UTILIZATION OF THE MVL SYSTEM IN QUALITATIVE REASONING ABOUT THE PHYSICAL WORLD

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Mine Ülkü Sencan
July, 1993

UTILIZATION OF THE MVL SYSTEM IN QUALITATIVE REASONING ABOUT THE PHYSICAL WORLD

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER

ENGINEERING AND INFORMATION SCIENCE

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Mine Ülkü Şencan

July, 1993

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Varol Akman (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. David Davenport

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. H. Altay Güvenir

Approved for the Institute of Engineering and Science:

Prof. Mehmet Baray
Director of the Institute

ABSTRACT

UTILIZATION OF THE MVL SYSTEM IN QUALITATIVE REASONING ABOUT THE PHYSICAL WORLD

Mine Ülkü Şencan
M.S. in Computer Engineering and Information Science
Advisor: Assoc. Prof. Varol Akman
July, 1993

An experimental program, QRM, has been implemented using the inference mechanism of the *Multivalued Logics* (MVL) Theorem Proving System of Matthew Ginsberg. QRM has suitable facilities to reason about dynamical systems in qualitative terms. It uses Kenneth Forbus's *Qualitative Process Theory* (QPT) to describe a physical system and constructs the envisionment tree for a given initial situation. In this thesis, we concentrate on knowledge representation issues, and basic qualitative reasoning tasks based on QPT. We offer some insights about what MVL can provide for writing Qualitative Physics programs.

Keywords: Multivalued Logics (MVL), Qualitative Process Theory (QPT), Qualitative Physics, Envisioning, Commonsense Reasoning.

ÖZET

MVL DİZGESİNİN FİZİKSEL DÜNYA HAKKINDA NİTEL USLAMLAMADA KULLANIMI

Mine Ülkü Şencan
Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans
Danışman: Doç. Dr. Varol Akman
Temmuz 1993

Matthew Ginsberg'ün MVL teorem tanıtlama dizgesinin çıkarım mekanizması kullanılarak deneysel bir program, QRM, gerçekleştirilmiştir. QRM dinamik dizgeler hakkında nitel terimler kullanarak uslamlama yapabilir. Kenneth Forbus'un Nitel Süreç Kuramı'na (QPT) göre tanımlanmış fiziksel bir dizgenin verilen başlangıç durumundan itibaren ulaşabileceği diğer durumları betimleyen ağacı öngörebilir. Bu tezde, bilgi gösterimi ve QPT'ye dayalı temel nitel uslamlama konuları üzerinde durulmaktadır. MVL'in Nitel Fizik programları yazılımında neler sağlayabileceğine dair bazı önerilerde bulunulmaktadır.

Anahtar Sözcükler: Çok-değerli Mantıklar (MVL), Nitel Süreç Kuramı (QPT), Nitel Fizik, Öngörme, Sağduyusal Uslamlama.

ACKNOWLEDGMENTS

I would like to thank my advisor Assoc. Prof. Varol Akman for his guidance, invaluable encouragement, and motivating suggestions throughout the development of this thesis. (N.B. I am grateful to Prof. Matthew Ginsberg, Stanford University, for his valuable help with MVL.)

I owe thanks to Asst. Prof. David Davenport and Asst. Prof. H. Altay Güvenir for their valuable comments on the thesis.

I would also like to thank Prof. Mehmet Baray and the rest of the faculty of the Department of Computer Engineering and Information Science for the stimulating research environment they have created.

Finally, I am grateful to my family for their infinite moral support, particularly in times of despair.

Contents

1 INTRODUCTION 1

2 QUALITATIVE PROCESS THEORY 4

2.1 Objects and Quantities 5

2.2 Processes 7

2.3 Basic Deductions 9

2.4 Some Other Qualitative Reasoning Tasks 12

3 THE MVL SYSTEM 14

3.1 The MVL Database 15

3.1.1 Representation 15

3.1.2 Truth Values 16

3.2 Inference Mechanism 17

4 THE IMPLEMENTATION: QRM 21

4.1 Representation of Domain Models 22

4.1.1 Individuals and Quantities 22

4.1.2 Processes and Individual Views 27

4.2 Basic Deductions 30

4.2.1 Finding Possible Processes 30

4.2.2 Determining Activity 30

4.2.3 Determining Change 31

4.2.4 Limit Analysis 34

4.3 Envisioning 34

4.4 Basic Experiments 37

4.5 Limitations of QRM 44

5 CONCLUSION 47

A A View on Fluids Domain Model 49

A.1 Heat Flow 49

A.2 Boiling 51

B Scenarios for Examples 53

B.1 Liquid Flow (Two Containers) 53

B.2 Heat Flow and Boiling 54

B.3 Liquid Flow (Three Containers) 55

C How to Access QRM 58

List of Figures

1.1 Comparison of Classical and Qualitative Physics (adapted from [39]) 2

2.1 A physical system where a liquid flow process can be active . . . 5

2.2 Some quantities that are used in the representation of liquids . . 6

2.3 A partial ordering among quantities. (Here arrows are implicitly labeled with the relation "$\leq$".) 7

2.4 Description of `Contained-Liquid` 8

2.5 An example process description in QPT 10

4.1 The algorithm for elaboration 31

4.2 The algorithm for resolving influences 33

4.3 The algorithm for limit analysis 35

4.4 The algorithm for envisioning 36

4.5 Envisionment for the two-container system. (VIS stands for individuals imposed by the views, PS indicates process structures in the situation, and LH denotes the limit hypothesis.) 38

4.6 Heat flow and boiling. (`STOVE` is assumed to be a temperature source, `CAN` is closed, and no explosion is considered.) 39

4.7 Envisionment for heat flow and boiling system. (LH1, LH2, and LH3 correspond to limit hypotheses.) 40

4.8 The three-container system 41

4.9 A part of the envisionment for the first alternative situation. The second alternative situation is similar. (EE denotes an external effect such as the controller.) 43

4.10 A part of the envisionment for the third alternative situation . . 44

Chapter 1

INTRODUCTION

Qualitative Physics, an active area of research in Artificial Intelligence (AI), deals with commonsense reasoning about the physical world [22, 30, 29, 44, 13, 32, 4, 12]. The motivation primarily comes from contemporary engineering problem solving in which techniques for automating engineering practice are sought (cf. [10] for an interesting personal account and motivation) as well as from outstanding problems of cognitive science, computer-aided education, simulation, etc. [34, 20, 39].

Qualitative Physics is an alternative approach for describing physical phenomena around us. To this end, scientists and engineers normally use the formalism of conventional physics and mathematics. However, while they describe physical phenomena using Classical Physics, they tend to understand and explain physical behavior in terms of causes and effects [31, 11, 3]. Clearly, most of the time we must describe the behavior of a physical system using accurate quantitative values and numerical equations, viz. Classical Physics. However, it is doubtful that we gain much insight into the functioning of the system when we do so.

Qualitative Physics provides valuable intuitions by giving a commonsense description of behavior while enabling us to describe physical situations in a simpler (yet formal) way using symbolic qualitative values [6]. This qualitative representation requires quantity values to be chosen from a discrete quantity space rather than from a continuous one. In Qualitative Physics, the behavior

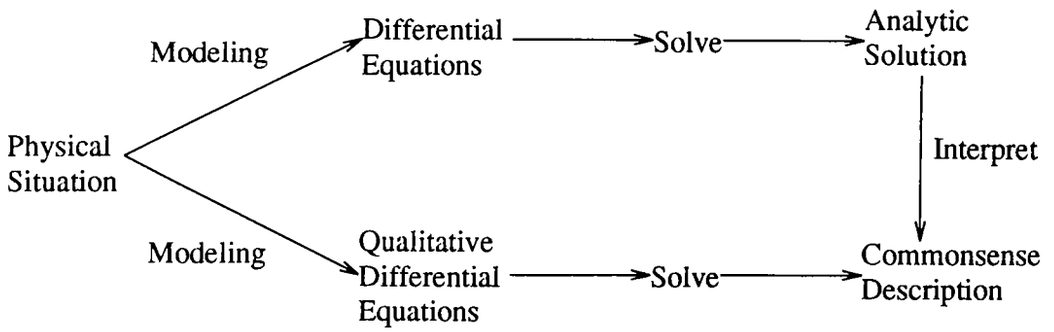


Figure 1.1. Comparison of Classical and Qualitative Physics (adapted from [39])

of a physical system is effectively characterized by the derivatives of its quantities. Hence, a quantity may increase, decrease, or stay unchanged when the sign of its first derivative takes the values 1, -1, or 0, respectively.

In Figure 1.1, Classical Physics and Qualitative Physics are compared. Initially, both of the physics attempt to formalize the physical situation, one using complicated numerical equations and the other using simple qualitative constraints. Then, both solve their related equations using their own methods. At the end, while Qualitative Physics arrives at a commonsense description of the solution, Classical Physics arrives at numerical values which may not say much intuitively.

A computer program for qualitative reasoning requires a qualitative model of a physical system as input. This model must be adequate for specifying what constitutes the physical system of concern and the observable relationships. Currently, there are various well-established formalisms that offer constraint-based, component-based, and process-based paradigms for the representation of physical systems [42]. The constraint-based approach uses a set of variables and assorted constraints on those variables [32]. In the component-based approach, the behavior of a physical system is obtained from the behaviors of its components [8]. The process-based approach uses the notion of active processes that potentially exist in a physical situation [14]. This is the approach we prefer to work with.

Among process-based approaches, Forbus's *Qualitative Process Theory* (QPT) provides the most comprehensive framework for modeling physical situations. Thus, it serves as a primary guide for many of the current Qualitative Physics programs—especially those dealing with dynamical systems [42, 17].

This thesis introduces a QPT-based experimental qualitative reasoning program, QRM (*Q*ualitative *R*easoner in *M*VL), that has been implemented using the formalism and the inference mechanism of the Multivalued Logics' (MVL) Theorem Proving System [24]. QRM gives some insights about what MVL can provide for writing qualitative reasoning programs. QRM is a more sophisticated version of a simple program, QREM, which we implemented some time ago [38, 2].

In this chapter, various general ideas have been pointed out in order to give a broad understanding of Qualitative Physics. A fine treatment which is much more detailed can be found in [22].

The remainder of this thesis is structured as follows.

In Chapter 2, an informal review of QPT is given. Readers interested in the details of QPT are referred to Forbus's seminal paper [14] and his Ph.D. thesis [15].

In Chapter 3, MVL is introduced. The terse but informative user's guide of the MVL system [26] may be helpful to learn more about the basic system functions and predicates. However, our experience with MVL taught us that learning by doing is a must.

Chapter 4 is a detailed account of QRM, our qualitative reasoning program. QRM uses Forbus's QPT for the description of physical situations within MVL. It performs basic reasoning tasks of QPT and constructs envisionments from some initial situation with the help of the inference mechanism of MVL.

Chapter 5 concludes the thesis with remarks on what has been achieved and what can be done to improve QRM further.

Chapter 2

QUALITATIVE PROCESS THEORY

QPT [15] is a formalism for describing physical situations. Forbus specifies his theory as one with which reasoning about dynamical systems can be made easily and effectively. He mentions this in [14]: "Qualitative Process Theory defines a simple notion of physical process that appears useful as a language in which to write dynamical theories."

According to QPT, a dynamical system changes its state as a result of active processes. In QPT, a *process* is understood as something that causes changes in objects over time [14]. Moving, colliding, flowing, and boiling are examples of processes acting on objects. A change may occur in a system only when there are inequalities between property values of objects. In Figure 2.1, a potentially existing process, `LIQUID-FLOW`, is represented in the framework of a dynamical system consisting of two containers `F` and `G`, and a fluid path `P` (which is supposed to hold no material in it). A flow from `F` to `G` imposes changes on the levels of liquid in containers `F` and `G`.

A *domain model* of a dynamical system consists of descriptions of existing objects in the system, relationships among those objects, and processes that can occur in some physical situation. A specific process becomes active when all of its preconditions hold. Active processes in each situation need not be given explicitly; they can be inferred using the process specifications. When a complete set of process descriptions is provided, QPT should be able to predict physical behavior by identifying active processes and their influences.

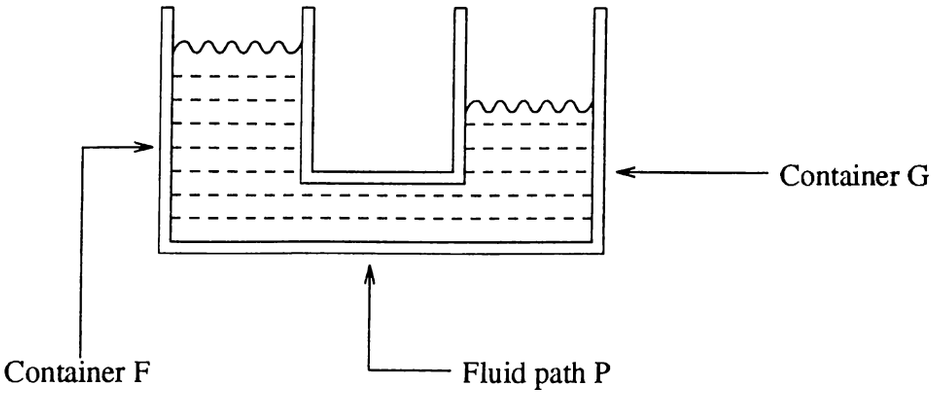


Figure 2.1. A physical system where a liquid flow process can be active

In general, a theory in Qualitative Physics is regarded as useful for qualitative reasoning if it has the following three properties [39, 14]:

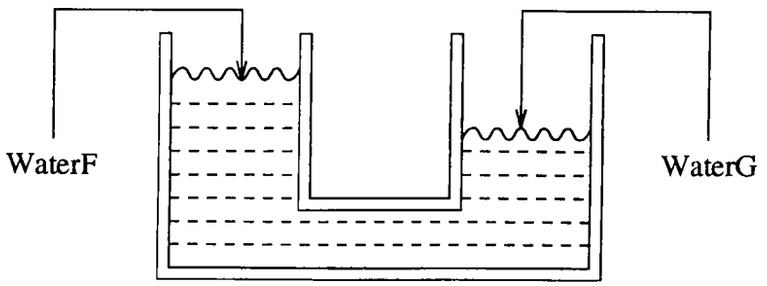
1. It allows specification of effects and the means by which effects are propagated.
2. It provides decomposable descriptions, i.e., a complicated situation can be described by its parts.
3. It allows graceful extensions to the basic theory.

These properties will be investigated for QPT in the upcoming sections.

2.1 Objects and Quantities

In QPT, objects are represented by *individuals*. The nature of an individual in general depends on the domain model used. However, process instances are always regarded as individuals.

Processes influence the objects in a situation and cause changes on their *quantities*. A quantity is a representation of an object property which takes values from a number space and changes as a result of active process instances (Figure 2.2).



amount-of	pressure
level	volume
bottom-height	top-height

A[level(WaterF)] > A[level(WaterG)]

Figure 2.2. Some quantities that are used in the representation of liquids

A quantity consists of *amount* and *derivative* that are numbers represented by their *sign* and *magnitude* [15]:

A_m—magnitude of the amount,
A_s—sign of the amount,
D_m—magnitude of the derivative,
D_s—sign of the derivative.

The value of a number is specified using its *quantity space*, a partial ordering among quantities. Two ordered quantity values in a quantity space with no other value between them are referred to as *neighbors*. The neighboring quantities in a quantity space facilitate the determination of process activities through time. Figure 2.3 illustrates an example quantity space for the system of Figure 2.2.

Individual views are individuals such as process instances. Their existence depends on the facts of a situation. Individual views are used to describe objects that can be put into existence and destroyed. Their properties can change [16], e.g., when water in a container starts boiling, it changes its state,

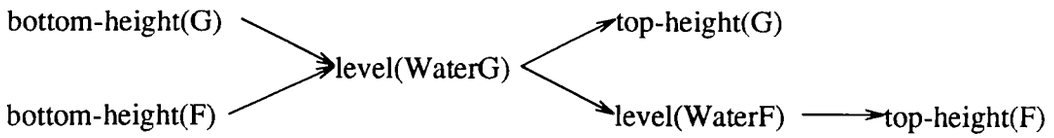


Figure 2.3. A partial ordering among quantities. (Here arrows are implicitly labeled with the relation "$\leq$".)

and if boiling continues long enough no liquid will be left in the container.

An individual view is designated by its *individuals*, *quantity conditions* and *preconditions* (inequalities between quantities and other conditions that are necessary for the view to hold), and *relations* (facts that are inserted into the situation when the view holds). A distinction between preconditions and quantity conditions needs to be made here. Quantity conditions are for predicting the changes whereas preconditions are for determining whether the theory is applicable to the situation at hand.

Figure 2.4 gives the description of an individual view `Contained-Liquid`. (Notation is somewhat mixed. Forbus explains the details of this in [14, 15].)

2.2 Processes

The main surmise of QPT is the so-called *sole mechanism assumption* which allows us to know all the potential changes in a physical situation [14]. It can be stated as follows: All changes in a physical system are caused directly or indirectly by processes.

QPT views a physical process as something which acts through time to change the parameters of objects in a situation. A process is formally described in QPT using the following:

- *Individuals*: Objects that the process acts on.
- *Preconditions*: Conditions that are imposed by the external world.
- *Quantity Conditions*: Conditions that relate quantity values.

```
Individual View Contained-Liquid

Individuals:
   c is a container
   s is a liquid

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

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

Relations:
   There is p which is a piece-of-stuff
   amount-of(p) = amount-of-in(c,s)
   made-of(p) = s
   container(p) = c
```

Figure 2.4. Description of Contained-Liquid

- *Relations*: Relationships a process imposes on the individuals it acts upon (i.e., what holds when the process is active).
- *Influences*: Direct effects of a process. (Each process has at least one direct influence.)

A process is specified just like an individual view, although it has one additional part, viz., influences (Figure 2.5). When there are objects in a situation that can be matched with the individual declaration of a process, a process instance (PI) is created. A PI is said to be active when its preconditions and quantity conditions hold. Preconditions are conditions outside of QPT, i.e., an open faucet remains open as long as nobody closes it. Quantity conditions are related to and can be predicted by the theory, i.e., an open faucet may be closed, if a physical process has indeed such an effect.

Influences become effective when process instances are active. *Direct* and *indirect* influences determine the cause of change in a quantity. If a quantity `Q` is directly influenced by a number `n`, this is written `I+(Q,n)` (resp. `I-(Q,n)`) when the influence is positive (resp. negative).

A quantity is said to be indirectly influenced if it is a function of some other changing quantity. The representation `Q1` α_{Q+} `Q2` (or its syntactically sugared version, `Q1 qprop+ Q2`) indicates that `Q1` is a function of some quantity `Q2` which is monotonically increasing whereas `Q1` α_{Q-} `Q2` (or `Q1 qprop- Q2`) says that `Q1` is a monotonically decreasing function of `Q2`.

According to QPT a quantity cannot be both directly and indirectly influenced at the same time. When this happens the domain model is considered to be inconsistent [14].

2.3 Basic Deductions

Making deductions is the ultimate goal of any representation for Qualitative Physics. QPT supports several deductions, including the basic ones: finding possible processes, determining activity, determining change, and limit analysis.

```
Process Liquid-Flow

Individuals:
   source is a contained-liquid
   destination is a contained-liquid
   path is a fluid-path
   fluid-connection(path,source,destination)

Preconditions:
   aligned(path)

Quantity-Conditions:
   A[pressure(source)] > A[pressure(destination)]
   status (liquid-flow-support,Active)

Relations:
   Let flow-rate be a quantity
   A[flow-rate] > ZERO
   flow-rate qprop+ (pressure(source) - pressure(destination))

Influences:
   I-(amount-of(source),A[flow-rate])
   I+(amount-of(destination),A[flow-rate])
```

Figure 2.5. An example process description in QPT

Finding Possible Processes Processes whose "individuals" specification is satisfied are potential processes, i.e., PIs, that can occur in a situation. Similarly, view instances (VIs) can be determined by having views whose "individuals" specifications are satisfied in the situation.

Determining Activity A process instance is *Active*, if its preconditions and quantity conditions hold. Changes in individuals are represented by *process* and *view structures*. Process structures are constructed by taking PIs that are `Active`. (View structures can be found similarly.)

Determining Change Changes in quantities of an individual are indicated by D_s. A quantity is said to be decreasing, increasing, or stable, when its D_s takes the values -1, 1, and 0, respectively.

In order to determine D_s, direct and indirect influences incurred by process and view structures need to be resolved. Direct influences are resolved by summing up the influences. If all influences have the same sign value then D_s will be that sign. In those cases where influences have conflicting sign values, it is still possible to arrive at a solution by using the inequality information among quantities. However, there is no guarantee that direct influences can always be resolved, because information on quantities, inequalities, and derivatives may not be adequate to determine the result all the time. (Obviously, this is a natural weakness of Qualitative Physics vis-à-vis Classical Physics.)

Resolving indirect influences is the most difficult task and requires qualitative proportionalities. Qualitative proportionalities specify functional relationships between quantities, so they do not give enough information on the strength of influences. While direct influences can be summed up easily, indirect influences cannot be. In fact, most of the time it will not be possible to resolve indirect influences in basic QPT [14].

Limit Analysis Changes in quantities may alter some process and view structures in a situation. *Limit analysis* is used to determine those changes and changes in the D_s values of quantities.

The set of changes in single inequalities and combinations of those changes

that are consistent in the situation constitute the *quantity hypotheses* for a situation. A *limit hypothesis* is a quantity hypothesis which not only causes a change in a D_s value, but also imposes a change in some view or process structure.

Some changes in quantity spaces occur instantaneously whereas some others span an interval of time as the *equality change law* below indicates [15]:

> "With two exceptions, a process structure lasts over 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 equality change law sometimes helps one obtain unique results from limit analysis within basic QPT [14].

2.4 Some Other Qualitative Reasoning Tasks

Prediction Prediction involves determining the future events of a situation. Since the information about a situation is usually incomplete, the exact events cannot be predicted most of the time. Instead, all possible events can be generated using some assumptions about the situation. *Envisioning* generates the alternative next situations and shows the future events of those situations as a directed graph [39, 36, 7].

Postdiction Postdiction is used to infer how a particular qualitative state has been reached. It is much more difficult than prediction because the assumptions that can be made about the situation are numerous [14].

Measurement Interpretation Measurement interpretation is used to infer what is happening in a situation given some observations (measurements) about the behavior of individuals [19]. The importance of this is emphasized in [44]:

> "The problem of interpreting observations of a system over time is fundamental to intelligent reasoning about the physical world. We view interpretation as the task of determining which possible behaviors predicted by the current model are consistent with the sensory data, including which are most plausible."

Chapter 3

THE MVL SYSTEM

MVL (written in Common Lisp) is an implementation of theoretical work done at Stanford University [24]. The core of the system relies on the *multivalued logics* paradigm of Ginsberg [23].

Broadly speaking, MVL is like any other logic programming language, if we view it as consisting of a database and an inference mechanism that tells whether or not a query follows from the information in the database [26]. However, MVL does not just serve as a logic programming language. It also provides facilities for making inferences using various techniques—default logic, circumscription, temporal logic are some of these—and allows one to define new logics.

The most important feature of MVL is its use of multiple truth values for logical statements. Unlike Prolog, MVL does not simply label a statement to be true; it considers "true by default," "true by some assumption," "false by default," etc., as reasonable values and uses them when answering a query [26].

In addition to above, MVL includes facilities for dealing with *modal operators* [24]. (N.B. MVL is still being developed with occasional changes and additions to the system. Bugs are being fixed whenever possible [25].)

3.1 The MVL Database

3.1.1 Representation

An MVL database consists of sentences that are represented as Lisp *s*-expressions and labeled with truth values. The logical connectives `not`, `or`, and `and` are used in prenex normal form. An example statement "Tweety is a penguin and it cannot fly" would be represented as:

```
(and (penguin Tweety) (not (flies Tweety))).
```

MVL provides two basic connectives for inference tasks: => for *forward chaining* and <= for *backward chaining*. The form of backward- and forward-chaining rules are as follows:

$$(<= \mathit{Conclusion}\ \mathit{Premise}_1\ \mathit{Premise}_2\ \ldots\ \mathit{Premise}_n)$$
$$(=> \mathit{Premise}_1\ \mathit{Premise}_2\ \ldots\ \mathit{Premise}_n\ \mathit{Conclusion})$$

In addition to these basic connectives `IF`, `IFF`, and <=> can also be used to represent various kinds of statements. An `IF` statement produces a pair of rules depending on the value of the system variable `*if-translation*`. When `*if-translation*` is set to `bc` (the default value),

```
(IF (penguin Tweety) (bird Tweety))
```

is translated into

```
(and (<= (bird Tweety) (penguin Tweety))
     (<= (not (penguin Tweety)) (not (bird Tweety)))).
```

When the `*if-translation*` is set to `fc` the equivalent statement will be the conjunction of the rule itself and its contraposition :

```
(and (=> (penguin Tweety) (bird Tweety))
     (=> (not (bird Tweety)) (not (penguin Tweety)))).
```

If the value is `mix`, a pair of contraposed rules are produced:

```
(and (<= (bird Tweety) (penguin Tweety))
     (=> (penguin Tweety)) (bird Tweety))))
```

The connectives IFF and <=> are used to represent equivalence. A database rule

```
(IFF (bird Tweety) (flies Tweety))
```

is equivalent to:

```
(and (IF (bird Tweety) (flies Tweety))
     (IF (flies Tweety)) (bird Tweety))))
```

Equivalence translations of <=> change according to the value of the global system variable `*equivalence-translation*`. Just like `*if-translation*`, `*equivalence-translation*` takes the values `bc`, `fc`, and `mix`. Corresponding translations of the previous equivalence for <=> produce the backward- and forward-chaining forms shown below:

```
(and (<= (bird Tweety) (flies Tweety))
     (<= (flies Tweety)) (bird Tweety))),

(and (=> (bird Tweety) (flies Tweety))
     (=> (flies Tweety)) (bird Tweety))),

(and (=> (bird Tweety) (flies Tweety))
     (<= (bird Tweety)) (flies Tweety))).
```

In MVL, LISP atoms beginning with a ? are regarded as variables. Unbound variables in database statements are assumed to be universally quantified. Existential quantification is handled using Skolem constants.

3.1.2 Truth Values

The truth value of a database sentence changes depending on the logic that MVL considers as the basis of inference. To give an example, if we are using

default logic, then the basic truth values will be "true," "false," "default true," and "default false." On the other hand, if we are using first-order logic, then the basic truth values will simply be "true" and "false." When we assert `(not (flies Tweety))` in default logic, negation of the statement, i.e., the truth value of `(flies Tweety)`, is assigned the value `df` (false by default). If we assert the same sentence in first-order logic the truth value assigned will simply be `false`.

The truth values that underlie the power of MVL are represented within a mathematical structure called a *bilattice*. The statements in the database are labeled using the elements of the bilattice given for the specified logic. Currently, some of the predefined bilattices in MVL are *first-order*, *ATMS*, *default*, *circumscription*, and *time* bilattices.

Truth values of the MVL bilattices that we use in QRM are:

`*first-order-bilattice*` The first-order logic has four truth values, namely, `true`, `false`, `unknown`, and `bottom` (both `true` and `false`).

`*default-bilattice*` The default bilattice contains seven truth values. Three extra truth values are `dt` (true by default), `df` (false by default), and $*$ (both true and false by default) [26]. Default values are valid until information is supplied to the contrary. `True` and `false` subsume the default values since they give more certain information about an MVL sentence than `dt` and `df` [23].

Besides the predefined bilattices, MVL provides facilities to incorporate new logics into the system. This is why we consider MVL as a valuable inference tool for Qualitative Physics. Depending on the nature of the physical system and the reasoning task, an appropriate logic with relevant operators can be described within MVL.

3.2 Inference Mechanism

The underlying inference mechanism of MVL is a first-order theorem prover. This "simple natural-deduction style complete" [24] theorem prover differs from a conventional one by having features to handle queries with negative subgoals

containing free variables.

A multivalued query may produce several proof tasks to be accomplished by the first-order prover. Each task requires an invocation of the prover and initiates a proof search. The inference, in general, proceeds according to the order of premises in the rule, and the proof search continues by choosing the most promising proof attempt. When all the premises are satisfied, the system returns the conclusion as the answer to the query.

Querying the Database In MVL, a query may initiate a proof search with or without free variables. If there are free variables in the query, then the returned answer will contain the bindings for those variables as well as the truth value for the answer. An example query,

```
(flies ?x)                    ;; who flies?
```

may return with a binding `((?x . Tweety))` and a truth value `dt`, meaning

```
(flies Tweety)                ;; Tweety flies
```

is true by default.

With the above representation, MVL provides various functions for searching and querying the database. Functions such as `lookup`, `lookups`, `contents`, and `prfacts` [26] are used to this end.

Backward Chaining Backward chaining is used to arrive at a conclusion by first proving the consequences of the rules in the database.

The functions `bc` and `bcs` are used to initiate inference for a given query:

`bc(query)` attempts to prove `query` from information in the database, looking only for a single answer.

`bcs(query)` attempts to prove `query` from information in the database, looking for all possible answers.

Those functions also accept some cutoff information in order to control the nature of the proof search, e.g., using the `succeeds` keyword we can terminate the search as soon as the final answer passes a given test.

Forward Chaining In MVL, forward chaining is somewhat complicated. Unlike some of the other logic programming systems, adding a new sentence into the database may cause a previous conclusion to be retracted, due to the nonmonotonic nature of multivalued inference [26].

The functions `fc` and `erase` are used to manipulate the database in case of forward inferencing:

`fc(proposition)` inserts `proposition` into the database and forward chains.

`erase(proposition)` removes `proposition` from the database and forward chains.

Procedural Attachment MVL allows the use of ordinary Lisp functions for database manipulation and inference. The macros `lisp-defined` and `lisp-predicate` are employed to make Lisp functions known to the system.

Modal Operators The MVL system can deal with various sorts of modal operators. Modal operators are defined as functions on the bilattice of truth values. Bilattices supplied with some modal operators are as follows [26]:

`*first-order-bilattice*` Modal operator `L`, which maps `proposition` into "necessarily `proposition`" and modal operator `M`, which maps the `proposition` into "possibly `proposition`" are defined in the first-order bilattice.

`*default-bilattice*` This bilattice inherits the modal operators of the first-order bilattice.

The modal operator `L`, in particular, provides retraction of the facts in the database that became unsupported during an `erase`. The following forward chaining rule inserts `(B)` into the database when an `(fc '(A))` is encountered and removes it when an `(erase '(A))` is chained upon the rule:

```
(=> (L (A)) (B))          ;; if (A) is known then (B)
```

The following rule, however, does not remove `(B)` although `(A)` is `erased` from the database:

```
(=> (A) (B))              ;; if (A) then (B)
```

QRM makes use of the modal operator `L` with the forward chaining rules for inserting and removing the deduced facts and the hypotheses about a physical situation.

`*time-bilattice*` provides both `L` and `M` as well as `delay`, `propagate`, and `at` modal operators which are specific to temporal reasoning [26].

Chapter 4

THE IMPLEMENTATION: QRM

Theoretical studies in qualitative physics frequently offer new ideas for the description of physical systems. However, these ideas need to be tested. As de Kleer, one of the pioneers of the field, mentions in [42]: "Significant AI progress can be made only by applying AI ideas to tasks. Otherwise, we tend to spin our wheels."

Forbus's GIZMO formed a test environment for QPT's ideas [15]. Although not all parts of QPT was implemented in GIZMO, this program provided highly valuable information about the applicability of QPT.

QRM is a program (written in Lucid Common Lisp) that experiments with QPT by making use of the MVL system. The reasoning tasks can be accomplished using MVL's default logic (as well as first order logic) in the case of incomplete information. The multivalued framework of MVL enables one to incorporate other logics, e.g., hierarchical default logic and especially temporal logic, in order to perform some demanding reasoning tasks about the physical world.

Currently, we are working on qualitative reasoning in the fluids domain. In the modeling of simple fluid dynamics an interesting representational problem arises. Liquids, unlike solids, cannot be individuated. In our case, the "contained-liquid" (liquid within a container) ontology introduced by Hayes [28] is used in the QPT formalism. Hence, our domain model consists of liquids, containers, and fluid paths in the process and view descriptions.

4.1 Representation of Domain Models

A domain model in QPT can be specified by defining quantities, individuals, and processes that exist in possible situations.

4.1.1 Individuals and Quantities

In QPT, objects, processes, and view instances are represented by *individuals* [14, 16]. In general, two classes of individuals are considered in a situation: static individuals and dynamic individuals. Static individuals do not vanish or appear. Dynamic individuals may do so as a result of changing quantity values.

The containers `F` and `G` (cf. Figure 2.1.) are given as static individuals; they do not appear or vanish as a result of processes in the situation. Hence, we simply use the `individual` predicate for indicating their existence: `(individual F)` and `(individual G)`.

Other individuals in fluids domain are dynamic individuals. They come into existence when some conditions are met:

```
(=> (L (substance ?s))
    (container ?c)
    (state ?st)
    (contains-substance ?c ?s ?st)
    (and (individual (?c ?s ?st))
         (has-quantity (?c ?s ?st) amount-of-in)))
```

If there is a substance `?s` in state `?st`, a container `?c` in the situation, and `?c` contains `?s` in `?st`, then an individual `(?c ?s ?st)` appears with a quantity `amount-of-in`. When the triggering fact `?s` does not hold anymore (or triggering facts in other related rules are not satisfied so far) the individual `(?c ?s ?st)` disappears.

Some individuals come into existence as a result of active process and view

instances in the situation. Those are inserted into the database when the instances are activated, and removed when they are inactivated. The "relations" part of the process and view definitions contains the new individual descriptions and properties associated with those individuals:

```
(=> (L (relations (view CONTAINED-STUFF (individuals (?c ?s ?st)))))
    (assign (c-s ?s ?st ?c) ?c-s)
    (and (introduces-uniquely contained-stuff ?c-s)
         (qprop+ (amount-of-in (?c ?s ?st)) (amount-of ?c-s))
         (Q= (amount-of-in (?c ?s ?st)) (amount-of ?c-s))))
```

`CONTAINED-STUFF` introduces an individual `(c-s ?s ?st ?c)` as a `contained-stuff` where `?s` is the substance in state `?st` (i.e., solid, liquid, or gas) and `?c` is the container in which the substance stays (when an individual `(?c ?s ?st)` exists). `Introduces-uniquely` indicates that the new individual appears when the view becomes active and vanishes when the view becomes inactive.

Some individuals and associated relations can be specified by making use of the existing ones. For example, `(c-s WATER LIQUID F)` can be inferred as a `contained-liquid` from the following forward chaining rule, when we already have an individual `(c-s WATER LIQUID F)` as a `contained-stuff` (`function-spec` defines some relationships between quantities):

```
(=> (L (contained-stuff (c-s ?s LIQUID ?c)))
    (assign (c-s ?s LIQUID ?c) ?c-s)
    (and (introduces-uniquely contained-liquid ?c-s)
         (has-quantity ?c-s level)
         (qprop+ (pressure ?c-s) (level ?c-s))
         (qprop+ (top-height ?c-s) (level ?c-s))
         (qprop+ (level ?c-s) (amount-of ?c-s))
         (qprop+ (bottom-height ?c-s) (bottom-height ?c))
         (function-spec (qprop+ (level ?c-s) (amount-of ?c-s)))
         (function-spec (qprop+ (pressure ?c-s) (level ?c-s)))
         (Q= (level ?c-s) (top-height ?c-s))
         (Q= (bottom-height ?c-s) (bottom-height ?c))
```

```
(correspondence (((A (level ?c-s)) (A (bottom-height ?c-s)))
                 ((A (amount-of ?c-s)) Zero)))))
```

The quantity types in the domain are defined by (quantity-type <*type*>). Some example quantity types in the fluids domain include amount-of, pressure, and bottom-height.

Individuals possess the quantity properties of the objects they represent, e.g., a container is a physical-object that inherits the following quantity properties:

```
(=> (L (container ?i)) (physical-object ?i))

(=> (L (physical-object ?i))
    (and (has-quantity ?i top-height)
         (has-quantity ?i bottom-height)
         (not (greater-than (A (bottom-height ?i))
                            (A (top-height ?i))))))
```

In QPT, relationships between quantities are basically indicated by *qualitative proportionalities*, *correspondences*, and *inequalities* [15]. These are coded in MVL with the same notational considerations as in QPT. Below ?c-s is a contained-liquid, Zero is the value 0, and A denotes the amount of a quantity:

```
(qprop+ (level ?c-s) (amount-of ?c-s))
(correspondence (((A (level ?c-s)) (A (bottom-height ?c-s)))
                 ((A (amount-of ?c-s)) Zero)))))
```

Here, qprop+ denotes that the level of ?c-s is qualitatively proportional to its amount. On the other hand, correspondence basically says that "The value of ?c-s's level is equal to its bottom height when the value of ?c-s's amount is equal to Zero," since level and amount-of are qualitatively proportional.

The information a correspondence carries is inserted into the database with the following rule:

```
(=> (L (correspondence ?pair-list))
    (insert-correspondence ?pair-list)
                              (correspondence-inserted))
```

`Insert-correspondence` is a Lisp function that is made known to MVL. It produces the rules needed to check the applicability of `correspondence` [15].

`Qprops` give little information about functional dependencies between quantities. `Function-specs` are used to specify further relationships. The following `function-spec` indicates a qualitative proportionality between `pressure` and `level` and specifies the equality of pressures for two `contained-stuffs` when their levels are also equal:

```
(function-spec (qprop+ (pressure ?c-s) (level ?c-s)))
```

The relationships are introduced by the following rules:

```
(=> (L (function-spec ?qprop))
    (insert-function-spec ?qprop)
                            (function-spec-inserted))

(=> (L (function-spec ?qprop))
    (car ?qprop ?rel)
    (cadr ?qprop ?q1)
    (caddr ?qprop ?q2)
                            (?rel ?q1 ?q2))
```

`Insert-function-spec` is a Lisp function to handle the insertion and the removal of the relationships a `function-spec` denotes.

A simple representation `Q=` is defined [15] for the illustration of more complex correspondences and qualitative proportionality groups:

```
(Q= (flow-rate ?ins) (Q- (pressure ?src) (pressure ?dst)))}
```

`Q=` produces the following (`?ins` is an instance of `LIQUID-FLOW` process where `?src` and `?dst` are the source and the destination `contained-liquids` in the situation):

```
(qprop+ (flow-rate ?ins) (pressure ?src))
(qprop- (flow-rate ?ins) (pressure ?dst))
(correspondence (((A (flow-rate ?ins)) Zero)
                ((A (pressure ?src)) (A (pressure ?dst))))
```

Inequalities can be given directly or inferred from the symmetry, duality, and transitivity rules defined on quantity values. `Greater-than`, `less-than`, and `equal-to` constitute a basis for algebraic manipulations:

```
(=> (L (greater-than ?q1 ?q2))
    (handle-inequalities greater-than ?q1 ?q2)
                                      (inequalities-inserted))

(=> (L (less-than ?q1 ?q2))
    (handle-inequalities less-than ?q1 ?q2)
                                    (inequalities-inserted))

(=> (L (equal-to ?q1 ?q2))
    (not (equal ?q1 ?q2))
    (handle-equalities ?q1 ?q2)
                      (equalities-inserted))
```

`Handle-inequalities` and `handle-equalities` are defined as Lisp functions known to MVL in order to reason about the equalities and the inequalities [40] in the database. Consistency checks (i.e., if $(q1 > q2)$ then $(q2 > q1)$ cannot be true), symmetry (i.e., if $(q1 = q2)$ then $(q2 = q1)$)), duality (i.e., if $(q1 > q2)$ then $(q2 < q1)$), and transitivity (i.e., if $(q1 > q2)$ and $((q2 > q3)$ or $(q2 = q3))$ then $(q1 > q3))$ rules are employed by these two functions.

4.1.2 Processes and Individual Views

Until now, we have given the basic components of a domain model; most of these consist of forward chaining rules and produce known facts about the situation. With only these rules and facts in the database, we can make simple inferences, e.g., "What kind of individuals exist in the domain?" or "What are the qualitative proportionalities?" However, we need more complex inferences for qualitative reasoning tasks. For this purpose, we are going to represent processes and other related concepts with backward and forward chaining rules.

In QRM, a *process description* is given in five parts, i.e., we have five rules for each process. One rule describes a process along with its individuals. If the specified individuals exist in the situation, the process is considered to be potentially active; i.e., a process instance. The status of an instance can only be determined by examining the preconditions and the quantity conditions. Hence, two of the rules are used to check whether preconditions and quantity conditions hold, and the remaining two employ relations and direct influences when the process is found to be active.

A sample process description for `LIQUID-FLOW` captures all necessary conditions (preconditions and quantity conditions) and individual specifications for that process. Individuals for `LIQUID-FLOW` are `contained-liquids`, `?src` and `?dst`, and `?path` which allows flow of liquid.

```
(<= (process LIQUID-FLOW (individuals ?src ?dst ?path))
    (contained-liquid ?src)
    (contained-liquid ?dst)
    (not (equal ?src ?dst))
    (fluid-path ?path)
    (fluid-connection ?path ?src ?dst)
    (supports
       (view LIQUID-FLOW-SUPPORTING
                     (individuals ?src ?dst ?path))))
```

When all the conditions hold, a process becomes `ACTIVE`. For `LIQUID-FLOW`, `(aligned ?path)` is a precondition guaranteeing that the fluid path is isolated from any other external effect:

```
(<= (hold-preconditions
       (process LIQUID-FLOW (individuals ?src ?dst ?path)))
    (aligned-path ?path))
```

If the pressure of `?src` is greater than the pressure of `?dst` and the geometric properties of `?path` allows, there will be a flow of liquid from `?src` to `?dst`:

```
(<= (hold-quantity-conditions
       (process LIQUID-FLOW (individuals ?src ?dst ?path)))
    (assign
       (process LIQUID-FLOW (individuals ?src ?dst ?path)) ?name)
    (assign
       (greater-than (A (pressure ?src)) (A (pressure ?dst))) ?c)
    (add-quantity-conditions ?name ?c)
    (hold-conditions ?c ?result)
    (not (null ?result)))
```

Lisp function `add-quantity-conditions` constructs `*comparison-table*` for limit analysis and determines the quantity conditions for process and view instances that are not found to be `ACTIVE` or `INACTIVE`. `Hold-conditions` checks whether the quantity conditions are satisfied in the situation.

The status of an instance is determined by a backward search via the following rules:

```
(<= (status (process ?name ?individuals) ACTIVE)
    (process ?name ?individuals)
    (hold-all-conditions (process ?name ?individuals)))

(<= (hold-all-conditions (process ?name ?individuals))
    (hold-preconditions (process ?name ?individuals))
    (hold-quantity-conditions (process ?name ?individuals)))
```

When the status of instance is determined to be active, the "relations" and the "influences" parts of it are invoked:

```
(=> (L (status (process ?name ?individuals) ACTIVE))
    (and (relations  (process ?name ?individuals))
         (influences (process ?name ?individuals))))
```

The relations and the influences of a LIQUID-FLOW process are defined as follows:

```
(=> (L (relations
          (process LIQUID-FLOW (individuals ?src ?dst ?path))))
    (assign
       (process LIQUID-FLOW (individuals ?src ?dst ?path)) ?ins)
    (and (has-quantity ?ins flow-rate)
         (individual ?ins)
         (greater-than (A (flow-rate ?ins)) Zero)
         (Q= (flow-rate ?ins)
             (Q- (pressure ?src) (pressure ?dst)))))

(=> (L (influences
          (process LIQUID-FLOW (individuals ?src ?dst ?path))))
    (assign
       (process LIQUID-FLOW (individuals ?src ?dst ?path)) ?ins)
    (and (I+ (amount-of ?dst) (A (flow-rate ?ins)))
         (I- (amount-of ?src) (A (flow-rate ?ins)))))
```

In QPT, processes are the only source of direct influences [14]. I+ and I- represent direct influences of flow-rate on the amount of ?src and ?dst when LIQUID-FLOW is ACTIVE. If flow-rate is increasing, then the amount of ?dst will increase whereas the amount of ?src will decrease.

The individual views are described using the same representation scheme.

4.2 Basic Deductions

4.2.1 Finding Possible Processes

Possible processes are characterized by their individuals. Processes whose individuals exist in a situation are potentially active and can be found by a simple inference on the rules of process descriptions. The same procedure is also applicable for individual views. The following MVL query (backward) searches for possible process and view instances in the situation:

```
(bcs '(p-v-instances ?name))
```

The process and view instances are added into `*process-view-instances*` using the rules:

```
(<= (p-v-instances ?p-name)
                (process ?p-name ?individuals)
                (add-process-instance ?p-name ?individuals t))

(<= (p-v-instances ?v-name)
                (view ?v-name ?individuals)
                (add-view-instance ?v-name ?individuals t))
```

In order to find all possible instances in the situation, new individuals that may exist after activating the instances should also be considered. The procedure which handles that is called *elaboration* (Figure 4.1).

4.2.2 Determining Activity

Process instances found by elaboration can be `ACTIVE` if they satisfy their conditions. To find which process instances are `ACTIVE`, we may pose the following query which tries to prove whether the potential processes are `ACTIVE`:

```
(bcs '(status ?instance ACTIVE))
```

```
ELABORATION

1. repeat until no new instances or individuals are found
1.2. for each process and view in the database
1.2.1 find the process and view instances
1.2.2. add comparisons of quantity conditions
       for instances into *comparison-table*
1.3. for each new instance
1.3.1. if instance is found to be ACTIVE then activate it
1.3.2. find new individuals after activating the instance
2. inactivate ACTIVE process and view instances
```

Figure 4.1. The algorithm for elaboration

When a process or view instance cannot be found `ACTIVE` or `INACTIVE` its status is `UNKNOWN`. The quantity conditions of such an instance are determined in order to complete the situation for further reasoning.

4.2.3 Determining Change

In QPT, change is imposed by active processes (processes are the only source of direct influences) [14, 16]. Quantities may change because of direct or indirect influences on them. A quantity is said to be *directly influenced* if there exists at least one process directly influencing it at a particular time. On the other hand, a quantity is *indirectly influenced* if it is a function of some other quantity that is changing. The derivative of a directly influenced quantity is equal to the sum of all the direct influences on it. In QRM, an *influence adder* is used to find this derivative value [15].

The algorithm used for resolving influences of the quantities of a process instance is shown in Figure 4.2.

The quantities for individuals in the situation are determined by backward searching with the following rule:

```
(<= (quantities ?obj) (individual ?obj)
                      (has-quantity ?obj ?q)
                      (sign-of-derivative ?q ?obj ?sign-val)
                      (mg-of-derivative ?q ?obj ?mg-val)
                      (add-quantity ?q ?obj ?sign-val ?mg-val)
                      (directly-influenced ?q ?obj))
```

Lisp functions `sign-of-derivative` and `mg-of-derivative` retrieve the sign and the magnitude of a quantity derivative from the database (if already known) whereas `add-quantity` forms `*quantities*`. `Directly-influenced` computes the sign of the derivative for a directly influenced quantity by constructing an influence adder [15].

`Add-p-influence` and `add-n-influence` are used to find the positive and the negative direct influences on a quantity:

```
(<= (inf+ ?q ?inf) (I+ ?q ?inf)
                   (add-p-influence ?inf))
(<= (inf- ?q ?inf) (I- ?q ?inf)
                   (add-n-influence ?inf))
```

Resolution of indirectly influenced quantities requires the constrainer and the constrainee sets for a quantity to be formed. Hence, Lisp functions `add-p-constrainer`, `add-n-constrainer`, and `add-constrainee` are defined:

```
(<= (p-constrainers ?q1 ?q2) (qprop+ ?q1 ?q2)
                             (add-p-constrainer ?q2))
(<= (n-constrainers ?q1 ?q2) (qprop- ?q1 ?q2)
                             (add-n-constrainer ?q2))
(<= (constrainees ?q1 ?q2) (qprop ?q2 ?q1)
                           (add-constrainee ?q2))
(<= (qprop ?q1 ?q2) (or (qprop+ ?q1 ?q2)
                        (qprop- ?q1 ?q2)))
```

```
                    RESOLVING-INFLUENCES

1. find *quantities* of individuals that exist in the situation
2. find all direct and indirect influences in the situation
3. find derivative values of *quantities*,
   determine the order of resolution
3.1. sort *quantities* depending on their depth of influence
3.1.1. for each Q in *quantitites* mark Q with 0
3.1.2. for each directly influenced Q MARK-DEPTH (Q,1)
3.2. return *quantities* in increasing order of marks into QUEUE
4. for each Q in QUEUE do
     if Q is directly influenced then
       if sum of influence adder is known
       then derivative of Q is this sum
       else
         begin
           find CONSTRAINERS of Q (i.e., take Q1 if Q qprop Q1)
           for each Q do
              for each Q1 in CONSTRAINERS do
                 form PLUS, MINUS, and UNKNOWN sets
                 if UNKNOWN is nonempty or both
                    PLUS, MINUS sets are nonempty
                 then the sign of Q's derivative is undecided
                 else
                   if PLUS is nonempty then the sign is 1
                   else
                     if MINUS is nonempty then the sign is -1
                     else the sign is 0
         end


                  MARK-DEPTH (Q, DEPTH)

1. if mark of Q is less than DEPTH then
1.1. mark Q with DEPTH
1.2. find CONSTRAINEES of Q (i.e., take Q1 if Q1 qprop Q)
1.3. for each Q1 in CONSTRAINEES MARK-DEPTH (Q1, DEPTH+1)
```

Figure 4.2. The algorithm for resolving influences

4.2.4 Limit Analysis

Limit analysis is the most complex of the basic deductions [15]. It is used to determine the changes in process and view structures, hence the changes in D_s values of quantities, when the activity of processes changes.

The algorithm employed for limit analysis (Figure 4.3) constructs a set of single quantity hypotheses by finding possible consistent changes in quantity orderings [14]. The neighboring points in the quantity space designate the changing quantities in the situation.

Since it is possible for some quantities to change together, combinations of single quantity hypotheses are formed and tested for consistency. Consistent hypotheses are then added.

Limit analysis may sometimes generate impossible physical situations [27, 33]. These situations and inherent ambiguities can be reduced using domain dependent information.

4.3 Envisioning

Predicting the dynamic behavior of physical systems has been a challenging research area in qualitative reasoning [14, 36, 33]. Envisioning, history generation, and measurement interpretation are all styles of reasoning related to prediction.

Envisioning is a kind of qualitative reasoning where a graph of transitions between the *qualitative states* of a given situation is formed by predicting the future events [34, 35, 18, 36]. In general, an envisioning algorithm relies on the generation of next situations and matching those situations with existing ones. The envisioning algorithm used by QRM is illustrated in Figure 4.4.

The graph of qualitative state descriptions formed by envisioning is called an *envisionment*. When constructing the envisionment two types of inference are needed most of the time: one for the current situation to be completed and the other for the next situations to be generated [15].

```
                    LIMIT-ANALYSIS

 1. find the set of quantity hypotheses
 1.1. create and update the quantity spaces for
      the quantities in *comparison-table*
      by finding the neighboring quantities
 1.2. update *comparison-table*
 1.3. for each quantity in the quantity space
 1.3.1. if inequality relationship can change then
        create a quantity hypothesis
        record assumptions about rate of change
 2. generate all possible combinations of hypotheses
 3. for each quantity hypothesis combination
 3.1. form new situation
 3.1.1. impose changes via hypotheses
 3.2. determine the consistency of the situation
 3.3. if hypotheses do not cause any inconsistency
      then
         record changes quantity hypotheses cause
         into *next-situations*
 3.4. form previous situation
 3.4.1. remove changes via hypotheses
```

Figure 4.3. The algorithm for limit analysis

```
                    ENVISIONING

1. let S be the initial situation, *situations* be {S}
   SITUATION-COMPLETIONS and *envisionment* be NIL
2. if process or view structures of S is incomplete
   then SITUATION-COMPLETIONS is R-COMPLETIONS(S-COMPLETIONS(S))
   else if any Ds value in S is unknown
        then SITUATION-COMPLETIONS is R-COMPLETIONS(S)
        else SITUATION-COMPLETIONS will consist of only {S}
3. while SITUATION-COMPLETIONS is not empty
3.1. take the first element of SITUATION-COMPLETIONS
     and assign this alternative situation to S1
3.2. perform LIMIT-ANALYSIS on situation S1 and
     form *next-situations* with quantity hypotheses
3.3. for each situation S2 in *next-situations*
3.3.1. if it matches any situation S3 in *situations*
       then
         form a transition from S1 to S3 and insert it
         into *envisionment* with quantity hypotheses
       else
         begin
           form a transition from S1 to S2 and insert it
           into *envisionment* with quantity hypotheses,
           add S2 into *situations*
           if all Ds values in S2 is known
           then add S2 to SITUATION-COMPLETIONS
           else add R-COMPLETIONS(S2) to SITUATION-COMPLETIONS
         end

                    S-COMPLETIONS(S)
1. find alternative situations with consistent status assignments

                    R-COMPLETIONS(S)
1. find alternative situations with consistent influence resolutions
```

Figure 4.4. The algorithm for envisioning

4.4 Basic Experiments

In this section, some qualitative reasoning tasks performed in the fluids domain (cf. Appendix A) will be presented with the input scenarios (cf. Appendix B) of the physical situations. The example physical systems are similar to ones that are also used to test GIZMO [15] and QPE [21]. They are naive models but are informative about the reasoning process in more complex systems.

A scenario for a physical situation captures the definitions of static individuals, facts about the physical system, and facts about the initial situation. The scenarios for situations are input using the following scheme:

```
(=> (L (scenario ?s))
                    (and (individuals ?s)
                         (facts ?s)
                         (always ?s)
                         (initial-situation-facts ?s)))
```

The assumptions are denoted by `dt` and they remain true until information to the contrary is given. Other facts inserted as the triggering effect of these assumptions will also be labelled by `dt`. Hence, the assumptions are propagated through the rules during the inference process. The physical situation is described by the propositions and their truth values in the MVL database.

Liquid Flow (Two Containers) In this experiment, there are two containers `F` and `G`, a fluid path `P` between them, and some water (cf. Figure 2.1). `F` and `G` have the same shape characteristics. `P` is not effected by the external world and allows the flow of liquid between `F` and `G`.

The amount of water in `F` is initially given to be more than the amount of water in `G`, and no steam exists in the situation. Then, we expect that the amount of water in `F` will decrease while the amount of water in `G` will increase.

In Figure 4.5 the envisionment is shown graphically. Water in the containers causes two individual views, namely `CONTAINED-STUFF`s for the water in `F` and `G`, to be activated along with the `LIQUID-FLOW` process. There is a flow of water

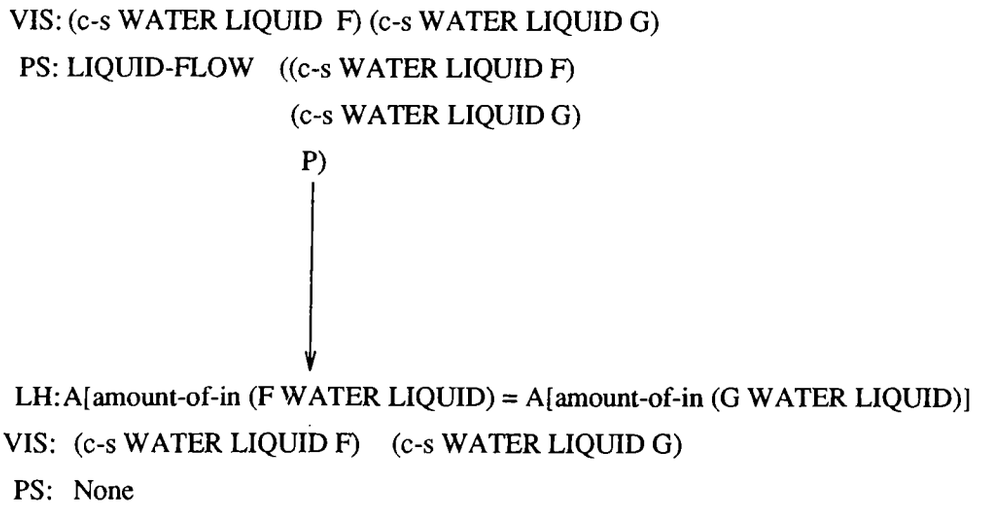


Figure 4.5. Envisionment for the two-container system. (VIS stands for individuals imposed by the views, PS indicates process structures in the situation, and LH denotes the limit hypothesis.)

from F to G initially. Envisioning predicts that after some time the amount of water in F will be equal to the amount of water in G and the flow of water will stop. Henceforth, no process will be active and the system will stay in this stable state.

Heat Flow and Boiling In this experiment, there is a container with some water, placed on a stove (Figure 4.6). The container is assumed to be closed and does not allow the flow of steam (assuming no explosion). STOVE is described as a constant heat source so that its temperature does not change. BURNER is the heat path between STOVE and the water in CAN. By default, BURNER does not allow heat flow, i.e., STOVE does not provide heat initially. This is inferred from the domain model and is indicated by df. Then, the scenario implies in its always specification that BURNER will allow heat flow, i.e., STOVE supplies heat—the truth value for (heat-aligned-path BURNER) will be updated as true. The heating of CAN will be ignored for simplicity's sake.

Initially, a HEAT-FLOW process is active in the situation indicating the flow

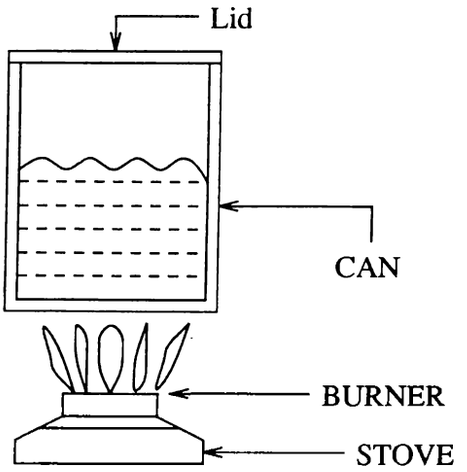


Figure 4.6. Heat flow and boiling. (STOVE is assumed to be a temperature source, CAN is closed, and no explosion is considered.)

of heat from STOVE to the water in CAN. Thus the temperature of the water increases. Figure 4.7 shows that envisioning predicts the following three alternative situations:

- The temperature of the water in CAN will be equal to the temperature of STOVE (LH1) and the flow of heat will stop. No process will be active afterwards.

- The temperature of the water in CAN will be equal to the boiling temperature of the water (LH2). Hence, BOILING will be active and steam will be generated causing a heat flow from STOVE to the steam in CAN. After some time, the liquid water in CAN will vanish and only steam will exist. When the temperature of the steam in CAN rises to the temperature of STOVE, heat flow will stop and no other change will take place.

- The temperature of the water in CAN will be equal to the temperature of STOVE and the temperature of the water in CAN will be equal to the boiling temperature of the water (LH3). Since the temperatures of water in CAN and STOVE are the same, HEAT-FLOW will no longer be active.

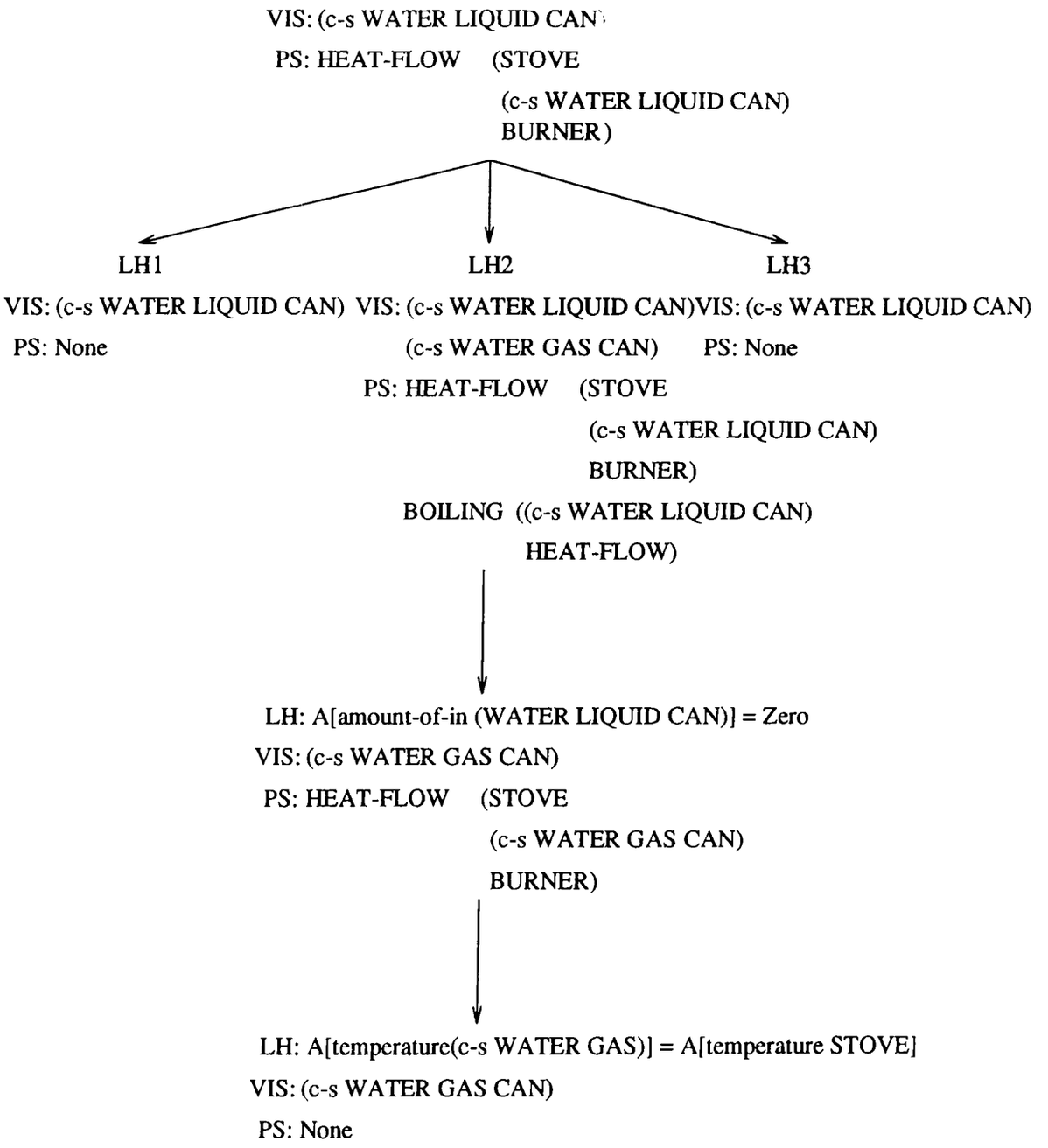


Figure 4.7. Envisionment for heat flow and boiling system. (LH1, LH2, and LH3 correspond to limit hypotheses.)

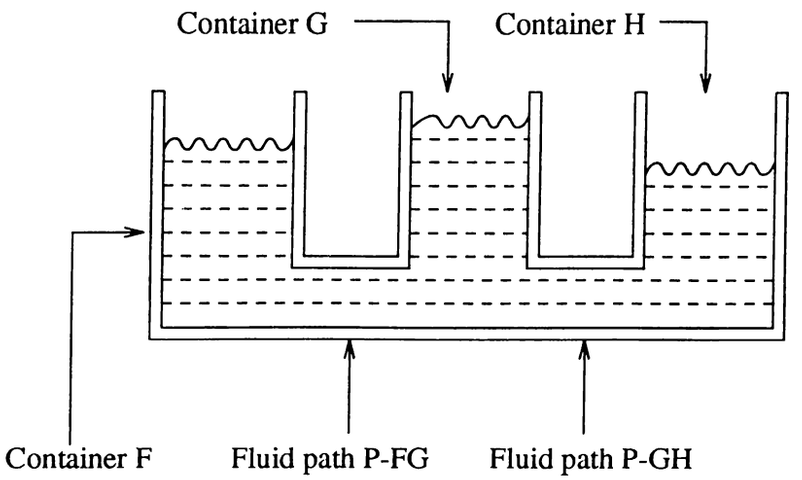


Figure 4.8. The three-container system

Liquid Flow (Three Containers) This experiment has three containers F, G, and H that have the same shape and some water in them. There are fluid paths between containers (Figure 4.8). The paths P-FG and P-GH are between F and G, and between G and H, respectively. The fluid paths are initially assumed to allow flow of water (indicated by dt in the scenario). However, an external effect, say a controller which measures the amount of water in the containers, may close the fluid paths. (P-FG will be closed when the amount of water in F is equal to the water in G and P-GH will be closed when the amount of water in G is equal to the water in H.) Initially, there is water flow from G to F and from G to H.

Envisioning predicts three alternative situations:

- The amount of water in F will be equal to the amount of water in G (cf. Figure 4.9). Hence, the flow of water from G to F will stop. On the other hand, the flow of water from G to H will decrease the amount of water in G. The amount of water in F would not change since the controller closed P-FG. After some time, water in G will be equal to the water in H and the water flow will stop. The controller will close P-GH.

- The amount of water in H will be equal to the amount of water in G. The flow of water from G to H will no longer take place and P-GH will be

closed. The flow of water from G to F will increase the amount of water in F. When the amounts of water in F and G are the same, water flow will stop. P-FG will be closed.

- Both the amount of water in F and the amount of water in H will be equal to the amount of water in G (cf. Figure 4.10). Both of the fluid paths will be closed. No process will be active and hence no change will occur.

First two predictions show impossible physical situations which we call *spurious* situations. The third prediction illustrates what will normally occur in this physical system.

Representation of external effects, e.g., the controller above, is needed when reasoning about some physical systems. Preconditions in QPT can initially be assumed to be true (dt), allowing them to be falsified by the external effects without causing an inconsistency in the database. For example, when someone closes the fluid path mentioned above, the default true fact which says the fluid path is open will be updated as `false`. If we consider first-order logic instead, the truth value for "the fluid path is open" would simply be `true`. When the fluid path is closed the truth value computed would be `bottom` which indicates an inconsistency about the fact—it is both `true` and `false`. Hence, effects outside of QPT can be incorporated into the reasoning process using the nonmonotonic nature of MVL's default logic.

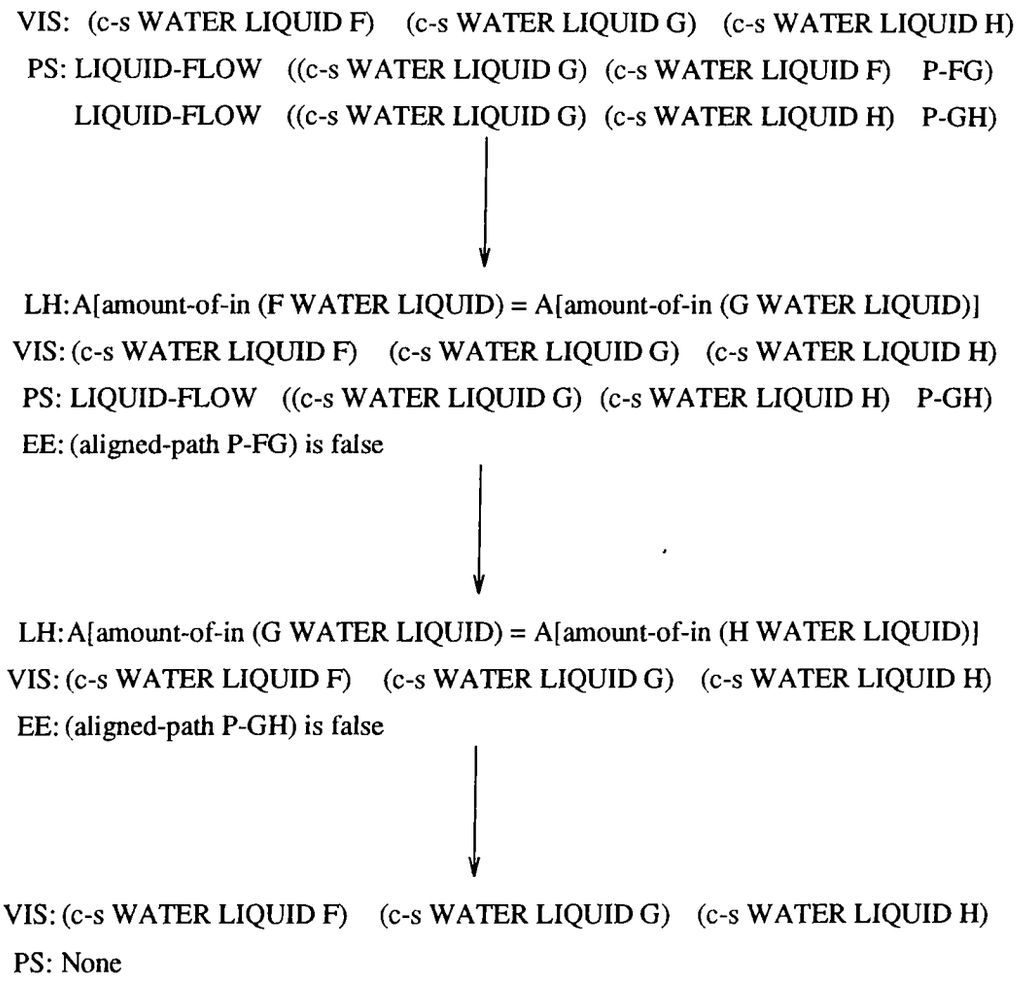


Figure 4.9. A part of the envisionment for the first alternative situation. The second alternative situation is similar. (EE denotes an external effect such as the controller.)

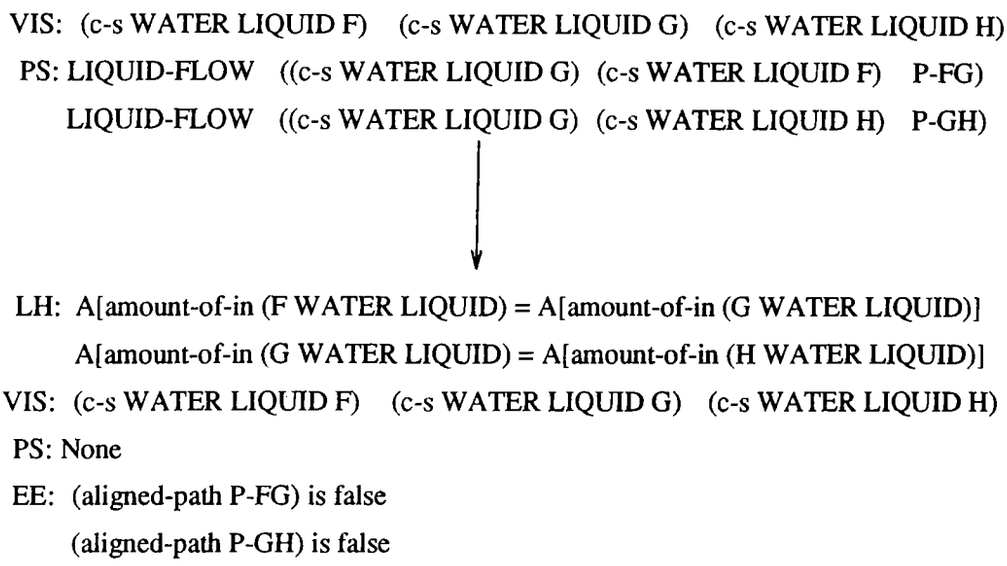


Figure 4.10. A part of the envisionment for the third alternative situation

4.5 Limitations of QRM

QRM works within the fluids domain. What about other domains? Is it still possible to reason about other physical systems? The answer to this question relies on the inherent limitations of QPT, and qualitative reasoning in general.

QPT provides an extensive language for modeling physical systems, especially dynamical systems. Currently, there are various theories for modeling physical systems [42]. However, these theories alone are not suitable for reasoning about all the systems in the physical world. (For example, the component-based theory [8] is fine for reasoning about systems such as electronic circuits but not for systems where individuals appear and vanish.) Since QRM uses QPT formalism, any physical system that can be modeled with QPT should be available to our program.

On the other hand, QPT has its own limitations; it may produce ambiguous or spurious behavior descriptions. In general, ambiguity can be identified by the prediction of several possible behaviors for a physical situation. Information about magnitudes of quantities or about relative magnitudes of numbers would

be helpful in reducing the ambiguity. D'Ambrosio [1, 5] proposes linguistic influence sensitivities along this line.

Spurious behaviors are physically impossible behaviors that are produced by the reasoning system. These may be avoided using domain dependent knowledge. Another trick would be to generate the next states of a situation considering not only the current state but the previous states. The liquid flow example (Figure 4.9) generates spurious behavior descriptions [15, 14].

Qualitative reasoning programs, in general, consume a considerable amount of computing power [15, 21]. This is also true for QRM. The assumptions made about situations and consistency checks for those assumptions along with the main reasoning tasks take substantial computer time and space. ATMS (Assumption-based Truth Maintenance Systems) [9] offer some promise in reducing the time and space complexity. Since QRM was implemented as an experimental tool, efficiency was not our major concern. However, this is clearly essential when reasoning about more complex physical systems.

In QRM, the behavior of a physical system is described using state transitions. Time inheritance is realized through propositions that do not change during the course of reasoning. Hence, no explicit representation for time is used. When envisioning, QRM takes only the initial situation description. Time varying inputs and external effects cannot be described. Several possible future events for a situation cannot be ordered, either.

The temporal logic in MVL provides three modal operators that facilitate reasoning about durations and delays as well as specific time points [26]:

`delay(t p)` pushes the truth value of `p` into the future by an amount of `t` time.

`propagate(p)` propagates the truth value of `p` between time points where the truth value is known at time points and not known between.

`at(t p)` returns the truth value of `p` at the time point `t`.

With these operators, time dependent effects can be reflected into the model of a physical system. The physical behavior can be described in time order by assigning time points to truth values of propositions.

The controller in the three-container experiment can close the fluid path for only an instant, say 1 second, using the following:

```
(delay 1 (not (aligned-path P-FG)))

(delay 1 (not (aligned-path P-GH)))
```

If the fluid paths are known to be open at time point 5 and closed at time point 8 then we may assume that they are also open at time points 6 and 7. The propositions `(propagate (aligned-path P-FG))` and `(propagate (aligned-path P-GH))` provide this during inference.

Having the time information about the state of a physical system proves to be useful for prediction. For example, the envisionment tree would be constructed more precisely if we could know which change occurs first.

The heat flow and boiling example presents three different next situations for the given initial situation. If we know that the temperature of water in `CAN` will be equal to the temperature of `STOVE` before it is equal to the boiling temperature then only the first prediction would be valid. For example, suppose that the system is at time point 3. The following queries give the truth value for the hypotheses:

```
(bc '(at 3 (equal-to (A (temperature (c-s WATER LIQUID CAN)))
                     (A (temperature STOVE)))))

(bc '(at 3 (equal-to
             (A (temperature (c-s WATER LIQUID CAN)))
             (A (boiling-temperature (c-s WATER LIQUID CAN))))))
```

Only the first hypothesis is found to be true at time point 3. Hence, the heat flow will stop and boiling will not occur at all. The only possible next situation will be the one specified by the first prediction. On the other hand, since QRM cannot order possible changes it gives all of them as potential next situations.

Chapter 5

CONCLUSION

An experimental program, QRM, for qualitative reasoning about dynamical systems has been introduced. The relevant knowledge representation and inference issues have been addressed.

In general, representation of physical systems plays an important role in qualitative reasoning. A clear and rich representation proves to be useful when building the domain models and reasoning about the domain [6, 39, 27]. Forbus's QPT [14] is considered to be a comprehensive modeling language in this regard, especially for dynamical systems. Accordingly, the domain models in QRM are constructed according to QPT.

QRM makes use of the MVL theorem proving system [24] for representation of domain models and basic reasoning tasks. MVL provides forward- and backward chaining rules (allowing Lisp functions) during the inference process, as well as other multivalued inference facilities. The underlying bilattice structure of the prover enables one to define new logics. For example, MVL's temporal logic constitutes a valuable tool for temporal reasoning [43].

An important style of qualitative reasoning, namely envisioning, has been developed within QRM in order to predict what might happen in the future of a given physical situation. The default logic of MVL guarantees the reasoning to continue even in the case of incomplete information.

In the future, other forms of qualitative reasoning, i.e., measurement interpretation, postdiction, fault diagnosis, and comparative analysis [42, 19, 41],

and extensions to QPT [1, 37, 21, 13], e.g., linguistic influence variables [1, 5], may be embedded into QRM.

Appendix A

A View on Fluids Domain Model

A.1 Heat Flow

```
;;
;; a temperature source
;;
(=> (L (temperature-source ?i))
    (and (physical-object ?i)
         (has-quantity ?i heat)
         (has-quantity ?i temperature)))

;;
;; initially a burner does not allow heat flow
;;
:value dt
(=> (L (heat-path ?path))
    (burner ?path)
    (not (heat-aligned-path ?path)))

:value true
```

```
;;
;; heat connection is not symmetric between ?src
;; and ?dst, if ?dst is a temperature source
;;
(=> (L (heat-connection ?path ?src ?dst))
    (not (equal ?src ?dst))
    (lookup (temperature-source ?dst) ?res)
    (null ?res)
    (heat-connection ?path ?dst ?src))

;;
;; HEAT-FLOW process
;;
(<= (process HEAT-FLOW (individuals ?src ?dst ?path))
    (physical-object ?src)
    (physical-object ?dst)
    (not (equal ?src ?dst))
    (has-quantity ?src heat)
    (has-quantity ?dst heat)
    (heat-path ?path)
    (heat-connection ?path ?src ?dst))

(<= (hold-preconditions
      (process HEAT-FLOW (individuals ?src ?dst ?path)))
    (heat-aligned-path ?path))

(<= (hold-quantity-conditions
      (process HEAT-FLOW (individuals ?src ?dst ?path)))
    (assign
      (process HEAT-FLOW (individuals ?src ?dst ?path)) ?name)
    (assign
      (greater-than (A (temperature ?src))
                    (A (temperature ?dst))) ?cond)
    (add-quantity-conditions ?name ?cond)
    (hold-conditions ?cond ?result)
    (not (null ?result)))
```

```
(=> (L (relations (process HEAT-FLOW (individuals ?src ?dst ?path))))
    (assign (process HEAT-FLOW (individuals ?src ?dst ?path)) ?ins)
    (and (has-quantity ?ins flow-rate)
         (individual ?ins)
         (process-instance ?ins)
         (greater-than (A (flow-rate ?ins)) Zero)
         (Q= (flow-rate ?ins)
             (Q- (temperature ?src) (temperature ?dst)))))
```

A.2 Boiling

```
;;
;; BOILING process
;;
(<= (process BOILING (individuals ?l ?heat-flow))
    (contained-liquid ?l)
    (process-instance ?heat-flow)
    (cadr ?heat-flow HEAT-FLOW)
    (caddr ?heat-flow ?i)
    (caddr ?i ?l))

(hold-preconditions
  (process BOILING (individuals ?l ?heat-flow)))

(<= (hold-quantity-conditions
      (process BOILING (individuals ?l ?heat-flow)))
    (assign
      (process BOILING (individuals ?l ?heat-flow)) ?name)
    (or (and (assign
               (greater-than (A (temperature ?l))
                             (A (boiling-temperature ?l))) ?cond)
             (add-quantity-conditions ?name ?cond)
             (hold-conditions ?cond ?result))
        (and (assign
               (equal-to (A (temperature ?l))
```

```
                        (A (boiling-temperature ?l))) ?cond)
            (add-quantity-conditions ?name ?cond)
            (hold-conditions ?cond ?result)))
    (not (null ?result))
    (status ?heat-flow ACTIVE))

(=> (L (relations (process BOILING (individuals ?l ?heat-flow))))
    (assign
      (process BOILING (individuals ?l ?heat-flow)) ?ins)
    (cadr ?l ?s)
    (cadddr ?l ?c)
    (assign (c-s ?s GAS ?c) ?c-s)
    (and (introduces contained-stuff ?c-s)
         (process-instance ?ins)
         (has-quantity ?ins generation-rate)
         (has-quantity ?ins absorption)
         (individual ?ins)
         (val (sign (D (heat ?c-s))) 0)
         (greater-than (A (absorption ?ins)) Zero)
         (greater-than (A (generation-rate ?ins)) Zero)
         (qprop+ (generation-rate ?ins) (flow-rate ?heat-flow))
         (Q= (temperature ?liquid) (temperature ?c-s))))

(=> (L (influences (process BOILING (individuals ?l ?heat-flow))))
    (assign
      (process BOILING (individuals ?l ?heat-flow)) ?ins)
    (cadr ?l ?s)
    (cadddr ?l ?c)
    (assign (c-s ?s GAS ?c) ?c-s)
    (and (I- (heat ?l) (A (flow-rate ?heat-flow)))
         (I- (heat ?c-s) (A (absorption ?ins)))
         (I+ (amount-of ?c-s) (A (generation-rate ?ins)))
         (I- (amount-of ?l) (A (generation-rate ?ins)))))
```

Appendix B

Scenarios for Examples

B.1 Liquid Flow (Two Containers)

```
(=> (L (individuals two-containers))
    (and (individual F)
         (individual G)
         (individual P)))

(=> (L (facts two-containers))
    (and (container F)
         (container G)
         (substance WATER)
         (contains-substance F WATER LIQUID)
         (contains-substance G WATER LIQUID)
         (fluid-path P)
         (fluid-connection P
                           (c-s WATER LIQUID F)
                           (c-s WATER LIQUID G))))

(=> (always two-containers)
    (and (equal-to (A (max-height P)) (A (bottom-height G)))
         (equal-to (A (max-height P)) (A (bottom-height F)))
         (aligned-path P)))
```

```
(=> (L (initial-situation-facts two-containers))
    (and
        ;; there is some water in F and G
        (greater-than (A (amount-of-in (F WATER LIQUID))) Zero)
        (greater-than (A (amount-of-in (G WATER LIQUID))) Zero)
        ;; water in F is more than water in G
        (greater-than (A (amount-of-in (F WATER LIQUID)))
                      (A (amount-of-in (G WATER LIQUID))))))
```

B.2 Heat Flow and Boiling

```
(=> (L (individuals boiling))
    (and (individual CAN)
         (individual BURNER)
         (individual STOVE)))

(=> (L (facts boiling))
    (and (closed-container CAN)
         (substance WATER)
         (burner BURNER)
         (temperature-source STOVE)
         (contains-substance CAN WATER LIQUID)
         (contains-substance CAN WATER GAS)
         (heat-path BURNER)
         (heat-connection BURNER STOVE (c-s WATER LIQUID CAN))
         (less-than (A (temperature (c-s WATER GAS CAN)))
                    (A (temperature STOVE)))))

(=> (always boiling)
    (and (heat-connection BURNER STOVE (c-s WATER GAS CAN))
         (heat-aligned-path BURNER)))

(=> (L (initial-situation-facts boiling))
    (and
        ;; there is some water in CAN
```

```
        (greater-than (A (amount-of-in (CAN WATER LIQUID))) Zero)
        ;; no steam exist
        (equal-to (A (amount-of-in (CAN WATER GAS))) Zero)
        (less-than (A (temperature (c-s WATER LIQUID CAN)))
                   (A (temperature STOVE)))
        (less-than (A (temperature (c-s WATER LIQUID CAN)))
                   (A (boiling-temperature (c-s WATER LIQUID CAN))))))
```

B.3 Liquid Flow (Three Containers)

```
(=> (L (individuals three-containers))
    (and (individual F)
         (individual G)
         (individual H)
         (individual P-FG)
         (individual P-GH)))

(=> (L (facts three-containers))
    (and (container F)
         (container G)
         (container H)
         (substance WATER)
         (contains-substance F WATER LIQUID)
         (contains-substance G WATER LIQUID)
         (contains-substance H WATER LIQUID)
         (fluid-path P-FG)
         (fluid-path P-GH)
         (fluid-connection P-FG
                           (c-s WATER LIQUID F)
                           (c-s WATER LIQUID G))
        (fluid-connection P-GH
                           (c-s WATER LIQUID G)
                           (c-s WATER LIQUID H))))
```

```
;;
;; fluid paths are assumed to be open
;;
:value dt
(=> (L (facts three-containers))
    (and (aligned-path P-FG)
         (aligned-path P-GH)))


:value true

(=> (always three-containers)
    (and (equal-to (A (max-height P-FG)) (A (bottom-height F)))
         (equal-to (A (max-height P-FG)) (A (bottom-height G)))
         (equal-to (A (max-height P-GH)) (A (bottom-height G)))
         (equal-to (A (max-height P-GH)) (A (bottom-height H)))
         (not (less-than (A (amount-of-in (G WATER LIQUID)))
                         (A (amount-of-in (H WATER LIQUID)))))
         (not (less-than (A (amount-of-in (G WATER LIQUID)))
                         (A (amount-of-in (F WATER LIQUID)))))))

(=> (L (initial-situation-facts three-containers))
    (and
        (greater-than (A (amount-of-in (F WATER LIQUID))) Zero)
        (greater-than (A (amount-of-in (G WATER LIQUID))) Zero)
        (greater-than (A (amount-of-in (H WATER LIQUID))) Zero)
        (equal-to (A (amount-of-in (F WATER GAS))) Zero)
        (equal-to (A (amount-of-in (G WATER GAS))) Zero)
        (equal-to (A (amount-of-in (H WATER GAS))) Zero)
        (greater-than (A (amount-of-in (G WATER LIQUID)))
                      (A (amount-of-in (F WATER LIQUID))))
        (greater-than (A (amount-of-in (G WATER LIQUID)))
                      (A (amount-of-in (H WATER LIQUID))))))
```

```
;;
;; the effects of the controller
;;
:value true
(=> (L (equal-to (A (amount-of-in (G WATER LIQUID)) Zero)
                 (A (amount-of-in (F WATER LIQUID)) Zero)))
    (and (initial-truth-value (aligned-path P-FG))
         (not (aligned-path P-FG))
         (final-truth-value (aligned-path P-FG))))

(=> (L (equal-to (A (amount-of-in (F WATER LIQUID)))
                 (A (amount-of-in (G WATER LIQUID)))))
    (and (initial-truth-value (aligned-path P-FG))
         (not (aligned-path P-FG))
         (final-truth-value (aligned-path P-FG))))

(=> (L (equal-to (A (amount-of-in (G WATER LIQUID)))
                 (A (amount-of-in (H WATER LIQUID)))))
    (and (initial-truth-value (aligned-path P-GH))
         (not (aligned-path P-GH))
         (final-truth-value (aligned-path P-GH))))

(=> (L (equal-to (A (amount-of-in (H WATER LIQUID)))
                 (A (amount-of-in (G WATER LIQUID)))))
    (and (initial-truth-value (aligned-path P-GH))
         (not (aligned-path P-GH))
         (final-truth-value (aligned-path P-GH))))
```

Appendix C

How to Access QRM

The code for QRM is available under the directory

```
~sencan/mvl/QRM
```

QRM consists of various .lisp files and .mvl files residing in this directory. Currently the total Lisp code is about 4000 lines and the total MVL code is about 1500 lines.

.lisp files:

QRM.lisp	complete-situation.lisp	determine-activity.lisp
elaboration.lisp	envision.lisp	functions.lisp
individuals.lisp	inequalities.lisp	init-globals.lisp
limit-analysis.lisp	lisp-p-f.lisp	macros.lisp
resolve-influences.lisp	save.lisp	

.mvl files:

domain-bc.mvl	domain-fc.mvl	elaboration.mvl
fluids-domain-tax.mvl	fluids-domain.mvl	individuals.mvl
inequalities.mvl	number-tax.mvl	resolve-influences.mvl
scenario.mvl		

The README file in this directory explains how to load QRM within MVL and how to perform qualitative reasoning tasks on the examples of liquids domain.

N.B. MVL (updated version) can be obtained via anonymous ftp from `t.stanford.edu`. The `mvl` directory in this machine contains all the source files, examples, and the documentation.

Bibliography

[1] V. Akman. Review of *Qualitative Process Theory Using Linguistic Variables. SIGART Bulletin*, 2:25–27, 1991.

[2] V. Akman and M. Ü. Şencan. Qualitative Process Theory and Multivalued Logics. In E. Gelenbe, U. Halıcı, and N. Yalabık, editors, *Proceedings of Seventh International Symposium on Computer and Information Sciences*, pages 221–227. Presses de l'Ecole des Hautes Etudes en Informatique, Université René Descartes, Paris, 1992.

[3] V. Akman and P. J. W. ten Hagen. The Power of Physical Representations. *AI Magazine*, 10:49–65, 1989.

[4] D. G. Bobrow, editor. *Qualitative Reasoning About Physical Systems.* MIT Press, Cambridge, MA, 1985.

[5] B. D'Ambrosio. *Qualitative Process Theory Using Linguistic Variables.* Springer-Verlag, New York, 1989.

[6] E. Davis. *Representations of Commonsense Knowledge.* Morgan Kaufmann, San Mateo, CA, 1990.

[7] J. de Kleer. Qualitative and Quantitative Knowledge in Classical Mechanics. Technical Report TR–352, MIT AI Lab., Cambridge, MA, 1975.

[8] J. de Kleer. A Qualitative Physics Based on Confluences. *Artificial Intelligence*, 24:7–83, 1984.

[9] J. de Kleer. An Assumption-based Truth Maintenance System. In M. L. Ginsberg, editor, *Readings in Nonmonotonic Reasoning*, pages 280–297. Morgan Kaufmann, San Mateo, CA, 1987.

[10] J. de Kleer. Qualitative Physics: A Personal View. In D. S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning About Physical Systems*, pages 1–8. Morgan Kaufmann, San Mateo, CA, 1990.

[11] J. de Kleer. A View on Qualitative Physics. *Artificial Intelligence*, 59:105–114, 1993.

[12] B. Falkenheiner and K. D. Forbus. Compositional Modeling: Finding the Right Model for the Job. *Artificial Intelligence*, 51:95–143, 1991.

[13] K. D. Forbus. Qualitative Reasoning About Space and Motion. In D. Gelentner and A. Stevens, editors, *Mental Models*, pages 53–73. Erlbaum, Hillsdale, NJ, 1983.

[14] K. D. Forbus. Qualitative Process Theory. *Artificial Intelligence*, 24:85–168, 1984.

[15] K. D. Forbus. *Qualitative Process Theory.* Ph.D. thesis, Electrical Engineering and Computer Science Department, MIT, Cambridge, MA, 1984.

[16] K. D. Forbus. The Problem of Existence. Technical Report UILU–ENG–85–1747, Department of Computer Science, University of Illinois at Urbana Champaign, Urbana, IL, 1985.

[17] K. D. Forbus. The Role of Qualitative Dynamics in Naive Physics. In J. R. Hobbs and R. C. Moore, editors, *Formal Theories of the Commonsense World*, pages 185–226. Ablex, Norwood, NJ, 1985.

[18] K. D. Forbus. The Logic of Occurrence. Technical Report UILU–ENG–86–1778, Department of Computer Science, University of Illinois at Urbana Champaign, Urbana, IL, 1986.

[19] K. D. Forbus. Interpreting Observations of Physical Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:350–359, 1987.

[20] K. D. Forbus. Intelligent Computer-Aided Engineering. *AI Magazine*, 9:23–36, 1988.

[21] K. D. Forbus. QPE: Using Assumption-based Truth Maintenance for Qualitative Simulation. *Artificial Intelligence in Engineering*, 3:200–215, 1988.

[22] K. D. Forbus. Qualitative Physics: Past, Present, and Future. In H. Shrobe, editor, *Exploring Artificial Intelligence*, pages 239–296. Morgan Kaufmann, San Mateo, CA, 1988.

[23] M. L. Ginsberg. Multivalued Logics: A Uniform Approach to Reasoning in Artificial Intelligence. *Computational Intelligence*, 4:265–316, 1988.

[24] M. L. Ginsberg. The MVL Theorem Proving System. *SIGART Bulletin*, 2:57–60, 1991.

[25] M. L. Ginsberg. Personal communication, January 1993.

[26] M. L. Ginsberg. *User's Guide to the MVL System.* Computer Science Department, Stanford University, Stanford, CA, 1993.

[27] S. G. Grantham and L. H. Ungar. Qualitative Physics. In R. B. Banerji, editor, *Formal Techniques in Artificial Intelligence, A Source Book*, pages 77–121. Elsevier, Amsterdam, 1990.

[28] P. Hayes. Naive Physics 1: Ontology for Liquids. In J. R. Hobbs and R. C. Moore, editors, *Formal Theories of the Commonsense World*, pages 71–107. Ablex, Norwood, NJ, 1985.

[29] P. Hayes. The Second Naive Physics Manifesto. In J. R. Hobbs and R. C. Moore, editors, *Formal Theories of the Commonsense World*, pages 1–36. Ablex, Norwood, NJ, 1985.

[30] P. Hayes. The Naive Physics Manifesto. In M. A. Boden, editor, *The Philosophy of Artificial Intelligence*, pages 171–205. Oxford University Press, New York, NY, 1990.

[31] Y. Iwasaki and H. A. Simon. Causality in Device Behavior. *Artificial Intelligence*, 29:3–32, 1986.

[32] B. J. Kuipers. Commonsense Reasoning About Causality: Deriving Behavior from Structure. *Artificial Intelligence*, 24:169–203, 1984.

[33] B. J. Kuipers. The Limits of Qualitative Simulation. In *Proceedings of Ninth International Joint Conference on Artificial Intelligence*, pages 128–136. Morgan Kaufmann, San Mateo, CA, 1985.

[34] B. J. Kuipers. Qualitative Simulation. *Artificial Intelligence*, 29:289–338, 1986.

[35] B. J. Kuipers. Qualitative Simulation: Then and Now. *Artificial Intelligence*, 59:133–140, 1993.

[36] B. J. Kuipers and C. Chiu. Taming Intractible Branching in Qualitative Simulation. In D. S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning About Physical Systems*, pages 261–267. Morgan Kaufmann, San Mateo, CA, 1990.

[37] E. Sacks. Qualitative Mathematical Reasoning. In *Proceedings of Ninth International Joint Conference on Artificial Intelligence*, pages 137–139. Morgan Kaufmann, San Mateo, CA, 1985.

[38] M. Ü. Şencan and V. Akman. Qualitative Reasoning Experiments with the MVL Theorem Proving System. In K. Oflazer, V. Akman, H. A. Güvenir, and U. Halıcı, editors, *Proceedings of First Turkish Symposium on Artificial Intelligence and Artificial Neural Networks*, pages 29–36. Bilkent University, Ankara, 1992.

[39] S. C. Shapiro, editor. *The Encyclopedia of Artificial Intelligence,* Volume 2, pages 807–813. John Wiley, Chichester, UK, 1987.

[40] R. Simmons. "Commonsense" Arithmetic Reasoning. In D. S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning About Physical Systems*, pages 337–343. Morgan Kaufmann, San Mateo, CA, 1990.

[41] D. S. Weld. Comparative Analysis. *Artificial Intelligence*, 36:333–374, 1988.

[42] D. S. Weld and J. de Kleer, editors. *Readings in Qualitative Reasoning About Physical Systems.* Morgan Kaufmann, San Mateo, CA, 1990.

[43] B. C. Williams. Doing Time: Putting Qualitative Reasoning on Firmer Ground. In *Proceedings of Fifth National Conference on Artificial Intelligence*, pages 105–112. Morgan Kaufmann, San Mateo, CA, 1986.

[44] B. C. Williams and J. de Kleer. Qualitative Reasoning About Physical Systems: A Return to Roots. *Artificial Intelligence*, 51:1–9, 1991.