correct, then the actual behavior of the system must be in the set of possible behaviors QPC predicts.

QPC has been successfully applied to problems in Botany and complex examples drawn from Chemical Engineering, as well as numerous smaller problems. Experience has shown that the modeling language is expressive enough to describe complex domains and that the inference mechanism is powerful enough to predict the behavior of substantial systems.

Contents

Chapter 1

Introduction

Predicting and reasoning about the behavior of physical systems is a difficult and important problem for any intelligent agent. It is as essential to everyday commonsense reasoning as it is to complex engineering tasks such as design, monitoring, control, or diagnosis. Knowledge about physical systems in both the everyday and engineering worlds is often incomplete. It is not only the case that relationships are complex and nonlinear with actual conditions incompletely known, but an agent may even be unaware of the complete set of objects and topology of a system.

1.1 Goals of this research

This dissertation presents an approach to and tools for automated reasoning about the behavior of physical systems in the presence of incomplete knowledge. This includes incomplete knowledge about parameter values, coefficients, and functional forms that is supported by current qualitative simulators. It also includes incomplete knowledge about the entities that exist in the world, the relations among them, the physical phenomena that affect them, and the appropriate choices for modeling them.

Reasoning about physical systems can be decomposed into a *model building* task, which builds a model to describe the system, and a *simulation* task, which uses the model to generate a description of the possible behaviors of the system.

In support of this claim, we present QPC, an implemented approach to qualitative reasoning that describes a physical situation with a set of model fragments employing influences and algebraic constraints in a style based on and extending qualitative process theory (QP theory) [Forbus, 1984b]. QPC composes model fragments into models that uniquely specify an initial value problem suitable for solution by the QSIM qualitative simulation engine [Kuipers, 1986], which generates behavioral predictions. In case predicted behavior extends across the boundary of a single model's applicability, a new model of the resulting situation is constructed. Figure 1.1 illustrates the relation between the input and output of QPC and the world.

QPC reduces incomplete knowledge about a situation by making use of a general purpose

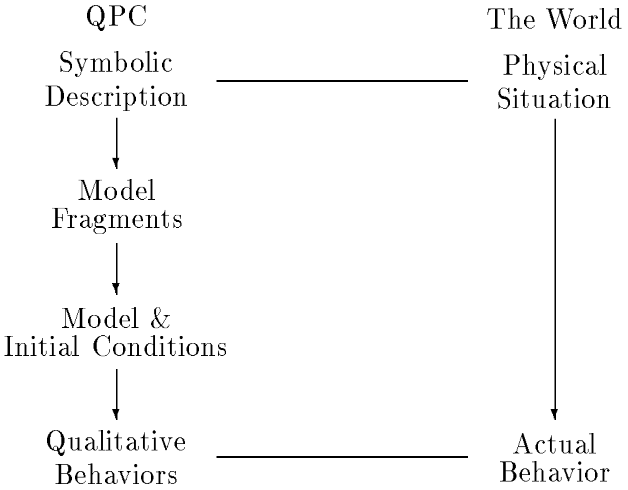

Figure 1.1: Abstract characterization of QPC. The symbolic description of the physical situation is used to identify a set of model fragments. These fragments are composed into a *model*. The model, together with a set of initial conditions define a qualitative initial value problem, which is solved to yield a set of qualitative behavioral predictions.

knowledge representation system to encode a theory of the domain. Thus, aspects that might be missing from the initial symbolic description of the situation may be inferred.

QPC handles incomplete knowledge about the structure of the situation by building several possible models to describe it. QPC handles incomplete knowledge about functional forms, coefficients, and values by employing qualitative differential equations (QDEs) and qualitative values.

QPC guarantees that if the domain theory is adequate, the correct model of the system will be in the set of constructed models. The qualitative simulator guarantees that if the initial value problem is an abstraction of the actual system, the actual behavior will be in the set of possible ones that are generated. Thus, given an adequate domain theory and description of the initial situation, QPC guarantees that the correct model and actual behavior will be among the ones generated.

QPC addresses the problem of building and simulating models of a system to predict its behavior. Solutions to this problem are critical in many important tasks such as diagnosis, monitoring, exploratory design, design verification, operating procedure design, hazard/operations analysis, and tutoring. QPC is currently being employed as an analysis module in the question-answering component of a tutoring system and in a hazard/operations analysis system.

1.2 Contributions

QPC makes several contributions to solving problems of automated reasoning about physical systems and to qualitative reasoning in particular.

Hybrid architecture QPC employs a hybrid architecture that enables the qualitative reasoning problem to be decomposed into the distinct tasks of model building and simulation. This allows the semantics of the modeling language to be precisely and comprehensibly defined. Each model entails a set of (qualitative) initial values and a QDE. The QDE representation and its solutions have a well understood semantics grounded in the theory of ordinary differential equations. Thus, the semantics of the modeling language is reduced to that of qualitative mathematics, and in turn to that of mathematics. This decomposition and reduction has a number of important consequences.

- It clarifies the distinction between influences and constraints, and the role of the closed world assumption in modeling.
- It allows a precise semantics for influences and correspondences to be defined.
- Both the model and the QDE are objects that can be explicitly reasoned about in first order logic, rather than being implicit in, for instance, the process of interpretation construction.
- Perhaps most importantly, the decomposition allows QPC, and qualitative reasoning generally, to benefit from independent advances in modeling and simulation.

The expertise, knowledge, algorithms, and data structures required to build a model are very different from those required to simulate one. Building a model requires detailed knowledge of the physical phenomena that may occur, the layout and structure of the system. This is primarily a problem of knowledge acquisition and representation. Simulating a model is primarily a mathematical endeavor.

QPC's hybrid architecture enables it to exploit independent advances in model building and simulation. For example, I developed a technique for performing relevancy reasoning by ignoring influences. It was implemented with a simple declarative description in the model builder's knowledge base. QPC has also exploited advances in qualitative simulation techniques such as attainable envisionments and "chatter" abstraction.

Incremental algorithm QPC is the first completely incremental compositional modeling engine. In contrast to engines such as QPE that generate total envisionments, the set of entities and the topology of the physical situation may change during the course of analysis, as may the granularity of the model, the modeling assumptions, and even the domain theory itself. This is of considerable practical interest. Information about the initial configuration of a system is often available, and drastically reduces the set of possibilities.

Statement and proof of correctness We prove that the QPC algorithm generates all possible behaviors of a scenario given its initial configuration. Establishing this property is challenging because

- the description of the initial situation may be incomplete,
- the behavior may cross through a number of different regimes (models), and
- the configuration of the situation may change over time.

Neither the statement of the theorem, nor its proof, would have been possible without the clear separation between modeling and simulation, the mathematical clarity and rigor of the QSIM QDE representation, and the soundness proof of the QSIM algorithm [Kuipers, 1986].

Synthesis of research programmes QPC seamlessly knits together the powerful expressive qualitative process approach to model building due to Ken Forbus, and the elegant rigorous work on qualitative simulation due to Ben Kuipers. Their two systems, QPE and QSIM, have had tremendous impact on the qualitative reasoning community.

Extends and formalizes QP theory QPC extends, clarifies, and formalizes many of the concepts and constructs introduced in QP theory.

QPC extends the modeling language introduced in qualitative process theory (QP theory) and removes some restrictions from it. There is no distinction between *processes* and *views*, as a single form is used to define all model fragments.[1] Arbitrary cycles in the directed graph of influences are allowed. Several new types of compositional equations (influences) are introduced; combinations of different types of influences on a single variable are mathematically well defined; "open world" influences may be combined with "closed world" constraints. The derivative of a variable is also explicitly representable, and is treated like any other variable.

The semantics of influences, correspondences, and constraints are clearly formalized in terms of QDEs and in turn standard mathematics. Each trajectory is an alternating sequence of states representing time points and open intervals between them. Models always hold over a *closed* interval of time, with a well defined beginning and ending. The QP theory problem of *stutter*, in which there is a series of models, each holding for only an instant, cannot occur.

In a fundamental ontological shift from previous approaches based on QP theory, QPC takes the stance that model fragments contribute pieces of equations, rather than representing cognitively motivated concepts. Thus, in contrast to the standard QP theory representation, in QPC it is not appropriate (although it is possible) to define a model fragment for fluid flow that is directed from source to destination. The equations that govern fluid flow apply regardless of the direction of flow, and even

[1] In QPE, Forbus' implementation of QP theory, the internal representation of the two also made no distinction.

apply when there is no flow. Consequently, a single model fragment should cover all of these cases.

QPC separates the cognitively compelling stories that one might want to tell about scenario from the mathematical models thereof. Such stories may also be readily derived from the QPC models by looking at the set of model fragments and parameter values together. There are many stories that one might want to tell about the same scenario, but there is no reason to rederive the facts of the matter.

Enables new research QPC is designed as a tool that can be used not just to pursue research on model building, but to support other performance tasks. It is currently being used in several different research efforts, two of which are briefly described in Section 5.3 and Section 5.4 of this dissertation. Rickel is using QPC as an analysis module for a botany tutoring system, and Catino is using QPC to generate behavioral descriptions in a chemical engineering hazard and operations analysis task.

The framework that QPC uses to reason about change may have important implications for knowledge representation and non-monotonic logic. The decomposition of the reasoning task into modeling and simulation may be used to distinguish between different sorts of default assumptions. Modeling assumptions have priority over ones that arise during simulation. Crawford and Etherington [Crawford and Etherington, 1992] have shown how this approach can be used to solve a number of non-monotonic reasoning problems that have proven difficult using other techniques.

1.3 An Example Illustrates the Process

As an example we will look at the simple physical situation depicted in figure 1.2. There is a double kitchen sink with two basins A and B, which have drains DA and DB respectively. There is a movable faucet F, which is located above the basins. The faucet is on. The problem is to model this situation and predict what happens.

Notice that much has been left unsaid. We have not specified the exact sizes or shapes of the sink basins, nor have we specified the dimension or location of the drains. We have not even specified the precise location of the faucet. Nonetheless, most people could reasonably predict that if the faucet is over A, then A will fill up until the inflow reaches equilibrium with the outflow from the drain. Another possibility is that the faucet is over B. A third possibility is that the faucet is between A and B, and that the water will pour into both of them.

In this section, we will show how this situation can be modeled and how behaviors are predicted using QPC. The example will illustrate each of the inferential steps and introduce several of the main issues. The model presented here has been pared down for the purpose of illustration. Catino has developed a much richer domain theory for the Chemical Engineering phenomena including multi-phase, multi-component flows, as well as devices such as condensers and flash-evaporators [Catino, 1993]. Section 5.4 summarizes and discusses this work.

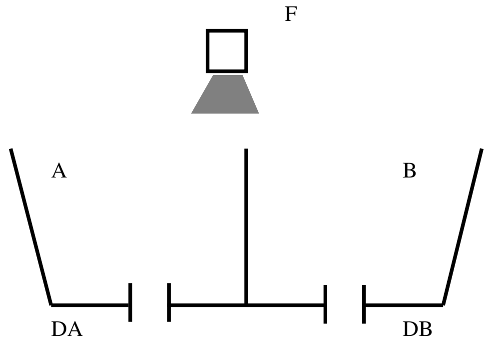

Figure 1.2: A simple physical situation — A double kitchen sink with basins A, B, and movable faucet F above them.

1.3.1 The Scenario Description

In order to reason about this situation, QPC needs a symbolic description of it. The symbolic description of the relevant objects and relations is the *scenario*. From all of the objects that are known to exist in the world, the scenario selects a small subset to be the entities that are (initially) of interest. The scenario also specifies many of the inter-relationships between these entities, as well as restrictions on the variables that may be used to describe them. Finally, the scenario description may specify constraints on the possible behaviors that will be considered. The scenario description is used as a starting point. The domain theory may sanction many additional inferences during the course of analysis.

```
(defscenario two-sinks
    :entities ((f :type sources)
               (a :type containers) (da :type drains)
               (b :type containers) (db :type drains))
    :relations ((considering  geometry)
                (above  f a) (above  f b)
                (fluid-connected  a da)
                (fluid-connected  b db)
                (= zero (left-side a))
                (= (right-side a) (left-side b))
                (< (left-side a)  (position f))
                (< (position f)   (right-side b))))
```

The preceding `defscenario` form defines a scenario for the double sink. The definition has two sections. In the first, the entities are introduced – the faucet F, the containers A and B, and the drains DA and DB. The second portion asserts relations that are true throughout the scenario. The drains are associated with their respective containers by asserting that there is a fluid connection between them. We assert that we are interested in some (simple) geometric properties in this scenario, and we assert some of them: that the faucet is above the sink, that the two basins are adjacent, and that the faucet is somewhere between the leftmost and rightmost points of interest. It is also possible to assert initial conditions.

1.3.2 The Domain Theory

The scenario definition makes use of a particular vocabulary for describing objects and relations between them. This vocabulary is defined in the domain theory, which consists of three parts: an ontology, a model fragment library, and modeling knowledge about how to build models in the domain. We will examine each of these in turn.

1.3.2.1 The Ontology

The types of objects that are particular to the domain are asserted in a taxonomic hierarchy.

```
(defTaxonomy
    (objects
     (containers)
     (sources)
     (drains)
     (liquids)))
```

There are four types of objects: containers, sources, drains, and fluids. A *type* is equivalent to a set of all elements of that type. The plural is used to indicate a set, which may be thought of as a type as well.

Next, the domain theory introduces logical relations used to describe objects.[2] Relations defined by the domain theory are sometimes called *structural relations*, because they help to define the structure of the world.

```
(defRelation contents  (containers liquids))
(defRelation container (liquids containers))
```

These two relations express that a container may have a contents, that is a liquid, and that the inverse of `container` is `contents`.

```
(defRelation fluid-Connected (objects objects))
```

The relation `fluid-connected` holds between any pair of objects that are connected in such a way that fluid may flow from the first to the second. This is a rather coarse abstraction, but is adequate for our example. Note that the connectedness of two objects need not be an universally true statement. Even in a simple domain theory, one might want to allow for connections to be constructed and destroyed. E.g. if basin A were allowed to overflow, there would be a new fluid connection between it and the adjacent basin B; if the faucet were initially over A, and moving to the right, the fluid connection to A would be broken, and a new connection to B constructed.

```
(defRelation over  (objects objects))
(defRelation above (objects objects))
```

[2]In order to simplify the exposition, the *scope* argument to these relations has been omitted. See Section 3.2.

Two very simple geometric relations are expressed using `over` and `above`. An object `F` is above another object `A` if the bottom-most point of `F` is above the top-most point of `A`. `F` is over `A` if something dropped from `F` would land on `A`.

```
(defRelation considering (scopes choices) :type scoped)
```

The `considering` relation allows for some simple modeling information to be expressed. The domain theory defines a set of *choices* to explicitly represent various modeling choices. In this example the set only contains `geometry`, but it might also include thermal, electrical, or magnetic properties, etc.. By conditioning model fragments on the appropriate choices, QPC implements a simple sort of relevancy reasoning.

Once the relations are defined, axioms may be written. Axioms are specified with `defRule`. The following form states that if a source is over a container then there is a fluid-connection between them.

```
(defRule
  (over ?scope ?source ?can)
  (isa ?source fluid-sources)
  (isa ?can containers)
  ->
  (fluid-connected ?scope ?source ?can))
```

This simple rule could be made more realistic by adding a condition that the path between the source and container is not blocked. One of the most difficult issues in representing knowledge about the infinitely complex real world is to identify the set of qualifications that must be considered when drawing any inference. Every symbolic representation is a coarse abstraction of reality and must leave out many potentially important conditions. The general issue is known as the *qualification problem*. The implicit representational decisions that are made by the domain theory author to address this problem are not reasoned about by QPC.[3]

The only remaining part of the ontology is to specify the quantities which may be used to describe objects. A *quantity* is simply a real-valued variable or constant. The `defQuantity` form is used to define quantities. In the typical case, the `defQuantity` form defines a function which returns the desired variable. E.g.

```
(defQuantity mass objects :sign :non-negative)
```

defines a function `mass` that returns the variable whose value is the mass of an object. The following specifies the remaining quantities in this ontology:

[3]While QPC's theory is based on monotonic inference, the Algernon representation system that is employed in the current implementation allows nonmonotonic inferences to be made. As a consequence, qualifications may be expressed via nonmonotonic inference rules.

```
(defQuantity pressure  liquids)
(defQuantity level     liquids)
(defQuantity flowrate  objects)
(defQuantity flowrate-into (objects liquids))
(defQuantity position   objects)
(defQuantity left-side  containers)
(defQuantity right-side containers)
```

In this ontology, the `pressure` and `level` are associated with a (contained) liquid. It would also be reasonable to associate these with a container. The quantity `flowrate-into` represents the flow from an object into a container. The taxonomy has been pared down for the example; it might be more correct to associate `position` with point objects and left and right sides with extended objects.

1.3.2.2 The Model Fragment Library

The second component of the domain theory is the library of abstract model fragments. The abstract model fragments comprise the primary part of the domain theory. They specify the phenomena that can affect objects and the ways that objects can be described.

Just as there are certain constraints that are particularly useful in constructing qualitative models (e.g. monotonic function), there are certain relations that are particularly convenient for asserting fragments of information that can be composed into constraints. These *compositional relations* include *influences* of one variable on another. Asserting X is positively influenced by Y, written X Q+ Y, means that in the absence of countervailing influences, an increase in Y causes an increase in X. The influence relationship is inherently causal, and by convention (introduced by Forbus) the causal arrow is directed from right to left. As an example, an increase of the amount of liquid in a container will cause the level to rise, unless there is a countervailing influence, such as a simultaneous increase in the diameter. This relationship is expressed by writing: `level Q+ amount`. In a model fragment definition, it would appear as: `(Q+ (level can) (amount can))`.

An important first step in formalizing influences is to recognize that an influenced variable, X, is functionally determined by the variables, Y_i, that influence it, and that the type of the influence determines the sign of the partial derivative of the function with respect to that variable. For example,

$$X\ Q^{+}\ Y_i \ \equiv\ x = f(\ldots Y_i \ldots) \text{ and } \frac{\partial f}{\partial Y_i} > 0. \tag{1.1}$$

The partial derivative captures precisely and mathematically the intuition of Forbus' *all else being equal*. Section 3.4.2 presents a complete and precise definition for influences that relaxes the restrictions of the original QP theory definitions and eliminates the ambiguities latent in the above definition. QPC also defines a number of other influences including `q-add` and `q-sub` that are additive and subtractive.

In the terminology of QP theory, the influences described above are all *indirect* influences. QP theory also provided a *direct* influence, written X I+ Y. This asserts that if there are no other countervailing influences, positive Y causes an increase in X. As an example, an inflow tends to increase the amount of liquid in a container. A countervailing influence would be an outflow.

QPC allows the direct influence forms to occur in model fragment definitions, but this is primarily to allow libraries to be easily ported to other QP theory based systems. The direct influence is redundant, and may be reduced to an additive influence on the derivative of a variable.

```
(I+ X Y)  ≡  (d/dt x x') (q-add x' y)
(I- X Y)  ≡  (d/dt x x') (q-sub x' y)
```

This reduction is not possible in QP theory, because the derivative of a quantity is not explicitly represented.

Together with the `d/dt` relation, the `q+` and `q-` influences are sufficient to describe a very large class of differential equations – those that are composed of "reasonable" piecewise monotonic functions.[4] These two influences, however, are very coarse. For this reason, we also provide the stronger influences `q-add` and `q-sub`. It would also be possible to provide other means of specifying partials in a composable fashion.

An influence does not provide enough information to actually *predict* anything. It is necessary to identify the complete set of influences on a variable in order to provide a constraining equation on its values. Logically inferring that a set is complete is often rather difficult. It requires proving that no other object could be a member of the set. One common technique that is used to simplify this type of inference is the *closed world assumption* or CWA. In this case, the CWA is that unless an object is explicitly proven to be a member of a set of influences, it is not.

In our earlier work, we stated that:

$$\text{Influences} + \text{CWA} = \text{Constraint}$$

It is now clear that this statement is somewhat stronger than necessary. The CWA is just one (computationally tractable) means of drawing the required inference. In his logical encoding of QP theory instantiated for a specific example, Davis [Davis, 1992] is able to prove monotonically that all of the influences are known.

Now that we have discussed the compositional equations that may be asserted by model fragments, we examine the model fragment definitions for the domain theory. Some of the model fragments help to establish basic properties of objects, some describe domain phenomena, and others establish the dependence of structural conditions on relations between quantities. The first and second types correspond to QP theory views and processes respectively.

The first model fragment establishes some basic properties for a liquid that is in a container. It is not defining a new type of object called a `contained-liquid`, but establishing some relations between quantities that are used to describe `liquids`.

[4] See Section 2.2.

```
(defModelFragment contained-liquid
     :participants ((l :type liquids)
                    (c :type containers))
     :structural-conditions   ((container l c))
     :consequences
     ((>=  (mass l)     zero)
      (q+0 (pressure l) (level l))
      (q+0 (level l)    (mass l))))
```

This says that a liquid in a container has a non-negative mass, that the pressure is qualitatively proportional to level, and that level is qualitatively proportional to mass. The influence `q+0` is a specialization of `q+` implying that, all else being equal, the function goes through the origin.

The next fragment defines flow out of a container into a drain. The mass of the liquid is negatively influenced by the flow rate, so a positive flow rate tends to reduce the amount of liquid.

```
(defModelFragment fluid-flow-out
     :participants ((c :type containers)
                    (out :type drains)
                    (l     :type liquids))
     :structural-conditions
      ((fluid-connected c out)
       (contents c l))
     :consequences
     ((>=  (flowrate out) zero)
      (q+0 (flowrate out) (pressure l))
      (I-  (mass l)         (flowrate out))))
```

Flow into a container from a source is very similar, except that the mass of the liquid is positively influenced by the flowrate of the source, so a positive flowrate tends to increase the amount of liquid. In addition, there is a slight subtlety. This model fragment has been constructed so that the flow from a source might flow into several different containers. A new variable is introduced called flowrate-into, which is the actual flow into a container. The total flow out of the source will be the sum of the flows into all of the containers that the source is fluid-connected to. This makes it possible for the domain theory to model the case in which the faucet is over both basins.

```
(defModelFragment fluid-flow-in
     :participants ((s :type sources)
                    (c :type containers)
                    (l :type liquids))
     :structural-conditions
      ((fluid-connected s c)
       (contents c l))
     :consequences
     ((> (flowrate s)        zero)
      (> (flowrate-into s l) zero)
      (q-add (flowrate s) (flowrate-into s l))
      (I+    (mass l)     (flowrate-into s l))))
```

Some simple geometric properties are defined by the next two fragments. The first says that for every extended object, the position of the left side is smaller than that of the right side. The second fragment enforces the semantics of `over`.

```
(defModelFragment extended-object
  :participants ((x :type containers))
  :structural-conditions ((considering  geometry))
  :consequences
  ((< (left-side x) (right-side x))))


(defModelFragment is-over
    :participants ((x :type objects)
                   (y :type objects))
    :structural-conditions
    ((considering geometry)
     (above x y))
    :operating-conditions ((>= (position x) (left-side y))
                           (<= (position x) (right-side y)))
    :consequences ((over x y)))
```

Model fragments have explicit bounds of applicability. The bounds on the fragments that constitute a complete model may be combined to determine the bounds for the model. The explicit association of bounds with model fragments is one of the most important contributions of QP theory to modeling.

In addition to the structural conditions that were present in the other definitions, the `is-over` model fragment also has a pair of *operating conditions*. Operating conditions are relations between quantities. There is an important difference between operating and structural conditions:

simulation directly changes operating conditions.

Structural conditions are only indirectly affected by simulation through the intermediary of quantity conditions. It is the asymmetry between operating and structural conditions that enables QPC to employ a hybrid architecture in which the simulator only computes behaviors from equations and values, and need know nothing about the logical relations used by the model builder.

The `is-over` model fragment also provides an example of a structural condition in its `consequences`. When an instance is active, the condition `(over x y)` is true (with `x` and `y` bound to appropriate objects).

1.3.2.3 Knowledge About Modeling

The modeling knowledge includes control knowledge about when to instantiate model fragments, and when to create new objects. In this domain, a liquid should be created when there is a fluid connection to an empty container.

```
(defEntity new-liquid
    :goal (contents dst new-liquid)
    :participants ((src :type objects)
                   (dst :type containers))
    :structural-conditions
      ((fluid-connected src dst)
       (no-previous-contents dst))
    :consequences ((contents dst new-liquid)
                   (isa new-liquid liquids))
    :initially ((= (mass new-liquid) zero)
                (= (level new-liquid) zero)))
```

This form specifies that whenever there is a fluid connection between some `src` and some container `dst`, and there is no contents in the `dst`, then a new liquid is created and its mass and level are initially zero.

1.3.3 Building Models

The first step in building a model is to *instantiate* every fragment whose structural conditions are satisfied. The next step is to partition this set into *active* model fragments, whose operating conditions are also satisfied, and *inactive* model fragments whose operating conditions are initially false, but which might become true during simulation. Given the scenario description and domain theory for our example, there are three model fragments that can be instantiated:

Fragment	Status
`(extended-object a)`	active
`(extended-object b)`	active
`(is-over f a)`	ambiguous
`(is-over f b)`	ambiguous

The first two model fragments are instantiated because A and B are containers, and we are considering geometry. They have no operating conditions, so they are not only instantiated, but also active. The second pair do have operating conditions. Furthermore, given the initial conditions, it is not possible to determine whether their operating conditions are satisfied. In other words, the proposed model is *ambiguous*. In order to build a complete model, QPC must eliminate any such ambiguity. The process of eliminating ambiguity is referred to as *model refinement*. QPC refines the ambiguous model into four possible models, in which the status of the ambiguous fragments is clear.

	`(is-over f a)`	`(is-over f b)`
1.	inactive	inactive
2.	inactive	active
3.	active	inactive
4.	active	active

Each of the refinements are subscopes of the original model. They are differentiated by the additional assertions needed to guarantee that the appropriate fragments will unambiguous. Model 1 is readily proven to be inconsistent. It would require the faucet to be over neither basin, which is impossible in this scenario. In model 2, the faucet is over B. In model 4, fluid flows into both basins. We will follow model 3 in which the faucet is over basin A.

In model 3, the domain theory implies that there is a fluid connection between the faucet and A. Once the fluid connection has been inferred, the domain theory implies the existence of a new entity – the liquid contents of the container A. QPC creates this object, calling it `a-contents`. Now things really get rolling, and QPC is able to instantiate several additional model fragments. The final set of instantiated model fragments is:

Fragment	Active?
`(extended-object a)`	yes
`(extended-object b)`	yes
`(is-over f a)`	yes
`(is-over f b)`	no
`(contained-liquid a-contents a)`	yes
`(fluid-flow-in f a a-contents)`	yes
`(fluid-flow-out a da a-contents)`	yes

1.3.3.1 Composing Fragments into a Model

An unambiguous model is a complete description of the situation. Adding more information to the KB will not result in any new model fragments being instantiated, unless the existence of new objects unsupported by the domain theory is posited.[5] Thus, in model 4, the complete set of active instances and consequently the complete set of influences has been identified. This sanctions two important types of inferences:

1. Sets of influences may be resolved into constraints.

2. Uninfluenced variables are inferred to be constant.

Several of the fragments contribute influences to the model:

Fragment	Influence
fluid-flow-out	`(q+0 da-flowrate a-contents-pressure)`
fluid-flow-out	`(q-sub a-contents-mass-derivative da-flowrate)`
fluid-flow-into	`(q-add a-contents-mass-derivative f-flowrate-into)`
fluid-flow-into	`(q+0 f-flowrate f-flowrate-into)`
contained-liquid	`(q+0 a-contents-level a-contents-mass)`
contained-liquid	`(q+0 a-contents-pressure a-contents-level)`

There are two influences on the derivative of `a-contents`' mass, one from the flow into the container, and one from the flow out the drain. The `q-add` on the derivative of the mass represents the `I+` in the model fragment definition, `q-sub` encodes the `I-`. These two

[5] This essential property of refinement is proven in Section 3.8.1.

influences are resolved into a single constraint:[6]

```
((sum + -) f-flowrate-into da-flowrate a-contents-mass-derivative)
```

The indirect influences between mass, level, pressure, and da-flowrate are each resolved into a single monotonic function constraint. The constraints resulting from influence resolution are:

```
((d/dt a-contents-mass a-contents-mass-derivative))
(((m +) a-contents-pressure da-flowrate))
(((sum + -) f-flowrate-into da-flowrate
            a-contents-mass-derivative))
(((sum +) f-flowrate-into f-flowrate))
(((m +) a-contents-level a-contents-pressure))
(((m +) a-contents-mass a-contents-level))
```

QP theory makes a powerful *sole mechanism assumption*, which states that all change results from the activity of processes, and is caused by direct influences. In QPC, there is no set of processes which cause change, nor are there really any direct influences. Nonetheless, there is an analog to this powerful assumption — *all exogenous variables are constant.* The set of exogenous variables is determined by defining the set of dependent or influenced variables. A variable is influenced if it has an explicit derivative, is on the receiving end of any influence, or is in an algebraic relationship with an influenced variable. There are a number of variables that are inferred to be constant:

```
((constant b-left-side))
((constant b-right-side))
((constant a-left-side))
((constant a-right-side))
((constant f-position))
((constant f-flowrate-into))
((constant f-flowrate))
```

The active model fragment instances contribute constraints that define the QDE for the model. Both active and inactive instances may contribute boundary conditions for the QDE so that the simulator can detect when a behavior has reached the boundary of applicability of the QDE. These bounds are characterized by the operating conditions of the active and inactive instances. E.g in this example if the faucet were moving, the `(is-over f a)` instance would become inactive as the faucet moved past the right side of `A`, and the `(is-over f a)` instance would become active at the same time.

1.3.4 Solving the Model Yields Behaviors

The completed model determines a single qualitative initial value problem, which comprises a QDE, and a set of initial values. This problem may be provided to QSIM to solve, resulting in a set of possible behaviors. In the model in which the faucet is over basin A, there is

[6] The sum constraint has the syntax: `((sum s`$_1$ `...` `s`$_n$`) x`$_1$ `...` `x`$_n$ `y)`, where s_i is $+$ or $-$. It is equivalent to the equation: $y = s_1\ x_1\ ...\ s_i\ x_i$. E.g. `((sum + -) a b c)` $\equiv +a - b = c$.

only a single qualitative behavior, which is displayed in figure 1.3. As expected, the flow from the faucet causes basin `A` to fill until equilibrium is reached. Figure 1.4 displays the models and behaviors generated from them.

1.3.5 Interpreting Graphs of Qualitative Behaviors

The qualitative plot in Figure 1.3 is the standard display of the output from QSIM. Each variable is graphed separately. If the qualitative direction of change (sign of the first derivative) of a variable in a state is increasing, decreasing, or steady, then it is depicted with an upward arrow, downward arrow, or circle respectively. Thus, in the state corresponding to the temporal interval between `t0` and `t1`, the variable `a.m-2.contents.level` is between `0` and `inf`, and increasing. The precise location of a point within an interval is completely arbitrary.

The graph in Figure 1.4 is a concise representation of QPC's output. It depicts both the models and the qualitative states representing their behaviors.

Object	Depiction
Incomplete model	hollow diamond
Inconsistent model	circle with cross
Complete model	solid box
Time point state	dot
Time interval state	circle
Transition state	circle with vertical bar
Quiescent state	circle with dot

The index of a model is written in a large font above it and to the right. The index of a state is written in a small font above it and to the right. Some final states have $n = m$, which indicates that the state n is equivalent to the state m. If the state is not quiescent, then the behavior from n continues with the successors of m. Thus, although the graph is represented is a tree, there may be back and cross edges from these final states.

1.3.5.1 Behaviors May Imply New Models

Given the simple domain theory described above, all behaviors were within the bounds of the initial model. Given a more reasonable domain theory, however, this might not have been the case. For example, if we had included the top of the tub in the model, there would have been a behavior in which the level reached the top, resulting in an overflow. This would have required QPC to build a new model to describe the situation as overflow began. The resulting initial value problem reaches equilibrium as the overflow matches the difference between the in and out flows. In Section 5.1.3 we describe an example using a similar domain theory in which overflow occurs.

Structure: M-2.
Initialization: (ONE-OF-SEVERAL-COMPLETIONS-OF S-13) (S-16)
Behavior 1 of 1: (S-16 S-20 S-21).
Final state: (GF QUIESCENT COMPLETE), (NIL), NIL.

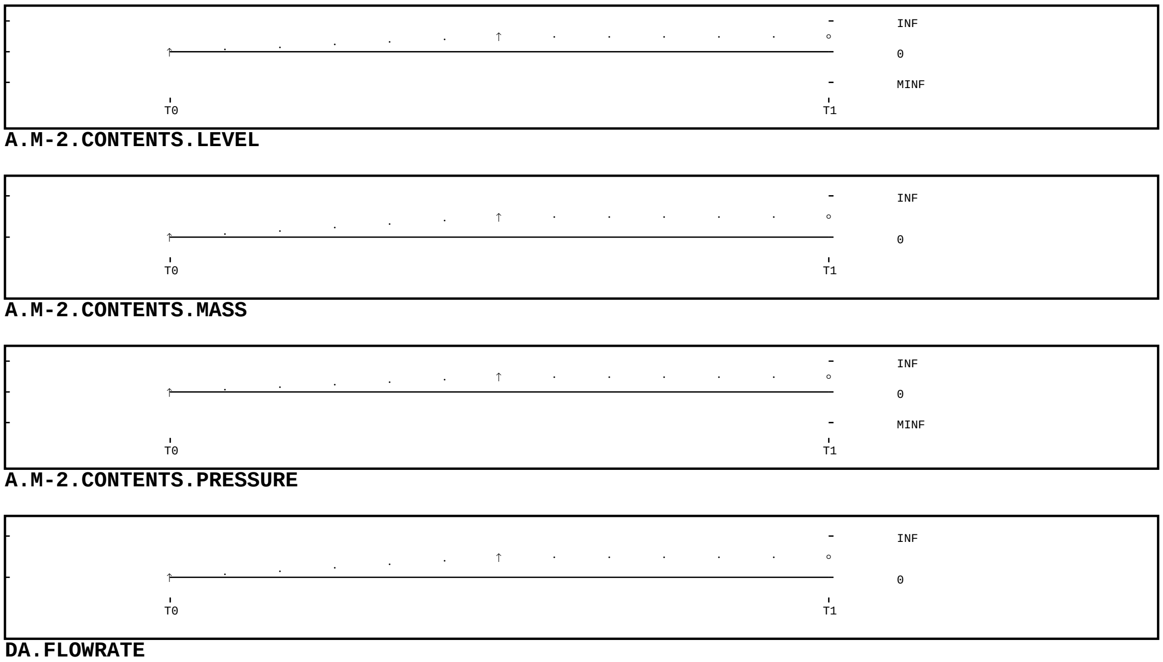

Figure 1.3: A qualitative plot of the behaviors predicted for the model in which the faucet is over basin A.

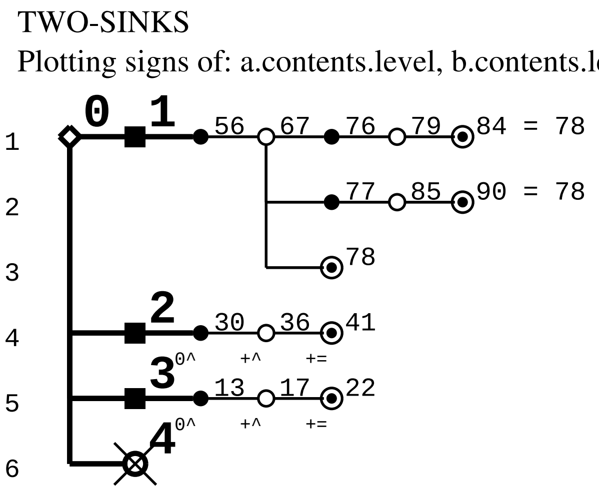

Figure 1.4: A tree of models and states describing the double sink scenario.

1.4 Road Map

This first chapter has summarized the main contributions of the QPC work and has presented a detailed example in which predictions were generated for a simple, but incompletely specified, scenario. This example illustrated the problems of forming complete descriptions from an initially incomplete one, constructing a QDE from a model, identifying the initial conditions, and generating behavioral predictions.

Chapter 2 discusses the goals of Qualitative Reasoning, paying particular attention to the QSIM work, which is employed directly by QPC. Chapter 3 gives a formal account of the QPC modeling language and algorithms, as well as the proofs of correctness. Chapter 4 discusses the issues of implementing a compositional modeling engine such as QPC in some detail, as well as describing aspects of the current implementation. Chapter 5 is divided into two main parts. In the first half, a number of illustrative examples are presented, similar in scope to the example in the introduction. They are used to highlight various aspects of the theory and practice of using QPC. The second half describes two more substantial applications of QPC in botany and chemical engineering. Chapter 6 discusses related research, focusing on Qualitative Process theory and its implementations. Chapter 7 concludes the dissertation, and discusses directions for future research.

Chapter 2

Background

This chapter provides background information about the goals, methods, and guarantees of qualitative reasoning about physical systems. Although particular attention is paid to QSIM, which is used directly by QPC, many of the points apply to other systems and approaches as well. We postpone the discussion of Qualitative Process theory to Chapter 6.

2.1 Qualitative Reasoning

There are two important goals of qualitative reasoning (QR) that distinguish it from other approaches to reasoning about physical systems. QR seeks to

1. provide the strongest inference possible, given incomplete information, and

2. require only the distinctions necessary to answer a query.

Note that neither of these points exclude precise or quantitative information. In fact, although well entrenched, one might argue that the *qualitative* term is outliving its usefulness or applicability.

There are two fundamentally different approaches to reasoning in the presence of incomplete information.

1. Make inferences that are expected to be correct most of the time, but cannot be guaranteed to be so.

2. Make correct, even if weak, inferences.

Default logics [Genesereth and Nilsson, 1987], fuzzy [Zadeh, 1965], and probabilistic reasoning methods [Pearl, 1988] all deal with incomplete information by making inferences that may turn out to be incorrect. They may be characterized as non-monotonic or defeasible methods. Surprisingly, numerical simulation methods also fall into this category. They require precise algebraic relations and initial values to generate predictions. Because the information is rarely available, the modeler is forced to select representative values. Predictions that follow from such representative values are defeasible in the same way that default logic inferences are when a default is learned to be false.

Qualitative reasoning methods deal with incomplete information by drawing inferences that can be guaranteed to be correct. Typically this is accomplished by identifying the ambiguities and performing an exhaustive case analysis.

2.1.1 History of Qualitative Reasoning

Humans have certainly been reasoning *qualitatively* about the physical world as long as they have been able to reason at all. Even though systems of measurement have been known since at least the times of the Pharos, people are typically unaware of the precise measures of the things around them. People do, however, have a very keen ability to detect distinctions between things. Reasoning about the distinctions between properties that have precise, but unknown, values is qualitative reasoning.

Early work on QR was inspired by protocol analyses of doctors engaged in diagnosis [Kuipers and Kassirer, 1983; Kuipers and Kassirer, 1984]. Doctors often use rough monotonic relationships to reason about physiology. For example, if a patient has appendicitis, then pressing harder on the right side of the abdomen causes an increase in the pain felt. Drinking fluids causes an increase in the blood volume. In the first example, the force of the press is unknown, and it is not at all clear how one would even define a scale for pain. In the second example, even if the fluid intake were precisely measured, the relation between it and the change in blood volume is unknown.

As a consequence of its dominant role in human reasoning, the weakly constrained monotonic function came to play a key role in qualitative reasoning. As the field has matured, the mathematical nature of the qualitative representation has become better understood. One can trace a decrease in research papers discussing such topics as protocol analysis and an increase in those discussing advanced mathematical topics such as phase portrait analysis and higher order derivatives. At the same time, the domains of interest have shifted more and more to engineering ones which have well understood mathematical theories. Furthermore, now that the mathematical underpinnings are better understood, one sees a subsequent change of emphasis to techniques for modeling and building domain theories. Perhaps some of the magic has been lost as the emphasis has drifted away from cognition towards mathematics and engineering, but that is common in science, and a sign that the field is maturing. The well understood substrate of QR is now able to support new classes of cognitively oriented tasks such as explanation in a way that was not possible just a few years ago.

Another direction that has been pursued recently is the incorporation of quantitative information. In some sense, it has been clear all along that quantitative and algebraic information are essential for producing precise descriptions of behavior. Once the qualitative representation and its mathematical character was well understood, the way became clear for other branches of mathematics to be incorporated. Nonetheless, the exclusion of quantitative information for a number of years served an important role — it allowed researchers to push hard on the qualitative methods, so that their strengths and limitations became known.

There are still a number of exciting and open issues left to address in the area of purely qualitative reasoning, as well as a host of problems at the juncture of qualitative and quantitative reasoning. At this point, it is important to return to the fundamental motivation for qualitative reasoning to attempt to characterize the types of queries that the techniques are meant to answer, and identify the knowledge that is necessary to answer such queries. QR engines need to be extended so that they can make use of whatever information is available, be it quantitative or qualitative.

2.2 The QDE Representation Language

The underlying representation for qualitative simulation is the qualitative differential equations, or QDE. A QDE consists of a set of variables, quantity spaces for the variables, and a set of constraints on them. The *solution* of a QDE under some initial conditions is a set of behaviors, which are sequences of qualitative states.

> *The key intuition is that, although the world changes continuously, it changes* qualitatively *only at isolated points. [Kuipers, 1993a, Page 82]*

This allows a symbolic description of continuous change to be constructed.

The *variables* in a QDE are reasonable functions of time. Typically, the time argument to the function will be left off. For example, instead of writing $v(t)$, we often write v.

A *reasonable function* is a continuous differentiable function of time defined over the extended real number line, $\mathcal{R}^*$, which includes the endpoints $-\infty$ and $+\infty$. It may only have a finitely many critical points in any bounded interval, and it must behave reasonably at its end points [Kuipers, 1993a, Pg83].

A *quantity space* is a finite, totally ordered set of landmark values. Each *landmark value* is a symbolic name, representing a particular value in $\mathcal{R}^*$. Even though the actual numeric value of a landmark may not be known, it nonetheless exists and is a single point in $\mathcal{R}^*$. This might be contrasted with a representation that employed symbolic names for interval values.

The landmark values of a variable $v(t)$ are meant to represent the set of important values that it might take on. Values define landmarks if they correspond to some important point in the range of the function, such as a critical point, if they correspond to some interesting event, such as a boundary of the current model, or if they correspond to some interesting cognitive construct. Because the set of landmarks implicitly defines the set of qualitatively important distinctions, the tractability of a qualitative simulation is strongly affected by them. For this reason, identifying the appropriate set of landmark values for a variable is becoming an important research issue.

In addition to the landmark values, it will be convenient to provide names for the decomposition of $\mathcal{R}*$ at zero.

Value	Sign	Qdir	Symbol
$[-\infty, 0)$	$-$	*dec*	$\downarrow$
0	0	*std*	Θ
$(0, +\infty]$	$+$	*inc*	$\uparrow$

It is also useful to provide a notation for determining the sign of a function.

$$[f(t)]_a \quad = \quad \mathrm{sign}(f(t) - a).$$

If a is not explicitly specified, then it is 0. So, by using the above naming conventions,

$$\begin{aligned} [2] &= + \\ [2]_4 &= -. \end{aligned}$$

The *qualitative value* of a function $f(t)$ with respect to the quantity space

$$l_1 < l_2 < \ldots < l_n$$

is a pair $\langle qmag,\ qdir \rangle$. The qualitative magnitude,

$$qmag \quad = \quad \begin{cases} l_j & \text{if} \quad f(t) = l_j \\ (l_j, l_{j+1}) & \text{if} \quad f(t) \in (l_j, l_{j+1}). \end{cases}$$

The qualitative direction of change is simply the sign of the first derivative of $f(t)$,

$$qdir \quad = \quad [\frac{d}{dt} f(t)]$$

and is typically expressed as either a member of the set {inc dec std} or $\{\uparrow\downarrow \Theta\}$ meaning $+$, $-$, or 0 respectively. Thus, the qualitative value of a function captures both its magnitude and the sign of its first derivative.

Notice that variables are truly continuous functions of time. It is a common misconception that because the landmark names come from a discrete set, and qualitative values are constructed from the landmarks and qualitative directions of change, the variables must somehow be discrete as well. The qualitative values provide a symbolic description of continuous change. Figure 2.1 shows three representations of continuous change.

The relationships between the variables in a QDE are determined by a set of constraints that hold at each time t. There are two constraints of fundamental importance in qualitative reasoning:

1. The monotonic function constraint has the general form:

$$((M\ s_1 \ldots s_n)\ x_1 \ldots x_n\ y) \quad \equiv \quad \begin{cases} y = f(x_1, \ldots, x_n) \\ [\frac{\partial f}{\partial xi}] = s_i \end{cases} \tag{2.1}$$

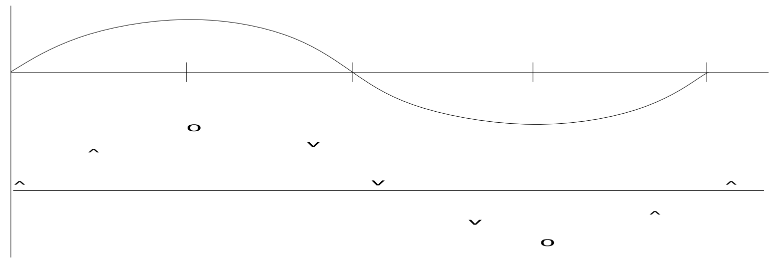

Figure 2.1: Three descriptions of continuous change. The values of the function $sin(t)$ are (a) graphed, (b) represented as a qualitative plot, and (c) represented as a sequence of qualitative values.

where each s_i is a sign. Each M constraint abstracts a set of multivariate monotonic functions. To ease fitting solutions of QDEs together, we allow these functions to be non-strict on their boundaries. Within the boundaries, M is slightly stronger than strict monotonicity. For example, the strictly monotonic function $f(x) = x^3$ is not abstracted by M, because $f'(0) = 0$.

2. The derivative constraint:

$$\texttt{((d/dt x y))} \quad \equiv \quad y(t) = \frac{d}{dt}x(t) \tag{2.2}$$

The M relationship may be further constrained by specifying points which satisfy it. These points are called *corresponding values*. For example, we may model the relationship between `amount` and `level` in some container with `((M +) level amount)`. This relation may be further constrained by specifying that when `amount` is 0, `level` must also be 0, and that when the `amount` has some maximum value, the `level` is at the top: `(((M +) amount level) (0 0) (max top))`.

Notice that we have chosen landmark values to characterize two important states of the container, namely when it is empty, and when it is full. Also notice that corresponding values support additional inferences. Since `level` is a monotonically increasing function of `amount`, and given that the function goes through $(0, 0)$, if `amount` is greater than 0, it must be the case that `level` is also greater than 0. This inference can readily be generalized to apply to arbitrary M constraints. During the course of simulation, it is possible to identify additional corresponding values for constraints. This helps to better define the shape of the under-specified functions.

Although the M and derivative constraints are sufficient to cover many ordinary differential equations (ODEs). They may not be the most convenient or the most precise means of doing so. In order to provide more readable and more precise QDEs, the representation is extended to include a number of additional constraints.

The following table shows five additional constraints that can be defined in terms of the fundamental M and `d/dt` constraints.

Constraint	Reduction to fundamental constraint
(M+ x y)	((M +) x y)
(M+0 x y)	(((M +) x y) (0 0))
(M- x y)	((M -) x y)
(M-0 x y)	(((M -) x y) (0 0))
(constant x)	(d/dt x y) where y = 0.

These constraints are syntactically much simpler and historically predate the M constraint by a number of years. The central role of the M constraint was not recognized until 1991 [Wellman, 1991].

Notice that two $M+$ constraints are not equivalent to a multivariate M constraint.

$$\begin{matrix} (\mathrm{M+}\ x_1\ y) \\ (\mathrm{M+}\ x_2\ y) \end{matrix} \quad \equiv \quad y = f(x_1),\ y = g(x_2).$$

Whereas

$$((M++)\ x_1\ x_2\ y) \quad \equiv \quad y = h(x_1, x_2).$$

These two relations differ. Suppose that x_1 is increasing and x_2 is decreasing. Given the two monotonic functions, this case is not possible since the first implies that y must increase, and the second implies that y must decrease. This case is, however, consistent with the multivariate monotonic function, which implies nothing about y's first derivative in this situation.

Another set of constraints actually add information to M by specifying the precise algebraic relationship between the variables.

Constraint	Covered by
((SUM $s_1 \ldots s_n$) $x_1 \ldots x_n$ y)	(((M $s_1 \ldots s_n$) $x_1 \ldots x_n$ y) (0 ...0))
(ADD x y z)	(((M + +) x y z) (0 0 0))
(MINUS x y)	(((M -) x y) (0 0))
(EQUAL x y)	(((M +) x y) (0 0))

This definition of these constraints differs slightly from the historical development thereof. For example, the `add` constraint intially represented a *qualitative addition* operator. The qualitative addition operator is semantically equivalent to the multivariate M constraint listed in the table above. Qualitative addition is much weaker than algebraic addition. It is now clear that it is of use to define `add` as true algebraic addition. Under the basic qualitative inferences employing distinct quantity spaces, corresponding values, and continuity, algebraic addition sanctions the same inferences as $(M++)$.

The last constraint describes multiplication:

$$(\mathrm{MULT\ x\ y\ z}) \quad \equiv \quad \mathrm{z = xy}$$

Multiplication cannot be so easily reduced to the M constraint, as the directions of the monotonic dependencies vary with the signs of x and y.

The algebraic constraints, `sum`, `add`, `minus`, `equal`, and (to a lesser extent) `mult` do not support any aditional qualitative inferences. They do, however, provide the information needed to make algebraic inferences. There are two important sorts of inference that are supported by algebraic information.

1. Algebraic relations allow inferences linking landmarks in different quantity spaces.

2. Algebraic relations allow new constraints to be inferred.

Algebraic inferences are discussed in more detail in Section 4.7.

A *state* is simply an assignment of a qualitative value to each variable in a QDE. A *consistent state* is such an assignment that also satisfies the constraints in the QDE. It is important to note that a state may have *duration*. The duration of a state is determined by the value of the variable `time` in that state. As with any other variable, `time` may take on either a landmark value, or an open interval between two landmarks. When `time` has a landmark value, then the duration of the state is instantaneous; when `time` has an open interval value, then the duration of the state is that of the open interval. A sequence of adjacent states alternates between time points, and the open intervals between them.

The notion of a state with duration and within which change is occurring, is an important contribution of qualitative reasoning. Although it has not yet been exploited, it suggests some new approaches to reasoning with the situation calculus, which normally assumes that situations are instantaneous and that change is the result of instantaneous actions that transform one situation to another.

A *behavior* is a sequence of temporally adjacent states.

2.3 Basic Techniques for Generating Solutions

QSIM solves a QDE by generating a behavior tree that has the following properties:

1. all states in every behavior are consistent with the QDE;

2. the initial state of every behavior is consistent with any provided initial information;

3. and every pair of adjacent states is consistent with the laws of continuity.

The general approach that QSIM takes is to generate and filter sets of possibilities. This is applied to determining the possible values for a variable in a successor state, the tuples of values for constraints, the vectors of values for states, to completed states, and to behaviors. If, at each step, 1) all of the possible values are included, and 2) no possible value is ever filtered, this approach provides convincing guarantees of correctness. Of course it is always possible to apply the filters earlier and to mix the filtering with the generation. This typically yields more efficient implementations at the cost of more complex intertwined algorithms.

2.4 Guarantees

Rather than restate the fundamental guarantees that the QSIM algorithm provides, we will quote Kuipers' recent retrospective article on them. The article, [Kuipers, 1993b], provides a very accessible review of QSIM and related research.

> *We can view an ordinary differential equation solver as a theorem prover for theorems of a special form:*
>
> $$\mathit{DiffEqs} \vdash \mathit{ODE} \wedge \ \mathit{State}(t_0) \rightarrow \mathit{Beh}.$$
>
> *A qualitative simulation algorithm can also be viewed as a special-purpose theorem prover:*
>
> $$\mathit{QSIM} \vdash \mathit{QDE} \wedge \ \mathit{QState}(t_0) \rightarrow \mathit{QBeh}_1 \vee \ldots \vee \mathit{QBeh}_n.$$
>
> *The soundness theorem says that when QSIM proves a theorem of the second type, it is true: that is, for any ODE described by the QDE, and* State(t_0) *described by* QState(t_0)*, , the solution* Beh *to the ODE is described by one of the qualitative behaviors,* QBeh$_1$, ..., QBeh$_n$*. [...]*
>
> *The incompleteness theorem says that some qualitative behaviors in the disjunct may be* spurious*: that is, not abstracting any real solution to an ODE corresponding to the QDE. [...] This situation is properly considered incompleteness, since QSIM has failed to prove the stronger theorem with fewer disjuncts.[Kuipers, 1993b]*

2.5 Advanced Techniques

2.5.1 Optimizing the Representation for Behaviors

The basic QDE language and solution methods are simple, elegant, and useful for many purposes. When addressing certain problems, however, the set of behaviors that the basic QSIM algorithm predicts may become intractably large. There are four reasons for this:

1. The representation of the set of behaviors is not compact. Behaviors that branch and then converge to an identical state [by some metric] are best represented by a directed acyclic graph [DAG]. Both cyclic and oscillatory behaviors are infinite sequences of states, which could be concisely represented by cyclic graphs.

2. The basic qualitative inferences may be too weak. QSIM relies on constraint satisfaction to determine if a state is consistent and continuity to determine if one state can follow another. These two techniques are both local in nature looking at either a single state or a single pair of states. It is also possible to look at longer sequences of states to determine if they are consistent.

3. The description may make unnecessary qualitative distinctions. For example, suppose that an emptying tank has a level whose quantity space is ($-\infty$ 0 LOW FULL ∞). There is a qualitatively distinct event when level = LOW. Thus, the single emptying behavior has 5 states instead of the interesting 3. This problem is exacerbated if there are multiple events of this type. If there were two such tanks, then there would not only be more states, but there would be multiple behaviors corresponding to the possible temporal orderings of the events. This sort of behavior is known as *occurrence branching* and may be addressed by the DAG representation mentioned above. Another approach, the focus of some current research, would be to control the generation and use of landmarks.

4. The purely qualitative description may not be sufficiently precise. The qualitative representation can be augmented to include algebraic and [partial] quantitative information.

Addressing these issues and improving the power of qualitative simulators has been the focus of considerable research. We will discuss some of the techniques for strengthening the inferences and representation in more detail.

2.5.2 Filtering Out Impossible Behaviors

The constraint filtering and continuity restriction are very local in nature. Constraint filtering looks at a single state at a time and continuity only applies to adjacent states. The field of *Dynamical Systems* [Hirsch and Smale, 1974] provides a wealth of techniques for analyzing the overall properties of systems of differential equations. In particular, phase space analysis identifies certain restrictions on the valid trajectories of a system. If a system has a two dimensional phase space then there are only a small number of valid patterns of trajectories through the phase space in the neighborhood around a fixed point: saddle, nodal, spiral, and circular. One property common to all is that a trajectory cannot intersect itself. This *non-intersection constraint* can be applied to qualitative characterizations of behavior as well as a quantitative or algebraic one. Any behavior that must intersect itself is mathematically impossible, and can be eliminated. Work by Struß [Struß, 1988a; Struß, 1988b], as well as by Lee and Kuipers [Lee and Kuipers, 1993; Lee, 1993; Lee and Kuipers, 1988], describe qualitative formulations of the non-intersection criterion. Sacks [Sacks, 1990] has been an outspoken advocate for the use of systems dynamics techniques in qualitative reasoning.

2.5.3 Incorporating Algebraic Information

As mentioned above, there are two desiderata for a qualitative reasoning system: to require no more information than necessary to make an inference, and to make use of whatever information is available. The pure qualitative algebra, however, does not make use of any algebraic or numeric information — which is often readily available, if only in approximate form. In this section, we examine two important aspects of incorporating algebraic information into our qualitative representation.

1. Algebraically redundant expressions may be qualitatively distinct.

2. Algebraic equations introduce relations between landmarks.

2.5.3.1 Algebraic Redundancy

The basic QR inferences based on continuity and constraint satisfaction among the constraints explicitly included in the QDE may miss certain inferences that are algebraically obvious. One example of this might be a QDE containing the pair of constraints:

$$(\mathrm{ADD}\ x\ y\ w)(\mathrm{ADD}\ x\ y\ z).$$

Surprisingly, a QR system such as QSIM would allow any of the following assignments:

$$\begin{array}{cccc} x & y & z & w \\ + & - & 0 & 0 \\ + & - & 0 & + \\ + & - & 0 & - \\ + & - & + & + \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

In other words, although it is *consistent* for w and z to have the same value or the same sign, there are many other possibilities (in fact without additional corresponding values, the signs of w and z are both unconstrained). Note that QSIM is fulfilling its guarantee here, since the correct values are among the disjunction of possibilities. Nonetheless, this behavior is not desirable in practice.

The basic QR algorithms are those of constraint satisfaction, not symbolic algebra. The main reason for this is that constraint satisfaction algorithms are simple, elegant, and typically fast. Symbolic algebra, on the other hand can be very difficult. Some classes of problems are known to be undecidable, such as determining if an expression involving transcendental functions is equal to zero. It remains an interesting open problem to determine the complexity of symbolic algebra on the QDE language.

Nonetheless, the inability to draw obvious conclusions such as the one above would be highly annoying. There are two different approaches to handling such situations.

1. Include algebraically redundant constraints. For example, the QDE containing the two ADD constraints could be augmented with the redundant constraint

$$(\mathrm{EQUAL}\ w\ z)$$

which would eliminate many of the undesirable influences. In fact, QPC does a modest amount of symbolic algebra to infer such relationships, and add them to a QDE. It would be interesting if methods could be developed for identifying which redundant constraints strengthen the qualitative inferences.

2. Strengthen the inference engine to filter out states that violate implicit algebraic constraints. Examples of this approach include the derivation and use of higher order derivatives, as well as the kinetic energy constraint.

2.5.3.2 Algebraic Equations Connect Landmarks

Algebraic expressions introduce relations between landmarks in distinct quantity spaces. The simplest form is to introduce inequality relations between landmarks. As we saw above, two ADD constraints might imply an EQUAL constraint. In that example, $w = z$, i.e. $w(t) = z(t)$. Thus, if w and z take on landmark values w_1 and z_1 at some time t_1, these landmarks must also be equal. If there is another landmark $w_0 < w_1$ for w, then we can also infer $w_0 < z_1$, and so on. The basic qualitative inference mechanisms do not make use of the information relating landmarks in distinct quantity spaces, unless they appear in a corresponding value for some constraint.

Algebraic expressions also introduce landmarks with specific algebraic properties, such as unity (1) for MULT, as well as other algebraic relations. For example, if $(x_1\ x_1\ y_1)$ satisfies (ADD $x\ x\ y$), then $x_1 = 2 * y_1$.

The qualitative monotonic functions provide little constraint on the algebraic relations between landmarks. For example M^+ abstracts both $y = x + 1$ and $y = e^x$. This pair of equations could be represented in a QDE by two identical montonic functions:

$$((M+\ x\ y)\ (0\ 1))$$
$$((M+\ x\ y)\ (0\ 1))$$

This provides very little constraint on x and y. A number of techniques have been developed to augment the purely qualitative monotonic function constraints with algebraic information.

1. Precise algebraic functions. Even in this exact case, there is still value in using the qualitative approach to simulation — one is guaranteed that every qualitatively distinct behavior of the system will be represented in the solution. Thus, if the initial conditions of the system are incompletely known, no important behavior will be missed. The alternatives such as Monte Carlo simulation do not provide the same guaranteed coverage.

2. Approximate algebraic functions. One approach that has been pursued is to provide a bounding envelope for each monotonic function. It is often possible to find relatively simple and mathematically tractable bounds without making unwarranted assumptions about the true function. A similar technique is to provide a function with error bounds.

3. Tables. When historical data is available to describe a function, various statistical estimation or interpolation techniques may be applied.

The algebraic specifications have the advantage of defining relations between the landmarks regardless of their numeric value. The tabular specification is only useful if some, perhaps approximate, numeric information can be associated with landmarks.

Methods for incorporating algebraic and quantitative information into QR is currently a topic of active research. One of the difficult aspects is to attain a precise description without violating the QR guarantees of coverage and soundness. One means of accomplishing this, used by [Berleant and Kuipers, 1992; Kay and Kuipers, 1992; Kay and Kuipers, 1993] is to start with a qualitative simulation and then use algebraic and numeric information to annotate the landmarks. The annotations are then used to identify states that are inconsistent. In addition to eliminating states and behaviors that violate the additional information, the annotations provide a much stronger more precise description of the behaviors. Numeric interval values may be associated with landmarks for all variables, including time.

2.6 Summary

Qualitative simulation employs a representation that has been grounded in ordinary differential equations (ODEs). The representation employs a qualitative differential equation (QDE), which is specified by a set of variables, quantity spaces, and constraints. Variable values are represented by a pair consisting of a qualitative magnitude (landmark or open interval from the variable's quantity space) and a qualitative direction of change. The QSIM system generates behavioral predictions consistent with a QDE and a set of initial values. The basic inferences may be strengthened by the addition of algebraic or semi-numerical information.

Chapter 3

Theory

One of the major of goals of AI research has been to provide engineers with tools to automate and support reasoning about the behavior of physical systems. By the nature of their work, engineers must be conservative about the tools that they use. A conclusion based on the output of an unreliable tool can have disastrous consequences, costing money, time, and human suffering. As a consequence, if we are to provide tools that can support engineering reasoning, we must also be able to accurately characterize their behavior, and guarantee that they fulfill these characterizations. Furthermore, as our tools become more flexible and powerful, it becomes even more important to provide a precise set of guarantees for them, because their behavior is less obvious. Although a concise characterization is desirable, accuracy and completeness are of over-riding importance.

In this chapter, I both define and prove the correctness of a hybrid compositional modeling engine, QPC, that incrementally constructs predictions of the temporal evolution of a physical situation. QPC is an extremely powerful tool, because it is able to make predictions about the behavior of an incompletely described situation. It has, however, been constructed in such a way as to enable a specification and proof of its correctness. The rigorous treatment would not have been possible without the clear separation between modeling and simulation, the mathematical clarity and rigor of the QSIM QDE representation, and the soundness proof of the QSIM algorithm [Kuipers, 1986].

QPC is provided with a domain theory, which describes the phenomena in a domain in terms of abstract *model fragments*, and a partial description of a physical situation (the scenario). It produces a description of the possible behaviors of the situation. QPC follows the general approach of Forbus' Qualitative Process theory [Forbus, 1984b] and the work on compositional modeling done by Forbus and Falkenhainer [Falkenhainer and Forbus, 1991] in describing the domain theory. Instances of the abstract model fragments are created based on the entities and relationships that appear in the scenario, and the mathematical relations entailed by them are composed into a qualitative differential equation (QDE), whose solutions describe the possible behaviors of the situation.[1] This process is complicated by the following:

[1] We use *instance* to mean "an instance of an abstract model fragment".

- *The description of the initial situation may be incomplete.*
- *The behavior may cross through a number of operating regions.*
- *The configuration of the situation may change over time.*
- *The description of the model fragments may be incomplete.*

This chapter is divided into two major portions. The first half is somewhat less formal. The modeling language is presented; each language construct is discussed and defined via a FOL axiom schema. Section 3.4 defines the conditions under which a model fragment instance is active. The second half is somewhat more formal. The language is represented in an equivalent, but more concise, form that is more convenient for the correctness proofs. The basic QPC algorithms are defined, the criteria for their correctness is developed, and their correctness is proven.

3.1 QPC Modeling Language

This section defines the syntax and the FOL translation of the essential constructs in the QPC modeling language.

3.1.1 Notation

The QPC language forms are described in a lisp-like syntax that is largely in accord with the conventions of the Knowledge Interchange Format (KIF) [Genesereth and Fikes, 1992]. The syntax is simple and easily readable by both humans and machines. As a consequence, it may be converted to any number of equivalent formats. The logical semantics of the language forms will be defined in a more conventional mathematical notation.

In KIF, as in Common Lisp, capitalization is unimportant, and keyword–value pairs may appear as the arguments to a form. A keyword is prefixed by a colon, and its value is the following expression. Unless otherwise specified, keyword arguments are optional and may appear in any order.

3.1.2 Defining Relations

Logical relations are defined by the form

```
(defrelation Name (T1 ... Tn) :type T)
```

This defines an n-ary relation, *name*, whose arguments, $a_1 \ldots a_n$, are of types $t_1 \ldots t_n$ respectively. The relation itself may also have a type, t. This allows some weak second order reasoning.

As a notational convenience, an n-ary function is sometimes written as an $n+1$-ary relation.

3.1.3 Defining Implications

Logical implications are defined by the form

```
(defrule A1 ... Am -> C1 .. Cn)
(defrule C1 <- A1 .. Am)
```

A `defrule` form introduces a logical implication $A_1 \wedge \ \ldots \ \wedge A_m \Rightarrow C_1 \ \wedge \ \ldots \wedge \ C_n$, where A_i and C_i are atomic formulae with all variables universally quantified. It is also a hint to an implementation that the implication should be used in a forward or backward fashion. Backward implications are restricted to be horn clauses ($n = 1$).

3.1.4 Defining Quantities

Quantities are introduced by the form

```
(defquantity Name (T1 ... Tn) (d1 e1 ... dm em) :range r)
```

This form is used to define both variables and magnitudes, which may be thought of as variables with constant values. A variable is a reasonable function of time whose domain and range are the extended reals.

In the case that $n > 0$, the form is equivalent to

$$\forall X_i : T_i \ \exists V : \text{variable} \mid \text{name}(X_1, \ldots, X_n, V)$$

where "name" is an n-ary function that associates a set of objects with the variable V. [2]

The d_i and e_i specify the dimensions of the quantity. Each d_i is a dimension such as mass, length, or time. The e_i are non-zero integers indicating the exponent of dimension d_i. E.g velocity ($\frac{m}{s}$) would have dimensions `(length 1 time -1)`, and force ($\frac{gm}{s^2}$) would have dimensions `(mass 1 length 1 time -2)`.

The range r can be either a numeric lower and upper bound (inclusive), or a symbolic constant such as `:positive`, `:negative`, or `:non-negative`.

An example shows the difference between the relation defined by `defquantity`, the variable that it associates with an object, and the value that the variable might have.

```
(defquantity mass (physobs) (mass 1) :range :nonnegative)
(mass Sebastian mass27)
(magnitude mass27 scenario1 ten-kg)
```

This defines `mass` to be a function that takes a `physob` and returns a variable. The mass of `Sebastian` is `mass27`, a real valued function of time, whose value must have the dimension mass. In `scenario1`, `mass27` happens to have the magnitude `ten-kg`.

In the case that $n = 0$, the `defquantity` form defines a landmark, which is a symbolic name for some magnitude. For example, the landmark `ten-kg` above could have been defined by:

[2] It is worth noting that V is existentially quantified. This is somewhat different from QP theory, in which the existence of quantities is introduced explicitly by views and processes. QPC is rather more liberal on the issue of logical existence. This eliminates certain difficulties in the semantics of QP theory which arise when a view attempts to reference a quantity that is not defined. In QPC a quantity is always there when you need it. It is left up to the implementation to avoid creating data structures to represent logically possible objects that are not necessary.

```
(defquantity ten-kg (mass 1) :range (10 10))
```

3.1.5 Defining Model Fragments

A model fragment is defined by the form

```
(defModelFragment mf
   :participants ((P1 :type T1) ...  (P_nP :type T_nP)
   :structural-conditions (C1 ...  C_nC)
   :operating-conditions (O1 ...  O_nO)
   :quantities (Q1 ...  Q_nQ)
   :consequences (R1 ...  R_nR))
```

This form defines an abstract model fragment named **mf**. The abstract model fragment is *instantiated* when the participants exist and the structural conditions are satisfied. The instance is *active* when the operating conditions are satisfied. The participant names, $P_1 \ldots P_{n_P}$, define functions that may be applied to an instance of mf, returning an actual participant. As a syntactic convenience, the participant names may be used throughout the body of the form to refer to the actual participants. In the structural condition clause they are replaced by variables, in the operating conditions and consequences they may be replaced by the functional reference. The model fragment name, mf, may also be used to refer to the instance in the consequences.

The typed *participants* are the objects that participate in the fragment. Structural conditions are any atomic formulae that do not include relations between quantities. In contrast, the operating conditions are atomic formulae consist solely of relations between quantities. When the structural conditions are satisfied, then an instance of the fragment exists. If the operating conditions are satisfied, then the fragment is active and the consequences are true throughout the model in which the instance is active. If the operating conditions are not satisfied for the instance, then it is said to be *inactive*.

The distinction between structural and operating conditions is essential to QPC. The simulator only predicts changes in quantities and the relations between them. Thus, the truth value of structural conditions does not change during the simulation of a single model, whereas the truth value of operating conditions may. Therefore, while a simulation can never result in a model fragment becoming instantiated, it can result in an active model fragment becoming inactive or an inactive one becoming active. A model ceases to apply when the activity of one of the instances comprising it changes. The operating conditions of the inactive fragments together with the negation of the conditions of the active ones define the boundary of a model's applicability.

The *quantities* $Q_1 \ldots Q_n$ are local to the model fragment. They are not accessible from other model fragments and provide a limited sort of encapsulation.

It is not strictly necessary to retain the notion of *abstract* model fragments. The logical translation could be rephrased to avoid the references to mf. It is, however, quite useful to be able to represent knowledge about the abstract model fragments, and it is extremely useful to be able to reason about the set of instances of a single model fragment. One example is for doing relevancy reasoning.

3.1.6 Defining Entities

Entities may be specified using the form

```
(defEntity (New :type T)
   :participants ((P_1 :type T_1) ...  (P_{n_P} :type T_{n_P})
   :structural-conditions (S_1 ...  S_{n_S})
   :operating-conditions (Q_1 ...  Q_{n_Q})
   :consequences (C_1 ...  C_{n_C}))
   :initially (I_1 ...  I_{n_I})
```

This form specifies that an entity *New* of type T exists when the participants P_i exist, the structural and operating conditions are satisfied, and the consequences are true of *New*. A new object is created only when no existing one can be found satisfying the consequences.

The participants and the conditions clauses have the same semantics as those in `defModelFragment`. In fact, `defEntity` can be reduced to the following:

```
(defModelFragment %New
   :participants ((P_1 :type T_1) ...  (Pn :type T_{n_T}))
   :structural-conditions (S_1 ...  S_{n_S})
   :operating-conditions (Q_1 ...  Q_{n_Q}))
(defrule (instance-of %New i) (active i) ->
   (exists New
      (isa New T)
      C_1 ...  C_{n_C})))
```

The `%New` is simply a convenient name different from `New`.

The `:initially` clause is unfortunately nonmonotonic. Intuitively, if a new object was really just created, then we want to provide some initial values for it. For example, if a flow creates a new fluid in a container, then the mass of that fluid must be 0. In practice, it is straightforward to determine if the entity is new. It is awkward, however, to express this logically. A first cut would be to make each of the I_j defaults. This would not have the desired semantics, as the defaults would be applied individually. Thus, even if the object were not newly created, any consistent defaults would be inferred.

3.1.7 Defining Scenarios

A scenario is defined using the form

```
(defScenario Name
   :entities ((E_1 :type T_1) ...  (E_{n_E} :type T_{n_E})
   :relations (R_1 ...  R_{n_R}))
```

```
:initially (Q1 ...  QnQ)
```

This form defines a scenario `Name`, in which objects with name E_i of type T_i are present. The atomic formulae R_i are asserted to be true in the scope of the scenario. Thus, they will be true in every model considered. The atomic formulae Q_i are asserted to be true in the scope of the initial time of the initial models of the scenario. Typically the Q_i are assertions about the relations between quantities that define the initial conditions. Just as the participant names may be used throughout the body of a model fragment definition, the entity names may be used throughout all other clauses of the scenario definition.

3.2 Representing Time and Scope

In order to reason about dynamic behavior in a logic based system, it is necessary to select a representation for time. There are three general approaches to representing predicates whose truth value is a function of time.

1. Introduce an explicit additional argument for every temporal predicate. For example, if $p(x)$ is a predicate whose truth value may change over time, then use $p(x, t)$.

2. Reify the temporal predicates. The reified temporal predicate is known as a *fluent*, a function whose value changes with time. In this scheme, $p(x)$ becomes a function whose value, f, is a function of time. A special predicate, `holds`, is introduced such that `holds`$(p(x), t)$ is true for exactly those times at which the original predicate was true.

3. Represent times with logical models. The extensions of each logical model is then the set of all predicates that can be true together at the time associated with the model. In this approach $p(x)$ remains $p(x)$, and the time at which it applies is implicit in the time associated with the model. Reasoning about temporal sequences becomes the meta-logical process of reasoning about the accessibility of one logical model from another. QPE uses this approach.

QPC takes the first approach; an additional argument is added to each predicate. The notion of *time*, however, is generalized to that of *scope*. A *scope* may be an instant or interval of time. It may also be used to represent choices or possibilities. It is similar to the notion of *worlds* as used in various knowledge representation systems such as KEE.

Scopes may be nested, and relations that are *scoped* are inherited from a scope to its subscopes. This inheritance is simple and monotonic. It behaves as if it were defined by the following second order axiom:

$$\forall R : \text{scoped},\ s_1, s_2, x_1 \ldots x_n |$$
$$R(s_1, x_1, \ldots, x_n) \wedge \text{subscope}(s_1, s_2) \Rightarrow R(s_2, x_1, \ldots, x_n).$$

The subscope relation defines a partial order of scopes, whose greatest element is called `true`.

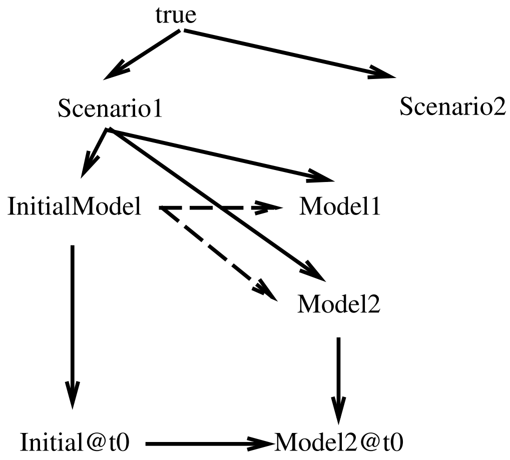

Figure 3.1: Several important scopes in QPC. The solid arrows depict a subscope relationship. Every scope is (transitively) a subscope of true. Every model within a scenario is a subscope of the scenario. Dashed lines are subscopes that are also refinements.

Predicates that are universally true may be thought of as implicitly scoped to `true`. A scenario defines a scope that applies throughout a single QPC problem solving session. A scenario may be described by multiple models. There is a distinguished subset of initial models. The initial models describe the scenario starting at the initial time point. Each complete model may have a set of states associated with it, each representing a time point or interval.

In addition to the subscope relation, we can also define a successor relation that is true of time-like scopes.[3]

The notion of scope is very general, and can be further specialized by adding various temporal axioms such as the ones introduced by Allen [Allen, 1983].

3.3 Entities

The set of entities is a distinguished set of logical symbols that are meant to correspond to *objects* in the real world. Although grounding these symbols in the world is an interesting philosophical problem, it is not one that we will discuss here. The distinction between entities and other logical objects is not always sharply defined. Intuitively, entities should be observable; they should have myriad properties; they should be capable of dynamic behavior. They are things that we like to think of as existing (at least for a while) in their own right. Most importantly, their existence should be invariant under many different

[3] I say time-like because it might be used to describe real world time, or simulation time, or a sequence of decisions.

points of view. Depending on our interaction with the world, the decomposition that we enforce on it in terms of objects may vary.

An entity can be said to exist in two very different ways [Forbus, 1985]. It may have

1. *logical existence*, or

2. *physical existence.*

Logical existence is normally introduced by the $\exists$ operator. For example,

$$\exists x : \mathrm{page}(x)$$

means that there is some object in the domain of discourse, that we may denote by a constant symbol, say `p21`, such that page(`p21`) is universally true. This is an *inherently atemporal statement.*

The notion of physical existence in time is a stronger commonsense notion. We indicate it with a predicate physicallyExists(x, t), meaning that the entity x can be (usefully) said to physically exist at the time t. It is clear that physical existence implies logical existence, but the converse is certainly not the case.

It is this distinction between logical and physical existence that allows me to speak of the arsenic in my coffee cup – even the arsenic in my cup right now. Fortunately, it also allows me to say that the mass of the arsenic is zero. There are a number of ways to axiomatize physical existence. For physical objects (which can have mass), a reasonable axiom might be:

$$\mathrm{physob}(x) \wedge \ \mathrm{mass}(x,t) > 0 \quad \Longleftrightarrow \quad \mathrm{physicallyExists}(x,t)$$

Such an axiom allows me to readily infer that the arsenic in my coffee cup does not exist. QPC discriminates between the problem of identifying a likely set of logical objects and that of inferring their various properties (including physical existence).

3.4 Model Fragments

The goal of compositional modeling systems, such as QPC, is to compose a model of a physical situation from a set of model fragments. The nature of the fragments and the means by which models are composed and chained together to form a coherent picture of a situation that passes through multiple operating regions, distinguish different compositional modeling systems from each other. In this section, we will explicate these issues within the context of QPC.

The primary role of a model fragment is to define the conditions under which a set of equations is to be included in a model. For example, the typical mathematical notation

$$y = \begin{cases} \text{if } x > 0 & \text{then } f(x) \\ \text{if } x \leq 0 & \text{then } g(x) \end{cases}$$

could be readily defined using two model fragments,

```
(defModelFragment mf1 :operating-conditions ((> x 0))
                      :consequences ((f x y)))
(defModelFragment mf2 :operating-conditions ((<= x 0))
                      :consequences ((g x y)))
```

QPC extends this basic ability to define multiple regime equations by allowing logical *structural conditions* and *consequences*.

Structural conditions allow the logical preconditions for a model fragment to be explicitly represented. This is one of the most important advances of QP theory and subsequent compositional modeling systems including QPC. Traditional modeling and simulation engines piece together models, but do not record the bounds of applicability of a model, nor the logical conditions within which it applies. Thus, it is possible to run a simulation well beyond the region within which its results are reliable. For example, a model may use a linear approximation to extrapolate beyond the region where the approximation provides a close match with the approximated function. It is also possible to use a model to make predictions for a situation in which it simply does not apply. For example, a laminar flow model may be used to describe a situation in turbulent flow.

Another class of preconditions include *modeling assumptions*. A modeling assumption does not necessarily follow from the physics of the situation, but is rather a decision on the part of the modeler that should simplify the modeling process. For example, in many physics problems, an object can be viewed as a point mass. These are heuristic simplifications that the modeler believes will not compromise the accuracy of the predictions. By making these assumptions explicit, they can be reasoned about. The modeler is able to change the set of assumptions, evaluate their coherence, and determine if model fragments from a library are suitable for a particular task.

3.4.1 Activity of a Model Fragment Instance

In this section, we consider when a model fragment instance is active. Our analysis is driven by two central desiderata:

1. The mathematical model must be amenable to standard analysis.
2. The simulation problem must be well defined and consist of a set of equations, a set of initial values, and easily evaluated boundary conditions.

Both of these desiderata are guaranteed by the following assumption:

Assumption 1 *A model fragment instance is active over a <u>closed interval</u> of <u>non-zero duration</u>.*

Let us explore how this assumption interacts with the semantics of operating conditions. Consider the example shown in figure 3.2.

Initially `a.x` is less than 0 and increasing; there are no active model fragment instances. The operating condition for `(f1 a)` is `a.x` $>$ 0 and is `a.x` $\geq$ 0 for `(f2 a)`. A transition occurs when `a.x` hits `zero`. The question is: what should happen following the transition? There are two possibilities:

```
(defQuantity x objects)

(defModelFragment f1
   :participants ((p :type objects))
   :operating-conditions ((> (x p) zero)))

(defModelFragment f2
   :participants ((p :type objects))
   :operating-conditions ((>= (x p) zero)))

(defScenario s
   :entities ((a :type objects))
   :landmarks ((max :range :positive))
   :relations ((increasing (x a)) (<= (x a) max))
   :initial-conditions ((< (x a) zero)))))
```

Figure 3.2: Example to illustrate operating regions. QPC constructs two models for the scenario `s`. In the first, no model fragments are active, and `a.x` increases towards `zero`. A transition occurs when `a.x` hits `zero`. In the second model, both `f1` and `f2` are active, with `x.a` initially `zero` and increasing.

1. A second model could be built to describe the situation when `x.a` $= 0$ and `(f2 a)` is active, and a third to describe the situation when `a.x` > 0 and `(f1 a)` is also active. The second would hold only for the instant that `a.x` $= 0$, and the third would hold over the subsequent open interval.

   QPE and GIZMO take this approach. This method has a fundamental problem — it requires the use of non-standard analysis. The value of `a.x` in the end of the first model is $0 - \epsilon$, and $0 + \epsilon$ at the start of the third model.

   A secondary problem of this approach is that model fragments and models may hold for only an instant. This gives rise to the phenomenon of *stutter* in QPE and GIZMO in which there is a cycle of models, each of which holds for only an instant. This implies a distinctly non-standard model of time.

2. A second model could built to describe the situation when `x.a` ≥ 0 in which both `(f1 a)` and `(f2 a)` are active.

   This is the approach used by QPC. It allows for a standard model of time and the use of standard analysis. Each model applies over a closed interval of non-zero duration.

   This is accomplished by requiring model fragments to meaningfully hold in the limit. Note, however, that in QP theory, as realized by QPE and GIZMO, variables are

> continuous throughout.[4] Thus, their values must be defined in the limit from either side of a transition. For this reason alone, it is only a very small step to require that the instances of model fragments also apply in the limits.

Under QPC's approach, both `(f1 a)` and `(f2 a)` are said to hold in the boundary case in which `a.x` = 0. This does not mean, however, that their operating conditions are effectively the same. Strict and non-strict conditions have distinct semantics.

Before precisely defining this semantics, we turn again to the small example of Figure 3.2, modifying it slightly. Consider a scenario without initial conditions, and with the relation `(increasing (x a))` moved from the scenario definition into the consequences of `f1`. There are three possible initial models depending on which instances are active:

Model	Active Instances		Condition
M-1			(< a.x 0)
M-2		(f2 a)	(= a.x 0)
M-3	(f1 a)	(f2 a)	($\geq$ a.x 0)

The second two models illustrate the difference between strict and non-strict conditions. In `M-2`, `(f2 a)` is active and `(f1 a)` is inactive. In order for this to be the case, `a.x` must be equal to zero for some closed interval of non-zero duration. In `M-3`, both are active. In this case, `a.x` may be initially equal to zero, but it must become greater than zero immediately. This corresponds to the QPE/GIZMO case where both instances are active during the open interval following the initial situation.

Definition 1 *A region is characterized by a relation between two quantities. Let r be a relation from the set $\{<, \leq, =, >, \geq\}$, and x and y be the quantities. The boundary of a region characterized by $x\ r\ y$ is characterized by the equality $x = y$.*

Definition 2 *A region whose characterizing relation is $x\ r\ y$ is active if*

1. *$x\ r\ y$ is satisfied, or*
2. *$x = y$ and the following condition is true. Let $u = x - y$.*

$$\exists i\ \frac{d^i u}{dt}\ r\ 0 \text{ and } \forall j\ 0 \leq j < i,\ \frac{d^j u}{dt} = 0$$

A region is active if its characterizing relation is true or if it could become true in an instant. The latter may occur if x and y are equal, but their difference is changing in the appropriate direction. This clarifies the distinction between regions whose characterizing relations are strict or non-strict. A strict relation need not hold initially, but must be satisfied immeadiatly.

Definition 3 *A model fragment instance is active if all of the regions characterized by its conditions are active.*

[4] QP theory variables are at least qualitatively continuous throughout. See Section 6.2 for more discussion of this point.

```
(defQuantity x objects)
(defQuantity y objects)

(defModelFragment f1 :participants ((p :type objects))
   :operating-conditions ((> (x p) zero))
   :consequences ((m+ (x p) (y p))
                  (correspondence (x p) zero (y p) zero)))

(defScenario s :entities ((a :type objects))
   :landmarks ((min :range :negative))
   :relations ((decreasing (x a))
               (>= (x a) min))
   :initial-conditions ((> (x a) zero)))
```

Figure 3.3: Example to illustrate the benefit of pushing the applicability of an instance to its boundaries.

Definition 4 *A model fragment instance is inactive if the region characterized by the negation of one of its conditions is active.*

It is worth noting that for many sets of initial conditions, it is not possible to determine whether or not a region is active. For example, given `a.x` $= 0$, any of the regions characterized by `a.x` < 0, `a.x` > 0, `a.x` $= 0$, and so on, could be active. The problem is that it may not be possible to determine the sign of the first (if any) non-zero derivative. This is manifested in the qualitative representation by the state transition from a value of `(0 std)` to a value of `((0 inf) inc)` or `((minf 0) dec)`. If fact, no matter how many explicit derivatives we may know for X, if they are all 0, it is possible for X to change in the next state due to the effects of an implicit higher order derivative.

An advantage of QPC's approach is that the model can be used more effectively to constrain behavior. Consider the problem of Figure 3.3. The model fragment `f1` is active when `(x a)` > 0. It asserts a monotonic function constraint between `(x a)` and `(y a)`, with a corresponding value at (0 0). Initially, since `a.x` > 0, `(f1 a)` is active. Under QPC's approach two models are built. In the first one, both `a.x` and `a.y` decrease until they hit zero together (because of the correspondence) and a transition occurs. In the second, `a.x` continues to decrease until it hits `min`, and `a.y` is inferred to be uninfluenced, remaining constant at 0. Now consider the QPE/GIZMO approach. The initial model only applies during the open interval where `a.x` > 0. Thus, *there are no constraints other than continuity* available to constrain the value of `a.y` in the following state. Using the standard qualitative formulation of continuity, it is consistent for `a.y` to be greater than zero in a subsequent situation. It would require a considerably more sophisticated analysis to conclude that both `a.x` and `a.y` must hit zero simultaneously in any subsequent state.

3.4.2 Influences and Constraints

Just as there are certain constraints that are particularly useful in constructing qualitative models (e.g. monotonic function constraints), there are certain relations that are particularly convenient for asserting fragments of information that can be composed into constraints. These *compositional relations* include *influences* of one variable on another. Asserting that X is positively influenced by Y, written X Q+ Y, means that in the absence of countervailing influences, an increase in Y causes an increase in X. The influence relationship is inherently causal, and by convention (introduced by Forbus) the causal arrow is directed from right to left. As an example, an increase of the amount of liquid in a container will cause the level to rise, unless there is a countervailing influence, such as a simultaneous increase in the diameter. This relationship is expressed by writing: `level Q+ amount`. In a model fragment definition, it would appear as: `(Q+ (level can) (amount can))`.

An important first step in formalizing influences is to recognize that an influenced variable, X, is functionally determined by the variables, Y_i, that influence it, and that the type of the influence determines the partial of the function with respect to that variable:

$$X\ Q+\ Y_i \ \equiv\ f(\ldots Y_i \ldots) \text{ and } \frac{\partial f}{\partial Y_i} > 0. \tag{3.1}$$

The partial derivative captures precisely and mathematically the intuition of Forbus' *all else being equal*. Although succinct and rather elegant, this definition is not quite sufficient. It does not precisely specify how multiple influences are combined, nor does allow for different types of influences to exist between a single pair of variables.

The semantics of the influences on a variable X are precisely defined in three steps.

1. Let $\mathcal{I}$ be a set of pairs consisting of an influence type and an influencing variable.

$$\begin{aligned} \mathcal{I} &= \{(I_j, y_j) : I_j(X, y_j) \text{ where } I_j \in \{\texttt{q-add q-sub q+ q-}\}\} \\ n &= |\mathcal{I}|. \end{aligned} \tag{3.2}$$

2. Let $f(a_1, \ldots, a_n)$ be an n-ary function such that for all j where $0 \leq j \leq n$,

$$\begin{array}{ll} \text{if } I_j = \texttt{q-add} & \text{then } \frac{\partial f}{\partial a_j} = 1, \\ \text{if } I_j = \texttt{q-sub} & \text{then } \frac{\partial f}{\partial a_j} = -1, \\ \text{if } I_j = \texttt{q+} & \text{then } \frac{\partial f}{\partial a_j} > 0, \\ \text{if } I_j = \texttt{q-} & \text{then } \frac{\partial f}{\partial a_j} < 0. \end{array}$$

3. Finally, let X be determined by the function f applied to the influencing variables.

$$X = f(y_1, \ldots, y_n). \tag{3.3}$$

There are several important consequences of this definition:

1. The combination of different types of influences is naturally and uniformly defined.
2. Superficially conflicting influences may be defined. For example,

   X `q-add` y, X `q-sub` y

   defines a function $f(a_1, a_2) = a_1 - a_2 + c$, and states that $X = f(y, y) = c$, where c is some constant (that is often implied to be 0). Such a situation may arise when there are different causal paths connecting x and y.
3. It implies a *single mechanism assumption*. That is, if two model fragments contribute the same influence, they are assumed to be describing an identical mechanism. So, if both model fragments m_1 and m_2 assert the influence X `q-add` y, then $X = y + c$, rather than $2y + c$. This is not the only reasonable semantics for influences. It is possible to modify the definition so that every influence implies a distinct mechanism.

In the terminology of QP theory, the influences described above are all *indirect* influences. QP theory also provided a *direct* influence, written X I+ Y. This asserts that if there are no other countervailing influences, positive Y causes an increase in X. As an example, an inflow tends to increase the amount of liquid in a container. A countervailing influence would be an outflow.

QPC allows the direct influence forms to occur in model fragment definitions, but this is primarily to allow libraries to be easily ported to other QP theory based systems. The direct influence is redundant, and may be reduced to an additive influence on the derivative of a variable.

```
(I+ X Y)  ≡  (d/dt x x') (q-add x' y)
(I- X Y)  ≡  (d/dt x x') (q-sub x' y)
```

This reduction is not possible in QP theory, because the derivative of a quantity is not explicitly represented.

Together with the `d/dt` relation, the `q+` and `q-` influences are sufficient to describe a very large class of differential equations – those that are composed of "reasonable" piecewise monotonic functions. This level of description, however, is very coarse. For this reason, we also provide the stronger influences `q-add` and `q-sub`.

In addition to the influences described above, it is possible to define many other types of compositional equations. Two that are used in QPC include `q-mult` and `q-div`, which are defined as follows:

$$\begin{aligned}
N &= \prod_{y \in S} y \text{ where } S = \{y : (\texttt{q-mult } x\ y)\} \\
D &= \prod_{y \in S} y \text{ where } S = \{y : (\texttt{q-div } x\ y)\} \\
X &= \frac{N}{D}.
\end{aligned}$$

The influences may be composed with each other in a very natural fashion. Similarly `q-mult` and `q-div` may be composed readily with each other. Although it would be possible to define a composition of the two groups, it would be rather arbitrary. Consequently, QPC disallows combinations of other influences with `q-mult` or `q-div`.[5]

3.4.3 Correspondences

Influences and the other components of equations described in the preceding section provide essential means for defining the equations introduced by model fragments. It is also desirable, however, to provide additional information about both qualitative landmark values, and the otherwise weakly specified monotonic function constraints. This is done by specifying tuples of landmark values that satisfy the constraints. If constraints are algebraic, such as `add` or `mult`, then this helps to define the landmarks. If the constraints are qualitative monotonic functions, then it restricts the set of actual equations that might satisfy them.

QPC distinguishes between three sorts of correspondences.

1. The QSIM *corresponding value* applies to a particular constraint.

$$\text{corresponding\_value}(s, c, x_1, v_1, \ldots, x_n, v_n) \tag{3.4}$$

   This states that in scope s, the n-ary constraint c on variables $x_1, \ldots, x_n$ is satisfied when $x_1, \ldots, x_n$ take on the values $v_1, \ldots, v_n$ respectively

2. The QP theory *correspondence* applies whenever there are a particular set of variables influencing a variable.

$$\begin{aligned} &\text{correspondence}(s, y, v, x_1, v_1, \ldots, x_n, v_n) \Rightarrow \\ &\quad \forall c \ \text{constraint}(s, c) \wedge \text{dependent\_var}(c, y) \wedge \text{args}(c) = \{x_1 \ldots x_n\} \\ &\qquad \Rightarrow \text{corresponding\_value}(s, c, y, v, x_1, v_1, \ldots, x_n, v_n). \end{aligned} \tag{3.5}$$

   A correspondence on variables $y, x_1, \ldots, x_n$ holds when there is a constraint on precisely these variables, and whose dependent variable is the first variable, y. If the correspondence holds, then it simply establishes a corresponding value for the constraint. The correspondence violates the principle of compositionality that QP theory and QPC otherwise attempt to maintain. It is used to impose restrictions on the constraints that result from resolving influences relating certain variables. The above definition is, I believe, equivalent to the correspondence used by Forbus in QP theory. It has the characteristic that the direction of the influences is irrelevant. For example, a correspondence `(correspondence x 0 y 0)` applies given the sole influence `(q+ x y)` or `(q- x y)`. In principle, this seems like a dangerous bit of freedom,

[5] Rickel has argued for allowing them to be combined, with the following semantics: (N/D)*f(...). The reason is to allow resistances to be succinctly modeled. In the simple case, resistance is ignored and, for example, ΔP influences flow. Resistance is described by a separate model fragment that asserts a resistance R is `q-div` to flow. If we allow `q-div` to be combined with other influences, we get the desired result: flow = $\Delta P/R$. Without this sort of combination, resistances may still be combined and made `q-` to flow, but then the actual algebraic relation is lost.

and might lead to surprising results. So far, however, this has not led to practical difficulties.

3. It is sometimes useful to provide a *strong correspondence* that holds regardless of the presence of a constraint. For example, when amount is zero, level is zero, regardless of any other variables that might influence level such as diameter.

$$\begin{aligned} &\text{strong\_correspondence}(s, y, v, x_1, v_1 \ldots, x_n, v_n) \Rightarrow \\ &\quad \text{value}(s, x_1, v_1) \ldots \text{value}(s, x_n, v_n) \Rightarrow \text{value}(s, y, v). \end{aligned} \tag{3.6}$$

3.5 The Importance of Soundness for a Model Builder

3.5.1 Sound Systems are Reliable

We would like QPC to be a trustworthy system. There are a number of desiderata for a model building and simulation system to be trustworthy. First and foremost, we must ensure that any correct model of the system will be built. Secondarily, it would be nice if only enough models were built to cover the set of correct ones. The first property is that of soundness – QPC will not draw incorrect inferences. The second property is that of completeness – QPC will make the strongest inference possible. Unfortunately, guaranteeing completeness in a system such as QPC (or in almost any system that can be viewed as a theorem prover for an interesting subset of FOL) is undecidable.

3.5.2 Soundness is not for Everyone

It is worth noting that in some settings, an unsound or heuristic approach may be adequate or indeed preferable to sound inference. There are two important classes of such settings: those in which an occasional incorrect answer is acceptable, and those in which an unsound subsystem may be embedded in a larger system whose overall behavior is sound.

A heuristic diagnosis system that quickly, inexpensively, and accurately diagnoses a large percentage of failures may be preferable to one that requires extensive intrusive tests to rule out rare failures, or which generates a large set of possible diagnoses. For applications, such as tutoring, it may be more important to provide a single, typical prediction than many possible ones. This is especially true if a small number of behaviors are much more likely than a rather large number of rare ones.

Heuristically guided search is the classic example of including an unsound subsystem, the evaluation function, into a sound system. An example is a diagnosis system that comprises a heuristic diagnosis engine and a tester that can identify a correctly functioning device. If the tester is sound, then the unsoundness in the heuristic component can be compensated for by actually performing suggested repairs, rejecting those that fail in testing, and diagnosing the device again.

3.5.3 Consequences of Soundness

A sound tool is a reliable one because its answers can be trusted, and for many tasks, a reliable sound tool is the most appropriate. QPC is reliable because it guarantees to produce all of the possible models of the scenario. Two important classes of tasks are readily supported:

1. Guaranteed outcomes. If all models predict a single outcome, then it is guaranteed to occur.

2. Guaranteed absence. If no model predicts an outcome, then it is guaranteed not to occur.

Intermediate situations can also be useful in a monitoring and control setting in which observations can be used to select paths through the possibilities, or in performing hazard/operations studies. These and other applications of QPC are discussed in Chapter 5.

If QPC or other modeling tools are to be trusted by engineers and other users, they need to provide readily understood soundness guarantees.

It is clear from the preceding discussion that the utility of a soundness guarantee is dependent on the quality of the domain theory and the accuracy of the scenario description. It is not a guarantee that there will be any correspondence between the physical situation being modeled and the predictions of the model. No system can make such a claim without being highly suspect. Thus, in domains without a precise well understood theory, the soundness guarantee is not of great use. Using qualitative models, however, tremendously extends the notion of "precision" and "well understood", so that a rather broad class of domains may be thought of as having adequate theories.

3.6 Formal Language

This section presents a slightly more concise presentation and axiomatization of the language elements, which is consistent with the preceding description and simplifies formal analysis. The QPC modeling language is translated into a set of proper axioms in a formal language, $\mathcal{L}$, a subset of predicate calculus. We follow Genesereth and Nilsson [Genesereth and Nilsson, 1987] in the (standard) definition of terms and sentences. We also allow an n-ary function to be written as an $n+1$-ary relation.

The constant symbols of $\mathcal{L}$ include the disjoint sets of entities, instances, variables, and magnitudes. The relations *entity*, *instance*, *variable*, and *magnitude* are true of elements of the corresponding sets of constants. A *quantity* is a variable or a magnitude.

The relation symbols of $\mathcal{L}$ may be divided into those that are defined within $\mathcal{L}$, and those that are introduced by the author of the domain theory to describe the structural relations between entities in the domain. The predefined relations include:

The unary relations *entity*, *magnitude*, *variable*, *quantity*, *instance*,*active*, and *inactive*.

The quantity relations $<, >, =$.

Mathematical relations whose relation including $+, *, /$, d/dt, and the composable relation $Q^+, Q^-, I^+, I^-, Q_{add}, Q_{sub}, Q_{mult}$, and Q_{div}. The mathematical relations must all have a defined mapping into the QSIM QDE constraint language.

A set of functions introduced by entity and model fragment definitions.

Structural relations are introduced by the domain theory author (via `defrelation`, and must be disjoint from the above.

In order to succinctly describe the legal sentences of $\mathcal{L}$, we introduce the following abstract syntax:

E A formula of the form $\forall$ $x_1, \ldots, x_n$ entity(x_1) $\wedge \ldots \wedge$ entity(x_n).

Q A conjunction of quantity relations of the form $\pi(\tau_1, \tau_2)$ or $\neg\pi(\tau_1, \tau_2)$ where $\pi \in \{<, =, >\}$, and the τ_i are terms denoting quantities.

S A conjunction of structural relations $\pi(\tau_1, \ldots, \tau_n)$ or $\neg\pi(\tau_1, \ldots, \tau_n)$ where π is a structural relation, and each τ_i denotes an entity.

C A conjunction of structural and quantity relations (conditions)

M A conjunction of mathematical relations.

Example 1 *The schematic sentence* **E** $\wedge$ **S** $\rightarrow$ **S** *represents sentences such as:* $\forall$ *x,y, entity(x)* $\wedge$ *entity(y)* $\wedge$ *contains(x,y)* $\rightarrow$ *containedBy(y,x)*
where contains and containedBy are structural relations.

A sentence in $\mathcal{L}$ may be:

1. A ground instance of any relation or its negation.

2. **E** $\wedge$ **S** $\rightarrow$ **S** (structural axioms)

3. **E** $\wedge$ **Q** $\rightarrow$ **Q** (quantity axioms)
   The quantity axioms must be consistent with the standard axioms of order and numbers. In fact, they may be viewed as a partial axiomatization of standard mathematics and the qualitative calculus.

4. An axiom introduced by a model fragment definition.
   A model fragment definition is a tuple $\langle$ μ, E, S, Q, C, M $\rangle$, where μ is the unique name of the model fragment and the remaining items are formulae of the syntactic types **E**,**S**, **Q**, **C**, and **M** respectively, any of which may be empty. Let n be the number of entities in E. The definition name μ is an n-ary function symbol. The definition introduces the following axioms:

   E $\leftrightarrow$ $\exists$ i : i=μ($x_1, \ldots, x_n$) $\wedge$ instance(i)
   $\forall$ $x_1, \ldots, x_n$,i : i=μ($x_1, \ldots, x_n$) $\wedge$ S $\wedge$ Q $\leftrightarrow$ active(i)
   $\forall$ $x_1, \ldots, x_n$,i : i=μ($x_1, \ldots, x_n$) $\wedge$ active(i) $\rightarrow$ C $\wedge$ M

And for each quantity condition q in Q, the axiom:
$\forall$ $x_1,\ldots,x_n$,i : i=$\mu(x_1,\ldots,x_n)$ $\wedge$ $\neg$q $\leftrightarrow$ inactive(i)

5. An axiom introduced by an entity definition.
   An entity definition is a tuple $\langle$ μ, E, S, Q, C $\rangle$, that is equivalent to the model fragment definition $\langle$ μ, E, S, Q, $\emptyset$, $\emptyset$ $\rangle$, and the additional axiom:

   $\forall$ $x_1,\ldots,x_n$,i : i=$\mu(x_1,\ldots,x_n)$ $\wedge$ active(i)
   $\rightarrow$ $\exists$ e : e=$\mu_e(x_1,\ldots,x_n)$$\wedge$ entity(e) $\wedge$ C

   The n-ary function symbol μ_e is a unique function introduced by the definition.

The syntactic restrictions embodied in the definition of $\mathcal{L}$ have the consequence that no structural relation can be inferred from quantity conditions except through the intermediary implication that some model fragment instance is active. An implication E $\wedge$ S $\wedge$ Q $\rightarrow$ C can be rewritten as a model fragment $\langle$ m,E,S,Q,C,$\emptyset$ $\rangle$

The syntactic restrictions also ensure that instances and entities are only introduced by the specific axioms that result from model fragment definitions.

Definition 5 *A proof of a sentence ϕ from a set of sentences Σ is a finite sequence of sentences in which (1) ϕ is the last element, and (2) every element is a member of Σ, a logical axiom, or the result of applying modus ponens to sentences earlier in the proof. We write Σ proves ϕ: $\Sigma \vdash \phi$.*

An implementation of QPC must provide a sound inference procedure. Although the procedure need not be complete, we do require that it be able to prove any sentence ϕ that follows from a set of sentences Σ using *only* modus ponens (i.e. no logical axioms are required). We write this restricted type of provability: $\Sigma \vdash_{mp} \phi$.

Definition 6 *A set Σ of sentences in $\mathcal{L}$ is stratifiable if the entities in Σ can be partitioned into levels L_0, ..., L_n such that for each element of L_i, a proof of its existence can be constructed from Σ using modus ponens without reference to any elements of L_i,...,L_n.*

This notion is used to delimit the set of entities that the algorithm needs to introduce. The underlying intuition is that all entities should be grounded in the original description of a scenario, and "spontaneous generation" should be prevented. For example, when modeling fluid flow into an empty container, it is reasonable to introduce a new liquid entity in the container because its existence is implied by the existence of the flow. In the same situation it would not be reasonable to introduce a second container connected to the first unless it were directly implied by the domain theory.

It is possible to determine if a domain theory is stratifiable by examining the axioms for introducing entities. A graph is induced by the defentity forms, in which each form contributes a tree whose root is the created entity, and whose leaves are the participants.

Two nodes in the graph are distinct if the corresponding entities can be proven to be distinct from each other. If the subgraph reachable from the nodes not distinct from the initial entities in a scenario is acyclic, then the theory is stratifiable for that scenario.

Definition 7 *A database (Δ) is a consistent finite stratifiable set of sentences in $\mathcal{L}$.*

The following functions provide a convenient notation for describing databases.

$$\begin{aligned} \text{entities}(\Delta) &\equiv \{e : \Delta \vdash_{mp} \text{entity}(e)\} \\ \text{instances}(\Delta) &\equiv \{i : \Delta \vdash_{mp} \text{instance}(i)\} \end{aligned}$$

Assumption 2 *The set of entities entailed by a database is finite.*

No compositional modeler that explicitly represents the set of entities and instances that are present in a model can handle an infinite set of entities. Even under this restriction, however, QPC is able to reason about systems that generate arbitrarily large finite numbers of entities in finite time. Examples include breaking chalk or dripping water. Each of these processes takes time to create a new object, so prediction over finite time will yield a finite number of objects. Systems that generate infinite sets of entities in finite time, however, cannot be handled. An example would be a domain theory including a defentity form that introduces a left half and a right half for every object, with each half also being an object. An infinite set of objects is implied to exist at the first instant.

It is possible to guarantee that a domain theory will introduce a finite number of entities by ensuring that there is a well founded measure that decreases whenever a new object is introduced.

Definition 8 *If $\Delta \vdash_{mp} \mu(t_1,\ldots,t_n,\ i)$, let $\langle\ \mu,\ E,S,Q,C,M\ \rangle$ be the definition of μ, and $x_1 \ldots x_n$ be the variables quantified in E. The* conditions *of i are the union of the conjuncts in S and Q with terms t_j substituted for the variables x_j. If $\Delta \not\vdash_{mp}$ active(i) and $\Delta \not\vdash_{mp}$ $\neg$active(i), then the set of ambiguous conditions of i in Δ is the subset of conditions such that $\Delta \not\vdash_{mp} c$ and $\Delta \not\vdash_{mp} \neg c$, otherwise it is the empty set.*

Note that if all of the ambiguous conditions could be proven true, then *active*(i) would follow. Contrariwise, if any of the ambiguous conditions could be proven false, then $\neg$*active*(i) would follow. This observation provides the basis for the refinement of incomplete databases.

Definition 9 *An instance i is ambiguous in a database Δ iff $\Delta \not\vdash_{mp}$ active(i) and $\Delta \not\vdash_{mp}$ $\neg$active(i).*

Definition 10 *A database Δ is complete iff Δ contains no ambiguous instances, and the set of instances and entities is maximal.*

The set of instances and entities is maximal in the sense that adding any entity or instance to Δ would make it inconsistent or violate the stratifiability requirement. The latter occurs if irrelevant entities, ones not rooted in the initial scenario description, are added to Δ.[6]

Definition 11 *A domain theory is a database consisting of sentences in $\mathcal{L}$ of types 2–5.*

Definition 12 *A scenario description is a database consisting of sentences in $\mathcal{L}$ of type 1 that are ground instances of* **E** *or* **C**.

3.7 Algorithm

QPC is provided with a domain theory, DT, and a scenario description consisting of a set of entities E_0, a set of ground structural relations S_0 on them, and a set of ground quantity relations Q_0 on their quantities. QPC constructs a tree to describe the possible behaviors of the situation. A node in the tree is either a database Δ or a qualitative state S with its associated QDE. The root of the tree is the database formed from $DT \cup E_0 \cup Q_0 \cup S_0$. An edge in the tree may be one of the following:

1. $\langle \Delta_1, \Delta_2 \rangle$ *refinement*
   Δ_1 is an incomplete description of a situation. Δ_2 is a more complete description that results from adding some ground conditions to Δ_1.

2. $\langle \Delta, S \rangle$ *initial state*
   Δ is a complete description of a situation. The qualitative state S is an initial state that follows from Δ.

3. $\langle S_1, S_2 \rangle$ *state successor*
   S_2 is a QSIM state successor of S_1. These edges represent the temporal evolution of the situation.

4. $\langle S, \Delta \rangle$ *transition*
   S is a transition state that is crossing the boundary of the applicability of its QDE. Δ is a description of the situation at the same time as S, after the transition.

Each path from the root to a leaf of the tree describes one possible temporal evolution of the physical situation being modeled: a sequence of models, and within each model, a sequence of qualitative states.

Figure 3.4 provides a pseudocode description of the QPC algorithm. The main steps are:

[6] An alternative way of characterizing completeness is to say that adding ground formula C to Δ (such that $\Delta \cup$ C is consistent) will not imply the existence of any additional entities or model fragments. This weaker characterization is implied by definition 10.

```
define QPC(Δ)
  forall Δ_r in refinements(Δ)
    edge(Δ, Δ_r)
    B = QSIM (qde(Δ_r), initial(Δ_r))
    forall b in B
      edge(Δ_r, first(b))
      if transition(last(b))
        then
          Δ_t = entities(Δ_r) ∪ conditions(last(b)) ∪ DT
          edge(last(b), Δ_t)
          QPC(Δ_t)
```

If initially, the tree is empty, DT is the domain theory, and $\Delta_0 = \mathrm{DT} \cup \mathrm{E}_0 \cup \mathrm{C}_0$, then QPC($\Delta_0$) creates a tree of the possible behaviors of the physical situation with entities E_0 and conditions C_0 under the domain theory DT.

Figure 3.4: Pseudocode description of the QPC algorithm.

1. Refine the incomplete description Δ into a set of complete descriptions Δ_r of the same situation. The refinement operation returns a set containing all complete databases that are consistent with Δ (see Lemma 8). An edge linking Δ to each of its refinements is added to the tree.

2. A complete database Δ_r defines a unique QDE and set of initial conditions (see section 3.8.3) which are provided to QSIM. QSIM produces a set B including all possible behaviors consistent with the QDE and initial conditions.

3. Each behavior b in B is then processed. An edge from Δ_r to the first state of b is added to the tree. The first state is already linked to its successor states by QSIM, which is why no state to state edges need be explicitly added. If the final state of b is a transition state, then a new description of the situation after the transition must be constructed. The transition database Δ_t is constructed from the entities of Δ_r, the quantity conditions derived from the transition state, and the domain theory. An edge from the final state of b to Δ_t is added to the tree.

4. QPC recurses on the transition database Δ_t to construct all possible behaviors of the situation following the transition.

Figure 3.6 provides a pseudocode description of the refinement algorithm. Refinement constructs the set of all complete databases consistent with and rooted in an initial database Δ. This is accomplished by creating a set of databases in which the ambiguous instances in

define refinement(Δ)
  if ambiguous(Δ) = $\emptyset$
    then $\{\Delta\}$
    else $\bigcup_{\forall c \in \mathrm{children}(\Delta)}$ refinement(c)

Figure 3.5: `Refinement` returns the set of all complete databases rooted in an initial database.

define children(Δ)
  S = forall i in instances(Δ) collect ambiguousConditions(i,Δ) **1**
  forall A in powerset(S) **2**
    — *instances with ambiguous conditions in A will be active.*
    — *instances with ambiguous conditions in S–A will be inactive.*
    for each D in deniers(S–A) **3**
      — *each D negates every conjunct C in S–A.*
      collect $\Delta \cup$ D $\cup \bigcup_{a \in A} a$ **4**

define deniers(S)
  — *S is a set of sets $\{s_1 \ldots s_n\}$.*
  — *Each s_i is a set of literals.*
  replace each literal in each element of
    cartesianProduct($s_1, \ldots, s_n$)
    with its negation.

Figure 3.6: The `children` of a database Δ force every ambiguous instance in Δ to be either active or inactive.

Δ are no longer ambiguous. These databases are referred to as the children of Δ. A child Δ_c of Δ is formed by adding conditions (ground in Δ) so that every ambiguous instance i in Δ can be proven to be either *active*(i) or $\neg$*active* in Δ_c. The children of Δ contain all possible partitions of the ambiguous instances in Δ into active and $\neg$active. Note that a child need not be complete, since new entities may be introduced by the new active instances, resulting in new (ambiguous) instances. Thus, each child must be recursively refined.

Example 2 *Suppose Δ has ambiguous instances a, b, and c, which have ambiguous conditions $\{a_1, a_2\}$, $\{b_1, b_2\}$, and $\{c_1\}$ respectively. In step 1, $S = \{\{a_1, a_2\}, \{b_1, b_2\}, \{c_1\}\}$. In step 2, A is bound to each element of the power set of S in turn. Consider the case in which $A = \{\{a_1, a_2\}\}$. In step 3, $S - A = \{\{b_1, b_2\}, \{c_1\}\}$. The deniers of $S - A$ are formed by first taking its Cartesian product, which is $\{\{b_1, c_1\}, \{b_2, c_1\}\}$, and then negating each of the literals, resulting in $\{\{\neg b_1, \neg c_1\}, \{\neg b_2, \neg c_1\}\}$. Each element of the deniers makes every instance, whose ambiguous conditions are in $S - A$, inactive. For this binding for A, two children are formed:*

$$\Delta \cup \{\neg b_1, \neg c_1\} \cup \{a_1, a_2\}$$
$$\Delta \cup \{\neg b_2, \neg c_1\} \cup \{a_1, a_2\}$$

In each of these children a is active and b and c are inactive. Two children are necessary because b can be made inactive in two different ways.

In the worst case, computing the set of refinements is extremely expensive. The worst case occurs when there are many instances with ambiguous conditions that do not overlap. The actual expense, however, is ameliorated by the following considerations:

- The number of ambiguous instances tends to be small compared to the number of instances.
- In a scenario the ambiguous conditions tend to be interdependent. There is usually a small number of key pieces of information that is unknown.
- Sets of conditions may be mutually contradictory. This can be used to eliminate members of the powerset and the set of deniers.

The actual cost tends to be considerably lower than the raw number of ambiguous instances would suggest.

3.8 Correctness of QPC

Informally, QPC is correct if it constructs all behaviors that are consistent with an initial (perhaps incomplete) description of a situation. After building up a series of intermediate results, this section formally states and proves the correctness of the QPC algorithm.

3.8.1 Refinement Returns All Complete Initial Models

Refinement is correct if it returns all of the complete models of a scenario. We show this by proving the three theorems:

1. refinement terminates,

2. every refinement is a complete model, and

3. every complete model of a scenario is in the scenario's refinements.

Lemma 1 *For all databases Δ, $\Delta_c \in$ children(Δ),*

$$ambiguous(i,\ \Delta) \rightarrow \neg ambiguous(i,\ \Delta_c).$$

Let i be an arbitrary ambiguous instance in Δ and C be the non-empty set of i's ambiguous conditions in Δ. Let us examine the definition of `children` to prove that in every child Δ_C of Δ, either Δ_c proves ($\vdash_{mp}$) that the conjunction of conditions C is true, in which case i is active in Δ_c, or false, in which case i is $\neg$active.

In step 1, each element of the set `S` is the set of ambiguous conditions for an ambiguous instance in Δ. The set `S` contains one such element for every ambiguous instance in Δ. Therefore, $C \in$ `S`. In step 2, `A` is bound in turn to each element of the powerset of `S`. We consider two cases:

In the first case, $C \in$ `A`. Every child, Δ_c is formed in step 4 and includes at least Δ and every member of `A`. Therefore, $\Delta_c \supseteq \Delta \cup C$, and since $\Delta \cup C \vdash_{mp}$ active(i), $\Delta_c \vdash_{mp}$ active(i).

In the second case, $C \notin$ `A`. In this case, every child proves $\neg$active(i). The `children` procedure ensures this by including the negation of some element of C into each child. Each child is formed from in step 4, and includes Δ, every member of `A`, and an element of the `deniers` of `S` $-$ `A`. In the case under consideration, $C \notin$ `A`. By hypothesis, C is an element of `S`, so C is also an element of `S` $-$ `A`. Each element `D` of the `deniers` of `S` $-$ `A` contains one negated element of C. Therefore, $\Delta \cup$ `D` $\vdash_{mp}$ $\neg$active(i) via one of the axioms induced by i's model fragment's definition.

Therefore, if $C \in$ `A`, $\Delta_c \vdash_{mp}$active(i), and if $C \notin$ `A`, $\Delta_c \vdash_{mp}\neg$active(i), which implies $i \notin$ ambiguous(Δ_c).$\square$

Lemma 2 *For all databases Δ, $\Delta_c \in$ children(Δ),*

$$entities(\Delta) = entities(\Delta_c) \rightarrow instances(\Delta) = instances(\Delta_c).$$

Since $\Delta_c \supseteq \Delta$ and inference ($\vdash_{mp}$) is monotonic, it follows that instances(Δ_c) $\supseteq$ instances(Δ). Thus, it will suffice to show instance $i \in$ instances(Δ_c) $\rightarrow i \in$ instances(Δ). Suppose that i is an arbitrary instance of model fragment μ in Δ_c. I.e. $\Delta_c \vdash_{mp}\mu(e_1, \ldots, e_n) = i$. By hypothesis, $e_1, \ldots, e_n$ are also entities in Δ, so $\Delta \vdash_{mp}\mu(e_1, \ldots, e_n) = j$. Since μ is a function, $i = j$. Therefore, instances(Δ) = instances(Δ_c). $\square$

Lemma 3 *For all databases* Δ, $\Delta_c \in$ *children(*Δ*),*

$$\textit{entities}(\Delta) = \textit{entities}(\Delta_c) \rightarrow \textit{ambiguous}(\Delta_c) = \emptyset.$$

Follows directly from Lemma 1, Lemma 2.□

Theorem 1 *Refinement terminates.*

Consider refinement(Δ). If ambiguous(Δ)=$\emptyset$, then refinement terminates immediately. If ambiguous(Δ) $\neq$ $\emptyset$, then refinement recurses on each of the children. Let Δ_c be a child. By Lemma 3, if entities(Δ_c) = entities(Δ), ambiguous(Δ_c) = $\emptyset$, and refinement(Δ_c) terminates. Otherwise, because $\Delta_c \supset \Delta$, entities(Δ_c) $\supset$ entities(Δ), so the number of entities increases. Therefore, each recursive call either terminates, or increases the number of entities. By Assumption 2, a database must entail a finite set of entities at any instant in time, therefore refinement must terminate.□

Lemma 4 *For all databases* Δ, $\Delta_r \in$ *refinement(*Δ*),*

$$\textit{ambiguous}(\Delta_r) = \emptyset.$$

Refinement only terminates if ambiguous(Δ_r)=$\emptyset$. By Theorem 1, refinement terminates.□

Definition 13 *A formula* σ *is* <u>*ground in*</u> *a set of formulae* Σ *if and only if* σ *is ground and every constant symbol in* σ *is also a constant symbol in* Σ*.*

For example, the atomic formula $p(a)$ is ground in the set of formulae $\{p(b), q(a)\}$.

Lemma 5 *For all databases* Δ*, if ambiguous(*Δ*)=*$\emptyset$*, then the set entities(*Δ*) is maximal. That is, if* Γ *is a set of conditions ground in* Δ*, then entities(*Δ*) = entities(*$\Delta \cup \Gamma$*).*

Let E_Δ = entities(Δ), E_Γ = entities($\Delta \cup \Gamma$), and $E_0, \ldots, E_n$ be a stratification of E_Γ. Let E_i be the first level such that $E_i \not\subseteq E_\Delta$. If there is no such E_i, then $E_\Delta = E_\Gamma$. Otherwise, choose $e \in E_i - E_\Delta$. From the definition of stratification, there must be a proof of the existence of e that does not employ any entity not in E_Δ. Since all entities are introduced by an instance of the axiom schema 5, there is some function μ_e such that $e = \mu_e(e_1, \ldots, e_n)$ where $e_1, \ldots, e_n$ are elements in E_Δ. Thus, $\Delta \vdash_{mp} \mu(e_1, \ldots, e_n) = i \wedge$ instance(i). Since Δ is unambiguous, $\Delta \vdash_{mp}$active(i), or $\Delta \vdash_{mp} \neg$active(i). In the first case, $\Delta \vdash_{mp} e = \mu_e(e_1, \ldots, e_n)$, which contradicts the construction of e. In the second case, the first axiom from the model fragment definition of μ_e yields $\Delta \vdash_{mp} \neg(e = \mu_e(e_1, \ldots, e_n))$, and thus $\Delta \cup \Gamma \vdash_{mp} \neg(e = \mu_e(e_1, \ldots, e_n))$, which contradicts the hypothesis. Therefore, there can exist no e in E_Γ that is not also in E_Δ.□

This lemma and proof makes clear two important points about an unambiguous database Δ. First, adding conditions ground in Δ to Δ will not result in any new entities being inferred. Second, it shows the role of stratification. Adding any entity to Δ would result in Δ becoming unstratifiable, in which case there would be entities that were not rooted in the initial description of the scenario.

Lemma 6 *For all databases Δ, if ambiguous(Δ) $= \emptyset$, then the set instances(Δ) is maximal. That is, if Γ is a set of conditions ground in Δ, then instances(Δ) = instances($\Delta \cup \Gamma$).*

Let $i = \mu(e_1, \ldots, e_n)$ be an instance of $\Delta \cup \Gamma$, where $e_1, \ldots, e_n$ are entities of $\Delta \cup \Gamma$. By lemma 5, $e_1, \ldots, e_n$ are also entities of Δ. Therefore $\Delta \vdash_{mp} i = \mu(e_1, \ldots, e_n)$ via an instantiation of the axiom schema 4.□

Lemma 7 *For all databases Δ, if ambiguous(Δ)=$\emptyset$, then Δ is complete.*

This follows directly from the definition of complete (Definition 10), Lemma 5, and Lemma 6.□

Theorem 2 *For all databases Δ, $\Delta_r \in$ refinement(Δ), Δ_r is complete.*

Follows trivially from Lemma 7 and Lemma 4.□

Next, we need to prove that `refinement` will construct all complete databases that are consistent with the database to be refined. The proof supposes that a daemon has presented a complete database Δ that describes the same scenario as the incomplete initial database Δ_0. Our job is to show that `refinement` generates a database Δ_r that has the same set of model fragments as Δ. If they have the same set of model fragments, then they will entail identical QDEs.

Formalizing the condition that Δ describes the same scenario as Δ_0 takes some care. Clearly $\Delta \supseteq \Delta_0$ must be true if Δ is at least as complete as Δ_0. It must also be the case, however, that the entities implied by Δ must be rooted in the entities of Δ_0. This requirement is enforced by a restriction on the stratification of entities in Δ and Δ_0.

The proof consists of a major lemma, in which we show that there is a refinement with the same set of entities as Δ. A simple lemma extends this result to the sets of instances, and this leads directly to the main theorem.

In order to simplify writing the proof, we introduce some additional notation. We write Δ^1 to indicate a child of Δ, Δ^2 to mean a child of a child of Δ, and so on. Furthermore, we write $E(\Delta)$ to indicate the entities of Δ.

Lemma 8 *For all databases Δ_0, and complete databases Δ, such that $\Delta_0 \subseteq \Delta$, the domain theory of Δ_0 is identical to the domain theory of Δ, and every stratification of $E(\Delta_0)$ and $E(\Delta)$ sharing the same first stratum E_0, there is a $\Delta_r \in$ refinement(Δ_0) such that $\Delta_r \subseteq \Delta$ and $E(\Delta_r) = E(\Delta)$.*

Suppose that Δ_0^1 is a child of Δ_0, Δ_0^2 one of its children, and so on through Δ_0^n. To prove the theorem, we need to show that $\Delta_0^n \subseteq \Delta$ and $E(\Delta_0^n) = E(\Delta)$. The central insight of the proof is the that the `children` operation establishes a stratification of the entities in Δ as well as each Δ_0^1. At each application of `children`, the set of entities in Δ_0^i is bounded from below by the entities in the first i strata of Δ and bounded from above by the entities in Δ. When refinement terminates with $\Delta_0^n = \Delta_r$, the upper and lower bounds have converged and the set of entities in Δ_0^n is equal to the set of entities in Δ.

We can restate the consequence of the theorem as follows:

$$\Delta_0 \subseteq \Delta_0^n \subseteq \Delta \text{ and } \bigcup_{0\leq j\leq n} E_i \subseteq E(\Delta_0^n) \subseteq E(\Delta).$$

The proof is by induction on i.
Base case: $\Delta_0 \subseteq \Delta_0^0 \subseteq \Delta$ and $E_0 \subseteq E(\Delta_0^0) \subseteq E(\Delta)$.
By definition, the zeroth child $\Delta_0^0 = \Delta_0$, and the remainder follows from the statement of the theorem.
Inductive hypothesis:

$$\Delta_0 \subseteq \Delta_0^i \subseteq \Delta, \text{ and } \bigcup_{0\leq j\leq i<n} E_j \subseteq E(\Delta_0^i) \subseteq E(\Delta).$$

Inductive step:

$$\Delta_0 \subseteq \Delta_0^{i+1} \subseteq \Delta, \text{ and } \bigcup_{0\leq j\leq i+1\leq n} E_j \subseteq E(\Delta_0^{i+1}) \subseteq E(\Delta).$$

First we show that $E_{i+1} \subseteq E(\Delta_0^{i+1})$.

Suppose ambiguous$(\Delta_0^i) = \emptyset$, then by lemma 5, adding the entities from E_{i+1} to Δ_0^i would result in an inconsistency, and consequently, imply that Δ is also inconsistent. Therefore, ambiguous$(\Delta_0^i) \neq \emptyset$.

Let e be an element of E_{i+1} such that $e = \mu_e(e_1, \ldots, e_m)$, μ is the function symbol (and name of the model fragment) introduced by the entity definition that introduced e. There are three cases to examine:

1. $\Delta_0^i \vdash_{mp} \mu_e(e_1, \ldots, e_m) = e$.
   In this case, $e \in E(\Delta_0^i)$. Since $E(\Delta_0^i) \subseteq E(\Delta_0^{i+1})$, it follows that $e \in E(\Delta_0^{i+1})$.

2. $\Delta_0^i \vdash_{mp} \neg\mu_e(e_1, \ldots, e_m) = e$.
   In this case, $\Delta_0^i \cup \mu_e(e_1, \ldots, e_m) = e$ is inconsistent, and consequently Δ is inconsistent, which is impossible.

3. $\Delta_0^i \not\vdash_{mp} \mu_e(e_1, \ldots, e_m) = e$ and $\Delta_0^i \not\vdash_{mp} \neg\mu_e(e_1, \ldots, e_m) = e$.
   In this case, the entities $e_1, \ldots, e_m$ are in $E(\Delta_0^i)$, so $\Delta_0^i \vdash_{mp} \mu(e_1, \ldots, e_m) = i$. There is a child $\Delta_0^{i+1} \vdash_{mp}$ active(i), and consequently $\Delta_0^{i+1} \vdash_{mp} \mu_e(e_1, \ldots, e_m) = e$. By generalization, there must be a child Δ_0^{i+1} such that $E_{i+1} \subseteq E(\Delta_0^{i+1})$.

Note that when $i = n-1$, $E(\Delta_0^{i+1}) = E(\Delta)$, since $\bigcup_{0\leq j\leq n} E_j = E_\Delta$.

Next, we show that $\Delta_0^{i+1} \subseteq \Delta$. Consider each ambiguous instance x in Δ_0^i. Since $\Delta_0^i \subseteq \Delta$, and Δ is complete, it must be the case that x is proven to be either active or not active in Δ. If x is active in Δ, then collect the ambiguous conditions of x in Δ_0^i into A. If x is not active, then collect into the set B the (non empty) subset of the ambiguous conditions that are false in Δ. Note that the union of Δ_0^i, the elements of A, and the negation of the literals in the elements of B is a subset of Δ. Now consider `children`(Δ_0^i). In step 1, each element of the set `S` is the ambiguous conditions of an ambiguous instance in Δ_0^i and there is one element for every such instance. Clearly, $A \subseteq$ `S`, thus in step 2, the variable `A` will eventually take on the value A. Consider the children created for this binding of `A`. The elements of the set B `S−A` can be paired with elements of the set `S−A`, such that each element of B is a subset of the corresponding element of `S−A`. Therefore, the Cartesian product of the elements of B is a subset of the Cartesian product of the elements of `S−A`, and `deniers`(B) is a subset of `deniers(S−A)`. In step 3, any denier `D` that is also in `deniers`(B) will form a child that is a subset of Δ.□

The above lemma has the corollary:

Corollary 1 *Given Δ_0, Δ, and Δ_r according to Lemma 8, then instances(Δ) = instances(Δ_r) and for all i in instances(Δ), $\Delta \vdash_{mp}$ active(i) $\leftrightarrow \Delta_r \vdash_{mp}$ active(i).*

The proof of lemma 6 also applies here because the axioms induced by model fragment definitions are the same in both Δ and Δ_r, thus instances(Δ) = instances(Δ_r). Since Δ_r is complete, it is strong enough to prove active(i) or ¬active(i) for every instance i. Since $\Delta_r \subseteq \Delta$, Δ is a stronger theory than Δ_r, so it must also prove active(i) or ¬active(i) accordingly.□

Theorem 3 *For all databases Δ_0, and complete databases Δ, such that $\Delta_0 \subseteq \Delta$, and every stratification of $E(\Delta_0)$ and $E(\Delta)$ share the same first stratum E_0, there is a $\Delta_r \in$ refinement(Δ_0) such that $\Delta_r \subseteq \Delta$, active($\Delta_r$) = active($\Delta$), and inactive($\Delta_r$) = inactive($\Delta$).*

The theorem follows trivially from Lemma 8 and Corollary 1.□

3.8.2 Defining a QDE from a Database

Each complete database defines a unique QDE. In this section, we reprise the definition of a QDE and show the mapping from a database into a QDE.

A qualitative differential equation, QDE, is specified by:

- A set of *variables*, which are continuous functions of time whose domain is the extended reals.
- A set of *constraints*, which are simultaneous joint restrictions on the values of variables. They include addition, multiplication, d/dt, generic monotonic functions, etc..
- A *region* specification, which is a conjunction of inequalities that must be true of a state for the QDE to be applicable.[7]

[7]This is a slight simplification. The QSIM QDE language only allows for transition conditions (equality between a variable and a qualitative value), which must be derived from the region specification.

A QDE differs from an ODE in two ways. First, it may employ generic monotonic functions, such as $z = ((\mathtt{m} + -)\ x\ y)$, which states that z is monotonically increasing in x and decreasing in y. Such a monotonic function constraint covers an infinite set of actual functions including $z = x - y$, or $z = e^x/y$, and so on. The solution to a QDE involving monotonic functions must be valid under any possible substitution for the generic function of an actual member of the set. Second, variable values are described *qualitatively.* A qualitative value is a pair $\langle$ mag, dir $\rangle$. The second element, dir, is the symbol inc, std, or dec, representing the sign of the first derivative of the variable (positive, zero, or negative respectively). The notion of a *landmark value* is used to represent the magnitude. A landmark value is simply a symbolic name for a (perhaps unknown) member of the extended reals. The magnitude is equal to a landmark or in the open interval between two landmarks. For example, `level = ((0 top) inc)` indicates that level has a value between 0 and `top` and is increasing towards `top`.

Given a database Δ, its QDE is determined as follows:

constraints The constraints are simply the set of all mathematical relations M of type **M** such that $\Delta \vdash_{mp} M$. Note that these are readily inferred, as they follow from a single application of modus ponens from the activity of an instance.

variables The variables are those that appear in constraints.

regions The regions are determined by the quantity conditions of active and inactive model fragment instances. If an instance is active, then its quantity conditions are collected. If the instance is inactive, then the negation of its quantity conditions are collected.

Note that the QDE is completely determined by both active and inactive model fragment instances. The active instances contribute both constraints and (positive) region conditions. The inactive ones contribute (negative) region conditions.

3.8.3 Mapping between Quantity Relations and Qualitative Values

A qualitative state entails a set of quantity relations. We define the `QC`(s) to be a function that takes a qualitative state s and returns a set of quantity relations that are consistent with s. If v is a variable in s, then if $v = \langle l, ?\rangle$, then the quantity condition $v = l$ is true; if $v = \langle (l, u), ?\rangle$, then the relations $l < v$ and $v < u$ are true. Furthermore, there is a set of quantity relations implied by the constraints. For example, if $x + y = z$ and $y = 0$, then $x = z$. Each algebraic constraint implies some orderings and sets of them may imply others. Fortunately, extracting all quantity relations is not necessary, but the more that are extracted, the stronger the resulting relations will be. The precise method by which `QC` is implemented is left open, as it is possible to use very sophisticated mathematical reasoning techniques to extract more quantity relations from the state.

A database entails a set of initial values. We define `initial`(Δ) to be a function that returns a set of values for the variables in QDE(Δ). Given that QSIM will find all possible behaviors consistent with a QDE and a set of initial values, providing an empty set of initial values will produce all possible behaviors consistent with the QDE. Thus, it is not strictly necessary to provide any initial values in order to guarantee that the actual behavior will be found. Nonetheless, it is of considerable practical importance to restrict the set of possibilities as much as possible. A simple version of `initial` would examine the quantity space for each variable and return the narrowest qualitative value from the quantity space that bounds the variables initial value. If it is possible to show that the derivative of a variable is greater, less than, or equal to zero, then the direction of change can also be provided. As with `QC`, it is possible to use sophisticated inequality reasoning methods to determine the tightest bounds for a variable.

3.8.4 The Interaction between Simulation and the Database

In order to separate the logical inference associated with specifying a QDE from the mathematical inference associated with generating predictions, QPC must ensure that the simulation will not invalidate the conditions under which the QDE was specified. A slightly strengthened version of this requirement is defined in the following central theorem.

Theorem 4 *For all complete databases, Δ, and ground quantity conditions Σ, if ϕ is a structural relation,*

$$(\Delta \ \cup \ \Sigma \ \vdash_{mp} \phi) \rightarrow (\Delta \ \vdash_{mp} \phi)$$

A structural relation ϕ can only be inferred via axioms of the form

$\mathbf{E} \wedge \mathbf{S} \rightarrow \mathbf{S}$,

active(i) $\rightarrow$ $\mathbf{S}$, or

active(i) $\wedge$ i=$\mu(e_1, \ldots, e_m)$ $\rightarrow$ $\mathbf{C}$.

We prove the theorem by contradiction. Suppose that there exists some ϕ such that $\Delta \ \cup \Sigma \vdash_{mp} \phi$, but $\Delta \ \not\vdash_{mp} \phi$. The contradiction is shown by induction on the length of the proof of ϕ. The base case is obvious, as $\phi \not\in \ \Sigma$, and by hypothesis, $\phi \not\in \ \Delta$, there is no immediate proof of ϕ from $\Delta \ \cup \ \Sigma$. Assume that there is no proof of ϕ of length less than i. A proof of length i+1 can only result from the application of modus ponens to an axiom of one of the two forms mentioned above. If the first applies, then ϕ follows from structural relations alone, but then $\Delta \ \vdash_{mp} \phi$, which contradicts the hypothesis. Next consider the case in which an axiom of the second form applies. Since Δ is complete (by hypothesis), we know from Lemma 6 that $\Delta \cup \Sigma \ \vdash_{mp}$ instance(i) implies $\Delta \ \vdash_{mp}$ instance(i). Furthermore, if $\Delta \ \vdash_{mp}$ active(i), then $\Delta \ \vdash_{mp} \phi$, which contradicts the hypothesis, otherwise $\Delta \ \vdash_{mp}$ $\neg$active(i) and the axiom does not apply. Therefore, adding the quantity relations Σ to Δ does not enable any additional structural relations to be proven.

The meaning of Theorem 4 is that within the bounds of the model, simulation can never contradict any structural relation. This is important, because it means that the simulator need only attend to quantities.

The statement of theorem 4 is somewhat stronger than necessary. It would be sufficient to require that the structural conditions that determine the activity of model fragment instances cannot be negated by simulation. The formulation used here, however, is simpler to prove, and allows the logical representation to treat simulation as a black box. In fact, only the initial and final transition states need be represented explicitly.

Theorem 4 enables QPC to employ a hybrid architecture in which predictions are generated by a special purpose simulation engine. Importantly, the simulation engine need know nothing about the logical structure of the model. It suffices for it to reason about quantities.

3.8.5 A Transition Defines a New Scenario

QPC makes two important assumptions about transitions.

Assumption 3 *Variables are continuous in magnitude, continuously differentiable everywhere except at transitions, and differentiable from either side of a transition.*

This assumption implies that any quantity conditions that are true in a transition state, are also initially true in the database describing the situation following the transition.

A corollary of the above assumption is that the database describing the situation "after" a transition represents the same point in time as the transition state. This view is consistent with QSIM's view of transitions. Behaviors begin and end at a point in time, and model fragments are taken to hold over closed temporal intervals.

Assumption 4 *Entities (logically) persist across transitions.*

This implies that the set of entities cannot decrease over time. See Section 3.3 for a discussion of how the logical existence need not imply its physical existence, or even its participation in (in)active model fragment instances. Note that this assumption is still considerably weaker than that required by QPE or GIZMO, which identified all entities that could possibly exist in a scenario up front — that would correspond to the set of entities being constant, rather than non-decreasing.

Both of these assumptions will certainly be weakened as better theories for handling discontinuity and the coming and going of entities are developed.

3.8.6 Statement and Proof of QPC's Correctness

Definition 14 *The databases Δ_1 and Δ_2 <u>match</u> if their domain theory components are identical, they have identical sets of entities, and they entail identical QDEs.*

The behaviors $(s_{11} \ldots s_{1n})$ and $(s_{21} \ldots s_{2n})$ <u>match</u> if each pair of states s_{1i} and s_{2i} match.

The states s_1 and s_2 <u>match</u> if they have the same set of variables and each variable has the same value in s_1 and s_2.

Theorem 5 *Let the database $\Delta_0 = DT \cup E_0 \cup \Gamma_0$ be an initial (incomplete) description of a situation. DT is a domain theory, E_0 a set of entities, and Γ_0 a set of conditions.*

Let the tree $T = QPC(DT, E_0, \Gamma_0)$.

For any sequence of complete databases and behaviors

$$S = \Delta_1 \rightarrow B_1 \rightarrow \ldots \rightarrow \Delta_n \rightarrow B_n$$

that describe a possible behavior of the situation initially described by Δ_0*, satisfying the properties enumerated below, then there is a path* P *through the tree* T *that matches* S *elementwise.*

The sequence S *must satisfy the following conditions:*

1. *The domain theory component of each* Δ_i *is the same as that of* Δ_0*.*
2. $\Delta_0 \subseteq \Delta_1$*, and every stratification of* $E(\Delta_1)$ *and* $E(\Delta_0)$ *share the same initial stratum.*
3. $\forall i \ 1 \leq i < n: \ E(\Delta_i) \subseteq E(\Delta_{i+1})$ *and* $E(\Delta_i) \supseteq E_0(\Delta_{i+1})$
4. *The continuity assumption holds. That is, if* s *is the final state of* B_i*, then* $QC(s) \subseteq \Delta_{i+1}$*).*
5. B_i *is a behavior in QSIM(QDE(*Δ_i*),initial(*Δ_i*)).*

Proof by induction on the elements of S.

Base Case: Δ_1 is matched by the first complete database in T and B_1 is matched by one of its behaviors.

The statement of the theorem enables Theorem 3 to apply, concluding that there is a $\Delta_r \in$ refinement(Δ_0) such that $\Delta_r \subseteq \Delta_1$. Thus, the initial conditions of Δ_r are a subset of the initial conditions of Δ_1. Also from Theorem 3, active(Δ_r) = active(Δ_1), and inactive(Δ_r) = inactive(Δ_1). Thus QDE(Δ_r) = QDE(Δ_1). The QSIM soundness result ensures that if behavior B_1 is indeed consistent with the QDE and the stronger initial conditions of Δ_1, then B_1 is also one of the disjuncts in the simulation of the QDE under the (perhaps) weaker initial conditions of Δ_r. Therefore, the path $\Delta_r \rightarrow B_1$, where Δ_r matches Δ_1, is a prefix of the path P.

Inductive hypothesis: There is a path through T such that the complete databases and behaviors in the path are

$$\Delta_1^p \rightarrow B_1^p \rightarrow \ldots \Delta_i^p \rightarrow B_i^p$$

and they match the respective elements of S

$$\Delta_1 \rightarrow B_1 \rightarrow \ldots \Delta_i \rightarrow B_i.$$

Inductive step: There is a complete database Δ_{i+1}^p, one of whose behaviors is B_{i+1}^p, following B_i^p in T such that Δ_{i+1}^p matches Δ_{i+1} and B_{i+1}^p matches B_{i+1}.

Let s be the final (transition) state of B_i^p. Consider the (possibly incomplete) database Δ' that follows the transition from s. The database Δ' is formed in `QPC` from the union Δ' = $E(\Delta_i^p) \cup$ QC(s) $\cup$ DT.

We show that Theorem 3 applies to Δ' and Δ_{i+1}. This implies that there is a refinement Δ_{i+1}^p of Δ' that matches Δ_{i+1}.

There are two preconditions of Theorem 3 that are not trivially satisfied: that $\Delta' \subseteq \Delta_{i+1}$ and that the entities Δ' and Δ_{i+1} share the same first stratum. From the inductive hypothesis (and the definition of match), $E(\Delta_i^p) = E(\Delta_i)$. From condition 3 above, $E(\Delta_i) \subseteq E(\Delta_{i+1})$. Therefore, $E(\Delta_i^p) \subseteq E(\Delta_{i+1})$. Because B_i^p and B_i match, the final state s^p of B_i^p matches the final state s of B_i and $\mathrm{QC}(s^p) = \mathrm{QC}(s)$. By condition 4, $\mathrm{QC}(s) \subseteq \Delta_{i+1}$ and therefore $\mathrm{QC}(s^p) \subseteq \Delta_{i+1}$. By condition 1, $\mathrm{DT} \subseteq \Delta_{i+1}$. Therefore, because each set in the union that defines Δ', $E(\Delta_i^p) \cup \mathrm{QC}(s) \cup \mathrm{DT}$, is a subset of Δ_{i+1}, the union is also a subset. This establishes the first precondition. The second precondition is that Δ' and Δ_{i+1} share the same first stratum. Since Δ_i^p and Δ_i match, $E(\Delta_i^p) = E(\Delta_i)$. Thus, the initial sets of entities in Δ' and Δ_{i+1} are identical. Therefore, their initial strata are the same, Theorem 3 applies to Δ' and Δ_{i+1}, and the refinement Δ_{i+1}^p of Δ' matches Δ_{i+1}.

We now show that there is a behavior B_{i+1}^p that follows Δ_{i+1}^p in T and matches B_{i+1}. From condition 5 above, the behavior B_{i+1} is in the set of behaviors $\mathbf{B}_{i+1} = \mathrm{QSIM}(\mathrm{QDE}(\Delta_{i+1},\mathrm{initial}(\Delta_{i+1}))$. Now consider the set of behaviors $\mathbf{B}_{i+1}^p = \mathrm{QSIM}(\mathrm{QDE}(\Delta_{i+1}^p,\mathrm{initial}(\Delta_{i+1}^p))$. Since Δ_{i+1}^p matches Δ_{i+1}, the 1QDEs in these two calls to QSIM are identical. Theorem 3 proves that $\Delta_{i+1}^p \subseteq \Delta_{i+1}$, which implies that the initial conditions of Δ_{i+1}^p are a subset of the initial conditions of Δ_{i+1}. The QSIM soundness theorem implies that $\mathbf{B}_{i+1} \subseteq \mathbf{B}_{i+1}^p$. Every QSIM behavior generated by `QPC` is part of the tree T. Therefore, there is a behavior B_{i+1}^p that matches B_{i+1} in the tree T.□

3.9 Practice Versus Theory

This chapter describes a set of correct algorithms for an incremental hybrid compositional modeling engine. The primary goal of these algorithms is to provide a standard against which an implementation can be measured. An implementation that followed the preceding description directly would be correct, but might be rather inefficient. This could be due to

- conservative assumptions,
- incomplete inference, or
- inefficient inference.

We take up each of these areas in turn.

3.9.1 Stronger Heuristic Assumptions

In reasoning about physical situations, it is often convenient to assume that *exogenous variables are constant.* In QP theory, this is known as the *single mechanism assumption*, which is motivated by the ontological principle that all change is due to the activity of processes, and supported by the restriction that all mathematical relationships be causally directed. This restriction makes it straightforward to determine if a variable is uninfluenced and thus constant. Influences define a directed graph; any nodes with an incoming `I+` or `I-` arc are influenced and any node which has an edge incoming from an influenced node is influenced; upon completion, any uninfluenced node is exogenous. In QPC, the graph may contain undirected arcs which reduce the number exogenous variables.

There are a number of plausible ways to determine which variables are exogenous. In some settings, it may be natural to list the exogenous variables explicitly. Algorithms such as the ones developed by Iwasaki [Iwasaki and Simon, 1986a; Iwasaki and Simon, 1986b; Iwasaki, 1988] may be used to establish a causal ordering among variables in a model. It is also possible to use the QP theory restriction of associating a causal direction with each mathematical relation.

3.9.2 Stronger Inference

There are several sources of incomplete inference in QPC:

- QPC's ability to infer structural conditions. QPC may be unable to infer structural conditions that determine the activity of model fragment instances due to an incomplete description of the situation being modeled, or due to the incompleteness of its logical inference methods. Under many circumstances, however, it may be reasonable to use a closed world assumption for structural conditions.

  Heuristic 1 *If ϕ is a structural condition of a model fragment instance in a database Δ, then unless $\Delta \vdash_{mp} \phi$, assume $\neg\phi$.*

  Although this may result in unsound inference, it is used by most compositional modeling engines, including the current implementation of QPC. A more conservative method is to identify a subset of the structural conditions which the domain theory author believes the theory is strong enough to prove (when true).

- QPC's ability to infer quantity conditions. The axiomatization of quantities and constraints within a database is only partial. The reason for this is to enable efficient inference about quantities and constraints to occur within the simulator, which uses special purpose datastructures and algorithms tailored for this purpose. The result, however, is that QPC may not recognize databases as being inconsistent, and may provide weaker initial conditions than necessary. Although most knowledge about quantities has been delegated to QSIM, some needs to be retained in the database. The optimal decomposition is an area of empirical research. The current implementation of QPC reasons transitively over inequalities, but does not draw inferences across constraints in the database.

- QSIM's ability to eliminate inconsistent behaviors. QSIM, and any qualitative simulation engine, is itself incomplete. Considerable research has and is currently addressing the issue of strengthening the inferences that can be made by a qualitative simulator. QPC readily exploits any such advances.

3.9.3 More Efficient Inference

The current implementation employs a number of techniques to reduce the amount of work that needs to be done, including the following.

- Create attainable envisionments. Instead of producing a set of behaviors, or even a tree of states (as is the normal practice with QSIM), QPC uses QSIM to produce an attainable envisionment. An attainable envisionment is a graph of states rooted in the set of states consistent with the initial conditions. The graph compactly represents an infinite set of possible behaviors. Since only the transition states result in new databases being constructed, this provides precisely the level of discrimination that is required by QPC. Envisionment guided simulation [Clancy and Kuipers, 1992] can produce the behaviors corresponding to a path through the envisionment, if one is needed.

- Coalesce equivalent transition states. If a set of transition states (under a particular database) have the same magnitudes for all variables, then each will entail the same set of quantity conditions. Therefore, the transition databases for each member of the set will be equivalent. Thus, it suffices to build a single transition database from a representative member of the set.

- Simplify and reduce the children of a database during refinement. It is possible that children will be formed from identical sets of added ground conditions. By examining the sets of added ground conditions prior to forming the child databases, it is possible to eliminate duplicates and inconsistent sets.

3.10 Discontinuous Change

One of our basic assumptions is that all change in the underlying phenomena is continuous. To achieve viable models and usable predictions, however, it is sometimes necessary to introduce apparent discontinuity. Two important sources of discontinuous change in what are ultimately continuously valued parameters are:

1. shifts in perspective, and

2. shifts in time scale.

Shifts in perspective are modeling decisions that do not reflect any inherent change in the underlying phenomena. One important class of shifts in perspective are negligability assumptions. For example, a tap is slowly opened to allow flow into a tub. Initially, one might want to consider the evaporation from the tub. Once the flow is established, however, the evaporation will be negligible. The decision to remove the evaporation view from the model does not indicate that evaporation no longer occurs, but rather that it is no longer desirable to model it.

Time scale abstraction can be viewed as an application of negligability reasoning to the temporal extent of processes [Kuipers, 1987], Even if all change is continuous, it may be valuable to treat certain continuous changes as if the were discontinuous. For example, the elastic collision of a falling ball with the floor occurs over a much shorter interval of time than its fall or subsequent rise. If the basic scale of description corresponds to the rising or falling, it seems reasonable to consider the collision as instantaneous. I.e. to pretend that there is a discontinuous change in the velocity of the ball at the point of impact. Such an apparent discontinuity, however, may be grounded in a model of the continuous behavior. Time scale abstraction may dramatically reduce the complexity of a model because it eliminates interactions between processes.

3.10.1 Local Decisions and Continuity

It is worth noting that a valid QPC (or QP theory) domain theory may be constructed in such a way as to introduce discontinuous changes, which the QPC model builder will eliminate as inconsistent. This places the modeler in an awkward position. Model fragments should be modular, but the continuity restriction effectively requires the modeler to consider possible sequences of models.

As an example, suppose that a modeler wishes to define a set of model fragments to exhaustively cover the range of some phenomenon. The domain theory consists of the following model fragments.

```
(defModelFragment f1 :participants (p)
   :operating-conditions ((< (x p) zero))
   :consequences (I+ (x p) (y p)))
(defModelFragment f2 :participants (p)
   :operating-conditions ((>= (x p) zero))
   :consequences (I- (x p) (z p)))
```

This would appear to be a reasonable way to describe a phenomenon that influences `(x p)`. When `(x p)` is below zero, then `(y p)` is the controlling influence, otherwise `(z p)` is. A problem arises, however, if `(y p)` is not equal to `(z p)` when `(x p)` reaches zero. In the case when `(y p)` $>$ zero and `(z p)` $<$ zero, the derivative of `(x p)` might change sign at the transition. Because QPC will infer that transitions that undergo such discontinuous changes are inconsistent, this pair of model fragments is adding an implicit constraint which the modeler may easily overlook. Similar problems arise with the other composable equations, and with correspondences. The difficulty in all of these cases is that the local specification of one model fragment may have unforeseen interactions with others.

3.11 Conclusion

This chapter has defined and shown the correctness of a hybrid compositional modeling engine, QPC, that incrementally constructs predictions of the temporal evolution of a physical situation.

Compared to non-incremental algorithms such as total envisionment, our approach enables large systems to be modeled when there is substantial information about their initial conditions. An incomplete description of an initial situation is refined into a set of complete descriptions. The incremental algorithm supports a hybrid architecture in which the model building and simulation tasks are decomposed and executed by special purpose tools. We prove that the simulation engine need know nothing about the logical structure of the model. Thus, it suffices for the simulator to reason only about quantities.

The hybrid architecture allows for efficient simulation, but more importantly, it finesses the difficult problem of axiomatizing (qualitative) differential calculus. This axiomatization is implicit in the soundness of the QSIM simulation engine. Thus, advances in (qualitative) simulation techniques may be exploited immediately. Furthermore, such advances need not be in the declarative application of new mathematical techniques, but in efficient algorithms for accomplishing them. Incorporating such algorithms into a declarative logical representation may border on the impossible.

The formal specification of the QPC algorithms makes both the underlying assumptions and overt guarantees clear. This is essential if a modeling engine is to be used for engineering reasoning. Furthermore, the formal description can be used as a standard against which specific implementations can be compared.

Chapter 4

Practice

This chapter discusses the design and construction of the implemented QPC system. The differences between theory and practice are emphasized and solutions to the difficult technical problems of implementing QPC are presented.

4.1 Design Principles

QPC has been structured so as to allow advances in modeling and in simulation to be readily incorporated. This has lead to a hybrid system that uses several well defined subsystems to achieve this goal. Furthermore, QPC is designed to support and enable research in both domain modeling and qualitative reasoning. Researchers in the first group use QPC to build detailed models of phenomena in their domain of expertise, such as chemical engineering. The second group includes researchers working on modeling techniques (relevancy reasoning), those extending QPC (discrete valued variables, discontinuous change, function templates), and those extending qualitative reasoning methods (algebraic reasoning).

In order to accommodate these divergent interests, QPC has been implemented with a number of hooks that sophisticated users can employ to extend, strengthen, or modify its behavior. It is expected that construction of domain theories will result in extensions and modifications to the existing language.

Based on experience gleaned from the QSIM project, a flexible, extensible, and sound approach to structuring a research tool is the use of generators and filters. A generator produces a collection of objects guaranteed to include all objects of interest. The filters remove objects guaranteed not to be of interest. The sets of filters are extensible, so that new expertise may be incorporated. Clearly, this strategy may be inefficient. It is well known that a filter may be pushed up into a generator to prune possibilities long before they are completely generated. It is also evident, however, that this typically complicates the generator. In QPC, as in QSIM, we have tried to err on the side of clarity. For smallish collections (less than 2^{10} or so) the cost may well not be too high.

The initial implementation of QPC did not follow this generate and filter approach. It used depth first search with backtracking to generate models. Unfortunately, given

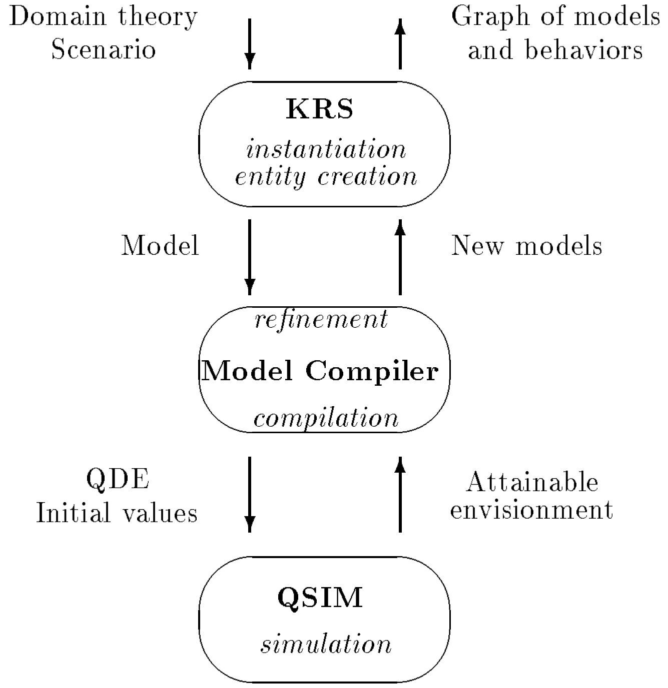

Figure 4.1: High level architecture of QPC.

the black-box nature of the components of a hybrid system, it was extremely difficult to determine appropriate backtracking points. As the code became more baroque and less correct, it became clear that another approach was needed.

In QPC the notion of filters has been extended to include critics that are licensed to perform certain modifications of data structures. A critic might, for instance, simplify the form of an equation or remove redundant relations from a set. QPC's critics provide a means of incrementally strengthening the inferences that are made. Critics are licensed to perform truth preserving transformations.

To summarize, we have taken the approach of generating objects, applying filters to remove uninteresting or inconsistent ones, and the applying critics to improve them. A filter should be conservative, and a critic should preserve correctness.

4.2 Architecture

QPC can be decomposed into three quite distinct subsystems: The Algernon knowledge representation system, the model compiler, and QSIM. Figure 4.1 outlines the relations between them. The Algernon *knowledge representation system* draws inferences from the domain theory, the facts asserted about the scenario, and any facts that are asserted by the model builder to define a new model. The primary operations are to infer the existence of entities and to instantiate model fragments. The *model compiler* queries the knowledge base to determine which instances are active and inactive, refines incomplete models, compiles complete models into a QDE and a set of initial values. QSIM solves this initial value problem and generates an attainable envisionment. The model compiler examines the envisionment and identifies any behaviors that cross the boundary of the current model's region of applicability. The final states of each of these behaviors are decompiled to define a new model.

4.3 The Algernon Representation System

QPC currently uses the Algernon knowledge representation system to implement its knowledge base [Crawford, 1991b; Crawford and Kuipers, 1991; Crawford, 1991a]. Although nothing in QPC requires the use of Algernon, many detailed implementation decisions were affected by this choice.

Algernon is an implementation of *access limited logic*. Access limited logic formalizes the intuition behind frame systems and semantic nets that some inferences are simple and fast to make because they follow a *path* between related objects. The logic reflects the representational decisions, which determine which inferences are easy and which are hard. The key formal properties of access limited logics are:

- Socratic completeness. Instead of being complete, an access limited logic is capable of proving any theorem given an appropriate sequence of queries. This is analogous to stating that any theorem can be proven if an appropriate sequence of lemmas can be proven first. This approach can be contrasted with that of, for instance, classifier systems, which restrict the logic and the set of possible queries in order to provide completeness and tractability guarantees[Levesque and Brachman, 1987].

- Tractability. Access limited logic guarantees that any query will be answered (perhaps with *maybe*) in time polynomial in the size of the knowledge base.

In order to achieve these properties, access limited logic places a restriction on the syntactic form of queries and assertions. They must conform to the *access path restriction*. The restriction is that both the relation and its first argument must be a ground term before it can be queried or asserted. Thus, `p(a,?x)` satisfies the restriction, but `p(?x,a)` does not. An ordered conjunction of queries may also be a valid path: `p1(a,?x), p2(?x,?y), p3(?y,?z,?w)`.[1]

Algernon represents relations using *frames*. Binary relations are easily represented in a frame based system. The relation `p(a,b)` is represented by storing `b` as a value in the `p` slot of the frame for `a`. We can denote this:

```
a:
   p: b
```

[1] By convention, any symbol whose first character is a '?' represents a variable and other symbols represent constants.

This can be generalized to n-ary relations by storing a tuple in the slot. E.g. we can represent `p(a,b,c,d)` as

```
a:
   p: (b, c, d)
```

Using access paths has several advantages:

- The access path restriction means that an access limited logic can be easily implemented using frames because the first argument (the frame) is always bound when a query or assertion is made.
- It institutionalizes the PROLOG programmers habit of using "first argument indexing".
- It aids in establishing complexity results.

There are, however, some disadvantages. Namely, the path restriction forces the programmer to carefully consider the *flow of control* whenever a logical relation is represented. With binary relations, this is a mere nuisance. It means that an inverse relation must often be defined. With n-ary relations, however, it can be something of a struggle. The programmer must guess which access paths will be used and ensure that they are available. E.g. suppose that we want to encode the logical relation `p(x,y,z,w)`. If we have an `x`, then we can use `p` directly. If we have a `y` and want to find the `x, z, w` that satisfy `p`, then we need to define a new relation `p1(y,x,z,w)`, and so on. I.e. we need up to n relations to define a single logical relation. Although such indexing considerations are important when *optimizing* a representation, it is not clear that they are appropriate when specifying one.

Algernon caches every inference that is made. This is necessary to retain certain theoretical complexity results of access limited logics. In practice, however, it can be quite inefficient. For example, it makes it very expensive to do algebraic simplifications which might have large intermediate terms that are completely uninteresting after the final simplification has been found.

Caching also makes it difficult to construct an efficient representation. Consider again the predicate `p(x,y,z,w)`. In non-caching system such as PROLOG, it is possible to use `p` as a uniform interface, while dispatching to more efficient indexes when appropriate. For example:

```
p(X,Y,Z,W) :- nonvar(X), !, p0(X,Y,Z,W).
p(X,Y,Z,W) :- nonvar(Y), !, p1(Y,X,Z,W).
...
```

The more efficient indices may be added transparently as performance analysis warrants. This is not possible in Algernon because any tuples inferred for `p` will be cached. Subsequent queries will search through all cached values before attempting to use the more efficient index.

4.4 Representing Disjunction

As described in Section 3.7, the output of QPC is a tree of models and states. Each path from the root to a leaf of the tree represents one possible behavior. A natural way to represent this in a knowledge base is with nested disjunction. Each branch point in the tree corresponds to a disjunction in the knowledge base.

The sort of disjunctive inference required by QPC would be readily supported by a multiple worlds database. In such a system, the truth of a proposition is relative to a world. Furthermore, worlds may be nested, sibling worlds are typically disjoint, and worlds may become inconsistent without making the entire database inconsistent.

Algernon provides an assumption mechanism that allows the multiple world database to be mimicked. QPC used this mechanism for quite a while. The assertions that distinguish one model from another are introduced under the explicit assumption that the model is consistent. If a contradiction is detected, then the assumption can be dropped and the model is inferred to be inconsistent without compromising the KB. Unfortunately, as the size of the problems that QPC was being applied to grew, so did the cost of this approach. This is partially due to idiosyncrasies of Algernon's implementation, however, such a mechanism is considerably more general and expressive than is needed by QPC.

QPC currently marks inconsistent models as `inconsistent`, and proceeds with inference. This is admittedly unsafe and unsound in general. Algernon, however, does not make the dreaded "contradiction implies all" inference. Furthermore, once a model is recognized as inconsistent, QPC will not employ it (or anything true under it) in any inference. Thus, this approach is safe for QPC.

4.5 Algorithm

The implementation of QPC follows the formal development of Chapter 3 rather closely. There are, however, several points that require description, definition, and comment.

The input to `qpc` is a domain theory and a scenario specification defined using the forms specified in Chapter 3. The output is represented by assertions in the knowledge base as follows.

1. *refinement* `(refinement m1 m2)`
   Model `m1` is incomplete. Model `m2` is a more complete description that results from adding some ground conditions not present in `m1`. The model `m2` is a subscope of `m1`, so it inherits any facts that were true in `m1`.

2. *initial state* `(initial-state m1 s1)`
   The model `m1` is a complete description of a situation. The qualitative state `s1` is an initial state for `m1`.

3. *state successor* QSIM state successor
   A state is a QSIM structure. The successors of a state are accessed via the QSIM `state-successors` function and are not currently represented in the knowledge base.

4. *transition* `(transition m1 s m2)`
   State `s` is a transition state in the simulation generated for the initial state of `m1`. The model `m2` describes the situation after the transition from `s`.

Figure 4.2 defines the agenda driven top level loop for `qpc`. The agenda is seeded with the initial model for a scenario. The loop is executed until the agenda is empty.

```
define qpc
  until agenda is empty                                  1
    remove m from agenda                                 2
    initialize m                                         3
    if incomplete(m)                                     4
      then
        for c in children(m) add c to agenda             5
      else
        if ordered(qspaces(m))                           6
          then
            resolve(m)                                   7
            for s in transitions(qsim(initial(m),qde(m)))  8
              add model(s) to agenda                     9
          else for o in orderings(m)                     10
                add model(o) to agenda.                  11
```

Figure 4.2: The agenda driven top level of QPC.

In step 3, `m` is initialized. Figure 4.3 defines `initialize`. First, any conditions cached on `m` are asserted into the knowledge base. If `m` is a refinement, then these are the the operating region conditions that distinguish `m` from its parent. If `m` is a transition, then these are operating region conditions that `m` must be entering (see step 9). If the model `m` is a refinement of `p`, then the relation `(refinement p m)` is asserted into the knowledge base. This implies that `m` is a subscope of `p`, so any fact true in `p` is also true in `m`. If the model is a the result of a transition from state `s` and previous model `p`, then the relation `(transition p s m)` is asserted into the knowledge base and the quantity conditions from `s` are asserted into the knowledge base scoped to the initial time of `m`. If `m` is a refinement or a transition model, there may be region conditions that it must satisfy which are also asserted (scoped to `m`). Initializing the model enables the knowledge base to infer the existence of new entities and to instantiate model fragments. Entities are inherited across both refinements and transitions.

In step 4, the model `m` is checked to determine if it is `incomplete`. A model is incomplete if there are any instances scoped to the model that are ambiguous (have not been proven active or inactive). The set of ambiguous instances is readily determined by querying the

```
define initialize(m)
  assert conditions(m)
  unless m is the initial model
    p = parent(m)
    if refinement(m)
     then assert (refinement p m)
     else
       — m follows a transition
       s = transition-state(m)
       assert (transition p s m)
       assert qspaces(s)
       assert QC(s)
```

Figure 4.3: Initializing a model asserts relations into the knowledge base. `QC` returns the quantity conditions for `s`; `qspaces` returns inequalities implied by the quantity spaces of `s`.

knowledge base.

In step 5, the `children` of an incomplete model are computed and added to the agenda. The children are computed as described in Chapter 3 except for two differences. First, the sets of ground conditions that are added to `m` to form each new model are checked for redundancy and consistency. Any duplicates or obviously inconsistent sets are deleted. Second, the sets of ground conditions are not immediately asserted into the knowledge base. Instead they are cached on the structure representing the model on the agenda. The conditions are asserted into the knowledge base when a child model is removed from the agenda and initialized in step 3.

In step 6, the quantity spaces are checked to determine if they are totally `ordered`. QSIM requires that the quantity space of each variable be totally ordered. The representation in the knowledge base, however, allows for partial orders. QPC handles this mismatch by creating a new model for each consistent total ordering in step 10.

In step 7, a complete model with totally ordered quantity spaces is `resolved`. Resolving a model consists of three steps. First, the influences are collected and resolved. The resulting constraints are asserted into the knowledge base in the scope of `m`. Second, the set of exogenous variables is determined. Third, the exogenous variables are asserted to be constant in the scope of `m`. QPC currently determines the set of exogenous variables by using a graph based algorithm (implemented in the knowledge base) similar to that used by Forbus.

In step 8, the QDE and initial conditions for the model are defined and passed to QSIM. QSIM generates an attainable envisionment and the transition states are identified. Computing the QDE and initial values are discussed in Section 3.8.3 and Section 4.7. The transition states are readily identified from the QSIM representation. QPC makes a slight gain in efficiency by forming equivalence classes out of the transition states. Any transition states that will result in the same quantity conditions being asserted into a transition model may be placed into an equivalence class. A new model need only be built in step 9 for one representative of each class.

In step 9, a structure to represent the transition from the current model `m` through a state `s` is built and added to the agenda. Information about the new model is not asserted into the knowledge base until it is removed from the agenda and initialized in step 3.

In steps 10 and 11, a model without totally ordered quantity spaces is handled. First, all total orders consistent with the knowledge base and the partially ordered quantity spaces are computed. Second, a new model is added to the agenda. These new models are handled as if they were children of `m`. Instead of caching operating region assertions on the new models, the quantity conditions that establish a total ordered quantity spaces are cached.

4.5.1 Structural Conditions and Transitions

There are three means for inferring that structural conditions hold after a transition.

1. Structural conditions that are asserted by model fragment instances are rederived if the instances are active in the model after the transition. Any structural conditions that are implied by them are also rederived.

2. Structural conditions that are true throughout the *scenario* are true in every model of the scenario, including transition models. These are *static* conditions.

3. The domain theory author may write axioms that imply the truth of structural conditions across transitions.

The theoretical exposition of Chapter 3 only dealt with the first type. This is primarily because the formalization in terms of *databases* removes axioms relating the contents of databases from the domain theory. Such axioms only exist in the meta-theory. Because the implementation reifies the databases, it is possible to express axioms relating them (and their contents) in the domain theory. The second type is extremely useful for problems in which much of the configuration is fixed. The third type retains the flexibility of the theoretical exposition, while providing significant constraints on the sets of models that need to be created. It allows the domain theory author to specify a kind of *frame axiom* [McCarthy and Hayes, 1969]. Figure 4.4 shows a sample structural condition rule to infer that if something is the contents of a container in one model, then it is still its contents after a transition.

4.6 The Knowledge Base

4.6.1 Overview of the QPC KB

The QPC KB can be separated into several distinct layers, as shown in Figure 4.5.

```
(defrule
  (entity ?p ?c) (isa ?c containers)
  (contents ?p ?c ?l)
  (transition ?p ?c ?m)
  (entity ?m ?c)
  (entity ?m ?l)
  ->
  (contents ?m ?c ?l))
```

Figure 4.4: Simple frame axiom for `contents`.

1. The lowest level contains basic knowledge about sets, objects, and relations in addition to scopes and inheritance between them.
2. Inequality reasoning. QR relies heavily on reasoning about the relations between quantities. In QPC most of this reasoning is done by QSIM. The KB, however, needs to do some inequality reasoning in order to determine whether or not model fragments are initially active.
3. Qualitative reasoning. Although QPC does not do any qualitative reasoning with the KB, it does need a representation for the objects that are employed in QSIM. These include quantities, quantity spaces, constraints, and so on.
4. QPC. The ontological commitments of QPC exist at the next level. These include scenarios, models, model fragments, entities, and the composable equations such as influences.

The domain theory and particular scenarios are build on top of the QPC level of the knowledge base.

The remainder of this section describes many of the relations and objects to be found in the QPC KB. We follow the layered ontology depicted in Figure 4.5 beginning with the most primitive elements.

4.6.2 Basic Knowledge

4.6.2.1 Skolem Slots

This is a set of slots of the form

$$\rho(x_1, \ldots, x_n, y)$$

for which the axiom

$$\forall x_1 \ldots, x_n, \ \exists y \ \rho(x_1, \ldots, x_n, y)$$

4. QPC

Instantiation	Activation
Scenarios	Models
Model Fragments	Influences

3. Basic QR

Constraints	Quantities
Quantity spaces	Time

2. Inequalities

1. Basic Knowledge

Sets	Inheritance
Transitivity	Scopes

Figure 4.5: Overview of the QPC knowledge base.

holds. I.e. the KB is licensed to introduce skolem individuals for the last argument. The `skolem-slots` rule (for binary relations) is:

```
(defrule skolem-slots
  (?r ?x ?y) <- (exists ?y (?r ?x ?y)))
```

which will look for a binding for `?y` and create one if none exists.[2]

4.6.2.2 Reification

Algernon does not have a uniform approach to reifying relations. As a consequence, the reification of relations in QPC is somewhat *ad hoc*. The exact form and means of reification varies slightly between relations depending on their use within the KB. A mild generalization of the scheme used is:

```
  (defrule reified-relations
       (?rel ?a1 ?a2 ?a3) -> (exists ?r
                                (member reifications ?r)
                                (relation ?r ?rel)
                                (arg ?r 1 ?a1)
                                (arg ?r 2 ?a2)
                                (arg ?r 3 ?a3)))
```

The index (second argument to `arg`) is necessary in a frame based system such as Algernon that supports neither sequences of values nor destructuring unification (like PROLOG).

[2] Skolem slots are not skolem functions, as Algernon does not support multi-arity functions. The relation `?r` is bound to each member of the set of `skolem-slots`.

4.6.2.3 Scopes

As described in Section 3.2, scopes are used to implement a kind of multiple world database with inheritance.

There is a set of `scopes`, whose subsets include `models` and `model-fragments`.

The set of `inherited-relations` are relations for which the following axiom holds:

```
(defrule inherited-relations
   (?r ?scope . ?args)
   (subscope ?scope ?subscope)
   ->
   (?r ?subscope . ?args))
```

In addition to the `inherited-relations`, which have scopes as the first argument, there are similar sets with the scope as the last argument.

The `subscope` relation is transitive, and has an inverse `super-scope`.

4.6.3 Inequalities

Reasoning about relations between quantities is the heart of qualitative reasoning. As we have explained, most of QPC's inequality reasoning is done by QSIM. Within the KB, however, certain inequalities are essential for determining whether or not model fragment instances are active. Consequently, some inequality reasoning must be done within the KB.

Inequalities are represented by the relations:

```
(q< ?x ?y ?s)
(q= ?x ?y ?s)
(q> ?x ?y ?s)
```

and their negations:

```
(not (q< ?x ?y ?s))
(not (q= ?x ?y ?s))
(not (q> ?x ?y ?s))
```

where `?x` and `?y` are quantities, and `?s` is a scope. [3]

There are two types of inequality reasoning currently done within the KB:

1. Inheritance. Inequalities are inherited from a scope to its subscopes (E.g. from a scenario to every model of that scenario.).

2. Transitivity. `Tc-scopes` are transitively closed. Because computing the transitive closure is a fairly expensive operation, it is limited to scopes in which it is absolutely necessary. In particular, the model at its initial time is a `tc-scope`. This is because

[3] Access considerations originally led us to place the scope as the last argument. Early on, experience indicated that a scope would have many inequalities, whereas an individual variable would be involved in relatively few. Thus, the variable was chosen as the better index. As scenarios became more complex and involved more scopes (models and instances), this becomes less and less the case. In fact, it is the combined index of variable and scope that is really desired.

model fragment instantiation relies on evaluating operating conditions at the initial time.

Transitivity reasoning is done primarily by forward reasoning. One concession to efficiency is the special handling of `zero`. The closure is computed separately above and below `zero`. There is little additional value in inferring that every negative number is below every positive one!

4.6.4 Basic QR objects

Time Time is represented by `scopes`. In order to represent the statement $p(x)$ is true at time t, we introduce a scope s_t to represent the state of affairs at time t, and assert $p(s_t, x)$. The only time currently used within QPC is the *initial time* point of a model. Each complete model is linked to its initial QSIM state and QSIM states contain pointers to their successors. It is straightforward to duplicate some or all of the temporal information represented by the QSIM states in the knowledge base, but inferences requiring this have not yet been a focus of work involving QPC.

Quantities Quantities are either `variables` or `magnitudes`. The relationships among `quantities`, `variables`, and `magnitudes` are illustrated by the following KB excerpt:

```
Mass:
  Isa: skolem-slots quantity-types slots

A:
  Isa:  physobs
  Mass: a.mass

A.mass:
  Isa: variables
  Q=: (zero m-1)         (l* m-2.subabs)
  Q>: (zero m-2.subabs)

L*:
  Isa: magnitudes
  q>: (zero true)
```

`Mass` is a quantity type. I.e. a relation between an object and a variable. `A` is a physical object whose `mass` is represented by the frame `a.mass`. `A.mass` is a variable (member of the set of `variables`). The value of `a.mass` is `zero` in the scope `m-1`, and `l*` in the scope `m-2.subabs`.[4] I.e. it is equal to `zero` throughout model `m-1` and is equal to `l*` for the time represented by `m-2.subabs`. The magnitude `l*` is greater than `zero` in the scope `true`. `True` is the top of the lattice of scopes, thus `l*` is always greater than `zero`. Transitivity reasoning has inferred that `a.mass` is greater than `zero` in `m-2.subabs`.

[4] A subabs represents a model at a particular time.

Qdirs The `qdir` of a variable is its qualitative direction of change — the sign of its first derivative. QPC allows information to be inferred about the qdir of a base variable without creating a second variable to explicitly represent the base variable's derivative. In fact, there is an important distinction between the two representations. Every variable in a model is continuous in magnitude across a transition. If both the derivative and the base variable are present in a model, then the base variable will be continuous in both magnitude and it its first derivative. This is because the magnitude of its first derivative will be the same across the transition. If the derivative had not been explicit, then the base variable's first derivative could have changed discontinuously across the transition.

Quantity spaces In QPC a quantity space is represented by a *set* of magnitudes. The order of elements in the quantity space is implicit in the relations between the magnitudes. This allows for quantity spaces to be partially ordered within the KB, and also allows for magnitudes to be members of multiple quantity spaces. This is considerably more general than QSIM's totally ordered, distinct quantity spaces. In order to map a complete model to a QDE,a total ordering must be established for the quantity space of each variable in the model. This requires an additional step in the algorithm to establish distinct total orderings.

Constraints Constraints are represented in two ways within QPC. First, they may be simple assertions such as

```
(m+ m-1 (mass a) (level a)).
```

Second, they may be reified:

```
M-1:
  Constraint: c-1
C-1:
  Constraint-rel: m+
  Isa:        constraints
  Var:        (1 a.mass) (2 a.level)
```

Because many of the constraints (such as the M constraint) have several facts that must be asserted about them, the reified form is preferred. Most of the QSIM constraints are supported including `M`, `Sum`, `Sum-Zero`, `Minus`, `Mult`, `Add`, `M+`, `D/Dt`, `Decreasing`, `Increasing`, and `Constant`.

Correspondences As described in Section 3.4.3 (page 45), a constraint may be strengthened by a *correspondence*. A correspondence is reified in the KB:

```
Correspondence8:
  Isa:        correspondences
  Determining:(b-contents.pressure zero)
  Determined: (db.flowrate zero)
```

As the order of the determining variables is unimportant, there is no numeric index for the determining variables.

4.6.5 QPC Objects

4.6.5.1 Models

In Chapter 3, we developed a formalization of QPC in terms of a *databases* which describe a situation within a single region of operation (during which the situation is modeled by a single QDE). A database is a set of logical axioms, and we proved a number of theorems relating distinct databases. In order to implement QPC, the databases have been reified into `models`. The relations within a database are reified by adding the model as an explicit extra argument (usually the first). The relations between databases are represented as relations between the models.

The tree that was described as the output of QPC in Section 3.7 is directly realized by the `refinement` and `transition` relations in the knowledge base.

```
(defrelation refinement (models models))
(defrule (refinement ?m1 ?m2)
   ->
  (subscope ?m1 ?m2)
  (subscope (model-at-initial-time ?m1)
            (model-at-initial-time ?m2)))
```

The relation `(refinement m1 m2)` holds when a refinement of `m1` is `m2`. This implies that `m2` is a `subscope` of `m1` and that the scopes representing `m2` and `m1` at their initial times are also in the `subscope` relation.

```
(defrelation transition (models states models))
```

The relation `(transition m1 s1 m2)` is true if the model `m1` has a transition at state `s1` to model `m2`.

4.6.5.2 Scenarios

A `scenario` holds together a complete statement of a QPC problem and its solution. Each scenario has a unique `initial-model`, which describes the initial configuration of the situation of interest. The `scenario` is a `superscope` to all of the models that are constructed during the course of solving the problem. Thus, it is possible to establish relations that are true in the scenario, and are consequently true in every model of that scenario. Multiple scenarios may be present in the KB at one time.

4.6.5.3 Entities

An entity is an object whose properties include variables and whose behavior may be predicted by QPC. Entities typically correspond to observable objects in the world, although they may also represent abstractions and collections of such objects. Entities are in the `entity` relation to models. We often speak of an object being "an entity in a model". Entities are introduced either by a scenario definition, or a `defentity` form.

4.6.5.4 Abstract and Concrete Objects

From an early stage, QPC was designed to support meta-level reasoning about the domain theory. To accomplish this, QPC represents two distinct type of objects that one might call *abstract* and *concrete*. The model fragment and entity definitions of the domain theory introduce abstract object as well as axioms of the types described in Chapter 3. E.g. a fluid flow model fragment introduces an abstract container and fluid.

The presence of abstract objects simplifies reasoning about the domain theory itself. It is possible to represent statements such as:

Heat flow is a process.
Heat flow affects thermal properties.
Enthalpy is a thermal property.

It is also simple to answer queries such as:

What model fragments constrain the flow from a drain?
What conditions would need to be established for an instance of fluid flow to exist?

Abstract objects may also be used to reduce the number of axioms required to represent the domain theory. Instead of having a distinct set of axioms to instantiate each model fragment, a single set of axioms suffices. The instantiation axioms match the abstract objects against concrete objects in a particular scenario. In fact, QPC currently uses this scheme for instantiation. Although the number of axioms is reduced, however, the computational costs are considerably than if a distinction set of axioms were used for each definition. There are two reasons for this. First, the set of possible bindings for the instantiation axioms is much larger. Second, the chain of inferences that must be made to instantiate a model fragment are considerably longer. The next release of QPC will use a distinct set of axioms for each model fragment.

4.6.5.5 Adding Landmarks to Quantity Spaces

One of the arts of building a qualitative simulation model is identifying the minimal set of landmarks that still covers the distinctions of interest. The granularity of the quantity spaces determines both the precision and size of the qualitative description. If the set of landmarks is insufficient, then the simulation will not provide answers to the user's real questions. If there are unnecessary landmarks, however, there will be unnecessary distinctions and detail in the behavioral description.

QPC infers that a landmark L should be in the quantity space for a variable V:

1. if a relation between L and V appears in the operating conditions of any instantiated fragment (active or inactive). These define the boundary conditions of the simulation model.

2. if a relation between V and another variable W appears in the operating conditions of an active instance, and W = L.

3. if there is a correspondence including V and L.

4. if V is in a constraint for which `zero` has a special meaning, then `zero` is added to the quantity space.

5. if L is explicitly inferred to be an `important-landmark` for V through additional axioms provided by the domain theory.

6. if there is an inequality between L and V in the model. This is really too general. For example, if $V > 0$, and $L < 0$, then $L < V$. I.e. Every negative landmark should be in V's quantity space. QPC does not make this stupid inference. Avoiding it (and some similar ones), however, requires complicating the axioms to exclude the special cases.

7. If V = L at any time. This is primarily used to include inferred initial values.

Depending on the type of user, these heuristics may or may not be adequate. Some scenario specifications tend to be sparse descriptions — everything mentioned really counts. Textbook problems and puzzles fit this pattern, but so might problems posed by engineers who have already analyzed the situation. Problems extracted from a large KB, however, are likely to include extraneous information, and consequently many landmarks may really be unnecessary for the actual generation of behavior.

4.7 Interface to QSIM

There are three technical problems with the QSIM representation and algorithm that had to be solved (or ameliorated) for QPC to become an effective system. We briefly outline the issues here, and describe QPC's solutions in the following subsections.

1. QSIM does not directly represent *inequality relations* other than the strict ordering of landmarks within a single quantity space. This makes it difficult for QPC to:

   (a) Assert initial conditions. QSIM only accepts landmarks or open intervals, whose endpoints are landmarks, for the initial values of variables. QPC may infer half-open or closed intervals, whose endpoints are either landmarks or variables.

   (b) Assert inequality constraints that hold throughout a model. The QDE representation directly supports only non-strict inequalities between a variable and its landmarks (via upper and lower bounds on the quantity space). Non-strict inequalities between variables and landmarks or any relation between variables other than equality is not directly supported.

   (c) Assert boundary conditions. The QSIM transition condition syntax only allows boolean combinations of variables having specific qualitative values. QPC's boundary conditions include arbitrary relations between quantities.

2. Qualitative reasoning may lose essential *algebraic information.* This may occur for two reasons:

   (a) Landmarks (other than zero) in distinct quantity spaces are distinct. QPC, however, may insert a landmark into several different quantity spaces. E.g. if `l*` is a landmark in the quantity space of `x` and `z`, QSIM will not infer the corresponding values `(l* 0 l*)` for the constraint `(add x y z)`.

   (b) Algebraic constraints that are implied by a QDE will not prune any possibilities. E.g. given `(add x y z)` and `(add x y w)`, if `x` is increasing and `y` is decreasing, then under QSIM's inference mechanism, it is consistent for `w` to increase while `z` decreases!

3. Qualitative simulation may *branch intractably.* As we outlined in Chapter 2, there may be an infinite number of qualitative behaviors that consistent with a QDE and a set of initial values. In the past, models were typically hand crafted to allow tractable simulation. This is not possible for an automatic model builder such as QPC.

4.7.1 Inequality Relations

Reasoning about relations between variables and magnitudes or among magnitudes is relatively straightforward. The key step in reasoning about the relations between variables is to reduce it to the simpler case of the relations between variables and magnitudes.

QPC encodes inequality relations between variables by introducing an auxiliary variable to represent their difference. The original relation is thus reduced to the relation between the difference variable and zero. To avoid unnecessary qualitative distinctions, the quantity space of the difference variables contains only `minf`, `zero`, and `minf` and no new landmarks are introduced during simulation. Furthermore, since only the magnitude of the difference is important, its qdir is ignored. Ignoring qdirs to reduce the size of a simulation is described in [Kuipers and Chiu, 1987].

A single difference variable can be used to represent any inequality relationship between a pair of variables and to encode constraints as well as initial or boundary conditions. QPC accomplishes this by defining a unique ordering among variables and converting all of the inequalities into a canonical form. For example, the inequalities `(> x y)` and `(> y x)` would be converted into the canonical form `(> x y)` and `(< x y)` respectively. The difference variable `diff-x-y` is generated and the inequalities are reduced to `(> diff-x-y 0)` and `(< diff-x-y 0)` respectively.

Inequality	Encoding	Filtered values
$x > 0$	`x = ((0 inf) nil)`	none
$x \geq 0$	`x = ((minf inf) nil)`	`minf`, `(minf 0)`
$x > y$	`dif-x-y = ((0 inf) nil)`	none
$x \geq 0$	`dif-x-y = ((minf inf) nil)`	`minf`, `(minf 0)`

Table 4.1: Summary of encoding of initial conditions. The variable `x` has the default quantity space `(minf 0 inf)`.

4.7.1.1 Initial Conditions

Strict initial inequalities between a variable and a magnitude are trivial to specify. This is the case that QSIM expects. E.g. `x` > `l*` is equivalent to saying `x` = `((l* inf) nil)` — the magnitude of `x` is in the open interval `(l* inf)` and its direction of change is unspecified (`nil` is simply a place holder). If it is also the case that `x` < `m*`, then the initial value for `x` would be `((l* m*) nil)`.

There are three likely approaches to expressing non-strict initial conditions:

1. Extend QSIM to handle disjunctive initial conditions. Although attractive in the long run, it would have far reaching consequences for the QSIM implementation, and I chose not to do this.

2. Compute the disjunctions explicitly and provide QSIM with several initial value problems. This is not very appealing, as the number of such problems grow exponentially. Furthermore, many of the possibilities may be ruled out by other initial values. E.g. given `x` ≥ 0, `z` > 0, and `(m+0 x z)`, QSIM will rule out the `x` $= 0$ case very quickly.

3. Weaken the inequality to the tightest strict inequality that it implies, use this to provide the initial values, and provide a filter to eliminate any initial states that do not satisfy the original inequality. This is the method we chose. It was easy to implement, and has not yet proven to be a bottleneck. It can also be promoted from a QSIM state filter that is applied to completed states, to a value filter that is applied to the values hypothesized for the variables in question before the states are formed [Farquhar *et al.*, 1991].

4.7.1.2 Inequality Constraints

Inequalities that hold throughout the model are constraints, although they are not part of the QSIM constraint language. Compiling them into QSIM's language requires a step similar to that for compiling the initial inequalities. The inequalities between variables are reduced to ones between between zero and a variable representing the difference between them. An inequality between a variable and a magnitude is simply represented by providing a bound on the variable's quantity space. Because the quantity space is *closed* at its end points, the non-strict inequality constraints are easier to represent. The strict ones require the additional assertion that a magnitude is *unreachable* for the variable [Farquhar *et al.*, 1991].

4.7.1.3 Region Conditions

Section 3.4.1 (page 3.4.1) defined the boundary of a region and specified when a region is active. The region of a *model* is the conjunction of the operating regions of its active instances and the negated regions of its inactive instances. Every behavior of a model must *enter* the model's region and remain within it for some non-zero interval of time. Any subsequent state that lies on the boundary of a model's region is a transition state and may give rise to a new model.

QPC must compile a model's region specification so that QSIM will respect the necessary semantics. There are three distinct aspects to this:

1. Initial conditions. Each region implies an initial condition. For example, `(region > x 0)` implies that `x` $>$ 0 or `x` $=$ 0 and the the sign of `x`'s first derivative is either 0 or positive in the initial state. This is slightly stronger than an initial condition `(>= x 0)`, because the case in which the `x` is decreasing can be ruled out. QPC does not currently exploit the additional constraints on the initial conditions.

2. Initial interval filter. While the initial state may be on the boundary of a region, all behaviors must immediately enter and remain there for some non-zero interval of time. For example, if `x=0` in the initial state `s0` and `(region > x 0)` must be satisfied, then `(> x 0)` must be true in every state immediately following `s0`. Every region condition `(region ?r ?x ?y)` implies a simple quantity relation `(?r ?x ?y)` in the first temporal interval. QPC implements the initial interval filter with a QSIM global filter that checks each state in the first interval and ensures that it has entered the model's region.[5]

3. Transition tests. Once a behavior has entered the model's region, QSIM must detect when the continuation of the behavior reaches the boundary of the model. This is accomplished by negating model's region condition and computing an equivalent QSIM transition test. The transition test need only be applies after the initial interval.

The region specification may require difference variables to be introduced into the QDE in the same way that other inequalities do.

4.7.2 Algebraic Reasoning

The compilation of inequalities results in difference variables being introduced, which must be provided with quantity spaces, as well as `add` constraints to define them. The introduction of the these new variables and constraints makes certain characteristics of the

[5] Most of QSIM's global filters eliminate a state because it is *locally* inconsistent. The initial interval filter eliminates a state because of the *path* that it lies on. The distinction between local and path based filters has not yet been fully incorporated into QSIM's attainable envisionment strategy.

qualitative representation manifest themselves. It particular, it becomes clear that QSIM does not exploit algebraic and quantitative information (it is, after all a qualitative reasoner, not a algebraic or numeric one). For instance, given

```
(add x z       y)
(add x dif-y-x y)
```

QSIM will not infer that `dif-y-x` = `z`. In fact, given that `z=0`, QSIM will hypothesize that `dif-y-x` is either `(minf 0)`, 0, or `(0 inf)`! This is because it does not recognize the algebraic identity between the two `add` constraints. A numeric example is provide by:

```
(add x y z)
qspace(X) = (0 L* inf)
qspace(y) = (0    inf)
qspace(Z) = (0 L* inf)
```

QPC may know that the `L*` in the two qspaces is actually the same landmark representing the same real number. QSIM, however, assumes that the quantity spaces for distinct variables are distinct. Each individual quantity space is totally ordered, but there is no ordering on landmarks across them. As a consequence, given `X=L*`, `Y=0`, QSIM will not infer that `Z=L*` (only that it must be positive).

Fortunately, both of these difficulties are readily solved. In the first case, it is trivial to do the algebraic reasoning necessary to conclude `(equal dif-y-x z)`, and add it as an explicit constraint. In the second case is also trivially solved by adding a corresponding value `(L* 0 L*)` to the `add` constraint linking the identical landmarks. QPC includes a simple term rewriting facility that makes these and similar modifications to the set of constraints. The set of rewrite rules is readily extensible, and enables the addition of algebraic manipulations as they become desirable.

In general, this sort of algebraic and numeric reasoning is very open ended. Identifying algebraically redundant constraints that will strengthen the qualitative predictions is an open research topic.

Most of the algebraic reasoning currently done by QPC involves addition and equality. The constraints are first canonicalized, and then a number of simple algebraic deductions are done. The canonical form uses the signed `sum` constraint to express a number of relations that can also be expressed with more specific constraints (e.g. `((sum -) x y)` is equivalent to `(minus x y)`). After the algebraic inference is done, the constraints are put back into the most specific form. Corresponding values are added to `add` constraints whose variables share landmarks.

4.7.3 Tractability

QPC employs a number of techniques to keep simulations tractable. Several of these were described in the preceding sections, including the use of fixed quantity spaces and ignored qdirs for difference variables and the addition of algebraically redundant constraints.

$$\begin{array}{rcl}
x+y=z, y=0 & \rightarrow & x=z \\
x+y=z, y=z & \rightarrow & x=0 \\
x+y=z, w=z & \rightarrow & x+y=w \\
x+y=z, w=x & \rightarrow & w+y=z \\
x-x=y & \rightarrow & y=0 \\
x+y=x, & \rightarrow & y=0 \\
x+y=z, x'=a, y'=-a & \rightarrow & z'=0
\end{array}$$

Table 4.2: Some of the rules used for adding algebraically redundant, but qualitatively useful, constraints.

The single most important tractability technique is the use of attainable envisionments, which were described in Chapter 2. The attainable envisionment, a graph representation of the set of behaviors, was studied by Fouché in [Fouché, 1992] (and reinvented in somewhat different form by the author). An attainable envisionment does not add new landmarks to quantity spaces, so the number of possible states must be finite (although potentially very large). Furthermore, the attainable envisionment provides exactly the information required by QPC — the set of initial, final, and transition states. The potentially infinite set of behaviors, paths from an initial to a final or transition state, is implicit in the graph. QPC does not require this set to enumerated.

Much to our surprise, generating attainable envisionments (even for QDEs with over two hundred variables) has not proven to be expensive for QPC. The traditional QSIM style simulation, however, would have been intractable even for many of the small problems.

As QPC's applications become even larger and more complex, more sophisticated techniques will need to be used. Fortunately, these techniques are being actively developed by a number of researchers. Recent work by Clancy [Clancy and Kuipers, 1992; Clancy and Kuipers, 1993], for example, substantially extends the techniques available for controlling branching in attainable envisionments. Because of its hybrid architecture, QPC inherits these and other results effortlessly.

4.7.3.1 Model Decompilation

The previous section described the process of compiling a complete model into a single QSIM initial value problem. Solving this problem produces a graph of QSIM states. Some of the states in this graph reach the boundaries of the model's applicability. These boundary (transition) states may give rise to new models. In this section, we discuss how information is recovered (decompiled) from the QSIM states so that it may be asserted back into the KB.

The QSIM state consists of two parts: an assignment of a qualitative value to each variable, and a set of corresponding values for each of the constraints that have been accumulated along the trajectory leading up to the state. There are, however, two other sources of information: 1) the difference variables and ADD constraints that were used to encode inequalities, and 2) the transition conditions that were satisfied by the state.

4.7.3.2 Decompiling Constraints

ADD and EQUAL constraints can be used to infer inequality relations between landmarks and variables in the transition state. This step is the inverse of the inequality compilation process, and is somewhat simpler because only strict inequalities will be generated. E.g. given

```
(ADD x y-x y)
x=x1, y=y1, y-x=0
```

we may infer that `x1 =y1`. Because the KB will compute the transitive closure of a set of inequalities, it is sufficient to assert those that are implied by the individual constraints.

4.7.4 Decompiling Transitions

A transition state touches or crosses some of the boundaries of its model's region of applicability. This information is essential in building the new model. As shown in a previous section, we have compiled the region conditions into the variable value tests of the QSIM transition language. When a state satisfies one of these tests, it is necessary to recover the boundary conditions that are satisfied. This is accomplished by retaining a table of boundary conditions and their compiled tests. The state is then re-evaluated against the tests in order to identify the boundaries that are being reached.

A state may reach a boundary in two distinct ways:

1. it can touch the boundary, or

2. it can be crossing it.

E.g. given boundaries x=0, y=0 and a state in which x=(0 inc) and y=(0 std), we say that the boundary x=0 is being *crossed*, whereas the boundary y=0 is being *touched*.

If boundaries are only being touched, the simulation may continue within the current model. A state that only touches boundaries may be a transition, but need not be terminal.

In either case, it is essential to assert that *the subsequent model must be entering the appropriate new region*. Thus, in the above example, although initially `x=0` in the new model, there would be no guarantee that it would become greater than 0 in the first interval. If we assert `(region > x 0)` in the new model, however, this will be guaranteed.

Multiple boundary crossings must be handled carefully. If some boundaries are touched, and others crossed at a transition, the two cases must be handled differently. The crossings are asserted conjunctively into the subsequent models. The touchings, however, may be ignored. This is because they are disjunctive in nature, any subset of them may result in boundaries actually being crossed. This disjunction will be handled correctly by the normal model refinement mechanism. It is only in the special case that a single boundary is reached and touched, that the corresponding region must be asserted.

4.8 Differences Between Theory and Practice

There are several ways in which the implementation of QPC varies from its theoretical exposition in Chapter 3.

1. The theory was developed in terms of distinct databases. The children of a database, for example, were created by adding ground formulae to their parent so as to ensure that instances could be proven active or not active. Although they allowed for a clean theoretical development, distinct databases would be a poor implementation choice for two reasons.

   (a) The children are supersets of the parent, and differ from it in only a few clauses. Furthermore, any inference that was true given the parent is also true given the child. Distinct databases do not exploit this redundancy.

   (b) It is useful to reason explicitly about and quantify over databases. This is not possible within the same logic that is used to reason about the formulae within a database.

   In order to address these issues, the implementation represents the databases as frames called *models* in the KB. `Subscope` relations between models, and rules enforce the semantics of monotonic inheritance between a parent and its children. Relations that were formerly true within a database retain their semantics by adding an additional an argument indicating the database in which they hold.

2. Meta-theoretical relations used in chapter 3 are implemented using reification. E.g. relations between databases hold between models. Relations such as `conditions` and `constraints` that are used by the QPC algorithm are also reified.

3. Instead of using a recursive formulation, the implementation of QPC uses an agenda to guide the exploration of models. This allows considerably more control over when (and whether) models are handled. The agenda may be dynamically reordered, equivalent entries may be merged, resource bounds may easily employed, and so on.

4. The refinement operation is expanded into the basic loop, so that children are added directly to the agenda. This is because determining whether or not a child is ambiguous (and requires additional refinement) is an expensive operation, and needs to be be under the control of the agenda.

4.9 Improvements

In this section, we will examine some of the shortcomings and possible improvements that could be made to the implementation of QPC. QPC's worst problem is that of speed. There are several reasons for this:

1. The strategy of generate and filter may be explosive if the filters are applied too late. Some of the filters used in QPC could probably be applied earlier.

2. Because QPC is a hybrid system, the knowledge needed to make certain inferences may not always be available in the context within which it is desired. For example, QSIM knows about constraints and how to propagate values through them. Thus, given x=0, and (m+0 x y), QSIM will readily infer that y=0. But QSIM will not do this until a complete initial value problem has been proposed to it. The KB, however, might well be able to make use of such an inference before it is able to form a complete initial value problem. One temptation here is to re-express aspects of QSIM's expertise in the KB. This temptation towards building a monolithic system, however, violated the purpose of QPC to begin with. Another approach is to modify the interface to the subsystems in order to make it possible to pose well defined sub-problems. A third option would be to go to the extra trouble of creating small well defined problems that satisfy the existing interfaces. Considerably more of this could be done. In the current implementation, we have erred on the side of maintaining the integrity of the system components.

3. Inequality reasoning is probably the single most time consuming component of QPC. It is both slow and undirected. Fortunately, this could be dramatically improved by using a special purpose inequality reasoner. There are a variety of algorithms and representations that are appropriate for doing inequality reasoning quickly. It would substantially speed up QPC if they could be used. Certain limitations of Algernon made this awkward to achieve.

   Another way to improve the efficiency is to provide more information about quantities. In particular, QPC makes no use of dimensional information. Thus, it is possible to compare quantities with incompatible dimensions. G. Brajnik is working to incorporate dimensional reasoning into QPC.

4. Algernon is slow. There main reasons for this are: 1) rule execution is completely interpreted, and 2) indexing is only done on the functor and first argument, 3) all results are cached. A system that compiled rules, provided more sophisticated indexing of facts, and selective caching should provide several orders of magnitude speedup. As most of QPC's runtime is spent within the KRS, this would result in a tremendous speedup of the entire system.

4.9.1 Scope of Implementation

QPC has been implemented in pure COMMON LISP, and should run with little or no modification on any COMMON LISP system. QPC is composed of four subparts: the model builder, the basic knowledge base, the Algernon knowledge representation system, and the QSIM qualitative simulation system. The model builder consists of around 8000 lines of lisp code in some 20 files. The basic knowledge base comprises another 2000 lines of facts, definitions, and rules which generate about 250 frames, and 550 rules. Algernon is also about 8000 lines of code in 20 files. QSIM is on the order of 50,000 lines of code.

Chapter 5

Experience

This chapter is organized into four sections. In the first section, we present several simple example problems for QPC with detailed descriptions of the actual inference process and output. These problems should give the reader a good sense for how the implementation works, and provide a concrete basis for interpreting many of the concepts discussed previously. In the second section, we discuss a benchmark problem and compare QPC with the GIZMO and QPE systems developed by Ken Forbus. The third and fourth sections discuss applications of QPC in the domains of botany and chemical engineering. These applications provide substantial evidence for both the expressive power of the QPC modeling language, and the inference methods employed. The botany application shows that QPC can readily answer simple questions extracted from a substantial view library. The chemical engineering application shows that QPC can solve mathematically more sophisticated problems that involve hundreds of variables and constraints. We also discuss some extensions to the language for the modeling engineering abstractions of sources and sinks.

5.1 Illustrative examples

In this section, we describe several simple examples to illustrate the functioning of QPC. We first describe the domain theory, which is shared by all of them. Thereafter, we will present the scenario definition and results of running QPC.

5.1.1 Simple Fluid Dynamics Domain Theory

The domain theory is patterned on the contained stuffs ontology of Hayes [Hayes, 1985], and existing QP theory domain theories reported in [Skorstad and Forbus, 1989; Forbus, 1984a].

In Algernon, and consequently in the implementation of QPC, types are represented as sets. E.g. `containers` is the set of all containers. In the ontology of Figure 5.1, the only objects with `mass` are the `physobs` including `contained-liquids`. Other objects such as

```
(deftaxonomy
    (objects
     (containers)
     (fluid-sources)
     (drains)
     (paths (heat-paths) (fluid-paths))
     (physobs
      (contained-stuff
       (contained-liquids)
       (contained-gases)))))
```

Figure 5.1: Ontology for simple fluids domain theory.

`containers` are massless objects with other physical properties such as `height` or `volume`.

```
(defrelation consider (scopes nil)
  :type inherited-relations
  :comment "Things to consider in a scope.")
(defrelation aligned (scopes paths booleans)
  :type inherited-relations)
(defrelation fluid-connection
   (scopes containers fluid-paths containers)
  :type inherited-relations)
(defrelation flows-into (scopes fluid-sources containers)
  :type inherited-relations)
(defrelation drains-outof (scopes drains containers)
  :type inherited-relations)
(defrelation container (contained-stuff containers))
(defrelation contents  (containers contained-stuff)
  :inverse  container)
```

Figure 5.2: Relations for simple fluids theory.

Figure 5.2 lists the structural relations used in the domain theory. The `consider` relation represents information about *modeling decisions*, rather than the domain itself. It provides a simple means of doing meta-level reasoning about the relevancy of certain model fragments. Some of the model fragments use `consider` to restrict their activity to scenarios in which they are relevant. As the size and scope of domain theories grows this sort of reasoning becomes more essential. Although a nascent research area, some excellent work has already been done. Some interesting approaches are discussed by Forbus and Falkenhainer

[Falkenhainer and Forbus, 1991], Rickel [Rickel and Porter, 1992], and Levy and Iwasaki [Levy and Iwasaki, 1992].

As has been discussed, structural conditions are normally true within a scope, thus they have a scope as their first argument. Furthermore, if a structural condition is true in one scope, then it should be true in any of its subscopes. This behavior is true of all `inherited-relations`. Therefore, the structural relations are specified to be of type `inherited-relations`.

Some of the relations in this ontology are not scoped. This is appropriate in case they are definitional. E.g. in this ontology, a `contained-stuff` is defined by its `container`. Thus, the `contents` relation does not change from one scope to another. The effect of a `contained-stuff` changing container is achieved by mass transfer to a `contained-stuff` in another `container` via `fluid-flow`. Therefore, the relations `contents` and its inverse, `container`, are not scoped.

```
(defrule models
  (entity ?m ?contents)
  (container ?contents ?can)
  ->
  (entity ?m ?can))
(defrule  models
  (entity ?m ?can)
  (:retrieve (contents ?can ?contents))
  ->
  (entity ?m ?contents))
(defrule
  (fluid-connection ?s ?po1 ?path1 ?po2)
  (aligned ?s ?path1 true)
  ->
  (entity ?s ?po1)
  (entity ?s ?path1)
  (entity ?s ?po2))
```

Figure 5.3: Rules for simple fluids domain theory.

Figure 5.3 lists the rules to do some simple inference within the domain theory. These rules extend the set of entities in a model along important topological relationships. This is a simple instance of an important class of inference that helps to complete the initial description of a situation. E.g. given that a container is an entity of interest in a model, QPC will infer that its contents (if any) is also of interest. Sound rules (and good heuristics) for such reasoning is an important area of research in automatic model building. Some initial work may be found in [Falkenhainer and Forbus, 1991], Rickel [Rickel and Porter, 1992], and Levy and Iwasaki [Levy and Iwasaki, 1992].

Figure 5.4 lists the quantity type definitions for the domain theory. To review, a quantity

```
(defQuantity mass physobs)
(defQuantity height          containers)
(defQuantity fluid-level     containers)
(defQuantity interior-volume containers)
(defQuantity bottom-pressure containers)
(defQuantity volume   contained-stuff)
(defQuantity pressure contained-stuff)
(defQuantity level contained-liquids)
(defQuantity pressure-difference fluid-paths)
```

Figure 5.4: Quantity type definitions for simple fluids domain theory.

type, such as `mass` is a second order function which, when applied to a physical object, returns the variable that is object's mass.

```
(defModelFragment PHYSOB
  :participants ((object :type physobs))
  :consequences ((>= (mass object) zero)))

(defModelFragment CONTAINER-MF
  :participants ((container :type containers))
  :consequences ((q+0 (interior-volume container) (height container))
                 (>= (height container) zero)
                 (>= (interior-volume container) zero)))
```

Figure 5.5: View-like model fragment definitions for the simple fluids domain theory.

The two model fragments in Figure 5.5 help to define the quantities of containers and physical objects. Another way of representing this sort of information would be to introduce a hierarchy of quantity types. Many quantities are what one might call `physical-quantities`, such as `mass`, `height`, `pressure`, `volume`, and so on, that are all non-negative quantities of physical objects. Note that the relations between `interior-volume` and `height` is a qualitative proportionality `q+0` rather than a functional constraint. This allows the domain theory to be extended to include other influences on the `interior-volume` such as the diameter of the container. The `q+0` influence differs from `q+` in that it suggests a correspondence at `zero`.

The `contained-liquid` model fragment is a key one for this domain theory. It specifies the relations between the basic variables describing a `contained-liquid`. This model fragment makes an implicit assumption that there is only a single `contained-liquid` in a `container`. The assumption is reflected in the consequences that the bottom pressure of

```
(defModelFragment CONTAINED-LIQUID
  :participants
  ((liquid :type contained-liquids
   (can :type containers))
  :structural-conditions ((container liquid can))
  :consequences
  ((q+0 (pressure liquid) (level liquid))
   (q+0 (level liquid)    (volume liquid))
   (q+0 (volume liquid)   (mass liquid))
   (>= (fluid-level can)     zero)
   (>= (bottom-pressure can) zero)
   (= (pressure liquid) (bottom-pressure can))
   (= (level liquid)    (fluid-level can))
   (correspondence (volume liquid) (interior-volume can)
                   (level liquid) (height can))))
```

Figure 5.6: View-like model fragment definition for contained liquid.

the container equals the pressure of the liquid, and that the level of the liquid is the same as the fluid-level of the container. The other proportionalities and inequalities would apply in a more sophisticated domain theory as well.

In Forbus's QP theory, the preceding model fragments would have been referred to as *views*, which help to describe objects, but cannot produce any dynamic behavior. The next model fragment for `fluid-flow` is what Forbus called a *process*.

The `fluid-flow` model fragment is active whenever there is an aligned fluid connection between two containers, each of which contain a liquid. This is somewhat different than the definition for the fluid flow process that Forbus uses. Forbus's definition is directional and applies only when there is a positive flow from source to destination. The QPC definition is not conditioned on the pressure difference between the two containers (or ends of the fluid path). This is not an accident, but reflects an important philosophical difference. The equations contributed by the model fragment apply correctly regardless of the pressure difference. QPC considers the equations to be the central object. The QP theory description can be readily recovered from the model fragment by examining the sign of the pressure difference or flow rate.

Once again, it is worth noting the compositional nature of the description. As the domain theory is extended to include other phenomena that affect the flow rate such as the viscosity of the liquid, the diameter of the fluid path, and so on, this basic model fragment need not be modified. This is also true of other processes that affect the masses of the liquids in the source and destination.

The `overflow` model fragment provides a very simple description of overflow in a container. The level of the contents of the container is allowed to rise above the top of the container, and the rate of the overflow is proportional to the difference between the two

```
(defModelFragment FLUID-FLOW
  :participants
  ((source :type containers)
   (s-liquid :type contained-liquids)
   (d-liquid :type contained-liquids)
   (destination :type containers)
   (fluid-path :type fluid-paths))
  :structural-conditions
    ((contents source s-liquid)
     (fluid-connection source fluid-path destination)
     (aligned fluid-path true)
     (contents destination d-liquid))
  :quantities (flow-rate)
  :consequences
  ((q+0 (flow-rate fluid-flow) (pressure-difference fluid-path))
   (add (pressure-difference fluid-path) (bottom-pressure destination)
        (bottom-pressure source))
   (I+ (mass d-liquid) (flow-rate fluid-flow))
   (I- (mass s-liquid) (flow-rate fluid-flow))))

(defModelFragment overflow
  :participants ((c :type containers)
                 (l :type contained-liquids))
  :structural-conditions
    ((consider overflow c)
     (contents c l))
  :operating-conditions
  ((> (level l) (height c)))
  :quantities (rate delta-height)
  :consequences
  ((I- (mass l) (rate overflow ))
   (add (height c) (delta-height overflow) (level l))
   (q+0 (rate overflow) (delta-height overflow))))
```

Figure 5.7: Process-like model fragment definitions for the simple fluids domain theory.

measures. The overflowing liquid does not go anywhere, its mass is simply removed from the contents of the container. This is only useful for very naive models, and it usefulness is explicitly limited by the `consider` condition. QPC must infer that overflow is to be considered in a particular container before this model fragment can be instantiated.

```
(defModelFragment fluid-source
  :participants ((source :type fluid-sources)
                 (c :type containers)
                 (l :type contained-liquids))
  :structural-conditions
    ((flows-into source c)
     (contents c l))
  :quantities (source-flow-rate)
  :consequences ((> (source-flow-rate fluid-source) zero)
                 (I+ (mass l) (source-flow-rate fluid-source))))
(defModelFragment fluid-drain
   :participants ((drain :type drains))
                  (c :type containers)
                  (l :type contained-liquids))
   :structural-conditions
     ((drains-outof drain c)
      (contents c l))
   :quantities (drain-flow-rate)
   :consequences ((>= (drain-flow-rate fluid-drain) zero)
                  (I- (mass l) (drain-flow-rate fluid-drain))
                  (Q+0 (drain-flow-rate fluid-drain)
                       (bottom-pressure c))))
```

Figure 5.8: Variants on flow in the simple fluids domain theory.

The `fluid-source` and `fluid-drain` model fragments are primitive models of liquid from a source into a container and from a container into a drain. Neither the source nor the drain have limited capacity. In Section 5.4, we will see how this same effect can be achieved by ignoring certain influences.

5.1.2 Example 1: Container with Drain

The first scenario consists of a single tank, `a`, with a drain. The tank initially contains some liquid `a-cnts`, whose `level` is below the `height` of `a`. After loading the domain theory and asserting the scenario definition (Figure 5.10), QPC creates an initial model for the scenario , and instantiates model fragments in it (Figure 5.11). At this point, we can examine the influences that are contributed by the model fragments (Figure 5.12).

Algernon automatically generates names for frames based on the inference path that leads to them. The first component of the name is the name of another frame, and the subsequent components are the names of relations. E.g. `a-cnts.mass.derivative` is the

```
(defentity c-liquid
  :goal (contents c c-liquid)
  :participants
  ((source :type fluid-sources)
   (c :type containers))
  :structural-conditions ((flows-into source c))
  :consequences ((contents c c-liquid)
                (isa c-liquid contained-liquids))
  :initially ((= (mass c-liquid) zero)
             (= (level c-liquid) zero)))
```

Figure 5.9: Entity definition to create a contained liquid. Newly created liquids initially have zero `mass` and `level`. No attempt is made to create the entity until the `:goal` clause is queried.

```
(defScenario scene
  :pname "Tub with drain."
  :entities ((a      :type containers)
             (a-cnts :type contained-liquids)
             (drain  :type drains))
  :relations ((contents a a-cnts)
              (drains-outof drain a)
              (> (height a) zero))
  :initial-conditions
  ((> (level A-cnts) zero)
   (< (level A-cnts) (height A))))
```

Tub with drain.
Plotting signs of: a.fluid-level.

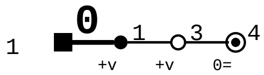

Figure 5.10: Liquid drains out of the container `a`.

```
Model SCENE.INITIAL-MODEL is the initial model.
Initializing model SCENE.INITIAL-MODEL.
Instantiated views for SCENE.INITIAL-MODEL:
    0. (container-mf a)
    1. (physob a-cnts)
    2. (contained-liquid a-cnts a)
    3. (drain-mf drain a a-cnts)
```

Figure 5.11: QPC output showing instances for the tank with a drain.

```
Influences:
  a-cnts.mass.derivative
    <- Q-SUB drain-mf.instantiation.drain-flow-rate
  drain-mf.instantiation.drain-flow-rate
    <- Q+0 a.bottom-pressure
  a-cnts.pressure
    <- Q+0 a-cnts.level
  a-cnts.level
    <- Q+0 a-cnts.volume
  a-cnts.volume
    <- Q+0 a-cnts.mass
  a.interior-volume
    <- Q+0 a.height
```

Figure 5.12: The influences implied by the active model fragments in the model.

derivative of the mass of `a-cnts`, or `(derivative (mass a-cnts))`. This heuristic works well for binary relations, but may yield names that are too long to read easily or be uninformative for relations with greater arity.

Because there are no ambiguous model fragments, the model is complete. This enables QPC to move on to the next step, resolving the influences into constraints and concluding that exogenous variables are constant.

```
Asserting that the world is closed.
Model SCENE.INITIAL-MODEL:
Status: MODEL-COMPLETE
Resolving influences.
Inferring that all uninfluenced vars are constant.
Resolving correspondences.
Resolving quantity spaces.
  The quantity spaces are totally ordered.
  Narrowing the qspaces.
  Asserting shared quantity spaces.
Collecting region transitions.
Defining a QDE.
  Cleaning up QDE for QSIM:
    every qspace must have a variable in a constraint,
    unreachable values must be in a qspace,
    corresponding values must be in qspaces.
Constructing the QSIM QDE.
```

Figure 5.13: QPC output. At this point, the model is complete and QPC has created a QSIM QDE.

We may now examine the constraints that QPC has constructed (Figure 5.14). Each of the influences has been resolved into a constraint. The `I-` influence has been converted into a `d/dt` constraint between `a-contents.mass` and its derivative, and a `(sum -)` constraint between the influencing variable and the derivative. The exogenous variable `a.height` has been inferred to be constant, as is `a.interior-volume` which is linked to it by a `m` constraint.

Once the constraints have been identified, QPC converts the correspondences into corresponding values on the constraints, and ensures that the quantity spaces are totally ordered. After collecting the region transitions, QPC is ready to define the QDE.

There are several minor pieces of cleanup that are done in order for the QDE to be acceptable for QSIM. Figure 5.15 shows the constraint portion of the QDE for this model. This differs slightly from the original set of constraints collect from the knowledge base. QPC uses the most specific QSIM constraint available. E.g. the `(sum -)` constraint is converted into `minus`. Quantity relations are encoded using difference variables, which are created and defined via `add` constraints. The meaning of the difference variable is eas-

```
Constraints:
  ((d/dt a-cnts.mass a-cnts.mass.derivative))
  (((sum -) drain-mf.instantiation.drain-flow-rate a-cnts.mass.derivative))
  (((m +) a.bottom-pressure drain-mf.instantiation.drain-flow-rate))
  (((m +) a-cnts.level a-cnts.pressure))
  (((m +) a-cnts.volume a-cnts.level))
  (((m +) a-cnts.mass a-cnts.volume))
  (((m +) a.height a.interior-volume))
  ((constant a.height))
  ((constant a.interior-volume))
```

Figure 5.14: The constraints from the knowledge base after influences have been resolved and exogenous variables inferred to be constant.

```
(constraints
 ((add a.height dif-<a.fluid-level>-<a.height> a-cnts.level))
 ((add a.height dif-<a-cnts.level>-<a.height> a-cnts.level))
 ((constant a.height))
 ((constant a.interior-volume))
 (((m +) a.bottom-pressure drain-mf.instantiation.drain-flow-rate)
  (0 0))
 (((m +) a-cnts.level a-cnts.pressure) (0 0))
 (((m +) a-cnts.volume a-cnts.level) (0 0))
 (((m +) a-cnts.mass a-cnts.volume) (0 0))
 (((m +) a.height a.interior-volume) (0 0))
 ((d/dt a-cnts.mass a-cnts.mass.derivative))
 ((equal dif-<a-cnts.level>-<a.height>
         dif-<a.fluid-level>-<a.height>))
 ((equal a-cnts.level a.fluid-level))
 ((equal a-cnts.pressure a.bottom-pressure))
 ((minus drain-mf.instantiation.drain-flow-rate
         a-cnts.mass.derivative))))
```

Figure 5.15: The constraints from the QDE that QPC defines for the model.

ily determined from its name: `dif-<v1>-<v2> = v1 - v2`. The uninfluenced variables are constrained to be constant, and some minor algebraic inference has been done. E.g. the two difference variables have been inferred to be equal, and the defining `add` constraint for `dif-<a.fluid-level>-<a.height>` has been rewritten into a canonical form with `a-cnts.level` replacing the equivalent `a.fluid-level`. QPC collects initial values

```
Building an initial state.
  Initial values:
          (dif-<a-cnts.level>-<a.height> ((minf 0) nil))
          (dif-<a.fluid-level>-<a.height> ((minf 0) nil))
          (a.height ((0 nil) nil))
          (a.fluid-level ((0 nil) nil))
          (a-cnts.level ((0 nil) nil))

Run time: 0.030 seconds to initialize (S-1 S-2).
  Built initial state S-0, which has at most 2 consistent completions.
Simulating from initial state S-0.

Run time: 0.050 seconds to simulate 3 states.
  Generated 1 behavior.
```

Figure 5.16: QPC output. The initial values for variables in the QDE are computed, and QSIM is called to quickly generate predictions.

for the QDEs from the knowledge base, and calls QSIM to generate the simulation (Figure 5.16). Figure 5.10 shows the graph of the tree of models and states that QPC generates for this scenario. There is a single model, and the level of fluid in the container decreases to zero.

5.1.3 Example 2: Tank with Drain and Overflow

In this example, the tank is being filled, and is allowed to overflow. The initial model is similar to that in example 1, however, there are three possible final states depending on the relation between the top of the container and the level of the fluid. Figure 5.17 shows both the scenario definition and the graph of models and states produced by QPC. The graph shows two models, and plots the sign and direction of change of the distance that the level of the contents of `a` is below the top of `a`. In model 0, `a` is filling. There are three possible behaviors. In behavior 1, the fluid reaches the top and is increasing. In behavior 2, the system reaches equilibrium below the top, and in behavior 3, the system reaches equilibrium exactly at the top. The first behavior ends in a transition state `S-4`.

```
(defScenario scene
   :pname "Tub with drain and inflow."
   :entities ((a      :type containers)
              (a-cnts :type contained-liquids)
              (drain  :type drains)
              (source :type fluid-sources))
   :relations ((contents A A-cnts)
               (drains-outof drain a)
               (flows-into source a)
               (> (height a) zero))
   :initial-conditions ((= (level A-cnts) zero)))
```

Tub with inflow and drain that may overflow.
Plotting signs of: dif-<a-cnts.level>-<a.height>.

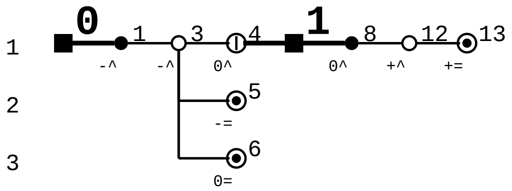

Figure 5.17: Scenario definition and graph of models and states generated by QPC for a tank with an inflow and a drain. Model 1 is created when the tank overflows.

QPC builds the model `m-1` to describe the situation after the transition from state `s-4`. Figure 5.18 show the output of QPC as the new model is being created. The transition state `s-4` satisfies the operating conditions for the `overflow` model fragment instance; the level of `a.contents` is moving past `a.height` in `s-4`. The new model, `m-1` is created an initialized by

1. asserting that the entities of the initial model are also entities in `m-1`,

```
  Created model #<Model: M-1>, a transition from state S-4.
Initializing model M-1.
  Linking M-1 to predecessor, SCENE.INITIAL-MODEL.
  Inheriting model assumptions:
      [none]
  Inheriting initial conditions assumptions:
      (region-active q> a-cnts.level  a.height)

Instantiating view OVERFLOW.
  A-CNTS -- ABSTRACT-L
  A -- ABSTRACT-C

  Assert the qspace orderings from S-4.
  Assert the correspondences from S-4.
  Inheriting values:
        (a.height ((0 inf) nil))
        (a.interior-volume ((0 inf) nil))
        (a-cnts.mass ((0 inf) nil))
        (a.fluid-level ((0 inf) nil))
        (a.bottom-pressure ((0 inf) nil))
        (a-cnts.level ((0 inf) nil))
        (a-cnts.pressure ((0 inf) nil))
        (a-cnts.volume ((0 inf) nil))
        (drain-mf.instantiation.drain-flow-rate ((0 inf) nil))
        (a-cnts.mass.derivative ((0 inf) nil))
        (fluid-source.instantiation.source-flow-rate ((0 inf) nil))
```

Figure 5.18: QPC output upon the creation of the transition model.

2. asserting that `m-1` must enter the operating region `(> a.cnts.level a.height)`, and

3. by asserting the quantity conditions from state `s-4`.

This results in an active instance of the overflow model fragment being created.

Once these assertions have been made, the new model is handled exactly like the initial model. The QDE is generated, the initial values for the QDE are computed, and QSIM generates an additional behavior. Given the naive model of overflow, the level of the fluid rises above the top of container, and equilibrium is reached.

5.1.4 Example 3: Tank with Drain and Inflow, but no Initial Conditions

In this example, we look at the tank with an inflow and drain again. The scenario definition and output are shown in Figure 5.19. The difference is that there are no initial conditions. The initial model is ambiguous. The initial level of the fluid in `a` is unknown, so it is not possible to decide whether `overflow` is active or not. QPC refines the initial model into two complete models. In model 1, `a` is overflowing, and in model 2 it is not. Refinement, of course, only guarantees that the models are complete, not that the specification of the initial state is complete. This example also illustrates how incomplete initial conditions are handled by QSIM. Both complete models have several possible initial situations. Let us examine model 2, in which `a` is not initially overflowing. State `s-3` is identical to the initial state of the preceding example, and the subgraph rooted at state `s-3` is identical to the subgraph rooted in `s-1` of the preceding example. The container is empty and fills, perhaps overflowing in state `s-60`. In behavior 10, the level is below the top, and decreases to an equilibrium. The equilibrium state `s-58` is identical to state `s-2` in the next behavior. In behavior 15, the level starts out below the top and increases. `S-63`, the second state in this behavior, is equivalent to state `s-59` in behavior 2, which may lead to an overflow. In behavior 16, the level starts out at the top, but decreases to equilibrium. In behavior 17, the system is in equilibrium with the level at the top. Behavior 18 is somewhat anomalous. Normally an initial state that is a transition can be eliminated, as QPC guarantees that the model will apply for an interval. State `s-9`, however, is equivalent to the state `s-60` in the envisionment graph, and marking it as inconsistent will result in QSIM marking `s-60` as inconsistent as well.

```
(defScenario scene
    :pname "Tank with inflow and drain that may overflow."
    :entities ((a      :type containers)
               (a-cnts :type contained-liquids)
               (drain  :type drains)
               (source :type fluid-sources))
    :relations ((contents A A-cnts)
                (drains-outof drain a)
                (flows-into source a)
                (consider overflow a)
                (> (height a) zero)))
```

Tub with inflow and drain that may overflow.
Plotting signs of: dif-<a-cnts.level>-<a.height>, a-cnts.level.

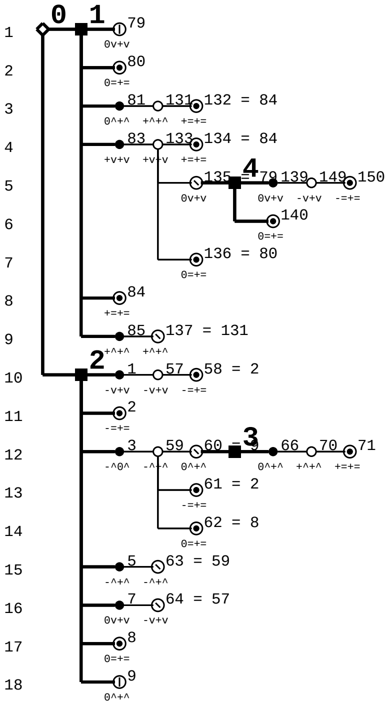

Figure 5.19: Scenario definition and output for a tank with inflow and drain that is allowed to overflow. There are no initial conditions. Because the level of the fluid is unknown, the `overflow` fragment is ambiguous. The initial model is refined into two new models. In model 1, the tank is initially overflowing. The transition from state `s-135` to model 4 occurs when the tank ceases to overflow. In model 2, the tank is initially not overflowing. The transition from state `s-60` to model 3 occurs when the tank overflows.

5.1.5 Examples 4 and 5: Two Tanks, with Inflow and Drain

This section describes two slightly more challenging problems. In the first, we have a two tank cascade. There is a feed into tank `a`, and a connection to tank `b`, which has a drain. Figure 5.22 shows two different graphs of the behavior of this system. Simulating the QDE that results from this problem is rather difficult for qualitative simulators. A simple QSIM like system branches intractably on the second derivative of the net flow from `a` to `b`. We do not show such a graph here. The graph in Figure 5.24 shows QSIM using an attainable envisionment. This is tractable, and indeed produces a rooted graph with only five branches and a single quiescent final state. If we look closely at the branches of the tree, however, we notice that each one is cyclic. State `s-14` is equivalent to its ancestor `s-10`, and so on. In fact, the subgraph rooted at state `s-9` contains numerous cycles with a single exit — `s-16`. Analyzing this sort of graph is the focus of recent work by Clancy on behavioral abstraction [Clancy and Kuipers, 1993]. By using a version of QSIM with Clancy's behavioral abstraction techniques included, QPC gets the very simple abstracted behavior tree shown in Figure 5.25. State `32` is labeled with a small box to indicate that it is an aggregation. It represents the entire subgraph between states `s-9` and `s-16` of Figure 5.24. A similar improvement is also seen in the next problem.

There are two tanks `a` and `b`. Initially `a` contains some liquid, and `b` is empty. We consider the possibility that the tanks may overflow. Figure 5.23 again shows two graphs similar to those of figure 5.22. In the initial model, there are three branches. In the first behavior, tank `b` overflows. Note that `a` will not overflow, as the level of fluid in it must decrease. A new model is constructed including the overflow. As in the preceding example, there are five branches using the envisionment. Using the behavioral abstraction methods, these five are reduced to a single aggregate behavior.

These two examples provide a nice example of the power of a hybrid inference system such as QPC. QPC leverages the work of other researchers who are developing sophisticated simulation techniques for QSIM.

```
(defScenario scene
    :pname "Utube with inflow into A and drain out of B."
    :entities ((A          :type containers)
               (B          :type containers)
               (fluid-path :type fluid-paths)
               (spigot)
               (drain))
    :relations ((aligned fluid-path true)
                (fluid-connection A fluid-path B)
                (flows-into spigot A)
                (drains-outof drain B)
                (> (height A) zero)
                (> (height B) zero)))
```

Figure 5.20: Scenario definition for a two tank cascade.

Utube with inflow into A and drain out of B.
Plotting signs of: a.fluid-level, b.fluid-level.

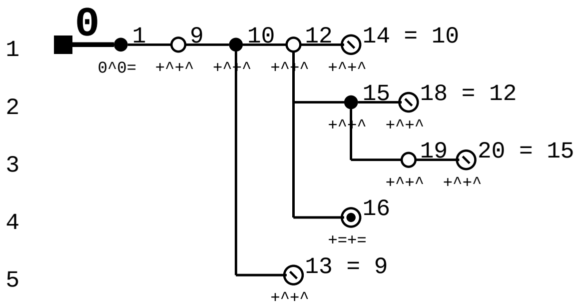

Figure 5.21: QPC output for the two tank cascade.

Utube with inflow into A and drain out of B.
Plotting signs of: a.fluid-level, b.fluid-level.

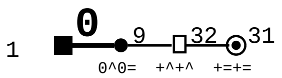

Figure 5.22: QPC output for the two tank cascade using Clancy's aggregation technique.

```
(defScenario scene
    :pname "Utube that may be overflowing."
    :entities ((A          :type containers)
               (A-cnts     :type contained-liquids)
               (B          :type containers)
               (B-cnts     :type contained-liquids)
               (fluid-path :type fluid-paths))
    :initial-conditions ((> (level A-cnts) zero)
                          (< (level A-cnts) (height A))
                          (= (level B-cnts) zero))
    :relations ((consider overflow a)
                (consider overflow b)
                (contents A A-cnts)
                (contents B B-cnts)
                (aligned fluid-path true)
                (fluid-connection A fluid-path B)
                (> (height A) zero)
                (constant (height a))
                (> (height B) zero)
                (constant (height b))))
```

Figure 5.23: Scenario definition for two connected tanks. Overflow is allowed.

Utube that may be overflowing.
Plotting signs of: a.fluid-level, b.fluid-level.

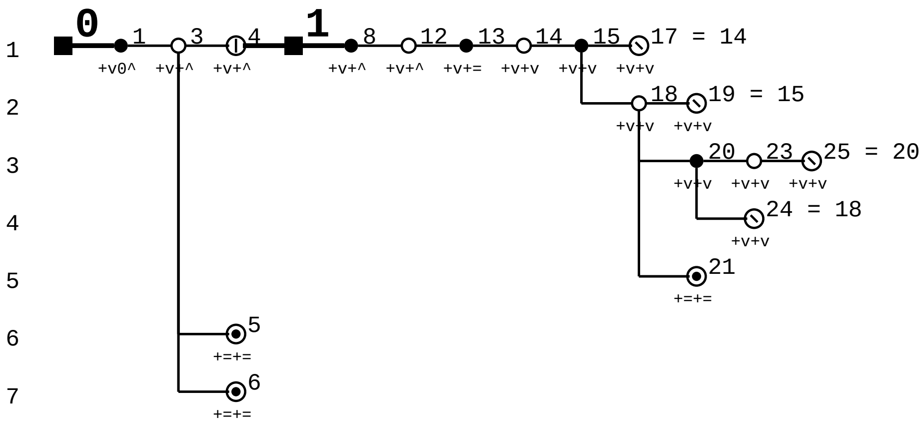

Figure 5.24: QPC output for two connected tanks, allowing overflow.

Utube that may be overflowing.
Plotting signs of: a.fluid-level, b.fluid-level.

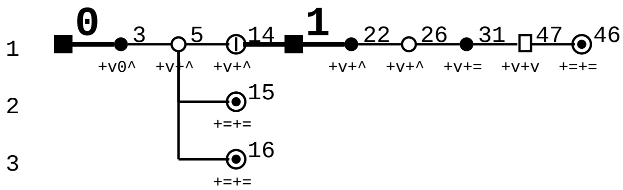

Figure 5.25: QPC output for two connected tanks, allowing overflow using Clancy's aggregation technique.

5.2 Four Blobs

This example is taken from Forbus' thesis [Forbus, 1984a] that described the GIZMO implementation of QP theory. It was subsequently discussed in [Forbus, 1990] in order to compare GIZMO with the newer implementation of QP theory, QPE. Thus, it is fitting that we use the example to compare both of those systems with QPC.

5.2.0.0.1 Scenario There are three masses `x`, `y`, and `z` in thermal contact with a fourth central mass `c` (Figure 5.26). We will examine the two scenarios defined in Figure 5.27. In the first, `c` is initially hotter than `x`, `y`, and `z`. This provides a unique initial state, and an attainable envisionment for comparison with GIZMO. In the second scenario, there are *no initial conditions*. This yields a total envisionment, and is used to compare against QPE.

5.2.0.0.2 Domain Theory The domain theory, shown in Figure 5.28, is extremely simple. The `physob` model fragment requires the temperature of a physical object to be greater than zero, and states that there is a qualitative proportionality between the temperature and heat of an object with a correspondence at zero (i.e. temperature is measured along the Kelvin scale). The `heat-flow` model fragment states that if there is a connection between two objects, there is a heat flow between them, and that their heats are directly influenced by the temperature difference.

Note that `heat-flow` is not conditioned on a non-zero temperature difference, as is typical in QP theory definitions. Again, this is not accidental, but reflects an important philosophical difference. Model fragment definitions should reflect the underlying equations, not necessarily cognitively useful description. The QP theory descriptions in terms of active processes can be extracted from the QPC descriptions. E.g. if `delta-t` > 0, then there is heat flow from `a` to `b`, `delta-t` < 0 then there is heat flow from `b` to `a`, otherwise there is no heat flow.

5.2.0.0.3 Results QPC solves the 4-blobs problem with initial conditions in about a minute. Loading the domain theory, instantiating the scenario, building the QDE, and collecting the initial conditions takes about 30 seconds. Solving the QDE to generate

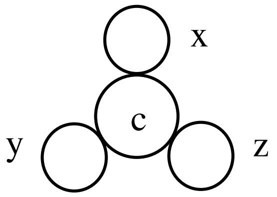

Figure 5.26: The four blobs problem. There are three objects `x`, `y`, and `z` in thermal contact with a center object `c`.

```
(defScenario blobs
   :entities ((c :type objects)
              (x :type objects)
              (y :type objects)
              (z :type objects))
   :relations ((connected c x)
               (connected c y)
               (connected c z))
   :initial-conditions
   ((> (temp c) (temp x))
    (> (temp c) (temp y))
    (> (temp c) (temp z))))
```

```
(defScenario blobs-total
   :entities ((c :type objects)
              (x :type objects)
              (y :type objects)
              (z :type objects))
   :relations ((connected c x)
               (connected c y)
               (connected c z)))
```

Figure 5.27: Scenario definitions for the four blobs problem. `Blobs` will generate an attainable envisionment, whereas `blobs-total` will produce a total envisionment.

```
(defrelation connected (objects objects))
(defQuantity temp (objects))
(defQuantity heat (objects))
(defModelFragment physob
  :participants ((a :type objects))
  :consequences ((> (temp a) zero)
                 (q+0 (temp a) (heat a))))
(defModelFragment heat-flow
    :participants ((a :type objects)
                   (b :type objects))
    :structural-conditions ((connected a b))
    :quantities (delta-t)
    :consequences ((add (temp b) (delta-t heat-flow) (temp a))
                   (I- (heat a) (delta-t heat-flow))
                   (I+ (heat b) (delta-t heat-flow))))
```

Figure 5.28: Domain theory for the four blobs problem.

System	Attainable		Total	
QPC	50 sec	327 states	1800 sec	396 states
GIZMO	3176 sec	89 sit	NA	NA
QPE	cannot generate		> several hours	> 5200 sit

Table 5.1: Comparison of QPC, GIZMO, and QPE on the four blobs problem.

behaviors takes about 20 seconds. There are 327 unique states and 691 branches in the behavior tree. Importantly, there is only one final state, in which the system is in thermal equilibrium.

The 4-blobs problem without initial conditions is somewhat harder. Setting up the problem is the same as the preceding one and also takes about 30 seconds. Solving the QDE takes much longer, about 1784 seconds, or *30 minutes*. It is worth noting that QSIM is not at all optimized for problems of this sort that require a large number of initial states to be generated. It generates all (combinatorially) possible tuples of values and then eliminates inconsistent ones. Employing more sophisticated constraint satisfaction techniques should yield considerable improvements.

5.2.0.0.4 Analysis The 4 blobs problem has a surprisingly large number of states and behaviors. It is for this reason that the problem is computationally challenging for qualitative reasoning engines. The reason is that the model is extremely general and underconstrained. Neither the conductivity of contact nor thermal mass is described in the model. Thus, behaviors need to be generated consistent with any possible conductivity (which may even be changing).

The 4-blobs problem was computationally very difficult for both GIZMO and QPE. The Table 5.1 contrasts QPC with the results reported in [Forbus, 1990].

The times reported by Forbus are for a Symbolics 3670 with 24MB RAM and 200MB virtual memory. Those for QPC are for a SPARCstation 10, which may be several times faster.

The runtime for GIZMO was 60 times longer than QPC. Even allowing a factor of 10 for the machine difference, QPC would still be 6 times faster at generating an attainable envisionment. In principle, it should be possible to use GIZMO to generate a total envisionment by posing the problem without initial conditions. To my knowledge, this was never tried, and would have taken many hours or exhausted available memory.

QPE is not able to generate an attainable envisionment. It is specifically designed for performing total envisionments.

> *[the four blobs problem] had run for hours and generated over 5200 situations before the machine was halted manually.[Forbus, 1990, page 233]*

Note, however, that QPE and, to a lesser extent, GIZMO allow the granularity of situations to be controlled by the user. We use the closest (most fine grained) approximation to that of a QSIM state in this comparison. QPE is capable of generating a coarser total envisionment of this problem in 800 seconds. In this coarser mode, assumptions are made about the combinations of influences.

The situations generated by QPE and GIZMO differ from the states generated by QPC. A state is a simple and well defined object. It has a QDE, a set of governing equations, and an assignment of qualitative values (magnitude and qualitative direction pairs) to each variable in the QDE. A state may represent either a time point or an open interval between two time points. A behavior is a sequence of states covering time from an initial to a final point. A behavior precisely specifies a qualitative trajectory satisfying the QDE.

A situation is closer to a database as defined in Chapter 3. It consists of a set of relations. In particular, the relations characterize a set of active processes and views (model fragments) along with *assumptions* about quantity relations. Thus, one can view a situation as having a set of governing equations along with a partial specification of qualitative values for variables. A situation corresponds to a set of states. There is a link between two situations in an envisionment if there could be a link between a state in the first and a state in the second. Since the states are never generated explicitly, however, the converse relation is not guaranteed. That is, there may be links in the envisionment graph even if there can be no link between the corresponding states. An example of this is what Forbus called stuttering, in which situations representing time points are successors of each other.

A situation is a more flexible representation than a state. It can be readily used to represent abstractions that states cannot. A sequence of situations, however, does not map readily to a sequence of states or a solution to the set of governing equations. This makes a mathematical analysis of envisionments of situations considerably more difficult that ones composed of states.

5.3 Botany

The Botany Knowledge Base project at the University of Texas is building a tutoring system for college-level botany [Porter *et al.*, 1988]. They are particularly interested in supporting "what if" questions, in which the student poses a hypothetical scenario (e.g., soil moisture decreasing) and asks for the effect on particular plant properties (e.g., growth rate). In this section, we report on Jeff Rickel's work to automatically construct a simplified description of the scenario appropriate for the question, use QPC to predict the possible behaviors for the scenario, and use the model and simulation results to explain what happens and why [Rickel and Porter, 1992].[1]

Qualitative simulation is appropriate for this task for several reasons. First, the textbook knowledge encoded in the Botany Knowledge Base is largely qualitative. Botany textbooks present mostly qualitative relations among plant processes and properties because quantitative details vary greatly among different plants and because these quantitative details often are not known. Second, student questions are unlikely to provide the quantitative details needed to support a numeric simulation. Finally, college-level botany focuses primarily on the qualitative mechanisms of the plant, so quantitative details are unnecessary.

The following examples were constructed to experiment with the use of QPC in the botany tutor. The domain theory pales in comparison to the scope of knowledge in the actual Botany Knowledge Base (BKB) [Porter *et al.*, 1988], but it is representative of the relevant knowledge in the BKB concerning these important plant mechanisms. The domain theory covers the uptake, transport, and loss of water in the plant; the plant's mechanisms for dealing with water stress (synthesis, consumption and transport of the ABA hormone, and movement of potassium ions and water into and out of guard cells that regulate water loss); and the production (via photosynthesis), consumption (via respiration), and transport (via the phloem) of carbohydrate in the plant. The descriptions of these processes are relatively simple, but they are fairly standard and reasonable high-level descriptions, and they are based both on textbook knowledge and consultations with domain experts. In addition to views containing influences, a number of the views provide reasonable range constraints on variables to prevent impossible or unlikely behaviors from being considered.

Several elements of the domain theory are described at multiple levels of detail. `Acc-gc-osmosis-tsa` represents a time-scale abstraction [Kuipers, 1987; Kuipers, 1988] of the influences in `acc-gc-osmosis`; that is, a given amount of potassium in the guard cells will determine an equilibrium level of water in the guard cells, and this can be modeled as a functional relationship from the perspective of a sufficiently slow time scale (e.g., minutes). `Water-balance` is a simplification of `water-balance-details` in which water uptake is viewed as transporting water directly from the soil to the xylem, abstracting out the intermediate compartments. `Water-balance-simple` is a further simplification, treating the plant as one big compartment without further decomposition. `Transpiration-regulation-simple` is a simplified version of `transpiration-regulation-detailed` in which the details underly-

[1] I would like to thank Jeff Rickel for contributing the original text for this section, which has been lightly edited to fit into the surrounding chapter.

ing the functional dependence of the stomatal aperture on the pressure in the leaf symplast are omitted. Finally, `transpiration-regulation-detailed` includes time-scale abstractions of `ABA-regulation`, `K+-regulation`, and `acc-gc-osmosis`.

The example questions and the domain theory provide only an early snapshot of ongoing work. The current focus is in selecting relevant subsets of plant influences in order to answer a question. In this example, we are not concerned with qualifications on the applicability of model fragments, so each view requires only a plant rather than a more detailed and accurate set of structural and operating conditions. Although some views could be represented as different instantiations of a single, more abstract view (e.g., `acc-gc-osmosis` is simply an instance of an osmosis process in which the source is the accessory cells and the destination is the guard cells), we are not currently concerned with these commonalities. Nevertheless, each combination of question and model provides an example to guide work in automated modeling [Rickel and Porter, 1992; Rickel, 1992]. Coupled with QPC's behavior predictions, these questions and models provide example input for work in explanation [Mallory, 1993]. Finally, these examples help demonstrate that QPC will provide an adequate bridge between the modeling and explanation programs.

Relatively few compromises were made in encoding the domain theory and scenarios. In order to encode the behavior of exogenous variables which increase or decrease to a new equilibrium value, a `clock` variable and an `event` landmark was introduced, so that variables could be specified as increasing/decreasing until the event, at which time they become constant. Also, QPC was sometimes unable to infer algebraic consequences of scenario conditions, requiring them to be added explicitly. Finally, in order to ignore obtain tractable simulation in some examples, it was necessary to ignore the qdirs of certain variables.

The simulation results were far better than expected. Figure 5.29 summarizes the current set of examples. Each of the dozen current examples includes some chattering variables, but using Clancy's chatter elimination code [Clancy and Kuipers, 1993], which is available as a QSIM option, all but two of the examples generate a single reasonable behavior. One of the remaining examples was made to generate two reasonable behaviors by ignoring the qdirs of the chattering variables. The final example, `muench-perturbation`, which is the most complex example, falls prey to intractable branching, and has not yet been tamed.

The simplest example is `gc-osmosis`, which addresses the question

> *Why do the stomates open when the level of potassium ions increases in the guard cells?*

The relevant influences for this question are contained in `acc-gc-osmosis`. The aperture of the stomates is a function of the amount of water in the guard cells; a stomate is the opening between a pair of guard cells, and these cells become banana-shaped (thus enlarging the stomate) when they take up water. The water in the guard cells is influenced by the process of osmosis, which transfers water between the guard cells and adjacent accessory cells. The rate of this osmosis process is influenced by the amount of solutes in the guard cells, and potassium ions are the primary solute used by the plant to regulate osmosis. If osmosis is in equilibrium (i.e., no transport) and the level of potassium ions is increased to a new

1. *Why do the stomates open when K+ increases in the guard cells?*
2. *Why does K+ leave the guard cells when ABA there increases?*
3. *Why does ABA build up in the guard cells when turgor pressure drops in the leaves?*
4. *Why do the stomates close when turgor pressure drops in the leaves?*
5. *What happens to turgor pressure in the leaves as the soil dries out?*
6. *What happens to turgor pressure in the leaves as plant temperature increases?*
7. *What happens to the amount of sugars in the roots when photosynthetic production of sugars exceeds respiratory consumption in the leaves?*

Figure 5.29: List of botany questions that have been solved by QPC. Depending on time scale assumptions, a single question may correspond to several QPC scenarios.

equilibrium value, water begins to osmotically enter the guard cells, and this increase in water there enlarges the stomates. When the increase in potassium levels off, the increasing pressure in the guard cells due to the increase in water eventually balances the influence of the increased potassium level, so osmosis stops and the system settles into a new equilibrium state. With Clancy's chatter elimination program, QPC predicts the single correct behavior (without chatter elimination, there are seven possible behaviors due to uninteresting chatter of several variables). Figure 5.30 shows QPC's output for this problem.

STOMATES-K+-QUESTION
Plotting signs of: plant.stomatal-opening.

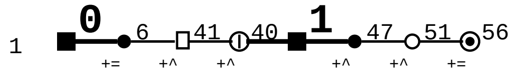

Figure 5.30: QPC output for the osmosis example. The amount of potassium in the guard cells increases up to a new equilibrium point (state 40/47). As the potassium increases, the stomates begin to open. Even after the potassium in the guard cells has become stable, the stomates continue opening until reaching a higher equilibrium (state 56).

Rickel's work required us to develop a technique for representing time varying exogenous variables. This is accomplished by introducing a `clock` and series of `event` landmarks. Model fragments trigger on the value of `clock`, whose consequences control the variable in question.

5.4 Chemical Engineering

QPC is being used to support the automatic generation of qualitative models of chemical plants. This work is being pursued in the dissertation research of Cathy Catino [Catino, 1993] in cooperation with Lyle Ungar at the University of Pennsylvania Chemical Engineering Department. Catino and Ungar are particularly interested in the task of hazard and operations analysis (HAZOP), and in the prerequisite task of building up a realistic library of model fragments for the chemical engineering domain.[2]

QPC is an appropriate analysis tool for this task and domain for several reasons. The phenomenon based compositional approach naturally describes chemical processes, which are not component-oriented, but occur whenever the requisite substances are present and conditions hold. The HAZOP task requires a sound model builder, because the task explicitly requires examining all possible outcomes to determine if undesirable ones may occur. In addition to unexpected reactions, this may also include unexpected changes in the topology of the plant (e.g. a valve incorrectly opened, or a leak in a heat exchanger). Although approximate quantitative information may be available for describing the normal operation of a chemical plant, the HAZOP analysis must consider what will occur when such constraints are violated. Qualitative simulation provides this ability.

Catino has constructed a substantial domain library, and used it in two case studies: a condenser, and a nitric acid plant for hazop analysis. The domain library contains definitions of mass flow, heat flow, chemical reaction, vaporization, and liquid-vapor equilibrium. It is capable of modeling controllers, phases consisting of any number of chemical species, thermal and mass fraction effects of mass flow, and moving landmark values as in a changing dew point. Chapter 4 of Catino's dissertation describes the domain theory and many of the design decisions that were made when constructing it.

The first case study is a dynamic, constant pressure condenser, in which a gas phase consisting of air and water enters the tube-side while a liquid water phase exists in the shell-side. Figure 5.31 shows a schematic for the condensor. Heat is transferred from the gas phase to the liquid phase in the shell. Once the temperature of the gas reaches its dew point, a new liquid water phase is produced in the tube and a gas/liquid mixture exits the tube-side.

In a binary system such as this one, as the mass fraction of the heavy component decreases, the dew point temperature also decreases. Previous qualitative models describing condensation ignored this relationship and held the dew point constant [Collins and Forbus, 1990]. In these models, the dew point affected the temperature of the container while heat transfer affected enthalpy (which did not affect temperature of the container when liquid-vapor equilibrium existed). During condensation, the temperature of the vessel remained constant at the original dew point of the gas.

Instead of forcing the dew point to take on one constant value, Catino's domain theory allows the dew point to change during simulation. A model fragment for liquid-vapor equilibrium maintains the dew point of the gas equal to the temperature of the container.

[2] I would like to thank Cathy Catino for contributing much of the text for this section.

Figure 5.31: Condenser in which a gas phase consisting of air and water enters the tube-side while a liquid water phase exists in the shell-side. A gas/liquid mixture exits the tube-side.

As condensation occurs, the mass fraction of the heavy component, water, decreases in the gas. This causes a decrease in the dew point of the gas while heat transfer causes decrease in the temperature of the tube-side. The decreased temperature of the container remains at the decreased dew point so that condensation continues to occur.

In this example, two different focusing methods were used to concentrate attention on the processes occurring in the tube-side. First, condensation is only allowed to occur in `focus-vessels`. Second, influences on the derivative of temperature of the shell-side are `ignored`. The ability to ignore influences is an important focusing technique provided by QPC. See Section 5.4 for a discussion.

The model is build using a plug flow perspective, which means that a piece of material is followed as it travels through the condenser. This is achieved by including a distance parameter that is increasing with time.

The second case study is a steady state model of the reactor section of a nitric acid plant. This model illustrates the fact that QPC can be used to build and solve large models of complex chemical processes. In general, previous approaches to modeling chemical facilities solved process unit models individually, using the output from one simulation as the input to the next [Grantham and Ungar, 1990]. This case study indicates that instead of building and simulating individual process unit models separately, a qualitative model of the entire plant can be built at once, and then this large model can be solved to describe the behavior of the whole plant.

A schematic diagram of the nitric acid plant is provided in Figure 5.32. Air flows through the air filter where impurities are removed. Liquid ammonia is pumped to the evaporator where it is vaporized. Energy is transferred to the evaporator from the steam-unit which is modeled as a heat source. A level controller maintains the level of liquid in the evaporator at a given setpoint. Compressed air and gaseous ammonia flow into the reactor at a ration specified by the ratio controller. the reaction $NH_3 + \frac{5}{4}0_2 \rightarrow \frac{3}{2}H_20 + NO$ occurs at constant pressure specified by a controller, and all four of these components flow into the product tank, a sink included so that steady state can be achieved (there is no accumulation of mass in the system). Influences are also ignored on the derivative of mass of solid and mass fraction of particles in the solid phase in the air filter. This is a kind of time scale abstraction; the accumulation of particles in the filter is considered to be negligible while the plant is operating at steady state.

The scenario definition specifies 41 initial entities in addition to various landmarks, initial conditions, and modeling assumptions. The model built by QPC consists of 81 active model fragments, 317 variables, and 387 constraints. It currently takes about an hour and a half to build and solve. A single steady state is generated describing how the plant operates when it is functioning as designed. This model is one of the primary inputs to the hazard identification system.

Catino's work required us to develop techniques for ignoring influences. This was driven by the desire to model sources and sinks in a way that is faithful to our informal understanding of them. In the following we refer to sources, but the same points apply to sinks as well. There are two distinct types of sources: those that are (approximately) inexhaustible

Figure 5.32: Nitric acid plant, reactor section basic instrumentation reproduced from [Coulson and Richardson, 1983]. This plant has been modeled and simulated with QPC [Catino, 1993].

supplies, those that are modeling artifacts representing a system boundary. The first is correctly modeled by ignoring the influences of outflows on the mass of the contents. The second is correctly modeled by a replenishing flow that matches whatever is taken out.

Chapter 6

Related Work

Constructing mathematical models of phenomena is perhaps the primary goal of the hard sciences. Assembling these models to predict the behavior of physical systems is the primary methodology of the engineering disciplines. Thus, it is with considerable trepidation and humility that a computer scientist ventures near these oceanic domains. It is my belief, however, that the automatic construction and simulation of models has much to offer. Providing formal description languages, reliable algorithms, and implemented tools to accomplish this is a suitable task for a computer scientist.

Research and experience in modeling and simulation is vast and profound. It would be beyond both the scope of this dissertation and the abilities of the author to adequately describe all that would be relevant. Consequently, this chapter focuses on approaches to modeling and simulation that are closely related to QPC.

6.1 Component and Connection Modeling

There is considerable practical and commercial experience with tools for building models of a system in terms of its components and their connections [Abelson and Sussman, 1985; Steele, 1980; Sussman and Steele, 1980; de Kleer and Brown, 1984; SPICE,]. The component and connection paradigm decomposes a system into components which interact with each other only through predefined connections among terminals. Each component is described by a set of equations. Each connection specifies equalities among terminal variables and allows variables to be shared across components.

This approach is very powerful for describing systems which have a well defined decomposition into (relatively) independent components with limited points of interaction. Well designed digital circuits that are not subject to cross-talk or other interference phenomena provide an excellent example of such systems. In fact, many devices that are designed to be easily understood, robust, easy to manufacture and maintain come close to this ideal.

The problem with the component and connection approach is that it tends to break down in the situations when modeling is most important. Situations in which the expectations about interactions are violated and unexpected phenomena arise are an important case. For example, a bridge fault in a circuit occurs when a blob of solder introduces a new connection between two logically distant, but physically close components [Davis, 1984]. This bridge is not part of the interface of any component, nor are its effects captured by the equations that model them. A leak between the shell and tube side of a heat exchanger introduces a new chemical into the product stream. The unexpected chemical, its byproducts, and subsequent reactions are not captured by the equations that model the downstream vessels.

Component and connection models are often hard to apply to systems that are not carefully designed to be decomposable. Very cleverly designed systems that use parts to serve multiple functions are hard to decompose. Biological or organic systems often exhibit this property.

Generating models and predicting behavior of a component and connection system is considerably easier than what QPC does. In fact, it is straight forward to use QPC as a component and connection modeling system.

Figure 6.1 describes a simple component and connection model in QPC. There are two distinct classes of objects: terminals and components. A *terminal* is an object that may have any number of variables associated with it. Variables are shared among components by sharing terminals. A *component* specifies a *type*, a *model fragment*, and an *interface*. The interface associates an instance of a component's type with its terminals, and is used as a condition for the model fragment. For example, a container is defined by the type `containers`, the model fragment `container-mf`, and the `interface` to a container comprising its input and output terminals. The model fragment for the container employs variables of the terminals (`(flow in)` and `(pressure out)`), as well as ones belonging to the component itself (`(amount c)`).[1] As is typically the case for component and connection models, the model fragments in this example include constraints, but no influences. The descriptions of components in this simple approach may be hierarchical as long as the interface does not change as the hierarchy is ascended.

More sophisticated component and connection approaches often associate several terminals with a *connection* [Franke and Dvorak, 1990]. This allows additional inferences to be made. For instance, Kirchhoff's current law (or its analog) may be applied at the connections [Rosenberg and Karnopp, 1983]. In the hydraulic domain, this implies that the sum of all flows into and out of a connection must be zero. Such a global inference is easily realized in QPC by a pair of model fragments such as those in Figure 6.3.

The component and connection paradigm is an important approach to describing physical systems so that models of them can be automatically built and simulated. Typical systems implementing this approach do not follow a system across multiple operating regions (sophisticated systems such as CC [Franke and Dvorak, 1990] do support multiple operating regions, but the transitions to be specified by hand), nor do they allow for composable equations beyond summation at connections. More importantly, the component and connection approach assumes

- a completely specified fixed set of components in a device, and

[1] The variable `(flow c)` is the flow through the container.

```
(deftaxonomy (objects
              (terminals)
              (components
               (containers)
               (flow-pumps))))
(defrelation interface (scopes components nil)
  :type inherited-relations)
(defQuantity pressure components)
(defQuantity flow components)
(defQuantity amount containers)

(defModelFragment container-mf
  :participants ((c  :type containers)
                 (in  :type terminals)
                 (out :type terminals))
  :structural-conditions ((interface c in out))
  :consequences
  ((add  (flow c)   (flow out) (flow in))
   (d/dt (amount c) (flow c))
   (m+ (amount c)    (pressure c))
   (q= (pressure c)  (pressure out))
   (m+ (pressure out)(flow out))
   (correspondence (amount c)     zero (pressure c) zero)
   (correspondence (pressure out) zero (flow out)   zero)))

(defModelFragment constant-flow-mf
  :participants ((p   :type flow-pumps)
                  (out :type terminals))
  :structural-conditions ((interface p out))
  :consequences
  ((constant (flow out))
   (> (flow out) zero)))
```

Figure 6.1: Simple component and connection description represented in the QPC modeling language. The `interface` relation accepts 3 or more arguments.

```
(defScenario cc
    :entities ((pump :type flow-pumps)
               (t1   :type terminals)
               (can  :type containers)
               (t2   :type terminals))
    :relations ((interface pump t1)
                (interface can t1 t2))
    :initial-conditions
    ((= (amount can) zero)))
```

Figure 6.2: Scenario definition for component and connection description represented in the QPC modeling language. The contents of this figure and Figure 6.1 run to produce a single model in which the can fills to equilibrium.

```
(defModelFragment connector
  :participants ((c :type connections))
  :consequences ((= (flow c) zero)))

(defModelFragment terminal
  :participants ((c :type connections)
                 (t1 :type terminals))
  :structural-conditions ((connect c t1))
  :consequences ((q-add (flow c) (flow t1))))
```

Figure 6.3: Model fragments for enforcing Kirchhoff's current law at connections.

- a fixed interface for each component.

6.2 Qualitative Process Theory and Its Implementations

QPC builds directly on the language of Qualitative Process theory [Forbus, 1984a]. QP theory has been one of the most influential and important pieces of work in the Qualitative Reasoning community. The QP theory approach to specifying and automatically constructing qualitative models of dynamic systems introduced several important ideas.

1. Phenomena are described in terms of abstract processes and views.
2. Process and views have explicit conditions that delineate when they are relevant and active.
3. Processes and views may contribute to composable equations.
4. The model of a system is composed from the equations contributed by active processes and views.
5. The range of real valued variables is described by a finite quantity space.

QP theory also makes the intriguing hypothesis that processes and views are an appropriate set of ontological primitives to generate both predictions and plausible cognitive explanations of behavior. QP theory makes strong assertions about causality in the world. Processes are the sole agents of change. Direct influences (`I+` and `I-`) which control the derivatives of variables may only be asserted by processes. All influences and other functional constraints are causally directed and the graph of causal relations imposed by indirect influences must be acyclic.

Aside from their names, the only difference between processes and views is that `I+` and `I-` influences may not appear in views. QPC merges views and processes into the slightly more general *model fragments*.

QPC remains agnostic on the issues of ontology and causality. There is not a distinguished set of processes. This does not, however, prevent a QPC domain theory from making such a distinction by dividing the set of model fragments into disjoint sets of processes and views. In QPC, influences remain causally directed. Other constraints, however, may be acausal. Thus, the QPC causal graph may be cyclic, and need not impose the strict ordering on variables that is to be found in QP theory.

The most important distinction between QP theory and QPC is that the semantics of QPC is grounded in ordinary differential equations. This grounding is only possible in light of the substantial advances in understanding the mathematics of the qualitative representation that resulted from the careful work of Kuipers [Kuipers, 1985; Kuipers, 1986; Kuipers and Chiu, 1987], Struß[Struß, 1988a; Struß, 1988b; Struß, 1990], and Wellman [Wellman, 1991].

There are three attributes of QP theory that prevent its semantics from being readily understood in terms of ordinary differential equations.

1. Time is not dense. It his thesis, Forbus explicitly states that "time is not dense" [Forbus, 1984a, page 14]. This is manifested in the problem of *stutter*, which is a sequence of situations, each of which represents an instant of time. Stutter is a consequence of the *equality change law* which states:

   > *With two exceptions, a state lasts for an interval of time. It lasts for an instant only when either*
   >
   > *(a) A change from equality occurs or*
   >
   > *(b) A change to equality occurs between quantities that were influenced away from equality for only an instant. [Forbus, 1984a, page 40]*

2. There is an explicit representation of quantities that differ only by an infinitesimal. This is also a consequence of the equality change law. "If the situation arose ...satisfying case 1 of the equality change law, then the situation includes the additional assertion that one number is 'starts infinitesimally greater than' [sic] the other" [Forbus, 1984a, page 127].

3. The continuity of variables is only defined qualitatively. For example, the *sign* of a derivative may change from `inc` to `std`, but not to `dec`. It is not, however, clear if the actual value must also change continuously. That is, even if a derivative is `inc` in a pair of successive situations, it may still have changed abruptly (e.g. from $50\frac{m}{s}$ to $10\frac{m}{s}$). Forcing all variables and derivatives to be strictly continuous is a rather strong restriction, because it would apply to changes in the model as well as changes in quantities. Allowing them to change discontinuously, would make it impossible to characterize a behavior in terms of reasonable functions. Neither choice is completely satisfactory.

In summary, the non-standard model of time and explicit use of infinitesimals prevents QP theory from having its semantics grounded directly in the theory of ordinary differential equations.

QPC resolves these issues so that its semantics may be expressed in terms of ODEs and their solutions. Time is dense. There is no need to reason about infinitesimals, because every model fragment holds at its boundaries. Variables are explicitly defined in terms of continuously differentiable functions (see Chapter 2). This means that the behaviors predicted by QPC are mathematically sound and do not require non-standard analysis to interpret them.

6.2.1 GIZMO

GIZMO, described in [Forbus, 1984a], was the first implementation of QP theory. GIZMO generates an attainable envisionment from a domain theory and a partial specification of an initial situation. At a high level, GIZMO's algorithm has three steps.

1. Elaborate the scenario.

2. Construct the initial situations and place them on the agenda.

3. Until the agenda is empty, do the following. Remove a situation S from the agenda. generate its successors. If a situation equivalent to a successor has already been generated, then discard the successor and link S to the equivalent situation. Otherwise, place the successor on the agenda and link S to it.

The output is a graph of situations rooted in the situations consistent with the initial conditions. A situation is a set of entities, instances, and quantity relations each of which is assigned a status of true or false (exists/not exists for entities).

The *scenario elaboration* is a structure containing all possible entities and instances. Entities and instances are created and added to the elaboration regardless of the initial conditions, logical or mathematical constraints imposed by the model fragment definitions. Whenever entities in the elaboration match the type signature of a model fragment (or entity) definition, an instance (or entity) is added to the elaboration. The structural and operating conditions are ignored during this process. Consequently, the elaboration will contain *at least* every entity and instance that would be present in the total envisionment of the system. It may even include objects that cannot occur in the total envisionment because their preconditions are never jointly satisfied. The up front cost of elaborating the scenario may be extremely high. Indeed, in the case of an unboundedly creative system, the cost is infinite.

The *initial situations* are created by finding all possible completions of the given partial specification. For a situation to be complete, it must assign a status to every entity, instance, quantity condition, and influenced variable in the elaboration. This is likely to include a huge number of assignments that are completely irrelevant to the current situation.

The central operation in *successor generation* is *limit analysis*. Limit analysis uses qualitative continuity rules to determine the possible changes in a situation's inequalities. The scenario elaboration is used to determine all of the possible effects that a particular (perhaps conjunctive) change might cause. Each set of possible changes, called *quantity hypothesis*, implicitly defines a successor state. Because the scenario elaboration does not make use of either logical or mathematical constraints, the changes implied by a particular quantity hypothesis may turn out to be inconsistent. For this reason, the complete successors are generated and checked for consistency.

Although GIZMO incrementally constructs a situation in the attainable envisionment from its predecessor, it must first construct the scenario elaboration. The scenario elaboration contains *at least the union of every instance and entity that could occur in the total envisionment.*

QPC does not elaborate the scenario. QPC fully exploits the initial conditions and the preconditions of model fragment and entity definitions. The sets of instances and entities are incrementally constructed.

QPC decomposes the construction of initial situations into two phases. Model refinement effectively assigns a status to the instances; QSIM assigns values to the variables. Both of these operations are very focused, and apply only to the instances and variables that are relevant to the situation.

QPC also decomposes the task of finding a successor into two distinct subtasks. The successors of a QSIM state are readily generated by QSIM. The successors after a transition are generated by refinement. Each subtask is simpler than the corresponding task in GIZMO and examines only the relevant subset of quantities or conditions.

6.2.2 QPE

The QPE implementation of QP theory [Forbus, 1990] aims at efficiently generating a total envisionment of a scenario. It is conceptually much cleaner than GIZMO and appears to be much more efficient as well.

QPE is built on an ATMS [de Kleer, 1986a; de Kleer, 1986b] that allows it to efficiently compute the sets of assumptions that underlie a proposition and to check if a set of assumptions (or propositions) is consistent.

The basic idea is to identify choice sets of assumptions such that selecting one element from each set will entail a complete situation. The set of all such consistent selections will then map directly to the set of all complete situations. Once the situations have been constructed, then successor links are installed wherever they are consistent. Each path through the graph represents a possible behavior.

It would be possible to have one choice set for each instance, entity, inequality, and influenced variable (similar to GIZMO). This would, however, be incredibly inefficient (similar to GIZMO). The main cleverness behind QPE is in how the choice sets are defined, ordered, and searched.

QPE does a good job of constructing a total envisionment, and is an excellent tool when many different models are required to account for a system's behavior. In scenarios that have few models, but many states, QPC may even be more efficient for the total envisionment task (e.g. the four blobs discussed in Section 5.2). For problems requiring the construction of a total envisionment in which the ratio of models to states is high, the approach taken by QPE is likely to be more efficient that of QPC.

QPC differs from QPE in several important ways.

- QPE is an implementation of QP theory and inherits the problems of QP theory discussed above. In particular, its semantics is not grounded in the theory of ordinary differential equations and their solutions.

- QPE generates a *total envisionment* consisting of all possible states that a system could ever be in. It cannot exploit information about the initial configuration of a system. In contrast, QPC constructs an *attainable envisionment* starting from the initial configuration.

- QPE does not employ a hybrid architecture. The model builder is not distinct from the simulator. This makes it difficult to exploit advances in qualitative or symbolic mathematics.

6.2.3 Compositional Modeling

More recent work by Falkenhainer and Forbus [Falkenhainer and Forbus, 1991] has extended QPE to address the issue of selecting a subset of a scenario description and domain library that will answer a specific query. E.g. if the query is "Will the radio turn off when I remove the key?", there is no reason to build a detailed model of a car's running engine. They make use of explicit simplifying assumptions and knowledge of the hierarchical structure of a device to identify the smallest subsystem (e.g. the electrical subsystem of the car) that contains the variables of interest.

This work employs a modeling language that is much closer to the one used by QPC in which the distinct syntactic forms for processes and views are replaced by a single `defmodel` form. It is otherwise similar to QPE.

6.2.4 Simgen

Forbus and Falkenhainer have worked to incorporate numerical knowledge into QPE [Forbus and Falkenhainer, 1990]. Interestingly, the output of SIMGEN is much more like QPC than its predecessor's.

> *SIMGEN works like this. The qualitative domain model is used to produce a total envisionment for the physical system. Next the math-model library is used to construct a set of ordinary differential equations for each qualitatively distinct region of behavior identified in the envisionment. [Forbus and Falkenhainer, 1990, page 381]*
>
> *It is more practical to divide states into equivalence classes, grouping together those governed by a common set of equations [Forbus and Falkenhainer, 1990, page 383]*

SIMGEN uses QPE to generate a total envisionment. After doing this, the situations that are governed by the same equations (have the same active model fragments) are collected into sets. SIMGEN then constructs algebraic equations for each of these sets. At this point, the representation is very close to QPC's. The qualitative states are effectively grouped under their models. The algebraic equations are used to generate numerical simulation code. When provided with a precise set of initial values, SIMGEN finds states which are consistent with them and uses the simulation code indexed under these states to generate predictions.

There are two important limitations to this work. First, it uses a math-model library that must contain equations for every combination of influences that appear in the total envisionment. This severely restricts the compositionality of the domain theory. Combinations of instances must be anticipated, and appropriate equations generated for them. Second, SIMGEN does not make any substantial changes in the semantics of QP theory. For example, there may still be sequences of instantaneous states, each with a separate model. This need not be a problem for SIMGEN, because such a path will simply not be followed by numerical behavior. The presence of infinitesimals, however, may cause SIMGEN to produce incorrect results. The problem occurs when a model is a result of a move off of equality. SIMGEN must choose some specific initial value for the difference that is *close enough* to zero that the numerical simulation will produce the same qualitative trajectory as a closer value. For example, suppose that $x = 0$ in S_1, there is a transition to S_2 as x becomes positive, and that S_2 has two successors S_3 and S_4. Suppose that for $x > 10^{-6}$, the transition condition to S_3 is true, and for $x \leq 10^{-6}$, the condition for S_4 is true. SIMGEN must pick an initial value for x that is smaller than this critical value. Of course, the critical value may be arbitrarily small. Automatically choosing such a value is far from trivial. Choosing too large a value may cause SIMGEN to produce a simulation that is simply wrong. Note that a system like SIMGEN that used QPC's formalism would not have this problem, because simulation in the model following the transition would begin with $x = 0$.

6.3 DME

The DME system [Iwasaki and Low, 1991] is very closely related to QPC. It is also an incremental compositional modeling system employing a hybrid architecture that uses QSIM for simulation. There are, however, a number of important differences.

In DME, QSIM is used to generate successors of a state, rather than a complete simulation. This approach has two major drawbacks.

1. It is slower. DME must assert quantity conditions into its knowledge base and determine whether the current model still applies after each new state is created. QPC compiles a set of easily evaluated transition tests that are made by QSIM. The cost of compilation is linear in the number of active and inactive model fragments; the cost of evaluating each test is little more than comparing the equality of two symbols. DME must consider creating a new model for every state, whereas QPC only considers creating models for transition states.

2. It loses inferential power. Many of the more advanced qualitative simulation techniques apply not to a single state, but to a behavior (e.g. the non-intersection constraint [Lee and Kuipers, 1993]) or to a subgraph of states (e.g. behavioral aggregation [Clancy and Kuipers, 1993]). Because DME does not allow QSIM to generate a complete simulation, it does not readily benefit from these and other advances.

DME is not sound. Its soundness is compromised in two ways. It may violate its own closed world assumptions without recognizing and recovering from the contradiction. It uses an unsound heuristic method for handling discrete valued variables.

In addition to using QSIM to generate state successors, DME also uses it to perform inequality reasoning over constraints in order to reduce ambiguity in a model. For example, suppose that an active model fragment provides a constraint `(M+ x y)` with a correspondence at `(0 0)` and `x` is known to be greater than 0. This can be built into a small problem for QSIM, which can be used to propagate the known values (`x = ((0 inf) nil)`) through the QDE to infer that `y = ((0 inf) nil)`. DME may then infer `(> y 0)`. This is an excellent use of QSIM and may provide strengthened inference about inequalities, thus reducing ambiguity in a model. The initial version of QPC [Crawford *et al.*, 1990] also did this sort of propagation. Unfortunately, DME employs it in an unsound fashion. The problem arises when constraints from influences are used for propagation. A closed world assumption (CWA) must be made — that the complete set of influences on a variable are known — in order to construct a constraint from a set of influences. Propagation may imply quantity conditions that activate new instances; the new instances may assert influences that violate the CWA that was used to infer that they were active.

DME employs discrete valued variables that change discontinuously. This is an important extension to the modeling language that is essential for representing many systems. For example, a valve may have only two positions `open` or `closed`.[2] DME uses heuristic methods to determine the effects of changes in the value of a discontinuous variable. Using discrete variables in a model also invalidates the soundness proof of QSIM, which requires variables to be continuous and real valued. Incorporating discontinuity while retaining soundness is a challenging problem.

To be fair, soundness has not been an over riding priority for DME. Applications have focused more on explanation [Gautier and Gruber, 1993; Gruber and Gautier, 1993] and exploratory design [Iwasaki and Chandrasekaran, 1992]. In these tasks, generating a single plausible behavior may be more appropriate than generating a potentially large set of possible ones. DME is also much more interactive than QPC. A user is given the ability to evaluate ambiguities and select the possibilities that are most interesting.

DME is capable of generating a (self explanatory) numerical simulation in the style of SIMGEN from a completely specified initial state.

6.4 Davis's Axiomatization of QP Theory

While formalizing QPC, I found Davis's delightful work on axiomatizing QP theory in first order logic[Davis, 1990; Davis, 1992; Davis, 1991] to be quite inspirational. His goal has not been to construct an inference engine, but to provide a logical theory. In principle, it should be possible to compare the output of a program (such as GIZMO, QPE, or QPC) to valid inferences in the theory. Such a comparison, however, is only useful if intended semantics of the program are correctly modeled by the formalization. Davis's work presents a careful case study of how "theories and inferences like those in QP can be expressed and justified in formal logic" [Davis, 1992, page 178]. The axiomatization in [Davis, 1991] contains a thoughtful first order logic axiomatization of the mathematical properties of variables.

The primary restriction of Davis's approach is that is instantiated for a particular domain theory and scenario. There is no translation of model fragment definitions into axioms as

[2] This is quite different from a QPC variable with a quantity space `(open closed)`. The quantity space imposes an order on the values. The QPC variable is continuous and differentiable. It may take on values such `(open inc)` or `((open closed) std)`.

is done by QPC. The reason for this is that Davis wants to provide a simple monotonic theory. This requires axioms to explicitly constrain the sets of objects that satisfy them. For instance, in his simple domain theory the mass of a liquid is only decreased (negatively influenced) by boiling. In [Davis, 1992, page 281], this characteristic of the entire domain theory is represented by the axiom:

$$2.5\ \text{d\_influence}(P,\text{lmass}(O),S) = \text{neg} \Leftrightarrow P{=}\text{boiling}(O).$$

If another model fragment were to influence the mass of a liquid (e.g. flow out of a container), this axiom would become invalid. The axiom set does not have the property of *additivity*. That is, if a model fragment is added to the domain theory, large portions of the axiom set will need to be changed. His formalization also does not allow new entities to be created. Axioms specifying all possible entities are required. For example,

$$\text{SC.1 in\_scenario}(0) \Rightarrow [\ \text{heat\_reservoir}(O) \Leftrightarrow O = \text{oflame}]$$

states that `oflame` is the only heat reservoir in the scenario [Davis, 1992, page 12]. Again, this is to allow for a completely monotonic theory. QPC elegantly draws these domain closure inferences by identifying when a model is *complete*.

Axioms of real analysis are to be used as necessary when constructing proofs. This makes his approach more difficult to implement, as the set of such axioms that will be useful is undefined.

6.5 Other Work

In addition to the closely related work describe above, there is another cluster of research on automated modeling. The focus of the research is to identify the model that best serves to answer a particular query. One important aspect of this work is to identify knowledge about the structure of systems, model fragments, and their inter-relations that aid in building the best model. QPC provides a modeling language and sound prediction capability that could be used by researchers in this area. QPC's modeling language is expressive enough for them to represent knowledge about model fragments and the structure of systems. QPC's predictions provide useful answers for many queries and the representation of models in the knowledge base provides information to answer many more. In short, QPC provides a substrate to support research on selecting the best model of a sytem.

Falkenhainer and Forbus [Falkenhainer and Forbus, 1991], Rickel and Porter [Rickel and Porter, 1992], Williams [Williams, 1989; Williams, 1991], and Nayak [Nayak, 1992] have been making important contributions to the problem of selecting the right set of model fragments to answer a particular query. QPC hopes to exploit their results wherever they are applicable (and Rickel is using QPC as his modeling tool). These approaches all start with a description of a device, a domain theory of model fragments, and a set of variables whose interrelations or behavior must be explained or predicted.

Falkenhainer and Forbus [Falkenhainer and Forbus, 1991] use information about the hierarchical structure of a device to identify the smallest subsystem that covers the variables of interest. The criteria for hierarchical decomposition are strong and do not apply to all devices. They also make excellent use of explicit modeling assumptions (e.g. that flow is inviscid, viscid, or turbulent) to select the simplest model fragments for a phenomena (e.g. fluid flow) that capture the behavior of interest. They search for the simplest model that covers the relevant variables. An important restriction on this work is that requires a scenario elaboration to define the space of possible models.

Rickel [Rickel and Porter, 1992] is extending the work of Falkenhainer and Forbus to exploit time scale information in addition to explicit modeling assumptions. He also exploits interaction paths [Williams, 1991] to identify relevant portions of the elaborated scenario. This provides greater sensitivity to the query and allows him to build models of systems without a rigid hierarchical structure. Such systems are ubiquitous in the botany domain in which boundaries are often blurred.

Williams [Williams, 1991] constructs a parsimonious description of a device from a complex one similar to a scenario elaboration. He uses influences to define interaction paths relating variables. The interaction paths guide the search for the relevant subset of equations. The notion of *critical abstractions* is used to build a more parsimonious model. The equations apply over a single operating region. From QPC's point of view, an important contribution of this work is that it does not rely on simulation to predict behavior. When possible, a qualitative description of behavior is extracted from the equations analytically by his MINIMA system [Williams, 1988].

Nayak [Nayak, 1992] is also interested in the problem of building the simplest possible model that connects the variables of interest. His approach only describes a device over a single operating region. Each component in the system is described by a single model fragment. The difficulty is in selecting the correct model fragment for each component to produce the simplest possible model of the device. His domain theory explicitly represents relations between model fragments (including `contradictory`, `approximation`, and `required-assumptions`).

Another cluster of work also aims at finding the best (simplest) model of a device that covers a behavior or variables of interest. Addanki introduced the notion of a *graph of models* [Addanki *et al.*, 1989; Addanki *et al.*, 1991], in which many possible models of a device are explicitly represented in a graph. Each node in the graph is a model. An arc between two nodes is labeled with the assumptions that must be changed in order move from one model to another. The graph is used to guide the search for model that best captures the behavior of a device. The size of the graph is exponential in the number of assumptions (if the assumptions are independent). The compositional modeling approach of QPC allows the graph to be implicitly represented by individual model fragments; the number of model fragments is linear in the number of assumptions.

Weld [Weld, 1990] introduced the notion of *fitting approximations* that can be used with the graphs of models approach. If model `a` is a fitting approximation of model `b`, then the behavior predicted by `a` approaches that predicted by `b` as some fitting parameter approaches its limit. For example, frictionless motion is a fitting approximation of motion with friction. If the graph of models can be organized according to fitting approximations, then it can be searched much more efficiently. In many cases fitting approximations can be either automatically generated or proven.

Explicitly representing relations between model fragments is an important source of domain knowledge. This research can be exploited by the authors of QPC domain theories.

Chapter 7

Conclusion

This dissertation has presented a hybrid compositional modeling engine, QPC, that incrementally constructs predictions of the temporal evolution of a physical situation. QPC decomposes the problem of reasoning about a physical system into a *model building* task which builds a model to describe the system, and a *simulation* task which uses the model to generate a description of the possible behaviors of the system. This central insight leads naturally to a hybrid architecture that supports an incremental algorithm and proofs of correctness. The clear decomposition also allows QPC and qualitative reasoning generally to benefit from independent advances in model building and simulation.

The QPC algorithm is incremental in that it begins with a (partial) description of a system in an initial configuration and constructs models that incrementally predicts its behavior over time. This can be contrasted with the total envisionment approach of QPE or systems such as GIZMO that compose models incrementally from a set that comprises all possible phenomena and objects.

The semantics of QPC has been grounded in the solutions of ordinary differential equations and their solutions. The rigorous treatment would not have been possible without the clear separation between modeling and simulation, the mathematical clarity and rigor of the QSIM QDE representation, and the soundness proof of the QSIM algorithm [Kuipers, 1986].

The representation and algorithm have been shown to be correct. If the domain theory is adequate and the initial description of the scenario is correct, then the actual behavior of the system will be among those predicted by QPC.

QPC has been used to solve a suite of illustrative examples that that exercise many of its capabilities. QPC is being applied to the problems of chemical engineering hazard operations analysis and tutoring college level botany. The chemical engineering application involves substantial QDEs with over 250 variables and constraints employed to model a nitric acid processing plant. This provides important evidence for both the expressiveness of the modeling language and the inferential power of the algorithm.

7.1 Future work

7.1.1 Extensions

QPC, as developed in this dissertation, produces useful predictions of behavior in the presence of incomplete knowledge about a physical situation. One of the desiderata of qualitative reasoning, however, is to be able to exploit as much information as is available. Perhaps the most important improvement of QPC is to extend it to exploit *seminumerical information.* Seminumerical information spans from the extreme of a qualitative representation to the extreme of a precise algebraic and numerical one. Recent progress in seminumeric simulation retains the soundness properties of QSIM, while providing much stronger inference capabilities. In cooperation with Giorgio Brajnik at the University of Udine, QPC is currently being extended to exploit seminumeric information. This information takes the form of interval bounds on landmark values and the specification of envelopes for monotonic functions.[1] Because the semantics of QPC is already grounded in ODEs and their solutions, this extension requires no changes in the underlying theory. QPC's soundness is retained. All models and behaviors that are consistent with the numerical information will be generated.

Compositional modeling systems may play an important role in *generating numerical models and predictions* that are more reliable and "self explanatory" [Forbus and Falkenhainer, 1990]. They can be more reliable because the boundary conditions of the model are explicitly known. Furthermore, the structure of the model and the contents of the knowledge base to provide an explanation of the behavior. These attributes are important for engineering tasks requiring numerical simulation and analysis, as well as tutoring systems that may want to provide an animation of a single behavior. QPC is well suited to the task of automatically generating specifications for a numerical simulator. The QPC modeling language includes composable equations that combine algebraically (`q-add`,`q-sub`,`q-mult`,`q-div`). This allows domain theories that are designed for numerical simulation to retain a high degree of composability. Modulo the limited precision of floating point numbers, QPC should retain its soundness guarantees when provided with precise initial conditions and algebraic equations.[2] QPC's hybrid architecture readily incorporates a numerical simulator — even one generated from the current model — instead of QSIM. (See the discussion in Section 6.2.4 page 137.)

Discontinuous change is ubiquitous in our approximations of the world. Even if all change is really continuous, it is essential for a system such as QPC to be able to approximate rapid continuous changes with discontinuous ones. It is also important for domain theory author to be able to intuitively represent discontinuous phenomena. Retaining both

[1] An envelope for a monotonic function f is a pair of functions u and l that bound f from above and below. For example, for $x > 0$ the f might be bound by the equations $y = x$ and $y = x^2$. This not restrict f to be linear — any reasonable function lying between those bounds is abstracted by f.

[2] If multiple behaviors are allowed, it should be possible to retain soundness without this caveat. The problem arises when the precision of the simulator is not sufficient to determine an instance's operating conditions. E.g. if the condition is $x > 0$ and $x \approx 0$.

soundness and tractability while allowing for discontinuous changes is a challenging and exciting area of research.

It is also useful to provide *discrete valued variables* in the domain theory. There are several straightforward ways of incorporating them into QPC. Since QSIM does not make any use of them, however, there has been little motivation to do so.

7.1.2 Tasks

QPC is currently being employed as a prediction module in a botany tutoring system and a chemical engineering hazard/operations analysis system. Both of these tasks are well suited to QPC.

The task of monitoring a dynamic system is well suited to QPC. QPC could be used to construct a monitoring system, similar to MIMIC [Dvorak and Kuipers, 1989; Dvorak, 1992], that uses observations to guide its search through the space of possible models. Mismatches between observations and predictions would lead it to hypothesize the activity of new model fragments (and truth of their preconditions) or the existence of new entities. By organizing the domain theory around physical phenomena rather than known failure modes of components, this system could identify and monitor a broad range of failures.

7.1.3 Implementation

Although it may be faster than QPE or GIZMO, QPC is currently too slow and consumes more resources than are necessary. An implementation that rather closely follows the semantics of Chapter 3 will be considerably faster than the current one. I expect this to result in a 10-100 fold speedup. A second factor of 10-100 should also be possible by employing standard rule compilation techniques, improving indexing, and eliminating the assumption mechanism in Algernon. Thus, we can conservatively expect a 100 fold speedup in the next version of QPC. This means that the large chemical engineering models of Section 5.4 will be solved in under a minute, and the botany questions of Section 5.3 in fractions of a second. For many applications, this level of performance is quite acceptable.

7.2 Synergy

Several reserchers (including Forbus, Falkenhainer, Gruber, Iwasaki, and Kuipers, as well as the author) are currently cooperating on the design of a common modeling language that will support the exchange of domain theories. As the capabilities of automated modeling systems grow, the construction, sharing, and reuse of domain theories becomes more important. Libraries of realistic model fragments for domains such as fluid dynamics are extremely difficult to construct. Catino [Catino, 1993], Collins, and Forbus [Collins and Forbus, 1990] have made important contributions in this direction.

7.3 Final words

Model based reasoning and automatic modeling are coming of age. In the near future we should expect to see systems like QPC support reasoning about truly large complex dynamic systems with many hundreds of parameters and components. To be trusted and used, these tools need come with a precise characterization, grounded in mathematics, of what they do. We can benefit from reliable automated tools to support the tremendously difficult tasks of understanding, design, analysis, and monitoring of large complex systems.

Bibliography

[Abelson and Sussman, 1985] Abelson, H. and Sussman, G. J. 1985. *Structure and Interpretation of Computer Programs.* MIT Press/McGraw-Hill.

[Addanki *et al.*, 1989] Addanki, S.; Cremonini, R.; and Penberthy, J.S. 1989. Reasoning about assumptions in graphs of models. In *Proceedings of the Eleventh International Joint conference on Artificial intelligence.* 1432–1438.

[Addanki *et al.*, 1991] Addanki, S.; Cremonini, R.; and Penberthy, J.S. 1991. Graphs of models. *Artificial Intelligence* 51:145–177.

[Allen, 1983] Allen, J.F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26(11):832–843.

[Berleant and Kuipers, 1992] Berleant, Daniel and Kuipers, Benjamin 1992. Combined qualitative and numerical simulation with Q3. In Faltings, Boi and Struß, Peter, editors 1992, *Recent Advances in Qualitative Physics.* MIT Press.

[Catino, 1993] Catino, Cathy 1993. *Automated Modeling of Chemical Plants with Application to Hazard and Operability Studies.* Ph.D. Dissertation, Department of Chemical Engeneering, University of Pennsylvania, Philadelphia, PA.

[Clancy and Kuipers, 1992] Clancy, D. J. and Kuipers, B. J. 1992. Aggregating behaviors and tractable simulation. In *AAAI Design from Physical Principles Fall Symposium Working Notes*, Cambridge, MA. 38–43.

[Clancy and Kuipers, 1993] Clancy, D. J. and Kuipers, B. J. 1993. Behavior abstraction for tractable simulation. In *Proceedings of the Seventh International Workshop on Qualitative Reasoning about Physical Systems.*

[Collins and Forbus, 1990] Collins, J. W. and Forbus, K. D. 1990. Building qualitative models of thermodynamic processes. Technical report, Beckman Institute, University of Illinois.

[Coulson and Richardson, 1983] Coulson, J.M and Richardson, J.F. 1983. An introduction to chemical engineering design. In Sinnot, , editor 1983, *Chemical Engineering*, volume 6. Pergamon Press, New York.

[Crawford and Etherington, 1992] Crawford, J. M. and Etherington, D. 1992. Formalizing reasoning about change: A qualitative reasoning approach (preliminary report). In *Proceedings of the Tenth National Conference on Artificial Intelligence.*

[Crawford and Kuipers, 1991] Crawford, J. M. and Kuipers, B. J. 1991. ALL: formalizing acess-limited reasoning. In Sowa, John, editor 1991, *Principles of Semantic Networks.* Morgan Kaufman, San Mateo, CA. 299–330.

[Crawford *et al.*, 1990] Crawford, James; Farquhar, Adam; and Kuipers, Benjamin 1990. QPC: A compiler from physical models into qualitative differential equations. In *Proceedings of the Eighth National Conference on Artificial Intelligence.*

[Crawford, 1991a] Crawford, J. M. 1991a. Algernon — a tractable system for knowledge representation. *SIGART Bulletin* 2(3):35–44.

[Crawford, 1991b] Crawford, James 1991b. *Access-limited logic — A language for knowledge representation.* Ph.D. Dissertation, The University of Texas at Austin. Also available as Technical Report AI90-141, Artificial Intelligence Laboratory, The University of Texas at Austin.

[Davis, 1984] Davis, R. 1984. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence* 24:347 – 410.

[Davis, 1990] Davis, Ernest 1990. *Representations of commonsense knowledge.* Morgan Kaufmann Publishers, Inc.

[Davis, 1991] Davis, Ernest 1991. Axiomatizing qualitative process theory. Technical Report 590, NYU. Long version of the KR-92 paper.

[Davis, 1992] Davis, Ernest 1992. Axiomatizing qualitative process theory. In *Principles of knowledge representation and reasoning (KR92).* Morgan Kaufmann Publishers, Inc. 177–188.

[de Kleer and Brown, 1984] de Kleer, J. and Brown, J.S. 1984. A qualitative physics based on confluences. *Artificial Intelligence* 24:7–83. Also in *Readings in Knowledge Representation*, Brachman and Levesque, editors, Morgan Kaufmann, 1985, pages 88-126.

[de Kleer, 1986a] de Kleer, Johan 1986a. An assumption-based TMS. *Artificial Intelligence* 28:127–162.

[de Kleer, 1986b] de Kleer, Johan 1986b. Problem solving with the ATMS. *Artificial Intelligence* 28:197–224.

[Dvorak and Kuipers, 1989] Dvorak, Daniel and Kuipers, Benjamin 1989. Model-based monitoring of dynamic systems. In *Proceedings of the Eleventh International Joint conference on Artificial intelligence*, Los Altos, CA. Morgan Kaufman. 1238–1243.

[Dvorak, 1992] Dvorak, Daniel Louis 1992. Monitoring and diagnosis of continuous dynamic systems using semiquantitative simulation. Technical Report AI92–170, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, Texas 78712.

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

[Farquhar *et al.*, 1991] Farquhar, Adam; Kuipers, Benjamin; Rickel, Jeff; Throop, David; and Group, The Qualitative Reasoning 1991. QSIM: The program and its use. Technical Report AI90–123, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, Texas 78712.

[Forbus and Falkenhainer, 1990] Forbus, Kenneth D. and Falkenhainer, Brian 1990. Self-explanatory simulations: An integration of qualitative and quantitative knowledge. In *Proceedings of the Eighth National Conference on Artificial Intelligence.* 380–387.

[Forbus, 1984a] Forbus, K.D. 1984a. *Qualitative Process Theory.* Ph.D. Dissertation, MIT AI Lab, Cambridge, MA.

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

[Forbus, 1985] Forbus, Kenneth 1985. The problem of existence. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society.*

[Forbus, 1986] Forbus, Kenneth D. 1986. Interpretating meaurements of physical systems. In *Proceedings of the Fifth National Conference on Artificial Intelligence.* 113–117.

[Forbus, 1990] Forbus, Kenneth 1990. The qualitative process engine. In Weld, D. and de Kleer, J., editors 1990, *Readings in Qualitative Reasoning about Physical Systems.* Morgan Kaufman, Los Altos, CA.

[Fouché, 1992] Fouché, Pierre 1992. *Towards a Unified Framework for Qualitative Simulation.* Ph.D. Dissertation, Département Génie Informatique, Centre de Recherches de Royallieu Université de Technologie de Compiègne.

[Franke and Dvorak, 1990] Franke, David and Dvorak, Daniel 1990. CC: Component-connection models for qualitative simulation – a user's guide. Technical Report AI90–126, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, Texas 78712.

[Gautier and Gruber, 1993] Gautier, P. O. and Gruber, T. R. 1993. Generating explanations of device behavior using compositional modeling and causal ordering. In *Proceedings of the Eleventh National Conference on Artificial Intelligence.* AAAI Press/The MIT Press.

[Genesereth and Fikes, 1992] Genesereth, Michael and Fikes, Richard 1992. Knowledge interchange format, version 3.0 reference manual. Technical Report Report Logic-92-1, Stanford University.

[Genesereth and Nilsson, 1987] Genesereth, Michael R. and Nilsson, Nils J. 1987. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann Publishers, Inc.

[Grantham and Ungar, 1990] Grantham, and Ungar, Lyle 1990. A first principles approach to automated troubleshooting of chemical plants. *Computers and Chemical Engineering* 14(7):783–798.

[Gruber and Gautier, 1993] Gruber, T. R. and Gautier, P. O. 1993. Machine-generated explanations of engineering models: A compositional modeling approach. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence.*

[Hayes, 1985] Hayes, Patrick 1985. The second naive physics manifesto. In Hobbes, J.R. and Moore, E.C., editors 1985, *Formal Thoeries of the Commonsense World.* Ablex Publishing Corporation, Norwood, NJ. 1–36.

[Hirsch and Smale, 1974] Hirsch, Morris W. and Smale, Stephen 1974. *Differential Equations, Dynamical Systems, and Linear Algebra.* Academic Press, New York.

[Iwasaki and Chandrasekaran, 1992] Iwasaki, Y. and Chandrasekaran, B. 1992. Design verification through function and behavior-oriented representations: Bridging the gap between function and behavior. In *Proceedings of the Second International Conference on Artificial Intelligence in Design*, Pittsburgh. Kluwer Academic.

[Iwasaki and Low, 1991] Iwasaki, Yumi and Low, Chee Meng 1991. Model generation and simultion of device behavior with continuous and discrete changes. Technical Report KSL 91-69, Knowledge Systems Laboratory, Stanford University.

[Iwasaki and Simon, 1986a] Iwasaki, Y. and Simon, H. A. 1986a. Causality in device behavior. *Artificial Intelligence* 29:3–32.

[Iwasaki and Simon, 1986b] Iwasaki, Y. and Simon, H. A. 1986b. Theories of causal ordering: Reply to de kleer and brown. *Artificial Intelligence* 29:63 – 72.

[Iwasaki, 1988] Iwasaki, Y. 1988. Causal ordering in a mixed structure. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, San Mateo, CA. Morgan Kaufmann. 313–318.

[Kay and Kuipers, 1992] Kay, Herbert and Kuipers, Benjamin 1992. Numerical behavior envelopes for qualitative models. In *Working Papers from the Sixth International Workshop on Qualitative Reasoning*, Edinburgh, Scotland. 252–267.

[Kay and Kuipers, 1993] Kay, Herbert and Kuipers, Benjamin 1993. Numerical behavior envelopes for qualitative models. Submitted to AAAI-93.

[Kuipers and Chiu, 1987] Kuipers, B. and Chiu, C. 1987. Taming intractible branching in qualitative simulation. In *Proceedings of the Tenth International Joint conference on Artificial intelligence*, Los Altos, CA. Morgan Kaufman. Describes ignoring qdirs.

[Kuipers and Kassirer, 1983] Kuipers, B. J. and Kassirer, J. P. 1983. How to discover a knowledge representation for causal reasoning by studying an expert physician. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Los Altos, CA. Morgan Kaufman.

[Kuipers and Kassirer, 1984] Kuipers, B. J. and Kassirer, J. P. 1984. Causal reasoning in medicine: analysis of a protocol. *Cognitive Science* 8:363 – 385.

[Kuipers, 1985] Kuipers, B. J. 1985. The limits of qualitative simulation. In *Proceedings of the Ninth International Joint conference on Artificial intelligence*, Los Altos, CA. William Kaufman.

[Kuipers, 1986] Kuipers, B. 1986. Qualitative simulation. *Artificial Intelligence* 29:289–338.

[Kuipers, 1987] Kuipers, B. 1987. Abstraction by time scale in qualitative simulation. In *Proc. of AAAI-87*, Seattle, WA. 621–625.

[Kuipers, 1988] Kuipers, B. 1988. Qualitative simulation using time-scale abstraction. *Int. J. Artificial Intelligence in Engineering* 3(4):185–191.

[Kuipers, 1993a] Kuipers, Benjamin 1993a. Qualitative reasoning: Modeling and simulation with incomplete knowledge. Unpublished manuscript.

[Kuipers, 1993b] Kuipers, Benjamin 1993b. Reasoning with qualitative models. *Artificial Intelligence* 59:125–132.

[Lee and Kuipers, 1988] Lee, W. W. and Kuipers, B. 1988. Non-intersection of trajectories in qualitative phase space: a global constraint for qualitative simulation. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Los Altos, CA. Morgan Kaufman.

[Lee and Kuipers, 1993] Lee, Wood Wai and Kuipers, Benjamin 1993. A qualitative method to construct phase portraits. In *Proceedings of the Eleventh National Conference on Artificial Intelligence.*

[Lee, 1993] Lee, Wood Wai 1993. *A Qualitative Simulation Based Method to Construct Phase Portraits.* Ph.D. Dissertation, Department of Computer Sciences, University of Texas at Austin, Austin, Texas.

[Levesque and Brachman, 1987] Levesque, H.J. and Brachman, R.J. 1987. Expressiveness and tractablility in knowledge representation and reasoning. *Computational Intelligence* 3:78–93.

[Levy and Iwasaki, 1992] Levy, A.Y. and Iwasaki, Y. 1992. Relevance reasoning to guide compositional modeling. In *submitted to AAAI-92.*

[Mallory, 1993] Mallory, R. 1993. Generating structured, causal explanations of qualitative simulations: A research proposal. Ph.D. Thesis proposal. University of Texas at Austin.

[McCarthy and Hayes, 1969] McCarthy, J. and Hayes, P.J. 1969. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence* 4:463–502.

[Nayak, 1992] Nayak, P.P. 1992. *Automated Modeling of Physical Systems.* Ph.D. Dissertation, Department of Computer Science, Stanford University.

[Pearl, 1988] Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, Inc., San Mateo,CA.

[Porter *et al.*, 1988] Porter, B.; Lester, J.; Murray, K.; Pittman, K.; Souther, A.; Acker, L.; and Jones, T. 1988. AI research in the context of a multifunctional knowledge base: The botany knowledge base project. Technical Report Technical Report AI88-88, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, Texas 78712.

[Rickel and Porter, 1992] Rickel, J. and Porter, B. 1992. Automated modeling for answering prediction questions: Exploiting interaction paths. In *Proceedings of the Sixth International Workshop on Qualitative Reasoning*, Heriot-Watt University, Edinburgh, Scotland.

[Rickel, 1992] Rickel, J. 1992. Automated modeling for answering prediction questions: Exploiting interaction paths. Technical Report AI 92-178, Artificial Intelligence Laboratory, University of Texas, Austin, TX, 78712.

[Rosenberg and Karnopp, 1983] Rosenberg, Ronald C. and Karnopp, Dean C. 1983. *Introduction to Physical System Dynamics.* McGraw-Hill, New York. Basic reference for bond graphs.

[Sacks, 1990] Sacks, E. 1990. A dynamic systems perspective on qualitative simulation. *Artificial Intelligence* 42:349–362.

[Skorstad and Forbus, 1989] Skorstad, Gordon and Forbus, Ken 1989. Qualitative and quantitative reasoning about thermodynamics. In *Proceedings of the Third Internationsal Workshop on Qualitative Physics (QPW-89).*

[SPICE,] Spice. Re component-connection descriptions.

[Steele, 1980] Steele, G. 1980. *The Definition and Implementation of a Computer Programming Language Based on Constraints.* Ph.D. Dissertation, MIT, Cambridge, MA.

[Struß, 1988a] Struß, Peter 1988a. Global filters for qualitative behaviors. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Los Altos, CA. Morgan Kaufmann.

[Struß, 1988b] Struß, Peter 1988b. Mathematical aspects of qualitative reasoning. *Int. J. Artificial Intelligence in Engineering* 3(3):156–169.

[Struß, 1990] Struß, Peter 1990. Problems of interval-based qualitative reasoning. In Weld, Daniel and de Kleer, Johan, editors 1990, *Qualitative Reasoning about Physical Systems.* Morgan Kaufmann. 288–305.

[Sussman and Steele, 1980] Sussman, G. J. and Steele, G. L. 1980. Constraints: a language for expressing almost-hierarchical descriptions. *Artificial Intelligence* 14:1–39.

[Weld, 1990] Weld, D. S. 1990. Approximation reformulations. In *Proceedings of the Eighth National Conference on Artificial Intelligence*. 407–412.

[Wellman, 1991] Wellman, Mike 1991. Qualitative simulation with multivariate constraints. In Allen, J.; Fikes, R.; and Sandewall, E., editors 1991, *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference (KR-91)*, San Mateo, CA. Morgan Kaufmann.

[Williams, 1988] Williams, Brian C. 1988. Minima: A symbolic approach to qualitative algebraic reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Los Altos, CA. Morgan Kaufman. 264–269.

[Williams, 1989] Williams, Brian 1989. *Invention from First Principles via Topologies of Interactions*. Ph.D. Dissertation, MIT.

[Williams, 1991] Williams, B.C. 1991. Critical abstraction: Generating simplest models for causal explanation. In *Working Papers of the Fifth International Workshop on Qualitative Reasoning about Physical Systems*. 77–92.

[Zadeh, 1965] Zadeh, L. 1965. Fuzzy sets. *Information and Control* 8:338–353.