COMBINING DISCRETE AND CONTINUOUS PROCESS MODELS

Daniel S. Weld

Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

ABSTRACT

Two notions of process have been used in programs that reason about change* discrete models, which represent changes as instantaneous, and continuous models, which represent changes as processes that act gradually over time. We describe a technique called aggregation which unifies the two types of models and allows both to be used when performing qualitative simulation.

Aggregation is a technique for recognizing cycles of processes and generating a continuous process that is equivalent to each cycle. A qualitative simulator may start prediction with a simple discrete model and use aggregation to generate a continuous process model when discrete simulation bogs down in a cycle. Aggregation thereby allows a simulator to switch back and forth between different types of models depending on which type is most expedient.

To test these ideas, we have written a program, PEPTIDE, which performs qualitative simulation in the domain of molecular genetics. The flexibility of PEPTIDE'S aggregator allows the program to detect cycles within cycles and predict the behavior of complex situations.

I INTRODUCTION

A number of Al programs perform some type of qualitative simulation – they predict the future behavior of a system in qualitative terms Why is this an interesting problem? Any program which does planning or problem solving, needs to predict the effects of proposed actions; qualitative simulation provides a useful description since it removes often irrelevant numeric detail. Qualitative simulation is also useful when generating explanations, especially when describing the mechanism of a complex device [2,12,6,1,13,15]. Finally, qualitative simulation is the *only* way to predict the behavior of the many systems which are incompletely specified.

To simulate a system, a program needs to represent the processes that cause change. Two fundamentally different qualitative models of change have been developed: discrete and continuous process models. STRIPS [5] illustrates a discrete model, in which actions are atomic; each action has add and delete lists which define the action's effect on the state by an abrupt change Qualitative Process Theory [7], on the other hand, employs continuous process models, in which actions take time and gradually change the value of quantities such as the level of fluid in a container.

In this paper we discuss the factors that make 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* The essential motivation behind aggregation is that processes often repeat--consider the cycles of an internal combustion engine When reasoning about repeating cycles of processes, it is frequently useful to eliminate irrelevant detail by considering the whole cycle as a single, more abstract process This is what an aggregator does: it recognizes cycles of discrete and continuous processes, and it generates a continuous process abstraction of the cycle A program called PEPTIDE has been written to test this idea in the domain of molecular genetics Implemented in Zetalisp on a Symbolics 3G00, PEPTIDE has run a dozen examples including the one in this paper and several thai are considerably more complex [14].

II REPRESENTATION

The operation of any simulator is crucially dependent on the representations used to model quantities of interest and the processes which cause change. We describe our assumptions about these representations below.

A. Quantities

The system parameters that the simulator models are called quantities, We distinguish between two types of quantities' *linear* and *nominal.* Linear quantities have an associated total order while nominal quantities represent categories with no internal structure. For example, the charge of a battery and the strength of the current through a circuit could be modeled with a linear quantity, but the state of a switch in the circuit might be better represented by a nominal quantity with two possible values: open and closed.

B. Processes

We assume that time is composed of both *instants* and *intervals.* Instants a"e infinitesimal, while intervals are extended. Between any two non-oveilapping intervals lie one or more instants. The distinction between instants and intervals allows us to classify types of change. If a change is best considered as occuring in an instant, we model it with a *discrete process.* If the change occurs

Thin report describes research done at the Artificial Intdligence Laboratory of the Massachusetts Institute of Technology Support for the laboratory's Artificial Intelligence: research is provided in part by the Advanced Research Projects Agency ct the Department of Defense under Office of Naval Research contract N00014 75-C-0643 and the Office of Naval Research under contract number N0014-80-C-0505.

It should be emphasized that we do not attempt to extend either model. Considerable work is already being done on that front. [3,4,8,11. 16].

over an interval, we use a *continuous process*

Discrete processes are appropriate when modeling changes in nominal quantities (e.g. the flip of a circuit's switch) and abrupt changes in linear quantities (e.g. the change in the current through a circuit following the switch's flip) It is often useful to assume that discrete processes are atomic- they either happen fully or not at all. Atomicity is useful when modeling situations where mutually exclusive actions are possible [0] Situation action rules are an efficient means for representing discrete processes.

Continuous processes, on the other hand, are appropriate when modeling gradual, strictly monotonic changes in linear quantities. It is important to note that continuous processes can't affect nominal quantities because there is no way to monotonically change a quantity with no inherent order. Continuous processes are ideal for describing the change that occurs when a battery is connected through a resistor to ground. We define continuous processes by a set of *preconditions* and a set of *influences.* The preconditions are predicates on quantities that tell when the process is active, i.e. when the influences are enfoiced. The influences say how various quantities will be modified by the activity of the process. The electric current process would be active whenever the switch was closed and the battery's voltage was greater than zero. The process would have two influences: a decrease in the charge in the battery and an increase in the temperature of the resistor.

The power of continuous models comes from a technique called *limit analysis* [8]. When a simulator performs limit analysis, it takes influences of a continuous process and predicts when these influences will cause the preconditions to be violated or the preconditions of another, inactive process to become satisfied. Thus, limit analysis predicts that the electric current process will be active until the battery's voltage drops to zero.

III AN EXAMPLE OF AGGREGATION

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. Since discrete processes are simpler, we make them the default. We only use continuous processes when the simulator needs the power of limit analysis, i.e. when there are repeating cycles.

The system starts by simulating the effects of discrete processes until it recognizes a cycle. Then the aggregator produces a continuous process with the same preconditions as the cycle's, and with influences equivalent to the net result of a single iteration of the cycle. At this point the continuous simulator can use limit analysis to predict the eventual result without laboriously simulating each iteration of the loop.

To make this clear, we present a simple example in the domain of molecular genetics.

A. Example Definition

Consider a container with two individuals, E and DNA.*" DNA is a long chain, and E has a large cleft (Figure 1).

Figure 1: Initial Situation

We define two discrete processes:

•BIND

If there is an E whose cleft is empty and there is a DNA with nothing bound to its right end, then the E will bind the right end of the DNA.

*SNARF

If there is an E bound to the right end of a DNA, then the E will digest the rightmost segment of the DNA. The result will have E floating free, and the length of DNA one shorter than before.

B. Discrete Simulation

Initially the process BIND is active, and SNARF is not. PEPTIDE creates an instance of BIND that is associated with E and DNA and simulates it. The resulting situation is shown in Figure 2.

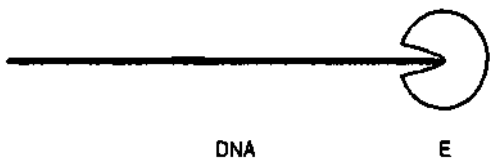

Figure 2: Situation After E Binds DNA

Then the process SNARF is active; an instance is created and simulated. Figure 3 shows the result.

Next the process BIND is again active, but before the discrete simulator can predict its effects, the aggregator recognizes a cycle.

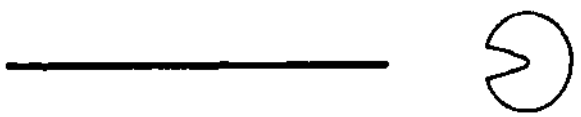

Figure 3: Situation After E Digests a Section

C. Cycle Recognition,

The notion of a repeating cycle of processes implies that the process instances in each iteration are "the same" in some sense. For the aggregator to recognize cycles, it must be able to detect when two processes are the same. But sameness is not equivalent to equality, as the example above shows. The two instances of the BIND process are not equal since they involve DNA chains of differing length, but the processes are clearly the same in an important sense.

We define two instances of a process to be the *same* if each of the individuals associated with the two instances are the same. Two individuals are the same if all their nominal quantities are equal; their linear quantities can be different. Thus the two instances of BIND are seen to be the same because the only difference is in the

For clarity of presentation, the example is described graphically. The appendix contains the actual PEPTIDE description

length of DMA, and length is a linear quantity. Ignoring differences in linear quantities makes sense when looking for cycles, because we wish to consider cycles as continuous processes which cause changes in linear quantities but not in nominal quantities. For the domain of molecular genetics, there are two other linear quanties (besides the length of a chain) which are important: the number of identical molecules in a group, and the position of a binder on a chain.

Once two processes instances have been recognized to be the same, PEPTIDE searchs through a history record [10] of the past simulation to determine exactly which processes are in the cycle [14]. In the case of our example, this is quite easy; the cycle is simply (BIND SNARF).

D. Generation of a Continuous Process

A continuous process consists of a set of preconditions and a set of influences. First, the aggregator analyzes one iteration of the cycle to determine the influences. For the simple (BIND SNARF) cycle, it is easy to determine that there is only one influence: a decrease in the length of DNA. Complex cycles, however, can require more sophisticated analysis (e.g. resolution of contradictory influences [14]).

To generate the continuous process' preconditions, the aggregator simply combines from each of the discrete processes in the cycle the preconditions that refer to linear quantities; nominal preconditions cancel out. For our example, this results in only two preconditions:

* There must be at least one free DNA.

• There must be at least one free E.

E. Continuous Simulation

Finally, the continuous simulator can predict the eventual result of the repeating cycle by performing limit analysis. Since PEPTIDE knows that if the length of a chain is ZERO the individual doesn't exist, it is easy for the continuous simulator to predict that eventually DNA will be gone and only E will remain. At this point the simulator summarizes the net effect of the continuous process and adds it to the history structure with the name CP1. Then PEPTIDE switches back to discrete mode. Since DNA is gone, no processes are active; thus simulation is complete.

F. Cycles Within CYCleS

If we had specified that initially there were *many* DNA individuals in the container, then there would still be DNAs present and BIND would be active. But before it could be simulated discretely, the aggregator detects that this BIND is the same as the first, since the only difference between the initial situation and the current one is a change in the linear quantity, the number of identical DNA molecules. A search of the history structure reveils the following cycle: (BIND SNARF CP1). The influence of this new cycle is a decrease in the number of DNAs. Continuous simulation of this cycle again results in the conclusion that eventually all the DNAs will be eaten.

The occurrence of nested cycles in this simple example demonstrates their ubiquity; it is essential that an aggregator be capable of handling them. The key point is that when aggregating, PEPTIDE considers each previous continuous processes (i.e. whole cycles of processes) as a single discrete process. Without this ability the aggregator could not use the information from previous aggregations when constructing the continuous process for the outside cycle.

IV LIMITATIONS AND FUTURE WORK

Although aggregation is a powerful technique, it can't detect all cycles, and should be tested on more domains.

A. Failure to Recognize all Cycles

As explained above, the aggregator recognizes cycles by ignoriny the value of linear quantities. Although this method works in many cases, it fails when new linear quantities are generated dynamically.

We have not discovered any cases where this happens naturally in the domain of molecular genetics, but we can create artificial pathological situations. This is unsurprising because 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't always decide whether simulation should halt.

B. 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 linear quantities.

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

We suspect that aggregation could be useful in many other domains. Digital electronics is a natural domain in which aggregation could be applied. There are many devices at different levels of detail whose states are usefully considered as linear quantities: counters, LIFO queues, and processor pipelines. Furthermore, many processes repeat cyclicly Detailed paper simulations suggest that aggregation could be usefully applied to this domain.

Another potentially fruitful domain is that of complex machinery. Internal combustion engines have many repetitious processes: the strokes of the pistons, the movements of valves, the rotation of gears and differentials. These cycles affect numerous interesting linear 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.

We feel that it is very important to test the utility of aggregation for prediction, instruction, and diagnosis in another domain.

Both discrete and continuous process models have their strengths. Discrete process models are simple, and can model changes in nominal as well as linear quantities. Continuous process models facilitate limit analysis of gradual changes in linear quantities. We described a technique, called aggregation, which

allows a qualitative simulator to use either model depending on which best fits the situation.

The aggregator recognizes cycles of processes by ignoring the values of linear quantities and generates a continuous process abstraction of the cycle by deducing the nut influences of one iteration of the cycle. A test program, PEPTIDE, has used aggregation to solve a dozen problems in the domain of molecular genetics. The flexibility of PEPTIDE'S aggregator allows the program to predict the behavior of complex systems by detecting cycles nested within other cycles.

ACKNOWLEDGMENTS

I owe many thanks to my advisor, Randy Davis; to Ken Forbus, Dave Chapman, and Brian Williams who contnbuted greatly; to Bruce Donald for insightful comments on a draft, and to the many other people who form the stimulating environment of the MIT Al Lab.

APPENDIX

The actual PEPTIDE code for the example (with only formatting changes) is displayed in Figure 4. Capitalized words are keywords defined by PEPTIDE.

```
(DEF-EXAMPLE exonuclease
  (DEF-INDIVIDUAL dna          (CHAIN))
  (DtF-IMDIVIDUAL exonuclease (BINDER (cleft)))

  (DEF-PROCESS bind
    (IF (AT-END? ?x RIGHT dna)
        (BIND oxonuclease>cleft dna>?x)))
  (DEF-PROCESS snarf
    (IF (AND (BOUND? exonuclease>clef t dna>?x)
             (AT-END? ?x RIGHT dna))
        (REACT exonuclease>cloft dna>?x
               (•DECR-CHAIN right))))

  (ASSERT (COMPLEMENTARY exonuclease>cleft
                         dna>?a))
  (ASSERT (EQ ONE (NUMBER-OF dna)))
  (ASSERT (EQ ONE (NUMBER-OF exonuclease))))
```

Figu re 4: PEPTIDE Code for the Example

REFERENCES

1. Brown, J., Burton, R., de Kleer, J. Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPIE I, II, and III. In Sleeman, D., Brown, J., Ed., *Intelligent Tutoring Systems,* Academic Press, 1982.

2. de Kleer, J. Causal and Teleological Reasoning in Circuit Recognition. MIT Al Lab, September, 1979.

3. de Kleer, J. and Brown, J. A Qualitative Physics Based on Confluences. *Artificial intelligence* (December 1984).

4. de Kleer, J. and Brown, J. The Origin, Form and Logic of Qualitative Physical Laws. Eighth International Joint Conference on Artificial Intelligence, IJCAI, August, 1983.

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

6. Forbus, K., Stevens, A. Using Qualitative Simulation to Generate Explanations. Tech. Rep. 4490, Bolt Beranek and Newman Inc, March, 1981.

7. Forbus, K. Qualitative Process Theory. Al Memo 664A (revised), MIT Al Lab, May, 1983.

8. Forbus, K. Qualitative Process Theory. MIT Al Lab, October, 1984.

9. Habermann, A. *Introduction To Operating System Design.* Science Research Associates, Inc., 1976.

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

11. Kuipers, B. Getting the Envisionment Right. Proceedings of the AAAI, August, 1982.

12. Stevens, A., et. al. STEAMER: Advanced Computer Aided Instruction in Propulsion Engineering. Tech. Rep. 4702, Bolt Beranek and Newman Inc, July, 1981.

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

14. Weld, D. Switching Between Discrete and Continuous Process Models to Predict Genetic Activity. MIT Al Lab, October, 1984.

15. Williams, B. Qualitative Analysis of MOS Circuits. *Artificial Intelligence* (December 1984).

16. Williams, B. The Role of Continuity in a Qualitative Physics. Proceedings of the AAAI, August, 1984.