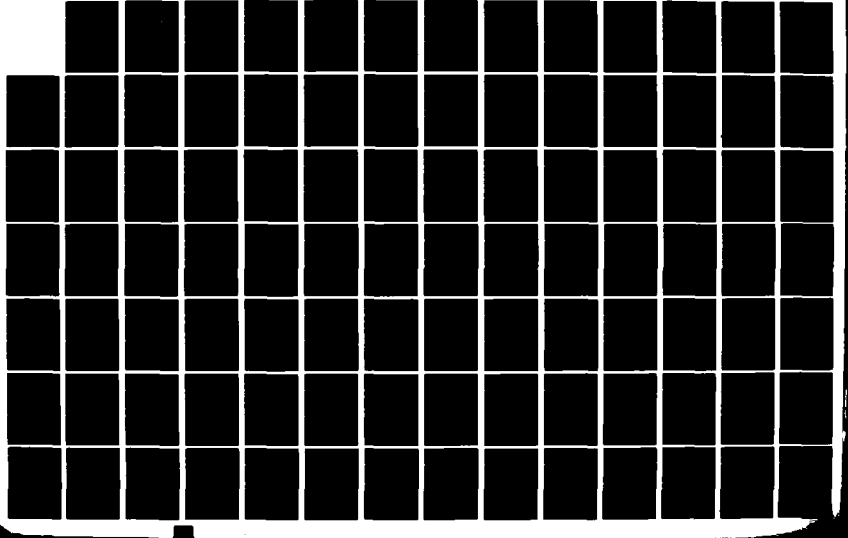
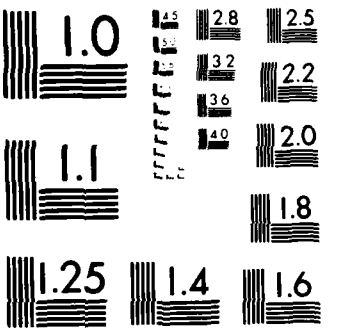
AD-A148 987 QUALITATIVE PROCESS THEORY(U) MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB K D FORBUS JUL 84 AI-TR-789 N00014-80-C-0505 1/2

UNCLASSIFIED F/G 9/4 NL





Technical Report 789

Qualitative Process Theory

Kenneth Dale Forbus

MIT Artificial Intelligence Laboratory

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AI-TR-789	2. GOVT ACCESSION NO. AD-A148 987	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Qualitative Process Theory		5. TYPE OF REPORT & PERIOD COVERED technical report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Kenneth Dale Forbus		8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0505
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, Massachusetts 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd Arlington, Virginia 22209		12. REPORT DATE July 1984 13. NUMBER OF PAGES 179
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Arlington, Virginia 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Distribution is unlimited

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

qualitative reasoning
common sense reasoning
naive physics
aritificial intelligence
problem solving
mathematical reasoning

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Objects move, collide, flow, bend, heat up, cool down, stretch, compress, and boil. These and other things that cause changes in objects over time are intuitively characterized as "processes". To understand common sense physical reasoning and make programs that interact with the physical world as well as people do we must understand qualitative reasoning about processes, when they will occur, their effects, and when they will stop. Qualitative Process theory defines simple notion of physical process that appears useful as a language

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

(over)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20 cont.

in which to write dynamical theories. Reasoning about processes also motivates a new qualitative representation for quantity in terms of inequaltites, called the "quantity space".

This report describes the basic concepts of Qualitative Process theory, several different kinds of reasoning that can be peformed with them, and discusses its impact on other issues in common sense reasoning about the physical world, such as causal reasoning and measurement interpretation. Several extended examples illustrate the utility of the theory, including figuring out that a boiler can blow up, that an oscillator with friction will eventually stop, and how to say that you can pull with a string, but not push with it. This report also describes GIZMO, an implemented computer program which uses Qualitative Process theory to make predictions and interpret simple measurements. The representations and algorithms used in GIZMO are described in detail, and illustrated using several examples.

Qualitative Process Theory

by

Kenneth Dale Forbus

Massachusetts Institute of Technology

Revised version of a thesis submitted to the Department of Electrical Engineering and Computer Science on July 4, 1984 in partial fulfillment of the requirements for the degree of Doctor of Philosophy

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research Contract N00014-80-C-0505.

ABSTRACT

Objects move, collide, flow, bend, heat up, cool down, stretch, compress, and boil. These and other things that cause changes in objects over time are intuitively characterized as *processes*. To understand common sense physical reasoning and make programs that interact with the physical world as well as people do we must understand qualitative reasoning about processes, when they will occur, their effects, and when they will stop. Qualitative Process theory defines a simple notion of physical process that appears useful as a language in which to write dynamical theories. Reasoning about processes also motivates a new qualitative representation for quantity in terms of inequalities, called the *quantity space*.

This report describes the basic concepts of Qualitative Process theory, several different kinds of reasoning that can be performed with them, and discusses its impact on other issues in common sense reasoning about the physical world, such as causal reasoning and measurement interpretation. Several extended examples illustrate the utility of the theory, including figuring out that a boiler can blow up, that an oscillator with friction will eventually stop, and how to say that you can pull with a string, but not push with it. This report also describes GIZMO, an implemented computer program which uses Qual ative Process theory to make predictions and interpret simple measurements. The representations and algorithms used in GIZMO are described in detail, and illustrated using several examples.

Thesis Supervisor: Gerald Jay Sussman
Title: Professor of Electrical Engineering

Acknowledgments

I would like to thank:

Gerry Sussman, my thesis supervisor, for prodding me at the right times.

Patrick Winston and Mike Brady, the other members of my thesis committee, for insightful comments, encouragement, and suffering through innumerable drafts.

Johan de Kleer, Pat Hayes, and John Seely Brown, for getting me started in this line of work and for countless fruitful, inspiring discussions.

Reid Simmons, Dan Weld, and Brian Williams, the other members of the lab's "qualitative mafia", for many stimulating discussions and for pointing out so many errors.

Al Stevens, Bruce Roberts, Albert Boulanger, and Glen Abrett, the STEAMER crew, for intellectual stimulation and for helping make ends meet.

Drew McDermott, whom I learn alot from by disagreeing with.

Allan Collins, for pointing out what wasn't obvious.

Dave McAllester, for insight into how to build inference engines and keeping me on my toes.

Blythe Heepe, who drew the figures which came out well.

Keith Moon and Jimi Hendrix, the 3600's near my office, which labored so long and mightily.

Dedre Gentner (last but in no way least), for inspiration, stimulating discussions, and for agreeing to marry me.

CONTENTS

FIGURES

1. Introduction

Many kinds of changes occur in physical situations. Objects move, collide, flow, bend, heat up, cool down, stretch, and boil. These and other things that cause changes in objects over time are intuitively characterized as *processes*. Much of formal physics consists of characterizations of processes by differential equations that describe how the parameters of objects change over time. But the notion of process is richer and more structured than this. We often reach conclusions about physical processes based on very little information. For example, we know that if we heat water in a sealed container the water can eventually boil, and if we continue to do so the container can explode. To understand common sense physical reasoning we must understand how to reason qualitatively about processes. We must be able to determine when processes will start and stop and what their effects will be. This thesis describes *Qualtitative Process theory* (abreviated QP), a theory I have been developing for this purpose.

I hope that Qualtitative Process theory will provide an important part of the representational framework for common sense physical reasoning. In addition, QP theory should be useful in reasoning about complex physical systems. Programs that explain, repair and operate complex engineered systems such as nuclear power plants and steam machinery will need to draw the kinds of conclusions discussed here. Figure 1 illustrates some of the common sense conclusions about physical situations that are discussed in this report.

How to reason qualitatively about quantities is a problem that has plagued AI. Many schemes have been tried, including simple symbolic vocabularies (TALL, VERY TALL, etc.), real numbers, intervals, and fuzzy logic. None are very satisfying. The reason is that none of the above schemes makes distinctions that are relevant to physical reasoning. Reasoning about processes provides a strong constraint on the choice of representation for quantities. Processes usually start and stop when orderings between quantities change (such as unequal temperatures causing a heat flow). In Qualitative Process theory the value of a number is represented by a *quantity space*, a partial ordering of it with quantities determined by the domain physics and the analysis being performed. The quantity space representation appears both useful and natural in modeling a wide range of physical phenomena.

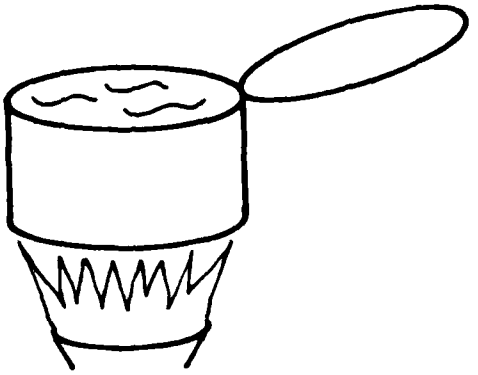
1.1 Motivation

The goal of *naive physics* [Hayes 1979a] is to represent the common sense knowledge people have about the physical world. This section examines why a theory of processes is needed, what representational burden it will carry in naive physics, and the properties such a theory must have.

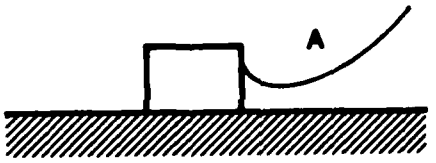
1.1.1 Change, histories, and processes

Reasoning about the physical world requires reasoning about the kinds of changes that occur and their effects. The classic problem which arises is the frame problem [McCarthy & Hayes, 1969], namely when something happens, how do we tell which facts remain true and which facts don't? Using the situational calculus to represent the changing states of the world requires writing explicit frame axioms that state what things change and what things remain the same. The number of axioms needed rises as the product of the number of predicates and the number of actions, and so adding a new action requires adding a large number of new axioms. There have been several attempts to fix this problem [Fikes & Nilsson, 1971][Minsky, 1974], but none of them are adequate. Hayes [Hayes, 1979a] argues that the situational calculus is fundamentally

Fig. 1. Examples of QP theory conclusions
Here are some conclusions QP theory can be used to draw.



Q: What might happen when the heat source is turned on?
A: The water inside might boil, and if the container is sealed it might blow up.



Q: Can we push the block with A if it is a string?
A: No, but you can pull the block if it is taut.
Q: Assuming A is an elastic band and the block is fixed in position, what might happen if we pull on it?
A: It would stretch and if pulled hard enough would break.

Q: What happens if we release the block?
A: Assuming the spring doesn't collapse, the block will oscillate back and forth. If there is friction it will eventually stop.
Q: What if it gets pumped?
A: If there is no friction the spring will eventually break. If there is friction and the pumping energy is constant then there will be a stable oscillation.

impoverished, and has developed the notion of *histories* as an alternative.

In situational calculus, *situations* are used to model the world at different times. Situations are connected by actions, and actions are specified in terms of the facts can be deduced about the situation which results from performing the action. A situation lasts as long as no action occurs, and is spatially unbounded. By contrast, histories are descriptions of objects that are extended through time but are always bounded spatially. Histories are divided into pieces called *episodes*, corresponding to what is happening to the object (episodes are defined more precisely in section 2.7).

Histories help solve the frame problem because objects can interact only when their histories intersect. For example, suppose we are building a clock in our basement. In testing parts of this gadget we look to see what parts touch each other, what parts will touch each other if they move in certain ways, and so on. By doing so we build descriptions of what can happen to the pieces of the clock. We do not usually consider interactions with the furnace sitting in the corner of the basement, because whatever is happening in there is spatially isolated from us (if it is summer it can also be "temporally isolated").

The assumption that things interact only when they touch in some way also permeates "non-naive" physics - action at a distance is banished, with fields and particle exchanges postulated to prevent its return. It means that spatial and temporal representations bear most of the burden for detecting interactions. While not easy, developing such representations seems far more productive than trying to develop more clever frame axioms.[1] In particular, the qualitative representations of space and time developed in Artificial Intelligence have precisely the desired properties for reasoning with histories -- they often allow ruling out interactions, even with very little information.

Histories are to qualitative physical reasoning what descriptions of state parameters over time are to classical numerical simulations. Processes are the analog of the differential equations used to describe the dynamics of the system.

While the classical frame problem is solved, two new problems arise to take its place.

- The *local evolution* problem: How are histories generated? Under what circumstances can they be generated for pieces of a situation independently, and then pieced together to describe the whole situation?

In the basement example above, for instance, we could safely ignore the furnace in the corner and concentrate on figuring out how pieces of the clock we are building will move. The divisions are only semi-independent, because certain kinds of changescan violate the conditions for isolation. For example, if the internal

---

1. For an example of histories in use, see [Forbus, 1981a] which describes a program called FROB that reasons about motion through space. FROB used a diagram to compute qualitative spatial representations which were used to rule out potential collisions between objects, as well as describing possible motions.

thermostat of the furnace gets stuck and it explodes, we can no longer safely ignore it.[2]

- The *intersection/interaction* problem: Which intersections of histories actually correspond to interactions between the objects?

Dropping a large steel ball through a flame, for example, won't affect its motion even if the flame is hot enough to melt it unless the gases are moving fast enough to impart significant momentum. A general solution to these problems requires knowing what kinds of things can happen and how they can affect each other -- in other words, a theory of processes.

In classical mechanics a *dynamics* describes how forces bring about changes in physical systems. For any particular domain, such as particles or fluids, a dynamics consists of identifying the kinds of forces that act between the classes of objects in the domain and the events that result from these forces. In general, we can view a *qualitative dynamics* as a qualitative theory about the kinds of things that "can happen" in a physical situation. Qualitative Process theory claims that such theories have a common character in that they are organized around the notion of ***physical processes.***

1.1.2 Reasoning tasks involving qualitative dynamics

Dynamics is central in many reasoning tasks involving naive physics. Each task is a different "style" of reasoning, appropriate for solving different classes of problems. The catalog below covers a large proportion of the cases. Examples of inferences from several of these categories will be presented later.

Determining activity: Deducing what is happening in a situation at a particular time. Besides providing direct answers to a class of questions ("what is happening here?"), it is also a basic operation in the other reasoning tasks.

Prediction: Deducing what will happen in the future of some situation. We often must work with incomplete information, so usually we can only generate descriptions of possible futures. de Kleer's notion of *envisioning* is a powerful theory about this type of deduction.[3]

Postdiction: Deducing how a particular state of affairs might have come about. ([Hayes, 1979b] contains a good example of this kind of deduction.) Postdiction is harder than prediction because of the potential necessity of postulating objects. If we have complete knowledge of a situation and have a complete dynamics, we know what objects will vanish and appear. But usually there are many ways for any particular situation to have come about. Consider walking back to our basement and finding a small pile of broken glass

---

2. Unless the physical situation is simulated by some incremental time scheme, the reasoning involved in extending histories will be inherently "non-monotonic" (in the sense of [McDermott & Doyle, 1980]). The reason is that conclusions reached by considering one part ;of a system may have to be reconsidered in the light of unexpected interactions. In incremental time simulations the changes in the entire system are computed over a very short timespan, and then the system is tested to see if any new interactions occur (such as objects colliding). The timespan is usually chosen to be small enough that interactions during a step can be ignored. The cost is that the work required to simulate a system is a function of the time scale rather than the actual complexity of the system's behavior.

3. Useful as it is, envisioning has certain limitations, especially as a sufficient model of human behavior on this task. See [Forbus, 1983a] for details.

on the floor. Looking at it we may deduce that a coke bottle was dropped, but we do not know much about its history before that, or about anything else that might have been in the room before we looked. There could have been a troupe of jugglers filling the basement, each manipulating six bottles, and a minor mishap occurred. The simplest explanation is that a single bottle was dropped, but our criteria for simplicity is not due solely to our theories of physics. Postdiction will not be considered further here.[1]

Skeptical analysis: Determing if the description of a physical situation is consistent. An example of this task is evaluating a proposed perpetual motion machine. This kind of reasoning is essential if a reasoner is to recover from inconsistent data and discover inadequacies in its theories about the world.

Measurement interpretation: Given a partial description of the individuals in the situation and some observations of their behavior, inferring what other individuals exist and what else is happening. The first part of a QP-based theory of measurement interpretation is described in [Forbus, 1983b] (see also section 5.5).

Experiment planning: Given knowledge of what can be observed and what can be manipulated, planning actions that will yield more information about the situation.

Causal reasoning: Computing a description of behavior that attributes changes to particular parts of the situation and particular other changes. Not all physical reasoning is causal, especially as more expert kinds of deductions are considered.[2] Causality seems mainly a tool for assigning credit to hypotheses for observed or postulated behavior. Thus it is quite useful for generating explanations, measurement interpretation, planning experiments, and learning (see [Forbus & Gentner, 1983]).

1.1.3 Desiderata for qualitative dynamics theories

There are three properties a theory of dynamics must have if it is to be useful for common sense physical reasoning. First, a dynamics theory must explicitly *specify direct effects* and *specify the means by which effects are propagated.* Without specifying what can happen and how the things that happen can interact, there is no hope of solving either the local evolution or intersection/interaction problems. Second, the descriptions the theory provides must be *composable.* It should be possible to describe a very complicated situation by describing its parts and how they relate.[3] This property is especially important as we move towards a complete naive physics that encompasses many domains. In dealing with a single style of reasoning in a particular class of situations an ad hoc domain representation may suffice, but sadly the world does not consist of completely separate domains. Transferring results between several ad hoc representations may be far more complex than developing a useful common form for dynamics theories.[4] Finally, the theory should

---

1. But see [Simmons, 1983], which explores the problem of reconstructing a sequence of events from a static final state in geologic map interpretation, an interesting combination of postdiction and measurement interpretation.
2. Constraint argument often seem magical to the uninitiated, which makes it unlikely that they are central in naive physics. In teaching, usually some kind of animistic explanation is proposed to justify constraint arguments to non-experts ("the particle senses which path has the least action").
3. Producing models with this property is a motivation for the "no-function-in-structure" principle [de Kleer & Brown, 1983].
4. An initial exploration of linking results from reasoning within multiple domains is described in [Stanfill, 1983].

allow *graceful extension*. First, it should be possible to draw at least the same conclusions with more precise data as can be drawn with weak data. Second, it should be possible to resolve the ambiguities that arise from weak data with more precise information.

These properties are not independent -- for example, specifying direct and indirect effects cleanly is necessary to ensure composability. Nevertheless, they are not easy to achieve. Graceful extension is bound up with the notion of good qualitative representations. Qualitative representations allow the construction of descriptions that include the possibilities inherent in incomplete information. If designed properly, more precise information can be used to decide between these alternatives as well as perform more sophisticated analyses. Representing quantities by symbols like TALL and VERY-TALL or free space by a uniform grid, for instance, does not allow more precise information to be easily integrated.

Importantly, although all qualitative descriptions are approximations, not all approximations are good qualitative descriptions. Changing a value in a qualitative representation should lead to qualitatively distinct behavior or change of state. Consider, for example, heating a pan of water on a stove. Suppose we represent the value of the temperature of the water at any time by an interval, and the initial temperature is represented by the interval [70.0 80.0], indicating that its actual temperature is somewhere between 70 and 80 degrees farenheit. Changing the "value" of its temperature to [70.0 85.0] doesn't change our description of what's happening to it (namely, a heat flow), whereas changing it to [70.0 220.0] changes what we think can be happening to it -- it could be boiling as well. While an interval representation always makes certain distinctions, they usually are not distinctions relevant to physical reasoning.

A purely qualitative theory cannot hope to capture the full scope of human reasoning about physical domains. However, by defining a basic theory using qualitative representations, we can later add theories involving more precise information -- perhaps such as intervals -- to draw more precise conclusions. In other words, we would like extensions to our basic theory to have the logical character of extension theories - more information should result in a wider class of deductions, not changing details of conclusions previously drawn. In this way we can add theories that capture more sophisticated reasoning (such as an engineer would do when estimating circuit parameters or stresses on a bridge) onto a common base.

1.2 Perspective

The present theory has evolved from several strands of work in Artificial Intelligence. The first strand is the work on *envisioning*, started by de Kleer [de Kleer, 1975](see also [de Kleer, 1979][Forbus, 1981a]). Envisioning is a particular style of qualitative reasoning. Situations are modeled by collections of objects with *qualitative states*, and what happens in a situation is determined by running simulation rules on the initial qualitative states and analyzing the results. The weak nature of the information means the result is a directed graph of qualitative states that corresponds to the set of all possible sequences of events that can occur from the initial qualitative state. This description itself is enough to answer certain simple questions, and more precise information can be used to determine what will actually happen if so desired.

Two distinct ontologies have been used in envisioning systems. In the first ontology, used for reasoning about motion, a collection of simulation laws associated with an object completely defined its behavior. While this ontology works for objects in isolation, there is no way to create simulation laws for a compound object from simulation laws for its components. This makes reasoning about several such objects connected together impossible. The second ontology is essentially a qualitative version of *system dynamics*

[Shearer, Murphy & Richardson, 1967], representing a system as devices connected together in a fixed manner. But even this representation is impoverished; the processes implicitly represented in the device laws often involve several objects at once in an interdependent fashion. Consider a liquid flow occuring between two tanks that are partially filled with water and connected by a pipe. In the device-centered ontology, this situation would be represented by the level in one tank rising, the level in the other tank falling, and motion of the liquid in the pipe from the source of the flow to the destination. But the cause of these changes -- the liquid flow -- is not explicilty represented. In addition, no means is provided to model changes in how devices affect each other (i.e., unanticipated changes in the connectivity of the system), nor for objects vanishing and appearing. This means many situations that we easily reason about can be represented at best unnaturally -- such as boiling water on a stove, in which steam appears and (eventually) the water vanishes. QP theory explicitly represents the existence of objects and provides ways to describe the conditions under which they can be created and destroyed, and thus should provide the basis for building more flexible and natural descriptions.

The second strand of work concerns the representation of quantity. Most AI schemes for qualitative reasoning about quantities violate what I call the *relevance principle* of qualitative reasoning -- qualitative reasoning about something continuous requires some kind of quantization to form a discrete set of symbols; the distinctions made by the quantization must be relevant to the kind of reasoning being performed. Almost all previous qualitative representations for quantity violate this principle. One exception is the notion of quantity introduced by de Kleer as part of Incremental Qualitative (IQ) analysis [de Kleer, 1979], which represented quantities according to how they changed when a system input was perturbed - increasing, decreasing, constant, or indeterminate. For more general physical reasoning a richer theory of quantity is necessary. IQ analysis alone does not allow the limits of processes to be deduced. For instance, IQ analysis can deduce that the water in a kettle on a lit stove would heat up, but not that it would boil. IQ analysis does not represent rates, so we could not deduce that if the fire on the stove were turned down the water would take longer to boil (Section 5.4 describes how this conclusion can be drawn). The notion of quantity provided by QP theory is useful for a wider range of inferences about physical situations than the IQ notion.

The final strand relevant to the theory is the naive physics enterprise initiated by Pat Hayes [Hayes, 1979a]. The goal of naive physics is to develop a formalization of our common sense physical knowledge. From the perspective of naive physics, Qualitative Process theory is a *cluster* - a collection of knowledge and inference procedures that is sensible to consider as a module. The introduction of explicit processes into the ontology of naive physics should prove quite useful. For instance, in Hayes' axioms for liquids ([Hayes, 1979b]) information about processes is encoded in a form very much like the qualitative state idea.[1] This makes it difficult to reason about what happens in situations where more than one process is occurring at once - Hayes' example is pouring water into a leaky tin can. In fact, difficulties encountered in trying to implement a program based on his axioms for liquids were a prime motivation for developing Qualitative Process theory.

---

1. See for example axioms 52 through 62.

1.3 Overview

This report describes the fundamentals of Qualitative Process theory, and describes an implemented computer program, called GIZMO, which uses Qualitative Process theory to draw conclusions about the physical world. The report is divided into two major sections, "theory" (chapters 2 - 5) and "practice" (chapters 6 - 10). Everything in the "practice" chapters is fully implemented and runs as stated. Anything in the "theory" chapters which is not mentioned in the practice chapters should not be assumed to be part of a currently running program. While perhaps unusual, I hope this organization will dispell the usual hazy boundaries in AI papers drawn between what has been demonstrated to run and what hasn't.

Here are the contents of the "theory" section:

Chapter 2 provides the basic framework for representing objects, states, histories, quantities, and relationships between quantities. The *quantity space* is introduced to provide a qualitative description of numerical values, and the idea of a *qualitative proportionality* is introduced to describe functional dependencies. *Individual views* are introduced to describe both the contingent existence of objects and states of objects.

Chapter 3 introduces processes, describes how to define them, and explores associated concepts such as *influences* and vocabularies of processes. It also describes the basic deductions sanctioned by the theory, including analyzing the net effects of several processes and predicting state changes.

Chapter 4 illustrates these deductions by several extended examples, including modeling a boiler, motion, materials, and an oscillator.

Chapter 5 explores additional consequences of QP theory, including detecting changing equilibriums, causal reasoning, a language for expressing causal connections, a notion of differential analysis, and a theory of interpreting measurements taken at a single instant.

The "practice" section includes:

Chapter 6 provides an overview of GIZMO.

Chapter 7 describes how the various constructs of QP theory are embodied in GIZMO.

Chapter 8 describes GIZMO's language for representing domain models, and simple models of fluids and motion that are used as a source of examples for the next chapters.

Chapter 9 describes the algorithms used in GIZMO to implement the basic deductions of QP theory. The algorithms are written in "structured english" for readability.

Chapter 10 describes the algorithms which implement the measurement interpretation theory presented in chapter 5.

Chapter 11 describes the envisioning algorithm, describes how the frame problem is solved for simulation within the QP ontology, and discusses the problem of summarizing descriptions of behavior.

Finally, chapter 12 provides a summary, discusses extensions and potential applications, and places the theory into the perspective of other recent work in Artificial Intelligence. An appendix describes the particulars of the DEBACLE inference engine which underlies GIZMO.

The casual reader should read chapters 2 and 3 to get the basic ideas of the theory, and skim the examples in chapters 4, 10, and 11 to get an idea of how these ideas can be used to solve problems. The reader interested in a deep understanding of the theory should of course just read straight through.

A word on notation. Axioms are used only when they will help the reader interested in the fine details. Although a full axiomatic description might be desirable, there are a host of complex technical details involved, few of which essentially contribute to understanding the ideas. When used, axioms are written in a more or less standard sorted predicate calculus notation. The following notational conventions are used in *axioms*: Predicates and relations are capitalized (e.g., `Fluid-Connection`), and functions are in lower case (e.g., `amount-of`, `made-of`). Sorts are underlined (e.g., time). Individuals (often physical objects) are written in upper case (e.g., `WA`) and variables are written in lower case (e.g., `p`). Small finite sets are enclosed by braces ("{" "}"). Meta-linguistic entities are italicized and surrounded by angle brackets, i.e.,

⟨a number⟩ + ⟨a number⟩ = ⟨a number⟩

When non-standard notation is introduced an effort will be made to show an interpretation of it in terms of logic. This should not necessarily be taken as an endorsement of logic as "the meaning of" the statements.

2. Objects and quantities

To talk about change we first establish some conventions for describing objects and their properties at various times. This section describes the temporal notation used and develops the representation of quantity and the *quantity space* representation for values. *Individual views* are introduced to describe both the contingent existence of objects and object properties that change drastically with time. The idea of a *qualitative proportionality* ($\propto_Q$) is introduced to describe functional dependencies between quantities. Finally *histories* are introduced to represent "what happens" to objects over time.

2.1 Time

We will use the representation of time introduced by Allen ([Allen, 1981]). To summarize, time is composed of intervals that may be related in several ways, such as one interval being `before`, `after`, or `equal` to another. A novel feature of this representation is that two intervals can *meet*: that is, the start of one interval can be directly after the end of another interval such that no interval lies between them (i.e., in this representation, time is not dense). Instants are represented as "very short" intervals which have zero duration but still have distinct beginnings and ends.

Some additional notation is required. The functions `start` and `end` map an interval to instants that serve as its start or end points. The function `during` maps from an interval to the set of intervals and instants contained within it. We will assume a function `time` which maps from instants to some (implicit) global ordering, and a function `duration` which maps from an interval to a number equal to the difference between the times for the start and the end of the interval. We further assume that the time of the end of a piece of time is never less than the time of its start, so that the duration of an instant will be zero while the duration of an interval will be greater than zero. Finally, we will use the modal operator `T` to say that a particular statement is true at some time, such as

```
(T Aligned(PIPE3) I1)
```

to say that `PIPE3` is aligned at (or during) `I1`. Often the temporal scope of a statement will be clear in context, in which case we will not bother to use `T`.

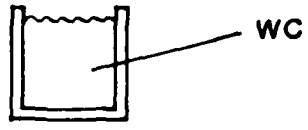
2.2 Quantities

Processes affect objects in various ways. Many of these effects can be modeled by changing *parameters* of the object, properties whose values are drawn from a continuous range. The representation of a parameter for an object is called a *quantity*. Examples of parameters that can be represented by quantities include the pressure of a gas inside a container, one dimensional position, the temperature of some fluid, and the magnitude of the net force on an object.

The predicate `Quantity-Type` will be used to indicate that a symbol is used as a function that maps objects to quantities. To say that an object has a quantity of a particular type we will use the relationship `Has-Quantity`. Figure 2 illustrates some quantities that pertain to the liquid in a cup.

Fig. 2. Quantities
Quantities represent continuous parameters of objects. Here are some quantities that are used in representing the liquid in the cup below.

```
Quantity-Type(amount-of)
Quantity-Type(level)
Quantity-Type(pressure)
Quantity-Type(volume)

Has-Quantity(WC, amount-of)
Has-Quantity(WC, level)
Has-Quantity(WC, pressure)
Has-Quantity(WC, volume)
```



2.3 Parts of quantities

A quantity consists of two parts, an *amount* and a *derivative*. The derivative of a quantity can in turn be the amount of another quantity (for example, the derivative of (one dimensional) position is the amount of (one dimensional) velocity). Amounts and derivatives are *numbers*, and the functions A and D map from quantities to amounts and derivatives respectively. Every number has distinguished parts *sign* and *magnitude*. The functions s and m map from numbers to signs and magnitudes respectively. For conciseness, the combinations of these functions that select parts of quantities will be noted as:

A_m - magnitude of the amount
A_s - sign of the amount
D_m - magnitude of the derivative, or rate
D_s - sign of the derivative

Numbers, magnitudes, and signs take on *values* at particular times. When we wish to refer to the value of a number or part of a number, we will write

```
(M Q t)
```

This statement is read as "the value of Q measured at t". (Notice that M is not the same as m.) Often we will find it convenient to speak of the value of a quantity, meaning the value of its amount. Figure 3 illustrates the use of M.

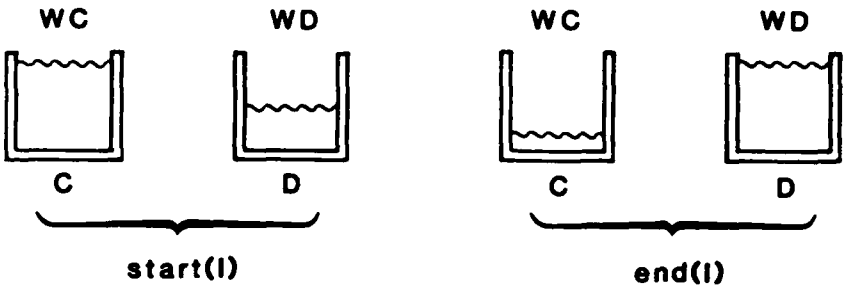
Signs can take on the values -1, 0, and 1. We will take elements of $\mathbb{R}$ as the model for the values of numbers and elements of the non-negative reals as our model for the values of magnitudes so that operations of comparison and combination are well defined.[1] Importantly, in basic Qualitative Process theory we will never know actual numerical values. What we do know about values is described next.

---

1. In this model, m becomes absolute value and s becomes signum, hence the choice of values for signs.

Fig. 3. M describes values at different times

Here are some facts about the liquids in the two containers below expressed as relationships between their quantities:




```
(M A[amount-of(WC) start(I)]) > (M A[amount-of(WD) start(I)])
(M A[amount-of (WC) end(I)]) < (M A[amount-of(WD) end(I)])
(M D_s[amount-of(WC)] I) = -1
(M D_s[amount-of(WD)] I) = 1
```

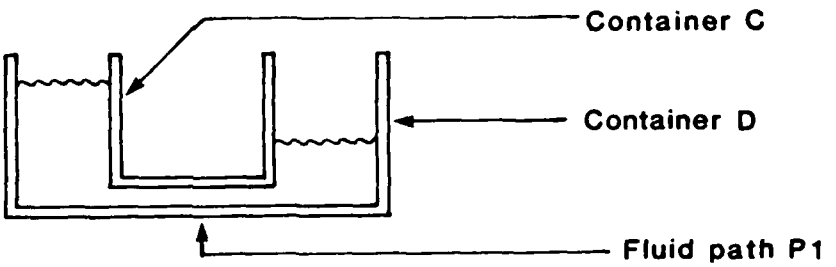
2.4 Quantity spaces

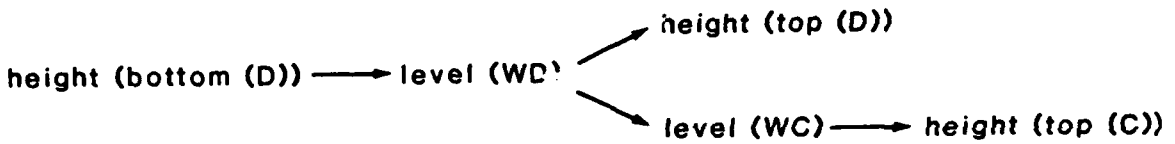
The value of a number is described in terms of its *quantity space*. A quantity space is a collection of numbers which form a partial order. Figure 4 illustrates a quantity space for the levels of fluid in two tanks C and D connected by a pipe. Note that the orderings and even the elements of a quantity space will not be fixed over time. The elements in a particular quantity space are determined by the comparisons needed to establish certain kinds of facts, such as whether or not processes are acting (we will soon see another kind of description that contributes elements to quantity spaces). There will only be a finite number of elements in any reasonable quantity space, hence there are only a finite number of distinguishable values. Thus the quantity space is a good symbolic description.

Two elements that are ordered and with no elements in the ordering known to be between them will be called *neighbors*. For the quantity space in figure 4, `level(WD)` has `height(bottom(D))`, `height(top(D))`, and `level(WC)` as neighbors, but not `height(top(C))`. Determining neighbors will be important in determining when processes start and stop acting.

We shall now be a bit more formal about defining quantity spaces and the relationships between parts of quantities. Readers who are uninterested in the details may wish to skip to the next section.

Fig. 4. Graphical notation for a quantity space
WC and WD are the pieces of liquid in containers C and D respectively. The arrow indicates that the quantity at the head is greater than the quantity at the tail. As drawn, level(WC) and height(top(D)) are unordered. For simplicity, we ignore temporal references here.





The quantity space of a number consists of a set of elements (numbers or magnitudes, often the amounts of quantities) B and a set of orderings. In basic QP theory the value of a number n will be described by the ordering relations between n and the other elements in the quantity space. The value is completely specified if the orderings between every element of B is known (i.e., the orderings form a total order), and is incomplete otherwise. Every quantity space can in principle be completely specified. A collection of inequality statements whose union with the orderings of an incompletely specified quantity space results in the quantity space being completely specified will be called a *completion* of that quantity space.

All quantity spaces have the distinguished element ZERO. ZERO serves to connect the sign of a number with inequality statements, as follows:

```
∀ n ∈ number ∀ t ∈ time
        (M n t) > ZERO ↔ (M s[n] t) = 1
        ∧ (M n t) = ZERO ↔ (M s[n] t) = 0
        ∧ (M n t) < ZERO ↔ (M s[n] t) = -1
```

Note also that the values of magnitudes are related to the values of signs and the number, in that:

```
∀ n ∈ number ∀ t ∈ time
        Taxonomy((M m[n] t) > ZERO, (M m[n] t) = ZERO)
        ∧ ((M m[n] t) = ZERO ↔ (M s[n] t) = 0)
```

(Taxonomy is drawn from [Hayes, 1979b] and means that exactly one of its arguments is true.) Thus if the value of D_s for some quantity is 0, then the derivative itself is zero and the quantity is unchanging. We will

sometimes need to combine sign values across addition. Figure 5 illustrates the algebra used.

Fig. 5. Combining sign values
This table specifies how sign values combine across addition. The cases marked by notes require additional information to determine the result.

For s[A + B]

		s[B] = -1	s[B] = 0	s[B] = 1
s[A]	-1	-1	-1	N1
	0	-1	0	1
	1	N1	1	1

N1: if m[A] > m[B] then s[A]
    if m[A] < m[B] then s[B]
    if m[A] = m[B] then 0

2.5 Individual views

Objects can be created and destroyed, and their properties can change dramatically. Water can be poured into a cup and then drunk, for example, and a spring can be stretched so far that it breaks. Some of these changes depend on values of quantities - when the amount of a piece of fluid becomes zero we can consider it gone, and when a spring breaks, it does so at a particular length (which may depend on other continuous parameters such as temperature). *Individual views* are used to model these states of affairs.

An individual view consists of four parts. It must contain a list of *individuals*, the objects that must exist before it is applicable. It has *quantity conditions*, statements about inequalities between quantities of the individuals and statements about whether or not certain other individual views (or processes) hold, and *preconditions* that are still further conditions that must be true for the view to hold. Finally, it must have a collection of *relations*, statements that are true whenever the view is true. Figure 6 illustrates a simple description of the fluid in a container.

For every collection of objects that satisfies the description of the individuals for a particular type of individual view, there is a *view instance*, or VI, that relates them. Whenever the preconditions and quantity conditions for a VI hold we say that its status is ACTIVE, and INACTIVE otherwise. Whenever a VI is active the specified relations hold between its individuals. An individual view can be thought of as defining a predicate on (or relation between) the individual(s) in the individuals field, and we will often write them that way. The contained liquid description of the previous figure is translated into logical notation in figure 7 to illustrate.

The distinction between preconditions and quantity conditions is important. The intuition is to separate changes that can be predicted solely within dynamics (quantity conditions) from those which cannot (preconditions). If we know how a quantity is changing (its D_s value) and its value (specified as a quantity space), then we can predict how that value will change (as we will see in section 3.6). We cannot predict within a purely physical theory that someone will walk by a collection of pipes through which fluid is flowing and turn off a valve. Despite their unpredictability, we still want to be able to reason about the effects of such

---

Fig. 6. Individual views describe objects and states of objects
Here is a simple description of the fluid in a container. This description says that whenever there is a container that has some liquid substance then there is a piece of that kind of stuff in that container. We will take "amount-of-in" to map from substances and containers to quantities. More elaborate models are presented later. later on.

```
Individual View Contained-Liquid(p)
        Individuals:
                con a container
                sub a liquid
        Preconditions:
                Can-Contain-Substance(con, sub)
        QuantityConditions:
                A[amount-of-in(sub, con)] > ZERO
        Relations:
                There is p ∈ piece-of-stuff
                amount-of(p) = amount-of-in(sub, con)
                made-of(p) = sub
                container(p) = con
```

Fig. 7. Translation of individual view notation into logic
Here is the contained liquid description of the previous figure translated into logical notation.

```
∀ c ∈ container ∀ s ∈ liquid
    Container(c) ∧ Liquid(s) ⇒
        (∃ IV ∈ view-instance
        ;names of individuals are used as selector functions
         con(IV) = c ∧ sub(IV) = s
        ;logical existence of individual is timeless
        ∧ (∃ p ∈ piece-of-stuff
                container(p) = c ∧ made-of(p) = s)
        ∧ (∀ t ∈ time
        ;it is active whenever Preconditions and Quantity Conditions hold
          (T Status(IV, Active) t)
           ↔[(T Can-Contain-Substance(con(IV), sub(IV)) t)
                ∧ (T A[amount-of-in(sub(IV), con(IV))] > ZERO t)]
        ;when active, p exists physically and its amount is the
        ;amount of that kind of substance in the container
        ∧(T Status(IV, Active) t) ⇒
                ((T Contained-Liquid(p) t)
                 ∧ Exists-In(p, t)
                 ∧ (M amount-of(p) t) = (M amount-of-in(s, c) t))))

;In general,
∀ IV ∈ view-instance ∀ t ∈ time
        (T Taxonomy(Status(IV, Active), Status(IV, Inactive)) t)
```

changes when they do occur, hence any dependence on these facts must be explicitly represented. This is the role of preconditions.

2.6 Functional relationships

A key notion of Qualitative Process theory is that the physical processes and individual views in a situation induce functional dependencies between the parameters of a situation. In other words, by knowing the physics you can tell what, if anything, will happen to one parameter when you vary another. In keeping with the exploration of weak information, we define

$$Q_1 \propto_{Q+} Q_2$$

(read "Q_1 is *qualitatively proportional* to Q_2", or "Q_1 q-prop Q_2") to mean "there exists a function that determines Q_1, and is increasing monotonic (i.e., strictly increasing) in its dependence on Q_2". In algebraic notation, we would write

$$Q_1 = f(\ldots, Q_2, \ldots)$$

If the function is decreasing monotonic (i.e., strictly decreasing) in its dependence on Q_2, we say

$$Q_1 \propto_{Q-} Q_2$$

and if we don't know whether it is increasing or decreasing,

$$Q_1 \propto_{Q} Q_2$$

For example, we would express the fact that the level of water in a cup increases as the amount of water in the cup increases by adding into the relations of the contained liquid description:

level(p) $\propto_{Q+}$ amount-of(p)

It is important to notice how little information $\propto_Q$ carries. Consider the relationship between level and amount-of stated above. Effectively, all we know is that, barring other changes, when amount-of rises or falls level will also. From this statement alone we do not know what other parameters might affect level, nor do we know the exact way level varies with amount-of. That $\propto_{Q+}$ statement is satisfied by all of the following equations (assuming appropriate range restrictions):

level(p) = amount-of(p)

level(p) = [amount-of(p)]^2

level(p) = sin(amount-of(p))

level(p) = amount-of(p) * temperature(p)

and many more.

Often we will leave the function implied by $\propto_Q$ unnamed. When it is necessary to name the function, we will say

Function-Spec(<id>, <specs>)

where <id> is the name of the function being defined and <spec> is a set of statements that further specify the function. Suppose for example that level is expressed in a global coordinate system, so that whenever two open containers whose bottoms are at the same height have fluid at the same level, the pressure each fluid exerts on the bottom of its container is the same. We might introduce a function p-l-fun that relates pressures to levels:

Function-Spec(p-l-fun, {pressure(p) $\propto_{Q+}$ level(p)})

Then if c1 and c2 are containers such that

(M level(c1) t0) = (M level(c2) t0)

then since

pressure(c1) = p-l-fun(level(c1))
pressure(c2) = p-l-fun(level(c2)),

by the equalities above we have

(M pressure(c1) t0) = (M pressure(c2) t0)

Notice that we could not draw this conclusion without knowing that the function which relates pressures to levels is the same for both containers.

Sometimes we want to express the fact that a function depends on something that is not a quantity. In that case we will say

F-dependency(<id>, <thing>)

In the contained liquid description, for instance, the level depends on the size and shape of the container as well as the amount of water. Assuming `shape` and `size` are functions whose range is something other than quantities, we would write

```
Function-Spec(level-function, {level(p) ∝Q+ amount-of(p)})
F-dependency(level-function, shape(container(p)))
F-dependency(level-function, size(container(p)))
```

to express this fact. Thus if two containers have the same size and shape, a particular amount of water will result in the same level, but if the size or shape is different we cannot deduce anything about the relative levels of water.

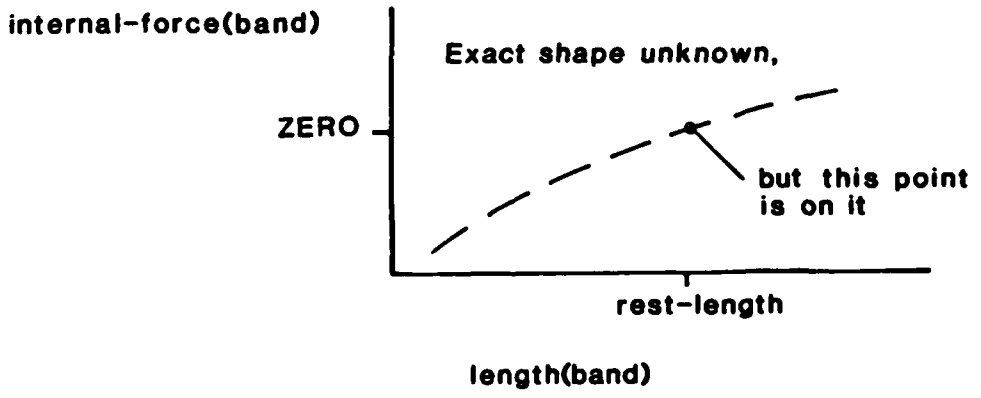
The definition of $\propto_Q$ is motivated in part by issues involved in learning and causal reasoning, and we postpone further discussion of its variants until Section 5.2. There is one other kind of information that can be specified about the function implied by $\propto_Q$'s, and that is a finite set of *correspondences* it induces between elements of the quantity spaces it connects. An example of a correspondence is that the force exerted by an elastic band is zero when it is at rest. This would be written:

```
Correspondence((A[internal-force(band)], ZERO),
               (A[length(band)], A[rest-length(band)]))
```

Correspondences are the means of mapping value information (inequalities) between quantity spaces via $\propto_Q$. For example, if the length of the band described above is greater than its rest length the internal force is greater than zero (figure 8).

Fig. 8. Correspondences link quantity spaces across $\propto_Q$

A correspondence statement allows information about inequalities to be transferred across qualitative proportionalities ($\propto_Q$'s). The rough shape of the graph below is determined by the $\propto_Q$, the equality between the two points is determined by the correspondence.




```
internal-force(band) ∝Q+ length(band)
Correspondence ((A[internal-force(band)], ZERO),
                (A[length(band)], A[rest-length(band)]))
```

2.7 Histories

To represent how things change through time we use Hayes' notion of a *history*. We assume the concepts introduced in [Hayes, 1979b] as our starting point. To summarize, the history of an object is made up of *episodes* and *events*. Episodes and events differ in their temporal aspects. Events always last for an instant, while episodes usually occur over an interval of time. Each episode has a `start` and an `end` which are events that serve as its boundaries. Following [Allen, 1981], we assume that episodes and events `meet`, that is, the start of some piece of history is directly after the end of the previous piece with no time in between. This allows us to say, for example, that the episode of heating water on a stove is ended by event of the water reaching its boiling temperature, yet during the episode the temperature was below the boiling point.

The particular class of histories Hayes introduced will be called *parameter* histories, since they are mainly concerned with how a particular parameter of a specific individual changes.[1] Objects can have more than one parameter, and these parameters often can change independently. For example, if we drop a steel ball past a flame, the ball will heat up a bit but the motion won't be affected (unless the combustion gases impart significant momentum to it). Thus the history of an object includes the union of its parameter histories. Figure 9 illustrates the parameter histories for the situation just described. The criteria for individuation, for breaking up time into episodes and events are changes in the values of quantities and their parts. The spatial component of parameter histories is inherited from the object they are a parameter of. In figure 9, for example, the events consist of the ball's position reaching `h2` and `h1`, because different values occurred before and after that time. The final component of an object's history are the histories for the processes it participates in, but this will be elaborated later in section 3.7.

Again following Hayes, a *slice* of a history denotes a piece of an object's history at a particular time. We denote the slice of an individual `i` at time `t` by

```
at(i, t)
```

If we let all functions, predicates, and relations that apply to objects apply to slices as well, with functions that map from objects to quantities map from slices to values, then we could be rid of `T` and `M` and just talk in terms of slices. For instance, instead of writing

```
(T Aligned(P1) t0)
(M A[amount-of(WC)] t0) > (M A[amount-of(WB)] t0)
```

we would write

```
Aligned(at(P1, t0))
A[amount-of(at(WC, t0))] > A[amount-of(at(WB, t0))]
```

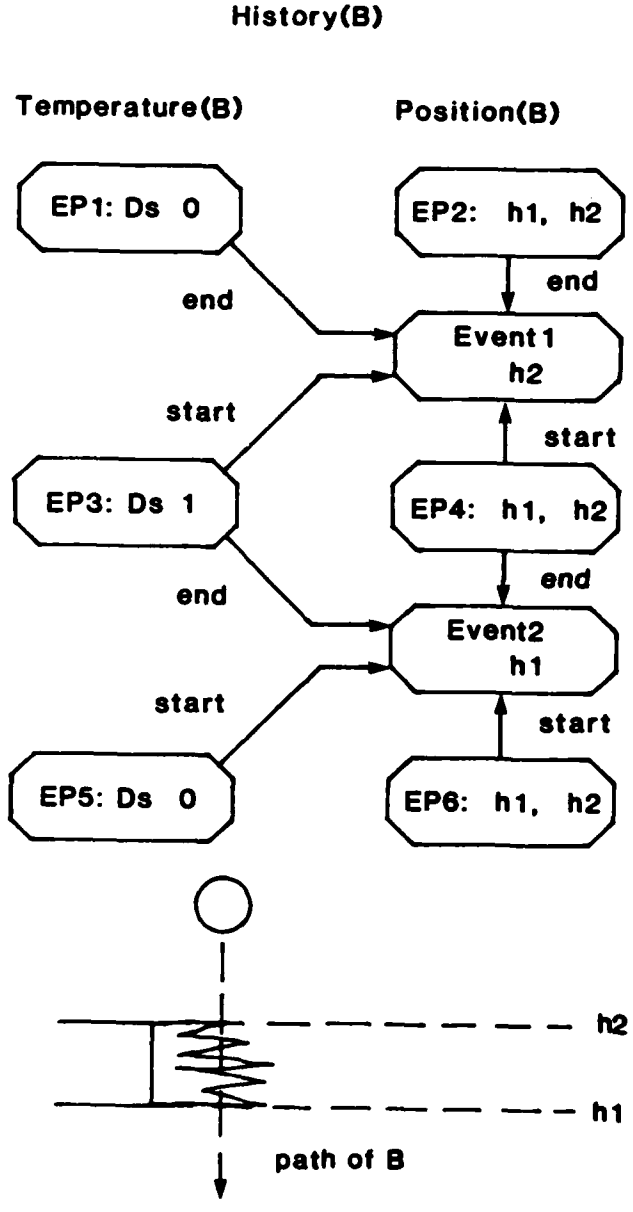
As we will see in the "practice" chapters, this in fact is the convention used in the implementation because it simplifies indexing. However, for clarity of exposition we will continue to use `T` and `M` in presenting the theory.

The notion of history so far is "object centered". Since processes will often act between several

---

1. In fact, Hayes' examples are parameter histories for "amount of stuff", representing an object solely as a piece of space-time.

Fig. 9. Parameter histories describe when values change
Part of the parameter histories for a ball being dropped through a flame are depicted below. Time runs from top to bottom, and the portion of the history that depicts what is happening (motion and heat flow) is not shown.



objects, we need a way of talking about several objects at a particular time. We will recycle the term *situation* to mean a collection of slices for a set of objects under consideration at some particular time. Unlike situational calculus, the temporal aspect of a situation can be either an instant or an interval. Also, a situation is now spatially bounded - its spatial extent is that of the slices that comprise it. In formulae where times are required, we will assume a coercion from a situation or event to its time so that we can freely use the names of situations in expressions involving T and M.

The question of what constitutes a useful situation brings us back to the local evolution problem described in the introduction. We may now state it more precisely: Given some collection of objects that we know about at a particular time, can we figure out some way to divide them up into situations that can be considered semi-independently?[1] For the moment we will leave the criteria of what constitutes useful situations unspecified; we will return to this problem in Section 3.7 after have discussed processes.

---

1. In current AI systems this problem usually does not arise because the situations under consideration are composed solely of relevant objects. However, as we attempt to make programs that can deal with more realistic problems this issue will become very important.

3. Processes

A physical situation is described in terms of a collection of objects, their properties, and the relationships between them. So far our description of the world has been static -- we can describe that things are different from one time to another, but have not provided the means by which changes actually occur. The ways in which things change are intuitively characterized as processes. A *physical process* is something that acts through time to change the parameters of objects in a situation. Examples of processes include fluid and heat flow, boiling, motion, stretching and compressing.

This section describes what processes are, including how to specify them, and elaborates the notion of *influences*. A catalog of basic deductions involving processes illustrates the kinds of conclusions that can be drawn within QP theory. Histories are extended to include occurrences of processes, and the role of processes in specifying a language of behavior is discussed.

3.1 Defining processes

A process is defined by five parts:

- The *individuals* it applies to,
- A set of *preconditions*, statements about the individuals and their relationships other than quantity conditions.
- A set of *quantity conditions*, that are either statements of inequalities between quantities belonging to the individuals (including domain-dependent constants and functions of them) or statements about the status of processes and individual views.
- A set of *relations* the process imposes between the individuals, along with new entities that might be created.
- A set of *influences* imposed by the process on the parameters of the individuals.

Figure 10 illustrates process specifications for heat flow and boiling. (For fans of logic, figure 11 illustrates how the boiling process would look translated into predicate calculus).

As you can see, a process is just like an individual view -- it is a time-dependent thing -- except it has something called *influences*. To recapitulate, for every collection of objects that satisfy the individuals specification for a particular type of process, there will be a *process instance*, or PI, that relates them. The process instance will be active, representing the process acting between these individuals, exactly whenever both the preconditions and the quantity conditions are true. Preconditions are those factors that are outside Qualitative Process theory, such as someone opening or closing a valve to establish a fluid path, but still relevant to whether or not a process occurs. The quantity conditions are those statements that can be expressed solely within QP theory, such as requiring the temperature of two bodies to be different for heat flow to occur, or a heat flow to occur as a prerequisite to boiling. The set of relations associated with a process are the relationships it imposes between the objects it is acting on. The relations component usually describes,

Fig. 10. Examples of physical process definitions

Heat flow happens between two objects that have heats and are connected via some path through which heat can flow. The predicate `Heat-Aligned` is true exactly when heat can flow through the path. Boiling happens to a contained liquid being heated, and creates a gas made of the same stuff as the liquid. `t-boil` represents the boiling point for the piece of stuff involved.

```
process heat-flow

Individuals:
        src an object, Has-Quantity(src, heat)
        dst an object, Has-Quantity(dst, heat)
        path a Heat-Path, Heat-Connection(path, src, dst)

Preconditions:
        Heat-Aligned(path)

QuantityConditions:
        A[temperature(src)] > A[temperature(dst)]

Relations:
        Let flow-rate be a quantity
         A[flow-rate] >  ZERO
         flow-rate ∝Q+ (temperature(src) - temperature(dst))

Influences:
        I-(heat(src), A[flow-rate])
        I+(heat(dst), A[flow-rate])

-----------------------------------------------

process boiling

Individuals:
        w a contained-liquid
        hf a process-instance, process(hf) = heat-flow
                               dst(hf) = w

QuantityConditions:
        Status(hf, Active)
        A[temperature(w)] = A[t-boil(w)]

Relations:
        There is g ∈ piece-of-stuff
        gas(g)
        substance(g) = substance(w)
        temperature(w) = temperature(g)
        Let generation-rate be a quantity
        A[generation-rate] > ZERO
        generation-rate ∝Q+ flow-rate(hf)

Influences:
        I-(heat(w), A[flow-rate(hf)])
        :The above counteracts the heat flow's influence
        I-(amount-of(w), A[generation-rate])
        I+(amount-of(g), A[generation-rate])
        I-(heat(w), A[generation-rate])
        I+(heat(g), A[generation-rate])
```

Fig. 11. Boiling expressed as an axiom
Here is how the boiling description could be written as an axiom. For clarity, temporal references have been omitted. Influence adders are explained in section 7.2.

```
∀ w ∈ contained-liquid ∀ hf ∈ process-instance
(process(hf) = heat-flow ∧ dst(hf) = w ⇒
  [∃ pi ∈ process-instance
   process(pi) = boiling ∧ w(pi) = w ∧ hf(pi) = hf
  ∧     [(Status(hf, Active) ∧ A[temperature(w)] = A[t-boil(w)])
          ↔ Status(pi, Active)]
  ∧ [Status(pi, Active) ⇒
        [∃ g ∈ piece-of-stuff ∃ generation-rate ∈ quantity
           Boiling(w, hf)
           ∧ gas(g)
           ∧ substance(g) = substance(w)
           ∧ temperature(w) = temperature(g)
           ∧ A[generation-rate] > ZERO
           ∧ generation-rate ∝Q+ flow-rate(hf)
           ∧ A[flow-rate(hf)] ∈ MinusInputs(InfluenceAdder(heat(w)))
           ∧ A[generation-rate]
                     ∈ MinusInputs(InfluenceAdder(amount-of(w)))
           ∧ A[generation-rate]
                     ∈ PlusInputs(InfluenceAdder(amount-of(g)))
           ∧ A[generation-rate]
                     ∈ MinusInputs(InfluenceAdder(heat(w)))
           ∧ A[generation-rate]
                     ∈ PlusInputs(InfluenceAdder(heat(g)))]]]
```

---

but is not limited to, indirect effects via functional relationships between quantities, such as the flow rate in fluid flow being qualitatively proportional to the difference in the pressures of the contained fluids involved. The relations also include descriptions of any new individuals created by the process, as for example the steam generated by boiling, and facts needed by external representations, such as describing appearances. Influences are discussed next.

3.2 Influences and integration

Influences specify what can cause a quantity to change. There are two kinds of influences, *direct* and *indirect*. The influences component of a process specifies the direct influences imposed by that process. For example, in a flow process the flow rate will typically correspond to the increase in the amount of "stuff" at the destination and to the decrease in the amount of "stuff" at the source. To indicate that the number `n` is a direct influence on the quantity `Q`, we write

```
I+(Q, n)
I-(Q, n)
I±(Q, n)
```

according to whether its influence is positive, negative, or unspecified. Importantly, processes are the only source of direct influences. If at least one process is directly influencing a quantity `Q` at some particular time, then we say that `Q` is *directly influenced*. If a quantity is directly influenced, then its derivative equals the sum of all of the direct influences on it.

An indirect influence occurs when a quantity is a function of some other quantity that is changing.

Qualitative proportionalities ($\propto_Q$'s), introduced earlier, are the means of specifying these effects. Sometimes we will refer to a process or quantity *indirectly influencing* some quantity. One quantity indirectly influences another if the second quantity is qualitatively proportional to the first. A process indirectly influences a quantity Q_1 if it directly influences some quantity Q_2 which in turn indirectly influences Q_1.

Notice that direct influences tell us much more about the relationship between quantities than indirect influences. Multiple direct influences on a quantity are combined by addition, but since $\propto_Q$ provides so little information about the exact form of the underlying function, the result of multiple indirect influences cannot always be calculated. Section 3.6.3 discusses this issue in detail.

At any particular time a quantity must be either directly influenced, indirectly influenced, or not influenced at all. Importantly, we assume that no quantity is both directly and indirectly influenced at once. A domain physics that allows a quantity to be both directly and indirectly influenced at the same time is considered to be inconsistent. This may seem odd, given that relationships between quantities in "real" physics are often specified as constraint equations. For example, we could express the equation

```
F = m * a
```

three different ways using qualitative proportionalities, each corresponding to one parameter being described as a function of the other two. How, and why, do we select a particular function to represent the constraint relationship?

The choice is made to reflect the way causality works in the domain. In thinking about motion, for instance, we cannot directly apply an acceleration - we must apply a force to cause an acceleration. Similarly, we cannot by accelerating something or pushing on it cause its mass to change, yet its mass will affect how much acceleration we get for a given push. Hence the proper rendering of `F=m*a` is

```
a ∝Q+ F
a ∝Q- m
```

There is a subtle issue lurking here. In a sense, directly influenced parameters are "independent", in that we can cause changes in them directly via active processes. All other changes in quantities are an indirect result of what the processes do to the directly influenced parameters. The choice of directionality for a constraint equation must respect this fact. The full importance of this distinction will be discussed later on when examining causal reasoning (section 5.2).

The influences on a quantity are combined to determine its derivative (we describe just how later). A notion of integrability - the relationship between the derivative of a quantity and its amount - is needed. Essentially, if the derivative is negative then the amount will decrease over an interval, if positive then the amount will increase, and if zero then the amount will be the same:

```
∀ q ∈ quantity ∀ I ∈ interval
(constant-sign(D[q], I) ⇒
        (M Ds[q] during(I)) = -1 ↔ (M A[q] end(I)) < (M A[q] start(I))
        ∧ (M Ds[q] during(I)) = 1 ↔ (M A[q] end(I)) > (M A[q] start(I))
        ∧ (M Ds[q] during(I)) = 0 ↔ (M A[q] end(I)) = (M A[q] start(I))
```

where

```
∀ n ∈ numbers, ∀ I ∈ time,
constant-sign(n I) ≡ (∀ i1,i2 ∈ during(I) (M s[n] i1) = (M s[n] i2))
```

This statement is very weak compared to our usual notion of integrability.[1] In particular, it does not rest on knowing an explicit function describing the derivative and thus does not require an explicit notion of integral.

3.3 Limit points

Recall that a quantity space consists of a collection of elements and ordering relations between them. The major source of elements for the quantity space of some number n are the numbers and constants that are compared to n via quantity conditions. Because they correspond to discontinuous changes in the processes that are occurring (or Individual Views that hold), they are called *limit points*. Limit points serve as boundary conditions. For example, the temperature quantity space for an object w might include the limit points:

```
t-melt(w)        t-boil(w)
```

where the object undergoes phase changes that result in qualitatively distinct behavior. As we have seen, these different modes of behavior are modeled by individual views.

3.4 The sole mechanism assumption and process vocabularies

A central assumption of Qualitative Process theory is the *sole mechanism* assumption, namely:

All changes in physical systems are caused directly or indirectly by processes.

As a consequence, the physics for a domain must include a vocabulary of processes that occur in that domain. This *process vocabulary* can be viewed as specifying the dynamics theory for the domain. A physical situation, then, is described by a collection of objects, their properties, the relations between them (including individual views), and the processes that are occurring.

The sole mechanism assumption allows us to reason by exclusion. If we make the additional assumption that our process vocabulary for a domain is complete, then we know what types of quantities can be directly influenced (since processes are the only sources of direct influences). If we understand the objects and relationships between them well enough, we know all the ways quantities can be indirectly influenced. Thus we know all the potential ways in which any physical situation will change. Without these closed world assumptions[2] about the form and contents of dynamical theories, it is hard to see how a reasoning entity could use, much less debug or extend, its physical knowledge.

---

1. If the time involved is an instant, then we assume that the quantity "doesn't change very much" during this time. To be more exact, we assume in that case the quantity is only different by an infintesimal amount, or equivalently, that influences are finite. This assumption underlies case 2 of the equality change law, which will be discussed shortly.

2. See [Collins et al., 1975],[Moore, 1975], [Reiter, 1980] for discussions of the general importance of closed world assumptions.

3.5 Reprise

Processes should be first class entities in the ontology of naive physics. It may be tempting to think that processes are mere abbreviations for "deeper" representations, such as constraint laws. However, they are not. The temptation arises both because constraint laws are often judged to be the most elegant physical descriptions in "non-naive" physics, and because constraint-based computer models have been fairly successful for analyzing engineered systems ([Stallman & Sussman, 1977], [de Kleer & Sussman, 1978]). However, the aims of naive physics are not the same as the aims of physics or engineering analysis. In physics we are trying to construct the simplest models that can make detailed predictions about physical phenomena. When performing an engineering analysis, even a qualitative one, we have chosen a particular point of view on the system and abstracted away certain objects. Unlike either of these enterprises, naive physics attempts to uncover the ideas of physical reality that people actually use in daily life. Thus the notions that physics throws away (objects, processes, causality) for conciseness in its formal theory -- the equations -- are precisely what we must keep.

QP theory concerns the form of dynamics theories, not their specific content. For example, the heat flow process illustrated previously adheres to energy conservation, and does not specify that "stuff" is transferred between the source and destination. The language provided by the theory also allows one to write a heat flow process that violates energy conservation and transfers "caloric fluid" between the source and destination. The assumptions made about the content of dynamics theories are quite weak. Aside from the ability to write a wide variety of physical models, the weakness of its assumptions allow other theories to be written that impose further constraints on the legal vocabularies of processes. For example, conservation of energy can be expressed as a theory about certain types of quantities and the allowable patterns of influences in processes that affect those types of quantities (see section 4.5). We do not, however, wish to saddle QP theory with these assumptions.

3.6 Basic deductions

To be useful, a representation must support deductions. Several basic deductions involving the constructs of QP theory are catalogued below. It may be helpful to skip momentarily to the example in section 4.1, which illustrates these deductions step by step.

3.6.1 Finding possible processes

A process vocabulary determines the types of processes that can occur. Given a collection of individuals and a process vocabulary, the individual specifications from the elements in the process vocabulary must be used to find collections of individuals that can participate in each kind of process. These *process instances* (PI's) represent the potential processes that can occur between a set of individuals. A similar deduction is used for finding view instances.

3.6.2 Determining activity

A process instance has a status of `Active` or `Inactive` according to whether or not the particular process it represents is acting between its individuals. By determing whether or not the preconditions and quantity conditions are true, a status can be assigned to each process instance for a situation.[1] The collection of active PI's is called the *process structure* of the situation. The process structure represents what is happening to the individuals in a particular situation. Similarly, the *view structure* is the collection of active VI's in the situation. Whenever we discuss the process structure, we will usually include the view structure as well.

3.6.3 Determining changes

Most of the changes in an individual are represented by the D_s values for its quantities. A D_s value of -1 indicates the quantity is decreasing, a value of 1 indicates that it is increasing, and a value of 0 indicates that it remains constant. As stated previously, there are two ways for a quantity to change. A quantity can be directly influenced by a process, or it can be indirectly influenced via $\propto_Q$. (By the sole mechanism assumption, if a quantity is uninfluenced its D_s value is 0.) Determining the D_s value for a quantity is called *resolving* its influences, by analogy to resolving forces in classical mechanics.

Resolving a quantity which is directly influenced requires adding up the influences. If all the signs of the influences are the same then the D_s value is simply the sign value of the influences. Since we do not have numerical information, ambiguities can arise. Sometimes an anwer can be found by sorting the influences on the quantity into positive and negative sets and using inequality information to prove that one set of influences must, taken together, be larger than the other set. Of course, we will not always have even that much information.

Resolving an indirectly influenced quantity involves gathering the $\propto_Q$ statements that specify it as a function of other quantities. Because we lack detailed information about the form of the function, in many cases indirect influences cannot be resolved within basic QP theory. An example will make this point clearer. Suppose we have a quantity Q_0 such that in a particular process structure:

$$Q_0 \propto_{Q+} Q_1 \wedge Q_0 \propto_{Q-} Q_2$$

If we also know that

$$D_s[Q_1] = 1 \wedge D_s[Q_2] = 1$$

then we cannot determine $D_s[Q_0]$, because we do not have enough information to determine which indirect influence "dominates". However, if we had

$$D_s[Q_1] = 1 \wedge D_s[Q_2] = 0$$

then we can conclude that

---

1. This can require searching the completions of the relevant quantity spaces if the required orderings cannot be deduced from what is already known about the values.

$D_s[Q_0] = 1$

because Q_1 is now the only active indirect influence.

Importantly, we assume the collection of qualitative proportionalities which hold at any particular time is loop-free, that is, if $A \propto_Q B$, then it cannot be the case that $B \propto_Q A$. At first glance it might seem that this assumption makes it impossible to model systems where two parameters are interdependent, such as feedback systems. This is not the case; the key observation is that, in physical systems, such loops always contain a derivative -- which is modeled by a direct influence, not a qualitative proportionality. In thinking about fluid flow, for example, we might observe that a change in amount of liquid causes a change in flow rate, which in turn acts to change the amount of liquid. But while flow rate is qualitatively proportional to the amount of liquid (via its dependence on pressure, which depends on the level, which in turn depends on the amount of liquid), the flow rate is a direct influence on the amount of liquid. The integral connection between them serves to "break" the loop, thus ensuring the system of qualitative propotionalities is loop-free.

Domain specific and problem specific knowledge often plays a role in resolving influences. We may know that a certain influence can be ignored, such as when we ignore heat loss from a kettle on a stove to the air surrounding it. Our knowledge about particular functions may tell us which way things combine. Suppose for instance that our model of fluid flow included influences to model the changes in heat and temperature that result from mass transfer. In the source and destination `temperature` would be indirectly influenced (via `Amount-of` and `heat`), and if we knew nothing but the D_s values we could say nothing about how they will change. From Black's law, however, we know that the temperature of the source is unchanged and the temperature of the destination will rise or fall according to whether the temperature of the source is greater or less than the temperature of the destination.

3.6.4 Limit analysis

The changes in a situation can result in the process and view structures themselves changing. Determining these changes and changes in D_s values is called *limit analysis*. Limit analysis is carried out by using the current D_s values and quantity spaces to determine which quantity conditions can change.

The first step is to find the neighboring points within the quantity spaces of each changing quantity. If there is no neighbor in some direction, then a change in that direction cannot affect anything. The ordering between each neighbor and the current amount of the quantity can be combined with the D_s values of each to determine if the relationship will change (see figure 12). If the neighbor is a limit point, some processes may end there and others begin. Thus the set of possible changes in orderings involving limit points determines the ways the current set of active processes might change.[1] The set of single changes plus consistent conjunctions of changes (corresponding to simultaneous changes) forms the set of *quantity hypotheses* for the current situation. A quantity hypothesis which imposes a change in either the view or process structure (as opposed to merely indicating a change in a D_s value) will be called a *limit hypothesis*.

---

1. This assumes that rates are not infinitesimals, so that if a quantity is "moving" towards some point in its quantity space it will actually reach that value in some finite time. This assumption rules out a simple form of Zeno's paradox. Note, however, that relaxing this assumption would result in only one additional state in the possibilities returned by the limit analysis -- that the current state never changes.

Fig. 12. Linking derivatives with inequalities
This table summarizes how the ordering relationship between two quantities may change according to the sign of their derivatives over some interval.

For A > B;

		Ds[B]		
	-1	0	1	
-1	N1	=	=	
Ds[A] 0	>	>	=	
1	>	>	N2	

N1: If $D_m[A] > D_m[B]$ then =; > otherwise

N2: If $D_m[A] < D_m[B]$ then =; > otherwise

For A = B;

		Ds[B]		
	-1	0	1	
-1	N3	<	<	
Ds[A] 0	>	=	<	
1	>	>	N4	

N3: If $D_m[A] > D_m[B]$ then <;
If $D_m[A] < D_m[B]$ then >;
If $D_m[A] = D_m[B]$ then =;

N4: If $D_m[A] > D_m[B]$ then >;
If $D_m[A] < D_m[B]$ then <;
If $D_m[A] = D_m[B]$ then =;

Determining which changes and conjunctions of changes are consistent involves several types of knowledge. First, one quantity hypothesis might render another moot. For example, if a particular quantity hypothesis causes an individual to vanish, then any other quantity hypotheses involving that individual which are to occur at the same time are irrelevant. Secondly, we assume that changes implied by a quantity hypothesis must be continuous both in quantity spaces and in D_s values. Continuous in quantity spaces means that all relationships between quantities must go through equality, i.e., that the relationship between N_1 and N_2 cannot change directly from > to < or from < to >. Continuous in D_s values means a D_s value cannot jump directly from 1 to -1 or from -1 to 1. Finally, domain dependent information can be used to determine that the situation resulting from the quantity hypothesis is inconsistent. For example, if the bottoms of two open containers are at the same height and the only thing happening is a fluid flow from one to the other, then it is impossible for the source of the flow to run out of liquid.

More than one change is typically possible, as the examples in the next section illustrate. There are three reasons for this. First, if the ordering within a quantity space is not a total order more than one neighbor can exist. Second, a process can influence more than one quantity. Finally more than one process can be occurring simultaneously. The basic theory does not in general allow determining which alternative

actually occurs. The hypothesis which occurs represents a quantity (or collection of quantities) that reachs its limit point before any others do. Using Calculus as the model for quantities, this would require solving an integral equation. Since the basic theory does not include explicit integrals, this question typically cannot be decided.

There are some special situations, due to the nature of quantities, where sometimes we can do better. Consider two quantities A and B that are equal, and C and D that are unequal. If all of the quantities are changing (D_S value of -1 or 1) in ways that insure the relationships between them will change, then the finite difference between C and D implies that the change in the equality between A and B occurs first. In fact, we assume that the change from equality occurs in an instant, while the change to equality usually will takes some interval of time. We further assume that a change to equality will take an instant only when the change in value was due to a process that acted only for an instant. These facts are summarized as the *equality change law:*

> *With two exceptions, a state lasts for an interval of time. It lasts for an instant only when either*
>
> *(1) A change from equality occurs or*
>
> *(2) A change to equality occurs between quantities that were influenced away from equality for only an instant.*

The first case assumes that the values of numbers aren't "fuzzy", and the second case assumes that the changes wrought by processes are finite (i.e., no impulses).

Remember that the set of quantity hypotheses consists of single changes and conjunctions of single changes. Consider the set of conjunctive hypotheses which contain only changes that occur in an instant, and in particular, the maximal element (in terms of inclusion) of the set. The quantity hypotheses that contain this maximal element are the ones which can occur next, because the duration of an instant is shorter than the duration of an interval. By using the equality change law to identify those quantity hypotheses that represent changes that occur in an instant, we can sometimes get a unique result from limit analysis within the basic theory.

For some kinds of tasks just knowing the possible changes is enough (e.g., envisioning). If required, knowledge outside the scope of QP theory can be used to disambiguate the possibilities. Depending on the domain and the style of reasoning to be performed there are several choices: simulation [Forbus, 1981a], algebraic manipulation [de Kleer, 1975], teleology [de Kleer, 1979], or default assumptions or observations [Forbus, 1983a].

3.7 Processes and histories

Adding processes to the ontology of naive physics requires extending the history representation of change. In addition to parameter histories, we will also use *process histories* to describe what processes are occurring when. The temporal extent of a process episode is the maximal time during which the status of the instance is constant, and the spatial extent is the spatial extent of the individuals involved in it. The events that bound episodes in the process history occur at the instants at which quantity conditions, preconditions, or the existence of objects involved in the instance change. *View histories*, describing the status of view instances, are defined similarly. Process and view episodes are included in the histories of the objects that participate in the process, and the union of the object's parameter histories and the history of the processes

and views it participates in comprise its total history. Figure 13 illustrates the full history over a small interval for the ball being dropped through a flame discussed previously.

As mentioned previously, the two key problems in reasoning with histories are the *local evolution* problem (extending the known portion of an object's history, preferably by carving up the situation into pieces that can be reasoned about semi-independently) and the *intersection/interaction* problem. The key to solving them lies in having explicit descriptions of the ways changes are caused.

Recall that the processes active in a situation form its process structure (as usual, we also implicitly include the view structure to simplify discussion). Processes interact by shared influences; two processes which affect the same parameter or a process that affects a parameter mentioned in the quantity conditions of another must be considered together when figuring out if, and how, they will change. If there is no way for two processes to "communicate" by common effects, then they can be considered independently. This suggests carving up what is happening at a particular time into "non-overlapping" pieces, subsets of the process structure that do not interact.

We define *p-components* as equivalence classes on the process structure as follows. A process instance P1 (or view instance) is in the same p-component as another process instance P2 (or view instance) if either: (a) P1 influences a quantity mentioned in P2's quantity conditions, (b) P1 influences a quantity influenced by P2, (c) P1's quantity conditions mention a quantity mentioned in the quantity conditions of P2, or (d) P2 contains a $\propto_Q$ that propagates an influence of P1.

As long as a particular process structure lasts, the p-components can be reasoned about independently. For example, we usually don't worry about getting our feet wet in a basement despite the proximity of flowing water and steam in our plumbing. Changes in the process structure can bring about changes in p-components, so the conclusions made in each p-component may have to be modified depending on how the process structure changes. If our plumbing leaks, for instance, there are now ways for our feet to get wet.

The individuals affected by the processes in each p-component define a collection of things that can be reasoned about in isolation, barring certain changes in process structure. Thus we can generate object histories by evolving situations that correspond to p-components, combining the results when the process structure changes to get new p-components, and so forth. The interaction problem becomes trivial - two episodes interact if and only if the processes that give rise to them are part of the same p-component of a process structure on a situation made up of slices from those particular episodes. Figure 14 provides a graphical illustration.

Fig. 13. History for a ball dropping through a flame
Here is a piece of the history for the ball again, but with process episodes added. As before, EP<n> are episodes, and time runs from top to bottom.

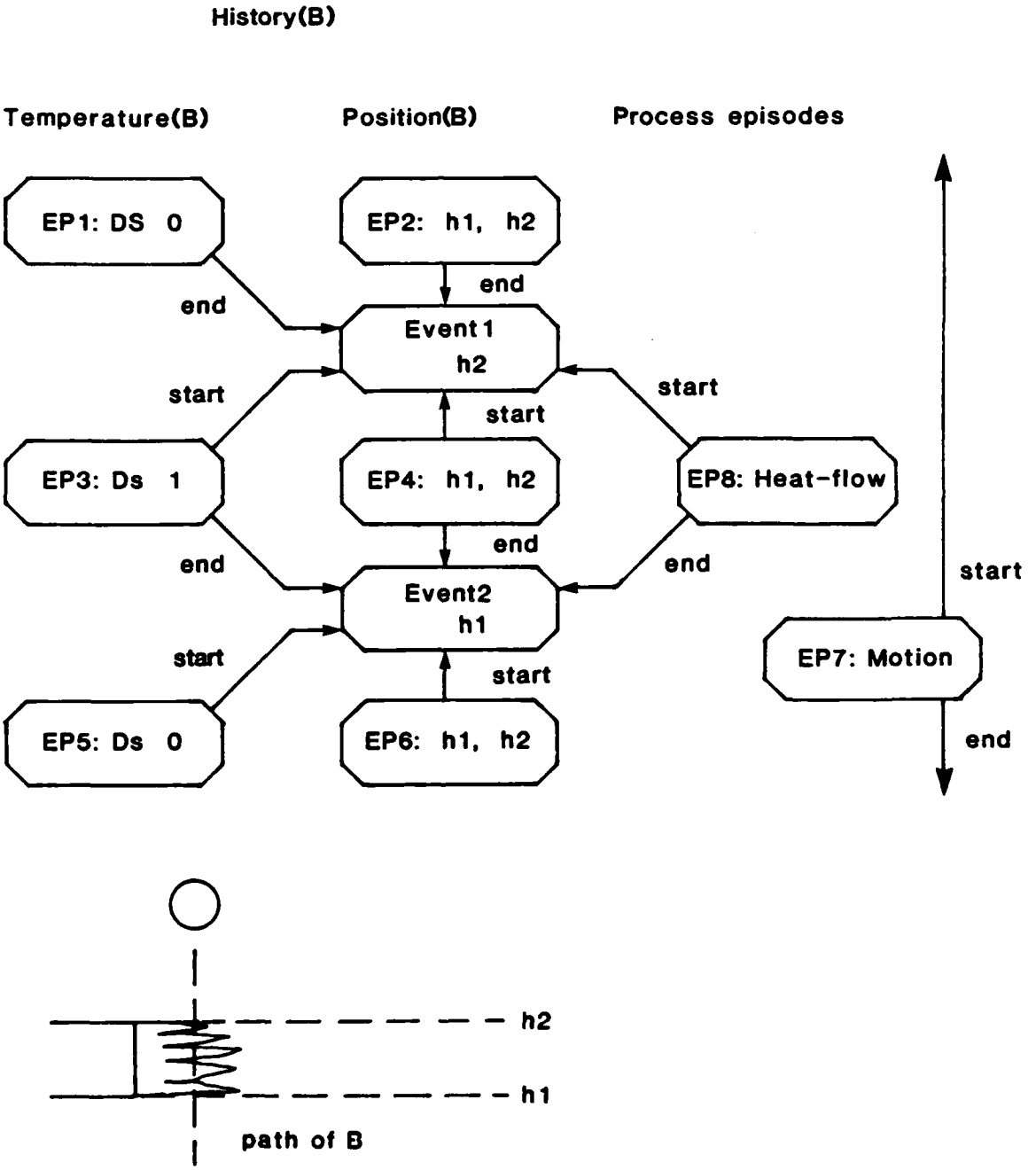
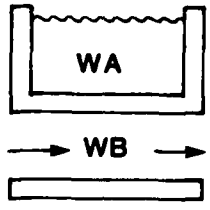


Fig. 14. Determining interactions
Suppose WA and WB are liquids, with WB being the fluid flowing through the channel below WA's container. Below are the process structures that result from different assumptions about the situation, with potential interactions indicated.




```
If the shared wall is not a heat path,
        PS: Fluid-Flow(WB, channel), no interaction
Otherwise, if A[temperature(WA)] = A[temperature(WB)]
        PS: Fluid-Flow(WB, channel), no interaction
        Otherwise, PS: Fluid-Flow(WB, channel) and a heat flow, hence they interact.
```

3.8 A language for behavior

QP theory concerns the structure of qualitative dynamics. It specifies a language in which certain common sense physical models can be written. Can this language be extended to form a full language of behavior for physical systems? Although I have not yet done so, I will argue that the answer is yes, and that several advantages would result from the extension.

A language should have primitives, some means of combining these primitives, and means of abstraction to allow new entities to be defined. Processes and individual views are the primitives in this language.[1] There are two sensible kinds of compound processes. The first kind consists of processes that form a p-component, a shared parameter combination. An example of a shared parameter combination is the intake stroke of a four cycle engine, which consists of a flow of air and gas into a cylinder and motion of the piston. The second kind consists of sequences of processes occurring over the same individuals. An example of a sequential combination is the sequence of intake, compression, combustion and exhaust strokes of a four-cycle engine. Treating these combinations as new "things" then allows properties of the system they describe to be reasoned about.

---

1. The choice of what is primitive in any particular domain's vocabulary will of course vary - for example, the description of a gas we use in section 4.1 is macroscopic. Presumably a richer process vocabulary would contain the "mechanisms" that induce these relations (i.e., the kinetic theory of gases), but there is no reason to always include such detail. Consider for example a resistor in a circuit that never exceeds its electrical capacity. The detailed mechanics of conduction hinder rather than help when calculating the current that will result from a voltage across it.

It should be clear that the shared parameter combination can be treated exactly as a simple process, specified by the union of the properties of the component processes. In other words, a shared parameter combination will have individuals, preconditions, quantity conditions, relations, and influences that work just like any other process. However, the sequential combination is not a process, because the same influences and relations do not hold over every distinct time within the occurrences of the sequential combination. A sequential combination is really a piece of a history! In particular, it is the history of the individuals affected by the processes, viewed as a system. In honor of this mixed ontological status such descriptions will be called *encapsulated histories*. Encapsulated histories (abreviated EH) are important for two reasons. First, some phenomena which can be described by them seem irreducible in terms of processes -- collisions, for example. Second, they serve as abstract descriptions for more complex behavior, e.g. in describing the pattern of activity in an oscillator.

When writing encapsulated histories, we will use most of the syntactic structure of processes and individual views, in that the combination will have individuals, preconditions, and quantity conditions. However, the relations component is restricted to holding a description of a piece of the history for the individuals, and the preconditions and quantity conditions are written relative to episodes in that piece of history. If the preconditions and quantity conditions are ever true for a partial history of a collection of objects matching the individual specifications, then the schematic history described in its relations is instantiated as part of the history of those objects.[1] We will see collisions described as an encapsulated history in section 4.3.

For those phenomena which are irreducible, the encapsulated history may be the only way to evolve the history of the object past that point. For systems where the encapsulated history serves as a summary, an interesting kind of perturbation analysis becomes possible. In performing an energy analysis, for example, the quantity conditions are re-written in terms of energy. Changes to the system, such as adding friction, are modeled by processes that influence energy, and the effects of these changes are determined by examining the episodes that comprise the encapsulated history (see section 4.5.1).

3.9 Classification and abstraction

A classification hierarchy is needed to account for the various kinds of conditions under which processes occur. For example, Hayes [1979b] identifies several distinct conditions under which fluid flow occurs. Another example is the process of motion - flying, sliding, swinging, and rolling are distinct types of motion, despite sharing certain common features. Sliding and rolling are examples of motion along a surface, and along with swinging comprise the motions involving constant contact with another object. Each of these conditions has slightly different properties, but they are sufficiently similar in the individuals they involve and the pattern of influences they engender to be considered the same kind of process. Having explicit abstract descriptions of processes should also be useful because they are often easier to rule out than more detailed descriptions. If, for instance, there is no path between two places through which an object can be moved, it

---

1. Many of diSessa's "phenomenological primitives" [diSessa, 1983] appear to be representable as encapsulated histories. Encapsulated histories are also good candidates for the first models people make of a new domain [Forbus & Gentner, 1983].

cannot get there by sliding, flying, rolling, or any other kind of motion that might exist.

Theoretically, disjunctions could be used within a single process description to cover the various cases. Doing so would lead to complicated descriptions that could not easily be reasoned about. Instead, every case will be represented by a different process. We will say that P1 is a case of P2, such as:

```
Case-of(Swinging, Motion)
```

The following restrictions hold on cases:

> *Specificity:* There is a subset of the individuals specified for P1 such that they or individuals whose existence is implied by them match the individual specifications of P2. The preconditions and quantity conditions for P1 imply the preconditions and quantity conditions for P2 respectively.
>
> *Inheritance:* All statements in the relations and influences fields of P2 hold for P1 unless explicitly excluded.

Figure 15 illustrates some specializations of the abstract motion process that will be discussed in section 5.3.

Fig. 15. Some specialized descriptions of motion

Cases of motion are organized around constraints on kinematics. The abstract motion process already includes the individuals `B`, a movable object, and `dir`, a direction. This process will be explained in section 4.3. In sliding and rolling there is contact with a surface, but different constraints on the kind of contact. Otherwise the same facts pertain to them as to the abstract version of motion.

```
Process Motion(B,dir)

individuals:
        B an object, Mobile(B)
        dir a direction

Preconditions:
        Free-direction(B, dir)
        Direction-Of(dir, velocity(B))

QuantityConditions:
         A_m[velocity(B)] > ZERO

Influences:
        I+(position(B), A[velocity(B)])
```

```
Process Slide

Case-of: Motion

Individuals:
        S a surface

Preconditions:
        Sliding-Contact(B, S)
        Along-Surface(dir, B, S)
```

```
Process Roll

Case-of: Motion

Individuals:
        S a surface

Preconditions:
        Contact(B, S)
        Round(B)
        Along-Surface(dir, B, S)
```

4. Examples

At this point a great deal of representational machinery has been introduced. It is time to illustrate how QP theory can be used in physical reasoning. The examples are fairly informal for two reasons. First, the formalization of the domains is still underway.[1] Second, while Qualitative Process theory provides an important part of any domain's theory, a complete model usually has to address several considerations besides dynamics, such as spatial reasoning (qualitative kinematics, as it were). Still, these examples are complex enough to provide an indication of the theory's utility. The assumptions about other kinds of knowledge required are noted as they occur.

4.1 Modeling fluids and liquid flow

This example illustrates some of the basic deductions sanctioned by Qualitative Process theory and introduces the representations of fluids used in other examples. (These representations are slightly more complex than the contained liquid description we have been using.) Consider the two containers illustrated in figure 16. What will happen here?

We first introduce descriptions of the fluids. Following Hayes[1979b], we individuate liquids according to what contains them. Figure 17 describes "pieces of stuff", and Figure 18 describes a particular class of pieces of stuff that are individuated by being inside a container. Any piece of stuff must be in some state, either solid, liquid, or gas. Figure 19 describes the states of substances. The interaction of state and containment is described in figure 20. Since initially there is some water in the containers, we will create individuals corresponding to the water in each container. Call the pieces of stuff in containers C and D WC and WD respectively. We will assume their temperatures are such that they are both liquids. For simplicity we will ignore the liquid in the pipe P1. We will also ignore the precise definition of fluid paths, except to note that P1 is one, connecting the two contained fluids.

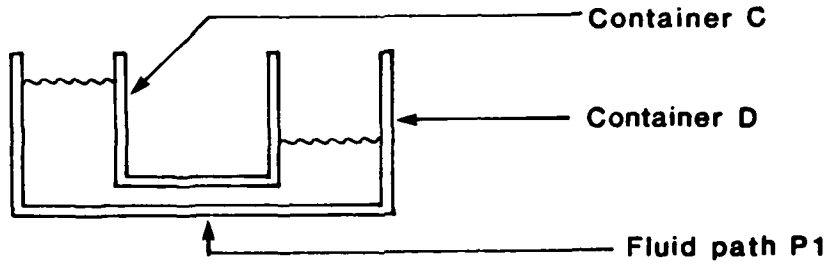
Suppose our process vocabulary consists of liquid flow, whose description is illustrated in figure 21. This model is very simple, because it ignores the possibility of different kinds of fluids and the details of how fluids move through the fluid paths ([Hayes, 1979b] illustrates some of the distinctions that should be drawn in a more detailed model).

With the situation we have so far, there are two process instances, one corresponding to flow from WC to WD and the other corresponding to flow from WD to WC. To determine if either is active (thus determining the process structure) we have to know the relative pressures of WC and WD. Assume we deduce from the relative levels that the pressure of WC is greater than the pressure of WD. Then the process instance representing fluid flow from WC to WD will be active and the process instance representing fluid flow from WD to WC will be inactive. Thus the process structure is the set consisting of

---

1. At present work is focusing on two domains: the *mechanism world*, and the *fluids world*. The mechanism world includes the blocks world but also more complex shapes and some non-rigid materials. The aim of work in the mechanism world is to understand devices such as mechanical watches and automobile transmissions. The fluids world is an attempt to extend Hayes' theory of liquids to include gases and more complex fluid systems such as found in steam plants.

Fig. 16. Two partially filled containers
Containers C and D are connected by a pipe. C contains more water than D. In general an "-in" suffix indicates a function that maps from a container and a substance to a quantity.




```
;structural description

Open-Container(C)
Open-Container(D)
Fluid-Path(P1)
Fluid-Connected(C, D, P1)

;the containers can hold water
Can-Contain-Substance(C, water)
Can-Contain-Substance(D, water)

;the levels are related
(M A[level-in(C, water)] Initial) > (M A[level-in(D, water)] Initial)
```

Fig. 17. Pieces of stuff
A piece of stuff has several quantities, a substance, and a location.

```
(∀ p ∈ piece-of-stuff
        Has-Quantity(p, amount-of)
        ∧ Has-Quantity(p, volume) ∧ Has-Quantity(p, pressure)
        ∧ Has-Quantity(p, temperature) ∧ Has-Quantity(p, heat)
        ∧ Substance(made-of(p)) ∧ Place(location(p))
        ∧ temperature(p) ∝Q+ heat(p)
```

`Liquid-Flow(WC, WD, P1).`

To find out what changes are occurring we must resolve the influences. In this situation resolving influences is simple. The fluid flow from C to D is the only cause of direct influences, changing `amount-of` for WC and WD. Each of them has only one influence, hence

D_S[amount-of(WC)] = -1

and

D_S[amount-of(WD)] = 1

These in turn influence `volume`, `level` and `pressure`, each of which has only one $\propto_Q$ applicable(see figure 20).

Fig. 18. Contained stuff

Contained-Stuff describes the conditions under which pieces of stuff exist inside a container.

```
Individual-View Contained-Stuff

Individuals:
        c a container
        s a substance

Preconditions:
        Can-Contain-Substance(c, s)

QuantityConditions:
        A[amount-of-in(c, s)] > ZERO

Relations:
        There is p ∈ piece-of-stuff
        amount-of-in(c, s) = amount-of(p)
        s = made-of(p)
        inside(c) = location(p)
```

Thus we can deduce that the volume, level and pressure of WC are all decreasing, and the volume, level and pressure of WD are all increasing. All other quantities are uninfluenced, hence unchanging. Limit analysis is similarly simple. The pressures will eventually be equal, which means the fluid flow will stop. It is also possible that the container C will run out of water, thus ending WC's existence (although it is not possible in the particular drawing shown). These results are summarized in figure 22. This graph of process structures can be used to generate a history by first creating the appropriate episodes for objects and processes from their initial slices, and then selecting one or the other limit hypothesis as the end event for that episode. Usually we will just represent the interconnections between possible process structures as we have done here. With only a single process and simple relationships between quantities, resolving influences and performing limit analysis is simple. In more complex situations resolving influences and disambiguating the possibilities raised by limit analysis will require more information, as we will see below.

Fig. 19. States of matter

The temperatures at which state changes occur are modeled by two functions `t-melt` and `t-boil`. `t-melt` and `t-boil` map pieces of stuff onto quantities, and we assume `A[t-boil]` is never less than `A[t-melt]`. The quantity conditions express the fact that a substance can be in either state at a phase boundary, but that a particular piece cannot be in both states at once. To determine the state of a piece of stuff at the phase boundary requires either knowing its history or making an assumption.

```
Individual-View Solid(p)

Individuals:
        p a piece-of-stuff

QuantityConditions:
        ¬ A[temperature(p)] > A[t-melt(p)]
        ¬ Liquid(p)
```

```
Individual-View Liquid(p)

Individuals:
        p a piece-of-stuff

QuantityConditions:
¬ A[temperature(p)] < A[t-melt(p)]
¬ A[temperature(p)] > A[t-boil(p)]
¬ Solid(p)
¬ Gas(p)
Relations:
        volume(p) ∝Q+ amount-of(p)
        t-boil(p) ∝Q+ pressure(p)
```

```
Individual-View Gas(p)

Individuals:
        p a piece-of-stuff

QuantityConditions:
        ¬ A[temperature(p)] < A[t-boil(p)]
        ¬ Liquid(p)

Relations:
        temperature(p) ∝Q+ pressure(p)
        pressure(p) ∝Q+ amount-of(p)
        pressure(p) ∝Q- volume(p)
        pressure(p) ∝Q+ heat(p)

;Instead of writing a constraint law, we use qualitative proportionalities
;to preserve the direction of physical effect.  The section on Causal
;reasoning explains why.
```

Fig. 20. Effects of state on containment

```
;Contained stuff has states as well -

(∀ p ∈ piece-of-stuff
        (Contained-Gas(p) ↔ (Contained-Stuff(p) ∧ Gas(p)))
        ∧ (Contained-Liquid(p) ↔ (Contained-Stuff(p) ∧ Liquid(p)))
        ∧ (Contained-Solid(p) ↔ (Contained-Stuff(p) ∧ Solid(p))))

;Contained liquids have levels, which are tied to amounts
;and in turn (assuming an open container) determines pressure

(∀ c ∈ contained-liquid
        Has-Quantity(c, level)
        ∧ level(c) ∝Q+ amount-of(c)
        ∧ Function-Spec(p-l-fun, (pressure(c) ∝Q+ level(c))))
```

Fig. 21. A process description of fluid flow
This simple model does not describe the existence and behavior of the liquid within the fluid path.

```
process liquid-flow

Individuals:
        src a contained-liquid
        dst a contained-liquid
        path a fluid path, Fluid-Connected(src, dst, path)

Preconditions:
        Aligned(path)

Quantityconditions:
        A[pressure(src)] > A[pressure(dst)]

Relations:
        Let flow-rate be a quantity
        flow-rate ∝Q+ (A[pressure(src)] - A[pressure(dst)])

Influences:
        I+(amount-of(dst), A[flow-rate])
        I-(amount-of(src), A[flow-rate])


;A fluid path is aligned if only if either it has no valves or every valve is open
(∀ p ∈ fluid-path
        ((number-of-valves(p) = 0) ⇒ Aligned(p))
        ∧ ((number-of-valves(p) > 0) ⇒ (∀ v ∈ valves(p) Open(v)) ↔ Aligned(p))
        ∧ ¬ (number-of-valves(p) < 0))
```

Fig. 22. Resolved influences and limit analysis
The results of resolving influences and limit analysis for the situation involving two containers are summarized below. The individuals in the situation are labeled IS, the Process Structure by PS, and limit hypotheses by LH.

```
Changing D_S values:

D_S[amount-of(WC)] = -1          D_S[amount-of(WD)] = 1
D_S[volume(WC)] = -1             D_S[volume(WD)] = 1
D_S[level(WC)] = -1              D_S[level(WD)] = 1
D_S[pressure(WC)] = -1           D_S[pressure(WD)] = 1
D_S[heat(WC)] = 0                D_S[heat(WD)] = 0
D_S[temperature(WC)] = 0         D_S[temperature(WD)] = 0

limit analysis:

                IS: {WC, WD}
                PS: {Liquid-Flow(WC, WD, P1)}

LH: A[pressure(WC)] = A[pressure(WD)]      LH: A[amount-of(WC)] = ZERO
IS: {WC, WD}                              IS: {WD}
PS: {}                                    PS: {}
```

4.2 Modeling a boiler

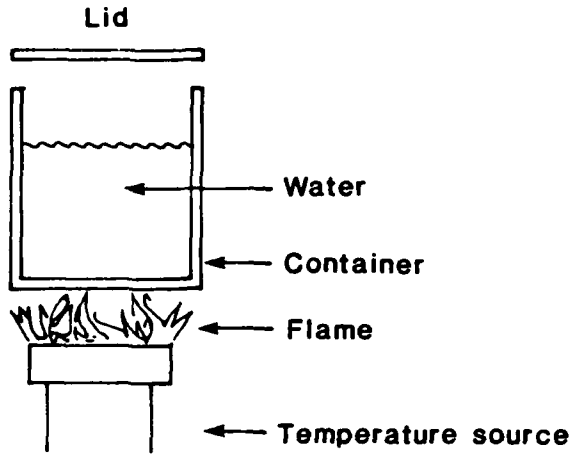
Let us consider the possible consequences of the situation shown in figure 23. The situation consists of a container partially filled with water. Initially the lid of the container is open; we stipulate that if boiling ever occurs, the lid will be closed and sealed. A flame, modeled as a temperature source, is placed so that heat can be conducted to the container and water (i.e., there is an aligned heat path between them). What sorts of things can happen?

To begin with, we need the contained substances defined in the previous example and a model of containers. We assume that if the pressure inside the container exceeds a particular pressure p-burst(CAN), the container will explode. Figure 24 describes the container model. We assume that, in addition to liquid flow, the process vocabulary includes heat flow and boiling, as presented in section 3.1. We ignore the rest of the details, such as the nature of heat and fluid paths and the detailed geometry of containers.

We start by assuming that no processes are active before the heat source is turned on; in other words that all temperatures, pressures, etc. are equal so there are no flows, and that the temperatures are in the appropriate regions of their quantity spaces so that no state changes are occurring. (Note that, as usual, we are making a closed world assumption both in assuming our process vocabulary is complete and that we know all of the relevant individuals). Since there is a heat path between the source and the container, if we turn the heat source on and if

```
A[temperature(SOURCE)] > A[temperature(WATER)]
```

Fig. 23. A simple boiler



there is a heat flow from the source to the water. We ignore the influence of the heat flow on the source by assuming

$$D_S[\text{temperature(SOURCE)}] = 0$$

The only influence on temperature(CAN) is that of the heat flow, so

$$D_S[\text{temperature(CAN)}] = 1$$

This in turn causes a heat flow to the air surrounding the container and to the air and the water inside the container. Since we are only thinking about the container and its contents most of these changes will be ignored, and from now on when we refer to heat flow it will be the flow from the flame to the contained stuff, using the container as the heat path. The temperature quantity space that results is illustrated in figure 25. If A[temperature(source)] > A[t-boil(water)] and the process is unimpeded (i.e., the preconditions for the heat flow remain true), the next process structure to occur will include a boiling.

Suppose the preconditions for the heat flow continue to be met and boiling occurs. Then by our initial assumptions the lid will be sealed, closing all fluid paths and thus preventing any flows. The amount-of quantity spaces that result are illustrated in figure 26. The influence of the boiling on amount-of(WATER) moves it towards ZERO. So one of the ways the process structure might change is that all of the water is converted to steam.

If all the water is converted to steam, the only active process is a heat flow from the heat source to the steam. Thus the sole influence on the heat of the steam is positive, and (because of $\propto_Q$) the temperature also rises. Heat indirectly influences pressure, so the pressure of the steam will also rise. By examining the quantity spaces for temperature and pressure we find there are two limit points which may be reached, namely that the temperature of the steam can reach the temperature of the heat source and that the pressure

Fig. 24. A simple container model

For simplicity we will model a container only as a collection of quantities, a set of pieces of stuff which are its contents, and an encapsulated history to describe the possibility of it exploding. The geometric information necessary to determine flow paths and the spatial arrangement of the contents will be ignored.

```
∀ c ∈ container
[Has-Quantity(c, volume) ∧ Has-Quantity(c, pressure)
∧ Has-Quantity(c, temperature) ∧ Has-Quantity(c, heat)
∧ (Rigid(c) ⇒ D_s[volume(c)] = 0)
∧ ¬ Open-Container(c) ⇒
          (∀ p ∈ contents(c)
              pressure(c) = pressure(p)
              ∧ temperature(c) = temperature(p))]

;note we are assuming instantaneous equilibration
;     within the container

Encapsulated History Explode

Individuals:
        c a container, rigid(c)
        e an episode

Preconditions:
        (T ¬ Open-Container(c) e)

QuantityConditions:
        (M A[pressure(c) end(e)) = (M A[p-burst(c)] end(e))
        (M A[pressure(c) during(e)) < (M A[p-burst(c)] during(e))

Relations:
        Let EV1 be an event
        end(e) = EV1
        Terminates(c, EV1)

;Terminates indicates that the object does not exist past
;that particular event
```

---

of the container (which is equal to the pressure of the steam) can reach the explosion point. In the first case there are no active processes, and in the second an explosion occurs. We have found one possible disaster, are there more? To find out, we must go back to the boiling episode and check the indirect consequences of the changes in amount-of(STEAM).

Consider some arbitrary instant I within the boiling episode. Because the steam is still in contact with the water their temperatures will be the same. Since we assumed the container would be sealed when boiling began, there are no fluid paths hence no fluid flows. Therefore during I the only influence on amount-of(STEAM) and on amount-of(WATER) is from boiling. So D_s[amount-of(STEAM)]=1 and D_s[amount-of(WATER)]=-1.

Because steam is a gas, there are several indirect influences on temperature(STEAM) and pressure(STEAM) (see figure 19). In particular,

Fig. 25. Quantity space for water temperature
The heat flow is increasing the heat and thus (via $\propto_{Q+}$) the temperature of the water. The lack of ordering information between the temperature of the source and the boiling temperature leads to uncertainty about what will occur next.

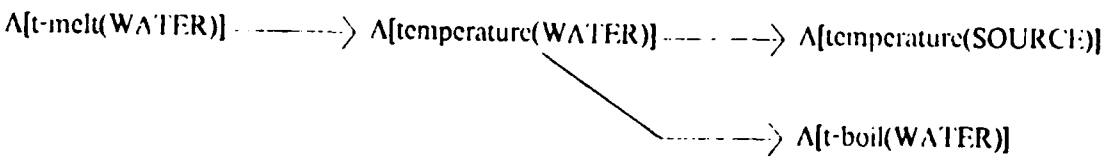
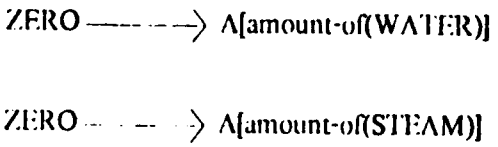


Fig. 26. `amount-of` quantity spaces



`temperature(STEAM)` $\propto_{Q+}$ `pressure(STEAM)`
`temperature(STEAM)` $\propto_{Q+}$ `heat(STEAM)`
`pressure(STEAM)` $\propto_{Q+}$ `amount-of(STEAM)`
`pressure(STEAM)` $\propto_{Q-}$ `volume(STEAM)`
`pressure(STEAM)` $\propto_{Q+}$ `heat(STEAM)`

Assuming the container is rigid, D_s`[volume(CAN)]=0`, and since the spaces of the steam and water are separate and fill the container,

`volume(CAN) = volume(WATER) + volume(STEAM)`

Since D_s`[volume(WATER)]=-1`, D_s`[volume(STEAM)]=1` and D_m`[volume(STEAM)]=`D_m`[volume(WATER)]`.

Assume the function that determines `pressure(STEAM)` is continuous in `amount-of(STEAM)`, `heat(STEAM)`, and `volume(STEAM)`. Then any particular `D[amount-of(STEAM)]` and `D[heat(STEAM)]`, we can find a corresponding `D[volume(STEAM)]` such that

(M D_s[pressure(STEAM)] I) = 0

i.e., the pressure at the end of I will be the same as it was at the start of I. Let β stand for that value of D[volume(STEAM)]. Then

(M A[volume(STEAM)] end(I)) = (M A[volume(STEAM)] start(i)) + β

is necessary for D_s[pressure(STEAM)] to be zero. A fact about steam is that, at any particular pressure and temperature, the volume of steam is very much greater than the volume of water it was produced from.[1] In other words,

D_s[pressure(STEAM)]=0 $\Rightarrow$ D_m[volume(WATER)]<<D_m[volume(STEAM)].

But in fact,

D_m[volume(STEAM)]=D_m[volume(WATER)], so D[volume(STEAM)]< β.

This means that (M A[volume(STEAM)] end(I)) will be less than

(M A[volume(STEAM)] start(I)) + β,

and because

pressure(STEAM) $\propto_{Q-}$ volume(STEAM),

the pressure of the steam will be greater than it was, i.e.,

D_s[pressure(STEAM)]=1.

Since D_s[heat(STEAM)] = 1, both of the influences on temperature(STEAM) are positive, so D_s[temperature(STEAM)] = 1.

So far we have discovered that

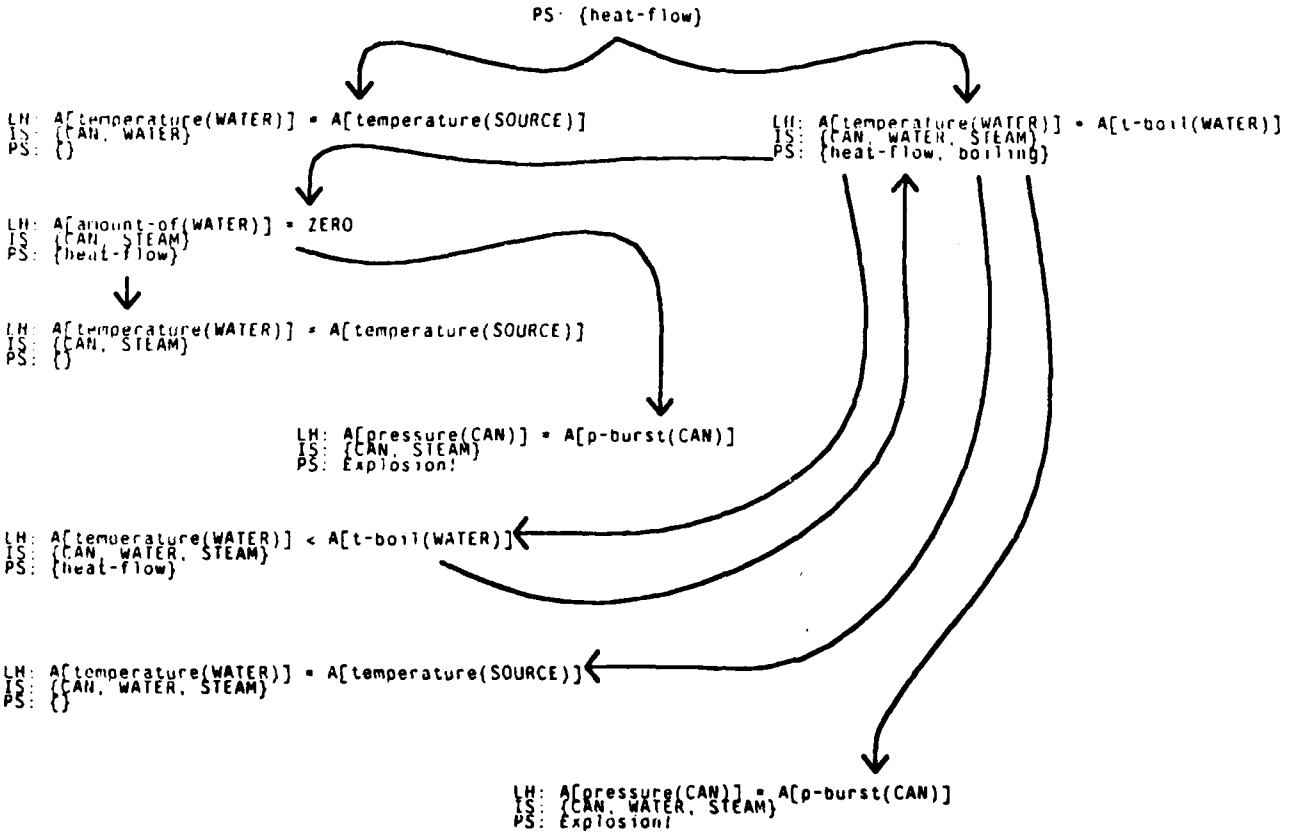
D_s[pressure(STEAM)] = D_s[temperature(STEAM)]=1.

Since the water and steam are in contact their pressures will be equal, and since pressure indirectly affects the boiling temperature, the boiling temperature will also rise. The possible relative rates introduce three cases. If the boiling temperature is rising faster (D_m[t-boil(WATER)] > D_m[temperature(STEAM)]) then the boiling will stop, the heat flow will increase heat(WATER) again, the temperature will rise, and the boiling will begin again.[2] In the other two cases (D_m[t-boil(WATER)] = D_m[temperature(STEAM)] and D_m[t-boil(WATER)] < D_m[temperature(STEAM)]) the boiling will continue, albeit at a higher temperature and pressure. In all three cases, the increasing pressure makes A[pressure(CAN)] = A[p-burst(CAN)] possible, in which case the container explodes. The alternatives are summarized in figure 27. To actually determine which of these occurs requires more information, but at least we have a warning of potential disaster.

---

1. At standard temperature and pressure, about 220 times greater in fact.
2. The astute reader will notice that this situation gives rise to a cycle of states that corresponds to a rising equilibrium rather than an oscillation. Section 5.1 discusses how to use the equality change law to distinguish between these cases.

Fig. 27. Alternatives for sealed container
Here are the process structures envisioned for water being heated in a sealed container, generated by repeated limit analysis.



4.3 Modeling motion

One process we reason about every day is motion. Motion is complex because it is intimately connected with our concepts of space and shape. Since QP theory only describes the form of qualitative dynamical theories, it cannot carry the entire representational burden imposed by motion. After developing a simple motion vocabulary, we compare the QP descriptions with the earlier *qualitative state* representation in

order to illustrate the strengths and weaknesses of the QP model.

4.3.1 A simple motion vocabulary

Consider a single object moving in one dimension. By ignoring the particular kind of motion it exhibits (FLY, SLIDE, SWING, ROLL) which depends on the particular shape and type of contact with other surfaces, we can develop an abstract vocabulary for describing motion. While very weak, such abstract descriptions have certain uses -- we can deduce that if we kick something but it is blocked, for instance, then it will not move, and if we can rule out the most abstract motion possible we have ruled out all the more specific kinds.

First we will need some simple descriptions of spatial relationships. The symbols `1` and `-1` will denote distinct directions along some axis, and for some quantity `Q`

```
Direction-Of(<direction>, Q)
```

is true exactly when A_S[Q] equals the indicated direction. The location of an object is modeled by a quantity `position`, and if there is no immobile object directly against an object `B` in direction `dir` we say

```
Free-Direction(B, dir)
```

If there is an object in that direction which is directly in contact with it, say `C`, then we say

```
Contact(B, C, dir)
```

Finally, when some object `C` lies along direction `dir` from object `B`, we will say

```
Direction-Towards(B, C, dir)
```

Figure 28 contains the process specifications for motion and acceleration. The motion process occurs when a mobile object has a non-zero velocity and is free to move in the direction of its velocity (i.e., no other objects in the way). Motion is a positive influence on position of an object, in that if the velocity is positive the position will "increase" and if the velocity is negative the position will "decrease". (The problem of mapping a quantity space onto more complex geometric frames of reference will be considered in detail below) Acceleration occurs when a mobile object has some non-zero net force in a free direction. Acceleration provides a positive influence on velocity, and in fact the influence is qualitatively proportional to the net force and qualitatively inversely proportional to the mass of the object -- the QP version of Newton's second law.

While this description is Newtonian, Aristotelian and Impetus theories can also be described.[1] One form of Aristotelian motion, for example, can be written as in figure 29. Here motion only occurs when an object is being pushed. An impetus theory is described in figure 30. Aristotelian theory has the problem of explaining what keeps a moving object going once it doesn't touch anything; impetus theory explains this by postulating that the push gives an object a kind of internal force or "impetus". While superficially like

---

1. [McCloskey, 1983] and [Clement, 1983] argue that naive theories of motion in our culture correspond to impetus theories, rather than Aristotelian theories as previously suggested.

Fig. 28. Process descriptions of Newtonian motion and acceleration
In this motion vocabulary we have abstracted away the kind of motion occuring (flying, sliding, swinging, etc.) and the complexities of motion in more than one dimension. We assume sign values are assigned to directions along some axis, with magnitudes indicating distance from some arbitrarily chosen origin.

```
Process Motion(B,dir)

individuals:
        B an object, Mobile(B)
        dir a direction

Preconditions:
        Free-direction(B, dir)
        Direction-Of(dir, velocity(B))

QuantityConditions:
         A_m[velocity(B)] > ZERO

Influences:
        I+(position(B), A[velocity(B)])

-----------------------------------------

Process Acceleration(B,dir)

Individuals:
        B an object, Mobile(B)
        dir a direction

Preconditions:
        Free-Direction(B,dir)
        Direction-Of(dir, net-force(B))

QuantityConditions:
        A_m[net-force(B)] > ZERO

Relations
        Let acc be a quantity
        acc ∝_Q+ net-force(B)
        acc ∝_Q- mass(B)
        ; The basic QP version of F = m * a
        Correspondence((A[acc] ZERO)
                       (A[net-force(B)] ZERO))
Influences: I+ (velocity(B) A[Acc])
```

---

momentum, impetus kinematics is very different.[1] Impetus also differs from momentum in that it can

---

1. In particular, impetus is not a vector quantity. Subjects vary in their beliefs as to the means of combination for impetus: they include rules like "The motion is in the direction of the biggest impetus." There are other oddities as well -- for example, impetus "remembers" not just the direction of the push but some of the previous history of directions, so that a moving object leaving a spiral tube will move in a spiral for a little while. See [McCloskey, 1983] for details.

Fig. 29. Aristotelian motion
Aristotle theorized that objects required a constant push to keep them going. Note that `velocity` does not have an existence independent of the motion process.

```
Process Motion

Individuals:
        B an object, Mobile(B)
        dir a direction

Preconditions:
        Free-Direction(B, dir)
        Direction-Of(dir, net-force(B))

QuantityConditions:
        A_m[net-force(B)] > ZERO

Relations:
        let velocity be a quantity
        velocity ∝_Q+ net-force(B)
        velocity ∝_Q- mass(B)

Influences:
        I+(position(B), A[velocity])
```

---

spontaneously dissipate. Compare the dissipation of impetus with the Newtonian model of sliding friction in figure 31. Here friction occurs when there is surface contact, and produces a force on the object that is qualitatively proportional to the normal force and acts in a direction opposite that of the motion. The effect of friction occurs indirectly, through providing a force that changes acceleration, rather than directly as in the impetus theory.

Collisions are complicated in any theory of motion, because they are usually described in terms of a piece of history. We will use an encapsulated history, as introduced in Section 3.8. The simplest description of a collision just involves a reversal of velocity, as illustrated in figure 32. As a simplification we have assumed c is immobile so that we won't have to worry about momentum transfer between moving objects and changes of direction in more than one dimension. Even our more complicated models of collisions appear to use such encapsulated histories, such as a compound history consisting of contacting the surface, compression, expansion, and finally breaking contact. The type of collision -- elastic or inelastic -- that occurs could be specified by referring to a theory of materials for the objects involved.

Fig. 30. An impetus dynamics for motion

In impetus theories of motion, a push imparts "impetus" to an object. An object's impetus is an internalized force that keeps on pushing the object, thus causing motion. Motion eventually stops because impetus dissipates with time.

```
Process Motion

Individuals:
        B an object, Mobile(B)
        dir a direction

Preconditions:
        Free-Direction(B, dir)
        Direction-Of(dir, impetus(B))

QuantityConditions:
        A_m[impetus(B)] > ZERO

Relations:
        let vel be a quantity
        vel ∝_Q+ impetus(B)

Influences:
        I+(position(B), A[vel])
```

```
Process Impart

Individuals:
        B an object, Mobile(B)
        dir a direction

Preconditions:
        Free-Direction(B, dir)
        Direction-Of(dir, net-force(B))

QuantityConditions:
        A_m[net-force(B)] > ZERO

Relations:
        Let acc be a quantity
        acc ∝_Q+ net-force(B)
        acc ∝_Q- mass(B)

Influences:
        I+(impetus(B), A[acc])
```

```
Process Dissipate

Individuals:
        B an object, Mobile(B)

QuantityConditions:
        A_m[impetus(B)] > ZERO

Relations:
        Let acc be a quantity
        A_s[acc] = A_s[impetus(B)]

Influences:
        I-(impetus(B), A[acc])
```

Fig. 31. Moving friction in newtonian sliding

Objects have a set `forces-on`, whose sum is the net force on the object. Friction is modeled by an individual view rather than a process because it contributes directly to the force on an object, rather than the derivative of the force.

```
Individual View  Moving-Friction

Individuals:
        B an object, Mobile(B)
        S a surface
        dir a direction

Preconditions:
        Sliding-Contact(B,S)

QuantityConditions:
        Motion(B, dir)

Relations:
        Let fr be a quantity
        fr ∝Q+ normal-force(B)
        As[fr] = -As[velocity(B)]
        fr ∈ forces-on(B)
```

Fig. 32. Colliding modeled as an encapsulated history

Sometimes all we know about a situation is the particular kind of behavior that will occur. While violating composability, encapsulated histories are the only way to evolve a history in such cases. This particular Encapsulated History describes a perfectly elastic collision with a fixed object in one dimension.

```
Encapsulated-History Collide(B, C, dir)

Individuals:
        B an object, Mobile(B)
        C an object, Immobile(C)
        dir a direction
        E an event

Preconditions:
        (T contact(B,C,dir) start(E))
        (T direction-towards(B,C,dir) start(E))

QuantityCondition:
        (T Motion(B,dir) start(E))

Relations:
        (M A[velocity(B)] start(E)) =  - (M A[velocity(B)] end(E))
        (M velocity(B) during(E)) =  ZERO
        duration(E) = ZERO
        (T contact(B,C,dir) end(E))
```

4.3.2 Relationship to qualitative states

Previous work on representing motion used a *qualitative state* representation for motion [de Kleer, 1975][Forbus, 1981a], an abstraction of the notion of state in classical mechanics. Some of the parameters that would appear in a classical description of state are represented abstractly -- for example, position is represented by a piece of space, and velocity by a symbolic heading. While in classical physics the type of activity is left implicit in the choice of descriptive equations, the qualitative state representation explicitly names the type of activity (FLY, SLIDE, etc). Qualitative states are linked by simulation rules that map a qualitative state into the set of qualitative states that can occur next. Envisioning using such simulation rules is simple; given an initial state use the rules to generate new states, and repeat until no new states are generated. Figure 33 illustrates a physical situation and the envisionment that results. The envisionment can be used to answer simple questions directly, assimilate certain global assumptions about motion, and plan solutions to more complex questions. By examining the relationship between the qualitative state representation and the QP representation we will understand both more clearly.

Consider a process vocabulary comprised solely of motion and acceleration. The limit analysis for a moving object will include only the possibilities raised by dynamics, such as the acceleration due to gravity reversing the velocity of a ball or friction bringing a sliding block to a halt. The possible changes in process structure caused by kinematics -- such as the ball hitting a wall or the block flying off a table - are not predicted within this vocabulary. To include them would require encoding the relevant geometry in such a way that it can be moved out of the preconditions into the quantity conditions. To do this, we must first describe space by a *place vocabulary*,[1] because we must break space up into distinct pieces that can be reasoned about symbolically. We might then try to use the entities in the place vocabulary as elements in the quantity space for position. Then the kinematic changes would be discovered by limit analysis just as the dynamical ones are.

Unfortunately, things are not so simple. First, we need to introduce an ordering between elements of the place vocabulary. (This ordering need not be total; we can use ambiguity in the ordering to represent our lack of knowledge about the precise heading of the moving object). For motion in two or three dimensions this requires specifying a direction, since total orders are only well-defined for one dimension. And because we have specified a direction, we now must also specify the place we are starting from, since that will determine what the neighbors in the position quantity space are. (Consider walking out your front door while throwing a ball up in the air. What the ball might hit changes dramatically.) However, this means the place and direction must be included in the specification of the motion process. If we could sucessfully add such information, an instance of the motion process in this vocabulary would begin to look like a qualitative state for the same collection of places and type of motion. The qualitative simulation rules would thus roughly correspond to a compilation of the limit analysis on this new motion vocabulary.

From this perspective we can see the relative strengths of the two representations. For evolving descriptions of motion the qualitative state representation seems superior, because kinematic constraints are essential to motion. However, simulation rules are an opaque form of dynamics theory -- they do not contain

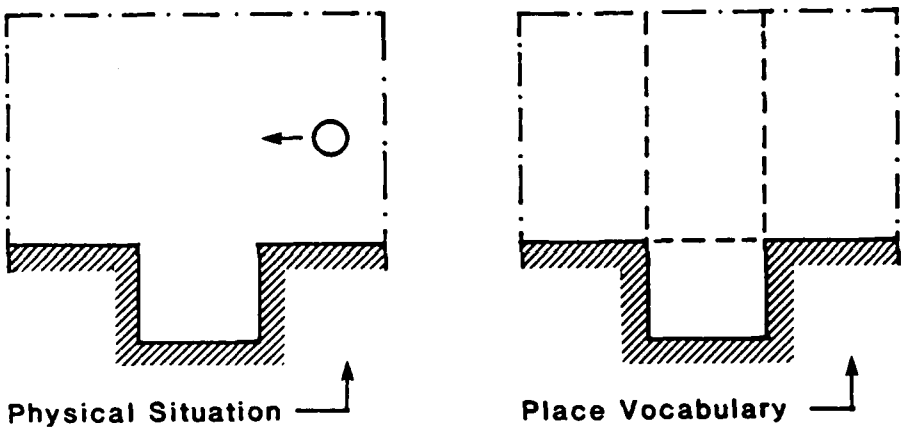
---

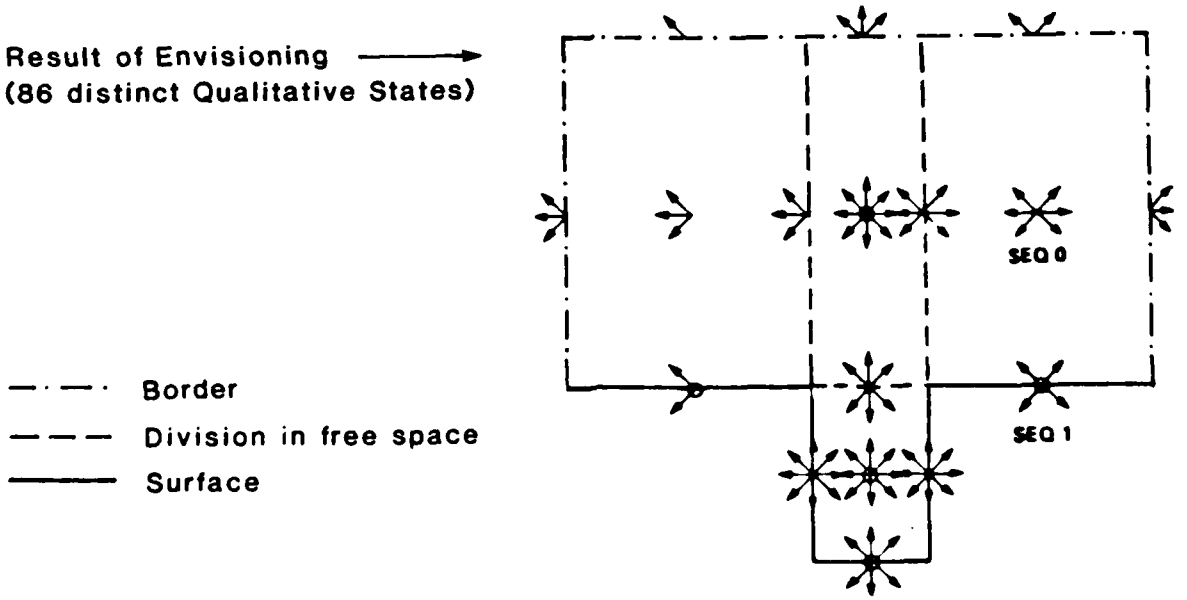
1. [Forbus, 1981a] describes the principles involved and defines a place vocabulary for motion through space in a simple domain.

Fig. 33. Qualitative state description of motion

Consider the ball moving leftwards as depicted below. A qualitative description of space (*place vocabulary*) can be computed from the diagram and the possible ways the ball can move given that initial state are depicted schematically over the places they occur. A detailed description of one state and its relationships between other states is also shown.

```
SEQ0: FLY Space-Region3 (Left Down)
leads to
        PASS Segment12 (Left Down)
        COLLIDE Segment3 (Left Down)

SEQ1: COLLIDE Segment3 (Left Down)
leads to
        SLIDE/STOP Segment3 (Left Down)
        FLY Segment3 (Left Up)
```





the assumptions under which they operate. Thus its "compiled" nature makes the qualitative state representation inappropriate for very simple deductions (where only part of a qualitative state is known), or for more subtle analyses that involve perturbing a system. In particular, the qualitative state representations for motion are not easily composable to form descriptions of more complex systems. The example of section 4.5 illustrates a more subtle analysis of motion made possible by Qualitative Process theory.

4.4 Modeling materials

Let us consider what happens when we pull on something. If it doesn't move, then its internal structure is "taking up" the force (this can happen even if it does move - try hitting an egg with a baseball bat - but we will ignore this case). Three things can happen - (1) it might do nothing (rigid behavior), (2) it might stretch (elastic behavior) or (3) it might break. For a push, the object might again do nothing, it might compress, or it might crumple. We can use the notions of quantity and process provided by QP theory to state these facts. In particular, we can express the changes between these kinds of behavior by creating individual views describing these properties, introducing new elements into the quantity space for forces on an object.

The concepts involved with elasticity can be thought of in terms of applied force versus internal force. If the magnitude of the applied force is greater than that of the internal force, then the length of the object will change. The change in length results in an internal force that will counterbalance the applied force. Three individual views describe the states of an elastic object, either stretched, relaxed, or compressed. Figure 34 illustrates the individual view for elastic objects and their states. To avoid the complications of shape and connectivity, we only model one-dimensional objects that have a fixed end. By convention, forces into an object (pushes) will be negative and applied forces directed outwards (pulls) will be positive.

Imagine that we apply a constant force to an elastic object (with, say, a robot hand under force, rather than position, control). An imbalance between internal and applied forces will result in the length changing. Exactly what occurs depends on the state of the elastic object (stretched, relaxed, compressed), the sign of the applied force, and the relative magnitudes of the forces (the dependence on the sign of the internal force is encoded in the state of the object via the $\propto_Q$ and correspondence.). The four possibilities are stretching, compressing, and two kinds of relaxing. These processes are described in Figure 35.

Of course, objects are not perfectly elastic. If the applied force is very small, objects will often behave rigidly. If too much force is applied an object can break or crush. The rigidity under small forces can be modeled by adding another quantity condition to stretching and compressing. For a partially elastic object the thresholds for compressing and stretching will be called $f_{compress}$ and $f_{stretch}$ respectively. The conditions under which crushing and breaking can be captured similarly by thresholds f_{crush} and f_{break}, which are functions of both the material and the object (to allow for dependence on the shape). The process descriptions for crushing and breaking are however more complex than compressing and stretching because they involve irreversible changes. This requires statements in the relations field that explicitly mention time, turning the description into an encapsulated history rather than a true process. Much of the information that must be included concerns deformations of shape and transformations of one object into several. As with kinematics, these issues are beyond the scope of Qualitative Process theory.

Figure 36 illustrates force quantity spaces corresponding to different kinds of materials. In theory a taxonomy such as this one could be used for classifying a material by applying forces to it and seeing what

Fig. 34. Descriptions of elastic objects
An elastic object stores energy in reversible deformations of shape. The basic view of an elastic object relates the internal force and length, and the other three views describe the states it can be in.

```
Individual-View Elastic-Object

Individuals:
        B an object

Preconditions:
        Elastic-Substance(made-of(B))

Relations:
        Has-Quantity(B, length)
        Has-Quantity(B, internal-force)
        Has-Quantity(B, rest-length)
        D_S[rest-length(B)] = 0
        internal-force(B) ∝_Q+ length(B)
        Correspondence((internal-force(B) ZERO)
                       (length(B) rest-length(B)))

-----------------------------------------------

Individual-View Relaxed

Individuals:
        B an elastic-object

QuantityConditions:
        A[length(B)] = A[rest-length(B)]

-----------------------------------------------

Individual-View Stretched

Individuals:
        B an elastic-object

QuantityConditions:
        A[length(B)] > A[rest-length(B)]

-----------------------------------------------

Individual-View Compressed

Individuals:
        B an elastic-object

QuantityConditions:
        A[length(B)] < A[rest-length(B)]
```

Fig. 35. Stretching, compressing, and relaxing

The continuous changes that can occur to elastic objects constrained by an applied force are described below. The individual views of stretched, compressed, and relaxed are described in the previous figure.

```
process Stretching

Individuals:
        B an elastic object

Preconditions:
        ¬ Position-Constrained(B)

QuantityConditions:
        ¬ Compressed(B)
        A_s[applied-force(B)] = 1
        A_m[applied-force(B)]
           > A_m[internal-force(B)]

Relations:
        Let SR be a quantity
        SR ∝_Q+ (A_m[applied-force(B)]
                 - A_m[internal-force(B)])

Influences:
        I+(length(B), SR)
```

```
process Relaxing-Minus

Individuals:
        B an elastic object

Preconditions:
        ¬ Position-Constrained(B)

QuantityConditions:
 Stretched(B)
      A_m[applied-force(B)]
         < A_m[internal-force(B)]

Relations:
        Let SR be a quantity
        SR ∝_Q+ (A_m[applied-force(B)]
        - A_m[internal-force(B)])

Influences:
        I-(length(B), SR)
```

```
process Compressing

Individuals:
        B an elastic object

Preconditions:
        ¬ Position-Constrained(B)

QuantityConditions:
        ¬ Stretched(B)
        A_s[applied-force(B)] = -1
        A_m[applied-force(B)]
           > A_m[internal-force(B)]

Relations:
        Let SR be a quantity
        SR ∝_Q+ (A_m[applied-force(B)]
                 - A_m[internal-force(B)])

Influences:
        I-(length(B), SR)
```

```
process Relaxing-plus

Individuals:
        B an elastic object

Preconditions:
        ¬ Position-Constrained(B)

QuantityConditions:
Compressed(B)
      A_m[applied-force(B)]
         < A_m[internal-force(B)]

Relations:
  Let SR be a quantity
SR ∝_Q+ (A_m[applied-force(B)]
                 - A_m[internal-force(B)])

Influences:
        I+(length(B), SR)
```

Fig. 36. Materials classified by quantity spaces
Distinct kinds of materials give rise to different quantity spaces because different combinations of processes can occur. This taxonomy should allow a material to be classified by applying forces and observing what kinds of things actually occur.

```
Rigid:
        <no processes affecting length>

Elastic:
        <stretching and compressing apply>

Breakable:
        ZERO < f_break

Crushable:
        f_crush < ZERO

Partially stretchable:
        ZERO < f_stretch

Partially compressible:
        f_compress < ZERO

Brittle:
        f_crush < ZERO < f_break

Partially elastic:
        f_compress < ZERO < f_stretch

Normal:
        f_crush < f_compress < ZERO < f_stretch < f_break
```

---

sorts of behavior result. In a richer model of materials forces along different directions would result in different behavior (such as attempting to bend balsa wood against its grain instead of along the grain) and the effects of plastic deformation would be included.

A classic conundrum is to be able to express in some usable form that "you can pull with a string, but not push with it".[1] This fact can be succinctly stated, at least to a first approximation, using QP theory. First, consider what pushes and pulls are. Both concepts involve one object making contact with another to apply force. Recall that if the direction of the applied force is towards the object it is a push, and if the direction is away from the object then it is a pull. Obviously a push can occur with any kind of contact, but pulls cannot occur when two objects merely touch each other.

Understanding how pushes and pulls are transmitted is fundamental to understanding mechanisms. For a first pass model, consider the notion of *push-transmitters* and *pull-transmitters*. We say an object is a push transmitter if when it is pushed, it will in turn push an object that is in contact with it, in the direction between the two contact points. Pull transmitters can be similarly defined. This particular set of definitions is

---

1. Marvin Minsky, personal communication

obviously inadequate for mechanisms,[2] and is only for illustration. Note also that push-transmitters and pull-transmitters need not be reflexive relations. Rigid objects are an exceptional case:

```
∀ B ∈ object
   rigid(B) ⇒  (∀ c1, c2 ∈ contact-points(B)
                    Push-Transmitter(c1, c2)
                  ∧ Push-Transmitter(c2, c1)
                  ∧ Pull-Transmitter(c1, c2)
                  ∧ Pull-Transmitter(c2, c1))
```

Strings, however, are more complicated. A string can never be a push-transmitter:

```
∀ s ∈ string
 (∀ t ∈ time (T (¬ Push-Transmitter(end1(s), end2(s))
                    ∧ ¬ Push-Transmitter(end2(s), end1(s))) t))
```

But if it is `Taut` it can be a pull transmitter:

```
∀ s ∈ string
(∀ t ∈ time
       (T taut(s) t) ⇒ (T Pull-Transmitter(end1(s), end2(s)) t)
                          ∧ (T Pull-Transmitter(end2(s) end1(s)) t))
```

Now the problem becomes how to define `Taut`. As a first pass, let `ends-distance` be a type of quantity representing the distance between the ends of the string. Then we can define `Taut` as an individual view:

```
Individual View Taut
       Individuals:
              s a string
       Quantity Conditions:
              ¬ Am[ends-distance(s)] < Am[length(s)]
```

This model assumes that only the ends of the string contact other objects - it would fail for a rope hanging over a pulley, for instance. A better model is to divide up the string into segments according to whether or not that part of the string is in contact with a surface. A string is then taut if each segment that is not in contact with a surface is taut:

```
∀ S ∈ string
       (∀ seg ∈ segments(geometry(S)) Free-Segment(seg, S) ⇒ Taut(seg))
              ⇒ Taut(S)
```

This of course ignores the fact that the non-free segments may not be tight, as say for string lying on the floor. A full definition would also require tension along the entire string, but we have strayed far enough from dynamics already.

---

2. Consider for example a rocker arm connected to a pivot or two blocks resting on the floor that are tied to together by a length of string. In the first case a push will be transmitted in a different direction, and in the second case it can be transformed into a pull. Better theories of push and pull transmitters will require representing kinematics in two and three dimensions.

4.5 An oscillator

Dynamical reasoning involves more than just simulation. By analyzing the possible behaviors of a situation we can produce a summary of its behavior and eventual disposition(e.g., [Forbus, 1981a]). In classical physics these analyses are often concerned with stability. Here we will examine a simple situation involving motion and materials, ascertain that it is an oscillator, and perturb it to figure out under what conditions it will remain stable.

Consider the block B connected to the spring S in figure 37. We will model the spring S as device satisfying Hooke's law (see figure 34). Initially we will assume the spring cannot break. To model the position constraint on the spring's length by being rigidly connected to the block and to set the origin of position to the location at which the spring is relaxed, we will assume:

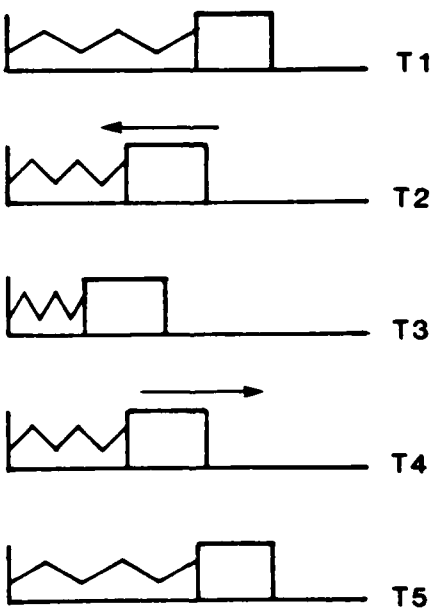
```
length(S) ∝Q+ position(B)
Correspondence((A[length(S)] A[rest-length(S)]) (A[position(B)] ZERO))
```

Suppose the block is pulled back so that the spring is extended. Initially we also assume that the contact between the block and the floor is frictionless. What will happen?

Since initially the spring is extended (i.e., `A[length(S)] > A[rest-length(S)]`), the spring will exert

---

Fig. 37. A sliding block
Here is a system we will analyze to determine what it does and how different factors, such as whether or not there is friction, affect its behavior.



a force. This will in turn exert a force on the block which, since the block is free to move leftwards (S is not immobile), will cause an acceleration. So the initial view and process structures are:

```
VS: {Stretched(S)}
PS: {Acceleration(B, -1)}
```

However, A[velocity(B)] will change from ZERO in an instant (by case 1 of the equality change law), so the process structure will become

```
PS: {Acceleration(B, -1), Motion(B, -1)}
```

Since D_s[position(B)] = -1, by the $\propto_{Q+}$ above we have D_s[length(S)] = -1 as well, and by the correspondence in the definition of elastic objects, D_s[internal-force(S)] = -1 as well. The next limit point is reached when A[length(S)] = A[rest-length(S)], making S relaxed instead of stretched. When this occurs A[net-force(B)] = ZERO, thus ending the acceleration. The motion, however, continues. The process and view structures become:

```
VS: {Relaxed(S)}
PS: {Motion(B, -1)}
```

This state of affairs will last but an instant, since position is still decreasing. As the position moves past ZERO we will have

```
VS: {Compressed(S)}
PS: {Motion(B, -1), Acceleration(B, 1)}
```

The only limit point that can be reached occurs in the quantity space for velocity, i.e. A[velocity(B)] = ZERO. When that occurs the motion will stop, leaving:

```
VS: {Compressed(S)}
PS: {Acceleration(B, 1)}
```

Since acceleration directly influences velocity, this state of affairs will instantly change to:

```
VS: {Compressed(S)}
PS: {Motion(B, 1), Acceleration(B, 1)}
```

The conclusion that the next change results in

```
VS: {Relaxed(S)}
PS: {Motion(B, 1)}
```

with an instantaneous change to

```
VS: {Stretched(S)}
PS: {Motion(B, 1), Acceleration(B, -1)}
```

which lasts for an interval and then yields

```
VS: {Stretched(S)}
PS: {Acceleration(B, -1)}
```

follows in the same way. However, this situation matches the initial situation - the quantity spaces, view and process structures, and D_s values are all the same. Thus we can conclude that an oscillation is occuring. Note

that the view and process structures must be the same, because in principle the preconditions might have changed.

Some of the assumptions made in producing the process history can now be perturbed to examine the effects of different physical models. For instance, suppose the spring is crushable and breakable, as defined previously. Then there are limit points around `rest-length(S)` that correspond to the occurrence of crushing or breaking. It seems crushing must be ruled out by assumption, since the machinery we have developed so far does not allow us to rule it out via contradiction. We can, however, deduce that the spring won't break under the conditions above.

If we can prove that the block will go out no further than when it started, then we can claim that it won't break because it didn't break in the first place. This requires an energy argument. The energy theory we will use is very simple. There are certain types of quantities that are *energy-quantities*, which are qualitatively proportional to certain other quantities and exist whenever they do. Two kinds of energy are kinetic energy and "spring" energy. For every object there is a total energy, which is the sum of its energy quantities. Figure 38 describes systems and energy quantities more formally, and and figure 39 describes sources, sinks, and conservation laws.

Here the system is the mass and spring combination. At time `t1` the block is still but the spring is stretched, i.e.,

```
(M A[velocity(B)] t1) = ZERO
(M A[length(S)] t1) > A[rest-length(S)]
```

which means that

```
(M total-energy(SYSTEM) t1) > ZERO
```

If energy is conserved and there is no influx of energy, then we know

```
∀ t ∈ time
      After(t, t1) ⇒ ¬ (M total-energy(SYSTEM) t) > (M total-energy(SYSTEM) t1)
```

This means that the block can only go out as far as it was at `t1`, since if it ever went out farther we would contradict the previous statement.

Fig. 38. A simple energy theory -- energy & systems

The predicate `Energy-Quantity` asserts that its argument is a quantity type representing a kind of energy. Energy quantities occur as a consequence of objects having particular other types of quantities. The energy of a system is the sum of the energy quantities for its parts.

```
Energy-Quantity(kinetic-energy)

;velocity gives rise to kinetic energy

∀ B ∈ object
        Has-Quantity(B, velocity) ⇒
                (Has-Quantity(B, kinetic-energy)
                 ∧ kinetic-energy(B) ∝Q+ m[velocity(B)]
                 ∧ (A[kinetic-energy(B)] = ZERO
                         ↔ A[velocity(B)] = ZERO))

Energy-Quantity(SpringEnergy)

;an internal force gives rise to "spring" energy

∀ B ∈ object
        Has-Quantity(B, internal-force) ⇒
                (Has-Quantity(B, spring-energy)
                 ∧ spring-energy(B) ∝Q+ m[internal-force(B)]
                 ∧ (A[spring-energy(B)] = ZERO
                         ↔ A[internal-force(B)] = ZERO))

;the total energy of an object is the sum of its energy quantities

∀ B ∈ object
        Has-Quantity(B, total-energy) ∧ Set(energy-quantities(B))
        ∧ ∀ q ∈ quantities(B) Energy-Quantity(q) ⇒ q(B) ∈ energy-quantities(B)
        ∧ total-energy(B) = sum-over(energy-quantities(B))

;the energy of a system is the sum of the energy in its objects

∀ sys ∈ system Set(objects(sys)) ∧ (∀ b ∈ objects(sys) Physob(B) ∨ System(B))
        ∧ Has-Quantity(sys, total-energy) ∧ Set(energy-quantities(sys))
        ∧ (∀ B ∈ objects(sys)
                Subset(energy-quantities(B), energy-quantities(sys)))
        ;ignore converse case (assume all members must be from some part)
        ∧ total-energy(sys) = sum-over(energy-quantities(sys))
```

Fig. 39. A simple energy theory - sources, sinks, and conservation
There are several forms of energy conservation, some stronger than others. The weakest says that if there is no inflow then the energy never increases. The strongest says that in a closed system the energy is always the same.

```
;processes can be sources and sinks w.r.t. a system

∀ pi ∈ process-instance ∀ sys ∈ system ∀ q ∈ quantity-type
        Source(pi, sys, q) ≡
        (∃ B ⊂ objects(sys) Influences(pi, q(B), +1))
        ∧ ¬(∃ B ∈ objects(sys) Influences(pi, Q(B), -1))
;define sinks similarly, ignore cross-flows

∀ pi ∈ process-instance ∀ sys ∈ system
        Energy-Source(pi, sys)
         ≡ (∃ q ∈ quantity Energy-Quantity(q)
                                                ∧ Source(pi, sys, q))
                ∧ (∀ q ∈ quantity Energy-Quantity(q) ⇒ ¬ Sink(pi, sys, q))
;energy sinks are defined analogously

;Conservation laws
;local version - each process conserves energy

∀ pi ∈ process-instance
        (∃ q1 ∈ quantity
                Energy-Quantity(q1) ∧ Influences(pi, q1, -1))
        ↔ (∃ q2 ∈ quantity
                Energy-Quantity(q2) ∧ Influences(pi, q2, 1))

;if you don't kick it it won't get any higher..

∀ sys ∈ system ∀ i ∈ time
        (∀ pi ∈ process-instance
                Energy-Source(pi, sys) ⇒
                 (∀ I1 ∈ during(i) (T Status(pi, INACTIVE) I1)))
        ⇒ ¬ (M total-energy(sys) end(i)) > (M total-energy(sys) start(i))

;more complex version:

∀ sys ∈ system ∀ i ∈ interval
[(∀ pi ∈ process-instance
                Energy-Source(pi, sys) ⇒
                 (∀ I1 ∈ during(i) (T Status(pi, INACTIVE) I1)))
∧ (∀ pi ∈ process-instance
                Energy-Sink(pi, sys) ⇒
                 (∀ I1 ∈ during(i) (T St. us(pi, INACTIVE) I1)))
       ⇒ (M A[total-energy(sys)] start(i)) • (M A[total-energy(sys)] end(i))
```

4.5.1 Stability analysis

To further analyze this system, we must treat the processes that occur as a compound process. We can start by writing an encapsulated history, including properties of the objects taken over the piece of history (a cycle of the oscillator) so defined. We want to perform an energy analysis, so we will include the total energy of the system (`total-energy(SYSTEM)`) and the maximum length of the spring over a cycle

(max-length(S)), since length(S) gives us an indication of "spring energy". We assume the relations for the compound process include:

```
max-length(S) ∝Q+ total-energy(SYSTEM)
correspondence((max-length(S), ZERO), (total-energy(SYSTEM) ZERO))
```

since during each cycle there will be some time during which all of the energy is in the spring. To perform an energy analysis we must re-write any inequalities in the quantity conditions in terms of energy, to wit:

```
Quantity Conditions:
        A[total-energy(SYSTEM)] > ZERO
```

Thus if the total energy of the system ever reaches ZERO during an occurrence of the compound process it will no longer be active, because the total energy of the system is zero only when the spring is relaxed and the block is unmoving. Note that the quantity condition is no longer tied to a particular episode of the encapsulated history. This means that, unlike the encapsulated histories previously encountered, we cannot use this one for simulation. Instead, we use it to analyze global properties of the system's behavior.

We can use the basic QP deductions on this new description to determine the consequences of perturbing the model of the situation in various ways. Each perturbation is represented by a process that influences one of the parameters that determines the energy of the system. For example, suppose friction were introduced into the system. Its effect will be modeled by introducing a new quantity, e-loss(SYSTEM), the energy lost during a cycle. Then D_s[total-energy(SYSTEM)]=-1, and we can deduce, via limit analysis, that the quantity condition above will eventually be false, and so the oscillation will eventually stop. Suppose the system is pumped so that its energy is increasing (i.e., D_s[total-energy(SYSTEM)]=1). While the quantity condition above will remain true, the energy will be continually increasing, which means the force on the spring will increase over time (since during part of the cycle the energy is all in the spring, and the spring energy is qualitatively proportional to the internal force of the spring). If the spring is breakable, then there will be a limit point in the quantity space for the spring's force that will eventually be reached. So the spring could break if the system is frictionless and pumped.

Let us examine in detail what happens if the oscillator is subject to friction, but we pump it with some fixed amount of energy per cycle, as would happen in a mechanism such as a clock. Is such a system stable? We will call the energy lost to friction over a cycle e-loss(SYSTEM) and the energy added to the system over a cycle e-pump(SYSTEM). The only things we will assume about the friction process in the system is that

```
Relations:
        e-loss(SYSTEM) ∝Q+ total-energy(SYSTEM)
        correspondence((e-loss(SYSTEM), ZERO), (total-energy(SYSTEM), ZERO))

Influences:
        I-(total-energy(SYSTEM), A[e-loss(SYSTEM)])
```

The loss being qualitatively proportional to the energy is based on the fact that the energy lost by friction is proportional to the distance traveled, which in turn is proportional to the maximum length of the spring, which itself is qualitatively proportional to the energy of the system, as stated above.

The lower bound for the energy of the system is ZERO, and an upper bound for energy is implicit in the possibility of the parts breaking. The result, via the $\propto_Q$ statement above, is a set of limit points on the

quantity space for e-loss(SYSTEM). If we assume e-pump(SYSTEM) is within these limit points then there will be a value for total-energy(SYSTEM), call it e-stable(SYSTEM), such that:

```
∀ t ∈ cycle
(M A[total-energy(SYSTEM)] t) = (M A[e-stable(SYSTEM)] t))
        ⇒ (M A[e-loss(SYSTEM)] t) = (M A[e-pump(SYSTEM)] t)
```

Note that e-stable(SYSTEM) is unique because $\propto_Q$ is monotonic. If the energy of the system is at this point, the influences of friction and pumping will cancel and the system will stay at this energy. Suppose

```
(M A[total-energy(SYSTEM)] t) > (M A[e-stable(SYSTEM)] t)
```

over some cycle. Then because the loss is qualitatively proportional to the energy, the energy loss will be greater than the energy gained by pumping, i.e., D_s[total-energy(SYSTEM)]=-1, and the energy will drop until it reaches e-stable(SYSTEM). Similarly, if total-energy(SYSTEM) is less than e-stable(SYSTEM) the influence of friction on the energy will be less than that of the pumping, thus D_s[total-energy(SYSTEM)]=1. This will continue until the energy of the system is again equal to e-stable(SYSTEM). Therefore for any particular pumping energy there will be a stable oscillation point. This result is actually a qualitative version of the proof of the existence and stability of limit cycles in the solution of a differential equation. It is surprising just how little information about the system we needed to draw these conclusions, and it will be interesting to see what other results from the classical theory of differential equations can be derived from qualitative information alone.

5. Further consequences

The concepts of Qualitative Process theory provide a representational framework for a certain class of deductions about the physical world. In this section we examine the consequences of this framework for some "higher-level" issues in common sense physical reasoning. Several of these issues arise in reasoning about designed systems, while others cover more general topics.

5.1 Distinguishing oscillation from stutter

As we have seen, envisioning -- generating all the possible behaviors of a system -- can be performed by repeated limit analysis. The result is a linked graph of situations, which can be traversed to form any of the possible histories for the objects that comprise the system. In walking this graph we may find a terminal state (either because the situation is quiescent, because we do not know how to evolve a history past a certain kind of event or because we simply haven't bothered) or we might find a loop. A loop must be summarized if we are to get a finite description of the system's behavior. There are several ways to produce such summaries. In some systems the major regularity is spatial, in which case we would produce descriptions like "the ball is bouncing around inside the well"[Forbus, 1981a]. Another type of concise summary is possible when a system is oscillating, since there is a pattern of activity that occurs over and over again.

While oscillation in the physical system results in loops in the envisionment, there are other circumstances that give rise to loops as well. Consider the situation illustrated in figure 40. Initially there are two flows, one from A to B and the other from B to C. What can happen? Limit analysis reveals three alternatives, corresponding to each of the flows stopping individually and to both ending simultaneously (see figure 41). In the cases where one flow stops before the other, the flow that continues will decrease the amount, and hence pressure, so the other flow will start again. These cycles of activity do not correpond to physical oscillations; they are an artifact of the qualitative physics. A better description of this behavior is that the change in level "follows" the other change. In other words, we have a decaying equilibrium. We will call the behavior represented by these degenerate cycles *stutter*. How can we distinguish stutter from true oscillation?

Physically an oscillation requires that the system have some form of inertia or hysteresis. This means that, at least for some part of the system, when the cause of the change stops acting, the change will continue

---

Fig. 40. Three container example
Suppose we have three containers partially filled with water and connected by pipes, as shown below. If we assume the water moves slowly, what can happen?

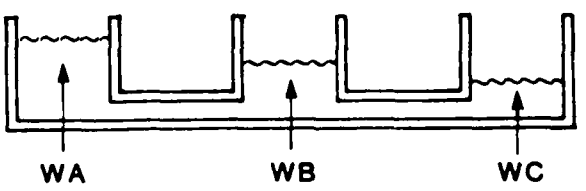
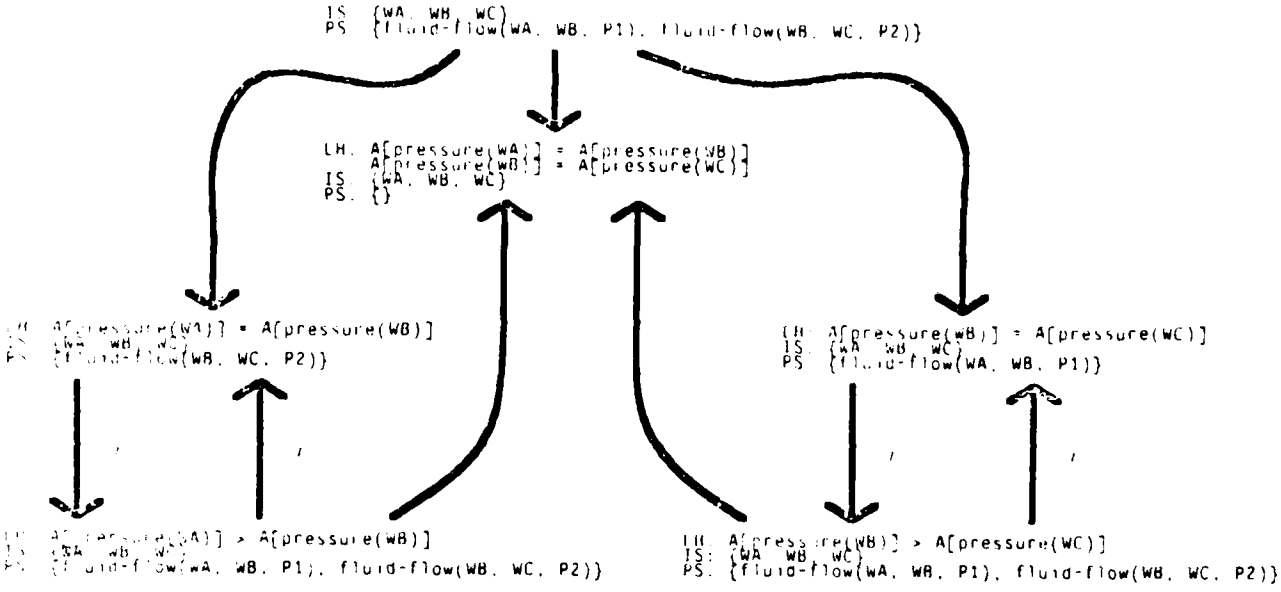


Fig. 41. Stutter in fluid flow
This graph of transitions between process structures contains two cycles, neither of which correspond to physical oscillations. For simplicity, we ignore the possibility of the contained liquids vanishing as a result of the flows.



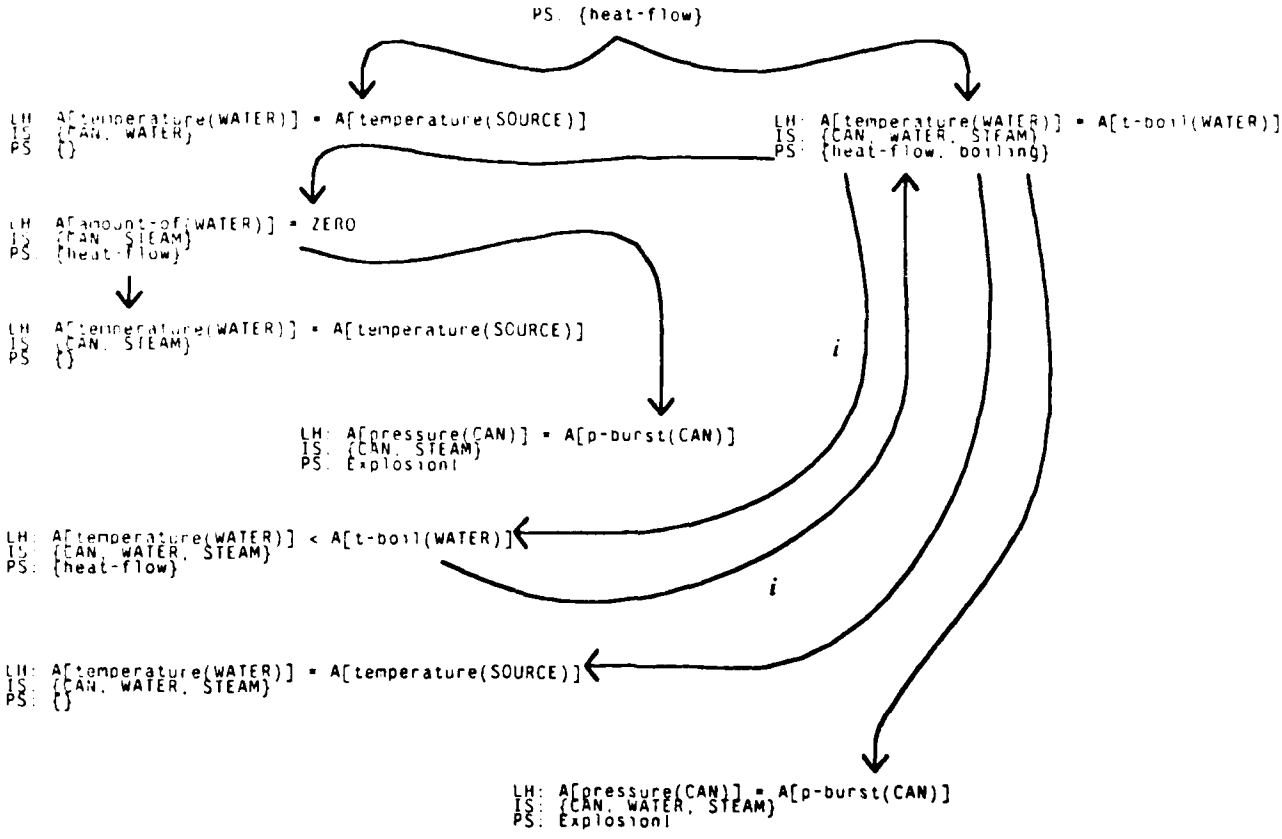
for a while afterwards. A real oscillation will therefore include process episodes that last over an interval of time, whereas stutter -- a kind of "mythical oscillation" -- will only include process episodes that last but an instant.

Fortunately the equality change law provides a way of distinguishing these cases. In the previous transition diagram, for example, the transitions marked with an "i" take place in an instant. Therefore we have two instances of stutter, corresponding to the two fluids participating in a decreasing equilibrium.

A similar phenomena occurred in the boiler model presented earlier (section 4.2). Figure 42 depicts the envisionment. Note that if `t-boil(WATER)` rises faster than `temperature(WATER)` the boiling will stop. Since this change in the inequality relationship between the quantities is a change from equality, by case 1 of the equality change law it will occur in an instant. This in turn means that `t-boil(WATER)` was only influenced for an instant. When the boiling stops only the heat flow is acting, so `temperature(WATER)` will rise, and thus by case 2 of the equality change law the return to equality will occur in an instant. Therefore this cycle is an instance of stutter as well, corresponding to a rising equilibrium.

Being able to distinguish stutter from oscillation means we can write rules that summarize the process history. For example, when stutter occurs we can note the D_s values for the quantities involved and assert that one kind of change is "following" another, a decaying or rising equilibrium. Informal observations

Fig. 42. Stutter in the boiler example
The temperature and pressure will be continuously increasing in the boiler, but unless the changes in the links marked "i" are recognized as occuring in an instant, the system will appear to be oscillating.



---

suggest that novices in a domain often confuse stutter and oscillation, and even experts who describe the situation as a decaying or rising equilibrium are able to reconstruct the view of stutter as an oscillation. These informal observations need to be examined in the light of empirical data, but if true it may be useful in testing QP theory as a psychological model.

5.2 Causal Reasoning

We use causality to impose order upon the world. When we think that "A causes B", we believe that if we want B to happen we should bring about A, and that if we see B happening then A might be the reason for it. Causal reasoning is especially important for understanding physical systems, as noted in [Rieger & Grinberg, 1977], [de Kleer, 1979]. Exactly what underlies our notion of causation in physical systems is still something of a mystery.

Consider the representations used in physics. Typically equations are used to express constraints that hold between physical parameters. A salient feature of equations is that they can be used in several different ways. For example, if we know X = A + B, then if we have A and B we can compute X, but also if we have X and A we can compute B. It has been noted that in causal reasoning people do not use equations in all possible ways [diSessa, 1983][Riley, 1981]. Only certain directions of information flow intuitively correspond to causal changes. I propose the following *causal directedness hypothesis:*

> *Changes in physical situations which are perceived as causal are due to our interpretation of them as corresponding either to direct changes caused by processes or propagation of those direct effects through functional dependencies.*

This section will attempt to justify that hypothesis.

First, I propose that causality requires some notion of mechanism.[1] Imagine an abstract rectangle of a particular length and width. If we imagine a rectangle that is longer, it will have greater area. There is no sense of causality in the change from one to the other. If however we imagine the rectangle to be made of some elastic material and we bring about the increased length by stretching it, then we are comfortable with saying "the increase in length causes the area to increase". QP theory asserts that processes are the mechanisms that directly cause changes. The quantities that can be directly influenced by processes are in some sense independent parameters, because they are what can be directly affected. All other quantities are dependent, in that to affect them some independent parameter or set of independent parameters must be changed. This suggests representing the relationships between parameters for causal reasoning in terms of functions rather than constraint relations.

Some examples will make this clearer, as well as emphasizing that the point is not academic. In generating explanations of physical systems, it is often useful to characterize how the system responds to some kind of change (this variety of qualitative perturbation analysis was invented by de Kleer, who calls it *Incremental Qualitative Analysis*, abreviated IQ). One way to perform IQ analysis is to model the system by a constraint network, in which the relationships are modeled by "devices" that contain local rules that enforce

---

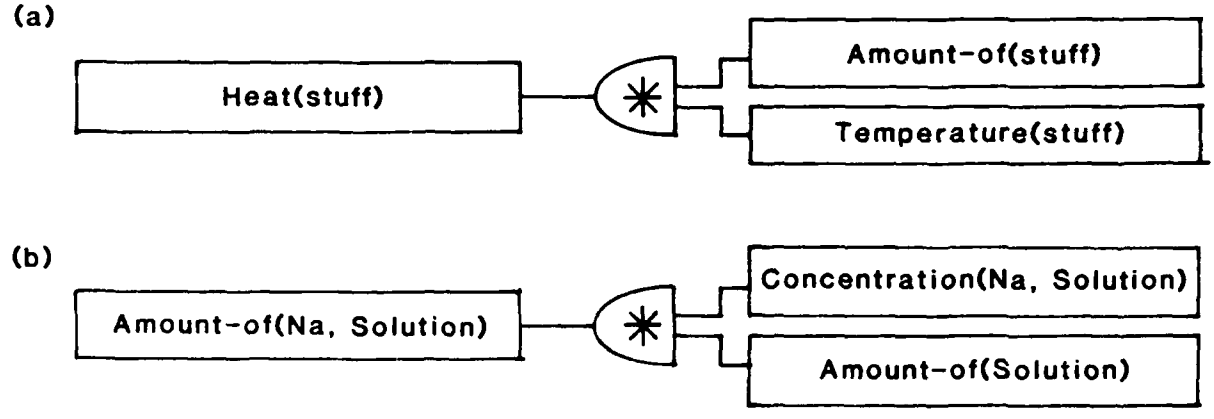
1. In its most general form, this proposal is not new ([Bunge, 1979] surveys various proposals concerning the nature of causality). For example, [Heise, 1975] proposes *operators* as a mechanism that underlies all causal relations. The proposal presented here is more specific.

the desired semantics.[2] The values of quantities are modeled by the sign of their change - increasing, decreasing, or constant. To perform an analysis, a value corresponding to a hypothesized change is placed in a cell of the constraint network representing the system. The rules associated with the constraint network are then used to deduce new values for other cells in the network. The propagation of information models the propagation of changes in the system, with dependency relationships between the cell values corresponding to causal connections. For example, if the value of cell A was used to deduce the value of cell B, we would interpret this as "The change in A caused the change in B". Figure 43 illustrates fragments from two

---

Fig. 43. Constraint representation of relationships
In the constraint networks below, the boxes (cells) denote quantities. The relationship between the parameters is expressed by a multiplier constraint connecting them.
(a) is drawn from the model for a piece of "stuff" used to represent a student's understanding of heat exchangers.
(b) is drawn from a model of a kidney to be used in explaining the syndrome of inappropriate secretion of anti-diarretic hormone (SIADH).

```
Correct causal argument: "The increasing heat causes the temperature to rise"
Incorrect causal argument: "The increasing heat causes the amount of fluid to rise"
```



---

2. These examples are drawn from systems implemented in CONLAN [Forbus, 1981c], a constraint language. The graphical notation for constraint networks is similar to logic diagrams, except that "terminals" are given explicit names and the "devices" are multi-functional. The technique described here is a simplification of de Kleer's algorithms, which are more subtle. However, the models in [de Kleer, 1979] sometimes used directional rules rather than constraint laws, although no theoretical criteria was provided for selecting which direction in a constraint law is appropriate for causal reasoning.

different models.[3] The top fragment states that heat is the product of the temperature of the "stuff" and the amount of "stuff", and the bottom fragment is the definition of sodium concentration in a solution.

In building a causal argument it is possible to reach an impasse - a quantity receives a value, but no further values can be computed unless an assumption is made. The safest assumption is that, unless you know otherwise, a quantity doesn't change. The problem lies in determining which quantity to make the assumption about. Suppose we assume that the amount of stuff is constant. Then we would conclude that an increase in heat causes an increase in temperature, which makes sense. However, suppose we assume instead that the temperature remains constant. We are left with the conclusion that an increase in heat causes the amount of stuff to decrease! Barring state changes, this does not correspond to our ideas of what can cause what. In the second fragment the problem is more serious - increasing sodium will cause the amount of water to increase, if the rest of the kidney is working as it should! To do this requires a complicated feedback mechanism that is triggered by detecting an increased sodium concentration, not by the definition of concentration itself.

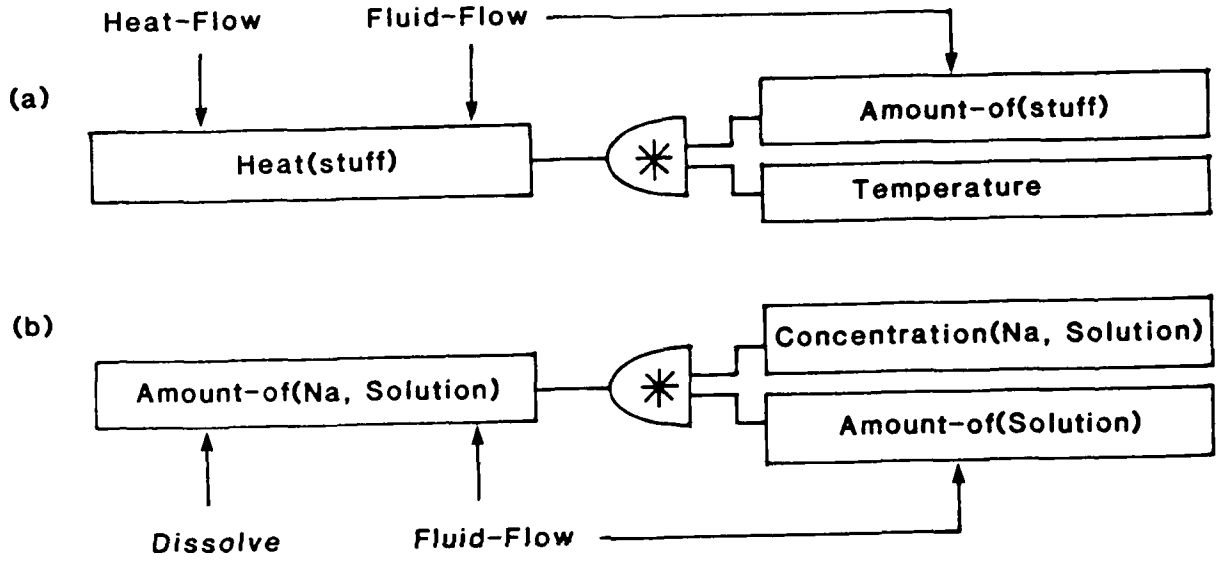
The problem lies in the ontological impoverishment of the constraint representation. Temperature and concentration are not directly influenced by processes (at least in most people's naive physics) - physically they are *dependent* variables, and thus are not proper subjects of assumptions. Amount of stuff, on the other hand, can be directly affected, so assuming it does not change will avoid generating ill-formed causal arguments. Figure 44 illustrates.

Of course, the proper assumptions to make concern what processes are active and how influences are resolved. *If we do not explicitly represent processes, we can only assume facts about quantities.* If we assume a quantity is constant and later discover that assumption is wrong, we are left in the dark about why that assumption was wrong. For example, if the amount of stuff turns out not to be constant, we can look for fluid flows or state changes to explain why it isn't. Since processes tend to have more than one effect, there is some chance that the contradiction can lead to discovering more about the system rather than just being a bad guess.

---

3. These model fragments are drawn from an attempt to implement the model of a student's understanding of a heat exchanger (described in [Williams, et al, 1983]) and an early version of the kidney model described in [Asbell, 1982].

Fig. 44. Model fragments with possible proceses
Here are the models from the previous figure with the quantities annotated with the (likely) processes that might affect them. Note that certain quantities (temperature, concentration) cannot be directly changed. These are *dependent* quantities, and should not be the subject of assumptions in building causal arguments.



5.3 Qualitative proportionalities revisited

The previous section proposed that functional dependence is central to the kind of "incremental" causality that people find useful in reasoning about the physical world. As discussed previously, developing a theory of observation should be a goal of naive physics. One use of observation is to interpret measurements in terms of theories ([Forbus, 1983b]), but another role for observation is in developing physical theories. While this problem has been studied before (c.f. [Langley, 1979]), the target representations have been equations. As a result the learning procedure has relied on numerical data and cannot build theories around weaker information. Learning constraint laws also differs from learning causal connections. As noted previously, an equation carries only part of what we know about a domain. Constructing a learning theory for physical domains will require ways to learn process descriptions and causal connections.

One way to learn about a system is to "poke" it and see what it does. The observed behavior can be used to make conjectures about causal connections between the parts of the system, and further experiments can be made to test the conjectures. This requires some notation to express the local causal connections conjectured on the basis of these simple observations. This requirement helped motivate the definition of $\propto_Q$ (see Section 2), which asserts that a functional dependence exists between two quantities. If whenever

parameter A in a system is increased parameter B increases, the result can be expressed as:

$$B \propto_{Q+} A$$

A physical explanation for the dependence comes from writing the $\propto_Q$ within the scope of a predicate, an individual view, or a process.

More powerful statements about a system or domain will require extensions of $\propto_Q$. To see what is involved, consider the analogous situation of learning how an old-fashioned typewriter works.[1] If the space bar is pushed, the carriage will move to the left. This is analogous to the kind of statement that can be made with $\propto_Q$. But lots of other things can happen to move the carriage, including all of the letter keys. Thus it would be useful to be able to state that we know all of the potential influences (at least, within the current grasp of the situation) on some particular parameter. Suppose also that we just wanted to move the paper up without changing anything else. The return bar would move the paper up, but before doing so would return the carriage to the right. Being able to say there are no (known) intervening parameters is then also a useful ability.

To see how these notions can be expressed, consider the collection of $\propto_Q$ relations that hold at some instant in time. For any quantity, the $\propto_Q$ statements relevant to it can be thought of as a tree with the dependent quantity at the root and the "independent" quantities at the leaves. A plus or minus denotes the sense of the connection (whether or not it will reverse the sign of the change in the input). Thus

$$Q_0 \propto_Q Q_1$$

only specifies that Q_0 is on some branch "above" Q_1.

Figure 45 illustrates such a dependency tree. Suppose we are trying to cause Q_0 to change. If we don't want to change Q_2, then Q_3 or Q_1 are our only choices. We need a way to express that (at least within our knowledge of the situation) there are no intervening parameters. To say this, we use

$\propto_Q$-immediate(Q_0, Q_1)

which can be modified by + or - as before. $\propto_Q$-immediate adds a single link to the tree of dependencies. Another problem is to find all the ways to bring a change about, or to prove that changing one thing won't cause a change in some other quantity of interest. We do this by stating that a particular collection of quantities together "closes off" the tree -- there will be exactly one quantity for each branch. Our notation will be

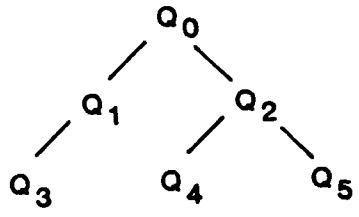
$\propto_Q$-all(<quantity>, <plus-set>, <minus-set>)

which means that there is a function which determines the quantity, relies on the quantities in the two sets solely, is strictly increasing in its dependence on the quantities in the plus-set, and is strictly decreasing in its dependence on the quantities in the minus-set. If a quantity is not mentioned in a $\propto_Q$-all statement, then either (1) it is irrelevant to the quantity of interest, (2) it depends on some quantity in the $\propto_Q$-all statement (above the slice of the tree it represents), or (3) some quantity in the $\propto_Q$-all statement depends on it. By ruling

---

1. This is not proposed as a serious example because the quantity definitions and $\propto_Q$ would apply only in some very abstract sense.

Fig. 45. A tree of functional dependencies



---

out the other two possibilities, independence can be established.

As a rule $\propto_Q$ statements will not hold for all time. In the typewriter analogy, imagine the carriage at the end of its travel - Hitting the space bar will no longer result in movement. More to the point, consider Q_0 given by:

$$Q_0 = (a - b * Q_2) * Q_1$$

Note that:

$$\begin{aligned} &\text{if } a > b * Q_2,\ Q_0 \propto_{Q+} Q_1 \\ &\quad a = b * Q_2,\ \neg\, Q_0 \propto_Q Q_1 \\ &\quad a < b * Q_2,\ Q_0 \propto_{Q-} Q_1 \end{aligned}$$

In the case of equality, Q_0 and Q_1 are not related at all, and in the other two cases the sign of the function connecting them is different. Thus the collection of $\propto_Q$ statements that are true for a system can vary as a function of the values of the quantities, which is why they usually appear within some individual view or process. The idea of a *mode* of a system in "real" physics roughly corresponds to particular process and view structures which hold during the system's operation.

5.4 Differential qualitative analysis

The idea of a comparison in IQ analysis suggests a complementary qualitative reasoning technique. IQ analysis concerns the relationship between two situations, one of which is the direct result of things happening in the other. Another case of interest concerns situations that are just slightly different from one another -- an "alternate world" interpretation. For instance, we often have an idea of the different consequences that would result if something were changing a bit faster -- if we put the heat up on the stove the water in the kettle would boil sooner, and if our arm were quicker the serve would have been returned. Such inferences are essential in debugging and monitoring execution of plans that involve physical action, and in performing sensitivity analyses to evaluate a proposed design. The language needed to express such

conclusions is in part the same as that used in IQ analysis -- amounts are either the same, increased, decreased, or indeterminate as compared with the old situation. Answering these kinds of questions will be called *differential qualitative* analysis.

Let us consider a situation S_1. If we can get a new situation S_2 by changing a single ordering in S_1 or by changing the status of a single process or view instance in S_1, we will call S_2 an *alternate* of S_1. There are two kinds of changes that may occur as a result of perturbing S_1. The first kind are changes in quantities, as noted above. Second, the process history for the situation itself may change, apart from any changes made to define S_2 in the first place. An example would be punching a hole in the bottom of a kettle, which could let all the water drain out before it it boils. Even changes in orderings can lead to further changes in the histories of the individuals involved -- e.g., if we reduce the intensity of a flame but still turn it off in five minutes, boiling may not occur.

Let DQ(q, S_1, S_2) for some number n be the sign of the difference between two alternate situations S_1 and S_2. Then the inequality order between them defines DQ values, as follows:

$$(M\ n\ S_1) > (M\ n\ S_2) \leftrightarrow DQ(n, S_1, S_2) = -1$$

$$(M\ n\ S_1) < (M\ n\ S_2) \leftrightarrow DQ(n, S_1, S_2) = 1$$

$$(M\ n\ S_1) = (M\ n\ S_2) \leftrightarrow DQ(n, S_1, S_2) = 0$$

The inequality orderings for instants must of course be extended to apply over intervals. For equality this is simple:

$$\forall\ n_1, n_2 \in \underline{number}\ \forall\ i \in \underline{time}$$
$$(M\ n_1\ i) = (M\ n_2\ i) \equiv \forall\ i_1 \in during(i)\ (M\ n_1\ i_1) = (M\ n_2\ i_1)$$

For the other cases the choice is less clear. The strongest version of `greater-than` is having it hold over every instant in the interval:

$$\forall n_1, n_2 \in \underline{number}, i \in \underline{interval}$$
$$(M\ n_1\ i) > (M\ n_2\ i) \equiv$$
$$(\forall\ i_1 \in during(i)\ (M\ n_1\ i_1) > (M\ n_2\ i_1))$$

however, the following will also suffice:

$$\forall\ N_1, N_2 \in \underline{number}, i \in \underline{interval}$$
$$(M\ n_1\ i) > (M\ N_2\ i) \equiv$$
$$[(\exists\ i_1 \in during(i)\ (M\ N_1\ i_1) > (M\ N_2\ i_1))$$
$$\wedge\ (\forall\ i_1 \in during(i)\ \neg\ (M\ N_1\ i_1) < (M\ N_2\ i_1))]$$

A version of < for intervals may be similarly defined.

An episode in a parameter history has several numbers associated with it. The relationships between these numbers allow new DQ values to be determined. The first number is *rate*, e.g., the D_m of the quantity the parameter history is about. The second number is the `duration` of the interval associated with the episode. The third number is the difference in the value measured at the beginning and end of the interval, which we will call the *distance*.

How are these numbers related? Intuitively we know that if the rate increases or decreases, the duration of time will decrease or increase, or the distance the value moves will increase or decrease for the same duration. Implicit in this simple intuition is the restriction that the rate is constant during the interval, i.e., that the function defining the change of the quantity is linear and time invariant. This often is not the

case, so we must require that either the beginning or the end of the two episodes being compared are the same. If we apply DQ analysis only to alternate situations as defined above this restriction will be satisfied.

With these assumptions, the DQ value of the distance is just the product of the DQ values of the rate and duration. Thus we can draw conclusions such as "If the rate is higher then over the same time the distance traveled will be greater." and "If the duration is shorter and the rate is the same then the distance traveled will be less." These inferences are illustrated in figure 46.

The direct historical consequences of these changes can be characterized by their effects on limit analysis. Consider a collection of quantity hypotheses for a p-component. Recall that each hypothesis concerns a possible change in state, brought about by non-zero D_s values that cause changes in quantity conditions. Suppose a particular quantity hypothesis is chosen as representing what actually occurs. This means the change it stands for happens before the changes represented by the other hypotheses. If in an alternate situation this hypothesis has an increased duration (a DQ value of 1) or one of the other limit hypotheses has a decreased duration (a DQ value of -1), then in fact a different change could occur. Once again, the weak nature of our information prevents us from actually deciding if a different change would occur -- but we at least know that it is possible in these circumstances.

Fig. 46. Differential qualitative analysis
Differential qualitative analysis answers questions about how a situation would change if parts of it are perturbed.

```
;definition of distance
∀ S ∈ episode distance(Q, S) = (M A[Q] end(S)) - (M A[Q] start(S))

Suppose we have alternate situations S1 and S2, with a
quantity Q in both of them.

DQ[distance(Q, S), S1, S2] = DQ[rate(Q, S), S1, S2] * DQ[duration(S), S1, S2]

Then assuming time(start(S1)) = time(start(S2)),

        DQ[rate(Q, S), S1, S2] = 1 ∧ DQ[duration(Q), S1, S2] = 0
                ⇒ DQ[distance(Q, S), S1, S2] = 1

        DQ[rate(Q, S), S1, S2] = -1 ∧ DQ[distance(Q, S), S1, S2] = 0
                ⇒ DQ[duration(S), S1, S2] = 1

i.e., "If N is going faster then it will get farther in the same time" and "If N
is going slower it will take longer to go the same distance."
```

5.5 Measurement interpretation

One role of a naive physics is to provide explanations for observed events. A particularly simple form of this problem is to interpret the changes one sees when taking a quick look at a system, explaining observed parameter changes in terms of the processes that cause them. Consider the three containers illustrated in figure 47. If we see that the level of water in G is dropping, then there are several possible explanations, but if we also note that the level in F is dropping as well, then both changes must be caused by water flowing from F to G and then from G to H (assuming we know all the pipes there are and that liquid flow is the only kind of process that can occur). Here we will present a theory of measurement interpretation that solves this class of problem. Appropriate notions of measurement and interpretation are defined, including an account of error due to limited resolution. Design considerations for interpretation algorithms are discussed, although a description of the implemented algorithm is postponed until chapter 9.

5.5.1 Measurements

First we must specify what kinds of things can be observed in principle and then add further conditions to specify what can be observed in fact. We start by assuming that a collection of individuals is known. The closed world assumption that these individuals are the only (relevant) individuals is called the *armchair assumption*. We will also assume the existence of a partial decision procedure for determining whether or not relationships defined outside QP theory (such as `Fluid-Connection`) hold, in order to confirm preconditions. To state that we have ascertained whether or not a fact is true via observation, we will write:

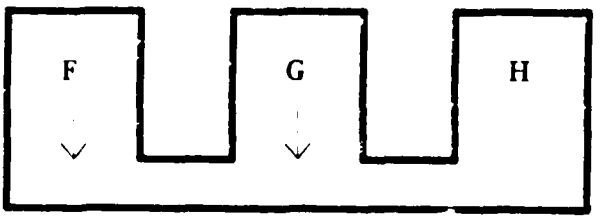
`Observed(`⟨*fact*⟩, ⟨*time*⟩, M`)`

where M is the instrument (such as our eyes) used in the observation.

Within the QP ontology, the kinds of facts that can be observed are inequalities, the values of signs,

---

Fig. 47. A measurement interpretation problem
Suppose we see the level of water in G dropping. What could be causing that? What could be causing the level of water in G and in F to drop?



and occurrences of processes. We will say

Observable(Q, B, *M*)

when quantity (or part of a quantity such as its D_s) Q of object B can be observed with instrument *M*.

The criteria for a type of process being observable often reduces to the observability of a particular kind of quantity and the uniqueness of that process (with respect to the reasoner's process vocabulary) in influencing it. A change in position, for example, is by definition the result of motion. Thus whenever we see a change of position we are seeing the result of a motion process. A process may also be observable because it produces some other observable effect, such as bubbles which exist in turbulent flow and boiling. In either case, we will say

Observable-PI(PI, *M*)

if PI can be observed to be active or inactive with instrument *M*. We will say

Measured(Q, B, t, *M*, *<value>*)

when measuring Q(B) with *M* at (during) time t yields the value given. In keeping with the quantity space representation, measuring a number or magnitude yields an inequality and measuring a sign yields one of -1, 0, or 1. For example, the statments

Measured(A_S[pressure], WA, t_1, *M*, 1)
Measured(A[pressure], WA, t_1, *M*, A[pressure(WA)] = A[pressure(WB)])

say that the pressure of WA was measured to be greater than zero and equal to the pressure of WB at time t_1. Measuring derivatives will be discussed shortly.

We wish to consider a wide variety of instruments as measuring means, such as eyes and gauges. A comprehensive theory of error lies outside the present discussion, but modeling the potential for error due to limited resolution is straightforward. The essence of the limitation is that when two things are "very close" a particular *M* might not distinguish them (for signs, we are seeing if the number is "very close" to the special value ZERO). For each measuring means, object, and quantity type, let there be a function Omin such that two values are considered *distinguishable* if and only if the magnitude of their actual difference is greater than Omin. In other words,

Distinguishable(Q, B, Q1, t, *M*)
  ↔ m[(M Q(B) t) - (M Q1 t)] > Omin(Q, B, *M*)

Omin will be chosen according to the particular physics and instrument being modelled; this particular form is chosen for simplicity.[1] A measured equality might be wrong due to limited resolution:

---

1. In particular, this form of Omin is simpler than *just noticable difference* in psychophysics, because the latter also depends on the value of the quantity. A taxonomy of possible forms for Omin is outside the scope of this discussion.

```
Measured(Q, B, t, M, (= Q(B) Q1))
        ⇒ [(M Q(B) t) = (M Q1 t)
          ∨ ¬ Distinguishable(Q, B, Q1, t, M)]
```

and if we measure a difference, then there really is a difference:

```
Measured(Q, B, t, M, (> Q Q1)) ⇒
       [(M Q(B) t) > (M Q1 t)
       ∧ Distinguishable(Q, B, Q1, t, M)]
```

A similar statement can be made for <. In measuring signs we are examining inequalities with `Q1 = ZERO`, so lack of resolution will show up as a sign value of `0`.

Measuring change is particularly important. First consider changes in a quantity over some interval. We must distinguish the values of the same quantity measured at two different times, so the relation we are looking for depends on the quantity type, the object, the measuring means, and an interval. We will say

```
D-distinguished(Q, B, I, M)
```

(read "differentially distinguished") exactly when

```
m[(M Q(B) start(I)) - (M Q(B) end(I))]
    > 0min(Q, B, M)
```

Since we are using Allen's ontology for time, an instant is simply a very short interval. Thus our criteria for observing changes over intervals can serve for measuring derivatives. However, capturing lack of resolution becomes more complicated:

```
Measured(Ds[Q], B, I, M, 0)
        ⇒ (∀ t ∈ during(I) (M Ds[Q(B)] t) = 0)
          ∨ ¬ D-distinguished(Q, B, I, M)
          ∨ ¬ Constant-Sign(D[Q(B)], I)]
```

This axiom says that if we measure the change of a quantity over an interval and find it constant, then either it really isn't changing (i.e., D_s value of `0`), it is changing too slowly to be distinguishe ' , our instrument *M*, or it has been changing faster than we can measure. Even when the measured value is non-zero, the sign may not have been constant over the interval:

```
Measured(Ds[Q], B, M, I, <1 or -1>)
 ⇒ [(∀ t ∈ during(I) (M Ds[Q(B)] t) = <1 or -1>)
                ∨ ¬ Constant-Sign(D[Q(B)], I)]
                ∧ D-distinguishable(Q, B, I, M)
```

Like the zero case, this axiom states (via the dis .. ction) that we cannot tell if the quantity was changing rapidly during the interval. However, it also says that if we do measure a non-zero D_s value, then the change really is distinguishable. The extension of distinguishability to intervals of non-zero duration allows us to say, for example, that while we cannot immediately see the effect of evaporation on the level of water in a glass, if we looked longer we could.

5.5.2 Interpretations

An interpretation must explain what is causing the changes that are occuring (including the special case of nothing changing). In QP theory processes are the only causes of changes, so an interpretation will include assumptions about the status of the process instances that occur between the individuals. Since more than one process can influence a quantity, interpretations must also include assumptions concerning influence resolutions. An interpretation must be *internally consistent*, *externally consistent*, and *sufficient*. Internally consistent means an interpretation assigns at most one status to any process instance and at most one D_s value to any quantity. Externally consistent means that the status assignments and D_s values assigned are consistent with the measurements. Sufficient means that every measured D_s value is explained.[1]

Some additional structure on interpretations will prove useful. A *unit cause hypothesis* (abreviated UCH) is a partial interpretation that forces the assignment of a particular D_s value to be consistent with its measured value. Any interpretation which satisfies the three criteria above will be a collection of UCHs, one for each D_s measurement, that is internally consistent. The *p-influencers* of a quantity is the set of process and view instances that can possibly influence that quantity, directly or indirectly. The *influencers* of a UCH is the subset of the p-influencers that are active in that UCH. In addition to the status assumptions that determine the influencers, a UCH with conflicting influences must include an assumption about their resolution. As noted above, for direct influences this will take the form of an inequality between (perhaps sums of) the influences.

5.5.3 Computational issues

There are two possible ways to organize the search for interpretations. One way is to search through the possible UCH's for each measurement to find a globally consistent collection. Finding the possible UCH's for a quantity is simple. The set of p-influencers can be computed from the process descriptions associated with the process instances, and each possible subset of influencers can be checked to see if it can be resolved consistently with the measured value. However, the potential number of UCH's can be quite large. Suppose we measure a quantity and find it is increasing. Then if we have p process instances that can provide a positive influence and N that can provide a negative influence, there are

$$(2^P - 1) \cdot (2^N)$$

possible UCHs. In practice this number will actually be much smaller, since the process instances are usually not independent. For example, a fluid path cannot have flows going in both directions at once because their quantity conditions would conflict. Also, the number of consistent interpretations will almost always be much smaller than the product of the number of UCHs since processes typically influence more than one quantity, providing mutual constraint. These facts suggest that we organize the search around the space of status assignments to process instances instead. If we wish a *total* interpretation we can use the entire collection of

---

1. By contrast, an interpretation in de Kleer's QUAL [1979] is a collection of device states and incremental changes in quantities, the latter assumed to occur sequentially in "mythical time". Despite profound ontological differences, the principles defining interpretations presented here are inspired by his work.

process instances, but if we want a minimal interpretation to explain the measurements we can just use the union of the p-influencers for the measured D_s values. Any collection of status assignments that cannot be consistently extended by assumptions about influence resolutions to provide a UCH for each D_s measurement can be thrown out, and each extension found is a valid interpretation.

Several kinds of knowledge can be used to prune the search space. Process instances that correspond to observable processes could have their status determined directly by observation, or indirectly by ascertaining the truth of their preconditions and quantity conditions. For example, if we can see that a valve in a fluid path is closed, then that fluid path is not aligned and no flows can occur through it.

Once a collection of status assumptions is chosen, it must be extended to form a collection of UCHs. There are several ways to accept or rule out a UCH. If the set of influences can be resolved then the UCH will stand or fall according to whether or not the resolved D_s value and the measured value agree. Again, this can require domain-specific information: we do not expect that evaporation will immediately cancel out the effect of pouring water into a cup. Distinguishability provides a means of ruling out small changes. For example, we can say:

```
∀ w ∈ contained-liquid ∀ pi ∈ process-instance
Influencers(Level(w)) = {pi} ∧ Process(pi) = Evaporation
 ⇒ ¬ D-Distinguished(Level, w, eyeball-time, eyes)
```

which will rule out evaporation as the sole explanation for why we are seeing the level of water in a glass fall.[1]

What if no consistent interpretation exists? Following [Davis, 1984], we view the analysis as relying on simplifying assumptions that must be re-analyzed in such cases. The assumptions, ordered in increasing certainty, are:

1. Any facts used in pruning are correct
2. The measurements are correct
3. The armchair assumption is correct
4. The process vocabulary is complete and correct

Other orderings are of course possible. Ultimately a global order on catagories of facts will probably prove inadequate, since our strength of belief can vary strongly on items within a catagory. For example, when the measuring means is indirect and the domain familiar, we often trust our theories more than the measurements. The opposite is true if the measurements are direct (sensory) and the domain unfamiliar.

---

1. Informal observations indicate that people appear to use the following *ineffectuality heuristic* -- if the result of a process instance is not distinguishable, assume it isn't acting. The intuition appears to be that its effect won't make that much of a difference anyway (unless the physical structure of the situation leads you to believe there are alot of them!). This heuristic prunes the search space of process instances enormously, and it seems likely that correct use of this assumption is a mark of an expert in a domain.

6. Practice

The proof of a pudding is in the eating, and the test of an AI theory is whether it can be used to construct programs that perform the type of reasoning the theory is about. However, evaluating a theory by examining a program that embodies it can be tricky. There are many engineering decisions which must be made in constructing a complex artifact, and many of these decisions are constrained more by the implementation technology than by the theory being tested. Perhaps the best view is to consider a program as a piece of experimental apparatus, a means of exploring a theory. Sometimes critical experiments can be set up, but more often programs provide a means of generating phenomena.

The program GIZMO is written in this spirit. GIZMO is a vehicle for exploring and developing QP theory. The fact that it works indicates the theory is not patently wrong, but the distance between it and a program whose competence matches ours in reasoning about the physical world is wide enough to warrent caution in making claims. Given that caveat, I will say that GIZMO has made me fairly optimistic that QP theory will play an important role in constructing powerful and useful qualitative reasoning systems.

The purpose of the next few chapters is to describe the representations and algorithms used in GIZMO in sufficient detail that a competent AI hacker could construct a "QP engine" with only moderate difficulty. The particular data structures and programs used will not be described. Since my goals were exploratory, speed was almost always sacrificed to flexiblity. Someone constructing "industrial-grade" naive physics systems will find little direct guidence therein. Although I have some ideas on the subject (see section 12.3.1), developing efficient inference techniques for QP theory remains a problem for the future.

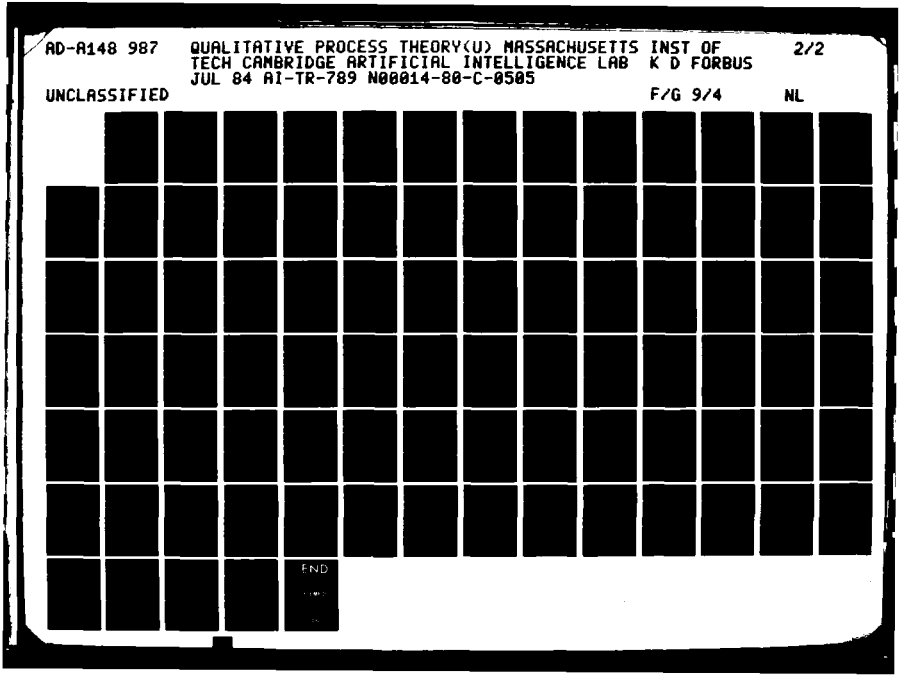
GIZMO is a large program, consisting of over 1,300 Zetalisp functions organized into 60 files. It includes representations of objects and quantities, algorithms for performing the basic QP deductions, a language for describing domain models, a "one-look" measurement interpretation algorithm, and an envisioner. The following chapters describe each of these components in detail. Each part and each example in these chapters is fully implemented and runs.

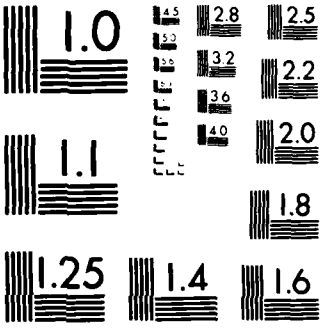
Sadly, theory often outstrips practice. Limitations of time and computing resources have prevented GIZMO from being a complete embodiment of QP theory. Of the history theory presented in sections 2.7 and 3.7, only slices and situations were implemented, since they are all that has been needed for the particular reasoning tasks examined. Neither encapsulated histories nor hierarchial process descriptions have been implemented. No significant external theories have been provided to establish preconditions, nor have any extension theories been added to resolve ambiguities. Nevertheless, the reader should find some indication of the utility of the theory by examining what has been implemented.

Chapter 7 describes the details of the representations of objects, quantities, individual views and processes used in GIZMO. Chapter 8 describes the language GIZMO provides for writing domain models, and presents simple vocabularies for reasoning about fluids and motion that will be used in later examples. Chapter 9 describes the algorithms used to implement the basic QP deductions. Chapter 10 describes a "one-look" measurement interpretation algorithm, and Chapter 11 describes the envisioning algorithm, including how the frame problem is handled.

The reader who is uninterested in algorithmic or implementation details would do well to skip chapters 7 and 9, focusing instead on chapter 8 to get a sense of what the domain models look like and the examples in chapters 10 and 11 to see what GIZMO can do with them. To ease the path for readers unfamiliar with Lisp, infix notation will be used when possible. The algorithms are expressed in a restricted

English format to be accessible to the general computer science reader. For variety, however, lisp-style prefix notation will be used when describing implementation details or domain models. The only chapter which assumes any familiarity with Lisp or AMORD-like languages is Chapter 8, since the domain models do include rules and occasional pieces of programs. The particular inference engine used, called *DEBACLE*, is described in the appendix.


AD-A148 987
QUALITATIVE PROCESS THEORY(U) MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB K D FORBUS
JUL 84 AI-TR-789 N00014-80-C-0505
2/2
UNCLASSIFIED
F/G 9/4
NL
END



7. Representing objects, quantities, and processes

This chapter describes the next level of detail for the representation of objects, quantities, and processes. First we examine how objects, slices, and situations are represented. Second, we examine the representations of quantities and relationships between them. Finally, we summarize the facts associated with individual views and processes.

7.1 Objects

Objects are represented by *individuals*. The criteria for what constitutes an individual will in general depend on the domain being represented. However, instances of processes are always considered to be individuals.[1] There are two distinct but related notions of existence. The first is *logical existence*, which simply means that it is not inconsistent for there to be some state of affairs in which a particular individual exists. A square circle is something which logically cannot exist. The second notion is *physical existence*, which means that a particular individual actually does exist at some particular time. Clearly an individual which physically exists must logically exist, and an individual which logically cannot exist cannot ever physically exist. An example of an individual which logically exists but which (hopefully) never physically exists is the arsenic solution in my coffee cup.

The predicate `Individual` is used to indicate that its argument is an individual. Being an individual means that its properties and relationships with other things can change with time, and that it may not always physically exist.

The relation `Exists-In(i, t)` indicates that individual `i` exists at, or during, time `t`. The import of this relationship is the creation of a *slice* to represent the properties of `i` at `t`. A slice of an object `B` at time `t` is denoted by `at(B, t)`. All predicates, functions, and relationships between objects can apply to slices to indicate their temporal extent. This means instead of refering to the value of a number or part of number by `M`, i.e.,

```
(M f(n) t)
```

we will say

```
f(at(n, t))
```

An interesting issue which did not arise in Hayes' original treatment of histories concerns the interaction between existence and predication. What is the truth of a predicate applied to a slice when the individual is not believed to physically exist at the time corresponding to that slice? Allowing all predicates to be true of an individual when it doesn't physically exist has the problem that every fact `f` which depends on a predicate must now also be explicitly justified by a statement of existence, such as

---

1. For convenience, instances of individual views are also represented as individuals in the implementation. Given their syntactic similarity, this choice allows most of the code and axioms implementing them to be shared, as we will see in section 7.3.

⟨predicate⟩(at(obj, t)) ∧ Exists-in(obj, t) ⇒ f

rather than just

⟨predicate⟩(at(obj, t)) ⇒ f

To avoid this, we simply indicate that the truth of certain predicates which depend on physical existence imply that the individual does exist at that time, i.e.

Predicate(at(obj, t)) ⇒ Exists-in(obj, t)

This allows the implications of the predication to be stated simply, while also providing a constraint on existence that is useful for detecting inconsistencies. However, care must be taken when specifying taxonomic constraints, such as saying that an object is either rigid or elastic. If we simply assumed

∀ sl ∈ slice Taxonomy(Rigid(sl), Elastic(sl))

we would be asserting the existence of the object at the time represented by that slice, since one of the alternatives of the taxonomy must be true. These statements must always be placed in the scope of some implication which will guarentee existence, such as

∀ sl ∈ slice Physob(sl) ⇒ Taxonomy(Rigid(sl), Elastic(sl))

to avoid inappropriate presumptions of physical existence.

Situations describe a collection of objects being reasoned about at a particular time. A situation simply consists of a collection of slices corresponding to the objects that exist in it. In GIZMO, the name of the situation serves to name the time of a slice; no other model of time is implemented. Each situation has the set of *next-situations* and *previous-situations*, which consist of the situations that can lead to it and the situations it can result in, respectively. The decision of what individuals should be considered together as a situation is left totally to the user; automatic segmentation into p-components (see section 3.7) is not implemented.

7.2 Quantities

We begin by describing the relationships between the parts of a number, and describe the particular facts about inequalities that GIZMO uses. We then examine the language used to describe functional dependencies, including the laws of qualitative proportionalities, correspondences, and explicit functions. Finally we describe adders, which are used to sum direct influences and compute other aggregate properties.

7.2.1 Numbers

The various relationships between the parts of a number, its magnitude, amount, and sign, are described by several axioms. For signs, the following facts hold:

∀ sn ∈ sign
    Taxonomy(sn = -1, sn = 0, sn = 1)

The Has-Value relationship is used in the implementation to say sign ⟨x⟩ takes on the value ⟨val⟩, i.e.,

(Has-Value ⟨s⟩ ⟨val⟩)

For magnitudes,

∀ mg ∈ magnitude
Taxonomy(mg = ZERO, mg > ZERO)

For numbers,

∀ n ∈ number
s[n] = 0 ↔ m[n] = ZERO

7.2.2 Inequalities

First, we assume that the normal inequality relationships > and < are defined over numbers with their usual properties, i.e., they are transitive, antireflexive, and asymmetric.

Next, we describe the interaction between inequality relationships and the parts of a number. The first time any pair of numbers N_1, N_2 is compared (i.e., any equality or inequality statement about them or their parts is placed in the database, whether or not that statement is believed), the following logical constraints are installed:

Taxonomy($N_1 < N_2$, $N_1 = N_2$, $N_1 > N_2$)
$s[N_1] = -1 \wedge s[N_2] = 0 \Rightarrow N_1 < N_2$
$s[N_1] = -1 \wedge s[N_2] = 1 \Rightarrow N_1 < N_2$
$s[N_1] = 0 \wedge s[N_2] = 1 \Rightarrow N_1 < N_2$
$s[N_1] = 0 \wedge s[N_2] = -1 \Rightarrow N_1 > N_2$
$s[N_1] = 1 \wedge s[N_2] = -1 \Rightarrow N_1 > N_2$
$s[N_1] = 1 \wedge s[N_2] = 0 \Rightarrow N_1 > N_2$
$[(N_1 > ZERO \wedge N_2 > ZERO) \Rightarrow (m[N_1] > m[N_2] \leftrightarrow N_1 > N_2) \wedge (m[N_1] = m[N_2] \leftrightarrow N_1 = N_2)]$
$[(N_1 < ZERO \wedge N_2 < ZERO) \Rightarrow (m[N_1] > m[N_2] \leftrightarrow N_1 < N_2) \wedge (m[N_1] = m[N_2] \leftrightarrow N_1 = N_2)]$

Notice that conclusions about equality fall out as a consequence of the implications for the inequality cases and the first taxonomy statement. Constraints that describe the relationships between their magnitudes (called here m_1 and m_2, respectively) and ZERO are also installed:

$m_1 = ZERO \wedge \neg m_2 = ZERO \Rightarrow m_1 < m_2$
$m_1 = ZERO \wedge m_2 = ZERO \Rightarrow m_1 = m_2$
$\neg m_1 = ZERO \wedge m_2 = ZERO \Rightarrow m_1 > m_2$

To avoid confusion in the implementation between the ordering relationships and the Lisp predicates >, <, and =, Greater-Than is used instead of >, Less-Than is used instead of <, and Equal-To is used instead of =.

7.2.3 Qualitative proportionalities

For a given quantity, its *constrainers* is the set of quantities to which it is either qualitatively proportional or inversely qualitatively proportional, i.e.,

$$\forall\ Q_1,\ Q_2 \in \underline{\text{quantity}},\quad Q_2 \in \text{constrainers}(Q_1) \leftrightarrow (Q_1\ \propto_{Q+}\ Q_2 \vee Q_1\ \propto_{Q-}\ Q_2)$$

Of course, the members of this set will change over time with changes in the view and process structures. The set of *constrainees*, the set of quantities which are qualitatively proportional or inversely proportional to it, is defined similarly. To simplify printing and reading routines, $\propto_{Q+}$ is written as `qprop+`, or just `qprop`, while $\propto_{Q-}$ is written `qprop-`. $\propto_{Q}$ is not used by the implementation at all.

7.2.4 Correspondences

Recall that a correspondence describes how quantity spaces are linked across a qualitative proportionality. For example, the statement

```
Correspondence ((A[internal-force(band)], ZERO)
                (A[length(band)], A[rest-length(band)]))
```

says that the internal force exerted by an elastic band is zero exactly when the length of the band equals its length at rest. Let N_1 be a number (typically, the amount of a quantity) with constrainers $c_1, c_2, \ldots, c_n$. A correspondence consists of a list of pairs,

$$((N_1\ v_1)(c_1\ v_2)(c_2\ v_3)\ldots(c_n\ v_{n+1}))$$

For a correspondence to be applicable, at most one of the constrainers can be different from the value indicated by the correspondence (the v_i's). Without loss of generality, assume that all but c_1 are the same, i.e.,

$$c_i = v_{i+1} \quad \text{if} \quad \neg\ i = 1$$

and that each c_i is in the constrainers by virtue of being the amount of some quantity Q_{i+1}. Then the relationship between N_1 and v_1 will be

$$\begin{aligned}
&c_1 = v_2 \Rightarrow N_1 = v_1\\
&c_1 > v_2 \Rightarrow\\
&\qquad [(Q_1\ \propto_Q\ Q_2 \Rightarrow N_1 > v_1)\\
&\qquad\ \wedge (Q_1\ \propto_{Q-}\ Q_2 \Rightarrow N_1 < v_1)]\\
&c_1 < v_2 \Rightarrow\\
&\qquad [(Q_1\ \propto_Q\ Q_2 \Rightarrow N_1 < v_1)\\
&\qquad\ \wedge (Q_1\ \propto_{Q-}\ Q_2 \Rightarrow N_1 > v_1)]]
\end{aligned}$$

Should there be only one constrainer, the outer $\Rightarrow$'s above can be replaced by $\leftrightarrow$.

7.2.5 Explicit Functions

The consequences of having the same function relate the parameters of two distinct individuals can be expressed via a correspondence. Given two objects o_1, o_2 and functions f_1, f_2,...f_n that map from o_1 and o_2 to parameters of them, suppose that

$$\texttt{constrainers}(f_1(o_1)) = \{f_2(o_1), \ldots f_n(o_1)\}$$
$$\wedge\ \texttt{constrainers}(f_1(o_2)) = \{f_2(o_2), \ldots f_n(o_2)\}$$

For example, o_1 and o_2 might be contained liquids, with f_1 being `pressure` and f_2 being `level`. If the function which contributes the constrainers is the same in both cases (i.e., is explicitly named by `function-spec` and is the same for both o_1 and o_2), then we can conclude

$$\texttt{Correspondence}((A[f_1(o_1)], A[f_1(o_2)]),$$
$$(A[f_2(o_1)], A[f_2(o_2)]),$$
$$\ldots, (A[f_n(o_1)], A[f_n(o_2)]))$$

7.2.6 Simple algebra

While algebraic manipulations are not part of basic QP theory, sometimes it proves convenient to write a few algebraic statements to express relationships in domain models. For example, in describing how the flow rate of a liquid-flow changes, we will write

```
(Q= flow-rate (- (pressure source)  (pressure dest)))
```

This algebraic statement is interpreted as syntactic sugar for combinations of qualitative proportionalities and correspondences. First, we assume the left-hand side of a `Q=` (read "quantity equals") statement is always a quantity, and the right-hand side is a simple binary combination of numbers or quantities. We will use the symbol `-->` to indicate that the expression on the left hand side is translated into the expression on the right-hand side. The translations are as follows:

```
(Q= ⟨a⟩ ⟨b⟩) --> (= (A ⟨a⟩) (A ⟨b⟩)) ∧ (= (D ⟨a⟩) (D ⟨b⟩))
(A (+ ⟨a⟩ ⟨b⟩)) --> (+ (A ⟨a⟩) (A ⟨b⟩))
(D (+ ⟨a⟩ ⟨b⟩)) --> (+ (D ⟨a⟩) (D ⟨b⟩))
(A (- ⟨a⟩ ⟨b⟩)) --> (- (A ⟨a⟩) (A ⟨b⟩))
(D (- ⟨a⟩ ⟨b⟩)) --> (- (D ⟨a⟩) (D ⟨b⟩))
(= ⟨a⟩ (+ ⟨b⟩ ⟨c⟩)) --> (+rel ⟨a⟩ ⟨b⟩ ⟨c⟩)
                          ∧ (∝Q+ number-of(⟨a⟩) number-of(⟨b⟩))
                          ∧ (∝Q+ number-of(⟨a⟩) number-of(⟨c⟩))
(= ⟨a⟩ (- ⟨b⟩ ⟨c⟩)) --> (-rel ⟨a⟩ ⟨b⟩ ⟨c⟩)
                          ∧ (∝Q+ number-of(⟨a⟩) number-of(⟨b⟩))
                          ∧ (∝Q- number-of(⟨a⟩) number-of(⟨c⟩))
number-of((A ⟨a⟩)) --> ⟨a⟩
number-of((D ⟨a⟩)) --> ⟨a⟩
```

The flow-rate expression above, for example, would result in the assertions:

```
(+rel (A flow-rate) (A (pressure source)) (A (pressure dest)))
(+rel (D flow-rate) (D (pressure source)) (D (pressure dest)))
(∝Q+ flow-rate (pressure source))
(∝Q- flow-rate (pressure dest))
```

The import of `+rel(a, b, c)` is:

```
a = ZERO ↔ [(b = ZERO ∧ c = ZERO)
                    ∨ (m[b] = m[c] ∧ Opposite-sign(b, c))]
```

where

```
∀ N1, N2 ∈ number
Opposite-sign(N1, N2) ↔
        [(s[N1] = 1 ∧ s[N2] = -1)
         ∨ (s[N1] = -1 ∧ s[N2] = 1)
```

The implications of `-rel(a, b, c)` are:

```
Correspondence((a, ZERO)(b, c))
Correspondence((a, ZERO) (b, ZERO) (c, ZERO))
```

7.2.7 Adders

Addition is the only arithmetic operation supported by the implementation, due to its importance in computing the combined effects of direct influences. Conceptually, an adder has three parts, a *sum*, which is a number, and two sets, *inputs* and *minus-inputs*, which at any particular time contain the numbers that are the contribution (positive or negative) to the sum. The sets are determined by making a closed-world assumption, in that the collection of numbers explicitly known to be elements of the set at some time are assumed to be all the members of the set.

An adder, whose sole purpose is to compute the sign of the sum, works as follows. The inputs and minus-inputs are sorted into three sets according to their signs - a negative number in the minus-inputs set, for example, becomes a member of the positive contributions. If all of these sets are empty or only the zero set is non-empty, then the sum is assumed to be zero. If either the positive or negative set has members, with the other set being empty, then the sign of the sum is that of the set which has members. If both the positive and negative sets are not empty, then inequality information is gathered in an attempt to settle the sign. Suppose that the positive set has three members while the negative set has two. Then if for each member of the negative set a distinct member of the positive set can be found that is greater than it, then the sign of the sum must be positive because the positive contributions will more than cancel the negative ones. Similarly, if the positive and negative sets are of the same size and enough equality information is known to set up a one to one mapping between the sets, then the sign of the sum is concluded to be zero. Only inequalities connecting numbers are used, i.e., inequalities between algebraic combinations or functions of numbers are not supported.

Associated with each quantity is a special adder, called its *influence adder*. Whenever a quantity is directly influenced, each direct influence is considered to be a member of the appropriate input set (i.e., a member of the plus inputs if the influence is positive and a member of the minus inputs if the influence is negative). If there are any plus or minus inputs to the influence adder, then the derivative of the quantity is constrained to be equal to the sum of the influence adder.

7.3 Individual views and processes

Since individual views and processes are so much alike, in the implementation they are treated as specializations of a more abstract type, the *conditionalized description*. There are three classes of facts about conditionalized descriptions to be considered, the facts which define one, the facts which hold when an instance of one is created, and the facts which hold when an instance of one is active. We examine each in turn.

When a conditionalized description is defined, facts describing its various components are created. The obvious parts are the individuals, preconditions, quantity conditions, relations, and (for processes) the influences portions of the specifications. An additional distinction is made between two classes of facts in the relations field, those which are to be asserted when an instance is found (the *creation facts*), and those which are to be asserted when the instance is active (called, appropriately, the *activation facts*). Creation facts declare individuals introduced by process and view instances and quantities representing their continuous parameters (such as flow rates). The creation facts also include indexing information, such as the role each individual plays in the instance.

In particular, recall that the `individuals` field specifies what type each individual must be. Here are some important facts asserted when a slice of an instance of a conditionalized description `cdi` is created for situation `sit`:

```
Taxonomy(Status(at(cdi, sit), Active), Status(at(cdi, sit), Inactive))
Exists-in(cdi, sit) ↔ <type specifications of slices for cdi's individuals>
Status(at(cdi, sit), Active) ⇒ Exists-in(cdi, sit)
```

The first fact simply states that an instance must be either active or inactive. The second fact says that an instance exists in a situation exactly when the individuals it occurs between exist and are of the appropriate type (since the predicates need not always be true). Notice that the instance cannot exist when one of the individuals it applies to doesn't, since in that case the type predicates for the non-existent individuals won't be true for that slice. The third fact enforces the constraint that the instance must exist at any time it is active.

The activation facts are simply the facts explicitly mentioned in the relations and influences fields. They are justified by the slice of the instance being active, i.e., they will be believed whenever the instance is active.

8. Domain models

This chapter describes GIZMO's language for writing domain models and two examples of domain models. While better domain models are being developed, at this writing these models are the only ones that have been run on several examples. They are quite crude from the standpoint of being reasonable-fidelity portions of a naive physics, but have been crucial in developing and debugging the ideas.

QP theory provides a framework that partially specifies a representation language for dynamical theories. But it does not specify all of the details of that language, and there are always a number of ways for such a language to be implemented. The next section describes the syntactic constructs of GIZMO's particular language for describing domain models. The language provides little insulation from the underlying DEBACLE (and LISP) foundation; given that this language is still under active development this is a feature rather than a bug. The last two sections describe the models of fluids and motion that will be used to illustrate the algorithms and certain dynamical issues in the next three chapters.

8.1 Specifying the models

Several kinds of information must be specified in developing a domain model using QP theory. First, one must define the various kinds of quantities, predicates, and individuals that exist in situations of that domain. Second, the processes and individual views which comprise its view and process vocabularies must be defined. Finally, a means of specifying a particular problem is needed. The constructs that serve each of these purposes will be introduced in turn.

8.1.1 Defining constants and facts

Logical constants, such as `WATER`, are defined with the `DefineConstant` form. General facts about a domain are expressed with the `DefFact` form.

8.1.2 Defining types of quantities

The form

```
(DefQuantity-Type <type>)
```

states that *<type>* is a function which maps from individuals to quantities and from slices to values. Quantity types must be declared in advance of their use, since terms that describe quantities are instantiated differently from other terms (see section 14.6 for details).

8.1.3 Defining predicates

To specify a one-place predicate, we will use the form

```
(DefPredicate <name> . <body>)
```

where *<name>* is the name of the predicate and *<body>* is a collection of statements and rules. The meaning of `DefPredicate` is that the statements are held to be true as a consequence of the predicate being true, and the

rules are defined in the scope of an environment where the variable `?self` is bound to the predicate's argument. For example,

```
(DefPredicate student
        (has-low-income ?self)
        (rule (?f (:true (graduate-student ?self)))
              (assume (-> ?f (Frantic (at ?self thesis-deadline)))
                      'Fact-of-Life)))
```

would expand into:[1]

```
(rule (:intern (student ?self))
      (assert (-> (student ?self) (has-low-income ?self)))
      (rule (?f (:true (graduate-student ?self)))
            (assume (-> ?f (Frantic (at ?self thesis-deadline)))
                    'Fact-of-Life)))
```

8.1.4 Defining types of individuals

Certain types of predicates are used to specify types of individuals. On the whole, these predicates can be treated as above. There are two additional complications. First, an individual often has parts, and it is useful to specify these explicitly, especially if some work must be done to update them (such as the set of pieces of stuff which comprise the contents of a place). This function is provided by extending the syntax of `DefPredicate` slightly, to wit:

`(defEntity` *⟨name⟩ ⟨parts-list⟩ . ⟨body⟩*`)`

---

1. For those unfamiliar with AMORD or RUP, a short note on rule syntax is in order.

Pattern variables are symbols with a "?" prefix. A rule trigger has three parts, a *statement variable*, a *condition*, and a *pattern*. In the trigger pattern

```
(?f (:true (graduate-student ?self))),
```

`?f` is the statement variable, `:true` is the condition, and `(graduate-student ?self)` is the pattern. A rule can have a list of triggers. The body of the rule consists of code which is executed whenever a collection of facts which match the triggers and satisfy their conditions are found. Several kinds of conditions are provided to allow rules to be used for different purposes. The conditions `:true` and `:false` are satisfied when the matched fact is first believed to be true or false, respectively. The conditions `:whenever-true`, `:whenever-false`, and `:whenever-change` cause the rule to be run every time the belief in the fact becomes true, false, or changes at all. Since most of the work in the rules occurs by adding clauses to the TMS, the most used condition is `:intern`, which means that the rule can be run as soon as the fact enters the database, whether or not it is believed.

The body of the rule is executed in an environment where the statement variables are bound to the facts which matched the trigger patterns and the variables in the trigger patterns are bound to the expressions provided by the match. The forms `assume` and `assert` are interpreted specially; the first argument is the fact that will be placed in the database, with the appropriate substitutions made for the pattern variables. The second argument names the type of assumption, information which is useful for debugging and backtracking.

⟨parts-list⟩ is simply a list of entries. Each entry contains the name of the part and a specification of what type of thing it is. We will describe what can be found in the parts list shortly.

The second complication is that, as described in the previous chapter, the interactions between the truth of instances of this predicate and existence of the argument must be made explicit. The implementation has slices and individuals as distinct syntactic types, which in turn are distinct from the representation of constants. Statements which are occurances of a predicate defined by `defEntity` will do different things according to the type of its argument. If the argument of a predicate is an individual, the additional rule

```
(rule (?f (intern (Exists-In ?self ?s))) ;?s is a situation
      (assume (-> (and (⟨name⟩ ?self) ?f) (⟨name⟩ (at ?self ?s)))
              'Existence-Law))
```

will be created to ensure that each slice inherits the proper type.[1] Notice that if the truth of the predicate will change over time it must not be asserted of the individual, only of those slices for which it is true. If the argument is a slice, then (as mentioned previously) the truth of that predicate on it is tantamount to believing the individual exists at that time, i.e.,

```
(assume (-> (⟨name⟩ (at i sit)) (Exists-in i sit)) 'Existence-Law)
```

If the argument is already known to be neither an individual nor a slice, then the predicate cannot be true of it, i.e.

```
(assume (not (⟨name⟩ ?self)) 'Existence-Law)
```

The parts list is used to specify properties of a type of individual. An entry in the parts list consists of a name and a specification. The specification describes the type of thing the part is, such as a quantity. A property that is a set is indicated by the specification

```
(set-of ⟨part-relation⟩)
```

where *⟨part-relation⟩* is the name of the relationship that specifies that something is in that set. For example,

```
(defEntity Grad-Student ((net-worth quantity)
                         (creditors (set-of has-creditor)))
        (not (?self creditors NIL)))
```

says that a graduate student has a financial net worth and a non-empty set of creditors.

Certain individuals are best specified as functions of other individuals, such as the collection agent assigned by a creditor to the case of a particular debtor. We will also use `defEntity` to specify the properties of this type of individual by allowing the name to be a compound expression, and allowing `--` in the specifications to denote that some of the arguments of the name plays a particular role for that individual.

---

1. For hackers, it should be noted that for efficiency reasons the actual trigger form for this rule is

```
(?self exists-in ?s)
```

since DEBACLE, like RUP, only indexes assertions by their first element and presumably there will be many more statements of existence than statments about any particular individual. See the appendix for more details.

Here, for example, we might say

```
(defEntity (collection-agent ?debtor ?creditor) ((victim (== ?debtor))
                                                 (oppressor (== ?creditor)))
        (Hassles ?self ?debtor)
        (Reports-To ?self ?creditor))
```

to indicate that a collection agent reports to the creditor and that the victim of the collection agent is the debtor.

8.1.5 Defining relationships

The `defRelation` form provides a means of defining relationships as opposed to type predicates, individuals, or compound individuals. Its syntax is:

```
(defRelation ⟨form⟩ . ⟨body⟩)
```

`defRelation` is mainly syntactic sugaring for DEBACLE rules, with ⟨*form*⟩ being the rule's trigger pattern and ⟨*body*⟩ being the body of the rule. Unlike normal DEBACLE rules, however, certain syntactic transformations, including those described in Section 7.2.6, are performed on the body. To continue our (grim) example, we might describe the `Hassles` relationship as follows:

```
(defRelation (Hassles ?creditor ?debtor)
        (or (Phones-Late-At-Night ?creditor ?debtor)
            (Accosts-on-Street ?creditor ?debtor)
            (Sends-Threatening-Letters ?creditor ?debtor))
        :justification Preditor-Definition)
```

The `:justification` keyword indicates the symbol (in this case, `Preditor-Definition`) that should be used as an assumption type for assertions made by rules created from this definition (See the DEBACLE appendix for an explanation of how assumption types are used).

8.1.6 Defining processes and individual views

Processes are defined with the `defProcess` form:

```
(defProcess ⟨form⟩ . ⟨keyword-list⟩)
```

where ⟨*form*⟩ is the syntax of the relationship which indicates an instance of the process occurs between its arguments, and ⟨*keyword list*⟩ takes the form of a list alternating between keywords (`individuals`, `preconditions`, `quantity-conditions`, `relations`, and `influences`) and the specification of that part of the process.

The best way to explain `defProcess` is to look at an example. Here is a heat-flow process we will use later on:

```
(defProcess (Heat-Flow ?src ?dst ?path)
            Individuals ((?src (Physob ?src)
                               (Has-Quantity ?src Heat))
                         (?dst (Physob ?dst)
                               (Has-Quantity ?dst Heat))
                         (?path (Heat-Path ?path)
                                (Heat-Connection ?path ?src ?dst)))
            Preconditions ((Heat-Aligned ?path))
            QuantityConditions ((Greater-Than (A (temperature ?src))
                                              (A (temperature ?dst))))
            Relations ((Local flow-rate (Quantity flow-rate))
                       (Q= flow-rate (- (temperature ?src)
                                        (temperature ?dst)))
                       (Greater-Than (A flow-rate) zero))
            Influences ((I+ (heat ?dst) (A flow-rate))
                        (I- (heat ?src) (A flow-rate))))
```

The first element of the process specification provides a unique name for instances of the process by substituting the individual bindings in the form provided. In this case the form is

`(Heat-Flow ?src ?dst ?path)`,

and an instance might be

`(Heat-Flow Stove Can Burner)`.

The individuals specification provides the binding environment for the rest of the description; no free variables are allowed. The type of each individual must be provided (e.g., `Physob` and `Heat-Path`) to constrain candidate collections of objects which are generated when finding process and view instances. Additional matching criteria can be provided as well (the `Has-Quantity` and `Heat-Connection` statements above) to further restrict the conditions under which instances are created. In theory these criteria could be placed in the processes' preconditions, but in practice it is worth placing them in the individual specifications if they won't change during the course of reasoning, thus reducing the number of process instances which must be considered.

Preconditions and quantity conditions are lists of statements which are interpreted conjunctively. Preconditions can be arbitrary predicate expressions, while quantity conditions must be either inequality statements or status statements (i.e., `(Status` *<instance>* `<ACTIVE` *or* `INACTIVE>)`). While statements in both preconditions and quantity conditions can be negated, no explicit disjunctions or any other logical connectives are allowed.

Three constructs are provided to introduce new terms in the relations field. Individuals which exist by virtue of the instance being active are specified by `Introduces` or `Introduce-Uniquely`. `Introduces` indicates that whenever the instance is active the individual will exist, and `Introduces-Uniquely` indicates that the individual exists if and only if the instance is active. Properties of the instance itself, such as quantities (in the heat flow example, `flow-rate`) are introduced with the `Local` specification. The syntax of the `Local` expression is

`(Local` *<name>* *<specification>*`)`

where *<specification>* is a predicate expression mentioning the name. In the heat flow process above, for instance,

```
(Local flow-rate (Quantity flow-rate))
```

indicates that `flow-rate` is a function which maps from instances of heat flow to quantities. When constructing a slice of an instance of a process, occurrences of the function name are replaced by a term

representing the application of this function to the slice. If, for example, PI-0 were an instance of heat-flow that was active in situation S0, then the following facts would be among those believed:

```
(I+ (heat (at ⟨dst⟩ S0)) (A (flow-rate (at PI-0 S0))))
(I- (heat (at ⟨src⟩ S0)) (A (flow-rate (at PI-0 S0))))
```

Individual views are defined using the defView form. Its syntax and interpretation is exactly the same as that of defProcess, except there can be no influences field.

8.1.7 Limit rules

Limit rules are "compilations" of conclusions that GIZMO has reached in performing limit analysis. When supplied with the model of the domain, they often allow GIZMO to reach the same conclusions with much less effort. Their exact role in limit analysis will be explicated later in section 9.4; for now we will merely describe their syntax.

A limit rule has the following form:

```
(defLimit-Rule ⟨name⟩ ⟨pattern⟩ ⟨body⟩)
```

The rule is run on all quantity hypotheses whose description satisfies ⟨pattern⟩, which consists of a pair of numbers, the current ordering between them, and the ordering between them proposed by the hypothesis. When a match is found ⟨body⟩ is executed in an environment where the variables in the patterns are bound to what they unified with. The body returns two values, a flag indicating what action is GIZMO is to take regarding the hypothesis and a list of reasons for why that action is appropriate. The flag can either be NIL, indicating that, as far as this rule is concerned, the hypohthesis is okay, T indicating that this particular hypothesis is inconsistent, or ALL, indicating that this hypothesis and any conjunctive hypotheses which include it are inconsistent.

8.1.8 Defining problems

It is convenient to have a means for specifying the initial conditions of a problem, such as what individuals exist, what relationships hold between them, and what situations they are involved in. The defScenario form does this. The syntax is:

```
(defScenario ⟨name⟩ . ⟨specifications⟩)
```

The specifications take the form of

```
(⟨key⟩ . ⟨statements⟩)
```

where ⟨key⟩ indicates the type of specification and ⟨statements⟩ are the particular content of it.

The first type of specification is Individuals, the initial set of individuals in the scenario. Constants serves the same role for problem-specific constants. The Facts specification provides a way to make initial assumptions. Particular situations are described within the In-Situation specification; the first element is the name of the situation, and the remaining elements are Individuals and Facts specifications which are

interpreted as stating that the individuals named exist in that situation and the facts are true about the slices of those individuals respectively. Facts that are true in all situations can be placed in an ALWAYS specification. Here is how defScenario would be used to describe a kettle that was always on a stove, starting at time t0:

```
(defScenario kettle-on-stove
 (Individuals stove burner kettle)
 (Facts (Temperature-Source stove)
        (Physob kettle)
        (Heat-Path burner))
 (Always (Heat-Path burner stove kettle))
 (In-Situation t0
        (Individuals stove burner kettle)
        (Facts (Greater-Than (A (temperature stove))
                             (A (temperature kettle))))))
```

8.2 Fluids

The first problem that arises in reasoning about fluids is deciding exactly what constitutes an object. In developing his axioms for liquids, Hayes introduced the idea of a "contained liquid", a piece of liquid considered as an object by virtue of being "the liquid in a place" [Hayes, 1978b]. As we have seen previously, this description is quite useful for reasoning about several of the processes which act on liquids and also gases.[1]

However, the vocabulary introduced in section 4.1 skirted an important issue. When a contained liquid was created it was given an arbitrary name. To see the problem with that, suppose there is steam and water inside a particular container. Part of the specification of boiling is that some steam will be created inside the container. Should boiling occur, how is the boiling process to know whether or not it must create a new individual, or merely add influences to an individual that is already there? In general, how are we to know that two fluid individuals are really the same?

The simplest solution seems to be the introduction of canonical names for such individuals. We will use the function c-s to denote fluid objects. c-s must depend on the substance, state, and container if we are to make all the required distinctions. All fluid individuals will be introduced via instances of the (redefined) individual view Contained-Stuff being active. Additional predicates are defined to describe the effects of state and the interactions between state and containment.

Before we can talk about contained stuff, however, first we must introduce the types of quantities we will be thinking about and the general definition of pieces of stuff. Here are the quantities we will be using:

```
(defQuantity-Type amount-of)
(defQuantity-Type amount-of-in)
(defQuantity-Type heat)
(defQuantity-Type pressure)
(defQuantity-Type temperature)
(defQuantity-Type tboil)
```

---

1. However, it seems the alternate "molecular collection" ontology is also required for certain kinds of reasoning. This point will be discussed in section 12.3.2.

```
(defQuantity-Type flow-rate)
(defQuantity-Type generation-rate)
(defQuantity-Type absorbtion)
(defQuantity-Type restorative)

(defQuantity-Type Level)
(defQuantity-Type Volume)
(defQuantity-Type height)
(defQuantity-Type (height top))
(defQuantity-Type (height bottom))
(defQuantity-Type Max-Height)

(defObservable-Quantity Level)
```

The first group are the thermodynamic properties we used in section 4.1. To summarize, we will consider `amount-of` as "the amount of stuff there is", roughly, the number of molecules in an individual (we must use `amount-of` to avoid confusion with "amount" in the sense of part of a quantity). The function `amount-of-in` maps from a substance, a state, and a container to a quantity that indicates how much of that substance in that state there is inside a particular container. Talking about `amount-of-in` as distinct from `amount-of` is necessary because `amount-of` is a property of the piece of stuff, while `amount-of-in` is used to state part of the conditions for that piece of stuff to exist. As before, `pressure` is assumed to be measured from the bottom of the container.

The second group are quantities which will appear in processes. The third group are quantities which represent various geometric properties. For simplicity in the implementation, quantities which might naturally be referred to as properties of parts (such as `(height (top container))`) are curried to apply to the object itself (such as `((height top) container)`). Finally, `level` is also marked as an *observable quantity*, a fact which will be used when performing measurement interpretation.

Next we define pieces of stuff. This description is independent of the particular criteria of individuation that is applied (i.e., it can be true of a collection of molecules or of a contained liquid). Since a piece of stuff is an individual, a `defEntity` is used to define it:

```
(defEntity piece-of-stuff ((amount-of quantity)
                           (volume quantity)
                           (pressure quantity)
                           (heat quantity)
                           (temperature quantity)
                           (tboil quantity))
  (Physob ?self)
  (Vertical ?self)
  (Qprop+ (temperature ?self) (heat ?self))
  (Taxonomy (Liquid ?self) (Gas ?self)))
```

Being a `Physob` in this context simply provides some geometric information; the exact specification of the `Physob` predicate appears below. Every piece of stuff will have the appropriate thermodynamic properties (the quantities specified in the parts list), and the temperature will always depend on the heat of the stuff. Furthermore, the stuff is either a liquid or a gas. Again, we will define these predicates shortly.

The approximation of a temperature source is sometimes useful. There are two ways to define them. First, a normal piece of stuff can be used and made subject to a special process, `heat-restore`, which provides an opposite influence to counterbalance any heat drawn from or added to it. A simpler alternative (although not as flexible, if we wish to add or remove this approximation at will) is to define `temperature-source` as a new type of object:

```
(defEntity Temperature-source ((heat quantity)
                               (temperature quantity))
  (physob ?self))
```

Since, unlike pieces of stuff, the temperature of a source isn't qualitatively proportional to the heat of the source, influencing the heat will leave the temperature unchanged.

Now we can describe `Contained-Stuff`.

```
(defView (Contained-Stuff p)
         Individuals ((?c (container ?c))
                      (?s (substance ?s))
                      (?st (state ?st)))
         Preconditions ((Can-Contain-Substance ?c ?s ?st))
         QuantityConditions ((Greater-Than (A (Amount-of-in ?s ?st ?c)) ZERO))
         Relations ((Introduces-Uniquely (c-s ?s ?st ?c))
                    (Q= (amount-of-in ?s ?st ?c) (amount-of (c-s ?s ?st ?c)))
                    (Qprop+ (amount-of-in ?s ?st ?c) (amount-of (c-s ?s ?st ?c)))))
```

In general, a qualitative proportionality must be provided when asserting an equality within a domain definition to indicate the appropriate direction of causality when resolving influences. As you can see, an instance of this view will be created for every combination of substance and state for each container. It is possible that a particular container cannot hold stuff of a particular type, for instance, storing nitric acid in a copper can, and this possibility is represented by the predicate `Can-Contain-Substance` as a precondition. The contained stuff exists exactly when the instance is active, as indicated by the `Introduces-Uniquely` statement.

We will need to specify a few more properties of individuals defined as contained stuffs. The `For-Slice` construct is syntactic sugar for a rule which triggers when the individual exists in a situation named by the first argument (here, `?sit`). Whenever a canonical individual exists, it is a piece of stuff, in its defined state, and the place where it is contains it.

```
(defentity (c-s ?substance ?state ?place) ((substance (== ?substance))
                                           (state (== ?state))
                                           (inside (== ?place)))
  (For-Slice ?sit
             (piece-of-stuff (at ?self ?sit))
             (?state (at ?self ?sit))
             (Contains (at ?place ?sit) (at ?self ?sit)))
  :code (cond ((eq ?state 'Liquid)
               (Impossible-fact (referent '(gas ,?self) T))
               (Impossible-fact (referent '(contained-gas ,?self) T)))
              ((eq ?state 'Gas)
               (Impossible-fact (referent '(Liquid ,?self) T))
               (Impossible-fact (referent '(Contained-Liquid ,?self) T))))
  :justification c-s-Definition)
```

The code in the specification (indicated by the `:code` keyword) indicates that a contained stuff defined to be in one state cannot be in the other. This information is used to reduce the number of spurious process and view instances introduced when analyzing a situation (see section 9.2).

Next substances are introduced. The nature of a substance doesn't change with time, hence they are domain-specific constants rather than individuals. For simplicity, we also assume that every container can hold every substance in every state:

```
(defPredicate substance)

(rule ((?f (:true (Substance ?s)))
       (?g (:true (Container ?c)))
       (?h (:true (State ?st))))
      (assert (-> (and ?f ?g ?h) (Can-Contain-Substance ?c ?s ?st)))
      (assert (-> (and ?f ?g ?h) (Quantity (amount-of-in ?s ?st ?c)))))
```

For example, a particular substance we know well is `water`:

```
(DefineConstant Water)

(DefFact (Substance Water))
```

The various empirical properties of water which distinguish it from other substances are not modelled in this vocabulary.

We will only model two states of matter:

```
(defPredicate State)

(DefineConstant LIQUID)
(DefineConstant GAS)

(DefFact (State LIQUID))
(DefFact (State GAS))
```

The next rule enforces the constraint that these states are the only states:

```
(rule ((?f (:true (State ?s))))
      (cond ((memq ?s '(LIQUID GAS))) :okay
            (t (assume (not ?f) :ABSOLUTE))))
```

The `:ABSOLUTE` means the TMS will never consider this assumption as a candidate for retraction.

Now we must define the additional properties a piece of stuff has by virtue of being in a particular state. Unlike the previous vocabulary, we can use predicates rather than individual views to represent these properties because we are dealing with canonical individuals, thus their state is always known.

```
(defEntity Liquid ()
        (not (Greater-Than (A (temperature ?self)) (A (tboil ?self))))
        (iff (Contained-Stuff ?self) (Contained-Liquid ?self))
        (Qprop+ (volume ?self) (amount-of ?self)))

(defEntity Gas ()
        (not (Less-Than (A (temperature ?self)) (A (tboil ?self))))
        (Qprop+ (pressure ?self) (amount-of ?self))
        (Qprop- (pressure ?self) (volume ?self))
        (Qprop+ (pressure ?self) (heat ?self)))
```

The inequalities represent the constraints on temperature imposed by each state. As in the previous vocabulary, the negated inequalities are used to allow a substance to be in either state at a phase boundary. The relationships between their thermodynamic properties which are independent of the particular individuating criteria are expressed by the qualitative proportionalities.

Since the individuating criteria for this vocabulary ensures that all liquid objects are contained liquids, seperating out the properties of contained stuff in particular states isn't strictly necessary. However, we define it seperately for contained-liquids for upward compatability:

```
(defEntity Contained-Liquid ((level quantity))
  (Q= (level ?self) ((height top) ?self))
  (Qprop+ ((height top) ?self) (level ?self))
  (Function-Spec level-function
                 (Qprop+ (level ?self) (amount-of ?self)))
  (Correspondence ((A (level ?self)) (A ((height bottom) ?self)))
                  ((A (amount-of ?self)) zero))
  (Function-Spec p-l-function
                 (Qprop+ (pressure ?self) (level ?self)))
  (rule (?f (:intern (?self container ?c)))
        (Q= ((height bottom) ?self) ((height bottom) ?c))
        (Qprop+ ((height bottom) ?self) ((height bottom) ?c))))
```

Naming the functions which determine level and pressure for contained liquids will allow us to compare values of them found in different containers; this information will be used in the envisioning chapter. The correspondence says that the level approaches the bottom of the container as the container runs out of liquid. The rule at the end of the specification points out a flaw in the implementation language; since general de-structuring mechanisms aren't used, such a rule is the only way to get hold of the container of the stuff in order to specify the relationship between the height of the bottom of the container and the height of the bottom of the liquid.

A crude geometry will suffice for our purposes. There are two orientations, `vertical` and `horizontal`:

```
(defPredicate orientation)

(DefineConstant vertical)
(DefineConstant horizontal)
```

We will treat orientation as a two-place relation, tying an object to one of these constants. The predicates `Vertical` and `Horizontal` are defined by these rules:

```
(rule (?f (:intern (Vertical ?obj)))
      (assert (iff ?f (Orientation ?obj vertical)) 'Domain-Definition))

(rule (?f (:intern (Horizontal ?obj)))
      (assert (iff ?f (Orientation ?obj Horizontal)) 'Domain-Definition))
```

Being a physob in this model just means that something has an orientation, and the relationship between the height of its top and the height of its bottom are constrained by this orientation:

```
(defEntity Physob (((height top) Quantity)
                   ((height bottom) Quantity))
          (Taxonomy (Orientation ?self VERTICAL)
                    (Orientation ?self HORIZONTAL))
          (iff (Orientation ?self HORIZONTAL)
               (Equal-To (A ((height top) ?self)) (A ((height bottom) ?self))))
          (not (Greater-Than  (A ((height bottom) ?self)) (A ((height top) ?self))))))
```

To better focus on dynamics, we will model as little information about piping systems, valves, and so forth as possible. A container will be modeled as a physob:

```
(defEntity Container () (Physob ?self))
```

We will simply assert such fluid paths and connections as exist. Both fluid and heat paths, however, are first-class individuals:

```
(defEntity Fluid-Path ((max-height Quantity)))

(defEntity Heat-Path ())

(defRelation (Fluid-Connection ?path ?source ?dest)
        (Fluid-Connection ?path ?dest ?source))

(defRelation (Heat-Connection ?path ?source ?dest)
        (Heat-Connection ?path ?dest ?source))
```

The predicate `Fluid-Connection` takes three arguments, a path and two pieces of stuff. When true, it indicates that the path can serve as a fluid path between one piece of stuff and the other. All heat and fluid connections are also considered to work both ways. Fluid paths also have a maximum height, since even with this simple geometry, we would like to prevent liquid from flowing up hill. Ignoring pumps and the siphon effect,[1] we can say that a path will support liquid flow only if the level of the source is greater than the maximum height of the fluid path. This will be represented by an individual view:

```
(defView (Liquid-Flow-Supporting ?path ?src ?dst)
  Individuals ((?src (Contained-Liquid ?src))
               (?dst (Contained-Liquid ?dst))
               (?path (Fluid-Path ?path)
                      (Fluid-Connection ?path ?src ?dst)))
  QuantityConditions ((Greater-Than (A (level ?src))
                                    (A (max-height ?path)))))
```

Now we can talk about liquid flow. Here is the liquid flow process we will use:

```
(defProcess (Liquid-flow ?src ?dst ?path)
            Individuals ((?src (Contained-Liquid ?src))
                         (?dst (Contained-Liquid ?dst))
                         (?path (Fluid-Path ?path)
                                (Fluid-Connection ?path ?src ?dst))
                         (?supported ((View-Instance Liquid-Flow-Supporting) ?supported)
                                     (?supported path ?path)
                                     (?supported src ?src)
                                     (?supported dst ?dst)))
            Preconditions ((Aligned ?path))
            QuantityConditions ((Greater-Than (A (pressure ?src))
                                              (A (pressure ?dst)))
                               (Status ?supported Active))
            Relations ((Local flow-rate (Quantity flow-rate))
                       (Q= flow-rate (- (pressure ?src)
                                        (pressure ?dst)))
                       (greater-than (A flow-rate) zero))
            Influences ((I+ (amount-of ?dst) (A flow-rate))
                        (I- (amount-of ?src) (A flow-rate))))
```

Notice that the dependence of flow on the geometric properties of the liquid path is expressed by including the instance of the support relation in the individuals and requiring that the relationship is true by placing it

---

1. The simplest way to represent these would use a hierarchial view description, with distinct cases representing normal flow, pumped flow, and siphoning. However, hierarchial descriptions are not currently supported in GIZMO.

in the quantity conditions.[2] We ignore gas flows in this vocabulary.

We have already examined heat flow, but for completeness we include it again:

```
(defProcess (Heat-Flow ?src ?dst ?path)
            Individuals ((?src (Physob ?src)
                               (Has-Quantity ?src heat))
                         (?dst (Physob ?dst)
                               (Has-Quantity ?dst heat))
                         (?path (Heat-Path ?path)
                                (Heat-Connection ?path ?src ?dst)))
            Preconditions ((Heat-Aligned ?path))
            QuantityConditions ((Greater-Than (A (temperature ?src))
                                              (A (temperature ?dst))))
            Relations ((Local flow-rate (Quantity flow-rate))
                       (Q= flow-rate (- (temperature ?src)
                                        (temperature ?dst)))
                            (Greater-Than (A flow-rate) zero))
            Influences ((I+ (heat ?dst) (A flow-rate))
                        (I- (heat ?src) (A flow-rate))))
```

Remember that the Q= expands into two qualitative proportionalities, hence the flow rate will be indirectly influenced by changes in the temperature of the source and destination.

Boiling is slightly more complicated because it involves state changes:

```
(defProcess (Boiling ?w ?hf)
            Individuals ((?w (Contained-Liquid ?w))
                         (?hf ((Process-Instance Heat-Flow) ?hf)
                              (?hf DST ?w))
                         (?s (Substance ?s)
                             (?w SUBSTANCE ?s))
                         (?c (Container ?c)
                             (?w CONTAINER ?c)))
            QuantityConditions ((Status ?hf ACTIVE)
                                (not (Less-Than (A (temperature ?w))
                                                (A (tboil ?w))))))
            Relations ((Introduces (c-s ?s GAS ?c))
                       (Contained-Stuff (c-s ?s GAS ?c))
                       (Local generation-rate (Quantity generation-rate))
                       (Local absorbtion (Quantity absorbtion))
                       (Has-Value (s (d (heat (c-s ?s GAS ?c)))) 0)
                       (Greater-Than (A absorbtion) ZERO)
                       (Greater-Than (A generation-rate) ZERO)
                       (Q= (temperature ?w) (temperature (c-s ?s GAS ?c)))
                       (Qprop+ generation-rate (flow-rate ?hf)))
            Influences ((I- (heat ?w) (A (flow-rate ?hf)))
                        (I- (heat (c-s ?s GAS ?c)) (A absorbtion))
                        (I+ (amount-of (c-s ?s GAS ?c)) (A generation-rate))
                        (I- (amount-of ?w) (A generation-rate))))
```

As in the other vocabulary, state changes are modelled by asserting that the canonical individual corresponding to the resulting phase exists, and its amount will increase as the amount of the stuff in the original phase decreases. Here we see Introduces at work: certainly boiling creates steam, but since there can be steam in the container even when boiling isn't occuring we use Introduces instead of

---

2. The predicate (View-Instance Liquid-Flow-Supporting) is simply a "curried" predicate automatically introduced to speed matching.

`Introduces-Uniquely` as in the contained stuff description. The substance (`?s`) and container (`?c`) are included in the individuals specification since the pattern-matching there is GIZMO's major mechanism for de-structuring compound objects.

Let us examine the causes and consequences of boiling more closely. Predicating boiling on a heat flow occuring models the necessity of a source of energy for phase transitions. Adding an influence which is the opposite of the flow rate of the heat flow to the heat of the water prevents the temperature of the water from changing. The flat assertion that the heat of the gas is unchanging (i.e., the `Has-Value` statement) and the `absorbtion` influence on the gas is a kludge; given that the heat flow to water is active, there will also be a heat flow from the same source into the newly-formed gas. By only constraining the sign of `absorbtion`, the assertion that the heat of the gas is unchanging will result in the assumption (when resolving influences) that the influence of the heat flow to the steam will be cancelled out.

Finally, there are two limit rules that reduce GIZMO's computational burden when envisioning. The first rule concerns changes in the ordering between the level of liquid in a container and the maximum height of a fluid path, the quantity condition for `Liquid-Flow-Supporting`. In particular, the rule concerns the possibility that the relationship between level of the contained liquid and the maximum height of the path will change from `Greater-Than` to `Equal-To`, as would happen, for instance, if there were flow out of a container. The body of the rule checks to make sure that the height of the bottom of the container isn't less than the maximum height of the path (i.e., that the maximum height of the path is less than or equal to the height of the bottom of the container). If it isn't, then it returns `ALL` to indicate that this possible change is impossible, either by itself or in concert with other possible changes:

```
(defLimit-Rule No-Hills
               (((A (level (at (c-s ?substance Liquid ?container) ?sit)))
                 (A (max-height (at ?path ?sit)))
                 Greater-Than
                 Equal-To))
  (multiple-value-bind (relation reasons)
      (ordering '(A ((height Bottom) (at ,?container ,?sit)))
                '(A (max-height (at ,?path ,?sit))))
    (if (and relation (not (eq relation 'Less-Than)))
        (values 'ALL reasons)
        (values nil nil))))
```

Without this rule, GIZMO comes to the same conclusion by considering what would be true after this change had occured. First, suppose this change happens at the same time the amount of liquid reaches zero. Then this change is moot, since the liquid no longer exists. Should the amount of liquid be greater than zero, the correspondence between `amount-of` and `level` in the `Contained-Liquid` description allows GIZMO to conclude that the level must be greater than the height of the bottom, thus contradicting the assumption that the level equals the maximum height of the path. Since that change leads to a contradiction, GIZMO will mark it as inconsistent.

The second rule embodies the conclusion that if the heights of the bottoms of two containers are the same then the source of a liquid flow between them won't vanish:

```
(DefLimit-Rule No-Sinks
               ((ZERO
                  (A (amount-of-in ?substance Liquid (at ?container ?sit)))
                  Less-Than
                  Equal-To))
  (multiple-value-bind (okay? reasons)
      (all-containers-at-same-level? ?sit)
    (if (and okay?
             (every-direct-influencer-of-type '(amount-of (at (c-s .?substance
                                                                  Liquid
                                                                  .?container)
                                                              .?sit))
                                              'Liquid-Flow))
        (values 'ALL reasons)
        (values NIL reasons))))
```

The pattern describes the possibility of the quantity condition for an instance of `Contained-Stuff` changing in such a way as to make it inactive (i.e., that the amount of stuff drops to `ZERO`). The function `all-containers-at-same-level?` tests to see whether or not the heights of the bottoms of all containers in the situation are equal to each other. If they are, and if every direct influence on the amount is due to some instance of liquid flow (checked by the function `every-direct-influencer-of-type`), then this change cannot occur, either alone or in concert with other changes. The types of processes which provide direct influences are limited to be `Liquid-Flow` because the argument originally used in ruling out this change (which we will see momentarily) doesn't apply when boiling occurs. Requiring that all containers be at the same level is quite conservative, since the only containers that really matter are those which are connected to the source by some fluid path.

The way GIZMO draws the conclusion represented by this rule is rather subtle. Consider the simple case of liquid flow occuring between two containers connected together by a single fluid path (as say in section 4.1). Suppose the hypothesized change occured. Then the contained liquid in the destination will exist and the contained liquid in the source won't. But if the contained liquid in the source did, what inequalities would hold between its parameters and the other parameters of the situation? Since the amount of the source liquid is zero, its level will be equal to the height of the bottom (by the correspondence in the `Contained-Liquid` description cited previously). Since the function which determines level as a function of amount is the same for all contained liquids (i.e., `level-function`, as defined by the `Function-Spec` statement in the `Contained-Liquid` definition), the heights of the bottoms of the two containers are the same, and the amount of contained liquid in the destination is greater than zero, which means that the level of the destination liquid is greater than that of the source. Because the function that relates pressures and levels is the same for all liquids (the function `p-l-function` introduced in the `Contained-Liquid` description), this means the pressure of the source liquid is less than the pressure of the destination liquid. But before the change, the pressure of the source liquid was greater than the pressure of the destination liquid (as evidenced by the flow from the source to the destination). Hence quantity space continuity (see section 3.6.4) has been violated, rendering this change impossible.

8.3 Motion

The description presented here is actually a simple set-up for simulating a spring and block oscillator, rather than a general domain description for sliding motion. Progress on building motion models was held up due to a deficiency in the implementation; until recently, GIZMO had no way of finding out what domain-dependent closed world assumptions it needed to make in the course of reasoning. This meant the set of net forces, crucial to modelling motion, could not change as conditions changed. This deficency has since been remedied, but the new motion vocabulary has not been adequately debugged and hence will not be presented.

First, we describe the types of quantities used.

```
(defQuantity-Type position)
(defQuantity-Type velocity)
(defQuantity-Type force)
(defQuantity-Type length)
(defQuantity-Type rest-length)
(defQuantity-Type acc)
```

The objects we will consider are blocks and springs. A block's position will be completely described by a quantity, as will its velocity and (net) force:

```
(defEntity Block ((position quantity)
                  (velocity quantity)
                  (force quantity))
  (Mobile ?self)))
```

All blocks are assumed to be mobile. A spring's force is determined by the difference between its length and rest length:

```
(defentity Spring ((force quantity)
                   (rest-length quantity)
                   (length quantity))
  (Qprop- (force ?self) (length ?self))
  (correspondence ((A (force ?self)) zero)
                  ((A (length ?self)) (A (rest-length ?self)))))
```

A spring has several states. If the length is the same as the rest length then it is relaxed, if the length is greater than the rest length then it is stretched, and if the length is less than the rest length then it is compressed. As before, these states will be modelled as individual views:

```
(defview (Relaxed ?s)
         Individuals ((?s (spring ?s)))
         QuantityConditions ((Equal-to (A (length ?s)) (A (rest-length ?s)))))

(defview (Stretched ?s)
         Individuals ((?s (spring ?s)))
         QuantityConditions ((Greater-Than (A (length ?s)) (A (rest-length ?s)))))

(defview (Compressed ?s)
         Individuals ((?s (spring ?s)))
         QuantityConditions ((Less-Than (A (length ?s)) (A (rest-length ?s)))))
```

We will need a way to talk about directions. The simplest way would be to use the values of signs as indicating directions along some (implicit) axis. Unfortunately, DEBACLE does not allow fixed-point numbers to be first-class individuals. Thus we shall introduce symbolic names for these directions:

```
(defpredicate direction nil)

(defineConstant Plus)
(defineConstant Minus)
(defineConstant Null)

(defFact (Direction MINUS))
(defFact (Direction PLUS))
(deffact (Direction NULL))

(defFact (Non-Null-Direction MINUS))
(defFact (Non-Null-Direction PLUS))
(defFact (not (Non-Null-Direction NULL)))

(Impossible-Fact (Non-Null-Direction NULL))
```

The additional distinction that a direction is not null is needed to rule out motion that doesn't go anywhere. The `Direction-Of` predicate links these constants with sign values:

```
(defRelation (Direction-Of ?dir ?number)
  :code (caseq ?dir
          (MINUS (rassume (iff ?self
                               (Has-Value (s ?number) -1)) 'Direction-Law))
          (NULL (rassume (iff ?self
                              (Has-Value (s ?number) 0)) 'Direction-Law))
          (PLUS (rassume (iff ?self
                              (Has-Value (s ?number) 1)) 'Direction-Law))
        (t (rassume (not ?self) 'Direction-Law))))

(defComputable-Relation Direction-of)
```

This definition ties the constants `MINUS`, `PLUS`, and `NULL` to sign values of `-1`, `1`, and `0` respectively. The second statement (the `defComputable-Relation` form) tells GIZMO that, unlike most statments in precondiuons, facts of this type can be re-computed and hence should not be automatically assumed to persist.

Somehow a block and spring must be connected. The right way would be to have a `Connected-To` relationship that would transmit forces (as in the examples in section 4.4). However, since this vocabulary assumes no means of updating the set of net forces (thanks to the implementation deficiency mentioned above), we will have to use a kludge:

```
(defRelation (Connected-to-Spring ?from ?to)
      ;?from is the block
      ;?to is the spring
  (Q* (force ?from) (force ?to))
  (Qprop+ (force ?from) (force ?to))
  (Qprop+ (length ?to) (position ?from))
  (correspondence ((A (length ?to)) (A (rest-length ?to)))
                  ((A (position ?from)) zero)))
```

This description is a kludge because it violates the "no function in structure" principle [de Kleer & Brown, 1983]. By asserting the force of the spring equals that of the block, it implicitly assumes that the spring and block have no other forces applied to them. This makes the description non-modular. Since we have already sullied ourselves, we also incorporate the assumption that the spring is at its rest length when the position of the block is at the origin (the second `Qprop+` and the `Correspondence`).

The process vocabulary we will use is only slightly different from the Newtonian model introduced in section 4.3:

```
(defProcess (Motion ?obj ?dir)
   Individuals ((?obj (Mobile ?obj))
                (?dir (Direction ?dir)
                      (Non-Null-Direction ?dir)))
   Preconditions ((Direction-Of ?dir (A (velocity ?obj))))
   QuantityConditions ((Greater-than (m (A (velocity ?obj))) zero))
   Influences ((I+ (position ?obj) (A (velocity ?obj)))))
```

As mentioned above, motion occurs when an object has a non-zero velocity. In this vocabulary, a distinct instance of motion will be created for each direction of motion. The `Direction-of` statement in the preconditions ensures that the instance appropriate to the velocity's direction becomes active. The difference between this description and the earlier model is the additional matching condition of the direction being non-null, which precludes creating an instance of motion that will never be active. A similar trick is used in modelling acceleration:

```
(defprocess (Accelerating ?obj ?dir)
   Individuals ((?obj (Mobile ?obj))
                (?dir (Direction ?dir)
                      (Non-Null-Direction ?dir)))
   Preconditions ((Direction-Of ?dir (A (force ?obj))))
   QuantityConditions ((Greater-than (m (A (force ?obj))) zero))
   Relations ((Local acc (Quantity acc))
              (Qprop+ acc (force ?obj))
              (Correspondence ((A acc) zero)
                              ((A (force ?obj)) zero)))
   Influences ((I+ (velocity ?obj) (A acc))))
```

9. Basic deductions

In this chapter we examine algorithms to perform the basic deductions used in Qualtitative Process theory. To avoid getting mired in uninteresting details, the algorithms will be specifed in "structured English". Occasionally, implementation tricks which allow signficant speedups will be described.

9.1 Finding view and process instances

In principle, finding view and process instances is simple. Given an initial set of objects, find all collections of them which match the individual specifications of each individual view and process in the view and process vocabularies. Each collection gives rise to an instance of that particular view or process. For example, there will be two view instances for every substance and every container, each representing that substance in a particular state inside the container. However, the ability of processes and individual views to introduce new individuals when active complicates this picture somewhat, since the matching computation may have to be performed again on the new individuals. For example, if we believe some of the view instances representing contained liquids in the three-containers scenario (section 5.1) are active, there will now be process instances corresponding to flows between them.

In practice, producing all possible individuals and instances from an initial collection of objects simplifies later computations. This procedure will be called *elaborating* a situation. Figure 48 presents the algorithm used. One subtlety is worth mentioning. A naive algorithm would result in an exponential number of assumptions about the status of instances, since all possible combinations of status assignments would have to be tried to ensure that all and only the possible instances have been found. For example, both view instances corresponding to contained-liquids in two containers connected by a fluid path would have to be assumed true to find the appropriate instances of liquid flow. However, the algorithm presented tests each instance only once. How can this work?

The trick lies in the willingness to match on statements that have some chance of being true, irregardless of their current belief status. This is formalized by the idea of a *possibly true* fact. A fact is possibly true if and only if (1) it has been mentioned (i.e., it appears in the database), (2) it is not marked as being impossible and (3) either it is true or there is some reason to believe it could be true. The second condition is useful in ruling out some absurd possibilities, as we saw in the fluid model of Section 8.2. The third condition, the existence of some reason to believe it could be true, is satisfied in GIZMO by the existence of any clause in the TMS which could make it true. Importantly, the rules which implement the domain knowledge (such as those created by `defEntity`, `defPredicate`, `defRelation`, and so forth) trigger on a fact being interned -- being in the database -- rather than being actually believed to be true. Thus all the predicates and relationships which might be true are in the database, the clauses they introduce are accessible to the TMS, and matches to find potential instances are generated on that basis rather than actual current belief. In the worst case this algorithm can generate instances which could never be active, but this seems a small price to pay for avoiding making and retracting an exponential number of assumptions.[1]

---

1 Notice that if every rule in the system triggers on `:intern`, step 4 isn't necessary at all.

Fig. 48. Situation elaboration

```
Procedure Elaborate

    1. Let SLICES be the individuals which initially exist
       in the situation.

    2. Compute a table of all the type predicates which apply
       to the members of SLICES

    3. For each process in the Process Vocabulary and each
       individual view in the View Vocabulary,

       3.1 Find all collections of objects from SLICES which match the
           individual specifications of the description

       3.2 Create an instance of that type of process or view
           for each such collection.

    4. For each new instance,

       4.1 Assume it is active.

       4.2 Note any new individuals which exist as a result
           of its being active.  Add these individuals to
           SLICES and update the type table accordingly.

       4.3 Retract the assumption that it is active.

    5. Continue from step 3 until no new instances or
       individuals are found.
```

Once the view and process instances are found for a situation, a *comparison table* is computed that summarizes which numbers are compared by quantity conditions of these instances. The comparison table is used in performing limit analysis.

9.2 Finding view and process structures

Finding out which process and view instances are active in any particular situation requires establishing their preconditions and quantity conditions. In the simplest case this is performed by collecting the preconditions and quantity conditions associated with all the instances and merging them to determine the minimal set of facts that need to be established. For example, if flows in both direction are possible, one needs only to ask once about the relationship between the pressures to determine which, if any of them, is active. How these facts are established will depend on exactly what the program is doing. Usually this information will be provided in the initial conditions or determined by asking the user. Often it is more natural to simply ask about the status of instances directly, and this option is provided as well.

In performing measurement interpretation, the space of possible view and process structures is searched by constructing a dependency-directed generator to enumerate the possibilities (see the appendix). A similar generator is used in envisioning if the view or process structure in the initial situation is not

completely specified, but the envisioning algorithm guarentees the process and view structures will be complete in every situation it constructs after that.

9.3 Resolving influences

Suppose the process and view structures are known. Then all of the direct influences on the quantities in the situation will be known, as well as the qualitative proportionalities which will propagate these direct effects to indirectly influence other quantities. Using this information, influence resolution attempts to calculate a D_s value for each quantity in the situation. Figure 49 presents the algorithm used to resolve influences. The first two steps set up the machinery needed for resolving each quantity. The third step orders the quantities so that no quantity will be considered before trying to resolve quantities it depends on. This is accomplished by a simple tree-walk outward from each directly influenced quantity along qualitative proportionalities, as figure 50 illustrates. Notice that this algorithm implicitly assumes that the graph of qualitative proportionalities which hold at any particular time is loop-free. This assumption is safe for several reasons. First, relationships between parameters are represented irredundantly since qualitative proportionalities express the assumed direction of causation as well as a functional dependence. Second, by assumption, no quantity can be both directly and indirectly influenced at the same time. This condition prevents forming loops that are not ruled out by the first condition. Finally, the distinction between direct and indirect influences "breaks" loops which would otherwise appear due to simultaneous equations in other types of models.[1]

Step 4 is where the actual work is accomplished. The effect of the sorting is that the iteration will begin with uninfluenced quantities, then the directly influenced quantities, and finally the indirectly influenced quantities. If a quantity is directly influenced then its derivative will be the sum of the influence adder constructed for it (step 2.1). The adder may be able to decide the sum directly, if for example the positive or the negative inputs set is empty. As mentioned previously, the adder may also use inequality information if there are both positive and negative direct influences. This information may change if the quantities providing the direct effects (such as flow rates) are influenced, and so these inequalities are added to the situation's comparison table so that limit analysis can take changes in them into account. As mentioned previously, if there are conflicting effects on an indirectly influenced quantity, then inequality information will not a priori help. However, if access to domain-specific or problem-specific rules is provided (which it isn't in GIZMO, yet), any use they make of inequality information should also be added to the comparison table.

---

1. In particular, the "feedback heuristics" required in [de Kleer, 1979], [de Kleer & Brown 1984] and [Williams, 1984] are unnecessary.

Fig. 49. Resolving influences

Procedure Resolve

1. Let QUANTITIES be the set of quantities belonging to the objects in the situation.

2. For each quantity Q in QUANTITIES,

   2.1 If directly influenced, create an influence adder. Make closed-world assumptions to determine the inputs and minus-inputs for the influence adder.

   2.2 Otherwise, find CONSTRAINERS(Q) by fetching all $\propto_{Q+}$ and $\propto_{Q-}$ statements which have Q as their first argument, closing this set by assumption.

3. Let QUEUE = SORT-BY-DEPENDENCY(QUANTITIES).

4. Until QUEUE is empty,

   4.1 Let Q = pop(QUEUE)

   4.2 If Q is directly influenced, then

      4.2.1 If the sum of its influence adder is known, the $D_s[Q]$ equals the sum.
      4.2.2 Add any inequality information used by the adder to COMPARISONS.

   4.3 Otherwise, if CONSTRAINERS(Q) is non-empty,

      4.3.1 Let PLUS, MINUS, UNKNOWN be empty
      4.3.2 For each Q1 in CONSTRAINERS(Q)

         4.3.2.1 If $D_s[Q1] = 1$ and $Q \propto_{Q+} Q1$ or $D_s[Q1] = -1$ and $Q \propto_{Q-} Q1$, then add Q1 to PLUS.
         4.3.2.1 If $D_s[Q1] = -1$ and $Q \propto_{Q+} Q1$ or $D_s[Q1] = 1$ and $Q \propto_{Q-} Q1$, then add Q1 to MINUS.
         4.3.2.1 If $D_s[Q1]$ is unknown, then add Q1 to UNKNOWN

      4.3.3 If UNKNOWN is non-empty or if both PLUS and MINUS are non-empty, then $D_s[Q]$ cannot be decided.
      4.3.4 If PLUS is non-empty, then $D_s[Q] = 1$
      4.3.5 If MINUS is non-empty, then $D_s[Q] = -1$
      4.3.6 Otherwise, $D_s[Q] = 0$

   4.4 Otherwise, $D_s[Q] = 0$

5. If COMPARISONS is non-empty, use it to update the situation's comparison table

Fig. 50. Determining the order of resolution

```
Procedure SORT-BY-DEPENDENCY(QUANTITIES)

        1. For each Q in QUANTITIES, let mark(Q) = 0.

        2. For each Q in QUANTITIES,

           2.1 If directly influenced, MARK-DEPTH(Q, 1)

        3. Return QUANTITIES, sorted by increasing marks.

Procedure MARK-DEPTH(Q, depth)

        1. If mark(Q) < depth,

           1.1 let mark(Q) = depth

           1.2 for each Q1 in CONSTRAINEES(Q),
                MARK-DEPTH(Q1, depth+1)
```

9.4 Limit analysis

Limit analysis is the most complex of the basic deductions. The purpose of limit analysis is to identify state changes, such as changes in process structure, view structure, or changes in D_s values, due to the activity of processes. In essence, it works by using the D_s values found by influence resolution to detect possible changes in the quantity space values, such as a quantity approaching a limit point. We will begin by examining the top-level procedure, then take a closer look at the individual steps. The top-level procedure is illustrated in figure 51. Recall that the hypothesis that a particular ordering relationship or set of ordering

Fig. 51. Limit analysis

```
Procedure Limit Analysis(S)

        1. Find the set of quantity hypotheses representing
           possible changes in quantity spaces for quantities
           in S.  Also determine the sets NEXT-QHS(S) and ECL-QHS(S).

        2. For each quantity hypothesis, annotate it with the changes
           it causes in the process and view structures (if any).

        2. Filter out those quantity hypotheses which lead to
           inconsistent situations.

        3. Assign a duration to the situation as follows:

           3.1 If NEXT-QHS(S) is non-empty, then INSTANT.

           3.2 If ECL-QHS(S) is empty, then INTERVAL

           3.3 Otherwise, duration is AMBIGUOUS
```

relationships represents the next change that occurs is called a *quantity hypothesis* (see Section 3.6.4). The first step, finding the set of quantity hypotheses, is the most complex and it will be examined shortly. Once found, the quantity hypotheses are marked with the change they will cause in the view and process structure for the situation, if any -- the hypothesis might simply correspond to a change in the way influences are resolved.

Finding out exactly what changes will result from a particular quantity hypothesis being true (step 2) is quite complicated because domain-specific knowledge may rule out the hypothesis as being inconsistent. We will discuss the procedure used in the chapter on envisioning. However, a simpler algorithm suffices to determine the what changes a quantity hypothesis will make in the view and process structures, assuming it turns out to be consistent. The idea is to make a simple constraint network model of the individuals, view and process instances, and quantity conditions which embodies the relationships between statuses and existence. This constraint model is then set up to represent what objects currently exist, the current statuses of view and process instances, and the current truth of quantity conditions. The changes represented by a quantity hypothesis are fed into the contraint model, and the resulting changes in statuses and existence read out from it.

There are three kinds of objects in this constraint model, models of individuals, models of instances, and models of quantity conditions. A model of an individual has an existence property. If it exists by virtue of some instance being active, the model of that instance is said to be a *supporter* of it. If a model of an individual has supporters, then it exists exactly when one of them is active. A model of a comparison has an existence property as well, tied to the existence of the individuals whose quantities it compares. It also has a property which describes the relationship which holds between the quantities it compares. A model of a view or process instance has an existence property and a status property. Like the other models, it has supporters -- those individuals upon whom its existence depends, and whether or not it is active depends on whether the state of the comparision models which represent its quantity conditions match its internal state specifications.

The constraint network for a situation is built as soon as the situation has been elaborated. To use it, the properties of models (existence, state of comparisons, and statuses of instances) are set up to reflect the current situation. The state of the comparision models which correspond to the change represented by a quantity hypothesis are changed accordingly, and these results propagated through the model. By keeping a record of the previous state and comparing this record with the model's current state, the changes which will occur can be determined. Notice, however, that this procedure is not informed by any particular domain-dependent information, such as the existence of correspondences or explicit functions. So the constraint network model cannot detect that a hypothesis is inconsistent, only provide information about what changes will result if it is. This constraint model is also useful in pruning certain irrelevant conjunctive hypotheses, as we will see shortly.

Returning to the top-level algorithm, the third and final step is to assign a duration to the situation, based on how soon it will change. Recall that we divide times into instants and intervals, and the equality change law determines how long each situation will last (see section 3.6.4). The sets NEXT-QHS and ECL-QHS are computed as side-effects of finding quantity hypotheses, as we will see below. Both sets, if non-empty, consist of quantity hypotheses which satisfy the equality change law. The distinction between them comes about because some quantity hypotheses depend on additional assumptions. Suppose, for example, that

$$A[Q_1] = A[Q_2] \wedge D_s[Q_1] = D_s[Q_2] = 1$$

Then Q_1 and Q_2 will change from equality if and only if

$$\neg\ D_m[Q_1] = D_m[Q_2]$$

Since the two rates might be equal, it is not certain that this change will occur. The hypotheses in NEXT-QHS are the subset of hypotheses covered by the equality change law which do not require such assumptions. Therefore if NEXT-QHS is non-empty, some instantaneous change must occur and so the situation lasts but an instant. The set ECL-QHS simply consists of all instantaneous changes, so if it is empty the situation must last for some interval of time. Otherwise the situation may last an instant if the assumptions underlying the hypotheses representing instantaneous changes are satisfied, or an interval of time if they aren't. So in that case the duration is marked as ambiguous.

Now we turn to the problem of finding quantity hypotheses. The basic algorithm is illustrated in figure 52. The quantity spaces are updated by using the comparison table to determine first what numbers require quantity spaces, and then to determine what elements should be in each one. A number requires a

---

Fig. 52. Finding quantity hypotheses

```
Procedure Quantity Change Analysis

    1. Retrieve comparison table for situation.

    2. Create and update the quantity spaces for the quantities
       mentioned in the comparison table.

    3. Update the state of the comparison table.

    4. Filter the comparison table by removing those
       entries whose comparisons aren't neighbors.

    5. For each element remaining,

       5.1 Determine if the inequality relationship can change.

       5.2 If it can, create a quantity hypothesis to represent
           that change.  Record any assumptions about rate required
           for the change to occur.

    6. Generate quantity hypotheses to represent all possible
       combinations of occurrences of the single change
       hypotheses, pruning out hypotheses which are moot
       or that are explicitly excluded by domain-specific limit rules.

    7. Apply the equality change law to determine which quantity
       hypotheses represent changes which can occur in an instant.
       Call the set which occur in an instant ECL-QHS(S), and call the
       remainder NON-QHS(S).

    8. If none of the members of ECL-QHS(S) requires an assumption
       about rate, then let NEXT-QHS(S) be ECL-QHS(S).  Otherwise, let
       NEXT-QHS(S) be empty.
```

quantity space if it is mentioned in any element of the comparision table.[1] A number is an element in the quantity space of another number if there is an element in the comparison table that compares it with that number. Since the comparison table was constructed using all the comparisons required to find the view and process structures, and augmented by any inequality information required in resolving influences, we are guarenteed that quantity spaces constructed from it in this manner contain all the information necessary to predict state changes.

Once the elements for each quantity space have been found, an effort is made to establish the ordering relationships between the elements.[2] This ordering information is used to compute the neighbors in the quantity space. The neighbors, as defined previously, are just the elements which are ... er greater or less than the number, with no other element in between (the set of elements which are equal to it is computed as well). The comparison table is updated by recording what the ordering between the numbers mentioned in each entry actually is, and entries representing comparisons between non-neighboring numbers are ignored (step 4) since they will be irrelevant - by definition, the quantity will reach the neighbor first.

Step 5 involves finding the possible changes for each relevant comparison. The current relationship between the comparison's quantities and their D_s values are used to look up in a table (described in section 3.6.4) what the next relationship can be. If some piece of information, such as a D_s value, is missing or the next relationship is the same, then the comparison is ignored.[3] If the relationship can change, then a quantity hypothesis is created to represent that possibility and annotated with any assumptions it requires about the rates (step 5.2).

Given the collection of hypothesized single changes, step 6 computes the collection of hypotheses representing the prospect that more than one of these changes occurs at once. It is important to prune hypotheses as quickly as possible, since if there are n hypotheses representing a single change then in theory there can be 2^n-1 total hypotheses. In practice there are rarely this many, for several reasons. First, certain conjunctions can be ruled out because one change makes the other moot. For example, it makes no sense to consider the possibility that two flow rates equalize at the same time one flow stops, since one of the flow rates will no longer exist. The constraint network mentioned above is used to find such moot conjunctions and prune them as they are generated. Second, other conjunctions will violate consistency constraints, either general ones such as continuity or domain-specific ones, such as implied by correspondences between functionally dependent quantities (e.g., the arguments about impossible changes in the end of section 8.2). Finding out which combinations are consistent in this way is fairly expensive, since it requires explicitly generating the situation which occurs next, a procedure we will defer explaining until the discussion of envisioning. Limit rules, introduced in Section 8.1.6, provide a mechanism to reduce this burden. A proposed quantity hypothesis is tested against a database of limit rules, which can either let it pass, rule out

---

1. The only exception is ZERO -- no quantity space is ever created for it. In addition, ordering statements involving ZERO are treated as statements about signs, for efficiency reasons. See the appendix for details.
2. In GIZMO, this consists of queries to a specialized quantity implementation which will give an answer if it follows from known inequalities and transitivity. Once again, see the ppendix for details.
3. An alternative is to search the possible orderings between the numbers. Given the way in which the comparison table is constructed, however, the ordering information will always be known when envisioning so this alternative was deemed undesirable.

just that hypothesis, or rule out all hypotheses which contain that hypothesis. So far, the only use of limit rules has been to "routinize" conclusions that GIZMO can make without them, albeit with much greater effort.[1] Although limit rules must provide reasons for their conclusions, if the answer they return is ALL, (as the rules in the previous chapter did), then no conjunctive hypotheses involving the ruled-out hypothesis will ever be generated. Thus even if some of the facts which support the limit rule's conclusion are withdrawn, limit analysis must be performed again to compute the entire set of possible changes.

The equality change law is applied in step 7. Recall that the equality change law says that, except in two cases, all changes require an interval of time to occur. The first case is that the change in the ordering relationship is away from equality; this respects the fact that numbers aren't "fuzzy". The second case is when a change to equality is occuring between quantities which were influenced away from equality for only an instant. This case assumes that the influences of processes are finite. Finding hypotheses which satisfy case 1 is simple; they are exactly the hypotheses for which one of the current relationships is equality. Satisfying case 2 requires using some additional information carried along with the situation. In particular, if the situation arose by a hypothesis satisfying case 1 of the equality change law, then the situation includes the additional assertion that one number is "starts infinesimally greater than" the other, such as

$$A[Q_1] \; I> \; A[Q_2]$$

A consequence of believing the assertion above is that any hypothesized change which includes Q_1 and Q_2 becoming equal must occur in an instant. The set of quantity hypotheses which satisfy either case of the equality change law comprise the set ECL-QHS, and those which don't comprise the set NON-QHS.

Finally, in step 8 the subset of quantity hypotheses which satisfy the equality change law is examined to see if any of them require additional assumptions about rates to be valid. If none of them do, then the next change that occurs must be one of the hypotheses satisfying the equality change law, and so the set NEXT-QHS is made equal to ECL-QHS to reflect this fact. Otherwise it is not clear whether or not a change from ECL-QHS will occur, so NEXT-QHS is set to the empty set to inform the top-level limit analysis procedure of this fact.

Now let us return to the problem of testing the consistency of a quantity hypothesis. As mentioned previously, we will defer discussion of how the next situation is generated for later. For now, it is enough to know that it can be done, and certain inconsistencies can be detected in the course of doing so. If a consistent next situation can be constructed, then several additional tests are made.[2] First, the inequality constraints implied by named functions (such as the function that determines pressure from level) are imposed. Second, quantity space continuity is tested if some individual has vanished in the new situation. If no individual vanishes, then these constraints are usually satisfied by the procedure which generates possible changes, and violations are detected by the procedure which constructs the next situation. Should an individual vanish, however, extra work is required to detect continuity violations. Recall the No-Sinks limit rule in the fluids

---

1. This "compiling" of the domain knowledge is currently done by hand; an interesting learning problem is to acquire such rules from experience. Such rules could also be used to encode heuristic knowledge, but in that case the implementation would have to be extended to allow graceful retraction of their results.

2. When performing limit analysis in the course of envisioning these tests are interleaved with the situation generation performed by that procedure, thus saving some work.

description of the previous chapter. Consider again a liquid flow between two containers. One possibility is that the liquid in the source will vanish, as the amount of it reaches zero. Another possibility is that the pressure in the two containers will equalize. If the heights of the bottoms are the same then the first change is impossible, since it would require that the pressure in the source was less than the pressure in the destination (assuming explicit functions linking level and pressure, as noted previously). This violation of continuity will not, however, be detected, since the liquid in the source, and hence its pressure, no longer exist! As alluded to previously, the solution is to re-install the facts about the quantities of the vanished individual as if it existed, and then look for continuity violations. This procedure is somewhat gory, so we won't look into it further. The final test is for D_s continuity, and it is performed by resolving influences in the new situation and seeing whether the D_s value of any quantity jumps from -1 to 1 or 1 to -1 from the current situation to the new one. If any do, then the quantity hypothesis is marked as inconsistent.

10. Measurement interpretation

This chapter describes an algorithm which constructs interpretations of measurements taken at an instant (the "one-look" case described in section 5.5). The pruning heuristics described in section 5.5 are not implemented because the domain models considered are not complex enough to warrent them. The input to this algorithm is a situation and a set of observations, and the output is a set of interpretations describing how the possible processes in the situation could explain those observations. First we describe the interpretation algorithm, then a procedure for automatically generating diagnostics given a set of interpretations, and finally illustrate these procedures with an example.

10.1 The algorithm

We assume the situation S has been elaborated, i.e., all the possible view and process instances in it have been found. The set of observations, called OBS, consists of expressions of the form

(Measured (has-value (s (D *<quantity>*)) *<-1, 0, or 1>*))

where *<quantity>* is a reference to a quantity belonging to an individual in S. Figure 53 presents the interpretation algorithm. In the course of setting up the situation to be interpreted several of the instances may have their statuses determined -- for example, measuring a property of an individual implies that the individual does indeed exist. In addition, the user is queried to determine if there are more assumptions to be made about the statuses of view or process instances. The process and view instances whose statuses are unknown are used to create a complete, irredundant generator which will produce all and only the consistent combinations of status assignments. The appendix describes the generator in more detail; the only property we need to know here is that it can be informed by adding clauses to the TMS, as happens when facts are asserted.

Recall that an interpretation consists of a set of status assignments and the collection of unit cause hypotheses (abreviated UCH) that provide its account of the measured D_s values. The search for interpretations proceeds by asking the generator for a collection of status assignments, constructing an interpretation which represents the hypothesis that the collection is responsible for the measurements, and testing this hypothesis to see if it is consistent. The first step (2.1) in testing the hypothesis is to resolve influences. In step 2.2, a UCH is constructed to account for each measurement in the context of the interpretation hypothesis, and is locally tested for consistency by seeing if the sign of the derivative for the measured quantity computed by influence resolution matches the observed value. If the D_s value is known but different then the UCH, and hence the whole interpretation, is inconsistent. If the D_s value is unknown, then the assumptions needed to make it be the measured value are recorded as part of the UCH.

Suppose several UCH's require making assumptions. Then it is possible that a consistent UCH can be constructed for each measurement, but when taken together the assumptions they make imply a contradiction. Step 2.3.2 takes this possibility into account by assuming the various facts needed to make the UCH's valid and propagating the results. If any contradictions ensue, the interpretation is marked as inconsistent.

When an interpretation is found to be inconsistent, either locally due to the failure to find a consistent UCH for some measurement, or globally due to inconsistent assumptions made in constructing

Fig. 53. One-look measurement interpretation algorithm

```
;Let OBS be the set of measurements

Procedure Interpret

    1. Make status assignments to process and view instances
       wherever possible

    2. Perform a dependency-directed search over the status
       assignments remaining.  For each set of status assumptions,

       2.1 Resolve influences.

       2.2 For each measurement M in OBS,

           2.2.1 Let Q be the quantity described by the measurement M,
                 VAL be the D_S value measured, and UCH be the Unit Cause
                 Hypothesis which is to account for M in this
                 interpretation.

           2.2.2 If D_S[Q] = VAL, then mark UCH as consistent

           2.2.2 If D_S[Q] is known, then mark UCH as inconsistent

           2.2.3 Otherwise,

                 2.2.3.1 If Q directly influenced, mark UCH as consistent
                         and record the inequality assumption needed to
                         insure D_S[Q] = VAL

                 2.2.3.2 If Q indirectly influenced, mark UCH as consistent
                         and record VAL as D_S[Q]

       2.3 Let UCHS be the set of Unit Cause Hypotheses constructed in
           the previous step, and INTERP be the interpretation representing
           the current collection of status assignments.

           2.3.1 If any UCH in UCHS is marked as inconsistent, mark
                 INTERP as inconsistent, giving the inconsistent UCHs
                 as a reason,  and install appropriate nogood sets.

           2.3.2 Temporarily make any assumptions required by the UCHs.

                 2.3.2.1 If any contradictions arise, mark INTERP as
                         inconsistent and install appropriate nogood sets.

                 2.3.2.2 Otherwise, mark INTERP as consistent.
```

UCHs, nogood sets[1] are constructed to prevent the generator from trying that collection of status assignments again, thus reducing the search space. The nogood sets will always include some status assignments. If the nogood is constructed for a particular UCH then it will include the measurement the UCH was intended to explain, and if the nogood is constructed from a global contradiction it will contain all the measurements.

Which status assignments should be included in a nogood set? The notion of *p-influencers* introduced in section 5.5 (the set of instances which provide direct or indirect influences on a particular quantity) would allow exactly that subset of the status assignments which was responsible for the particular D_s measurement to be in the nogood set. This would provide maximum constraint by ensuring that no hypothesis containing those assignments as a subset would ever be generated again. However, finding the p-influencers proved difficult in this implementation; since qualitative proportionalities are often introduced by virtue of type predicates as well as instances, additional indexing is required to construct the set of p-influencers. Thus it proved simpler to just use all of the status assumptions in the nogood set. While it doesn't prune the search space as much as using p-influencers would, it does provide some constraint if the same situation is examined with two non-disjoint sets of measurements. We will see an example of this in section 10.3.

10.2 Making diagnoses

Given a collection of consistent interpretations for a set of measurements, predictions can be made about the values of other measurable parameters. By testing these predictions the set of consistent interpretations can be further constrained. Figure 54 shows how. The first step is simply to find all measurements that have not yet been made. Step 2 classifies each interpretation according to what it predicts about that measurement, using information saved during the interpretation process. If the interpretation doesn't constrain the quantity, then for simplicity it is placed in all the sets, even though it is possible that the assumption necessary to make it be some particular value might lead to a global contradiction. If it is necessary to make absolutely sure that only consistent diagnoses are made, then a test similar to the global consistency test in procedure INTERPRET could be used. Should there be only one consistent interpretation the table constitutes a set of predictions, since an interpretation can assign at most one D_s value to each quantity. Otherwise, the interpretations associated with each possible value of a measurable parameter are presented as a diagnostic. GIZMO currently it makes no further use of the results itself, although the results are presented to the user on request.

10.3 Three containers

This example consists of three containers F, G, and H, each containing some water, as depicted in figure 40. F and G are connected by fluid path P1 and G and H are connected by fluid path P2. Figure 55 describes the scenario. Suppose we measure the level in G and find that it is decreasing. What could be

---

1. A nogood set is a collection of premises which are mutually contradictory. Truth-maintence systems usually include some means of annotating these sets when discovered, to prevent wasting extra effort deriving their consequences.

Fig. 54. Diagnosis algorithm

```
;Let OBS be the set of measurements taken
;Let INTERPS be the set of consistent interpretations of OBS

Procedure Diagnosis

        1. Let NEW be empty.  For each type Q of measurable parameter,

           1.1 For each individual I that has Q,

               1.1.1 If Q(I) is not constrained by OBS, add it to NEW

        2. For each quantity Q in NEW, construct a table entry
           consisting of three sets, INCREASE, DECREASE, and CONSTANT.

           2.1 For each interpretation I in INTERPS,

               2.1.1 If in I D_S[Q] = 1, add I to INCREASE

               2.1.2 If in I D_S[Q] = 0, add I to CONSTANT

               2.1.3 If in I D_S[Q] = -1, add I to DECREASE

               2.1.4 If in I D_S[Q] is unknown, add it to
                     INCREASE, DECREASE, and CONSTANT.

        3.  If there is just one interpretation in INTERPS, present
            the table contents as predictions.

        4.  Otherwise, present the table contents as a diagnosis.
```

causing this?

To begin with, we must find the possible view and process instances. If we perform elaboration on the situation `3C-START`, we will get view instances representing each state of water (the only substance in this domain) in each container and representing the possibility for flow through the two paths. The only process instances will be fluid flow, one instance corresponding to flow in each direction of each path. No heat paths were specified, hence no instances of heat flow were found, and therefore no instances of boiling exist. Figure 56 provides the names of these instances. The initial conditions in the scenario ensure that only water in the liquid form exists, and we will further assume that all the paths can support flow in all directions (i.e., that all the instances of `liquid-flow-supporting` are active). The only instances whose statues are not known are the four process instances, and so they will comprise the search space.

If we trace the interpretation search, we see:

```
Beginning search on 3C-START...
Status Assignment 1
     All assumed inactive.
   -- Inconsistent, for LOCAL reasons.
Status Assignment 2
     PI-3
   -- Inconsistent, for LOCAL reasons.
Status Assignment 3
     PI-2
   -- Consistent.
```

Fig. 55. Three Containers Scenario

```
(defscenario three-containers
  (Individuals F G H P1 P2
               (c-s WATER LIQUID F)
               (c-s WATER LIQUID G)
               (c-s WATER LIQUID H))
  (Facts (Container F)
         (Container G)
         (Container H)
         (Fluid-Path P1)
         (Fluid-Path P2)
         (Fluid-Connection P1 (c-s WATER LIQUID F) (c-s WATER LIQUID G))
         (Fluid-Connection P1 (c-s WATER LIQUID G) (c-s WATER LIQUID F))
         (Fluid-Connection P2 (c-s WATER LIQUID G) (c-s WATER LIQUID H))
         (Fluid-Connection P2 (c-s WATER LIQUID H) (c-s WATER LIQUID G)))
  (Always (Equal-To (A (max-height P1)) (A ((height bottom) G)))
          (Equal-To (A (max-height P1)) (A ((height bottom) F)))
          (Equal-To (A (max-height P2)) (A ((height bottom) G)))
          (Equal-To (A (max-height P2)) (A ((height bottom) H)))
          (Aligned P1)
          (Aligned P2))
  (In-Situation 3C-START
                (Individuals F G H P1 P2
                             (c-s WATER LIQUID F)
                             (c-s WATER LIQUID G)
                             (c-s WATER LIQUID H))
                (Facts (Greater-than (A (amount-of-in water liquid F))
                                     ZERO)
                       (Greater-than (A (amount-of-in water liquid G))
                                     ZERO)
                       (Greater-than (A (amount-of-in water liquid H))
                                     ZERO)
                       (Equal-to (A (amount-of-in water gas F))
                                 ZERO)
                       (Equal-to (A (amount-of-in water gas G))
                                 ZERO)
                       (Equal-to (A (amount-of-in water gas H))
                                 ZERO))))
```

```
Status Assignment 4
     PI-1
   -- Consistent.
Status Assignment 5
     PI-1
     PI-3
   -- Consistent.
Status Assignment 6
     PI-1
     PI-2
   -- Consistent.
Status Assignment 7
     PI-0
   -- Inconsistent, for LOCAL reasons.
Status Assignment 8
     PI-0
     PI-3
   -- Inconsistent, for LOCAL reasons.
Status Assignment 9
     PI-0
     PI-2
```

Fig. 56. View and process instances for three containers scenario

```
For Situation 3C-START
  IVI-0: VIEW-INSTANCE(LIQUID-FLOW-SUPPORTING, C-S(WATER, LIQUID, F),
                       C-S(WATER, LIQUID, G), P1)
  IVI-1: VIEW-INSTANCE(LIQUID-FLOW-SUPPORTING, C-S(WATER, LIQUID, G),
                       C-S(WATER, LIQUID, F), P1)
  IVI-2: VIEW-INSTANCE(LIQUID-FLOW-SUPPORTING, C-S(WATER, LIQUID, G),
                       C-S(WATER, LIQUID, H), P2)
  IVI-3: VIEW-INSTANCE(LIQUID-FLOW-SUPPORTING, C-S(WATER, LIQUID, H),
                       C-S(WATER, LIQUID, G), P2)
  IVI-4: VIEW-INSTANCE(CONTAINED-STUFF, F, WATER, LIQUID)
  IVI-5: VIEW-INSTANCE(CONTAINED-STUFF, F, WATER, GAS)
  IVI-6: VIEW-INSTANCE(CONTAINED-STUFF, G, WATER, LIQUID)
  IVI-7: VIEW-INSTANCE(CONTAINED-STUFF, G, WATER, GAS)
  IVI-8: VIEW-INSTANCE(CONTAINED-STUFF, H, WATER, LIQUID)
  IVI-9: VIEW-INSTANCE(CONTAINED-STUFF, H, WATER, GAS)
  PI-0: PROCESS-INSTANCE(LIQUID-FLOW, C-S(WATER, LIQUID, F), C-S(WATER, LIQUID, G), P1,
                         IVI-0)
  PI-1: PROCESS-INSTANCE(LIQUID-FLOW, C-S(WATER, LIQUID, G), C-S(WATER, LIQUID, F), P1,
                         IVI-1)
  PI-2: PROCESS-INSTANCE(LIQUID-FLOW, C-S(WATER, LIQUID, G), C-S(WATER, LIQUID, H), P2,
                         IVI-2)
  PI-3: PROCESS-INSTANCE(LIQUID-FLOW, C-S(WATER, LIQUID, H), C-S(WATER, LIQUID, G), P2,
                         IVI-3)
```

```
   -- Consistent.
There are 5 consistent interpretations.
```

Notice that the generator is informed by the constraint that the quantity conditions be consistent, for although combinatorially sixteen status assignments are possible, only the nine assignments which result in unique inequality relationships between pressures are generated. Of these nine, four are inconsistent with the measured decrease in the level in G. The consistent interpretations are shown graphically in figure 57.[1] If we ask for a diagnosis, we get:

```
 Suggested Measurements:

   For (M LEVEL(C-S(WATER, LIQUID, F)) 3C-START):
Increasing indicates M-I-5, M-I-4, M-I-3.
Constant indicates M-I-2.
Decreasing indicates M-I-8.

   For (M LEVEL(C-S(WATER, LIQUID, H)) 3C-START):
Increasing indicates M-I-8, M-I-5, M-I-2.
Constant indicates M-I-3.
Decreasing indicates M-I-4.
```

Suppose we measure the level in F and find that it is decreasing. If we trace the interpretation search again, we see:

---

1. Notice that if we knew the measurement was taken over an interval of time rather than an instant, we could further rule out M-I-2 and M-I-3, since they describe situations that can only last for an instant. GIZMO's algorithms assume the measurement was made at an instant.

Fig. 57. Consistent interpretations for D_S[level(G)] = -1
The active instances of fluid flow are indicated by arrows pointing in the direction of the flow.

```
M-I-2:          F     G --> H

M-I-3:          F <-- G     H

M-I-4:          F <-- G <-- H

M-I-5:          F <-- G --> H

M-I-8:          F --> G --> H
```

```
Beginning search on 3C-START...
Status Assignment 1
     PI-3
Status Assignment 2
     PI-2
   -- Inconsistent, for LOCAL reasons.
Status Assignment 3
     PI-1
   -- Inconsistent, for LOCAL reasons.
Status Assignment 4
     PI-1
     PI-3
   -- Inconsistent, for LOCAL reasons.
Status Assignment 5
     PI-1
     PI-2
   -- Inconsistent, for LOCAL reasons.
Status Assignment 6
     PI-0
Status Assignment 7
     PI-0
     PI-3
Status Assignment 8
     PI-0
     PI-2
   -- Consistent.
There is one consistent interpretation.
```

The assignments where no comment was made as to consistency were immediately ruled out by nogood sets created during the previous search, thus only five status assignments were considered in detail. Of these five, only M-I-8 is consistent with both measurements. If we ask for a new diagnosis we get a prediction:

```
There is just one consistent interpretation.
   It predicts:
     (M Ds[LEVEL(C-S(WATER, LIQUID, H))] 3C-START) = 1
```

Given the interpretations depicted above, this answer is correct.

11. Envisioning

Envisioning is a technique that generates all possible outcomes of some initial situation. Although envisioning is important in its own right as a means of making predictions, it also illustrates how the basic QP deductions can be weaved together to perform more complex deductions. This chapter describes the basic envisioning algorithm and discusses two critical problems which arise in envisioning -- namely how to match situations and how to decide what a situation looks like after changes have occured, the frame problem for simulation. The results of the algorithm on several examples are presented. Also rules are described for summarizing the envisionment in order to draw conclusions about final states, changing equilibriums, and oscillations.

11.1 The algorithm

Figure 58 describes the basic algorithm. Roughly, here's how it works. If the initial situation is not completely determined, it is either because the status of some process or view instance is unknown or some influences are unresolved. In this case, the envisioner creates new situations to represent the various possibilities (step 2). The bulk of the envisioner consists of two loops. The outer loop uses limit analysis to determine what state changes can occur for each situation. The inner loop constructs the situation which would result from each quantity hypothesis, testing to see if that situation has been generated before. If it has, then the fact that the transition being considered leads from the current state to the state matched is noted. Otherwise, the new state (or states, if some influences cannot be resolved) representing the results of that change is added to the list of situations generated so far and queued so that changes in it will be determined in turn.

The sections which follow describe certain aspects of this algorithm in detail.

11.1.1 Finding completions

Indeterminacy can arise during envisioning in two ways. First, the initial state may not be completely determined (step 2). Second, influence resolution may yield ambiguities, preventing full knowledge of the D_s values for a situation (step 3.2.3.2). In both cases, new situations are constructed which represent the different alternatives for the undetermined information. The situations which represent alternative status assignments are called *s-completions*. The situations which represent alternative influence resolutions are called *r-completions*.

S-completions are generated by a dependency-directed search over the unknown status assignments, creating a situation for each consistent set. A similar dependency-directed search over unknown D_s values is used to generate r-completions, with one additional subtlety. Some possible values are ruled out by continuity constraints - if, for example, a D_s value was -1 in a previous situation, then by continuity it cannot be 1 in the next situation. These constraints are honored by passing in assumptions about which D_s values are illegal due to continuity constraints.

Fig. 58. Envisioning algorithm

```
Procedure Envision

    1. Let S be the initial situation and the set SITUATIONS initially
       be empty.

    2. If the process structure or view structure of S is incomplete,
       then let QUEUE = SITUATIONS = R-COMPLETIONS(S-COMPLETIONS(S)).

       if any Ds values in S are unknown, let
       QUEUE = SITUATIONS = R-COMPLETIONS(S)

       otherwise, let QUEUE = SITUATIONS = {S}

    3. While QUEUE is not empty, let S1 = first(QUEUE) and
       QUEUE = rest(QUEUE).

       3.1. Perform limit analysis on S1.  Initialize QH-QUEUE
            to be the set of quantity hypothesis which result.

       3.2. While QH-QUEUE is not empty, let
            QH = head(QH-QUEUE), QH-QUEUE = tail(QH-QUEUE)

            3.2.1 Let S2 = NEXT-SITUATION(S1, QH).

            3.2.2 If any situation S3 ∈ SITUATIONS matches S2,
                  install a transition pointer from S1 to S3 via QH.

            3.2.3  Otherwise, install a transition pointer from S1 to S2
                   via QH.

                   3.2.3.1 If S2 is r-complete then add S2 to QUEUE and
                           SITUATIONS.

                   3.2.3.2 If S2 is not r-complete then add S2 to
                            SITUATIONS and its r-completions to
                            SITUATIONS and QUEUE.
```

11.1.2 Matching

Matching situations is simple. Two situations are identical if they contain the same individuals, have the same view and process structures, if each corresponding quantity has the same D_s value, and if each quantity space is the same. In practice it is simpler to match over the contents of the situation's comparison table rather than the quantity spaces, since it contains the union of the information in the quantity spaces. For conciseness, we will call this set of inequality information the *qstate* of the situation.

11.1.3 Temporal inheritance

Deducing what the world looks like after some change has occured is a form of the frame problem. Using the QP ontology greatly simplifies this problem. We will stay entirely within QP theory by only considering the changes in inequalities predicted by limit analysis, and ignoring changes in preconditions that might be predicted by external theories.[1]

The algorithm is formulated in terms of *temporal inheritance*, carefully making assumptions about which facts will hold in a new situation. Three remarks are in order before describing the procedure in detail. First, we will call an individual whose existence is predicated on the status of a view or process instance a *dynamic* individual, and a *static* individual otherwise. An example of a dynamic individual is the water in a container, and an example of a static individual is the container itself. Second, we will assume that, unless we know otherwise, individuals which exist remain in existence. Third, note that the inequality relationships which comprise the qstate can be divided into two classes, those which are mentioned as possibly changing by some quantity hypothesis and those which aren't. The subset of the qstate which might change is called Ω. Importantly, if we assume that the change represented by a particular quantity hypothesis occurs, we are also assuming that the changes it explicitly mentions occur and no others from Ω change. This follows from the definition of a quantity hypothesis.

Figure 59 describes the temporal inheritance algorithm. We begin by constructing a situation and slices to represent the individuals (including view and process instances) after the change. Constructing a slice, of course, is not tantamount to assuming that the individual actually exists. While we can predict the comings and goings of dynamic individuals, any changes in the existence of static individuals must be predicated on considerations outside QP theory. Step 2 acknowledges this fact by assuming that any static individuals remain in existence. Step 3 recognizes that, on the whole, changes in preconditions cannot be predicted within QP theory. There are certain exceptions (such as the `Direction-Of` relationship introduced when discussing motion in section 8.3), and these are explicitly marked as being deducible and thus are ignored by the temporal inheritance algorithm because they will automatically be recomputed from other information for the new situation.

Step 4 simply enforces the requirement that the quantity hypothesis `QH` really is the change which occurs - the subset of the relationships in Ω which it mentions change, and no others in Ω do. A contradiction at this point indicates that `QH` represents an impossible change. Step 5 implements the persistence assumption for dynamic individuals; a dynamic individual will vanish only if the inequalities it depends on change. Step 6 implements the persistence assumption for inequalities; namely that an inequality will remain true unless you explicitly know (or have deduced that) it has changed.

The motivation for the particular ordering in these steps can best be illustrated by example. Clearly the static individuals must be around to determine whether or not view and process instances can be active (since being active implies the instance exists, and the instance exists if and only if the individuals which it occurs among do), and hence whether or not dynamic individuals can exist. Consider a can containing water

---

1. While predicting that preconditions will change is generally outside the relm of QP theory, it appears that determining the effects of such changes on QP descriptions would be a straightforward modification of the algorithm presented here.

Fig. 59. Temporal inheritance

```
Procedure Next-Situation(S, QH)

        1. Construct a situation NEW.

        2. For each static individual i in S,
           assume i exists in NEW.

        3. For each fact f which serves as a precondition
           in S which cannot be recomputed, assume f holds
           between the corresponding slices of NEW.

        4. Assume the new inequalities specified by QH hold in the
           new situation, and that all inequalities in Ω but
           not specified by the QH hold in the new situation.

        5. When consistent, assume that dynamic individuals
           which exist in S exist in NEW.

        6. When consistent, assume that the inequality relationships
           in the qstate but not in Ω still hold.

        Note: If any required assumption (steps 2, 3, and 4) leads to a
              contradiction, then the QH represents an inconsistent change.
```

but no steam which is being heated on a stove. If we fail to distinguish between the inequalities in Ω and its complement with respect to the qstate (in effect, merging steps 4 and 6), then we could never conclude that the water would boil, because boiling forces the existence of steam in the can, which would conflict with the inherited assumption that the amount of steam in the can was zero. Suppose we inherited the existence of dynamic individuals before updating the changed inequalities (i.e., reverse steps 4 and 5). Then we could never conclude that, given boiling occurs, that the water will all boil away, since the assumption that the water exists will contradict the assumption that the amount of water becomes zero. Steps 5 and 6, on the other hand, can safely be performed in either order, since any changes in existence will be predicated on the set of inequalities that change.

11.2 Summarizing behavior

Envisioning results in a description of all behaviors possible from the initial situation. This description, the *envisionment*, can be used to directly answer some types of questions. To determine if a situation with particular properties is possible, for instance, one needs only to inspect the envisionment to see if a state having those properties is present. If such a state is present then the situation is possible, and if it isn't then, because the envisionment is complete, the situation is impossible. However, further processing is required to answer other kinds of interesting questions, such as whether or not the system being analyzed will oscillate or what its eventual fate will be. In addition, the envisionment is often large and complex; summarization techniques are required to make some kinds of dynamical information explicit, such as changing equilibriums.

The envisionment is a rooted, directed graph, with the possible situations comprising the nodes and

the transitions between them (changes represented by quantity hypotheses) constituting the links. In earlier work on envisioning [Forbus, 1981a], dynamical properties were identified with graph-theoretic properties of the envisionment. The final states a system may take on are identified with states that have no links leading out. Oscillations were identified with cycles in the graph. On the whole this same theory works with the envisionments produced with QP theory. Two new factors require modifications, however. First, the phenomena of stutter means that not all cycles correspond to oscillations (see section 5.1). Second, ambiguities can arise in the D_s values, resulting in some links and states which represent lack of knowledge rather than state transitions.

Figure 60 describes GIZMO's summarization algorithm. The first step is to make a copy of the envisionment that can be mutated to serve as the summary. Then the situations which represent r-completions of some situation are merged together to reduce the branching factor of the graph (step 2). Since stutter cycles can easily be distinguished, it makes sense to find them next and replace those states with a description of the overall behavior manifested during the cycle (Steps 3 and 4). Step 5 simply finds and records all remaining cycles in the graph. Finally, all states with no links leading out are found and marked as final states.

As mentioned previously, stutter can be distinguished from oscillation because every state in a stutter cycle will only last an instant. Stutter often appears in conjunction with situations that are incomplete in D_s values. Rather than find distinct stutter cycles for each r-completion and merge them later, we just include all the r-completions in the same summary state.

The following heuristics have been developed for computing the summary of a stutter cycle. The process and view structure of the summary state is taken to be the union of the view and process structures for the states of the cycle. If the D_s value for some quantity is the same for all states in the cycle, then the D_s value for that quantity in the summary will be that value. If the D_s value has a net change in a single direction (such as being 1 in some states and 0 in the rest), then the value for the summary will be the value of the net change. It is less clear what to do if the D_s value takes on both -1 and 1 as values during the cycle. One possible

---

Fig. 60. Summarization algorithm

```
Procedure SUMMARIZE

        1. Build a copy ENV of the envisionment.

        2. Merge all r-completions

        3. Find all stutter cycles in ENV

        4. For each stutter cycle found,

           4.1 Compute a situation which describes the overall
               behavior during the cycle.

           4.2 Replace the cycle with the summary situation.

        5. Find and mark all oscillations.

        6. Find and mark final states.
```

stratagy is to examine situations before and after the cycle in order to ascertain the net change in it during the cycle. GIZMO, however, simply marks the D_s value for the summary as unknown.

These rules are viewed as heuristics because it is not clear they will always work. For instance, taking the process structure of the summary to be the union of the process structures for the cycle could lead to a situation that normally would be considered inconsistent. I have not yet constructed such a counterexample, nor have I been able to prove that such examples don't exist. It also may not matter, depending on the kinds of questions the summary is used to answer. Clearly the issue requires further study.

11.3 Examples

11.3.1 Two Containers

This example illustrates the basics of the envisioning algorithm. Figure 16 depicts the situation graphically, and figure 61 shows the scenario specifying it. Two containers, named F and G, are connected by a fluid path P1. The scenario specifies that initially, each container contains some water in the liquid form and none in the gas form, that the heights of the bottoms of F and G are at the same height as P1, and that the level of the water in F is greater than the level of the water in G. We will also assume that P1 is aligned, and that the pressure in F initially is greater than the pressure in G. Figure 62 plots the situations that result. Here is

---

Fig. 61. Two container scenario

```
(defscenario two-containers
  (Individuals F G P1
               (c-s WATER LIQUID F)
               (c-s WATER LIQUID G))
  (Facts (Container F)
         (Container G)
         (Fluid-Path P1)
         (Fluid-Connection P1 (c-s WATER LIQUID F) (c-s WATER LIQUID G))
         (Fluid-Connection P1 (c-s WATER LIQUID G) (c-s WATER LIQUID F)))
  (Always (Equal-To (A (max-height P1)) (A ((height bottom) G)))
          (Equal-To (A (max-height P1)) (A ((height bottom) F)))
          ;assume they are on a flat table
          (Equal-To (A ((height bottom) F)) (A ((height bottom) G)))
          (Aligned P1))
  (In-Situation 2c-start
                (Individuals F G P1
                             (c-s WATER LIQUID F)
                             (c-s WATER LIQUID G))
                (Facts
                  ;there is water
                  (Greater-than (A (amount-of-in WATER LIQUID F))
                                ZERO)
                  (Greater-than (A (amount-of-in WATER LIQUID G))
                                ZERO)
                  ;No steam
                  (Equal-to (A (amount-of-in WATER GAS F))
                            ZERO)
                  (Equal-to (A (amount-of-in WATER GAS G))
                            ZERO))))
```

Fig. 62. Two containers plot

GIZMO's description of what occurs:[1]

```
      In situation S0, there is a flow of the water in F
to G.  The height of the top of the water in G, the volume
of the water in G, the pressure of the water in G, the level
of the water in G, and the amount of the water in G are
increasing.  The height of the top of the water in F, the
volume of the water in F, the pressure of the water in F, the
level of the water in F, the flow rate of the a flow of the
water in F to G, and the amount of the water in F are decreasing.
 All other quantities are constant.
      After some time, the pressure of the water in F
and the pressure of the water in G become equal.  This
leads to situation S3.

      In situation S3, no processes are acting.  The
pressure of the water in F equals the pressure of the water
in G.  All quantities are constant.
      No changes are possible.
```

Note that the assumptions have ruled out two possibilities. First, the reason that the amount of water in F cannot go to zero is that in S0 (a copy of 2c-start) the pressure of the water in F is greater than the pressure of the water in G, while if the water were to vanish, there would be a time in which the pressure in F was less than the pressure in G (from the correspondences linking height with level and level with pressure). That cannot be, since it must first pass through equality (by continuity), which would make a different hypothesis true.

---

1. GIZMO has a very simple built-in text generator.

Second, it is not possible for the level in F to reach the maximum height of P1 (thus preventing flow) because the maximum height of P1 is the same as the height of the bottom of F. Thus GIZMO concludes that the only change possible is for the pressures to equalize.

11.3.2 Simple Boiling

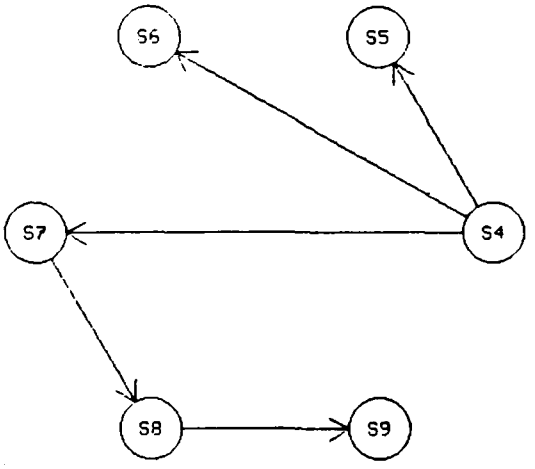
This example consists of a can containing only liquid water sitting on a stove (see Figure 63). The burner of the stove forms a heat path between the water in the can and the stove (we will ignore any effects of heating the can itself). The stove is modelled as a temperature source, meaning that, unlike other objects, its temperature is not a function of heat or amount of stuff. Explosions are not modelled in this vocabulary, so we will assume that the can is closed to avoid any flows of steam. Figure 64 illustrates situations that result, and here is GIZMO's synopsis of them:

```
      In situation S4, there is a heat flow from STOVE to the
water in CAN.  The temperature of the water in CAN and the
heat of the water in CAN are increasing.  The heat of STOVE
and the flow rate of the heat flow from STOVE to the water in
CAN are decreasing.  All other quantities are constant.
      The possible changes are:
   1 After some time, the temperature of the water in CAN
     rises to the temperature of STOVE and the temperature
     of the water in CAN rises to the boiling temperature
     of the water in CAN.  This leads to situation S5.
   2 After some time, the temperature of the water in CAN
     rises to the temperature of STOVE.  This leads to
     situation S6.
   3 After some time, the temperature of the water in CAN
     rises to the boiling temperature of the water in CAN.
     This leads to situation S7.
```

Fig. 63. Boiling scenario

```
(defScenario Can-on-Stove
   (Individuals CAN
                (c-s WATER LIQUID CAN)
                (c-s WATER GAS CAN)
                BURNER
                STOVE)
  (Facts (Container CAN)
         (Heat-path BURNER)
         (Heat-Connection BURNER STOVE (c-s WATER LIQUID CAN))
         (Temperature-Source STOVE))
  ;the following two facts follow as consequences of
  ;unimplemented touch relationships
  (Always (Heat-Connection BURNER STOVE (c-s WATER GAS CAN))
          (Heat-Aligned BURNER)
          (Equal-to (A (temperature (c-s WATER LIQUID CAN)))
                    (A (temperature (c-s WATER GAS CAN)))))
  (In-Situation Start
                (Individuals CAN BURNER STOVE (c-s WATER LIQUID CAN))
                (Facts (Greater-Than (A (amount-of-in WATER LIQUID CAN)) ZERO)
                       (Equal-To (A (amount-of-in WATER GAS CAN)) ZERO)
                       (Less-Than (A (temperature (c-s WATER LIQUID CAN)))
                                  (A (temperature STOVE))))))
```

Fig. 64. Boiling Plot



In situation S5, no processes are acting. ZERO equals the amount of steam in CAN, ZERO is less than the amount of water in CAN, the boiling temperature of the water in CAN equals the temperature of the water in CAN, and the temperature of STOVE equals the temperature of the water in CAN. All quantities are constant.
No changes are possible.

In situation S6, no processes are acting. ZERO equals the amount of steam in CAN, ZERO is less than the amount of water in CAN, the boiling temperature of the water in CAN is greater than the temperature of the water in CAN, and the temperature of STOVE equals the temperature of the water in CAN. All quantities are constant.
No changes are possible.

In situation S7, there is a heat flow from STOVE to the water in CAN, the water in CAN is boiling, and a heat flow from STOVE to the steam in CAN. The pressure of the steam in CAN and the amount of the steam in CAN are increasing. The height of the top of the water in CAN, the volume of the water in CAN, the pressure of the water in CAN, the level of the water in CAN, the heat of STOVE, and the amount of the water in CAN are decreasing. All other quantities are constant.

After some time, the amount of water in CAN drops to ZERO. This leads to situation S8.

In situation S8, there is a heat flow from STOVE to the steam in CAN. The temperature of the steam in CAN, the pressure of the steam in CAN, and the heat of the steam in

```
CAN are increasing.  The heat of STOVE and the flow rate of
the heat flow from STOVE to the steam in CAN are decreasing.
 All other quantities are constant.
      After some time, the temperature of the steam in CAN
rises to the temperature of STOVE.  This leads to situation S9.


      In situation S9, no processes are acting.  ZERO is
less than the amount of steam in CAN, ZERO equals the
amount of water in CAN, and the temperature of STOVE
equals the temperature of the steam in CAN.  All
quantities are constant.
      No changes are possible.
```

Examining the envisionment step by step, we see that GIZMO deduces that, at first, only heat flow between the container and the water occurs. This flow occurs because there is a temperature difference between the water and the stove. Since initially no steam exists, then boiling cannot be happening since boiling produces steam. Either the heat flow will stop (situations S5 and S6), if the temperature of the stove is less than or equal to the boiling temperature of the water, or boiling will occur (situation S7). If boiling occurs then steam will come into being. Since we are ignoring flows out of the container or any indirect effects of pressure on the boiling temperature, the next thing that happens is that the water vanishes (situation S8), ending the boiling. The heat flow from the stove to the steam will continue, raising the steam's temperature until it reaches that of the stove (situation S9).

11.3.3 Three containers

So far, the examples have exhibited very simple behavior. Few alternative changes were possible, and each change resulted in situations with all D_s values determined. Let us reexamine the three containers example used to introduce the idea of stutter in section 5.1. The scenario describing the situation is just the scenario in section 10.3. The situation plot is illustrated in figure 65. The dashed lines indicate links between a situation and its r-completions. Note that two stutter cycles exist, corresponding to the prospect of one flow stopping before the other. Figure 66 plots GIZMO's summary of the situation. Notice that the two collections of stutter cycles have been merged into two distinct states, making the envisionment much simpler. Here is GIZMO's explanation of the summary:

```
      The final state is S3-0.
      The stutter cycles are summarized as S17 and S16.

      In situation S0-0, there is a flow of the water in G to F
and a flow of the water in G to H.  The height of the top
of the water in H, the height of the top of the water in F,
the volume of the water in H, the volume of the water in F,
the pressure of the water in H, the pressure of the water in F,
the level of the water in H, the level of the water in F, the
amount of the water in H, and the amount of the water in
F are increasing.  The height of the top of the water in G,
the volume of the water in G, the pressure of the water in G,
the level of the water in G, the flow rate of the flow of the
water in G to H, the flow rate of the flow of the water in G to F,
and the amount of the water in G are decreasing.  All other
quantities are constant.  The possible changes are:
   1 After some time, the pressure of the water in G
     equals the pressure of the water in H and the pressure
```

Fig. 65. Three containers plot

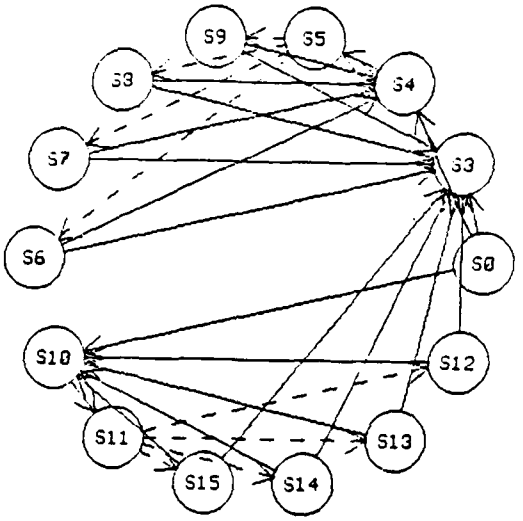
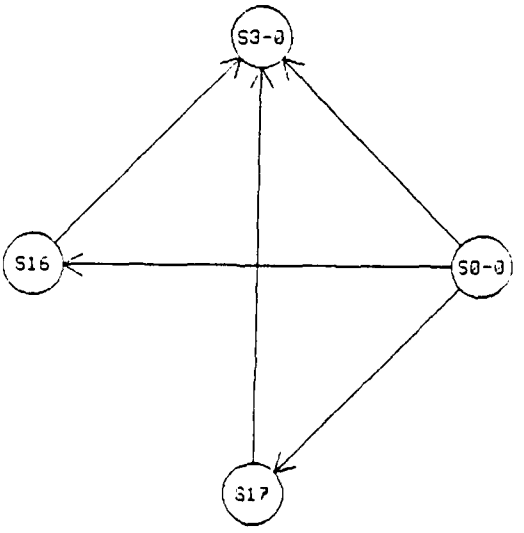


Fig. 66. Three containers summary plot



```
   of the water in F equals the pressure of the water in G.
   This leads to situation S3-0.
2  After some time, the pressure of the water in G
   equals the pressure of the water in H.   This
   leads to situation S16.
3  After some time, the pressure of the water in F
   equals the pressure of the water in G.   This
   leads to situation S17.


      In situation S3-0, no processes are acting.   The
pressure of the water in G equals the pressure of the water in H,
the pressure of the water in F equals the pressure of the water
in G, ZERO equals the amount of steam in H, ZERO equals the
amount of steam in G, and ZERO equals the amount of steam in F.
All quantities are constant.
      No changes are possible.

      In situation S16, there is a flow of the water in G to F
and a flow of the water in H to G.   The pressure of the water
in H starts just barely above the pressure of the water in G.
The amount of the water in F, the level of the water in F,
the pressure of the water in F, the volume of the water in F,
and the height of the top of the water in F are increasing.
The amount of the water in G, the amount of the water in H,
the flow rate of the flow of the water in G to F, the level
of the water in G, the level of the water in H, the pressure
of the water in G, the pressure of the water in H, the volume
of the water in G, the volume of the water in H, the height
of the top of the water in G, and the height of the top of the
water in H are decreasing.   With one exception, the other
quantities aren't changing.   The flow rate of the flow of the
water in H to G might be changing.
      After some time, the pressure of the water in G
equals the pressure of the water in H and the pressure of the
water in F equals the pressure of the water in G.   This
leads to situation S3-0.

      In situation S17, there is a flow of the water in F to G
and a flow of the water in G to H.   The pressure of the water
in F starts just barely above the pressure of the water in G.
The amount of the water in H, the level of the water in H,
the pressure of the water in H, the volume of the water in H,
and the height of the top of the water in H are increasing.
The amount of the water in F, the amount of the water in G,
the flow rate of the flow of the water in G to H, the level
of the water in F, the level of the water in G, the pressure
of the water in F, the pressure of the water in G, the volume
of the water in F, the volume of the water in G, the height
of the top of the water in F, and the height of the top of the
water in G are decreasing.   With one exception, the other
quantities aren't changing.   The flow rate of the flow of the
water in F to G might be changing.
      After some time, the pressure of the water in G
equals the pressure of the water in H and the pressure of the
water in F equals the pressure of the water in G.   This
leads to situation S3-0.
```

Notice that we do not know how the amount of G is changing during the dynamic equilibriums - a closer inspection of the original envisionment would reveal that it is either constant or falling, but cannot be rising.

11.3.4 Four blobs

Even very simple situations can give rise to complex stutter, making summarization rules a necessity. We will examine one of those situations now to see just how well the summarization heuristics work. Figure 67 describes an object G which is surrounded by, and in heat contact with, objects F, H, and I. Initially we will assume the temperature in G is highest, which means heat will flow out from G to F, H, and I. The envisionment is sufficiently complicated that examining the situation plot would tell us nothing -- it consists of 86 situations, linked by 181 quantity hypotheses and 150 r-completion links.

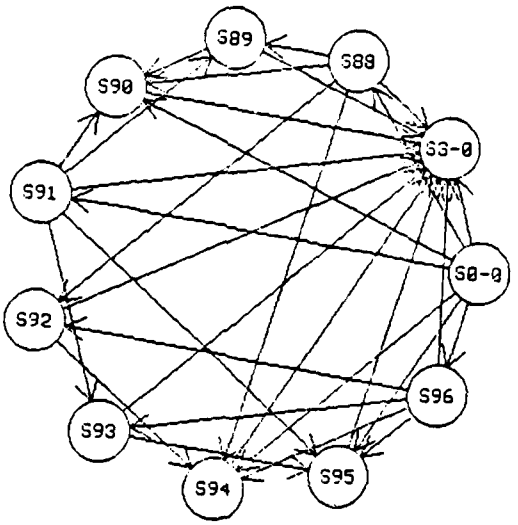
Figure 68 shows a plot of the envisionment's summary. It is still rather complicated (11 states), but it is almost an order of magnitude simpler than the original description. Here is GIZMO's synopsis of it:

```
      The final state is S3-0.

      The stutter cycles are summarized as S102, S101, S100, S99,
S98, S97, S96, S95, and S94.

      In situation S0-0, there is a heat flow from G to I, a heat flow
from G to H, and a heat flow from G to F.  The temperature of I, the
temperature of H, the temperature of F, the heat of I, the heat of H,
and the heat of F are increasing.  The temperature of G, the heat of
G, the flow rate of the heat flow from G to F, the flow rate of the
heat flow from G to H, and the flow rate of the heat flow from G to I
are decreasing.  All other quantities are constant.
      The next states can be S102, S101, S100, S97, S96, S94,
and S3-0.

      In situation S3-0, no processes are acting.  The
temperature of G equals the temperature of I, the temperature
of G equals the temperature of H, and the temperature of F
equals the temperature of G.  All quantities are constant.
      No changes are possible.
```

Fig. 67. Four blobs scenario

```
(defscenario four-containers
  (Individuals F G H I p1 p2 p3)
  (Facts (Piece-Of-Stuff F)
         (Piece-Of-Stuff G)
         (Piece-Of-Stuff H)
         (Piece-Of-Stuff I)
         (Heat-Path p1)
         (Heat-Path p2)
         (Heat-Path p3)
         (Heat-Connection P1 F G) ;everything is connected through g
         (Heat-Connection P1 G F)
         (Heat-Connection P2 G H)
         (Heat-Connection P2 H G)
         (Heat-Connection P3 G I)
         (Heat-Connection P3 I G))
  (Always (Heat-Aligned P1)
          (Heat-Aligned P2)
          (Heat-Aligned P3))
 (In-Situation 4C-START
               (Individuals F G H I P1 P2 P3)))
```

Fig. 68. Plot of four blob summary



In situation S94, there is a heat flow from G to H, a heat flow from G to F, and a heat flow from I to G. The temperature of I is just barely above the temperature of G. The heat of F, the heat of H, the temperature of F, and the temperature of H are increasing. The flow rate of the heat flow from G to H, the flow rate of the heat flow from G to F, the heat of G, the heat of I, the temperature of G, and the temperature of I are decreasing. With one exception, the other quantities aren't changing. The flow rate of the heat flow from I to G might be changing.

The next states can be S100, S98, S96, S95, and S3-0.

In situation S95, there is a heat flow from G to F, a heat flow from H to G, and a heat flow from I to G. The temperature of I is just barely above the temperature of G and the temperature of H is just barely above the temperature of G. The heat of F and the temperature of F are increasing. The heat of H, the heat of I, the temperature of H, and the temperature of I are decreasing. With some exceptions, the other quantities aren't changing. The flow rate of the heat flow from I to G, the temperature of G, the heat of G, the flow rate of the heat flow from H to G, and the flow rate of the heat flow from G to F might be changing.

The next states can be S96 and S3-0.

In situation S96, there is a heat flow from G to F, a heat flow from H to G, and a heat flow from I to G. The temperature of I is just barely above the temperature of G and the temperature of H is just barely above the temperature of G. The heat of F and the temperature of F are increasing. The flow rate of the heat flow from G to F, the heat of G, the heat of H, the heat of I, the temperature of G, the temperature of H, and the temperature of I are decreasing. With some exceptions, the other quantities aren't changing. The flow

rate of the heat flow from I to G and the flow rate of the heat flow from H to G might be changing.
The next state is S3-0.

In situation S97, there is a heat flow from G to I, a heat flow from G to F, and a heat flow from H to G. The temperature of H is just barely above the temperature of G. The heat of F, the heat of I, the temperature of F, and the temperature of I are increasing. The flow rate of the heat flow from G to I, the flow rate of the heat flow from G to F, the heat of G, the heat of H, the temperature of G, and the temperature of H are decreasing. With one exception, the other quantities aren't changing. The flow rate of the heat flow from H to G might be changing.
The next states can be S101, S99, S3-0, S96, and S95.

In situation S98, there is a heat flow from F to G, a heat flow from G to H, and a heat flow from I to G. The temperature of I is just barely above the temperature of G and the temperature of F is just barely above the temperature of G. The heat of H and the temperature of H are increasing. The heat of F, the heat of I, the temperature of F, and the temperature of I are decreasing. With some exceptions, the other quantities aren't changing. The flow rate of the heat flow from I to G, the temperature of G, the heat of G, the flow rate of the heat flow from G to H, and the flow rate of the heat flow from F to G might be changing.
The next states can be S100 and S3-0.

In situation S99, there is a heat flow from F to G, a heat flow from G to I, and a heat flow from H to G. The temperature of H is just barely above the temperature of G and the temperature of F is just barely above the temperature of G. The heat of I and the temperature of I are increasing. The heat of F, the heat of H, the temperature of F, and the temperature of H are decreasing. With some exceptions, the other quantities aren't changing. The flow rate of the heat flow from H to G, the temperature of G, the heat of G, the flow rate of the heat flow from G to I, and the flow rate of the heat flow from F to G might be changing.
The next states can be S101 and S3-0.

In situation S100, there is a heat flow from F to G, a heat flow from G to H, and a heat flow from I to G. The temperature of I is just barely above the temperature of G and the temperature of F is just barely above the temperature of G. The heat of H and the temperature of H are increasing. The flow rate of the heat flow from G to H, the heat of F, the heat of G, the heat of I, the temperature of F, the temperature of G, and the temperature of I are decreasing. With some exceptions, the other quantities aren't changing. The flow rate of the heat flow from I tc G and the flow rate of the heat flow from F to G might be changing.
The next state is S3-0.

In situation S101, there is a heat flow from F to G, a heat flow from G to I, and a heat flow from H to G. The temperature of H is just barely above the temperature of G and the temperature of F is just barely above the temperature of G. The heat of I and the temperature of I are increasing. The flow rate of the heat flow from G to I, the heat of F, the heat of G, the heat of H, the temperature of F, the temperature of G, and the temperature of H are decreasing. With some exceptions, the other quantities aren't changing. The flow rate of the heat flow from H to G and the flow rate of the heat flow from F to G might be changing.
The next state is S3-0.

In situation S102, there is a heat flow from F to G, a heat flow from G to I, and a heat flow from G to H. The temperature of F is

```
just barely above the temperature of G.  The heat of H, the heat of I,
the temperature of H, and the temperature of I are increasing.  The
flow rate of the heat flow from G to I, the flow rate of the heat flow
from G to H, the heat of F, the heat of G, the temperature of F, and
the temperature of G are decreasing.  With one exception, the other
quantities aren't changing.  The flow rate of the heat flow from F to
G might be changing.
     The next states can be S3-0, S100, S101, S98, and S99.
```

Despite the complexity, the results look somewhat reasonable. Importantly, every state has a consistent process structure -- the rules may someday fail, but they have survived this example.

11.3.5 Sliding block

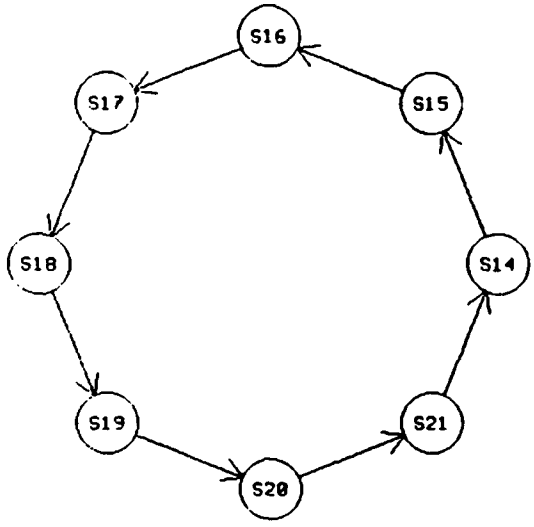
Let us turn to an example involving motion. Figure 69 describes a block connected to a spring. Figure 70 shows the situation plot, and here is GIZMO's synopsis of it:

```
     In situation S14, SPR is stretched.  There is B
accelerating to the left.  The velocity of B is decreasing.
 All other quantities are constant.
     Instantly, the velocity of B drops below ZERO.  This
leads to situation S15.


     In situation S15, SPR is stretched.  There is B moving
to the left and B accelerating to the left.  The force of SPR,
the force of B, and the acceleration of B are increasing.
The velocity of B, the position of B, and the length of
SPR are decreasing.  All other quantities are constant.
     After some time, the length of SPR drops to the rest
length of SPR and the force of B rises to ZERO.  This
leads to situation S16.


     In situation S16, SPR is relaxed.  There is B moving
to the left.  The force of SPR and the force of B are increasing.
 The position of B and the length of SPR are decreasing.
All other quantities are constant.
     Instantly, the length of SPR drops below the rest
length of SPR and the force of B rises above ZERO.  This
leads to situation S17.
```

Fig. 69. Sliding block scenario

```
(defScenario Kludge-oscillator
   (Individuals B
                SPR)
  (Facts (Block B)
         (Spring SPR))
  (Always (Connected-To-Spring B SPR))
  (In-Situation T0
                (Individuals B SPR)
                (Facts ;some initial conditions
                  (Equal-To (A (velocity B)) ZERO)
                  (Greater-Than (A (Position B)) ZERO))))
```

Fig. 70. Sliding block plot



In situation S17, SPR is compressed. There is B moving to the left and B accelerating to the right. The velocity of B, the force of SPR, the force of B, and the acceleration of B are increasing. The position of B and the length of SPR are decreasing. All other quantities are constant.

After some time, the velocity of B rises to ZERO. This leads to situation S18.

In situation S18, SPR is compressed. There is B accelerating to the right. The velocity of B is increasing. All other quantities are constant.
Instantly, the velocity of B rises above ZERO. This leads to situation S19.

In situation S19, SPR is compressed. There is B accelerating to the right and B moving to the right. The velocity of B, the position of B, and the length of SPR are increasing. The force of SPR, the force of B, and the acceleration of B are decreasing. All other quantities are constant.
After some time, the length of SPR rises to the rest length of SPR and the force of B drops to ZERO. This leads to situation S20.

In situation S20, SPR is relaxed. There is B moving to the right. The position of B and the length of SPR are increasing. The force of SPR and the force of B are decreasing. All other quantities are constant.
Instantly, the length of SPR rises above the rest

```
length of SPR and the force of B drops below ZERO.  This
leads to situation S21.


     In situation S21, SPR is stretched.  There is B
accelerating to the left and B moving to the right.  The
position of B and the length of SPR are increasing.  The
velocity of B, the force of SPR, the force of B, and the
acceleration of B are decreasing.  All other quantities are
constant.
     After some time, the velocity of B drops to ZERO.  This
leads to situation S14.
```

Since this cycle includes situations which last over an interval, this cycle is a true oscillation rather than stutter, as GIZMO points out below:

```
     There is no discernable final state.
     The states S14, S15, S16, S17, S18, S19, S20,
and S21 comprise an oscillation.
```

12. Discussion

This report has described Qualitative Process theory, which attempts to capture common sense reasoning about dynamics. Here we summarize the important ideas and discuss how they might be evaluated. Then the future directions suggested by this work are explored, followed by an examination of past and current work in this area.

12.1 Summary

- Our common sense theories about how things change in the physical world have a particular character. Physical processes are the mechanisms by which change occurs. Reasoning about processes -- their effects and limits -- form an important part of our common sense physical reasoning.

- Numerical values can be usefully represented by the quantity space, which describes the value of a number in terms of a partial order. The quantity space is the appropriate representation because processes usually start and stop when order relationships between particular quantities change.

- QP theory provides the means to draw several types of basic qualitative deductions, including describing what is happening in a physical situation (finding view and process structures), reasoning about the combined effects of several processes (resolving influences), and predicting when processes will start and stop (limit analysis). These basic deductions can be woven together to perform more complex inferential tasks, such as envisioning and measurement interpretation.

- QP theory can be used to model several interesting physical phenomena for common sense reasoning, including flows, state changes, motion, materials, energy, changing equilibria, and oscillation.

- QP theory provides a highly constrained account of physical causality (all changes are due to a finite vocabulary of processes) and a useful notation for expressing causal connections between quantities ($\propto_Q$).

- QP theory provides a structured role for the use of experiential and default knowledge in physical reasoning -- for example, in resolving influences and choosing or ruling out alternatives in limit analysis.

- QP theory partially specifies a language for writing qualitative dynamical theories. In particular, the primitives are simple processes and individual views, the means of combination are sequentiality and shared parameters, and the means of abstraction are naming these combinations, including encapsulating a piece of a history.

12.2 Has the thesis been proven?

There are two ways to judge QP theory. The first way is as a prescription for a representation language for programs which must reason about the physical world. The second way is as a psychological theory about the structure of people's common sense theories of dynamics. Since both considerations motivated the development of the theory, let us examine each in turn.

A representation language for qualitative physics should be judged by the range of phenomena it can be used to model, the deductions it sanctions, and how perspicuous its descriptions are. There are additional criteria concerning the esthetic of particular models written in the language (see [de Kleer and Brown, 1983]), but we will discuss these later.

To understand the range of phenomena QP theory can describe, let us consider what kinds of differential equations can be represented. First, we will restrict derivatives to be taken with respect to time. Next, note that derivatives in the equation (e.g., `dx/dt`) can be represented by introducing explicit quantities which are the derivative and making the original quantity be directly influenced by the derivative (e.g., let `v = dx/dt`, and `I+(x, v)`). If the derivative of some quantitiy is equal to a sum, each term of the sum can be represented by additional direct influences. Finally, note that qualitative proportionalities can be used to represent combinations other than addition (e.g., `dx/dt = f(y, z)` becomes $v \propto_{Q+} y \wedge v \propto_{Q+} z$). As mentioned previously, functions which are non-monotonic must be modelled by breaking them up into monotonic segments, using an individual view to limit the scope of applicability for each $\propto_Q$. By constructing the appropriate process and view vocabularies, any differential equation (with respect to time) can be represented. Since differential equations are the language of choice for "real" physics, this suggests the potential scope of QP theory is quite broad.

A more precise determination of the scope of the theory, however, rests on two additional factors. First, in "real" physics the knowledge of when the equations are appropriate isn't part of the formal theory. The information made explicit in QP theory's individual specifications, preconditions, and quantity conditions is usually specified informally. It might be the case that QP theory is inadequate for specifying these criteria. Second, it is possible that such criteria may always be specified in it, but in some cases only with great difficulty (much as it is theoretically possible to write an operating system in turing machine code, but is hardly something one would attempt in practice). The examples presented previously suggest the descriptive language of QP theory is both adequate and natural, but the question will only be settled by undertaking the construction of many domain models. Evaluating the usefulness and naturalness of the deductions it sanctions provides a similar problem; whether or not the basic deductions it provides can be composed to implement all the kinds of dynamical reasoning desired is an empirical matter, since we do not already have a full understanding of the class of deductions which comprise common sense physics. While the envisioning and measurement interpretation algorithms illustrate its promise, only by exploring additional styles of reasoning will its adequacy be demonstrated. Particular suggestions are made in section 12.3.3 below.

To be a good psychological account, QP theory must explain observed psychological phenomena, such as characteristic inferences, errors, and learning patterns. QP theory should also lead to falsifiable predictions. Making such predictions is complicated because people appear to have several different kinds of models for the physical world. Their models can vary widely in scope and completeness, and often contain inconsistencies. While the overall structure of people's knowledge about the physical world is not yet known, there are indications that people's models include descriptions of prototypical behaviors, which we call

protohistories (see [Forbus & Gentner, 1983] for details). Protohistories are (abstracted) memories of observed events; hence questions about behavior in familiar domains may be answered by referring to them rather than an explicit naive physics. Determining the extent to which protohistories verses an explicit naive physics is used in reasoning about the physical world is still an open empirical question.

There are features of QP theory which may lead to testable predictions. First, QP theory distinguishes between independent quantities (i.e., those which can be directly influenced) and dependent quantities (i.e., those which cannot). If a subject does not honor this distinction in making some deduction, then QP theory is not being used to draw it. QP theory is the only qualitative physics that makes this prediction, so if the distinction is honored then this provides support for QP theory. Second, QP theory suggests that an explicit notion of physical process is used in human thinking about the physical world. One way to test this is to look at protocols to see if processes are mentioned (although some effort is needed to distinguish actual deductive work from mere figures of speech). Another way is to make people extrapolate behavior in an unfamiliar domain, testing to see whether the representations they acquire include processes and whether these processes can be specified within QP theory.

One specific prediction is that stutter phenomena will occur in human reasoning This kind of "mythical oscillation" which represents changing equilibriums violates the classical model of continuity; we cannot account for people generating behavioral descriptions which include stutter by classical physics, nor even the other qualitative physics which have been developed (as we will see below). Observing this phenomena, however, won't necessarily be easy. If protohistories are used to generate the behavioral description, for instance, then stutter -- being unobservable -- will not appear. If no evidence of stutter is found, then it suggests that case 2 of the equality change law is not psychologically realistic.

So far, the only psychological experiments which have been performed are pilot studies aimed at understanding how physical domains are learned.[1] Preliminary indications are that the distinction between dependent and independent parameters is honored. When asked about how one parameter will change when another changes, subjects knowledgable in a domain balk -- even to the extent of refusing to answer -- if the parameter to be changed is a dependent one. Less knowledgable subjects appear to be less fastidious; they will answer questions (often incorrectly) no matter what quantities are involved. Clearly much more work will be needed to adequately test the psychological relevance of QP theory.

12.3 Future work

There are several directions suggested by this work, both extensions and applications. Let us examine them now.

---

1. These experiments are being designed and carried out by Dedre Gentner at BBN and Lance Rips at University of Chigago, as part of a study funded by Schlumberger-Doll Research Center.

12.3.1 Implementation

As written, GIZMO is simply too slow. Generating an envisionment with 12 states takes over an hour and 45 minutes, with the time increasing non-linearly with the number of states due to increased paging. This limits the complexity of the domain models and scenarios that can be explored. Applying QP theory to practical problems, of course, will require much better performance.

There is nothing inherently inefficient in reasoning with QP theory, indeed the present inefficiency is unsurprising given the exploratory nature of the program. But the ideas are now stable, and it is time to look into better implementation strategies. For instance, [de Kleer, 1984] describes a new variety of TMS (used in de Kleer's ENVISION program) which should prove useful. In addition to recording justifications, as most truth maintence systems do, de Kleer's TMS also records the various sets of assumptions which support a fact. This means that generating alternate states in measurement interpretation and envisioning can be performed "all at once", using a separate processing step to gather together collections of assumptions which correspond to consistent states. Judging from observed run-times of his system, it would appear that two orders of magnitude performance improvement is not out of the questions. There is some chance that the speed-up would be even greater, since his system is currently running on a slower LM-2.[1]

Such estimates should be viewed with caution, since QP theory takes on more modelling work than deKleer's ENVISION. However, the pre-computation of possible individuals performed in situation elaboration (section 9.1) might provide a simple way to use deKleer's TMS directly. Providing easy interaction between the QP module and external theories is also something that will require re-thinking in this new implementation.

12.3.2 Domain models

Developing qualitative models of the physical world around us is an intellectually demanding task; while QP theory makes this task easier, it still is not trivial. The particular domain models described in this thesis reflect more on the limited time available for development than on what is possible using QP theory.

There are several dimensions along which the present model of fluids can be improved. First, some theory of solids and the phase transistions involving them is needed. Second, the individuating criteria for fluids also needs to be more sophisticated, to model more complex types of mixtures (suspensions and solutions, for instance) and more varied mixture geometries (such as liquids which don't mix and bubbles). Third, the "kinematics" of fluids, the theory of fluid paths and heat paths, needs to be extended to cover real piping systems, flows caused by mixing, and flows through free space. Finally, the molecular collection ontology, which describes fluids in terms of "little pieces of stuff" that are imagined to cohere as they pass through a system, must be better developed and integrated with the contained fluid ontology.

The interaction between the contained fluid and molecular collection ontologies bears remarking. There are certain questions which can only be asked in the context of the molecular collection ontology, such as "What happens to the water fed into the boiler?" Other questions require using the molecular collection

---

1. GIZMO ran over five times faster when transported from an LM-2 to a Symbolics 3600.

ontology to generate an answer. Here is one such question: "Suppose the boiler's feedwater temperature increases. What happens to the temperature of the steam at the superheater outlet?" (see figure 71). The answer runs something like this:

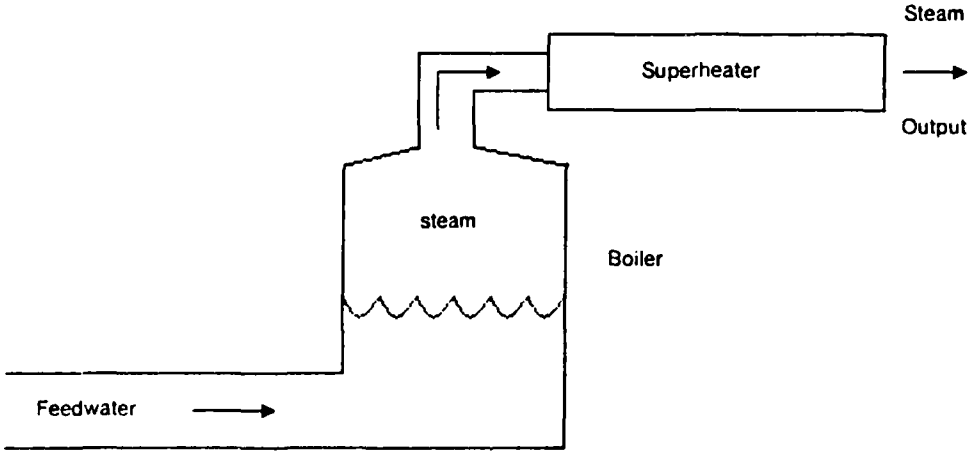
> *The increased feedwater temperature means less heat will be required to make it boil, hence the rate of steam generation will increase. This means it will be flowing through the superheater faster (assuming the load sinks it all), and since it remains in the superheater for a shorter time the amount of heat transferred will be less. Thus the temperature at the superheater outlet will decrease.*[1]

There are several interesting features of this description. First, the contained fluid ontology is clearly insufficient, since in it we cannot talk about stuff remaining in the superheater for less time, only "the steam in the superheater". Second, time and rate differences are explicitly mentioned, suggesting that differential qualitative analysis (see section 5.4) is being used. Finally, notice that the contained fluid ontology is still required to deduce the overall structure of what is happening in the system -- that heat is flowing to the water in the boiler (part of which is the now hotter feedwater) and that steam will flow from the boiler into the superheater and out to the load. Drawing these conclusions from an arbitrary piece of stuff seems well nigh impossible, since they depend on the properties of all of the stuff in a place and what that stuff is connected to.[2]

These considerations lead to an interesting conjecture about the relationship between the two fluid ontologies. In reasoning about a fluid system, it seems the contained-fluid ontology is constructed by default

---

Fig. 71. A Boiler Problem



---

1. This question is among the most difficult asked of trainees in the Navy Surface Warfare Officer's School propulsion engineering course.
2. At least, neither I nor anyone else to my knowledge has succeeded in doing so, despite much effort.

and used to establish what is happening in the situation (i.e., the process structure). If the question requires using the molecular collection ontology, the history for a piece of stuff is created by transformations operating on the contained fluid description. For example, if a piece of stuff is initially in a particular place and there is a flow out of that place, a motion episode is constructed to represent the fact that eventually those particular molecules will be found in the destination of that flow. If a boiling is occuring to the contained-liquid the collection is a part of, then a state-change episode will be added to its history, and so on. The questions about the molecular collection can then be answered using this history, either by accumulating properties along the history (such as temperature changes) or by performing differential qualitative analysis. Investigation into exactly what kinds of questions require the molecular collection ontology is proceeding, and computational experiments are planned.

The models of motion presented could also stand substantial improvement. The present model is impoverished due to a flaw in the implementation; until recently, there was no way for GIZMO to know that certain domain-dependent sets had to be updated as it went -- in this case, the set of forces on an object. This limitation has been fixed, and a better motion model is under development. Moving to more than one dimension, however, will lead to a new set of problems. In some cases (particularly mechanisms) it should be possible to break a multi-dimensional motion up into one-dimensional pieces which, when combined, yield the actual motion of the object.[1] As remarked in section 4.3, the most useful general stratagy might be to use the qualitative state representation of motion (see section 4.3.2) to generate descriptions of possible motion, and use QP theory to provide further analyses. But these are empirical questions.

Other domains should be explored as well. The most tempting domain is electricity, since it also minimizes kinematic information. Another interesting feature is that electronics is precisely where a device-centered ontology works best, because the high interconnectivity makes finding explict flow paths computationally intractible for circuits of moderate complexity. By contrasting device-centered models with process-centered models, we should be able to sharpen our criteria about when each is appropriate.[2]

12.3.3 Styles

This report explored two styles of reasoning, envisioning (as a means of prediction) and measurement interpretation. There are several other styles to explore as well. Here we will discuss extensions to measurement interpretation and prediction, and issues which arise in planning and design.

---

1. According to [Reuleaux, 1875] the ability to use such decompositions is a central constraint on the well-designed mechanism. Many mechanisms, however, do not have this property.

2. In [Forbus & Gentner, 1984], it is conjectured that people's early models of electricity are process-centered, and as more complex circuits are studied a device-centered model is generated. Exploring this ontological shift might improve our understanding of how people use both types of models.

12.3.3.1 Measurement interpretation

The theory presented here for interpreting measurements appears adequate for its intended purpose, although the pruning heuristics must still be tested empirically. It also appears that the theory for the "one look" case can be extended to interpret measurements taken across time. An interpretation would be generalized to a *history*, and measurements would correspond to partial information about this history. The description of measurements remains unchanged, the only difference being that differential distinguishability will be used over intervals of significant duration. There is an additional problem of segmentation, finding intervals where the process structure does not change. A heuristic is to use changes in D_s values as a minimal set of boundaries, since these must correspond to changes in the resolving of influences. Additional divisions may be necessary, because changes in unobserved quantities may take time to propagate to distinguishable changes in observed quantities -- for example, a stove may be on for some time before you deduce that fact by seeing steam pour out of a kettle on top of it.

The "one look" algorithm could be used to build interpretations for what is occuring during each episode implied by the boundaries. The pruning constraints and heuristics described above still hold -- even if we watch for five minutes, evaporation still won't empty a drinking glass. Because the episodes are connected, the interpretation for any particular episode has to be consistent with the interpretations for the ones around it. In particular, the interpretation for each episode has to correspond to a process structure implied by some limit hypothesis for the process structure implied by the interpretation of the episode before it. This constraint also suggests an alternate strategy - use the "one look" algorithm to develop an interpretation for the earliest set of measurements, then use limit analysis to construct the situations that could come next. If any of those situations are legitimate interpretations for the next set of measurements then continue, otherwise try a new interpretation for the initial set of measurements. The advantage of this stratagy is that it exploits the constraints imposed by the (hypothesized) initial conditions.

12.3.3.2 Prediction

Making predictions by envisioning is like problem-solving by exhaustively generating the search space -- it works, but surely we can generally get by with less work. Two kinds of prediction questions are "Can this state occur?" (questions of *attainability*) and "Where will the system end up eventually?" (questions of *disposition*). We consider attainability questions first.

In classical problem solving, heuristic search is used to construct a path from the initial state to the goal state. To cast prediction in this mold we must introduce some notion of operator. The obvious notion is that an occurrence of some process or processes should count as an operator that reduces the difference (described as quantity space values and the statuses of instances) between a situation and the goal. In many problem-solving situations it is more convenient to work backwards from the goal. For prediction this does not appear to be the case, since the number of potential operators is large and because it ignores the constraints imposed by the initial conditions. Working forward corresponds to selecting a subset of the possible changes presented by limit analysis to carry on with. While this might speed up finding a situation that does occur, if the situation doesn't occur (and if we don't introduce arbitrary resource limitations) then we will do the same amount of work as envisioning. This suggests developing seperate methods to quickly rule out states which cannot exist. If, for instance, there is no water in any form in a situation and no process

which generates it from some other substance, then a situation in which water appears is impossible. I suspect that ruling out possible states might profit from consequent reasoning, but I have no evidence one way or the other.

To figure out what final states a system might reach, we might try ignoring the initial conditions and determine all states that are possible from the individuals and instances in the elaborated situation.[1] Those states which have no consistent quantity hypotheses are possible final states. Now the problem is reduced to the problem of finding which of these states are attainable.

12.3.3.3 Planning

Another way we use our common sense knowlege of physics is in constructing, analyzing, and debugging plans that involve the physical world. Consider assembling a printed circuit board. If we were constructing an assembly plan from first principles, a fragment of our reasoning might look like this:

A component is connected by soldering it to pads; this requires melting solder that is in contact with both the pad and the wire. A soldering iron has a temperature which is greater than the melting point of solder, so placing it in contact with the wire and pad to establish a flow path for heat will do the job. However, the soldering iron's temperature is high enough to damage many of the components. So whatever path the soldering iron moves through had better not come too close to those components.

Two things have occured in this scenario: knowledge of objects and processes has been used to generate a piece of a plan for accomplishing a task (use a soldering iron to create a heat flow which will cause the solder to melt), and constraints are placed on any elaboration of the plan to avoid undesired effects (don't bring the iron near heat-sensitive components on the way). Suppose further that we have generated a complete plan, and executed it on a particular circuit board. The report we receive is that several heat-sensitive components on the board were damaged. We might then decide:

The heat-sensitive components were brought to too high a temperature sometime during assembly; the only time there was an influence on their temperature was during soldering. Next time I won't hold the iron on them for so long.

This example of debugging requires a differential analysis; we must find what aspects of the situation are responsible for the undesired event and figure out how to modify the plan to prevent them. QP theory should be useful in generating and analyzing such plans. The first step in plan generation might be to describe the desired behavior as a partial history for the objects involved. Initially plan steps might be generated by synthesizing difference operators from the process vocabulary and using these operators to create a plan by standard means-end analysis. Further constraints on the plan steps might be determined by

---

1. This is essentially how de Kleer's ENVISION proceeds -- it generates consistent states for the device network, and then finds all the transitions between them. In general it will do more work than the envisioning procedure presented here, because it will generate states that are unattainable from the initial state.

analyzing the possible transitions between distinct episodes of the planned behavior; the plan must be designed so that the transition to the next planned behavior is the one that actually occurs. Conditions to prevent, such as heat damage above, are modelled by requiring that certain transitions and processes not occur.

Debugging a failed plan (and sometimes elaborating a plan) requires differential diagnosis to figure out what should be changed so that what is desired occurs, as opposed to what actually occured (or might occur). Differential qualitative analysis should be a useful technique for providing such answers.

12.3.3.4 Design

Another interesting problem is design, the problem of developing specifications for a constructable artifact which exhibits a desired behavior. Aside from the importance of understanding the design process in general, examining mechanical design should illuminate two important issues in naive physics. First, mechanical design could drive the development of extension theories for quantities and functions, since estimates of empirical factors such as strength of materials often seem important in ruling out classes of designs. Second, evaluating potential designs will require a better understanding of how to integrate contraint-oriented reasoning with causal reasoning. Consider designs for perpetual motion machines. While many explanations of how they work can be rejected because they ignore certain physical processes (usually some form of friction), other explanations can only be rebutted by detailed consideration of energy conservation. While a start has been made on such reasoning (see section 4.5), much remains to be done.

12.3.4 Qualitative Kinematics

Qualitative theories of shape and space will be required in addition to a qualitative dynamics to construct a full naive physics. While the broad outlines for theories of space ([Forbus, 1981],[Simmons, 1984]) and shape ([Brady, 1984]) exist, we seem to be a long ways from theories which can duplicate the wide range of everyday phenomena people reason about. Even so, the eventual goal of uniting dynamics and kinematics suggests two constraints on the structure of a qualitative kinematics. First, the primary goal of a qualitative kinematics is to determine possible geometric interactions between objects. Examples include determining whether or not something will roll on a particular surface and what objects lie in a particular direction from another. Second, when possible, a qualitative kinematics should suggest ways of decomposing a multi-dimensional problem into a connected set of one-dimensional problems. For instance, when reasoning about what a rolling object will collide with, the various object boundaries along the direction of its path could be modelled as elements in a position quantity space, ordered by their distance along the path from the object. Limit analysis would then predict that it might collide, might stop rolling due to friction, or might stop just at the boundary of the first obstacle.

12.3.5 Learning

Unlike formal physics, most of our common sense models of the physical world appear to be derived from experience in the world. Since everyone does learn some version of naive physics, it seems a better candidate for exploring experiential learning than domains people don't learn without a teacher. Also, learning research is often hampered because the form of the target representation is itself a research problem.

Without claiming QP theory is the final word on qualitative dynamics, its existence makes studying how such knowlege is acquired more attractive.

A theoretical framework to account for human learning in physical domains is already under construction [Forbus & Gentner, 1984]. The framework proposes a learning sequence of four types of physical models in each domain, using concepts from QP theory to help describe the contents of the models and Gentner's *Structure-Mapping theory* of analogy and non-literal comparisons [Gentner, 1983] to help describe the computations that move a learner from one type of model to another. We have begun psychological experiments to explore this framework; computer experiments are also planned.

12.3.6 ICAI & engineering problem solving

Since many engineered devices are implemented as physical systems, QP theory should be useful in reasoning about them. One application is providing part of a representation language for intelligent computer-aided instruction (ICAI). An important part of an expert's knowledge of a system is a qualitative understanding of how it works. To the extent that QP theory models our qualitative understanding of dynamical systems, a program using it can generate explanations that a student will find easy to understand. Indeed, QP theory was developed in part to be used in the STEAMER project, whose goal is to provide instruction about steam propulsion plants for Navy trainees.[1] Only now are the domain models approaching the quality needed to provide such instruction, and better implementation techniques will of course be needed to provide the desired quality of interaction.

One interesting implementation strategy is to construct a *tutor compiler*. Current qualitative reasoning programs work from first principles in constructing explanations. This is analogous to setting a human domain expert down in front of a system he has never seen before and expecting him to understand it well enough to generate coherent explanations in real time. Using human instructors to teach this way is obviously a bad idea, so why should we expect our programs to do better? The alternative is to construct a program which takes as its input a system to be understood and a specification of the class of questions which are to be asked about it. The output of this program would be a specialized tutoring program that could, in real time, answer that class of questions about the system in question. This technique would have several advantages, for example, the qualitative reasoning system in the compiler itself need not be especially fast, and more sophisticated techniques for generating explanations could be employed than would otherwise be possible (such as McDonald's MUMBLE [McDonald, 1983]).

As extension theories are developed, QP theory should become useful in other kinds of engineering and control tasks as well. Individual views could be used to express desirable and undesirable operational characteristics. For example, in operating a boiler the fuel-air ratio must not become too rich or too lean; in either case smoke pours out the boiler stack, which is bad if you want not to be seen and combustion efficiency, hence fuel economy, will drop. A good boiler design will provide operating regions in which the individual views representing these undesired conditions are inactive. Similarly, these descriptions could be

---

1. The STEAMER project is a joint enterprise of Bolt, Beraneck, and Newman, Inc. and the Navy Personnel Research and Development Center. See [Stevens, et al., 1981] for an overview.

used in synthesizing control strategies, by determining what measurements indicate a state from which a view instance representing an undesirable condition will become active and what corrective action must be taken to ensure that the particular change will not occur.

Another interesting possibility is building a *hypothesizer*. A hypothesizer is an interpretation module which either takes measurements from operators of a system or gathers data itself from instruments, and will evaluate an operator's theories about what is happening in the system. Such a program could serve as a devil's advocate, pointing out inconsistencies in an operator's theory or suggesting alternate interpretations for the data. This would be useful because it seems that a common source of human error in operating complex systems (such as nuclear power plants) is premature commitment to a particular theory about the state of the system (see [Pew et al., 1982]). For example, the incident at the Three Mile Island reactor probably wouldn't have happened if the operators had thought of the alternate explanation for the overpressure in the reactor vessel -- that instead of being too high, the level of cooling water was too low, thus causing a boiling that raised the pressure.

12.3.7 Other applications

Historically, the success of differential equations in physics led to attempts to apply them to problems from other fields, such as economics. To the extent that differential equations prove useful in reasoning about a domain, QP theory should be similarly useful. In economics, for example, physical limitations often prove important. Storage capacities, transportation capacities, and rates of manufacture are important examples (see [Forrester, 1968], [Stansfield, 1980]).[1] The features which make qualitative models useful for physical reasoning, such as the ability to characterize the classes of things that can happen even with very little data, should be useful in other domains, especially in domains where numerical data is unreliable or hard to come by.

However, caution seems advisable in attempting such applications. There seems to be no real agreement on what mathematical descriptions are appropriate in economics, hence it will be hard to judge whether a qualitative model is correct. In addition, the very structure of the domain can change with time; for instance, the tax code can change. These factors make modelling economics much harder than modelling physics.

12.4 Previous Work

The first attempts to formalize processes modelled them as collections of interacting automata [Brown et al., 1973] or extended STRIPS-like operators [Hendrix, 1973]. Let us examine each in turn.

Brown's automata-based system was designed to generate explanations for intelligent

---

1. Interestingly, Samuelson was one of the first to describe the possibilty of using qualitative models and to point out that their inherent ambiguity would make prediction difficult. Subsequent developments in qualitative modelling, however, suggest his views were overly pessimistic, at least for physical domains.

computer-aided instruction. Quantities and processes were represented by individual automata whose states represented classes of values or activities (such as a quantity decreasing or a particular activity in a sequence occuring). Time was modelled by specifying that automata representing quantities changed instantly while automata representing processes took an interval of time to change. Although arbitrary LISP code was permitted in specifying state transitions, in practice state changes were predicated on state changes in other automata. While adequate for generating explanations of fixed phenomena, the automata representation is too brittle for most reasoning tasks. For example, there is no influence-like mechanism for dynamically combining effects, thus all interactions must be foreseen in advance by the model builder. The process models are similar to encapsulated histories, in that they presuppose the outcome of the activity of the processes they describe. Hence such models will be insensitive to changing conditions.

Hendrix's system was designed to provide a world model for robot planning. While a significant advance over the models of action available at the time, the importance of qualitative descriptions had not yet been understood. For example, the values of numbers were known real numbers, and relationships between parameters were expressed as numerical constraint equations. The process descriptions were used for simulation, solving simultaneous equations to determine precisely when the collection of active processes would change. Since the goal was to model general processes (non-physical as well as physical), add lists and delete lists were also used to specify effects. Qualitative Process theory, by using qualitative descriptions and focusing on physical processes only, can be used in several other kinds of deductions in addition to simulation, often requiring less information to draw interesting conclusions.

Recently several attempts have been made to model temporal reasoning, including [Allen, 1981], [McDermott, 1982]. Allen's model is the one assumed here, both because `meet` seems to be the appropriate relationship between pieces of a history and because modeling instants as "very short" intervals makes formalizing certain facts involving derivatives easier. McDermott's axioms for time contain several interesting ideas, including the chronicle representation of possible futures and its implications for planning. Unfortunately, McDermott expects too much of his temporal logic. For example, the logic includes the notion of a "lifetime", i.e., how long you can assume a fact to be true once you have observed it to be true. McDermott claims lifetimes must be provided outside the logic, by fiat ("The senses actually tell you about persistences") because having axioms that provide persistences could lead to contradictions. This ad hoc notion is needed precisely because the logic is developed independently from a theory of dynamics. Given a dynamics (and the ability to make closed world assumptions about individuals and relationships), we can deduce what will and will not change. If we need an estimate of how long something will remain true, we can figure out how long it is likely to be before something that can change it occurs. To use McDermott's example, if you look at a boulder you might be able to estimate that if you came back in 50 years it would still be there (a weaker conclusion than implied by the notion of lifetimes, but it will do). However, if you are told that there is dynamite underneath, your estimate will be considerably different. In either case, if you came back the next day and discovered the boulder was some distance from its original location, you would have some theory about why, not just the feeling that your senses had lied to you. In addition, McDermott's model of quantities uses average rate instead of derivatives, which means many of the dynamical conclusions described here (such as distinguishing oscillation from stutter) cannot be drawn.

12.5 Current Work

Since the original publication of Qualitative Process theory ([Forbus, 1981]), several projects have adopted or extended some of its ideas. We will examine them here.

Johan de Kleer and John Seely Brown have continued to develop their device-centered qualitative physics [de Kleer & Brown, 1984]. In particular, they have adopted the quantity space representation for numerical values to allow more precise descriptions of state and state transitions. Brian Williams has also developed a similar device-centered physics, intended for reasoning about VLSI circuits [Williams, 1984]. Williams focuses on the classical notion of continuity in an attempt to bring intuitive and formal mathematical models in line. While the organizing principle for these dynamical frameworks are different from QP theory, on the whole their notion of quantity is now the same. The major exception concerns the equality change law; both deny case 2 of the law, which requires that transitions back to equality in quantities that are infinesimally different occur in an instant. This means stutter cannot occur in their systems, depriving them of a simple means of detecting dynamical equilibriums.

Device-centered ontologies have also been used to simulate and explain the operation of turbojet engines [Rajagopalan, 1894] and diagnose failures in circuits [Pan, 1983]. Both systems use a mixture of qualitative knowledge to describe states and rough behavior, with quantitative information used to reduce the degree of ambiguity in the descriptions.

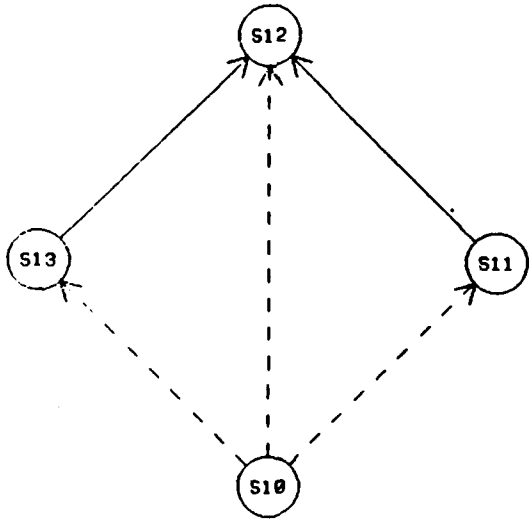
Reid Simmons has applied process descriptions to the problem of geological map interpretation [Simmons, 1983]. Given a diagram that represents a formation, his system ascertains whether or not a proposed sequence of occurrences of geological processes can give rise to it. Since in this domain it is assumed that only one process occurs at a time, his system represents occurrences of processes, that is, encapsulated histories, rather than processes. Changes in the existence of objects, such as an occurrence of a process creating or destroying an individual, are modelled by explicit statements in the description of the occurrence. This means all changes in existence must be forseen in advance by the model builder, which is reasonable for the geological domain. Importantly, these descriptions also provide equations that describe the net effects of each occurance. Given a set of numerical measurements from the diagram, these equations are woven together to check the consistency of the measurements against the hypothesized sequence of events. To make all this work, Simmons also developed a representation of intervals to serve as an extension theory for representing quantities.

Ben Kuipers has applied some of the ideas of QP theory in understanding causal reasoning in medicine ([Kuipers, 1982][Kuipers & Kassirer, 1983]). While adopting the ideas of the quantity space and qualitative proportionalities (he calls them the "value space" and "M" respectively), he does not explicitly represent processes or even objects. His "causal structural descriptions" are equations, as illustrated by the fact that his program subjects them to algebraic manipulation. This prevents his system from drawing conclusions about changes in the existence of individuals. By abandoning processes, he also loses an important source of constraint. Consider the scenario in figure 72, which is equivalent to the example in [Kuipers, 1982]. In this scenario, an object is placed in thermal contact with two temperature sources, with the object being initially hotter than one source and colder than another. In Kuiper's system this problem leads to "intractable branching" in the envisionment, forcing his system to re-write the equations involved and then perform a "perturbation analysis" on the new set of equations. GIZMO's answer, by contrast, is quite simple (figure 73 shows the situation plot, while figure 74 shows GIZMO's synopsis). The initial process structure (a

Fig. 72. Scenario for Kuiper's double heat flow example

```
(defScenario Double-Heat-Flow
   (Individuals STOVE
                ATMOSPHERE
                BLOB)
  (Facts (Piece-of-Stuff BLOB)
         (Temperature-Source STOVE)
         (Temperature-Source ATMOSPHERE)
         (Heat-path BURNER)
         (Heat-path BLOB-SURFACE))
  (Always (Heat-Aligned BURNER)
          (Heat-Aligned BLOB-SURFACE)
          (Heat-Connection BURNER STOVE BLOB)
          (Heat-Connection BLOB-SURFACE BLOB ATMOSPHERE)
          (Greater-Than (A (temperature STOVE))
                        (A (temperature ATMOSPHERE))))
  (In-Situation Start
                (Individuals STOVE BLOB ATMOSPHERE BURNER BLOB-SURFACE)
                (Facts (Greater-Than (A (temperature STOVE))
                                     (A (temperature BLOB)))
                       (Greater-Than (A (temperature BLOB))
                                     (A (temperature ATMOSPHERE))))))
```

Fig. 73. Double heat flow envisionment



heat flow from STOVE to BLOB and a heat flow from BLOB to ATMOSPHERE) never changes because doing so would violate D_s continuity. There is a three way ambiguity in the change in the blob's heat (and hence temperature) because the relative magnitudes of the flow rates are unknown. But if the blob's temperature is increasing or decreasing the flow rates will eventually become equal, causing the temperature to be constant, and if the temperature is ever constant it will remain so forevermore. The only way for "intractable branching" to arise

Fig. 74. Double heat flow synopsis

```
     In situation S10, there is a heat flow from BLOB to
ATMOSPHERE and a heat flow from STOVE to BLOB.  The
heat of ATMOSPHERE is increasing.  The heat of STOVE is
decreasing.  With some exceptions, the other quantities aren't
changing.
     The changes in the flow rate of the heat flow from
BLOB to ATMOSPHERE, the flow rate of the heat flow from
STOVE to BLOB, the heat of BLOB, and the temperature of BLOB
are not known at S10.  The situations S13, S12, and S11
represent the various possibilities.


     S11 represents a particular hypothesis about S10.
S11 assumes that the flow rate of the heat flow from
BLOB to ATMOSPHERE is decreasing, the flow rate of the heat
flow from STOVE to BLOB is increasing, the heat of
BLOB is decreasing, and the temperature of BLOB is decreasing.

     After some time, the flow rate of the heat flow from
BLOB to ATMOSPHERE and the flow rate of the heat flow from
STOVE to BLOB become equal.  This leads to situation S12.


     S12 represents a particular hypothesis about S10.
S12 assumes that the flow rate of the heat flow from
BLOB to ATMOSPHERE is constant, the flow rate of the heat
flow from STOVE to BLOB is constant, the heat of BLOB is constant,
and the temperature of BLOB is constant.
     No changes are possible.


     S13 represents a particular hypothesis about S10.
S13 assumes that the flow rate of the heat flow from
BLOB to ATMOSPHERE is increasing, the flow rate of the heat
flow from STOVE to BLOB is decreasing, the heat of
BLOB is increasing, and the temperature of BLOB is increasing.

     After some time, the flow rate of the heat flow from
STOVE to BLOB and the flow rate of the heat flow from
BLOB to ATMOSPHERE become equal.  This leads to situation S12.
```

---

is if irrelevant elements are added to the quantity spaces.[1] Without processes, Kuiper's program has no clear guide about what to put in a quantity space.

QP theory is being used in several psychologically-oriented projects as well. Al Stevens, Dan Weld, and Albert Boulanger are using QP theory in constructing a theory of explanations for machines [Weld, 1984]. Also, Allan Collins and Dedre Gentner are using QP theory to express theories of evaporation in order to understand how to shift from one level of description to another.

---

1. As it happens, Kuiper's program placed all quantities of the same type (e.g., all the temperatures) into the same quantity space (personal communication).

13. Bibliography

Allen, J. "Maintaining knowledge about temporal intervals" TR-86, Computer Science Department, University of Rochester, January 1981

Asbell, Irwin "A constraint representation and explanation facility for renal physiology", MIT SM Thesis, August 1982

Brown, J. S., Burton, R. R., and Zdybel, F "A model-driven question-answering system for mixed-initiative computer-assisted instruction", IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-3, No. 3, May 1973

Bunge, Mario "Causality and modern science" Dover Publications, Inc., New York, 1979

Davis, R. "Diagnostic reasoning based on structure and behavior", MIT AI Lab memo No. 739, June, 1984

Collins, A., Warnock, E. Aiello, N. and Miller, M. "Reasoning from incomplete knowledge", in Representation and Understanding, D. Bobrow and A. Collins, editors. New York, Academic Press, inc., 1975

de Kleer, J. "Qualitative and quantitative knowledge in classical mechanics" TR-352, MIT AI Lab, Cambridge, Massachusetts, 1975

de Kleer, J. "Causal and teleological reasoning in circuit recognition" TR-529, MIT AI Lab, Cambridge, Massachusetts, September 1979

de Kleer, J. "Contradictions without backtracking", to appear in AAAI-84

de Kleer, J. and Brown, J. "Assumptions and ambiguities in mechanistic mental models" in Mental Models, D. Gentner and A. Stevens, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983 Inc, 1983

de Kleer, J. and Brown, J. "A qualitative physics based on c nfluences" to appear in *Artificial Intelligence*, 1984

de Kleer, J. and Sussman, G. "Propagation of constraints applied to circuit synthesis" MIT AI Lab Memo No. 485, Cambridge, Massachusetts, 1978

diSessa, A. "Phenomenology and the evolution of intuition" in Mental Models, D. Gentner and A. Stevens, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983

Forbus, K. "A study of qualitative and geometric knowledge in reasoning about motion" TR-615, MIT AI Lab, Cambridge, Massachusetts, February, 1981

Forbus, K. and Stevens, A. "Using qualitative simulation to generate explanations" BBN Technical Report

No. 4490, March 1981; also in the Proceedings of the third annual conference of the Cognitive Science Society, August 1981

Forbus, K. "A CONLAN primer" BBN Technical Report No. 4491, prepared for Navy Personnel Research and Development Center, March 1981.

Forbus, K. "Qualitative reasoning about physical processes" Proceedings of IJCAI-7, 1981

Forbus, K. "Qualitative process theory" MIT AI Lab Memo No. 664, February, 1982, revised May 1983.

Forbus, K. "Qualitative reasoning about space and motion" in Mental Models, D. Gentner and A. Stevens, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983

Forbus, K. "Measurement interpretation in qualitative process theory" Proceedings of IJCAI-8, August 1983

Forbus, K. and Gentner, D. "Learning physical domains: towards a theoretical framework" in Machine Learning: Recen, Progress R. Michalski, J. Carbonell & T. Mitchell (Eds.), Tioga Press, 1984

Forrester, Jay W. "Principles of systems" MIT Press, Cambridge, Massachusetts, 1968

Fikes, Richard, and Nillson, N. "STRIPS: a new approach to the application of theorem proving to problem solving" *Artificial Intelligence*, Volume 2, No. 3, 1971

Hayes, Patrick J. "The naive physics manifesto" in Expert Systems in the Micro-Electronic Age, edited by D. Michie, Edinburgh University press, May 1979

Hayes, Patrick J. "Naive physics 1 - ontology for liquids" Memo, Centre pour les etudes Semantiques et Cognitives, Geneva, 1979

Heise, David R. "Causal analysis" John Wiley & Sons, New York, 1975

Hendrix, G. "Modeling simultaneous actions and continuous processes" Artificial Intelligence, Volume 4, 1973. pp145-180

Kuipers, B. "Getting the envisionment right", Proceedings of the National Conference on Artificial Intelligence, 1982

Kuipers, B. and Kassirer, J. "How to discover a knowledge representation for causal reasoning by studying an expert physician" Proceedings of IJCAI-8, 1983

Langley, P. "Rediscovering physics with BACON.3" Proceedings of IJCAI-6, 1979

McAllester, D. "An outlook on truth maintenance", MIT AI Lab memo No. 551, August, 1980

McAllester, D. "Reasoning utility package user's manual - version one", MIT AI Lab memo No. 667, April, 1982

McCarthy, J. "Circumscription -- a form of non-monotonic reasoning", Artificial intelligence, Volume 13, Number 1, April 1980

McCarthy, J. and Hayes, P. "Some philosophical problems from the standpoint of artificial intelligence", Machine Intelligence 4, Edinburgh University Press, 1969

McClosky, M. "Naive theories of motion" in Mental Models, D. Gentner and A. Stevens, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983

McDermott, D. "A temporal logic for reasoning about processes and plans" Cognitive Science, Vol. 6, No. 2, April-June, 1982

McDermott, D. and Doyle, J "Non-monotonic logic I", Artificial Intelligence, Volume 13, Number 1, April 1980

McDonald, D. "Natural language generation as a computational problem: an introduction" in Computational models of discourse, J. M. Brady and R. C. Berwick, editors, MIT Press, Cambridge, Massachusetts, 1983

Minsky, M. "A framework for representing knowledge" *MIT AI Lab Memo No. 306, June 1974*

Moore, R. "Reasoning from incomplete knowledge in a procedural deduction system" MIT AI Lab TR-347, Cambridge, Massachusetts, December 1975

Moore, R. "Reasoning about knowledge and action" MIT PhD thesis, February, 1979.

Pan, Yung-Choa "Qualitative reasoning with deep-level mechanism models for diagnoses of dependent failures" Coordinated Science Laboratory Report T-132, University of Illnois at Urbana-Champaign, December, 1983

Pew, R., Miller, D. and Feeher, C. "Evaluation of proposed control room improvements through analysis of critical operator decisions" Electric Power Research Institute Report NP-1982, August, 1982

Rajagopalan, R. M. "Qualitative modeling in the turbojet engine domain" Coordinated Science Laboratory Report T-139, University of Illnois at Urbana-Champaign, March, 1984

Reiter, R. "A logic for default reasoning" Artificial Intelligence, Vol. 13, 1980, pp 81-132.

Rieger, C., and Grinberg, M. "The declarative representation and procedural simulation of causality in physical mechanisms" Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 1977

Riley, M. Bee, N. and Mokwa, J. "Representations in early learning: the acquisition of problem solving strategies in basic electricity/electronics" University of Pittsburgh Learning Research and Development Center, 1981

Samuelson, P. "Economics", McGraw-Hill, N.Y., 1973

Simmons, R. "Representing and reasoning about change in geologic interpretation" TR-749, MIT AI Lab, Cambridge, Massachusetts, December 1983

Stallman, R. and Sussman, G. "Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis" Artificial Intelligence, Volume 9, pp 135-196, 1977

Stanfill, C. "The decomposition of a large domain: reasoning about machines" Proceedings of the National Conference on Artificial Intelligence, August 1983.

Stansfield, J. "Conclusions from the commodity expert project" MIT AI Lab Memo 601, November, 1980

Stevens, A. et. al. "STEAMER: advanced computer aided instruction in propulsion engineering" BBN Technical Report No. 4702, July 1981

Weld, D. "Explaining complex engineered devices" BBN Technical Report No. 5489, January 1984

Williams, Brian "Qualitative analysis of MOS circuits", MIT M.S. Thesis, January, 1984

Williams, M., Hollan, J. and Stevens, A. "Human reasoning about a simple physical system", in Mental Models, D. Gentner and A. Stevens, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983

14. Appendix 1 - DEBACLE

This appendix describes DEBACLE, the inference engine used to implement GIZMO. DEBACLE is a descendent of McAllester's Reasoning Utility Package, called RUP [McAllester, 1982]. There are several design decisions in RUP which proved inappropriate in implementing QP theory, and several extensions were needed. First we examine the problems with RUP. Then we describe the organization of DEBACLE, including the "hooks" for adding specialized representations. DEBACLE's closed-world assumption mechanism and stack-discipline premise controller are described next. Finally, the quantity representation used in GIZMO is presented as an example of how specialized representations may be implemented.

14.1 Why not RUP?

14.1.1 What is RUP?

McAllester's RUP has several novel features which make it useful in implementing reasoning programs. His overall model of an inference engine (see [McAllester, 1980]) contains three parts. First, a rule-like, pattern-directed component instantiates quantified knowledge. Second, a truth-maintenance system (which performs unit clause resolution) provides propositional reasoning and records dependencies. Third, a premise controller decides what to do when a contradiction occurs. Despite misgivings about the details of RUP, McAllester's overall model has been very productive.

To instantiate quantified knowledge, AMORD-like rules are provided. Unlike AMORD, however, the rules are not used to implement rules of logical inference (such as modus ponens). Instead, the rules assert propositional statements whose consequences are developed by the TMS. This frees the user from writing a tedious collection of inference rules for propositional logic, especially those rules which require careful control to avoid runaway generation of assertions.[1] McAllester thus advocates the strategy of seperating the creation of a description from reasoning with it, a strategy which has also proven useful in constraint networks.

McAllester's truth-maintenance system is organized so that propositional reasoning is performed via constraint propagation. Asserting an implication, for instance, causes a rule to run that inserts disjunctive-normal clauses in the TMS which provide all the deductive import of that implication. Similar rules are provided for the other logical connectives. Since, unlike full resolution, the TMS never introduces new terms, the deductions the TMS makes are incomplete but rapid, indeed roughly linear.

RUP provides two default handlers and hooks for user programs that decide what to do when a contradiction occurs. One default handler simply asks the user which premise should be retracted. The other default handler uses optional numerical certainties that can be attached to TMS nodes in order to retract the "least likely" fact. Once a node is chosen for retraction, the TMS automatically creates clauses that serve as

---

1. The now-classical example is the rule for AND introduction: given A and B, it is always the case that (AND A B) is true. But the simplest rendering of this inference rule will lead to generating an unbounded number of assertions, since an AND statement itself can be an argument to an AND statement. Usually problems like this one are avoided by relying on consequent reasoning.

nogoods, to prevent that particular conjunction of premises from ever being believed again.

In addition, RUP provides an equality sub-system that allows certain conclusions to follow from substitutions of equals for equals. Given an arbitrary term, the "simplest name" of that term can be computed on demand. The intention is that a user-provided function to determine the notion of simplicity will allow the equality system to provide much of the inferential power the user needs.

14.1.2 The equality system is inefficient

While the equality system seems to be adequate for small examples, paging problems cause it to perform badly with a medium-sized database (several hundred assertions). Suppose one uses equality to assign values to sign references, e.g.,

```
(= (s (d (level (at (c-s water f) s0)))) -1)
```

Using an early RUP-based implementation, determining the simplest name under equality for the change in the water level in `f` (i.e., `-1`), can take up to 5 minutes. Such performance is unacceptable, even for experimental purposes.

If the equality system were an independent component then one could simply not use it. However, the equality system strongly constrains the design of RUP's database. To make the equality system work, each subexpression of every term must be a database item in its own right. The reason is that evaluating an expression involves looking at each component to see if there is a better substitution - for `f`, `water`, `c-s`, then `(c-s water f)`, etc. in the example above. This leads the retrieval all over the database and makes the database needlessly large, assuming that we do not always need to refer to every term seperately.

14.1.3 The premise controllers provided are too crude

The premise controllers provided by RUP are too simple for complicated reasoning tasks. Merely asking the user precludes writing programs that deliberately introduce assumptions to use proof by contradiction. Using the numerical premise controller assumes that one can assign numerical certainty values to premises which may be interpreted globally, i.e., any two premises can be ordered with respect to relative certainty when handling a contradiction. These shortcomings can be overcome, of course -- McAllester wisely provided hooks for users to write their own contradiction handlers. A more serious limitation is the automatic construction of nogood clauses. Constructing a nogood clause implicitly assumes the underlying reason is monotonic, that is, no additional assumptions could ever lead to the premises involved in a contradiction being consistent. While McAllester eschews non-monotonicity, there are many cases (such as the set mechanism described below) where it is quite useful.

14.1.4 Adding specialized representations is hard

RUP proved to be a valuable tool for prototyping. Simple experimental systems can be quickly built using a combination of pattern-directed rules and LISP code to test out ideas. However, to increase overall efficiency it becomes necessary to add specialized representations to speed up critical deductions. This is

more difficult in RUP, due to its many assumptions about the form of the term database. It can be done (Reid Simmons, personal communication), but by abandoning the term database entirely greater efficiency can be gained.

14.2 DEBACLE organization

DEBACLE attempts to retain the good features of RUP while avoiding its shortcomings. Here the differences between DEBACLE and RUP are briefly described, rather than providing a manual or primer.

14.2.1 Assertions

The function `referent` maps from s-expressions to assertions in the database. `referent` does not add the assertion to the database if it isn't there, like RUP's `term-soft`. A default second argument is provided which, if non-nil, causes an appropriate assertion to be created. It is important to remember that merely being in the database does not mean that the assertion is believed to be true; each assertion has an explicit TMS node whose state (`TRUE`, `FALSE`, or `UNKNOWN`) represents the system's belief in it.

Two differences from RUP are worth mentioning. As in RUP, a term is associated with a *class*, which is used as an index for retrieval. The class of `(-> p q)`, for example, is `->`. Assertions are indexed by their CAR if their form is a list, and themselves otherwise. However, several s-expressions can refer to the same assertion. This allows an assertion to be indexed several ways to allow more efficient retrieval. For example, a qualitative proportionality is indexed by these three expressions:

```
(qprop A B)
(A constrained-by B)
(B constrains A)
```

The second form allows the system to efficiently retrieve all constrainers of `A`.

The other important difference is that creation and referencing assertions can be data-directed. Specialized representations can be constructed by creating new data structures which include the same properties as a standard assertion but have additional properties as well. We will see how this technique can be used to implement an efficient quantity representation below. These specialized data structures are created and referenced by functions associated with the class data structure.

14.2.2 Rules

RUP's rules are an improvement over AMORD rules in that there are more conditions under which they can fire. In AMORD, a rule fired only when the pattern it matched was true. In RUP a rule can fire when the pattern is true or false, when the fact is merely placed in the database, independently of whether or not it or its negation is believed (the `:INTERN` condition), when the fact changes belief state, or fire every time it becomes true, false, or changes. These distinctions allow rules to be used in constructing TMS structures for propositional reasoning and for signalling and updating external representations to track changes in the database. These distinctions are kept in DEBACLE, and have proved quite useful. Making domain

knowledge trigger on interning, for example, makes reasoning by contradiction relatively simple and can be used to speed up certain algorithms immensely (see situation elaboration in section 9.1).

Rules in DEBACLE differ in two ways from those in RUP. First, the trigger variables are bound to the s-expression they matched rather than a database item whose referent is that s-expression. This choice was made both because in using RUP one almost always converts to s-expressions anyway and because in DEBACLE there isn't always a database item to refer to for every s-expression. Second, an assertion can have an optional symbolic *assumption type*. An assumption type names the source of the premise, such as `Existence-Law` in the domain models we saw previously. This information is used by the premise controller when analyzing a contradiction, so we will defer further discussion of them until section 14.4.

14.3 Closed-world assumptions

Making closed-world assumptions appears to be an integral part of common sense reasoning [McCarthy, 1980]. Examples of closed-world assumptions used in QP theory include assuming that the objects one knows about are the only relevant ones, and that one knows all the influences on a quantity. A particular form of closed-world assumption that is useful in reasoning about naive physics is the closed-world assumption on set membership. In particular, given some explicit list of members in a set, one assumes that the set is composed solely of these members.

While this kind of closed-world assumption could be implemented by the now-classic "negation by failure" rule, in DEBACLE it is implemented more directly. A set is always considered to be the property of some object, and its structure is defined by an *element relation* and a *members relation*. The element relation describes the form that assertions about set membership take. For example, if `HAS-ACTIVE-PROCESS` is an element relation and `PROCESS-STRUCTURE` is a members relation, then if

```
(S0 HAS-ACTIVE-PROCESS PI-0)
```

is the only such assertion about `S0`, the assertion

```
(S0 PROCESS-STRUCTURE (PI-0))
```

would be believed. This assertion is justified on the basis of the positive and negative instances of the element relationships (in this case, the positive statement that `PI-0` is in the set is the only one we know about) and an explicit node representing the closed-world assumption. Whenever such an assumption is made, rules are created to look for new facts which would make the assumption false.

Importantly, these assumptions are made consequently, i.e., only when explicitly requested by the user or a program. The reason is subtle -- suppose a program is making assumptions by selecting an element from each of a number of sets, such as occurs in performing a dependency-directed search. If closed-world assumptions are made whenever possible, there can be a new set constructed whenever a choice is made. These sets can result in a contradiction that wouldn't occur if choices from the other sets had been made. The resulting inappropriate backtracking can lead to entire subspaces being skipped in the course of a dependency-directed search.

14.4 Premise control

What should be done when a contradiction occurs? RUP allows the user to bind a variable that the TMS uses to select a premise to retract from the set of assumptions underlying the contradiction. But this mechanism is not enough, since what should be done depends both on what premises underlie the contradiction and on what computation is underway at the time. For example, if the contradiction involves an assumption being made as part of a dependency-directed search then a new alternative should be selected. This suggests re-binding the contradiction handler when performing such a search. However, even if such a search is underway, a contradiction may arise which doesn't depend on any of the search assumptions. So merely re-binding the contradiction handler is insufficient. If we assume that control decisions are made by programs rather than rules, then a stack-discipline for contradiction handlers seems appropriate.

Here's how DEBACLE's premise controller works. First, the premises are tested to see if any of them involve closed-world assumptions, as outlined above. If so, it tests to see if any of them are invalid. If any are, the sets involved are updated and control is returned to the TMS. Otherwise, the controller goes down the stack of premise handlers, running each to see if it is relevant. A handler can either return NIL to indicate that it is irrelevant to the current contradiction, a premise to indicate what should be retracted, or T to indicate that the contradiction has been handled internally. A default handler which asks the user resides at the bottom of the stack.[1]

Programs which make assumptions must do two things. First, if any special action should be taken when a contradiction occurs the appropriate handler must be placed on the premise controller's stack. A *macro* is provided to do this:

```
(With-Contradiction-Handler
        <contradiction handler>
        <body>)
```

Aside from placing the handler on the stack, this form also ensures that the handler will be removed in case of non-local exits. Second, if the program is drawing conclusions that rely critically on particular assumptions, then it must test whether or not these assumptions still hold after performing some operation that might change them. For example, when influences are resolved the results are justified by the current process and view structures. For efficiency, the assumption about the current process structure is fetched at the beginning of the computation. If a contradiction occurs during processing this assumption might be retracted. Carrying on without noticing this change can result in patently incorrect assertions being added to the database. Thus the program checks these assumptions each time something has happened which could make this assumption invalid.

---

1. In addition, a display-oriented contradiction inspector is provided that allows the premises to be sorted by assumption type, dumped for future inspection, and their truth values manipulated to resolve the contradiction. Since contradictions with over 150 premises drawn from over twenty assumption types have arisen in the course of debugging, the contradiction inspector is a necessity rather than a frill.

14.5 Dependency-directed search

Dependency-directed search is an important problem-solving technique [Sussman & Stallman, 1976]. In DEBACLE, dependency-directed searches are performed by constructing a generator which will construct sets of consistent choices from a collection of alternatives. Each set of alternatives is represented by a *search frame*. The alternatives are assumed to be mutually exclusive and exhaustive, such as the status of a process instance or the D_s values of a quantity.

To get a generator, the macro `With-DD-Generator` is used. Its syntax is:

`(With-DD-Generator` ⟨*name*⟩ . ⟨*body*⟩`)`

where ⟨*name*⟩ will be bound to a new generator that will exist during the execution of ⟨*body*⟩. Generators are implemented as flavors, so messages are provided for adding search frames, initializing the generator, and making choices. A contradiction handler is automatically added to the premise controller's stack to catch contradictions due to inconsistent collections of choices. When making a set of choices, the generator will respect the constraints in the TMS. In particular, if one element of the set of alternatives is already known to be true then only it is selected, and if some element is false then that choice is skipped.

14.6 Quantity representation

GIZMO uses a specialized representation to draw conclusions about quantities efficiently. Quantities, signs, magnitudes, and inequality statements are all instantiated as database items with extra properties. Items that represent quantities contain pointers to items that represent their amounts and derivatives. Items that represent numbers have pointers to items that represent their signs and magnitudes and an index that describes what comparisons with other numbers have been made. Items representing signs include three TMS nodes which represent the possible values for the sign (`-1`, `0`, `1`). Items representing magnitudes have TMS nodes representing the possibility of the magnitude being greater than or equal to `ZERO`, as well as an index of ordering statements. These items are constructed whenever a `Has-Quantity` assertion is made, when an attempt is made to reference some part of quantity, or an inequality statement is made.

When an assertion corresponding to an inequality statement is made (`Greater-Than`, `Less-Than`, or `Equal-To`), a special object is created to represent the comparison. This special object has three TMS nodes which represent the belief in the two numbers being less than, greater than, or equal to the other. All other inequality statements concerning the two numbers are constructed so that these TMS nodes are associated with the appropriate statements.

Most of the inferences concerning quantities are drawn by TMS clauses that are created when the entity involved is created. Sections 7.2.1 and 7.2.2 describe the particular facts used.

Finding the ordering between two numbers (or magnitudes) is done consequently. The asserted inequality relationships form a lattice which is searched to answer ordering questions. The procedure is complete in the sense that if the ordering can be deduced by transitivity on the set of inequalities, it will be deduced. The search results are fully justified by recording the dependence of the answer on the particular nodes used.

One additional feature bears remarking. The enemy of lattice models is connectivity; if too many

items are linked then the search can encompass all of the inequality assertions if the answer cannot be found. Placing ZERO in the lattice would have precisely this effect: walking "down" from a positive number will result in visiting every negative number! Clearly this should be avoided, and indeed it can -- signs are God's way of disconnecting a lattice. ZERO is never placed in the lattice, and sign information is used both to answer certain questions quickly (such as when asking for the ordering of two numbers of different signs) and to limit search by cutting it off when the sign changes.

END

FILMED

2-85

DTIC