Technical Report 793

Switching Between Discrete and Continuous Process Models To Predict Genetic Activity

Daniel Sabey Weld

MIT Artificial Intelligence Laboratory

This blank page was inserted to preserve pagination.

SWITCHING BETWEEN DISCRETE AND CONTINUOUS PROCESS MODELS TO PREDICT MOLECULAR GENETIC ACTIVITY

by

Daniel Sabey Weld

ABSTRACT

Two kinds of process models have been used in programs that reason about change: discrete and continuous models. We describe the design and implementation of a qualitative simulator, PEPTIDE, which uses both kinds of process models to predict the behavior of molecular genetic systems. The program uses a discrete process model to simulate both situations involving abrupt changes in quantities and the actions of small numbers of molecules. It uses a continuous process model to predict gradual changes in quantities. A novel technique, called aggregation, allows the simulator to switch between these models through the recognition and summary of cycles. The flexibility of PEPTIDE's aggregator allows the program to detect cycles within cycles and predict the behavior of complex situations.

This report is a revision of a thesis submitted to the Department of Electrical Engineering and Computer Science on May 11, 1984 in partial fulfillment of the requirements for the degree of Master of Science.

Copyright (C) Massachusetts Institute of Technology 1984.

Table of Contents

1. Introduction 1
1.1 A Simple Example 2
1.2 Discrete and Continuous Process Models 4
1.3 Aggregation: Switching Between Discrete and Continuous Models 6
1.4 Overview 8
2. Biology Background 9
2.1 Biochemistry 9
2.2 Regulation of Gene Expression 10
3. The Structural and Discrete Process Models 13
3.1 Model vs. Reality 13
3.1.1 Coarse Spatial Representation 14
3.1.2 Lack of Statistical Understanding 14
3.2 Structural Model 15
3.2.1 Molecules 15
3.2.2 Infinite Range Quantities 16
3.2.3 Histories 18
3.2.3.1 Nodes 19
3.2.3.2 Merging Histories Together by Fusing Nodes 20
3.3 Discrete Process Model 21
3.3.1 Generics 21
3.3.1.1 BIND 22
3.3.1.2 DROP 22
3.3.1.3 FOLD 23
3.3.1.4 SLIDE 23
3.3.1.5 REACT 23
3.3.2 User Defined Processes 25
3.3.3 Active Processes 26
3.3.4 Simulation 27
3.3.5 An Example 27
4. Aggregation 31
4.1 Recognizing Cycles 31
4.1.1 Sameness 31
4.1.1.1 Example Continued 32
4.1.1.2 When Groups Are Not Recognized as the Same 33
4.1.2 What Is Being Repeated? 34
4.2 When to Aggregate 35
4.3 Pruning Redundant Cycles 36
4.3.1 Repeating Cycles 36
4.3.2 Subsumed Singleton Cycles 36
4.4 Influence Generation 37
4.4.1 The Origin of Influences 37

4.4.1.1 Influences From Discrete PIEs 38
4.4.1.2 Influences From Continuous Processes 38
4.4.1.3 Deduced Influences from Implicit Diffusion 39
4.4.2 Canceling Influences 39
4.4.3 Example Continued 40
4.5 Cycles Within Cycles 41
5. Continuous Process Model **43**
5.1 Generating Possible Limit Points 43
5.1.1 Analyzing Process Preconditions 43
5.1.2 Filtering Out Irrelevant Limit Points 45
5.2 Selecting the Closest Limit Point 46
5.3 Simulation 47
5.4 Example Concluded 48
5.5 Deficiencies 50
5.5.1 Moving Limit Points 50
5.5.2 Loop Induction 51
6. More Examples **53**
6.1 Exonuclease 53
6.2 Protease 54
6.3 Transcription 57
7. Related Work **63**
7.1 Qualitative Simulation 63
7.2 Representations of Physical Change 64
7.2.1 Continuous Models 64
7.2.2 Discrete Process Models 66
7.3 Multiple Models 67
7.4 Other Applications to Biochemistry 69
8. Future Work **71**
8.1 Qualitative Simulation of Other Domains 71
8.2 More Complicated Biochemical Reasoning 72
9. Conclusions **73**

Acknowledgements

Many people have contributed to this thesis. In particular, I would like to thank:

My thesis advisor, Randy Davis, for substantively improving the clarity of these ideas, both in my mind and on paper;

Ken Forbus and Dave Chapman, for many formative ideas, interesting arguments, and helpful criticism;

Phil Agre, Dan Brotsky, Jim Davis, Dedre Gentner, Walter Hamscher, Mark Shirly, Chuck Rick and Brian Williams for useful comments and engaging conversations;

The intense and fulfilling research environments at the MIT AI lab and at BBN, including many more people than I can mention here;

Jill Hoskins, for support, encouragement and fierce debate over proper diction.

1. Introduction

A number of AI programs perform some type of qualitative simulation -- they predict the future behavior of a system in qualitative terms. Two fundamentally different models of change underly these simulations: discrete and continuous process models. STRIPS [Fikes 71], for example, uses a discrete model, in which actions were atomic; they have add and delete lists which define the action's effect on the state by an abrupt change. Qualitative Process Theory [Forbus 83], on the other hand, describes continuous process models, in which actions take periods of time and gradually change the value of quantities such as the level of fluid in a container.

In this thesis we explain what makes a domain well suited to using either a discrete or continuous process model and we present a new technique, ***aggregation***, that can be used to unify the two models of change. A program called PEPTIDE[1] has been written to test these ideas in the domain of molecular genetics. PEPTIDE starts its simulation using a detailed discrete model. Whenever it notices a repeating ***cycle*** of discrete processes, it aggregates the cycle into one continuous process by determining the net changes on all varying quantities. The program then switches to its continuous simulation mode and determines what values the influenced quantities will have "in the limit", i.e. at the time when currently active discrete processes will stop or an inactive one will start. Once these quantities have reached their limiting values, PEPTIDE switches back to discrete simulation. By noticing cycles, determining the limit points of these cycles, and performing the continuous simulation in one step, PEPTIDE sidesteps the time consuming work of simulating every iteration of a cycle.

---

[1] PEPTIDE is a system for the "Prediction of Enzymatic Process inTeractions from Individual Descriptions of Enzymes."

1.1 A Simple Example

Imagine a test tube filled with two kinds of molecules: A, which is small, and E, which is larger and has an active site that can grab an A. Initially, there is only one E, but there are very many A's. Suppose that there are three things (discrete processes) that can happen:

1. If E's active site is empty, it will grab an A.

2. If E is holding an A, E will chemically change the A to another small molecule, B, which will still be bound in E's active site.

3. If E is holding a B, E will drop the B, causing both molecules to float free.

What would the tube contain if left for a very long time? Clearly, all the A's would be replaced by B's. But how did you arrive at that answer? It is certain that you did not simulate E's molecular level behavior for all the A's. We didn't even tell you how many A's there were. More likely, you simulated E's behavior for a short while, then realized that the "same" things were happening repeatedly, in a cycle that would end only when all the A's were gone.

This is exactly how PEPTIDE solves the problem. The initial situation is displayed in figure 1-1.

Figure 1-1: Initial Situation

The program starts simulation by predicting the effects of the individual discrete processes. First, the E grabs one of the A's; the result is shown in figure 1-2. Then E changes the A into a B, producing the situation shown in figure 1-3. Next,

Figure 1-2: Situation After E Grabs an A

Figure 1-3: Situation After A is Transformed to B

E drops the B, leaving the active site ready to grab another A. The resulting situation is shown in figure 1-4.

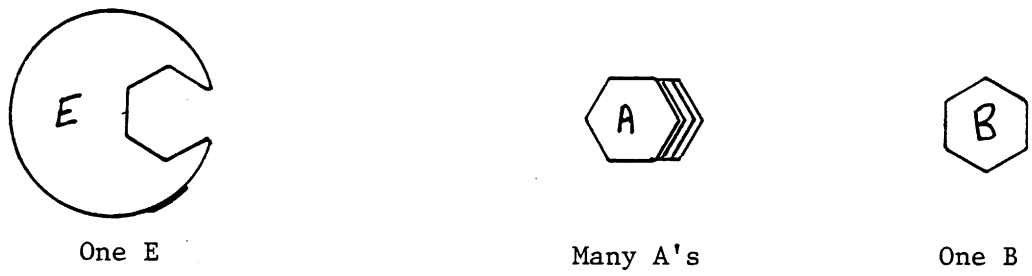

Figure 1-4: Situation After E Drops B

PEPTIDE prepares to simulate the first process again, then stops when it notices that it is the "same" as a previously simulated process. PEPTIDE determines that there is a cycle: the three discrete processes can be repeated many times. Instead of doing this slow, laborious simulation, PEPTIDE aggregates the cycle of discrete processes into a continuous process and simulates that.

A continuous process is a collection of influences produced by one iteration of the cycle--in this case, there is an increase in the number of B's and a decrease

in the number of A's. PEPTIDE simulates the continuous process by anticipating how the influences will affect the preconditions of the discrete processes--in this case, the first discrete process (the binding of E to A) will stop when there are no more A's for E to bind. This analysis allows PEPTIDE to quickly predict that all the A's will eventually get transformed into B's, as shown in figure 1-5.

Figure 1-5: Final Situation After Continuous Simulation

1.2 Discrete and Continuous Process Models

The distinction between discrete and continuous processes is a result of our temporal model. We assume that time is composed of ***instants*** and ***intervals.*** Instants are infinitesimal, while intervals are extended. There are an infinite number of instants in the shortest interval.

Some types of change are naturally modeled as happening in an instant. The flip of a light switch is a good example, as are many biochemical mechanisms. We use a form of situation-action rules, which we call ***discrete processes***, to model these instantaneous changes. ***Continuous processes***, on the other hand, are used to represent changes that are extended over time. The flow of water from a faucet to a tub is an example of a change that might be best modeled as continuous. Continuous processes are defined by the set of ***influences*** that they cause; fluid flow causes two influences: a decrease in the level of water in the source and an increase in the level of water in the destination. Regardless of the representation of change, discrete or continuous, we assume that all quantities (i.e. all parameters of the system being modeled) are defined at all times.

From the assumption that discrete processes act instantaneously, we can conclude that they are atomic--they either happen fully or not at all. We make no restrictions on the magnitude of change produced by a discrete process. Continuous processes, on the other hand, we assume act strictly monotonically. If a quantity is being influenced by a single continuous process during an interval, then the value of that quantity must be monotonically increasing or decreasing during the entire interval. This implies that the influenced quantity must take on different values for each instant in the interval.

From this monotonicity assumption and the assumption that there are an infinite number of instants in an interval, we can conclude that only quantities with an infinite number of possible values, ***infinite range (IR) quantities*** can be influenced by a continuous process.[2] Quantities with finite number of possible values (e.g. the possible positions of a light switch, on and off) are called ***finite range (FR) quantities***. Although continuous processes can only affect IR quantities, discrete processes can affect both FR and IR quantities.

Because of their atomic action, discrete process models are useful for modeling situations where mutually-exclusive actions are possible [Habermann 76]. Discrete processes are also required when modeling a system in which certain parameters are best represented as FR quantities. In the domain of molecular genetics, for example, discrete processes are the most convenient way to model enzymatic reaction mechanisms.

Continuous process models are best for reasoning about long gradual changes. These extended changes occur frequently in biochemistry; they are usually caused by the combined effects of many molecules, or the effect of one molecule which repeats its action many times.

---

[2]PEPTIDE uses a qualitative arithmetic in which very large numbers are considered "infinite". Thus there were an infinite number of A's in the initial situation in the example of section 1.1.

1.3 Aggregation: Switching Between Discrete and Continuous Models

With two fundamentally different ways of thinking about how a system behaves, an important problem is how to choose which one to use at a given time. We solve this by making the discrete model the default and using the continuous model only when cycles are detected.

The system starts by simulating the effects of discrete processes until it recognizes a cycle in which all discrete processes are being repeated. It then aggregates the trace of a single iteration of the cycle of discrete process executions, producing a continuous process. The resulting continuous process is simply a list of the net influences on IR quantities produced by one iteration of the cycle.

The recognition of cycles (i.e. repeating sequences of processes) depends on the definition of process ***sameness***. How did we detect the cycle in the example above? Although the situation was different the two times the first process was active (because of the change in the number of the molecules A and B), the situation was the same "in some sense".

Our definition of "in some sense" is to consider two processes the same if they are equal once they have been generalized with a set of domain-dependent abstractions. For the domain of molecular genetics, these abstractions are:

* ***The number of molecules.*** This is the abstraction used in the example above; the two instances of the first process are the same because only the number of A's and B's has changed.

* ***The length of chains.*** This abstraction is required when reasoning about processes which cause the length of a polymeric chain (e.g. DNA or RNA) to grow or shrink.

* ***The position of binders on a chain.*** This abstraction is required when reasoning about molecules which grab a chain at a specific

position and then move (e.g. RNA Polymerase or ribosomes).

Once a cycle is recognized, the aggregator generates a set of influences that summarize the net changes to IR quantities which result from one iteration of the cycle. In a sense the aggregator uses an inductive technique to generate a continuous process abstraction of the cycle of discrete processes. In the example above, the overall influence of the cycle was a decrease in the number of A's and an increase in the number of B's.

When the aggregator passes the newly generated continuous process to the continuous simulator, the continuous simulator determines what change will stop the process, i.e. what is its ***limit point***? A limit point is a value of a IR quantity where a currently active discrete process[3] will stop, or an inactive discrete process will start. The continuous simulator examines all such limit points for the IR quantities being influenced and selects the one nearest to its quantity's current value. The continuous simulator assumes that when the chosen quantity reaches this limit point, the situation will become more complex, so it updates the state with the limit values and returns to the discrete simulator. In the example above, there was only one limiting value to choose from: that when the number of free A's reached zero. This was a limit point because the E can no longer bind an A when all the A's are gone; if one of the cycle's discrete processes becomes inactive, then the continuous process is inactive. When PEPTIDE switched back to the discrete simulator at that point, no discrete processes were active, so the simulation was finished.

---

[3] I.e. a member of the cycle which was aggregated to form the continuous process.

1.4 Overview

Chapter two contains a brief summary of biochemistry and molecular genetics. In chapter three we discuss PEPTIDE's representation of molecules and the discrete process model. Aggregation, the focus of the thesis, is presented in chapter four. Chapter five explains what PEPTIDE does with the continuous processes that are generated by the aggregator. Chapter six is a description of five different examples, all of which run. In chapters seven and eight we discuss related work and suggest future directions for research. Chapter nine is our conclusion.

2. Biology Background

The following concepts from molecular biology will clarify the aims of this research. For the sake of simplicity, we give a highly abbreviated description; for more detail, see [Wood 81] or [Watson 77].

2.1 Biochemistry

To reason about biochemical systems, PEPTIDE needs a model for two classes of chemicals, proteins and nucleic acids, which form the foundation of the molecular mechanism of life. ***Proteins*** are long sequences of amino acids linked together end to end. Once formed, each protein folds into a specific three dimensional shape called a ***conformation***. Although proteins have other functions, we will be most interested in the class of proteins, called ***enzymes***, which catalyze chemical reactions. Enzymes often form conformations with neatly shaped holes or clefts, called ***active sites***, into which other molecules may fit. When a substrate molecule (which could be another protein, DNA or any other molecule) fits into the active site (a process called ***binding***) the enzyme may change its conformation by folding differently or catalyze some reaction. Enzymes speed up chemical reactions so much that by comparison uncatalyzed reactions just do not happen.

The nucleic acids, DNA and RNA, are very long strings of units called nucleotides. In both cases only four nucleotides are used to form the chains. DNA is a very stable molecule which is uniquely suited to store vital information for long periods of time by encoding it with its alphabet of four nucleotides. Whenever a cell divides, it must copy its DNA (through the action of enzymes) by a process called replication. See figure 2-1.

To influence the actions of a cell, the information stored in the DNA must be converted into protein. This action is called ***gene expression*** and is accomplished with a two-step process. The first step, called ***transcription***,

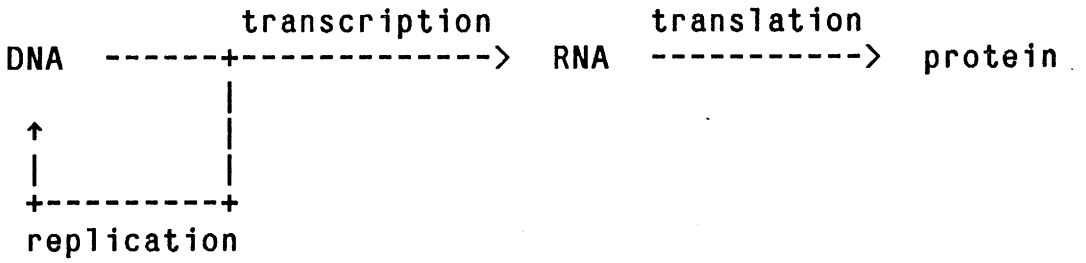

Figure 2-1: Gene Replication and Expression

copies an individual message, or gene, from DNA into RNA. This process is carried out by the enzyme RNA Polymerase (RNAP) by a multi-step process. First RNAP binds to a specific region of the DNA (called a ***promoter***) that is adjacent to the gene. Then the enzyme slides towards the gene until it recognizes the start of the gene. The RNAP enzyme then slides along the gene, at each step adding to the RNA chain it is building, by appending a nucleotide which matches the nucleotide at its current position on the DNA template. Finally, when the enzyme recognizes the gene's end, it drops off the DNA and frees the RNA it has constructed.[4]

The second step, called ***translation***, takes the RNA transcript and from it builds a protein. Ribosomes execute this function by dividing the transcript into small subsequences, called codons, formed of three nucleotides each and associating each codon with a specific amino acid. The protein is constructed by appending the amino acids with a mechanism that is more complicated than that of transcription and will not be explained here.

2.2 Regulation of Gene Expression

The products of different genes are needed in different amounts at different times. To meet these needs, cells have evolved methods of both analog and digital control of transcription and translation. One such method, called negative repression, involves the addition of a new region of DNA, an ***operator***, between the promoter and the gene (figure 2-2).

---

[4]The example in section 6.3 is a simplified version of transcription.

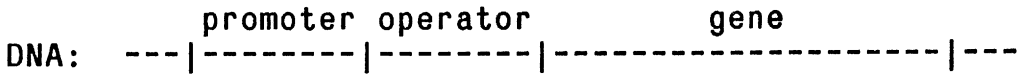

Figure 2-2: A Repressible System

An enzyme called a ***repressor*** acts like a watchdog, monitoring the cell's need for the product of the gene. If the concentration of the gene's product rises above a threshold, the repressor binds to the operator region of the DNA and blocks RNAP from transcribing the gene. This is just one simple example of a series of regulatory mechanisms which have been discovered; there are many others.

3. The Structural and Discrete Process Models

Discrete processes are PEPTIDE's most detailed description of change. Understanding the mechanistic nature of discrete processes requires knowledge of PEPTIDE's structural model including the representations of molecules, IR quantities and histories. Before presenting these models, we explain the simplifying assumptions which cause PEPTIDE's models to diverge from reality.

3.1 Model vs. Reality

We believe that experts in molecular genetics, like experts in many other fields, reason with multiple models, each at a different level of abstraction. These levels delineate the curve of a power/accuracy tradeoff peculiar to the domain. When reasoning at a more abstract level, an answer is easier to reach but more likely to be incorrect. For the field of molecular genetics, we suspect the existence of the following levels:

1. CHEMISTRY LEVEL
   This level supports concepts such as equilibrium, entropy, acids, and electrophilic attraction which are useful in understanding detailed enzyme reaction mechanisms.

2. TINKERTOY LEVEL[5]
   Models at this level make biochemical systems resemble a complex factory. Enzyme machines step down polymeric chains, building parts, clamping subassemblies together and swinging them about, etc. This is the level of reasoning that PEPTIDE uses.

3. OPERON LEVEL
   We suspect that when experts consider the behavior of large genetic systems, such as phage λ, they use models built from control cliches such as negative and positive repression, induction, attenuation, etc. This type of knowledge closely resembles the programming cliches described in [Rich 80].

Although each of these levels is useful in predicting the behavior of some genetic

[5]Also called the mechanochemical level.

system, the construction of a program which moves between levels was deemed too difficult for the present. As a result, PEPTIDE models only at the tinkertoy level. Although the tinkertoy level is quite powerful it suffers from a coarse spatial representation and lack of statistical understanding.

3.1.1 Coarse Spatial Representation

Since two things can not happen at once in the same place, some molecular processes are mutually exclusive. A realistic mechanism for determining spatial contention would be to model the size, shape and orientation of the molecules involved and calculate whether the two processes would cause any collisions. Since we feel that this type of spatial reasoning, appropriate at the chemistry level, is too complex for the tinkertoy level, we use a much simpler model and define a "place" to be a connected group of molecules. Thus only one process may act on a molecule or connected complex of molecules in a single time unit. The cost of this simplicity is an overly restrictive model. Although it allows unrelated molecules to be modified independently, it restricts large molecules unnecessarily. It means, for example, that only one molecule can bind to a DNA chain at a time, no matter how long the chain or how small the binders; this is clearly unrealistic.

3.1.2 Lack of Statistical Understanding

In situations where two mutually-exclusive processes are competing for a molecule, there is no fair arbitration in our model. The lack of statistical reasoning frequently causes PEPTIDE to repeatedly select one of two mutually-exclusive processes, allowing the other to starve (i.e. a livelock situation [Habermann 76]). As a partial solution to these problems, the tinkertoy level does rank mutually-exclusive processes by priorities in order to choose the process most likely to obtain the scarce resource. Section 3.3.3 contains more details. Also (if desired) the user can manually select which process PEPTIDE will choose.

3.2 Structural Model

In this section we discuss PEPTIDE's representations of molecules, IR quantities, and histories. Both the discrete and continuous process simulators use these structural representations.

3.2.1 Molecules

For PEPTIDE to simulate a biochemical system, it must know what kind of molecules exist in the system and what the initial conditions are. The types of molecules are defined first, in a step akin to defining user types in a conventional programming language. Since groups of molecules are represented by a representative member and the group's size, the second step is to state how many molecules of each type are initially present.

Each molecule type is described with respect to possible ***views***, i.e. different ways that the molecule can be treated. There are three possible views: the ***bindee***, ***binder***, and ***chain*** views. Aside from the restriction that no molecule may have both a bindee and a chain view, molecules may have one of each view, or no views at all.

A ***bindee view*** corresponds to a named set of ***handles***, which each have a conformation or "shape". If a molecule has a bindee view, then any handle may be bound (grabbed) by another molecule through the actions of a BIND process. In the example in section 1.1, molecule A has a bindee view which gets bound by E.

A molecule with a ***chain view*** is similar to one with a bindee view because it can be bound by other molecules, perhaps at multiple places. A chain view imparts additional structure, however, in the form of an ordering and metric among these handles. While a molecule with a bindee view is considered a clump with unrelated handles sticking out at random, a molecule with a chain view (e.g. a strand of DNA) has a linear chain of handles. Chains have endpoints, but can be

arbitrarily dense. Since positions on a chain are modeled as IR quantities, they are further described in section 3.2.2.

A molecule with a ***binder view*** has a set of ***active sites***. These sites are termed "active" because they can bind molecules with bindee or chain views and they can change conformations. To specify an active site, a user lists (by name) the possible conformations of that site; it must have at least one. The molecule, E, in the example of section 1.1 has a binder view with one active site.

Since three dimensional shapes are not modeled at our level of abstraction (the tinkertoy level), PEPTIDE can not calculate which conformations are ***complementary***. Instead, PEPTIDE reasons about conformations symbolically, and the user asserts which ones are complementary as part of the structural description. For PEPTIDE to know that the conformation of E's active site matches the conformation of A's handle in the example of section 1.1 for example, the user must tell PEPTIDE that they are complementary.

3.2.2 Infinite Range Quantities

The number of molecules, positions on a chain, and the length of chains are all IR quantities. There were three major constraints that directed our representation of these IR quantities:

* The representation must facilitate the task of locating limit points during continuous simulation. Thus it must be easy to determine the relationship between the current value of a quantity and distinguished points and also the relationship between different distinguished points.

* Since we wanted PEPTIDE to simulate underspecified and partially ambiguous systems, the representation must allow ambiguity.

* Finally, because IR quantities in biochemistry can take on such a vast range of values, we wanted PEPTIDE to be able to qualitatively represent when two values differed by several orders of magnitude.

Inspired by Forbus's representation of quantity spaces [Forbus 83], we represent IR quantities as partial orders. To capture the diverse types of relationships (including different orders of magnitude) we have developed a vocabulary of qualitative relations with which we describe differences in a quantity's possible values. If two quantities, p and q, are related then one of seven relations is true between them: EQ, 1GT, FGT, MGT, MLT, FLT, 1LT. EQ implies that p and q are equal. 1GT means that p is "one greater than" q; in other words, q plus the smallest positive quantity, one, EQ p. This implies that there is no quantity between p and q. The notation, p FGT q, means p is "finitely greater than" q. In other words, there is a finite length sequence of quantities, x_1, x_2, ... , x_n such that p 1GT x_1, x_i 1GT x_{i+1}, and x_n 1GT q. If p MGT q, then p is "much greater than" q; in other words they differ by orders of magnitude. Figure 3-1 gives the complete semantics for MGT. The most important quality of the relation is:

* If p MGT q and (x 1LT p or x FLT p), then x MGT q. This implies that there can be no finite length sequence of quantities, each 1GT from the other, bridging the gap between p and q (as there was when p was FGT q).

If the MGT relation were not included, the possible values of IR quantities would be isomorphic to the integers. Although the inclusion of MGT in our qualitative algebra adds complexity, the ability to consider one value insignificant compared with another is quite powerful. The remaining relations, MLT, FLT, 1LT are analogous to the ones we just described, but they specify that p is less than q to some extent.

The partial order which relates the values of IR quantities is implemented as a set of rules in a simple forward-chaining propositional constraint propagator. The rules of figure 3-1 cause the constraint propagator to fill out all possible relations according to symmetry and transitivity laws. To increase the speed of propagation, we store the assertions for each IR quantity in a separate group called a truth module (TM).

```
  ;; notation: symbols that start with a question
  ;; mark ("?") are considered to be variables.

  ;; equality stuff
(and (EQ  ?a ?b) (EQ  ?b ?c)) => (EQ ?a ?c)
(and (1GT ?a ?z) (1GT ?b ?z)) => (EQ ?a ?b)
(and (1GT ?z ?a) (1GT ?z ?b)) => (EQ ?a ?b)
(EQ  ?a ?b)                   => (EQ ?b ?a)

  ;; one greater than transitivity
(and (1GT ?a ?b) (EQ  ?b ?c)) => (1GT ?a ?c)
(and (1GT ?a ?b) (1GT ?b ?c)) => (FGT ?a ?c)
(and (EQ  ?a ?b) (1GT ?b ?c)) => (1GT ?a ?c)

  ;; finitely greater than transitivity
(1GT ?a ?b)                   => (FGT ?a ?b)
(and (FGT ?a ?b) (EQ  ?b ?c)) => (FGT ?a ?c)
(and (FGT ?a ?b) (1GT ?b ?c)) => (FGT ?a ?c)
(and (FGT ?a ?b) (FGT ?b ?c)) => (FGT ?a ?c)
(and (1GT ?a ?b) (FGT ?b ?c)) => (FGT ?a ?c)
(and (EQ  ?a ?b) (FGT ?b ?c)) => (FGT ?a ?c)

  ;; much greater than transitivity
(and (MGT ?a ?b) (FGT ?c ?b)) => (MGT ?a ?c)
(and (MGT ?a ?b) (EQ  ?b ?c)) => (MGT ?a ?c)
(and (MGT ?a ?b) (FGT ?b ?c)) => (MGT ?a ?c)
(and (MGT ?a ?b) (MGT ?b ?c)) => (MGT ?a ?c)
(and (FGT ?b ?a) (MGT ?b ?c)) => (MGT ?a ?c)
(and (EQ  ?a ?b) (MGT ?b ?c)) => (MGT ?a ?c)
(and (FGT ?a ?b) (MGT ?b ?c)) => (MGT ?a ?c)
```

Figure 3-1: Quantity Relation Rules

3.2.3 Histories

PEPTIDE records the change in molecules over time as a set of interconnected ***histories***, as suggested in [Hayes 83]. The history of a group of molecules is a sequence of snapshots of the group, called ***nodes***, connected together by ***change links*** which represent changes to the group caused by discrete or continuous processes. For example, figure 3-2 shows the history of the molecule E in the example of 1.1.

Histories are not always simple unbranching chains. When multiple processes

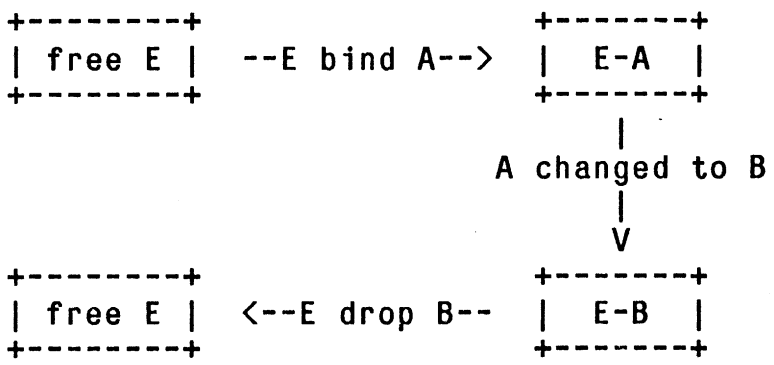

Figure 3-2: The History of Molecule E

are acting on a group of molecules (i.e. a node), the history splits; some of the molecules are affected one way and some the other. Because different sequences of processes can cause identical changes to molecules, split histories or even two unrelated histories can merge together.

Whenever PEPTIDE needs to consider the global state of the system at a particular time, it creates a ***situation*** by going through each history and collecting the node which represents each group's state for the required instant. For example, the situation after the first process (E bind A) in the example of section 1.1 contains three nodes: one representing the large group of free A's, one representing an A molecule bound in E's active site, and one representing the E molecule that is binding an A.

3.2.3.1 Nodes

A node represents a group of identical molecules at one instant of time. Nodes represent a group by depicting the state of a representative individual and saying how many such individuals are in the group. To completely describe a group's representative individual, the node records information about each view:

* For a bindee view, the node notes whether any of the handles are bound. For each handle that is bound, the state records the binder and the active site involved.

* For a chain view, the node contains information about the the positions on the chain and a record of all binders including each

binder's active site and position of attachment.

* For a binder view, the node records the status of each active site. This status includes the conformation of the site, whether the site is binding another node, and if it is, the bound node and the handle or position of attachment.

For example, the initial situation in the system described in section 1.1 was a collection of two nodes, one for A and one for E. The A-node represented "many molecules" which means that the number slot of the node contained a number which was much greater than (MGT) zero. The sample individual of the A-node was of type A, and had only an unbound bindee view.

3.2.3.2 Merging Histories Together by Fusing Nodes

The nature of biochemistry is such that different processes can independently result in identical molecules. PEPTIDE notices that this has happened when a situation contains two or more nodes with identical representative individuals. Whenever PEPTIDE detects this redundancy, it fuses the two nodes together to create a single new node in which the number of the new node is equal to the sum of the old nodes. By merging histories in this way, PEPTIDE keeps the description of a situation from getting fragmented.

Nodes that represent an empty group, i.e. nodes whose number slot is zero, are always removed from the current situation.

3.3 Discrete Process Model

Discrete processes form a detailed tinkertoy-level description of the mechanical behavior of a molecular genetic system. Users define a discrete process by instantiating one or more ***generics***.

3.3.1 Generics

To describe specific discrete processes, users need a vocabulary of possible types of action. We had three main goals in mind when we devised this vocabulary: simplicity, orthogonality and coverage of known biochemical mechanisms. Because molecular mechanisms are spatially complex, these goals are difficult to satisfy. As a compromise between goals, we chose a set of five generics: BIND, DROP, FOLD, SLIDE, and REACT. Although there are some real life mechanisms that can not be encoded (e.g. phage λ's recombination in lysogeny [Herskowitz 80]), the set chosen is quite simple, orthogonal, and covers many cases.

Each generic is specified by three parts, its ***pattern***, a set of ***preconditions***, and a set of ***postconditions***. The pattern is a list of formal parameters that must unify with the actual parameters. The preconditions are a set of predicates that are 'anded' with any user-specified conditions to determine when a user process is active. PEPTIDE supplies about a dozen built-in predicates which may be used in the preconditions of a process. These predicates test qualities in the current situation such as the relative size of numbers, the complementarity of conformations, the relative positions on a chain, the length of a chain, and whether a molecule is bound to another. The postconditions summarize the changes that would be caused if a user process containing this generic were active.

3.3.1.1 BIND

The BIND generic takes as parameters the active site of one molecule and a handle (or chain position) on some other molecule. To be active, there must be at least one binder and at least one molecule to be bound, the active site must be empty, the handle must be free, and the conformations of the active site and the handle must be complementary. After the BIND generic has been simulated, the handle of the bindee molecule will be bound by the active site of the binder.

Knowledge of the following notational conventions will help understanding of the summary in figure 3-3. Functions whose name end in a question mark ("?") are predicates that return true or false. Symbols that start with a question mark denote variables that may be unified to any value. Arrows (">") separate molecule specifiers from view specifiers; for example, "?binder>?site" denotes any active site of any molecule with a binder view.

```
(BIND ?binder>?site ?bindee>?handle)

(and (gt? (number-of ?binder) zero)
     (gt? (number-of ?bindee) zero)
     (not (bound? ?binder>?site ?molecule1))
     (not (bound? ?molecule2>?site2 ?bindee>?handle))
     (complementary? ?binder>?site ?bindee>?handle))

(bound? ?binder>?site ?bindee>?handle)
```

Figure 3-3: BIND Pattern, Preconditions, and Postconditions

3.3.1.2 DROP

The DROP generic is the inverse of BIND. To be active, there must be two molecules bound together. After DROP simulation, the two molecules are no longer bound. See figure 3-4.

```
(DROP ?bindee>?handle ?binder>?site)

(and (gt? (number-of ?binder) zero)
     (gt? (number-of ?bindee) zero)
     (bound? ?binder>?site ?bindee>?handle))

(not (bound? ?binder>?site ?bindee>?handle))
```

Figure 3-4: DROP Pattern, Preconditions, and Postconditions

3.3.1.3 FOLD

FOLD is the simplest generic because it requires only one molecule. It is active whenever there is a molecule whose site is not already in the specified conformation. Simulation of FOLD causes the conformation of the binder's active site to assume the new value. See figure 3-5.

```
(FOLD ?binder>?site ?conformation)

(and (gt? (number-of ?binder) zero)
     (not (has-conf? ?binder>?site ?conformation)))

(has-conf? ?binder>?site ?conformation)
```

Figure 3-5: FOLD Pattern, Preconditions, and Postconditions

3.3.1.4 SLIDE

The SLIDE generic captures the ability of binders to move along linear chains. As specified in figure 3-6, SLIDE is only active when there is nothing blocking the sliding movement. Thus the binder can not slide if it is already at the end of the chain, nor can it slide if there is another binder directly in its way. The effect of simulating SLIDE is the movement of the binder to a new position one unit from the starting position in the direction specified.

3.3.1.5 REACT

REACT has two parameters: the active site of a binder, and the specification of a new molecule that will be created by the process. See figure 3-7.

Because REACT is a very complicated generic capable of specifying arbitrary

```
(SLIDE ?binder>?site ?chain ?dir)

(and (gt? (number-of ?binder) zero)
     (gt? (number-of ?chain) zero)
     (bound? ?binder>?site ?chain>?start-pos)
     (not (at-end? ?start-pos ?dir ?chain))
     (next-pos? ?adjacent-pos ?dir ?start-pos)
     (not (bound? ?blocker>?bsite ?chain>?adjacent-pos)))

(bound? ?binder>?site ?chain>?adjacent-pos)
```

Figure 3-6: SLIDE Pattern, Preconditions, and Postconditions

```
(REACT ?binder>?site ?add-spec)

(gt? (number-of ?binder) zero)

<<see text for postconditions>>
```

Figure 3-7: REACT Pattern and Preconditions

chemical reactions (at the tinkertoy level of detail), we can not easily describe its effects with predicate calculus; so we substitute an English description for the list of postconditions.[6] Whenever REACT is active, the molecule bound in the binder's active site (if any) is destroyed. Depending on the value of the ?add-spec variable, one of three things can be placed in the binder's active site: 1) nothing, 2) any defined molecule, or 3) if a molecule with a chain view was just destroyed, a modified version of that molecule with a slightly longer or shorter chain. Although it might seem strange to provide a generic which does not conserve mass, this is essential: biochemists frequently use abstractions which violate conservation, so the tinkertoy level must support them.

[6]Due to the graph representation of the history structure, the postconditions of all of the generics are implemented procedurally rather than in logic.

3.3.2 User Defined Processes

PEPTIDE users describe the behavior of a system by creating discrete processes from generics. Three kinds of modification are possible: combination, specialization, and the addition of preconditions; all three are common, and all may be combined.

Specialization is used to restrict the pattern of a generic. Figure 3-8 shows a simple BIND process in which the binder molecule and active site have been specialized. Whenever site1 of enzyme1 is empty and there is any complementary molecule around, this process will be active. Note that there is no restriction on what can or can not be specialized.

```
(def-process specialized-bind
  (BIND enzyme1>site1 ?bindee>?handle))
```

Figure 3-8: Process Definition with Specialization

To restrict the times when a process is active, one can add preconditions which get added to the default preconditions of the generic. Figure 3-9 shows a simple, unspecialized process with a single precondition. This process says that whenever there are two molecules of the same type that can bind each other, they will.

```
(def-process self-bind
  (if (equal? ?binder ?bindee)
      (BIND ?binder>?site ?bindee>?handle)))
```

Figure 3-9: Process Definition with an Additional Precondition

More complex types of behavior can be defined by combining several generics into one process. The process defined in figure 3-10 is active whenever the union of the default preconditions of the two generics are satisfied: If slider is bound to the middle of DNA and RNA and no other binders are blocking it, then slider will move along both DNA and RNA at the same time.

```
(def-process double-slide
  (SLIDE slider>site1 dna right)
  (SLIDE slider>site2 rna right))
```

Figure 3-10: Process Definition with Combination and Specialization

3.3.3 Active Processes

When the preconditions of a process are satisfied, then that process is said to be ***active***; it will have an effect. Before a process can be tested for activity, it must be associated with an environment that binds all free variables; we call the discrete-process, environment pair a ***PIE***. Given a situation at some time, t, PEPTIDE generates the set of PIES by computing all combinations of variable bindings for each process, and then tests the PIEs for activity.

The first step in the test for activity is the evaluation of the preconditions in the environment. Even if all the preconditions are satisfied, however, the PIE is not necessarily active--two or more active PIEs may be mutually exclusive. As we mentioned in section 3.1.1, only one PIE at a time may access a molecule, or a connected complex of molecules. Whenever contention is detected, one of the PIEs must be chosen. Since both choices are possible, it is important to allow the user to see all possible outcomes; PEPTIDE provides this ability by optionally letting the user select the PIE to run next.[7] If the user decides to let PEPTIDE select between competing PIEs, the PIEs are priority ranked (by a method explained below), and the best is chosen. In the event of a tie, the choice is arbitrary.

As detailed below, the ranking system is motivated by a simple qualitative model of diffusion: the highest priorities are awarded to PIEs that are very local. PIEs that are catalyzed by molecules that are very plentiful are also favorably ranked.

---

[7] This facility could be easily extended to one that generated all possible futures by backtracking.

* Any PIE that modifies only one molecule (i.e. is a specialization of the single generic FOLD) has the highest priority.

* Any PIE that involves only molecules that are connected together (via active site binding) has second priority.

* Any PIE that involves molecules that are not connected has the lowest priority. Since many PIEs are in this priority class, PEPTIDE uses a secondary priority scheme to distinguish amongst PIEs at this level: the number of the least plentiful of the required molecules.

3.3.4 Simulation

Discrete simulation is straightforward. For each active PIE, PEPTIDE determines which groups of molecules contain members that will be modified by the PIE. The size of each of these groups is decremented. Then PEPTIDE creates new groups each containing the one member molecule that is being modified by the PIE and updates the state of the representative molecules in the new groups to reflect the effect of the PIE. The history structure is updated by connecting the nodes representing the new groups to the nodes representing the old groups using a change-link which lists the PIE and records that the change occured in an instant.

3.3.5 An Example

Many biochemical pathways are regulated by negative feedback systems; when the concentration of a product molecule reaches a certain threshold, the processes which produce the molecule are shut off. In this section we will show how to describe such an enzyme and we will follow PEPTIDE as it simulates the discrete processes. In chapters four and five we will show how the aggregator recognizes a cycle and how the continuous simulator predicts the long term behavior of the system.

First we define the molecules; there are three in this example. Both substrate and product have a single bindee view with one handle. Frobase has a binder view

with two active sites: the catalytic site has two possible conformations, active and inactive, but the control site has only one.

```
(def-obj substrate (bindee))
(def-obj product   (bindee))
(def-obj frobase   (binder (cat-site active inactive)
                           (control-site on)))
```

We declare that the active conformation of frobase's catalytic site is complementary to substrate's handle, the inactive conformation of the catalytic site is not complementary to anything, and the frobase's control site is always complementary to product's handle. We define two qualitative numbers, million and threshold; threshold is much greater than zero, and million is much greater than threshold. Initially, there are a million substrates, no products, and one frobase:

```
(def-qrel (complementary frobase>cat-site>active
                         substrate>bindee))
(def-qrel (complementary frobase>control-site>on
                         product>bindee))
(def-qrel (mgt threshold zero))
(def-qrel (mgt million   threshold))
(def-qnum substrate million)
(def-qnum product zero)
(def-qnum frobase one)
```

Now we define the discrete processes. The first says: whenever frobase's catalytic site is in the active conformation, it will try to bind a substrate. Since each user process inherits the preconditions of the generics from which it is defined, when we write

```
(def-process bind-sub
  (bind frobase>cat-site substrate>bindee))
```

it will be expanded to

```
(def-process bind-sub
  (if (and (gt? (number-of frobase) zero)
           (gt? (number-of substrate) zero)
           (not (bound? frobase>cat-site ?molecule1))
           (not (bound? ?molecule2>?site2
                        frobase>cat-site))
           (complementary? frobase>cat-site
                           substrate>bindee))
      (bind frobase>cat-site substrate>bindee)))
```

If a substrate molecule is bound in the catalytic site, a reaction will change it into a product molecule:

```
(def-process react
  (if (bound? frobase>cat-site substrate>bindee)
      (react frobase>cat-site product>bindee)))
```

If frobase has a product molecule bound in the catalytic site, it will drop the product molecule:

```
(def-process drop-prod
  (drop product>bindee frobase>cat-site))
```

If the number of product molecules is ever greater than threshold, frobase will bind one of the product molecules in the control site:

```
(def-process bind-prod
  (if (gt? (number-of product) threshold)
      (bind frobase>control-site product>bindee)))
```

If a product is bound in frobase's control site and frobase's catalytic site is in the active conformation, frobase will change the conformation of the catalytic site to become inactive:

```
(def-process repress
  (if (bound? frobase>control-site product>bindee)
      (fold frobase>cat-site inactive)))
```

PEPTIDE starts simulation in discrete mode. A set of PIE's are formed from the set of processes, and preconditions are tested. The only active PIE is bind-sub acting on the free frobase and one of the million free substrate molecules; it is

simulated. The resulting situation has a million minus one free substrate molecules, and a single frobase/substrate complex.

At this point, the sole active PIE is the react process acting on the frobase and the bound substrate; it is run. Now the situation contains a million minus one free substrate molecules, and one frobase bound to a single product molecule.

A PIE formed from the drop-prod process acting on the solitary frobase and product molecules is active, so it is simulated. The resulting situation consists of a group of a million minus one free substrate molecules, one free frobase molecule, and one free product molecule.

PEPTIDE tests all possible PIE's for activity again, and finds that a bind-sub PIE is active. Before it is simulated, however, the aggregator checks and finds that a cycle has been formed--further discrete simulation would be repetitious. As a result, the aggregator converts the cycle of three PIEs (bind-sub, react, drop-prod) into a continuous process so that the net effect of many iterations can be computed directly. The techniques involved in this transformation are the subject of our next chapter.

4. Aggregation

Many mechanical systems are repetitious--certain patterns of behavior repeat. Since these cycles continue for some time, it is natural to reason about them with a continuous process model. Aggregation is a technique for converting a recurring cycle of discrete processes into a continuous process for more efficient simulation.[8] PEPTIDE's aggregator consists of two parts: a cycle recognizer which compares and classifies discrete PIEs, and an influence generator, which produces a continuous process from a set of cycles of discrete PIEs.

4.1 Recognizing Cycles

A ***cycle*** is defined as a sequence of discrete PIEs such that the starting PIE is the ***same*** as the ending PIE. To recognize a cycle, the aggregator must be able to tell when two PIEs are the same.

4.1.1 Sameness

The intuition that we are attempting to capture is that two PIEs are the same if they represent an identical action on groups of molecules whose FR quantities are identical but whose IR quantities may have changed. Only these cycles are useful to aggregate, because only these will result in a continuous process with no discrete effects.

Two PIEs are the same if they are instantiated from identical processes and the corresponding groups of molecules specified in the PIE's environments are the same. We check for sameness between two groups of molecules by computing abstractions of the two groups and comparing the abstractions for equality. For the domain of molecular genetics, two groups are declared to be the same if they are identical after:

---

[8]The reader may find it useful to consider the aggregator as an inductive process abstractor. It computes a cycle's induction formula and generates a more abstract description of the ongoing behavior in the form of a continuous process. It would be useful to explore the applicability of our techniques for the analysis of programs with loops.

* Ignoring the number of molecules in the groups;

* Ignoring the length of chains;

* Ignoring the position of binders on chains.

In different domains, different parameters are usefully considered as IR quantities. This implies that the sameness abstractions are necessarily domain-dependent. Since cycles can only change the IR quantities in a system, the IR quantities can be used to derive a new set of abstractions. When reasoning about internal combustion engines, for example, variables like temperature and pressure will be important IR quantities. A new abstraction will be necessary to recognize cycles that change the temperature of the system. Because of the direct link between the abstractions and the common IR quantities of the domain, we suspect that it will be relatively easy to develop cycle recognizers for other domains.

In conclusion, two objects (e.g. PIEs) are tested for sameness recursively:

* Two values for a FR quantity (e.g. a molecule type, a conformation, a direction, or a bindee handle) are the same if and only if they are equal.

* Two values for a IR quantity (e.g. the size of a group, the position of a binder on a chain, or the length of a chain) are always considered the same.

* Two values for a composite object are the same if all attributes are the same.

4.1.1.1 Example Continued

When we left the example of section 3.3.5, the discrete simulator had run three PIEs in a row: bind-sub acting on the solitary frobase and a member of the group of a million free A's, react acting on the frobase and the bound A, and drop-prod acting on the frobase and the bound product. At this point the simulator determines that there is one active PIE: bind-sub acting on the frobase and a

member of the group of a million minus one free A's. Before the PIE can be simulated, however, the cycle recognizer notices that this new PIE is the same as the first PIE that was simulated--both PIEs are instantiated from the process bind-sub, and the environments of both PIEs specify two groups that are the same. The two groups of one free frobase are clearly the same, because there are no differences, discrete or continuous. The group of a million free A's is the same as the group of a million minus one free A's, because the only difference is a change in the IR quantity, the size of the group.

4.1.1.2 When Groups Are Not Recognized as the Same

Although these abstractions allow the detection of many common cycles, it is possible to create examples in which PEPTIDE will not recognize a repeating cycle and will never halt. To create a situation which will cause PEPTIDE to fail, it is necessary to devise a new IR quantity that PEPTIDE will not recognize as such. For example, although PEPTIDE considers the length of a chain to be a IR quantity, it requires that the chain view of a molecule be predeclared; PEPTIDE can not recognize chains that form spontaneously.

Consider a situation with many G molecules that each have a bindee view with one handle and a binder view with an active site complementary to the handle. There is one discrete process defined: if a G has an empty site, it will grab the free handle of another G. PEPTIDE will not realize that a complex of two G's is the same as a complex of three and will not shift to the continuous simulation mode to predict that eventually all G's will be connected in a single linear complex.

In fact, there is no way to extend PEPTIDE to recognize all of these new IR quantities and generate the necessary abstractions dynamically, because arbitrarily complex situations can be devised. The chains could include three binders: -A-B-C-A-B-C-, or form lattices in two or more dimensions. In fact, PEPTIDE's representation language is powerful enough to specify an arbitrary type 0 (unrestricted) grammar. Since this implies that we could use the discrete

model to build a Turing machine, it becomes quite clear why PEPTIDE can not always decide whether the simulation will halt.

4.1.2 What Is Being Repeated?

Once the cycle recognizer notices that two PIEs are the same, it still needs to discover what cycle of discrete processes is being repeated. There are two kinds of cycles that could be causing the PIE to repeat: serial and parallel cycles.

* ***Serial Repetition***
  When the two PIEs both require a single molecule that is repeating its actions over and over the cycle is repeating serially. The two bind-sub PIEs, which were declared the same in section 4.1.1.1, are serial because both required a single frobase molecule. Serial cycles usually contain several PIEs, three in this case: bind-sub, react, drop-prod.

* ***Parallel Repetition***
  When the two PIEs modify physically different but otherwise identical molecules, then the PIEs are repeating in parallel. For example, consider a beaker with many A's and the same number of E's. Each A has a bindee view with one handle, and each E has an active site that is complementary to the handle of an A. Assume that one discrete process, called bind-a, is defined--if an E's active site is empty, bind an A. Left alone, eventually all the E's will be bound to A's. Each of the bind PIEs is the same, but the repetition is parallel, because no one molecule binds more than once. Because the PIEs in parallel cycles are independent, all parallel cycles are of length one.[9]

PEPTIDE's cycle recognizer detects both kinds of cycles, serial and parallel; the program differentiates between the two types of cycles by analyzing the history record. Once a newly active PIE is found to be the same as a previously run PIE, the cycle recognizer looks back through the history structure to see if any of the molecules modified by the PIEs are are physically the same (i.e. are connected by

---

[9]Sometimes several parallel cycles start one after another, giving the appearance of greater length.

intervening PIEs).

If the PIEs are connected in the history structure, then the cycle is repeating serially. PEPTIDE marks the first iteration of the cycle as beginning at the first of the two PIEs that were the same followed by the sequence of intervening PIEs.[10]

This is how the recognizer concluded that the cycle in the example in section 4.1.1.1 was composed of three PIES: bind-sub, react, drop-prod.

If the PIEs are not connected in the history structure, then the PIEs are repeating in parallel. PEPTIDE lists the first PIE as the first iteration of the cycle.

4.2 When to Aggregate

PEPTIDE can not always aggregate a cycle of discrete processes into a continuous process immediately after detecting the cycle.

Suppose the cycle recognizer has found a serial cycle of three discrete processes that converts P's into Q's. If that is all there is to the situation, it is safe to aggregate. But if there is another discrete PIE active, PEPTIDE can't know how the PIE will change the system's state without simulating the PIE discretely. Perhaps the PIE will modify some FR quantity that inactivates the previously recognized cycle.

The aggregator can only generate influences from a set of cycles when it knows that all the discrete preconditions to the members of the the cycles are satisfied. It is safe to aggregate when all the active PIEs are members of some recognized cycle, and each cycle has completed an iteration since the time when the

[10]This is not strictly correct, since some of the intervening PIEs could be independent of the cycle and unrepeatable. A more thorough implementation would do a planner-style analysis of the proposed cycle to remove unrelated PIEs.

discrete simulator ran a PIE which is not part of a cycle.[11]

4.3 Pruning Redundant Cycles

There are two kinds of cycles that are redundant and should be pruned before the aggregator generates influences: repeating cycles and subsumed singleton cycles.

4.3.1 Repeating Cycles

Recall the cycle discovered in section 4.1.1.1. There were three processes: bind-sub, react, drop-prod. In this example, the aggregator recognized the (bind-sub, react, drop-prod) cycle. But suppose that there were some other unrelated active PIEs; PEPTIDE would have to continue discrete simulation (looking for more cycles) until the other PIEs stopped, or formed a cycle of their own. After simulating for another three time steps, the cycle-recognizer would find a second cycle: (bind-sub, react, drop-prod, bind-sub', react', drop-prod'). This repeating cycle is clearly redundant and is pruned.

PEPTIDE also prunes cycles which are shifted copies of earlier cycles. If (bind-sub, react, drop-prod) is a cycle, the program will prune a cycle of the form: (react, drop-prod, bind-sub).

4.3.2 Subsumed Singleton Cycles

The problem with subsumed singles is related to that of repeating cycles. Consider what would happen if the example had many frobases instead of one.

---

[11] Even this criteria is not strictly correct. If two cycles with different lengths can deadlock, PEPTIDE might not detect this within the first few iterations. This problem could be solved by noticing when two cycles have intersecting histories (i.e. could possibly deadlock) and simulating discretely for a number of time units equal to the lowest common multiple of the lengths of the two cycles; at this point all possible interactions will have been checked. But this type of compulsive rigor does not seem reasonable given the inherent inaccuracy of the assumption that all discrete processes act at the same speed.

PEPTIDE would start simulation by predicting the effect of bind-sub. One frobase would be bound to one substrate, but many substrates and frobases would still be free. During time step two, react would be active, but so would bind-sub. The aggregator would recognize the singleton cycle (bind-sub) due to parallel repetition, but simulation would continue with the discrete model. At time three, all three processes would be active. At the start of time four, the cycle recognizer would have found four cycles: (bind-sub), (react), (drop-prod), and (bind-sub, react, drop-prod). Since the singleton cycles formed by parallel repetition are subsumed by the serial cycle, the singletons are pruned.

4.4 Influence Generation

For each active cycle, PEPTIDE generates a set of influences that correspond to the net effect of one iteration of the cycle. Since we assume that the time required to perform one iteration of a cycle is roughly the same for every cycle, we can sum the influences for the active cycles to determine the total influences for this set of cycles.

Although this procedure is conceptually simple, complications arise from the need to deduce certain influences and modify others.

4.4.1 The Origin of Influences

Almost all influences are generated by the activity of a process, either a discrete PIE, or a continuous process. Whenever a process is simulated and the system state is updated, a record is made of the influences that have occured. In the interests of simplicity, however, we chose not to model diffusion explicitly; as a result the influences corresponding to diffusional movement are not recorded and must be deduced.

4.4.1.1 Influences From Discrete PIEs

Whenever a discrete PIE is simulated, PEPTIDE records the influences it causes IR quantities such as the size of a group. Currently, an influence is represented by of the quantity being influenced and the direction of movement, but eventually we hope to add a qualitative measure of magnitude to each influence (see section 4.4.2). These influences are generated when PEPTIDE runs the code that enforces generic process postconditions.

4.4.1.2 Influences From Continuous Processes

As discussed in section 4.5, cycles may contain other cycles, and hence continuous processes, as elements. Continuous processes generate influences in much the same manner as do discrete PIEs. However, when aggregating a cycle containing another cycle, it is sometimes useful to shift representations.

When the continuous simulator predicts that the length of a chain will change by a large amount, the natural influence to record is a change in length. But when the aggregator needs to analyze the influences of this continuous process for an enclosing cycle (for example, see section 6.3), it is often useful to consider the new molecule as completely distinct from the original, in which case different influences should be recorded: a decrease in the size of the group containing the original molecule and an increase in the size of the new group.

There are also cases for which it is essential to consider the molecules as the same except insofar as having different lengths. The difference between the two cases seems to hinge on the relation between the molecules being modified in subsequent iterations of the loop. If it is the physically identical molecule with different state, which is being modified both times, then the length influences are likely to be cumulative and it is best to reason in terms of influenced lengths. If the two different molecule's states are equal, however, then there is no cumulative effect, and the aggregator should recognize the net flow of molecules from one group to another, i.e. convert to number influences.

The distinction is clearly related to the question of identity, which is a tricky issue when old molecules get REACTed to form new molecules and new molecules get grouped with old molecules. Consequently, we have not yet implemented logic which automatically selects when to convert length influences to number influences.

4.4.1.3 Deduced Influences from Implicit Diffusion

In the interests of simplicity, PEPTIDE's discrete representation does not explicitly model the diffusion of molecules from one place to another. This movement is allowed to occur implicitly, however: whenever a binder drops off a given position on a chain and later rebinds the same chain at a new position, there has been an implicit influence which must be deduced. The aggregator calculates these influences by scanning through each cycle, looking for ***drop-bind pairs***. A drop-bind pair is a pair of PIEs, 1 and 2, such that PIE1 caused a binder to drop off a chain, PIE2 caused the same binder to reattach the chain, and PIE1 occured before PIE2.

Not all drop-bind pairs cause implicit influences, however. A IR quantity (such as the position of a binder on a chain) can only be influenced if it is defined. If the binder of a drop-bind pair is not bound to its chain when the aggregator is deducing diffusion influences, then the position quantity is not defined and there is no influence. If the position quantity is defined (i.e. the binder is currently bound), the implicit influence is generated by computing the direction between the position before the drop and the position after the bind.

4.4.2 Canceling Influences

Currently, when the aggregator detects two or more influences which are acting on the same quantity, but in different directions, it cancels them. This approach is wrong in principle, but frequently results in the right answer because of the simplicity of our qualitative arithmetic; unless two opposing influences are of different magnitudes, the correct answer is generated. We anticipate rectifying

this problem by adding a more sophisticated influence simplifier which will reason about the magnitudes of various influences when canceling influence pairs.[12]

In addition to opposing influence pairs, regular position influences are canceled when the position quantity is undefined at aggregation time, i.e. when the binder has dropped off the chain.

4.4.3 Example Continued

It is quite simple to generate influences for the cycle detected in our example of section 4.1.1.1. The cycle has three elements all of which are discrete PIEs. The influence generator collects the influences resulting from these PIES:

* Influences from the bind-sub PIE:
  A decrease in the number of free substrates.
  A decrease in the number of free frobases.
  An increase in the number of substrates bound to an E.
  An increase in the number of frobases bound to an A.

* Influences from the react PIE:
  A decrease in the number of substrates bound to an E.
  A decrease in the number of frobases bound to an A.
  An increase in the number of products bound to an E.
  An increase in the number of frobases bound to a B.

* Influences from the drop-prod PIE:
  A decrease in the number of products bound to an E.
  A decrease in the number of frobases bound to a B.
  An increase in the number of free products.
  An increase in the number of free frobases

The aggregator completes the construction of the continuous process by canceling conflicting influences. Once this is done, only the following influences

[12]Ranking influences with magnitudes will also allow PEPTIDE to be able to describe situations where certain processes really are happening faster than others. For example, consider the enzyme E which converts A's to B's, and the enzyme F which converts A's to C's. We would like PEPTIDE to be able to reason about situations where there are many more E's than F's.

remain:

* Final influences generated by aggregator:
  A decrease in the number of free substrates.
  An increase in the number of free products.

4.5 Cycles Within Cycles

PEPTIDE's aggregator is flexible; it can recognize cycles which contain other cycles as members. This ability is quite useful since many examples contain nested cycles, and a simulator which could not aggregate these complex cycles into a continuous process would thrash between discrete and continuous models without determining the final limit. The following example, although artificial, demonstrates PEPTIDE's ability to recognize complex cycles.

Consider a series of processes, catalyzed by enzyme P which converts an A into a B (as in the example of section 1.1). Assume that there is another series of processes, catalyzed by Q, which convert a thousand B's into a single, mammoth C. We define a thousand to be MGT zero and a million to be MGT a thousand. Initially, there are a million A's, one P, one Q, and no B's or C's.

The first series of discrete processes will run converting an A to a B. Then the aggregator will notice that there is a cycle and will produce a continuous process which is decreasing the number of A's and increasing the number of B's. There are two possible limit points: the first set of discrete processes will stop when the number of A's reaches zero, and the second set of processes will start as soon as the number of B's hits a thousand. Since a thousand is closer to zero than a million is, PEPTIDE predicts that the number of B's will go to a thousand. Simulation returns to discrete mode.

At this point both sets of discrete processes are active: one more A is turned into a B while a thousand B's get transformed into a single C. Now only the first set of processes are active.

But instead of recognizing the same cycle as before and again predicting a limit of the number of B's going to a thousand, PEPTIDE detects a larger cycle which consists of the first cycle, the continuous process, another simulation of the first cycle and a simulation of the processes which turned the B's into a C. The regular influences in this cycle are decreasing for the number of A's and B's and increasing for the number of B's and C's. Once the aggregator cancels opposing influences, we are left with a negative influence on the number of A's, and a positive influence on the number of C's. There is only one limit point. So the continuous simulator predicts that all the A's get converted to C's.

At this point PEPTIDE switches back to the discrete simulator, but no more processes are active.

Although this example is somewhat artificial, cycles containing subcycles are quite common when reasoning about chains. See the examples in sections 6.3 and 6.1.

5. Continuous Process Model

While discrete processes are specified by the user before simulation commences, continuous processes are generated dynamically by the aggregator during simulation. Although PEPTIDE's continuous processes were inspired by qualitative process theory [Forbus 83], they are much simpler. We represent a continuous process as a list of ***influences***. Each influence specifies a IR quantity that is changing and the direction of change.

The continuous simulator predicts the effects of a continuous process by computing how much the influences will change the system state before the continuous process reaches a ***limit point***, a value in the range of IR quantities at which the structure of active processes changes, necessitating the end of continuous simulation. The continuous simulator selects the limit point which will be reached first and updates the state to reflect the effect of the influences. Then PEPTIDE returns to discrete simulation mode.

5.1 Generating Possible Limit Points

A limit point is a value in the quantity space of a IR quantity at which a currently-active discrete PIE (i.e. a member of an aggregated cycle) could stop or a currently-inactive PIE could start. Whenever a IR quantity passes a limit point, the structure of discrete process activity changes. Since the aggregated cycle may no longer be repeating, continuous simulation must stop.

5.1.1 Analyzing Process Preconditions

Many limit points can be derived by scanning through the preconditions for all defined discrete processes. Whenever a predicate tests the value of a IR quantity with a constant, the constant is a possible limit point. For example, if a process had the predicate

```
(gt? (number-of glucose) threshold)
```

as one of its preconditions, then theshold would be a possible limit point for the

IR quantity, the number of glucose molecules.[13]

However, the preconditions of some processes are not so simple. The interaction between multiple predicates can cause limit points which are more difficult to detect. Consider the preconditions specified in figure 5-1.

```
(and (bound? ?binder>?site ?chain>?pos)
     (next-pos? ?bpos ?dir ?pos)
     (not (bound? ?blocker>?bsite ?chain>?bpos)))
```

Figure 5-1: Three Predicates Which Together Specify a Limit Point

These predicates are part of the preconditions for a slide generic. The predicates work together to specify that the slide process is active only when the binder is bound to the chain at a position which is not adjacent (in the direction specified) to another binder, blocking the first binder's way.

It is beyond the scope of our research to develop a general algorithm that would detect limit points that result from multiple interacting predicates.[14] To allow the continuous simulator to run real examples which contain these kinds of limit points, we wrote special purpose code which recognizes certain common patterns of multiple preconditions (such as the one in figure 5-1) and generates the correct limit points. However, PEPTIDE will not catch every possible multi-predicate limit point.

For the example defined in section 3.3.5, PEPTIDE would generate potential limit points for each of the IR quantities listed below:

* Number of free substrates: zero.

* Number of substrates bound to frobases: zero.

---

[13] The technique is a bit more complicated than this. Because of the existence of variables, the preconditions must be checked after they have been instantiated in all possible environments.

[14] Qualitative process theory will provide just such a facility [Forbus 83].

* Number of free frobases: zero.

* Number of frobases binding to substrates: zero.

* Number of frobases binding to products: zero.

* Number of products bound to frobases: zero.

* Number of free products: threshold.

5.1.2 Filtering Out Irrelevant Limit Points

Although analysis of process preconditions has produced a large number of potential limit points, most of these are irrelevant. Unless the IR quantity is influenced by the continuous process, no potential limit point for that quantity can be real. In section 4.4.3, the aggregator concluded that this continuous process has only two influences:

* A decrease in the number of free substrates.

* An increase in the number of free products.

This means that of the potential limit points, only two could really be reached during this continuous process. The continuous process will stop if:

* The number of free A's reaches zero, or

* The number of free B's reaches threshold.

In more complex situations, the precondition analyzer will often generate several possible limit points for each IR quantity. When this happens, additional filtering is often possible. If a quantity is being increased by the influence, only the limit points that are greater than the present value for the quantity need be considered. If the possible limit points are ordered, only the point nearest the quantity's present value need be considered.

5.2 Selecting the Closest Limit Point

To determine the effect of the continuous process, PEPTIDE must choose which influenced quantity will reach its limit first. If none of the quantities has a limit point, then the continuous process will continue influencing the system forever, and if more than one quantity has a limit point, then the simulation is potentially ambiguous.[15] To simplify the reasoning, we make the assumption that all quantites are being influenced at the same rate.

The first step in the selection procedure is to determine, for each quantity, the distance between its current value and the limiting value. Next PEPTIDE discovers what the shortest distance is. Since all quantities are being influenced at the same rate, only quantities that are as near to their limit point as is the nearest are contenders; the rest are eliminated.

Before continuing, PEPTIDE checks to see if this shortest distance is qualitatively finite or infinite. If the distance is finite, the program reverts back to the discrete simulation mode to ensure accuracy.

If the distance is qualitatively infinite, PEPTIDE must now choose a limit point. If there is only one possibility, then the choice is clear. If there are several possibilities and they limit IR quantities with different ranges, then there is no way to make an unambiguous choice. However, if all the influenced quantities have the same range (e.g. they are all numbers or all positions on a single chain), then there is a chance that PEPTIDE can make an unambiguous choice. If two influenced quantities, A and B, have the same range and the situation matches one of the possibilities shown in figure 5-2, then PEPTIDE can conclude that A will reach its limit first. Each horizontal line in figure 5-2 represents the range of a IR quantity. The arrows above the line indicate the directions that the two

---

[15] As with the choice between mutually exclusive PIEs, there is a flag which allows the user to manually select the desired limit point.

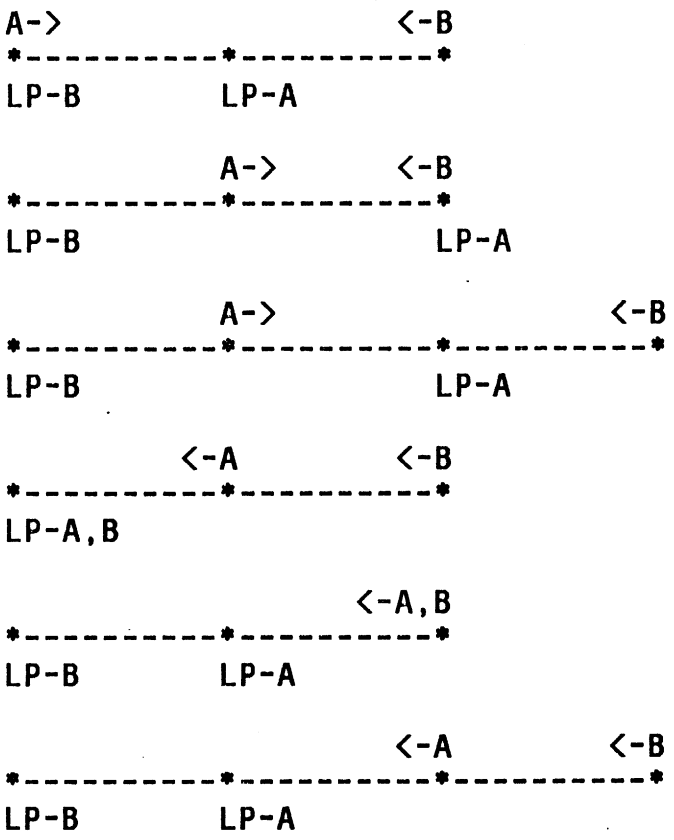

Figure 5-2: Resolvable Limit Analysis Situations

quantities, A and B, are being influenced. Possible limit points are displayed beneath the line. If the situation matches one of the ones in figure 5-3, however, there is no way for PEPTIDE to resolve the ambiguity.

5.3 Simulation

Simulation of a continuous process with a known limit proceeds similarly to simulation of a discrete process (section 3.3.4). PEPTIDE makes new copies of modified groups of molecules and updates the values of the influenced IR quantities. For the quantity going to its limit point, it is easy to determine the new value: it is the limit point. However, it is not clear how far the other quantities will have moved towards their limit points during the interval. In the interest of simplicity we assume that all other quantities move part way towards their limit point (but two quantities never reach their limit at the exact same time).

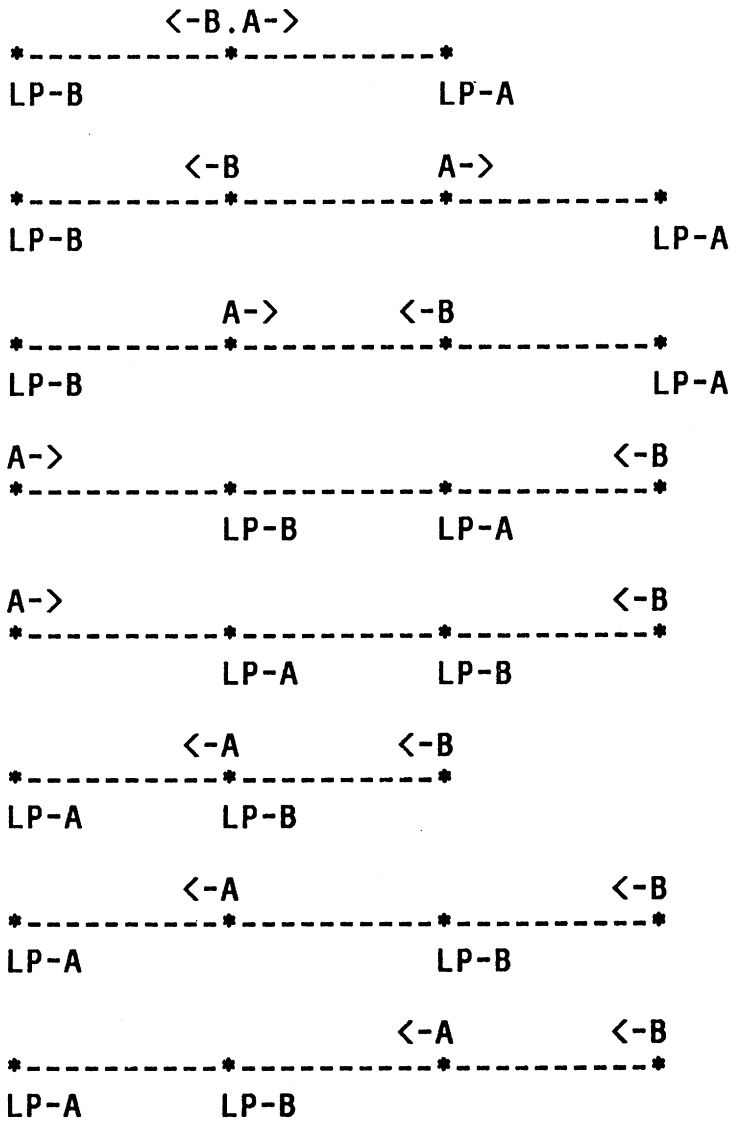

Figure 5-3: Ambiguous Situations for Limit Analysis

5.4 Example Concluded

At last, we know enough to conclude the example that we started in section 3.3.5. The aggregator detected a three member cycle consisting of the bind-sub, react, and drop-prod PIEs, and aggregated it into a continuous process with two influences: a decrease in the number of free substrates and an increase in the number of free products. The limit point generator produces two possible limits--the continuous process will stop whenever the number of substrates reaches

zero or the number of products reaches threshold.

It turns out that PEPTIDE can choose unambiguously between the two possible limit points in this case, because the definition in section 3.3.5 specified that the number million was greater than threshold which was greater than zero. Since there are still nearly a million free substrates and barely more than zero products, PEPTIDE matches the situation with the first pattern in figure 5-2. Assuming that both quantities are influenced at the same rate, the number of products will reach threshold before the number of substrates hits zero.

Once the continuous simulator has made this prediction, it updates the state, and PEPTIDE shifts back to discrete simulation. The situation now contains a group of free substrates with size somewhere between zero and a million molecules, a group of free products with size equal to threshold, and a single free frobase.

At this point the discrete simulator computes that two PIEs are active: bind-prod acting on the frobase and one of the free products, and bind-sub acting on the frobase and one of the free substrates. Because both these PIEs require the sole frobase molecule, only one can run. Since PEPTIDE does not know the relative numbers of free substrates and free products, it has no deterministic way of choosing between the PIEs. For this discussion, we shall assume that bind-prod is chosen. Once bind-prod runs, the situation is as follows: there is one product bound to the control site of the frobase, a large group of free substrates and a group of threshold minus one free products.

Again there are two PIEs active: repress acting on the bound frobase, and bind-sub acting on the frobase and a free substrate. Because the repress PIE is internal to one molecule, it is of higher priority and is chosen to run. The situation that results has an indeterminate number of free substrates, threshold minus one free products, and one product bound to the control site of frobase. The resulting situation is shown is figure 5-4. Since frobase's catalytic site is inactive

as a result of the repress PIE, no more PIEs are active. The simulation is complete.

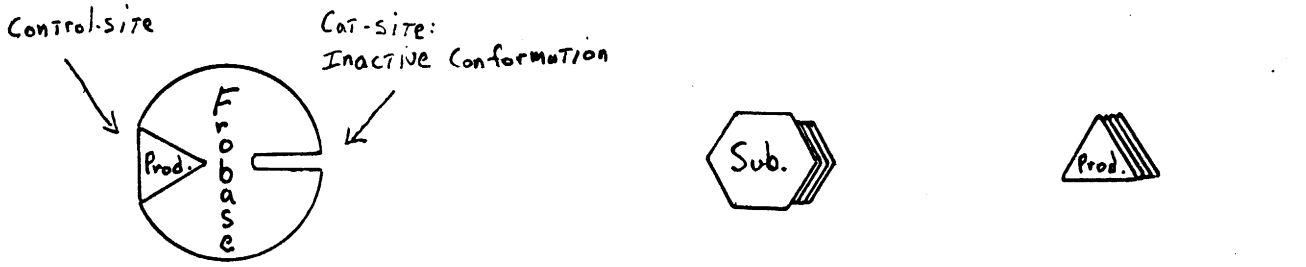

Figure 5-4: Final Frobase Situation

5.5 Deficiencies

PEPTIDE's continuous simulator satisfies its design goals: it is simple, fast and flexible, but it does have deficiencies. Because the technology for the design of continuous simulators is not yet well understood, we felt it wise to accept these problems and concentrate on the issue of linking the discrete and continuous styles of reasoning rather than attempt to fully duplicate the work of Forbus on QP theory [Forbus 83]. In this section we will discuss the two major problems with PEPTIDE's continuous simulator: moving limit points and the lack of loop induction.

5.5.1 Moving Limit Points

Interacting processes can cause situations in which limit points move. Two errors can result from moving limit points: unreal limit points, and erroneous limit points.

For an example of a unreal limit point, imagine a situation consisting of two molecules: a binder and a chain. Assume that the binder was bound to the rightmost end of the chain. Suppose that the aggregator had composed a continuous process from a cycle composed of the following two discrete processes:

1. If the binder is at the rightmost end of the chain, add another unit to

the chain, increasing its length, and leaving the binder attached one position to the left of the right end.

2. If the binder is not at the right end, slide right.

Two influences are caused by this continuous process: the length of the chain is increasing on the right end and the position of the binder on the chain is moving right. If the two processes above are the only ones defined, then infinity is the nearest limit point for the length of chain quantity; the position quantity, however, has a limit point (the end of the chain) that is only 1GT away.

Since PEPTIDE's continuous simulator does not realize that the limit point is moving along with the position and thus the limit will never be reached, it defers to the discrete simulator (the limit point is only 1GT away after all).

For an example of a moving limit point which causes an erroneous prediction, consider a situation with two binder molecules and one chain molecule. Initially the binders are attached on opposite ends of the chain and are sliding towards the center. Rather than conclude that they will collide and stop someplace in the middle of the chain, PEPTIDE's continuous simulator would predict that one of the sliders would go all the way to its limit, the position adjacent to the starting position of the other (right by the opposite end), and the other will go part-way towards its limit, resulting in a position near the middle of the chain. This is a serious error the implementation of our continuous simulator which has no simple solution.[16]

5.5.2 Loop Induction

Since continuous processes are formed from cycles, there are often constraints between quantities being influenced. For instance, in the first example (section 1.1), an enzyme is converting A's into B's. Although PEPTIDE correctly predicts

[16] Limit analysis in QP theory does not suffer from this deficiency.

that all the A's will be turned into B's, the continuous simulator can not conclude that the eventual number of B's is equal to the initial number of A's.

When one attempts to use PEPTIDE to predict the behavior of RNA Polymerase (RNAP), one becomes reminded of this deficiency. RNAP copies a DNA gene into a complementary segment of RNA, unit by unit. Although PEPTIDE's language is fully capable of describing RNAP, and the continuous simulator will predict that the net result is to produce a piece of RNA, it can not recognize that the RNA segment is even the same length as the DNA gene, much less that it is complementary.

We feel that this loop induction problem is a good topic for future research.

6. More Examples

In this section we present three more examples: the degenerative effect of exonuclease, the cannabalistic action of protease, and a simplified, tinker-toy model of transcription of a DNA gene to an RNA copy. All of the examples have been tested and run; PEPTIDE correctly predicts the system's behavior in every case.

6.1 Exonuclease

Exonuclease is an enzyme which degrades pieces of DNA by biting off chunks from one of the ends.

There are only two molecules in this example. DNA has a chain view with one subinterval, gene, made up of nucleotides. Endonuclease has a binder view with one site, active-site, which is complementary to the left end of DNA. There is one piece of DNA and one endonuclease:

```
(def-obj  dna          (chain  (gene)))
(def-obj  exonuclease (binder (active-site conf)))
(def-qrel (complementary exonuclease>active-site>conf
                         dna>?position))
(def-qnum dna one)
(def-qnum exonuclease one)
```

Whenever exonuclease's active site is empty and the left end of DNA is free, exonuclease will try to bind it:

```
(def-process bind
  (if (at-end? ?x left dna)
      (bind exonuclease>active-site dna>?x)))
```

Whenever the left end of DNA is bound to the active site of exonuclease, exonuclease will destroy the leftmost unit of the DNA and ending up floating free with its active site empty:

```
(def-process snarf
  (if (and (bound? exonuclease>active-site dna>?x)
           (at-end? ?x left dna))
      (react exonuclease>active-site
             (*decr-chain left))))
```

At first only a bind PIE is active; so exonuclease binds to the DNA. Then a snarf PIE runs, eating a segment of DNA and causing exonuclease and DNA to unbind. At time 3, a bind PIE is active once more, but before it runs, the cycle recognizer detects a cycle: (bind snarf). Since all active PIEs, i.e. bind, are members of a cycle, the cycle is aggregated, producing one influence: the length of DNA is decreasing from the left hand side.

The continuous simulator realizes that the only limit is when the length of the chain reaches zero, i.e. when the DNA has been completely destroyed. When PEPTIDE simulates this and returns to the discrete simulation mode, no more processes are active.

If we specified that there were many pieces of DNA initially, then the bind PIE would be active when PEPTIDE shifted back to the discrete level. In this case the aggregator would detect a repeating cycle that contained another cycle as a member: (bind snarf continuous-process). Since PEPTIDE knows that a molecule has been destroyed when its length goes to zero, the net influence of this cycle is a decrease in the number of DNA. PEPTIDE correctly predicts that the limit will be reached when all the DNA is gone.

6.2 Protease

Protease is an enzyme (found in the digestive organs) that breaks down proteins. Since protease is itself a protein, it acts like cannibal when left by itself. Although our definition of protease is just as simple as that of exonuclease, its behavior is somewhat more complex.

In this example, there is only one type of molecule, but it has two views. In the bindee view, protease has one handle, bindee. In the binder view, protease has one active site, site. The conformation of the active site is complementary to that of the handle. There are many proteases present initially:

```
(def-obj protease (bindee)
  (binder (site)))

(def-qrel (complementary protease>site>normal-conf
                         protease>bindee))
(def-qrel (mgt million one))
(def-qnum protease million)
```

Whenever protease's active site is empty, it will bind any molecule with a free handle:

```
(def-process bind
  (bind protease>site ?target>?conf))
```

Whenever protease has something bound in its active site, it will detroy it, leaving no remains:

```
(def-process kill
  (if (bound? protease>site ?target>?place)
      (react protease>site *nothing)))
```

For this example environments are important, so we will distinguish between processes and PIEs. We also need to distinguish between groups of protease moleules with different state. The initial, unbound, protease is called protease0.

Initially, only one PIE is active: protease0 can bind another protease0; we shall refer to this PIE as PIE0. After simulating PIE0, the situation contains three groups: many protease0's, and one protease1 which is bound by the active site of one protease2. See figure 6-1 to visualize the various states of protease.

At time two there are several candidate PIEs in contention for the scarce protease1-2 complex: PIE1: protease2 could kill protease1, PIE2: protease1 could bind protease2, PIE3: protease1 could bind a protease0, or PIE4: a protease0

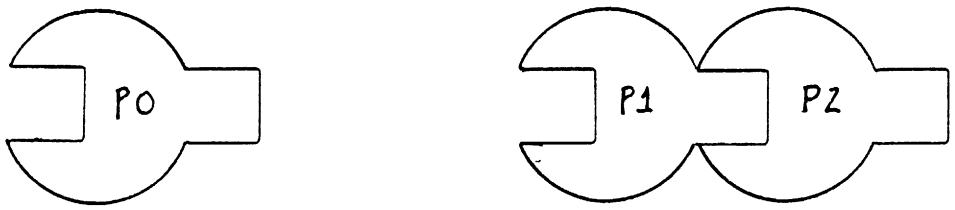

Figure 6-1: The States of Protease0, Protease1-Protease2

could bind protease2. PIE1 and PIE2 have priority two, since they only involve molecules in the connected complex, but the other PIEs have priority three. Since the priority of PIE1 and PIE2 are the same, PEPTIDE chooses between the two at random. Suppose it picks PIE1. Then at time two, protease2 will kill protease1, resulting in another protease0 which gets merged in with the others.

Also at time two, PIE0 is active again: a protease0 will bind another protease0, resulting in new protease1 and protease2 molecules. The cycle recognizer detects this parallel repetition, and produces the single element cycle (<protease0 bind protease0>), but simulation continues discretely because of the activity of PIE1 (above) which is not a member of a cycle.

The result, after simulating time unit two, is many protease0's, and one protease1 bound to the active site of protease2.

At time three PIE0, protease0 binding protease0, is again active. And again there are the same series of PIEs competing for the protease1-2 complex; the two with highest priorities are PIE1, protease2 killing protease1, and PIE2, protease1 binding protease2 to form a doubly-bound complex. Suppose that once again PEPTIDE chooses PIE1.

Before the discrete simulator can simulate the two PIEs, the aggregator notices a cycle which contains both active PIEs: (<protease0 bind protease0> <protease2 kill protease1>). The aggregator computes the net influence of one iteration of

the cycle as a decrease in the number of protease0, and shifts control to the continuous simulator. The continuous simulator determines that the cycle will continue until the number of protease0 is zero, then switches back to the discrete simulator. The situation at this point is: no protease0, one protease1 bound to the active site of protease2.

Again there are two high-priority competing PIEs. Again we will assume that protease2 will kill protease1. Discrete simulation of this PIE results in a state with one protease0 molecule. There are no active processes so simulation is complete. The final result is a single protease0 molecule--all the others have been cannabalized.

6.3 Transcription

Transcription is the process by which cells create RNA copies of their DNA genes. We present a simplified version of that process here. To correctly predict the behavior of this system, PEPTIDE has to recognize cycles which contain other cycles as members.

This example has three types of molecules: DNA has a chain view that consists of one large gene, RNA is a chain of length one, and duplicase is a binder with two active sites, dna-site and rna-site:

```
(def-obj dna        (chain (gene)))
(def-obj rna        (chain first-nuc))
(def-obj duplicase  (binder (dna-site)
                            (rna-site)))
```

Duplicase's dna-site is complementary to the left end of DNA's gene, while the single unit RNA fits in duplicase's rna-site. Initially, there are many short RNA pieces, one strand of DNA, and one duplicase:

```
(def-qrel (complementary duplicase>dna-site>normal-conf
                         dna>left-gene))
(def-qrel (complementary duplicase>rna-site>normal-conf
                         rna>first-nuc))
(def-qrel (gt million one))

(def-qnum duplicase one)
(def-qnum dna one)
(def-qnum rna million)
```

We define four discrete processes. Dup-bind is active whenever both active sites of duplicase are empty, nothing is already bound to the left end of the gene, and there is a free piece of RNA of length one. When dup-bind is active, duplicase grabs the left end of the gene in its dna-site and the short RNA element in its rna-site:

```
(def-process dup-bind
  (if (same-length? (length-of rna) one)
      (bind duplicase>rna-site rna>first-nuc)
      (bind duplicase>dna-site dna>left-gene)))
```

The dup-add-step process is active whenever duplicase is binding DNA at some point on the gene and duplicase is binding RNA *at its right end*. The effect of this process is to increase the length of the RNA chain at the right end, leaving duplicase bound to the RNA one unit left of the right end:

```
(def-process dup-add-step
  (if (and (bound? duplicase>dna-site dna>?x)
           (bound? duplicase>rna-site rna>?y)
           (at-end? ?y right rna))
      (react duplicase>rna-site (*INCR-CHAIN right))))
```

Dup-slide is active when both the gene and the growing RNA chain are bound by duplicase at positions to the left of their right ends. If dup-slide is active, duplicase will move one position to the right on both chains:

```
(def-process dup-slide
  (if (and (bound? duplicase>rna-site rna>?x)
           (not (at-end? ?x right rna)))
      (slide duplicase>dna-site dna right)
      (slide duplicase>rna-site rna right)))
```

Whenever duplicase is bound to the right end of the DNA gene, dup-drop will become active, and duplicase will drop both DNA and the new RNA chain:

```
(def-process dup-drop
  (if (and (bound? duplicase>dna-site dna>?x)
           (at-end? ?x right dna))
      (drop rna>?y duplicase>rna-site)
      (drop dna>?x duplicase>dna-site)))
```

Initially, dup-bind is active, so the discrete simulator predicts that duplicase will grab an RNA fragment in its rna-site and attach to the DNA at the left end of the gene. The resulting duplicase-DNA-RNA complex is shown in figure 6-2; in addition there are many single element RNA fragments still floating free.

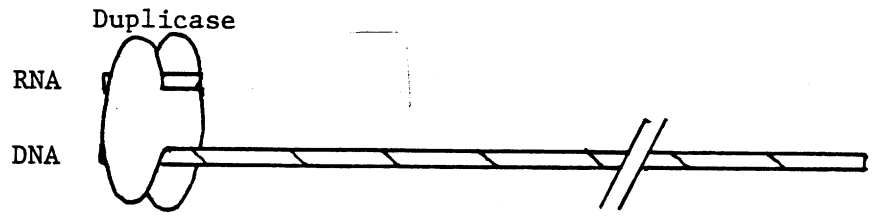

Figure 6-2: Duplicase State After Dup-bind Process

Since the RNA is only one unit long, duplicase must be at the right end and dup-add-step is active. The RNA chain grows to length two, but duplicase is still binding RNA and DNA at their left ends. The situation is as shown in figure 6-3.

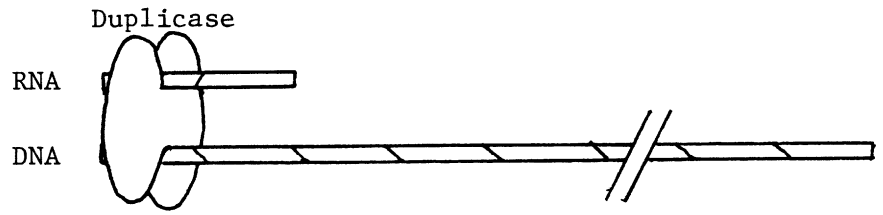

Figure 6-3: Duplicase State After First Dup-add-step Process

At this point, dup-slide is the sole active process, so it is simulated: duplicase moves right one unit on both the DNA and RNA chains. Duplicase is now at the

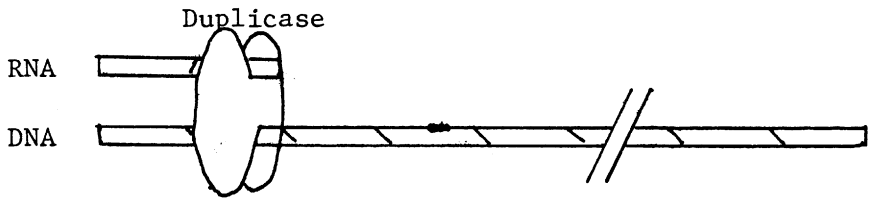

Figure 6-4: Duplicase State After First Dup-slide Process

right end of the RNA, as shown in figure 6-4.

Since duplicase is back at the right end of RNA, dup-add-step is active once more. But before the discrete simulator can predict its effects, the aggregator notices that the current duplicase-DNA-RNA complex is the same as the one two time-steps ago (only the position of duplicase on the chains has changed). We must have a cycle: (dup-add-step dup-slide). The aggregator analyzes one iteration of the cycle and determines that the net influences are: an increase in the length of RNA, an increase in the position of duplicase on RNA, and an increase in the position of duplicase on DNA. Simulation shifts to the continuous mode.

The continuous simulator locates two possible limits: the position of duplicase on DNA could increase until it reached the end of the gene, or the position of duplicase on RNA could increase until it reached the right end of the RNA. Left to its own devices, the continuous simulator would choose the later limit because duplicase is closer to right end of RNA than the right end of DNA.

This choice is incorrect, however, because it does not take into account the fact that the limit point (the right end of the RNA) is moving away from duplicase at the same speed that duplicase is moving towards it.[17] To obtain a reasonable

[17] Fixing this bug would require a complete overhaul of the continuous simulator, duplicating much of Forbus' work on QP theory [Forbus 83]. For more information on the limitations of PEPTIDE's continuous simulator, see the discussion is section 5.5.

simulation the user must manually select the first limit point. Once this is done the continuous simulator updates the state and switches back to discrete simulation. The duplicase-DNA-RNA complex now resembles figure 6-5.

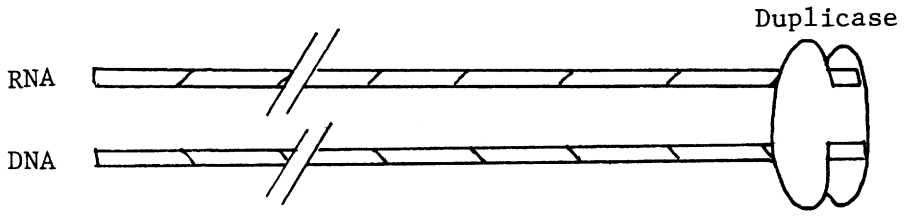

Figure 6-5: Duplicase State After First Limit Analysis

The discrete simulator determines that dup-drop is the sole active process. Simulation of it leaves duplicase, DNA and the new long RNA chain floating free. In addition there are still a large number of short RNA fragments.

Next the discrete simulator realizes that the dup-bind process is active. But before it can be simulated, the aggregator recognizes a cycle. Since the current situation is the same as the initial situation (only the number of RNA molecules has changed), the sequence (dup-bind, dup-add-step, dup-slide, continuous-process, dup-drop) must be a cycle.

Determination of the net influences for this cycle is somewhat difficult. The position influences generated by dup-slide and continuous-process are canceled because neither quantity is currently defined.[18] This leaves a single increasing influence on the length of RNA. As explained in section 4.4.1.2, there are times when length influences must be changed to number influences. This is one of those times.

---

[18] A position quantity is only defined when the binder is actually bound to the chain. That is not currently the case: dub-bind has not been simulated, so duplicase is bound to neither DNA nor RNA. Since the two position quantities are undefined, they can not be influenced.

The net influences for the cycle are: an increase in the long RNA strands, and a decrease in the short RNA fragments. Since there is only one possible limit, the continuous simulator predicts that the resulting situation will contain zero RNA fragments and many long RNA gene transcripts.

When the simulator shifts back to discrete mode, there are no active processes, so the simulation is complete.

7. Related Work

PEPTIDE has been influenced by other programs which do qualitative simulation, existing representations for changing systems, and previous attempts to write programs which use multiple models. However, PEPTIDE has evolved considerably with respect to these influences.

7.1 Qualitative Simulation

There are several reasons to study qualitative simulation. Many aspects of plans can be checked by simulating them qualitatively. Qualitative simulation is also useful when generating explanations, especially when describing the mechanism of a complex device [Weld 83, Forbus 81]. Finally, qualitative simulation can help predict dangers in an underspecified and dangerous world. Consider the can of hair spray that your companion just threw into your blazing living room fireplace. What is your instinctive reaction? Even in the absence of numeric information about the temperature of the fire and the pressure of the hairspray, you can predict that there is likely to be a heat flow from the fire to the contents of the can, and that the pressure of the can is going to rise, perhaps to a value that will cause the can will explode. This kind of qualitative reasoning is clearly useful.

Many people have worked on predicting the behavior of qualitatively-specified systems. As a result, the terminology has become quite confusing. "Envisioning" was introduced in [de Kleer 75] and meant taking the qualitative description of a physical system and predicting its future behavior. Since a qualitative description may be under-constrained, the predicted behavior could be ambiguous. Thus the envisioner returned the list of all possible behaviors [de Kleer 79a]. In recent papers, however, de Kleer and Brown have changed the definitions. In [de Kleer 82a] they describe it as the first step of qualitative simulation: envisioning takes a structural device description and produces a functional description called a causal model (also called an envisionment). Then a technique called running takes the causal model and generates the specific behavior of the device,

completing the qualitative simulation. But in [de Kleer 82b], they say that qualitative simulation is a degenerate form of envisioning; apparently, this is because a form of qualitative simulation is a useful tool for (as well as the goal of) envisioning. To further complicate matters, envisioning is equated with qualitative simulation in [Kuipers 82]. Because of this confusing terminology, we avoid the word. Instead we prefer the phrase, qualitative simulation, which we take to mean the act of producing one possible behavior of a system given its qualitative description.

7.2 Representations of Physical Change

We classify models of change as either discrete or continuous. Discrete models only define the state at the beginning and the end of the change (either because the change takes but an instant, or because the state is undefined during the change). Change takes finite, nonzero time in continuous models and the state is always defined.

7.2.1 Continuous Models

Considerable work has been done on programs which reason about continuous change [de Kleer 75, Rieger 75, Rieger 77, Hayes 83, de Kleer 79b, de Kleer 82a, Patil 81a, Forbus 83, Kuipers 82, Williams 84, Doyle 84]. Our continuous model is based on a simplified version of qualitative process theory [Forbus 83].

Qualitative process theory is a very general, domain-independent framework for representing and reasoning about continuous change. Processes are the most important idea in QP theory; processes only cause change when they are active. A process is defined by specifying a group of ***individuals*** which must be present for activity, a set of ***preconditions*** and ***quantity conditions*** which must all be true for activity, a set of ***relations*** which are true during activity, and a set of ***influences*** which are generated by activity. As the following comparison show, PEPTIDE's continuous processes are very similar to those of QP theory.

* Individuals
In QP theory, the set of required individuals is stated explicitly, but PEPTIDE deduces them during aggregation. Determining the set of individuals is achieved by computing the union of all the groups of molecules that unified part of the pattern of one of the discrete processes that formed an aggregated cycle. Since the cycle of discrete processes is equivalent to a continuous process, they both require the group of individuals.

* Preconditions
QP theory preconditions are conditions that can not be expressed within QP, yet must be true for a process to be active. These conditions refer to structural information, spatial information, and the value of FR quantities (which QP theory does not represent). For the process of fluid flow from one container to another, the preconditions might be the existence of a pipe connecting the containers and a valve being open. Activity of our continuous processes may also be dependent on the condition of FR quantities and structural state, but, unlike QP theory, PEPTIDE has a discrete model which it uses to explicitly model changes in this type of information. The aggregator could produce the set of preconditions for any continuous process by analyzing the discrete processes which form the aggregated cycle. This is not necessary, however, because by detection of a cycle, the aggregator has already verified that the preconditions are true, and (as we discussed in section 1.2) can not be changed by the actions of a continuous process.

* Quantity Conditions
All conditions necessary for process activity than *can* be expressed in QP theory (i.e. conditions that depend on IR quantities) are called quantity conditions. A fluid-flow process might have the following quantity condition: the fluid's pressure at the source must be greater than the pressure at the destination. Activity of PEPTIDE's continuous processes are also often dependent on the values of IR quantities.

* Relations
QP theory allows the specification of relations that are true during the activity of a process. Continuing the fluid-flow example, a new variable called flow-rate might be introduced and declared proportional to the difference in the pressures of the source and destination. In the interests of simplicity, PEPTIDE does not support relations that hold through the activity of a continuous process.

* Influences
  Influences cause change. QP theory allows a process to influence IR quantities by variable amounts. Fluid flow might negatively influence the level of fluid in the source container by an amount proportional to the value of the flow-rate quantity. Influences in PEPTIDE's continuous processes are more restricted. They can increase or decrease any IR quantity, but only at a single rate.

From QP theory we have also taken an important simplifying assumption about the nature of IR quantities--their actual value is irrelevant; all that is important is the value's relation to the distinguished values that are mentioned in quantity conditions (i.e. values at which a process' activity starts or stops). This suggests that IR quantities can be represented with a partial order.

Finally, we use QP theory's technique of ***limit analysis*** as our sole continuous simulation technique. Since the method is fully described in section 5, we will not discuss it here.

QP theory is quite large, and we chose not to implement all of it. Chief amongst the eliminated features are individual views and qualitative proportionalities between quantities.

7.2.2 Discrete Process Models

However, it is not always convenient to consider processes as continuous; some processes (such as collisions) are best considered instantaneous or discontinuous.

Forbus reasons about these discrete changes with ***encapsulated histories***, which are historical records of two sequential processes happening back to back. For example, a collision can be represented by an encapsulated history composed of an interval in which the ball is moving towards an obstacle followed by an interval in which it is moving away. Encapsulated histories are not processes, because although relations from the first process are true during the

first interval and those of the second process hold during the second interval, there is no set of relations that are true during both intervals. Since many of Forbus' techniques depend on relations holding uniformly through a process' active interval, encapsulated histories may not be considered processes. However, Forbus shows how they can facilitate certain types of reasoning such as energy analysis. Forbus' hypothetical analysis of an oscillator closely resembles aggregation.

Since encapsulated histories were not powerful enough to support the types of discrete reasoning that we required, we chose to adopt a primitive set of discrete processes which we could use to model any of the discrete biochemical mechanisms that we expected. We were influenced in this by a general corpus of knowledge (probably founded by the work on STRIPS [Fikes 71]) about the implementation of situation-action rules.

7.3 Multiple Models

There have been a number of programs which used multiple models of the same data to simplify or speed up reasoning. In this section, we shall consider some of these systems and discuss whether the models used the same representation and how the program chose which representation to use.

MACSYMA [MACSYMA 83, Moses 71] uses three different representations for polynomials. By coverting a polynomial into the correct representation for a given manipulation, the program achieves increased efficiency.

The database for the Programmer's Apprentice project [Rich 82] uses two different representational frameworks for program plans, predicate calculus and plan diagrams. As with MACSYMA, the choice of representation is linked to the function being performed. Operations such as maintaining type consistency are easily implemented using the predicate calculus representation, but other operations, such as analysis by inspection, are better performed using plan

diagrams.

ABEL, a program for diagnosing acid base disorders [Patil 81a, Patil 81b], is interesting because it represents patient models with five views of the same data at different levels of abstraction using the *same* representational framework. At each level of the model, causal relations among normal or abnormal states are represented in a semantic network. The models' different levels provide a remarkable breadth of coverage: depending on the level, a causal link could signify an empirical disease association or the interaction between ion transport systems which cause the association. ABEL's control structure is different from the other programs we are discussing. Instead of switching between levels to answer different questions, ABEL diagnoses diseases by building several consistent multilevel models from input data, and then rating them on how well each explains the data.

PEPTIDE represents biochemical processes at a single abstraction level, the tinkertoy level, with two different representational frameworks: discrete and continuous. PEPTIDE's means of switching from one model to another, aggregation, is unique among AI programs, but it resembles an independently-developed, psychological theory of evaporation [Collins 83, Collins 82]. From detailed protocol analysis, Collins and Gentner postulate that human reasoning about evaporation is performed using mental models at three abstraction levels: macroscopic models, aggregate models, and molecular models. Collins and Gentner claim that there are multiple models at each level, and that each high-level model is supported by multiple models at the next lower abstraction level (i.e. a macroscopic model is supported by several aggregate models, etc.) As yet, they have not advanced a mechanism for information flow between models at different levels or a trigger for reasoning shifts from one level to another.

7.4 Other Applications to Biochemistry

MOLGEN and GENEX are two recent AI systems with molecular genetics as a problem domain. MOLGEN [Stefik 81a, Stefik 81b] is a system for planning experiments in molecular genetics. Through the use of hierarchical planning and least commitment operator selection with constraint propagation, it worked for a small number of examples. But MOLGEN uses a very simplified domain model: proteins, for example, are not modeled at all. MOLGEN seems to assume that the biochemical objects it works with are static; there is no representation of processes. Operators are the only causes of change, and they are modeled discretely.

GENEX [Koton 83] hypothesizes operon-level mechanisms of regulation of a mutated system, given a description of the system before mutations and data on observed expressional behavior. The program reasons backwards using high-level rules to determine an initial model of an experimental system. It then attempts to justify this model as a change from the known structure given uncertainties introduced by any mutations. Next it makes a set of assumptions about the nature of the mutations, and then simulates the postulated structure as a test. Since much of GENEX's reasoning is of a high-level diagnostic sort, GENEX uses a large set of empirical production rules. GENEX's forward simulation of proposed structures is less detailed than PEPTIDE's, and consists solely of discrete simulation.

8. Future Work

As is often the case with research, our work on aggregation raises more questions and suggests more possibilities than it resolves.

8.1 Qualitative Simulation of Other Domains

For aggregation to be useful as a qualitative simulation technique, two conditions must hold for the domain in question:

* It must be possible to characterize interesting parts of the domain by continuous processes and IR quantities. Some processes must occur gradually over an interval, and some quantities must have an infinite range of possible values.

* Some processes must repeat their actions cyclically. These repetitious actions can be either discrete, continuous or both.

There may be other domains in which aggregation could be useful. For example, aggregation might be helpful for reasoning about complex machinery. Internal combustion engines have many cyclic processes: the strokes of the pistons, the movements of valves, the rotation of gears and differentials. These repetitious processes affect numerous interesting IR quantities. For example, the friction from each piston's stroke increases the engine temperature, and the spurt of fuel-injected gas lowers the level in the tank and increases it in the combustion chamber. A major advantage of using aggregation to generate the influences on these IR quantities (rather than hard-wiring them in to the domain model) is the ability to model the effects of malfunctioning parts. To simulate the behavior of an engine with broken spark plugs, one could simply substitute a new process description for the plugs.

There are two steps necessary to apply aggregation to a new domain:

* A vocabulary must be selected for describing the primitive processes, discrete and continuous. Although PEPTIDE's vocabulary for molecular-genetics processes consists exclusively of discrete processes, there is no reason why this will be true for other domains.

Some domains may require primitive continuous processes.[19]

* A set of abstractions must be selected for use in the cycle recognizer. Since almost no loop returns the system to *exactly* the same state, we identify cycles by testing the state for equality with respect to certain abstractions. Although it would be nice if the aggregator could learn these abstractions dynamically, a more practical solution is to provide the program with a set that is useful for each domain.

We suggest that useful abstractions can be found for other domains by considering the set of IR quantities. One would probably wish to consider the state of an auto engine to be the same at time t and time s, if the the only difference was the temperature of the engine block. By considering all important IR quantities (e.g. temperature, fuel level, etc.), the critical abstractions will become apparent.

8.2 More Complicated Biochemical Reasoning

There are many possibilities for future work in automated biochemical reasoning even while sticking with the task of qualitative simulation. One promising approach would be to combine PEPTIDE with systems which reasoned at the chemistry and operon levels. The abstraction difference between those levels and PEPTIDE's tinkertoy level presents challenging problems for the control of reasoning and the possibilites for a very powerful reasoner.

---

[19] If PEPTIDE modeled diffusion explicitly, it might be represented by a primitive continuous process.

9. Conclusions

We have attempted to integrate two distinct process models in a program, PEPTIDE, which performs qualitative simulation of tinker-toy models of molecular genetic systems. PEPTIDE uses a discrete process model to reason about changes that occur very quickly, and a continuous process model when dealing with gradual changes that happen over a time interval. A novel technique, called aggregation, has been developed to switch between the two models.

PEPTIDE starts by simulating discrete processes. Whenever the aggregator notices a cycle, PEPTIDE produces a continuous process that has the same effect as multiple iterations of the cycle. Simulation now proceeds at the continuous level as PEPTIDE determines what changes will cause the cycle to terminate. Once the limit analysis is complete, PEPTIDE switches back to the discrete simulator. The use of two process models provides simulation accuracy when it is necessary yet allows PEPTIDE to quickly predict large, gradual changes in the target system.

Since aggregation's model switching powers are critically dependent on the recognition of cycles, the ability to detect when two situations are the same is very important. We define two situations to be the same when they are equal relative to a set of domain dependent abstractions. For the domain of molecular genetics, the abstractions are: the number of molecules, the length of a chain, and the position of a binder on a chain. PEPTIDE's aggregator is so flexible that it can recognize cycles which enclose other cycles.

References

[Collins 82]
Collins, A., Gentner, D.
Constructing Runnable Mental Models.
In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*. August, 1982.

[Collins 83]
Collins, A., Gentner, D.
Multiple Models of Evaporation Processes.
Technical Report, BBN Labs, July, 1983.

[de Kleer 75]
Johan de Kleer.
Qualitative and Quantitative Knowledge in Classical Mechanics.
AI-TR-352, MIT AI Lab, December, 1975.

[de Kleer 79a]
de Kleer, J.
The Origin and Resolution of Ambiguities in Causal Arguments.
In *Proceedings of the Sixth IJCAI*. 1979.

[de Kleer 79b]
Johan de Kleer.
Causal and Teleological Reasoning in Circuit Recognition.
AI-TR-529, MIT AI Lab, September, 1979.

[de Kleer 82a]
de Kleer, J., Brown, J.
Assumptions and Ambiguities in Mechanistic Mental Models.
Cognitive and Instructional Sciences Series CIS-9, Xerox PARC, March, 1982.

[de Kleer 82b]
de Kleer, J., Brown, J.
Foundations of Envisioning.
In *Procceedings of the AAAI*. August, 1982.

[Doyle 84]
Doyle, R.
Learning How Things Work.
Master's Thesis, MIT AI Lab, In Publication, 1984.

[Fikes 71]
Fikes, R., Nilsson, N.
STRIPS: A new Approach to the Application of Theorem Proving to Problem Solving.
Artificial Intelligence 2(3/4), 1971.

[Forbus 81]
Forbus, K., Stevens, A.
Using Qualitative Simulation to Generate Explanations.
Technical Report 4490, Bolt Beranek and Newman Inc, March, 1981.

[Forbus 83]
Forbus, K.
Qualitative Process Theory.
AI Memo 664A (revised), MIT AI Lab, May, 1983.

[Habermann 76]
Habermann, A.
Introduction To Operating System Design.
Science Research Associates, Inc., 1976.

[Hayes 83]
Hayes, P.
The Second Naive Physics Manifesto.
URCS 10, University of Rochester Cognitive Science Department, October, 1983.

[Herskowitz 80]
Herskowitz, I., Hagen, D.
The Lysis-Lysogeny Decision of Phage Lambda: Explicit Programming and Responsiveness.
Ann. Rev. Genetics 14:399-445, 1980.

[Koton 83]
Koton, P.
A System to Aid in the Solution of Problems in Molecular Genetics.
MS Thesis, MIT Laboratory for Computer Science, May, 1983.

[Kuipers 82]
Kuipers, B.
Getting the Envisionment Right.
In *Proceedings of the AAAI.* August, 1982.

[MACSYMA 83]
The Mathlab Group.
MACSYMA Reference Manual.
MIT Laboratory for Computer Science, 1983.

[Moses 71]
Moses, J.
Algebraic Simplification: A Guide for the Perplexed.
Communications of the ACM 14(8), August, 1971.

[Patil 81a]
Patil, R.
Causal Representation of Patient Illness for Electrolyte and Acid-Base Diagnosis.
TR-267, MIT Laboratory for Computer Science, October, 1981.

[Patil 81b]
Patil, R., Szolovits, P., Schwartz, W.
Causal Understanding of Patient Illness in Medical Diagnosis.
In *Proceedings of the Seventh IJCAI.* 1981.

[Rich 80]
Rich, C.
Inspection Methods in Programming.
AI-TR-604, MIT AI Lab, June, 1980.

[Rich 82]
Rich, C.
Knowledge Representation Languages and Predicate Calculus: How to Have Your Cake and Eat it Too.
In *Proceedings of the AAAI*, pages 193-196. 1982.

[Rieger 75]
Rieger, C.
One System for Two Tasks: A Commonsense Algorithm Memory that Solves Problems and Comprehends Language.
Working Paper 114, MIT AI Lab, November, 1975.

[Rieger 77]
Rieger, C., Grinberg, H.
The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms.
In *Proceedings of the Fifth IJCAI.* 1977.

[Stefik 81a]
Stefik, M.
Planning with Constraints (MOLGEN: Part 1).
A. I. Journal , 1981.

[Stefik 81b]
Stefik, M.
Planning and Meta-Planning (MOLGEN: Part 2).
A. I. Journal , 1981.

[Watson 77]
Watson, J.
Molecular Biology of the Gene.
The Benjamin/Cummings Publishing Company, Menlo Park, CA, 1977.

[Weld 83]
Weld, D.
Explaining Complex Engineered Devices.
Technical Report 5489, Bolt Beranek and Newman Inc, November, 1983.

[Williams 84]
Williams, B.
Qualitative Analysis of MOS Circuits.
Artificial Intelligence , To Appear, 1984.

[Wood 81]
Wood, W., Wilson, J., Benbow, R., Hood, L.
Biochemistry, A Problems Approach.
The Benjamin/Cummings Publishing Company, Menlo Park, CA, 1981.

This blank page was inserted to preserve pagination.

CS-TR Scanning Project
Document Control Form

Date : 2/6/96

Report # AI-TR-793

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 81(87-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☒ Other:_____

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☒ Cover Page
☒ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description :	Page Number:
IMAGE MAPS	(1-81)
UNN'ED TITLE PAGE	i-ii
UNN'ED ACKNOWLEDGEMENTS	1-77
SCANCONTROL, COVER, SPINE, TARGETS	(82-87)

Scanning Agent Signoff:
Date Received: 2/6/96 Date Scanned: 2/7/96 Date Returned: 2/15/96

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 2/6/1996

M.I.T. Libraries
Document Services