IMPLEMENTATION OF MOLECULAR COLLECTION THEORY

BY

VIJAY SRINIVASAN

A Research paper submitted to
Oregon State University

in partial fulfilment of the
requirements for the degree of
Master Of Science

August 10th 1989.

Acknowledgements

I would like to take this oppurtunity to express my heartfelt gratitude to Dr. Bruce D'Ambrosio for giving me the oppurtunity to work with him and providing me with a lot of help and guidance towards the completion of my degree. I would not have finished it had it not been for his unending patience and understanding especially at times when I faced difficulty at understanding the issues.

I would also like to express my sincere thanks to Dr. Tom Dietterich for agreeing to be my minor professor and being my Phd advisor. I have thoroughly enjoyed his courses and learned a lot from them. I am looking forward to working with him.

I am also very grateful to Prof. Ted Lewis for agreeing to be on my committee. I have taken a number of courses from him and have enjoyed every one of them.

I would like to thank my family members for their unbounded affection, constant support and encouragement. I would not have been to achieve much without the self confidence and courage that they have infused within me.

Last but not the least, I would like to express my thanks to the Computer Science Staff and other faculty members especially Dr. Walter Rudd and George Beekman for their help. I have learned a lot more than just Computer Science, here at OSU, Corvallis.

Contents

Abstract

Hayes, in his *Naive Physics Manifesto*, identified two alternate ontologies for reasoning about liquids, an ontology based on the notion of a contained substance and one based on the notion of a molecular collection. Qualitative Process theory, proposed by Forbus, lends itself easily to encoding the contained substance ontology. It does not, however, provide any mechanism to perform molecular collection reasoning. The primary objective of this research is to implement a mechanism for supporting molecular collection reasoning and evaluate its usefulness in various domains.

Introduction

The subfield of Qualitative Reasoning arose due to the disparity between how humans reason and act and the formalism introduced by the more rigid classical physics. Human beings with little or no mathematical background seem to gain an fairly good understanding of such concepts as mass, force, speed, etc. which seems to indicate that these are perceived in a qualitative rather than quantitative way. As humans are able to reason and react to the real world well, it may be believed that in order to build computers capable of doing the same we need to have a qualitative theory based on our mental model. An immediate gain of developing such a physics would be building expert systems equipped with commonsense models. (The lack of such a model in expert systems has been a major drawback as they have been unable to solve simpler versions of problems they were designed to solve and are hence brittle and inflexible [1].)

Developing models of the real world entails constructing a suitable ontology of the world and formalizing the 'commonsense' knowledge. The ontology defines the objects and their relationships. As objects in the real world are created and destroyed dynamically and at random, it is impossible to enumerate all the objects in a domain. It is therefore necessary to establish general criteria for individuating objects. Current research at formalizing the physical world has identified criterias for individuating solids and liquids [2].

Two alternative ontologies for individuating liquids have been constructed. They are:

- Contained Substance ontology.
- Molecular Collection ontology.

The Contained Substance ontology views a piece of liquid in a container as a generic liquid substance occupying a contained space enclosed by the container. The ontology does not refer to the liquid in the cup as an individual. Properties may be associated with the object modelled and these may be influenced by various processes. The result of viewing an object in this perspective is that the molecular identity of the object being modelled is not preserved. Hence, though the ontology identifies a new object coming into existence when a cup of coffee is poured into another cup, it treats a cup of coffee being emptied and refilled as essentially the same object. This shortcoming makes it impossible to reason with this ontology in situations where the molecular identity of the fluid needs to be preserved.

Molecular Collection ontology overcomes this shortcoming by considering a liquid as a particular collection of molecules. Reasoning about fluids proceeds by considering a small collection of molecules (of the fluid) and constructing its history as it participates with other objects in various processes.

Objective of this Project

There have been a number of programs developed to encode commonsense knowledge and reason about the real world. One such implementation, Qualitative Process Theory, proposed by Forbus[3], lends itself easily for encoding the Contained Substance ontology described above. It, however, does not support any mechanism to perform Molecular Collection reasoning.

The primary objective of this research is to implement a mechanism for supporting MC reasoning (based on Forbus & Collins [4] work) and evaluate its usefulness in various domains.

Ontologies For Modelling Fluid Flows

Two alternative ontologies exist for reasoning about fluids. They are: Contained Substance Ontology and Molecular Collection Ontology. The following sections discuss these ontologies in more detail and relates them to the modelling task.

2.1 The Contained Substance Ontology

A Contained Substance ontology is a means of individuating liquids without referring to pieces of liquids, but rather to containers which contain them. The liquid is viewed as a generic liquid substance occupying a contained space enclosed by the container. A contained space is a connected volume of three dimensional space which has a contiguous rigid boundary below and around it. The container is the solid object supporting the contained space, which is not a physical object but is characterised by a certain capacity and by being in a certain relation to a container. Figure 2.1 shows two containers connected by a conduit. A Contained Substance ontology views the liquid in the containers as a 'generic' water occupying the contained space enclosed by the containers.

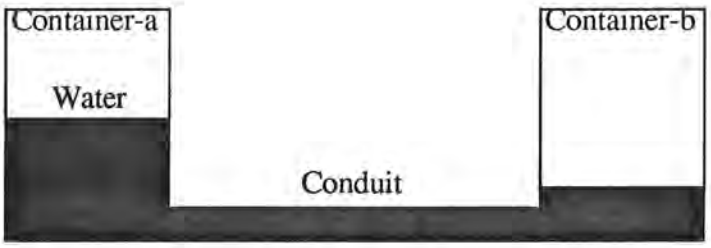


Figure 2.1: Two Containers Example

A Contained Substance ontology thus enables us to talk about a certain liquid in a container without having to say the same for each and every molecule of the liquid. Thus we can now say that the *Water in Container-a* is participating in a flow from Container-a to Container-b.

The Contained Substance ontology allows us to associate macroscopic properties to the liquid object under consideration. For example, we can associate 'amount-of' with a contained substance and talk about changes to 'amount-of' occurring due to interactions between the contained substance and other objects. (Associating 'amount-of' with a particular object, either a solid or a molecule (as in the MC perspective), does not seem to make much sense as it is always a constant. Because of the generic stuff part of the contained substance ontology we can consider 'amount-of' for that liquid object.) Macroscopic properties can be compared to detect possible interactions between contained substances. Thus we can now compare the pressures between the two contained substances of figure 2.1 and determine that a fluid flow process is possible through the conduit. This ability - to recognize interactions and predict how current interactions may change with time based on how the properties are changing - is very useful in describing behaviors of fluid systems.

The Contained Substance ontology, however, does not preserve the molecular identity of the liquid. Thus, though it enables us to describe behaviors of a system, it does not provide us with a mechanism which can describe the places and changes which a particular liquid molecule may be subjected to within a system. This ability requires tracing the molecule's history through the interactions in which it participates. The Molecular Collection ontology enables exactly this type of reasoning. This ability, however, requires that the current set of interactions be known already, as macroscopic properties cannot be associated with individual molecules. Thus the Molecular Collection ontology depends on

Contained Substance ontology to identify the current set of interaction in a system.

2.2 The Molecular Collection Ontology

Molecular Collection Ontology is a specialization of *Hayes'* Piece-of-stuff ontology and is another way to individuate liquids. Piece-of-stuff ontology is very rigid in that it considers a particular piece of stuff as an object (like a particular fixed collection of molecules say, M1, M2, M3 ...) and any physical change caused to this entity such as addition, splitting or cutting results in the destruction of the original entity and creation of new ones. For example, consider a glass containing 100cc of water. The 100cc of water is considered as a specific object. If we pour 10cc into another container, then the intial object vanishes and two new objects - a 90cc piece-of-water and a 10cc piece-of-water - appear. If we pour the 10cc piece back then we have the original 100cc object reappearing! (The 100cc object is however not destroyed if we transfer the entire 100cc into a different glass.) This disappearance and appearance of objects poses a problem as it is difficult to keep track of them and reason about them. Molecular Collection ontology is a specialization of the piece-of-stuff ontology in that it considers a piece-of-stuff so small that it never occupies more than one place at any time and hence cannot be subjected to such physical changes as splitting or cutting. (Thus the identity of the object is not lost unlike in the contained substance ontology or Piece-of-stuff ontology.) This tiny piece-of-stuff is viewed as a collection of molecules - as opposed to a single molecule - so that it may possess such macroscopic properties as pressure and temperature. The possibilities (transitions of substance, state and location) of the whole piece-of-stuff is constrained by considering this unit Molecular Collection.

Molecular Collection reasoning is a technique that traces the history of this arbitrary

collection of molecules (which can be considered as a single unit) through the various interactions in which the whole piece-of-stuff participates. Thus applying this reasoning on the system of figure 2.1 (two containers example) provides us the information that a MC unit will transit from container-a to the conduit and into container-b.

The macroscopic properties associated with an MC unit are primarily those associated with the contained substance view. The CS view identifies the overall changes occuring in the system and hence the values to the macroscopic properties. The values of the macroscopic properties for a MC unit will change according to the place and state transitions that it participates in, as it travels through the system. Hence it is now possible to answer questions that require molecular identity to be preserved i.e. changes that an MC unit undergoes in a system. This was not possible in the contained substance view due to the fact that an object ceases to exist once it changes place.

2.3 Relationship between CS and MC

Research in the area of Qualitative Reasoning has evolved a number of different theories and programs to model physical systems. Prominent among the theories is the Qualitative Process Theory propounded by Forbus[3]. (For a more detailed discussion of QP theory, refer appendix A.)

QP theory provides a convenient and flexible tool to model and reason about physical situations. It is based on Hayes' notion of histories (described in [2]) and Forbus' idea of processes to model real world situations. The basic idea is to construct the history of the objects in the system (a history is a description of the object through time as it participates in various interactions). A history may be generated at two levels of detail for a given

system. One predicts the future states and interactions of the objects in the system while the other traces the changes that a particular object is undergoing in the current situation. The first level of detail, that of identifying the current and future states of a physical system has been incorporated in the QP theory through the idea of processes (that identifies the current state), slices (which allow a projection of the object through time to be constructed) and limit analysis (to predict future states). At the second level of detail a MC reasoning is performed on the current state. Thus MC reasoning takes place within one episode of the CS history.

As explained in the previous chapter, Molecular Collection reasoning is a technique that traces the history of an arbitrary collection of molecules (which can be considered as a single unit) through the various interactions in which the whole piece-of-stuff participates. A MC unit is identified by three factors: the substance it is, the state it is in and the place it is in. MC history construction involves the determination of changes caused by the set of active processes to these factors and to the properties associated with the MC. This information i.e changes in properties of the MC, is provided by the modeller through a set of rules, each rule stating the process and the changes it causes to the derivatives of the properties.

(A detailed discussion of MC reasoning is provided in [4] and Appendix.)

We have implemented the MC reasoning discussed above as an extension to the OSU implementation of QP. The implementation is written in Lisp on Hybrid Uncertainity Manager (HUM, an ATMS based shell incorporating numerical methods for uncertainity management, developed at OSU by Dr. Bruce D'Ambrosio).

The next section contains details of some physical situations modelled and tested on the MC method.

Examples

In this section I shall illustrate, through a few examples, the task of modelling and generating the behavior of a system and the histories of the objects in it. The prediction of the behavior is enabled through QP and the history generation through MC reasoning.

A MC unit is modelled based on the substance it is, the state it is in and the place it is in. Properties are also associated with MC units. Modelling in the MC ontology must therefore specify changes occurring to each of these features due to the active processes in the system. This information is specified using rules having preconditions. The properties associated with an MC are, in general: Pressure, Temperature, Heat, Volume and Height. In the current implementation this information is specified in the form:

```
(defrule :name mc-properties
 :if ((:intern (Exists-in (mc ?substance liquid ?location) ?qpstate)))
 :then ((quantity-mc (heat (mc ?substance liquid ?location)))
        (quantity-mc (temp (mc ?substance liquid ?location)))
        (quantity-mc (pressure (mc ?substance liquid ?location)))
        (has-qp-value (pressure (mc ?substance liquid ?location)))
        (quantity-mc (volume (mc ?substance liquid ?location)))
        (quantity-mc (height (mc ?substance liquid ?location)))))
```

Properties that assume values from their QP counterparts (ex: Pressure, above) are indicated by the proposition (has-qp-value ....). (In such cases it means that the MC property acquires its value from the surrounding contained substance.) Modelling the other features described above (i.e property changes) will be illustrated in the following examples. Further details are listed in the appendix.

3.1 Two Containers Problem

The Two Containers problem consists of a two containers connected by a pipe. The containers contain unequal amounts of a liquid initially. The task is to model the system and generate the possible and consistent future states of the system. The system is illustrated pictorially below:

Figure 3.1: Two Containers Problem

3.1.1 QP Model

A representation of the system, in QP theory, contains structural information about the system, properties associated with the components of the system, and processes and views that are possible in the system. The processes and views encode the physics of the domain.

In the above system, the liquid in each of the containers has been modelled in QP as a *contained substance* of the form (C-S Substance State Place). Hence the water in Container-A is represented as (C-S Water Liquid Container-A). The containers and the conduit have water carrying/containing capability. The fluid flow process is modelled to be *possible* only if the two containers contain some liquid and there exists a fluid path between

the containers, and *active* if there is a pressure difference between the two containers. The fluid flow process in turn affects the amount of liquid in each of the container. Details of the actual model are given in the Appendix. The system is modelled to have a pressure gradient across the containers initially.

A run of QP on this model leads to the recognition of a state in which contained substance views are active in each of the containers and a fluid flow process is active in the system from Container-A to Container-B across the Conduit. QP also identifies the changes occurring to the macroscopic properties associated with the objects of the modelled system. The properties undergoing change are summarised below:

At Time T0:

Amount of water in Container-A is *positive* and *decreasing*.

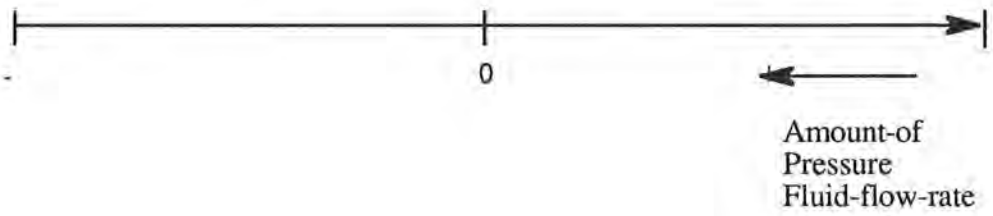
Pressure of water in Container-A is *positive* and *decreasing*.

Amount of water in Container-B is *positive* and *increasing*.

Pressure of water in Container-B is *positive* and *increasing*.

Fluid flow rate is *positive* and *decreasing*.

The above changes to the water in Container-A translate to the x-axis as follows:



Based on the qualitative mathematics used in QP theory, a further prediction of future states results in three possible next states: one in which the contained substances views are active and the fluid flow process continues to remain active; one in which the contained substances views are active but the fluid flow process is inactive; and a third state in which

the contained substance view in Container-A and the fluid flow process are inactive and the contained substance view in Container-B is active. (This third state is an invalid state as the contained substance views in Container-A and Container-B can never be inactive. This is, however, predicted as QP cannot determine which of pressure and flow rate, tends to 0 first.)

3.1.2 MC model

The MC model encodes information about the changes to a MC unit's properties by the various processes of the system. This information is provided in a rule form and can be classified into transition rules and Ds rules. Transition rules specify the changes made to either the substance, state or place of a MC unit while Ds rules specify the changes that the MC unit's properties are undergoing. Transition rules may be further classified according to the particular attribute they effect. Thus the implementation contains separate predicates such as Create-Substance, Transition-State and Transition-Place for representing changes to each of substance, state and place of the MC unit.

The MC model for the Two Containers Problem encodes the changes effected by the Fluid Flow Process. This process affects the 'Place' attribute and Pressure, Volume and Height of a MC unit. The corresponding MC rules for the Fluid Flow Process are shown below:

```
;;  Transition rules

(defrule :name fluid-flow-src-to-path
 :if ((:intern (status (at (fluid-flow ?mc1 ?mc2 ?src-cntr ?dst-cntr ?path) (state ?s)) active))
     (:intern (Exists-in ?mc1 (state ?s))))
 :then ((Transition-Place ?mc1 ?path (state ?s))))
```

```
(defrule :name fluid-flow-path-to-dst
 :if ((:intern (status (at (fluid-flow ?mc1 (mc ?substance ?state ?dst)
                                    ?src-cntr ?dst-cntr ?path) ?qpstate) active))
     (:intern (Exists-in (mc ?substance ?state ?path) ?qpstate)))
 :then ((Transition-Place (mc ?substance ?state ?path) ?dst ?qpstate)))

;;  DS rules.

(defrule :name ds-rules-fluid-flow
 :if ((:intern (status (at (fluid-flow ?mc1 ?mc2 ?src-cntr ?dst-cntr ?path) (state ?s)) active))
    (:intern (Exists-in ?mc1 (state ?s)))
    (:intern (Exists-in ?mc2 (state ?s))))
 :then ((ds (height ?mc1) -1)
        (ds (height ?mc2) 1)))
```

A run of MC on State 1 yields the following information. The implementation sketches MC's history and the changes to properties (as derivatives). A summary of the latter is shown here for better readability. The actual outputs are shown in detail in the Appendix.

MC's HISTORY :

(PLACE (MC WATER LIQUID CONTAINER-A) (STATE 1))
(NEXT-PLACE (MC WATER LIQUID CONTAINER-A) (MC WATER LIQUID CONDUIT) (STATE 1))
(NEXT-PLACE (MC WATER LIQUID CONDUIT) (MC WATER LIQUID CONTAINER-B) (STATE 1))

MC's DS VALUES :

ds values for (MC WATER LIQUID CONTAINER-A) are:
(HEIGHT (MC WATER LIQUID CONTAINER-A)) is *decreasing*.
(PRESSURE (MC WATER LIQUID CONTAINER-A)) is *decreasing*.

ds values for (MC WATER LIQUID CONDUIT) are:
(PRESSURE (MC WATER LIQUID CONDUIT)) is *increasing/steady/decreasing*.

ds values for (MC WATER LIQUID CONTAINER-B) are:

(HEIGHT (MC WATER LIQUID CONTAINER-B)) is *increasing*.

(PRESSURE (MC WATER LIQUID CONTAINER-B)) is *increasing*.

The pressure of the MC unit in the conduit is unknown and is hence represented as an ambiguous value. Note that this is quite different from the value *'steady'*.

Having done the MC reasoning we can now explain in more detail (than was possible before) about what is actually is going on in the system. The water in Container-A comes down into the conduit, passes through it and up into Container-B. This information was not available before in the CS reasoning.

3.2 The Furnace Problem

This system consists of a furnace with an inlet (bin), and two outlet (off-gas-chamber and slag outlet). Raw materials enter into the furnace via the bin, undergoes warming up as they descend in the furnace. Reaction occurs when the raw materials reach reaction temperature in the reaction zone. The products of the reaction are slag and off-gas which are both at a higher temperature than the raw materials above the reaction zone. As the off-gas moves up the furnace into the off-gas-chamber, it warms the incoming raw material. Heating is accomplished by passing a current across carbon electrodes that are inserted into the raw material in the furnace, from the top and by the slag below the reaction zone. The task is to model the furnace with all its components and processes and generate the QP and MC history for the system. A pictorial representation of the system is shown below followed by a discussion of the QP and MC models.

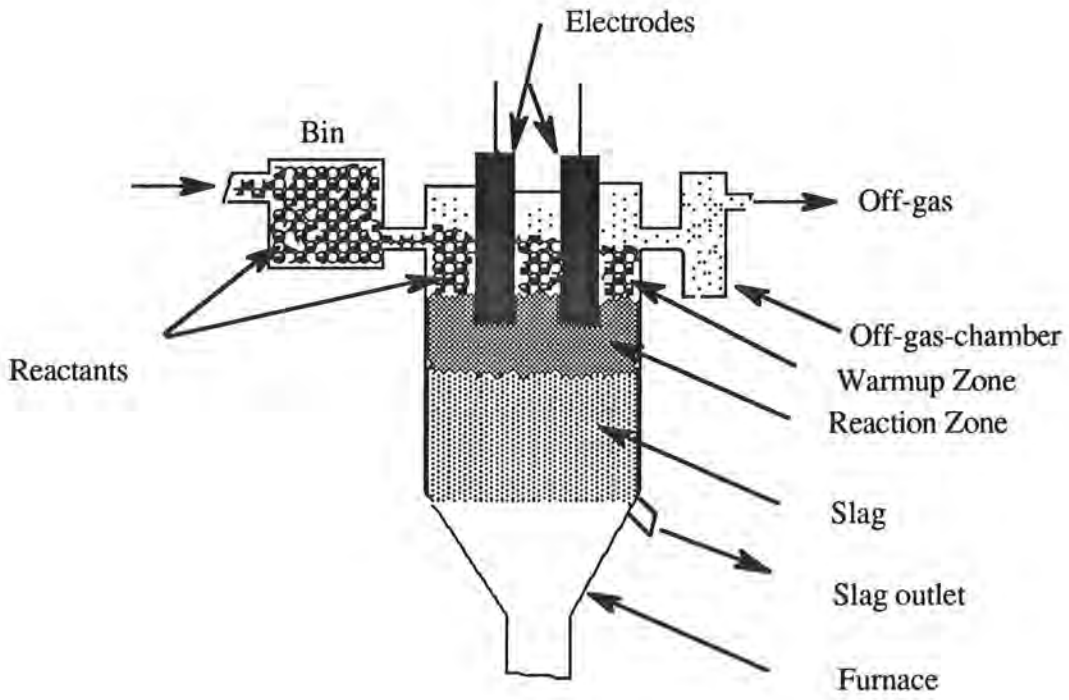


Figure 3.2: Furnace Example

3.2.1 QP and MC Models

A model of the above system has been developed in QP and MC. Details of the modelling are listed in the Appendix. The various processes and views modelled include the following: Material Flow Process, Heat Flow Process, Reaction Process, Contained Substance View and Counter Current Heat Flow View. In the steady state the following processes are identified by QP:

- Material Flows:
  - Reactants
    - Bin to Furnace-Warmup-zone.
    - Furnace-Warmup-zone to Furnace-Reaction-zone.
  - Off-gas
    - Furnace-Reaction-zone to Furnace-Warmup-zone.
- Reaction Process

- Heat Flows:
  - Heater to Reactants in Furnace-Reaction-zone.
  - Off-gas to Reactants in Furnace-Warmup-zone.

Resolving the changes to the properties of the contained susbstances in the system at steady state indicates that derivatives for amount-of, pressure and heat are all 0 for all contained substances as they all gain and lose equal amounts of heat.

For this steady state, MC reasoning generates the history shown below.

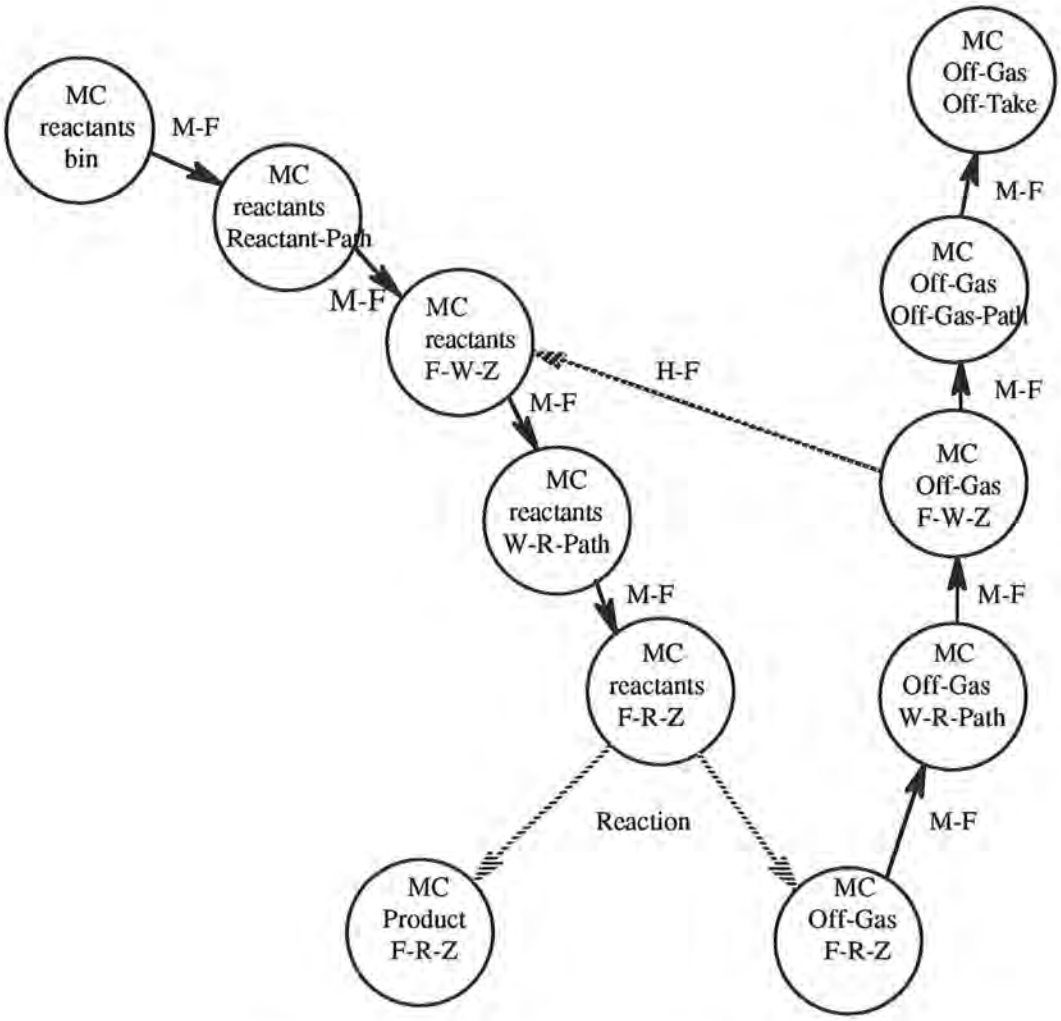


Figure 3.3: MC History for the Furnace Problem

The information generated by the MC reasoning can be summarized as follows:

`The Place of the MC unit changes from Bin to the reaction-zone during the material flow process and the MC unit gains in heat during this process. Also the MC unit representing the Off-gas decreases in heat as it participates in the heat flow process in the Warmup Zone.'

3.3 Heated Conduit Problem

The Heated Conduit problem is very similar to the Two Containers Problem except that the Conduit is heated by a heat source. A figure of the system is shown below along with a summary of the QP and MC models.

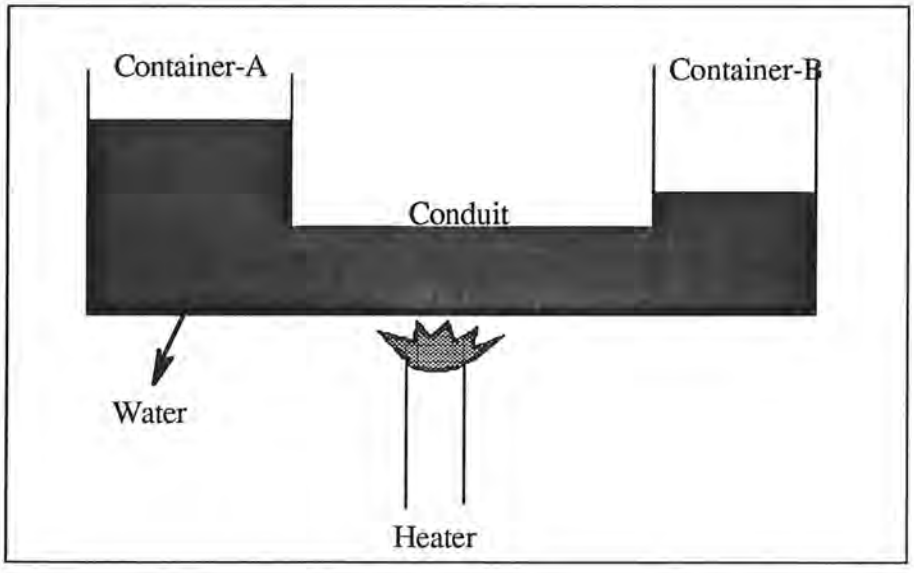


Figure 3.4: Heated Conduit Problem

3.3.1 QP and MC models

One of the ways to model this system in QP would require that the water in the Conduit be modelled as a contained substance which then participates in a heat flow process. This, however, has the disadvantage that QP will indicate that the temperature of the contained substance is increasing due to the heat flow, which would lead the water in the conduit to boil which will be inconsistent with the behavior observed in the system because the heated water also participates in the fluid flow process. Hence this would be an inconsistent model. A better way to model the system would be to encode the fact that the water flowing into the conduit contains a certain amount of heat which increases with the heat flow process. This view would indicate that the temperature of water in the conduit is a constant as the total gain in heat in the conduit is equal to the lose of heat that goes out of the conduit with the water. This is, infact, a consistent observation in the system.

Modelling the system in MC ontology indicates, however, that the heat of the water actually increases due to the fact that the MC unit passes through the Conduit where it gains heat from the heater.

3.4 Discussion

3.4.1 Why MC reasoning

From the above examples it is evident that MC reasoning provides additional information which cannot be derived through CS reasoning. While CS reasoning identified the possible states of the system and the changes that the contained substances were undergoing in the states, it did not identify the changes that the piece-of-stuffs in the system were being subjected to. The difference between the two types of information is

subtle. CS ontology enables a global view of the system and hence provides only global, 'aggregated' information like: 'pressure in Container-A is decreasing'. MC ontology takes a local view of the changes in the system and hence explicates information about the changes that a piece-of-stuff in the system is undergoing. Hence though the overall effect of a process may be Zero, its effect on a single piece-of-stuff may not be Zero.

Thus MC reasoning is essential to reasoning about fluids as it provides more information by preserving molecular identity.

3.4.2 Problems encountered

The main problems encountered in the project were that of modelling in MC ontology and the choice of problems to test. Modelling in MC ontology, presently, requires that the designer determine the set of properties that may be of interest in the objects being modelled and the set of changes that the properties will undergo in each of the active processes in the system. The models seem ad hoc sometimes and the lack of a facility to check for consistency with the QP models adds to problem. Research in this area has tried to derive some of the MC rules from the QP models. The second problem mentioned above, namely, that of selecting interesting problems, is probably a problem in any venture that involves testing theories about the real world. Without a dense cluster of formalisations of the real world, it is difficult to determine the boundaries of applicability of the theories that are being tested. In other words, we need to some tools to identify potentially interesting physical situations. For example, the furnace example described above, though large and complex, was uninformative (probably due to sheer complexity!) and difficult to model.

Future Research

Qualitative Physics is, basically, aimed at modelling physical systems and reasoning about them the way humans beings do. MC reasoning is a particular way of reasoning about liquids that preserves molecular identity. The types of questions that can be answered by an MC reasoning (which cannot be answered by QP) are usually of the type:

1) Where did the water in Container-B come from?
2) Did the water from Container-A gain in temperature?
3) Is there a heat transfer in the system (ex: refridgerator)?

Analysing just what types of questions MC reasoning answers will provide scope on where this type of reasoning may be more beneficial.

The difficult task of modelling a physical system in MC ontology raises the issue of new methodologies and techniques to help in the task of modelling physical systems.

Also, the varying levels of details at which humans beings reason may be indicative of other problem solving areas where a MC type of reasoning may be useful.

MC reasoning may also be applied to systems involving currents. It is our intuition that MC reasoning may be used to model heat flows and momentum currents where an object is considered as a entity capable of carrying a certain amount of fluid 'heat' or fluid 'momentum'. MC reasoning may also be extended to cases where multiple MC units (representing different piece-of-stuffs) may be simultaneously undergoing change in a single state of the system.

References

1. De Kleer, J. and Brown, J.S., A Qualitative Physics based on Confluences, *Artificial Intelligence* **24** (1984).
2. Hayes, P.J., The Naive Physics Manifesto, in: D. Michie (Ed.), *Expert Systems in the MicroElectronic Age* (Edinburgh University Press, Edinburgh, 1979).
3. Forbus, K.D., Qualitative Process Theory, *Artificial Intelligence* **24** (1984).
4. Forbus, K.D., and Collins, J.W., Reasoning About Fluids Via Molecular Collections, *Proceedings of AAAI 87*, (Vol 2), (1987).
5. Bobrow, D.G., (Ed.), *Qualitative Reasoning About Physical Systems*, MIT Press, (1985).

Appendix

Construction of MC Reasoning

Constructing the MC history occurs in three steps. The first step is to generate the total envisionment for the given system using the domain knowledge and the Qualitative Process Engine (QPE). QPE generates all consistent situations connected by the possible transitions between them. For each of the consistent situations it determines the active processes and views and indicates the direction of change for the properties of the objects.

Next, a single situation is selected for which the MC history is desired. In order for the history to be meaningful and interesting, the situation should involve some active processes. MC reasoning begins with determining the possible locations and states of the MC. (These are interned into the ATMS by detecting their QP counterparts.) The active processes in the situation are also determined from the output of the QPE and are interned into the ATMS in an appropriate MC format. These are then used along with the MC domain rules (transition rules) to establish how the locations and states of the MC can change. Each process (transition rule) specifies a fragment of the MC's history. A graph of the transitions is output.

In the final step, the Ds values for the MC's properties are computed. Some of the values are determined from the output of influence resolution of the QPE while the others need to provided by the user through the MC domain rules. Ambiguities which cannot be resolved are indicated appropriately.

Thus with the above information it is possible to recognize such phenomena as branching

or cycles of flow. Note that a single state in the envisionment can give rise to a number of episodes inthe MC history.

Modelling in MC

The following is a description of the MC implementation that has been developed.

At the surface, MC reasoning involves modelling a system in QP theory, determining the active processes in the system and resolving the changes to the properties of the objects in the system, modelling the system in MC theory, generating the MC history and resolving the changes to MC's properties for one of the QP states. The following treatment outlines the latter three steps listed above.

A MC unit is represented as a 3 slot list of the form (MC Substance State Place). Modelling a system in MC terms requires determining the set of properties of the MC such as Temperature, Pressure, Place etc., that are of interest and providing information (in a rule form) as to how these are being varied by the active processes in the QP state of interest in the system.

Modelling the properties of a MC unit in the implementation takes the form shown below:

```
(defrule :name mc-properties
 :if ((:intern (Exists-in (mc ?substance liquid ?location) ?qpstate)))
 :then ((quantity-mc (heat (mc ?substance liquid ?location)))
        (quantity-mc (temp (mc ?substance liquid ?location)))
        (quantity-mc (pressure (mc ?substance liquid ?location)))
        (has-qp-value (pressure (mc ?substance liquid ?location)))
        (quantity-mc (volume (mc ?substance liquid ?location)))
        (quantity-mc (height (mc ?substance liquid ?location)))))
```

As a MC unit is considered to be a fixed collection of molecules, it does not therefore have the 'amount-of' property. The properties of the MC may be macroscopic or microscopic. Macroscopic properties obtain their values form the contained substance counterparts or from explicit statements about the set of active processes in the system. The (has-qp-value ...) proposition informs the MC engine (MCE) that the value for that property is obtained form QP's influence resolution phase. Properties that do not obtain their values form the QP objects are modelled explicitly in a rule form. Each rule specifies a process and the changes it causes to the properties (possibly, a subset) of the MC unit. Such rules generally fall into one of two types: Transition rules and Ds rules. Transition rules specify changes to the 3 attributes based on which the MC unit is identified, i.e., Substance, State and Place. Ds rules specify changes made to the properties of the MC. An example of each is illustrated below. (Ds values to the properties must be fully engineered.)

```
;;  Transition rules

(defrule :name fluid-flow-src-to-path
 :if ((:intern (status (at (fluid-flow ?mc1 ?mc2 ?src-cntr ?dst-cntr ?path) (state ?s)) active))
    (:intern (Exists-in ?mc1 (state ?s))))
 :then ((Transition-Place ?mc1 ?path (state ?s))))

;;  DS rules.

(defrule :name ds-rules-fluid-flow
 :if ((:intern (status (at (fluid-flow ?mc1 ?mc2 ?src-cntr ?dst-cntr ?path) (state ?s)) active))
    (:intern (Exists-in ?mc1 (state ?s)))
    (:intern (Exists-in ?mc2 (state ?s))))
 :then ((ds (height ?mc1) -1)
        (ds (height ?mc2) 1)))
```

As may be evident from the above examples, the preconditions of the rules have to carefully engineered so as not to contradict with the QP reasoning, nor should they be triggered before the actual MC reasoning is done. For such reasons, the rules represent the processes in a slightly differently than QP does. In order to distinguish between the representations for a time slice in QP and MC, the latter explicitly represents slices in the form of a (State ?s) list. The '?s' corresponds to a QP state thus validating the fact that MC reasoning is performed on a QP state. Thus, for example the precondition of a rule in MC for an active Fluid-flow process takes the form,

(status (at (fluid-flow ?mc1 ?mc2 ?src-cntr ?dst-cntr ?path) (state ?s)) active)

Sometimes, the structure of the MC unit may also be used to distinguish between the QP and MC rules. For example the same rule may equivalently be written in the form:

(status (at (fluid-flow (mc ?substance ?state ?place) ?mc2 ?src-cntr ?dst-cntr ?path)
?qpstate) active)

This has the added advantage that it not only distinguishes itself from QP rules, but also provides access to information (within the body of the rule) to the substance, state and place of the MC unit.

Having modelled the system in MC ontology, the MC reasoning procedure is applied on the model by calling the function 'mc-analysis' with the desired state as the parameter. This in turn generates the history and resolves the Ds values for the MC unit. A complete example with the QP and MC models and output of a run is shown below for convenience. The code for the MC reasoning is also provided at the end of this section.

The Two Containers Problem - QP Model

```
(defProcess (fluid-flow ?src ?dst ?src-cntr ?dst-cntr ?path)
 :Individuals (
  (?src (Contained-substance ?src))
  (?src-cntr (contains ?src-cntr ?src))
  (?dst (Contained-substance ?dst))
  (?dst-cntr (contains ?dst-cntr ?dst))
  (?path (fluid-path ?path) (fluid-Connected ?src-cntr ?dst-cntr ?path)))
 :Conditions (
  ((A (pressure ?src)) greater-than (A (pressure ?dst))))
 :Relations (
  (Local flow-rate (Quantity flow-rate))
  (flow-rate Q= (- (pressure ?src) (pressure ?dst))))
 :Influences (
  ((amount-of ?dst) I+ flow-rate)
  ((amount-of ?src) I- flow-rate)))
```

```
(defscenario two-containers
 :Individuals (
  Container-a Container-b Conduit
  (c-s water liquid container-a)
  (c-s water liquid container-b))
 :Facts (
  (Exists-always Container-a)
  (Exists-always Container-b)
  (Exists-always Conduit)
  (Exists-always Water)
  (substance water)
  (substance heat)
  (state heat gas)
  (state water liquid))
 :Always (
  (container Container-a)
  (container Container-b)
  (fluid-path Conduit)
  (fluid-connected Container-a Container-b Conduit)
  (can-contain-substance Container-a water)
  (can-contain-substance Container-b water)
  (aligned Container-a Container-b Conduit)
  (aligned Container-b Container-a Conduit))
 :In-Situation (T0
  ((A (amount-of-in water Container-a)) greater-than zero)
  ((A (amount-of-in water Container-b)) greater-than zero)
  ((A (Pressure (c-s water liquid Container-a))) Greater-than
   (A (Pressure (c-s water liquid Container-b))))))
```

The Two Containers Problem - MC Model

```
;;  Transition rules

(defrule :name fluid-flow-src-to-path
 :if ((:intern (status (at (fluid-flow ?mc1 ?mc2 ?src-cntr ?dst-cntr ?path) (state ?s)) active))
    (:intern (Exists-in ?mc1 (state ?s))))
 :then ((Transition-Place ?mc1 ?path (state ?s))))

(defrule :name fluid-flow-path-to-dst
 :if ((:intern (status (at (fluid-flow ?mc1 (mc ?substance ?state ?dst)
                                  ?src-cntr ?dst-cntr ?path) ?qpstate) active))
     (:intern (Exists-in (mc ?substance ?state ?path) ?qpstate)))
 :then ((Transition-Place (mc ?substance ?state ?path) ?dst ?qpstate)))

;;  DS rules.

(defrule :name ds-rules-fluid-flow
 :if ((:intern (status (at (fluid-flow ?mc1 ?mc2 ?src-cntr ?dst-cntr ?path) (state ?s)) active))
    (:intern (Exists-in ?mc1 (state ?s)))
    (:intern (Exists-in ?mc2 (state ?s))))
 :then ((ds (height ?mc1) -1)
        (ds (height ?mc2) 1)))
```

Load the above files along with the files mc.cl, quantities.cl, qp.cl, and transition-rules.cl.

A run of QP on this model produces the following states:

```
> (elaborate)
nil
> (determine-vps)

"Solution: "
A5 (STATUS (AT (FLUID-FLOW (C-S WATER LIQUID CONTAINER-A) (C-S WATER LIQUID
CONTAINER-B) CONTAINER-A CONTAINER-B CONDUIT) T0) ACTIVE):  TAXONOMY
```

A3 (STATUS (AT (CONTAINED-SUBSTANCE WATER CONTAINER-A) T0) ACTIVE): TAXONOMY
A1 (STATUS (AT (CONTAINED-SUBSTANCE WATER CONTAINER-B) T0) ACTIVE): TAXONOMY
" "
(STATE 1)
T
> (influence-res '(state 1))

"Solution: "
" "
(AMOUNT-OF (AT (C-S WATER LIQUID CONTAINER-B) T0)): (+) / (+)
(AMOUNT-OF-IN WATER (AT CONTAINER-B T0)): (+) / (+)
(AMOUNT-OF (AT (C-S WATER LIQUID CONTAINER-A) T0)): (+) / (-)
(AMOUNT-OF-IN WATER (AT CONTAINER-A T0)): (+) / (-)
(HEAT (AT (C-S WATER LIQUID CONTAINER-B) T0)): (+) / (0)
(PRESSURE (AT (C-S WATER LIQUID CONTAINER-B) T0)): (+) / (+)
(TEMP (AT (C-S WATER LIQUID CONTAINER-B) T0)): (+) / (0)
(HEAT (AT (C-S WATER LIQUID CONTAINER-A) T0)): (+) / (0)
(PRESSURE (AT (C-S WATER LIQUID CONTAINER-A) T0)): (+) / (-)
(TEMP (AT (C-S WATER LIQUID CONTAINER-A) T0)): (+) / (0)
(FLOW-RATE (AT (FLUID-FLOW (C-S WATER LIQUID CONTAINER-A) (C-S WATER LIQUID CONTAINER-B) CONTAINER-A CONTAINER-B CONDUIT) T0)): (+) / (-)

A run of MC on State 1 yields the following information.

> (mc-analysis '(state 1))

MC's HISTORY :

(PLACE (MC WATER LIQUID CONTAINER-A) (STATE 1))
(NEXT-PLACE (MC WATER LIQUID CONTAINER-A) (MC WATER LIQUID CONDUIT) (STATE 1))
(NEXT-PLACE (MC WATER LIQUID CONDUIT) (MC WATER LIQUID CONTAINER-B) (STATE 1))

MC's DS VALUES :

ds values for (MC WATER LIQUID CONTAINER-A) are:

(DS (HEIGHT (MC WATER LIQUID CONTAINER-A)) -1)

(DS (VOLUME (MC WATER LIQUID CONTAINER-A)) 0)

(DS (PRESSURE (MC WATER LIQUID CONTAINER-A)) -1)

(DS (TEMP (MC WATER LIQUID CONTAINER-A)) 0)

(DS (HEAT (MC WATER LIQUID CONTAINER-A)) 0)

ds values for (MC WATER LIQUID CONDUIT) are:

(DS (HEIGHT (MC WATER LIQUID CONDUIT)) 0)

(DS (VOLUME (MC WATER LIQUID CONDUIT)) 0)

(DS (PRESSURE (MC WATER LIQUID CONDUIT)) (-1 0 1))

(DS (TEMP (MC WATER LIQUID CONDUIT)) 0)

(DS (HEAT (MC WATER LIQUID CONDUIT)) 0)

ds values for (MC WATER LIQUID CONTAINER-B) are:

(DS (HEIGHT (MC WATER LIQUID CONTAINER-B)) 1)

(DS (VOLUME (MC WATER LIQUID CONTAINER-B)) 0)

(DS (PRESSURE (MC WATER LIQUID CONTAINER-B)) 1)

(DS (TEMP (MC WATER LIQUID CONTAINER-B)) 0)

(DS (HEAT (MC WATER LIQUID CONTAINER-B)) 0)

The files mc.cl and transition-rules.cl are listed below:

mc.cl

```
;;; -*- Package: Common Lisp;
;;;    date:        sept. 3rd, 1988;
;;;    creator:     Vijay Srinivasan.
;;;    modified:    Oct. 6th 1988;
;;;
;;; file: mc.cl
;;;
;;; This file contains the code that performs the MC analysis.
;;; The file is to loaded after loading HUM and QP.
;;;
;;; Oct 6th, 1988. Modifications : (exists-in (mc ...) ?t) is no longer
;;; interned during mc-analysis phase. Previously mc-analysis used to intern
;;; the above prop for every corresponding (exists-in (c-s ..) ?t) of qp
;;; BEFORE building the actualhistory. Now (exists-in (mc..) ?t) is interned
;;; only for the starting placeof mc history by a rule triggered by
;;; (place (mc...) ?t).


(defmacro printdot ()
  (list 'format 't "."))

(defvar *MC-STATE* nil)

(defun mc-analysis (state-prop
        &aux (time (fifth (rval-value
                (lookup `(time-of state ,(second state-prop) is ?x))))))
  (setq *MC-STATE* (lookup state-prop))
  (intern-active-processes state-prop time)
  (generate-mc-history state-prop)
  (resolve-ds-values state-prop))
```

```
(defun mc-analysis1 (state-prop
            &aux (time (fifth (rval-value
                (lookup `(time-of state ,(second state-prop) is ?x))))))
 (setq *MC-STATE* (lookup state-prop))
 (intern-active-processes state-prop time))


(printdot)


(defun intern-active-processes (state-prop time &aux process-bindings process rval
                                                bindings continue)
 (multiple-value-bind (rval bindings continue)
   (lookups `(status (at (*var* . process-bindings) ,time) active) nil)
   (loop
    (if (null rval) (return t))
    (if (env-superset-of-label-p
  (car (rval-label *MC-STATE*))
  (rval-label (car rval)))
  (progn
   (setq process-bindings (cadr bindings))
   (setq process (subst 'mc 'c-s process-bindings))
   (->-1 `((status (at ,process-bindings ,time) active))
         `((status (at ,process ,state-prop) active)))))
    (multiple-value-setq (rval bindings)
  (funcall continue)))))

(printdot)

(defun generate-mc-history (state-prop)
 (format t "~%MC's HISTORY :~%")
 (run-rules))
```

```
(defun show-history (list)
 (format t "~%~s" list))


(defun mc-quantities (substance state location &aux quantities)
 (setq quantities nil)
 (multiple-value-bind (r b c)
    (lookups `(quantity-mc ((*var* . prop-name) (mc ,substance ,state ,location))) nil)
    (loop
    (if (null r) (return quantities))
    (setq prop-name (cadr b))
    (push (list prop-name `(mc ,substance ,state ,location)) quantities)
    (multiple-value-setq (r b) (funcall c)))))

(printdot)

(defun get-ds-values (state-prop env substance state location quantities
            &aux (time (fifth (rval-value
                    (lookup `(time-of state ,(second state-prop) is ?x))))) ds-list)
 (progn
  (dolist (quantity quantities)
  (multiple-value-bind (rval binds)
    (lookup `(ds ,quantity (*var* . sign)))
    (cond ((null rval)
                (push (get-qp-d-value env time quantity substance state location) ds-list))
           ((env-superset-of-label-p env (rval-label rval))
            (progn
                (setq sign (cadr binds))
                (push `(ds ,quantity ,sign) ds-list))))))
  ds-list))

(printdot)


(defun resolve-ds-values (state-prop &aux (env (car (rval-label *MC-STATE*)))
                quantities substance state location ds-values)
```

```
 (format t "~%~%MC's DS VALUES :~%")
 (multiple-value-bind (rval bindings continue)
   (lookups
    `(exists-in (mc (*var* . substance) (*var* . state) (*var* . location)) ,state-prop) nil)
   (loop
    (cond ((null rval)
    (progn
       (format t "~%******** end of mc analysis. ********~%")
       (return t)))
   (t
    (progn
       (setq substance (getf bindings 'substance))
       (setq state (getf bindings 'state))
       (setq location (getf bindings 'location))
       (setq quantities (mc-quantities substance state location))
       (setq ds-values (get-ds-values state-prop env substance state location quantities))
       (print-ds-values substance state location ds-values))))
    (multiple-value-setq (rval bindings) (funcall continue)))))

(printdot)

(defun print-ds-values (substance state location ds-values)
 (progn
  (format t "~% ds values for (MC ~s ~s ~s) are:~%" substance state location)
  (dolist (ds-value ds-values)
  (format t "     ~s ~%" ds-value))))

(printdot)

(defun get-qp-d-value (env time quantity substance state location &aux relation)
 (cond ((lookup `(has-qp-value ,quantity))
  (multiple-value-bind (rval bindings continue)
   (lookups `((D (,(car quantity) (at (c-s ,substance ,state ,location) ,time)))
              (*var* . relation) Zero) nil)
   (loop
    (cond ((null rval)
```

```
                (multiple-value-bind (rval1 bindings1)
                        (lookup `(Zero greater-than
                                (D (,(car quantity) (at (c-s ,substance ,state ,location) ,time)))))
                    (if (and rval1 (env-superset-of-label-p env (rval-label rval1)))
                        (return `(ds ,quantity -1))
                        (return `(ds ,quantity (-1 0 1))))))
            ((env-superset-of-label-p env (rval-label (car rval)))
             (if (equal (getf bindings 'relation) 'greater-than)
                 (return `(ds ,quantity 1))
                 (return `(ds ,quantity 0)))))
    (multiple-value-setq (rval bindings) (funcall continue)))))
(t `(ds ,quantity 0))))
```

Transition-rules.cl:

```
;;; -*-  Mode: LISP;  Syntax: Common Lisp;  Package: ATMS;  -*-
;;;
;;; -*-  date:              Sept. 5th, 1988.
;;; -*-  file:              Transition-rules.cl
;;; -*-  Creator:           Vijay Srinivasan
;;; -*-  last modified:     26th October 1988.
;;;
;;;        This file contains the intermediate rules that intern propositions with their
;;;        appropriate transistions substituted.
;;;



(defrule :name intern-exists-from-place
 :if ((:intern (place (mc ?substance ?state ?place) (state ?s))))
 :then ((exists-in (mc ?substance ?state ?place) (state ?s))))
```

```
(defrule :name Transition-Place
 :if ((:intern (Transition-Place (mc ?substance ?state ?Place1) ?Place2 ?qpstate)))
 :then ((Exists-in (mc ?substance ?state ?Place2) ?qpstate)))


(defrule :name Transition-State
 :if ((:intern (Transition-State (mc ?substance ?state ?Place) ?Newstate ?qpstate)))
 :then ((Exists-in (mc ?substance ?Newstate ?Place) ?qpstate)))


(defrule :name Create-substance
 :if ((:intern (Create-substance (mc ?substance ?state ?place)
                                 ?newsubstance ?newstate ?qpstate)))
 :then ((Exists-in (mc ?newsubstance ?newstate ?place) ?qpstate)))


(defrule :name history-generation1
 :if ((:intern (Place (mc ?substance ?state ?Place1) ?qpstate))
    (:intern (Transition-Place (mc ?substance ?state ?Place1) ?Place2 ?qpstate)))
 :then ((Next-Place (mc ?substance ?state ?Place1) (mc ?substance ?state ?Place2)
?qpstate))
 :do ((show-history '(Place (mc ?substance ?state ?Place1) ?qpstate))
    (show-history '(Next-Place (mc ?substance ?state ?Place1)
                               (mc ?substance ?state ?Place2) ?qpstate))))


(defrule :name history-generation2
 :if ((:intern (Next-Place (mc ?substance ?state ?Place1) ?Place2 ?qpstate))
    (:intern (Transition-Place ?Place2 ?Place3 ?qpstate)))
 :then ((Next-Place ?Place2 (mc ?substance ?state ?Place3) ?qpstate))
 :do ((show-history '(Next-Place ?Place2 (mc ?substance ?state ?Place3) ?qpstate))))
```

```
(defrule :name history-generation3.1
 :if ((:intern (Place (mc ?substance ?state1 ?Place) ?qpstate))
    (:intern (Transition-State (mc ?substance ?State1 ?Place) ?State2 ?qpstate)))
 :then ((Place (mc ?substance ?State2 ?Place) ?qpstate)
        (Next-State (mc ?substance ?State1 ?Place) (mc ?substance ?State2 ?Place)
                                                ?qpstate))
 :do ((show-history '(Place (mc ?substance ?State2 ?Place) ?qpstate))
    (show-history '(Next-State (mc ?substance ?state1 ?Place)
                                (mc ?substance ?state2 ?Place) ?qpstate))))

(defrule :name history-generation3.2
 :if ((:intern (Next-Place (mc ?substance ?state ?Place1) (mc ?substance ?state ?Place2)
                                                ?qpstate))
    (:intern (Transition-State (mc ?substance ?state ?Place2) ?newstate ?qpstate)))
 :then ((Place (mc ?substance ?newstate ?Place2) ?qpstate)
        (Next-State (mc ?substance ?State ?Place2) (mc ?substance ?newstate ?Place2)
                                                ?qpstate))
 :do ((show-history '(Next-State (mc ?substance ?state1 ?Place)
                                (mc ?substance ?state2 ?Place) ?qpstate))))


(defrule :name history-generation4.1
 :if ((:intern (Place (mc ?substance ?state ?Place) ?qpstate))
    (:intern (Create-Substance (mc ?substance ?State ?Place) ?newsubstance ?newstate
                                                ?qpstate)))
 :then ((Place (mc ?newsubstance ?newstate ?Place) ?qpstate)
        (New-Substance (mc ?substance ?State ?Place) (mc ?newsubstance ?newstate
                                                ?Place) ?qpstate))
 :do ((show-history '(New-Substance (mc ?substance ?state ?Place)
                                (mc ?newsubstance ?newstate ?Place) ?qpstate))))


(defrule :name history-generation4.2
 :if ((:intern (Next-Place (mc ?substance ?state ?Place1) (mc ?substance ?state ?Place2)
                                                ?qpstate))
    (:intern (Create-Substance (mc ?substance ?state ?Place2) ?newsubstance ?newstate
```

?qpstate)))
:then ((Place (mc ?newsubstance ?newstate ?Place2) ?qpstate)
(New-Substance (mc ?substance ?State ?Place2) (mc ?newsubstance ?newstate
?Place2) ?qpstate))
:do ((show-history '(New-Substance (mc ?substance ?state ?Place2)
(mc ?newsubstance ?newstate ?Place2) ?qpstate))))

END