COGNITIVE SCIENCE 6, 101–155 (1982)

A Temporal Logic for Reasoning About Processes and Plans*

DREW MCDERMOTT
Yale University
Department of Computer Science

Much previous work in artificial intelligence has neglected representing time in all its complexity. In particular, it has neglected continuous change and the indeterminacy of the future. To rectify this, I have developed a first-order temporal logic, in which it is possible to name and prove things about facts, events, plans, and world histories. In particular, the logic provides analyses of causality, continuous change in quantities, the persistence of facts (the frame problem), and the relationship between tasks and actions. It may be possible to implement a temporal-inference machine based on this logic, which keeps track of several "maps" of a time line, one per possible history.

1. INTRODUCTION

A common disclaimer by an AI author is that he has neglected temporal considerations to avoid complication. The implication is nearly made that adding a temporal dimension to the research (on engineering, medical diagnosis, etc.) would be a familiar but tedious exercise that would obscure the new material presented by the author. Actually, of course, no one has ever dealt with time correctly in an AI program, and there is reason to believe that doing it would change everything.

Because time has been neglected, medical diagnosis programs cannot talk about the course of a disease. Story understanding programs have trouble with past events. Problem solvers have had only the crudest models of the future, in spite of the obvious importance of future events.

*This research was supported by NSF grant MCS 8013710.

*Thanks to Ernie Davis for technical assistance and ideas; and to Chris Riesbeck and all the members of the Yale Learning Group, who came up with problems for a temporal notation in the field of economics; and to Tony Passera for work on the implementation. I had useful discussions with James Allen, Eugene Charniak, Patrick Hayes, and Robert Moore. The referee is responsible for some improvements in intelligibility. I am responsible for residual confusion and error.

Many researchers have compensated by modeling the course of external time with the program's own internal time, changing the world model to reflect changing reality. This leads to a confusion between correcting a mistaken belief and updating an outdated belief. Most AI data bases have some sort of operator for removing formulas. (e.g., ERASE in PLANNER, Hewitt, 1972) This operator has tended to be used for two quite different purposes: getting rid of tentative or hypothetical assertions that turned out not to be true, and noting that an assertion is *no longer* true. The confusion is natural, since some of the same consequences must follow in either case. For example, if "The car is drivable" follows from "There is gas in the car," then the former statement must be deleted when the latter is, whether you have discovered there to be no gas after all, or the gas has been used up.

But in many cases, the two behave quite differently, and efforts to make them the same have resulted in awkward, inextensible programs. For example, from "x is beating his wife," you are entitled to infer, "x is a bad man." But if x pauses to catch his breath, only the former statement must be deleted from the data base. Clearly, the proper inference is from "If x has beat his wife recently, he is a bad man," and "x is beating his wife," to "For the next year or so, x will have beaten his wife recently," and hence to "For the next year or so, x is a bad man." (We must allow for reform.) As far as I know, no AI program has been capable of such inferences.

An even worse flaw than the inability to model present change is the inability to model future possibility. To make this clear, I will sketch an example of where the standard approaches fail.

Say a problem solver is confronted with the classic situation of a heroine, called Nell, having been tied to the tracks while a train approaches. The problem solver, called Dudley, knows that

> "If Nell is going to be mashed, I must remove her from the tracks."

(He probably knows a more general rule, but let that pass.) When Dudley deduces that he must do something, he looks for, and eventually executes, a plan for doing it. This will involve finding out where Nell is, and making a navigation plan to get to her location. Assume that he knows where she is, and she is not too far away; then the fact that the plan will be carried out is added to Dudley's world model. Dudley must have some kind of data-base-consistency maintainer (Doyle, 1979) to make sure that the plan is deleted if it is no longer necessary. Unfortunately, as soon as an apparently successful plan is added to the world model, the consistency maintainer will notice that "Nell is going to be mashed" is no longer true. But that removes any justification for the plan, so it goes, too. But that means "Nell is going to be mashed" is no longer contradictory, so it comes back in. And so forth.

Exactly what will happen depends on implementation. The data base manager might loop forever, or it might conclude erroneously that Nell is safe without any action by Dudley. The problem, however, lies deeper than the implementation level. The naive logic we used, a non-monotonic first-order situation calculus (McCarthy, 1968; McDermott, 1980a), is just inadequate: no implementation can do the right thing here, because the logic doesn't specify the right thing. We need to be able to express, "Nell is going to be mashed *unless* I save her," and *unless* is a non-trivial concept (Goodman, 1947; Lewis, 1973).

In this paper, I will begin an attempt to rectify these problems, by providing a robust temporal logic to serve as a framework for programs that must deal with time. This is in the spirit of Hayes's "naive physics" (Hayes, 1979a), and might be thought of as a "naive theory of time." I will sketch approaches within this framework to what I consider the three most important problems of temporal representation: causality, continuous change, and the logic of problem solving.

One difference between Hayes and me is that I have not been able to turn my eyes away from implementational details as resolutely as Hayes. Consequently, later in the paper I will discuss how these ideas might be embodied in a program. Of course, the use of logic does not constrain us to making the program look like a theorem prover.

So why do I plan to spend any time at all on logic? There are two reasons:

1. We want to be assured that our special-purpose modules are not prone to absurd interactions such as the one I just sketched. One way to guarantee this is to be sure that the modules' actions are sound with respect to an underlying logic. (It is relatively unimportant and in practice unattainable that the programs be logically complete.)
2. Recently it has become clear that a reasoning system must keep track of the justifications for its conclusions, in order to update the data base as assumptions change (Doyle, 1979). For example, a picture of the future based on the assumption that dinner will be done at 6:00 must be revised if there is a power failure at 5:30. It turns out that constructing and maintaining these justification records, called *data dependencies,* is not trivial. One useful guide is that the data dependencies be equivalent to proofs in the underlying logic.

Many cognitive scientists will not find these reasons reassuring enough. On the one hand, many of them will be intimidated by the use of logical notation. On the other, there is a widespread feeling that psychological experiments have proven that people cannot handle simple syllogisms (see, e.g., Johnson-Laird, 1980), and that, therefore, people cannot possibly operate on logical principles. Together, these considerations cause them to reject papers like this one out of hand.

Let me be a little more reassuring. There is no difference between logical notation and notations like those of Schank (1975) or Charniak (1981), except emphasis. The logical approach aims at expressing the implications used for *inference,* as well as providing an ontological framework (or set of primitives, or vocabulary) for expressing *facts.* But face it—we're all talking about computers performing formal operations on data structures representing beliefs. The only issue is which to nail down first, the organization of the information in memory, or the structure of the inferences.

The experimental results on human processing of syllogisms are much less relevant than they first appear. At best, they show that people have no natural syllogistic machinery accessible to consciousness. This says nothing about logics underlying various kinds of thinking. One might as well investigate frequency-domain analysis in the visual system by asking people to do Fourier transforms in their heads.

In any case, I hope that appreciation of the difficulties raised by time will cause you to stick with me.

2. ONTOLOGY

We shall be doing logic in the style of Robert Moore (1980). The logic of time appears at first glance to be like modal logic, with different instants playing the role of different possible worlds. An expression like "President of the US" seems to denote an intensional object, with a different denotation in different times (worlds). In fact, historically the exploration of this relationship has fueled temporal logic (Prior, 1967; Rescher, 1971).

Moore encountered a similar tradition in his study of knowledge. "Know" had typically been taken as a modal operator. This made it difficult to handle computationally (Moore, 1980). Moore's contribution was to work with a first-order, extensional language that described the *interpretation* of the original modal language. He retained the original modal language as a set of objects manipulated by the first-order semantic language.

We will carry this idea one step further and dispense with the object language altogether, although some of the terminology will hint at vestiges of it. We will talk about a temporal model using a first-order language. The resulting enterprise will look like a hybrid of Moore's work and that of Hayes (1979a).

There are two key ideas to capture in our logic: the "openness" of the future, and the continuity of time. The first idea is that more than one thing can happen starting at a given instant. We model this by having many possible futures. The second idea is that many things do not happen discontinuously. We model this by having a continuum of instances between any two instants. It will be clear eventually why these features are so important.

To capture these ideas, our language will talk of an infinite collection of *states* of the universe. A state is an instantaneous snapshot of the universe. States are partially ordered by a relation "=<." We write (=< s1 s2) to mean that s1 comes before or is identical to s2.

I use "Cambridge Polish" notation for logical formulas. Every term, atomic formula, and combination is of the form (*p*...), where *p* is a function, predicate, or connective. The rest of the formula after *p* will be the arguments or other subparts. If *p* is a quantifier ("forall" or "exists"), then the subparts are a list of variables and a formula:

(forall (-*vars*-) *fmla*)
(exists (-*vars*-) *fmla*)

For other connectives, the subparts are formulas, as in

(not *fmla*)
(if *fmla1 fmla2*)
(and *fmla1 fmla2* ...)
(or *fmla1 fmla2* ...)
(iff *fmla1 fmla2*)

If *p* is a binary transitive relation, (*p w x y*...*z*) is an abbreviation for (and (*p w x*) (*p x y*) ... (*p* ... *z*)). I will generally use lower case for logical constants; upper case for sorts (which I will discuss shortly), for Skolem constants, and for domain-dependent predicates and functions; and italics for syntactic variables.

```
Axiom 1:  (iff (and (=< ?s1 ?s2) (=< ?s2 ?s1)) (= ?s1 ?s2))
          (iff (< ?s1 ?s2) (and (=< ?s1 ?s2) (not (= ?s1 ?s2))))
```

As usual, if (=< s1 s2) and s1 and s2 are distinct, we write (< s1 s2).

```
Axiom 2:  (Density)
  (forall (s1 s2)
    (if (< s1 s2) (exists (s) (< s1 s s2))))

Axiom 3:  (Transitivity)
  (forall (s1 s2 s3)
    (if (and (=<s1 s2) (=<s2 s3))
      (=< s1 s3)) )
```

Notice that I assume a sorted logic. Variables beginning with s are states. All this means is that a formula (forall (*x*) *p*), where *x* is a sorted variable, is an abbreviation for

(forall (*x*) (if (is *sort x*) *p*)),

where *sort* is *x*'s sort, or "data type." Sorts will not appear very often, and will be capitalized when they do. They are not very important, and will only save a little typing. We can read (forall (s) ...) as "for all states ...," without having to mention explicitly the condition (is STATE s).

Unbound variables (prefixed with "?") are universally quantified with scope equal to the whole formula (after adding the sort conditions). Anonymous constants of a given sort (used in proofs), so-called "Skolem constants," will be written beginning with the appropriate upper-case letter.

Every state has a time of occurrence, a real number called its *date*. The function d gives the date of a state, as in (= (d S1) D1). Any real number is a valid date: time is infinite and noncircular. Of course, no one in the universe can tell where zero is or what the scale is, so this is harmless. It does mean that two states will have comparable dates, even when they are not related by =<. I will use =< and < for ordinary numerical ordering as well as the partial ordering on states, since the use of sorts will disambiguate. I will not be rigorous about axiomatizing real numbers, but will just assume whatever properties I need as I go. Variables beginning with "r" or "t" are real numbers.

The two orderings are compatible:

```
Axiom 4:  (if (< s1 s2) (< (d s1) (d s2)))
```

States are arranged into chronicles. A *chronicle* is a complete possible history of the universe, a totally ordered set of states extending infinitely in time.

```
Axiom 5:  (Definition of Chronicle)
  (iff (is CHRONICLE ?x)
     (and ;a set of states
          (forall (y) (if (elt y ?x) (is STATE y)))
          ;totally ordered
          (forall (s1 s2)
             (iff (and (elt s1 ?x) (elt s2 ?x))
                  (or (< s1 s2) (> s1 s2) (= s1 s2))))
          ;infinite in time
          (forall (t)
                (exists (s)
                  (and (elt s ?x) (= (d s) t))))))
```

(elt *a x*) means that *a* is an element of set *x*. We won't need any deep set theory, but I will feel free to introduce sets of elements previously introduced, including sets of sets of them. (If variables of some sort begin with a letter "l," then variables bound to sets of objects of that sort begin "ll." So "?ss" is a set of states.)

An immediate consequence of Axiom 5 is that a chronicle is "convex":

```
(if (is CHRONICLE ?x)
   (forall (s1 s2)
      (if (and (elt s1 ?x) (elt s2 ?x))
         (forall (s)
            (if (< s1 s s2)
               (elt s ?x))))))
```

Having defined (is CHRONICLE x), we can conceal most uses of it by declaring variables beginning "ch" to be of sort "CHRONICLE."

A chronicle is a way events might go. There may be more than one of them, according to this logic. (See Figure 1.)

Every state is in a chronicle. In fact,

```
Axiom 6:
  (if (=< ?s1 ?s2)
     (exists (ch) (and (elt ?s1 ch) (elt ?s2 ch))))
```

whence, by convexity, every state between ?s1 and ?s2 is in ch.

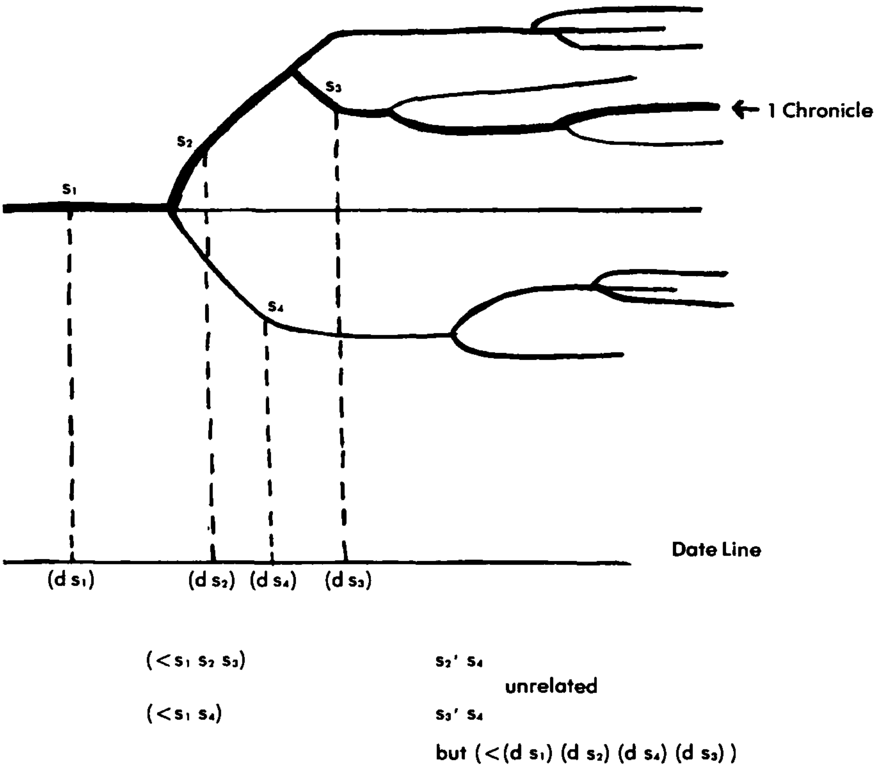


Figure 1. A Tree of Chronicles

Chronicles branch only into the future. (See Figure 1.)

```
Axiom 7:
  (if (and (=< ?s1 ?s) (=< s2 ?s))
     (or (=< ?s1 ?s2) (=< ?s2 ?s1)))
```

The reason why this is so is that the future is really indeterminate. The past may be unknown, but there is only one past. By contrast, there may be more than one future from a given state. The reason for designing the logic this way is to provide for "free will," in the form of reasoning about actions that *select* one future instead of another. If there were only one future, the most we could do is *discover* it. Of course, both alternatives have unpleasant consequences: the one-future account implies that what we are going to do is unknown but fixed, while the many-futures account implies that the alternative futures to yesterday are as real as this one. For this reason, I do not include any reference to "yesterday" or even "now" in the logic, but simply talk about states in the abstract. The application to the state "now," and the fondness we feel for the "real" chronicle, are matters I defer until the section on implementation.[1]

States and chronicles are important only because they are the stage where facts and events are acted out. *Facts* change in truth value over time. By the usual mathematical inversion, we will take a fact to be a set of states, intuitively those in which it is true. For example, (ON A B) denotes the set of states in which A is on B. ON is a function from pairs of objects to sets of states, that is, facts; it is *not* a predicate.[2] This way of looking at facts is analogous to the logicians' trick of letting propositions denote sets of possible worlds (see e.g., Montague, 1974).

I will let variables beginning with "p" and "q" denote facts. The fact "always" is the set of all states. The fact "never" is the empty set.

We indicate that a fact is true in a state by (elt *s p*). As syntactic sugar, we usually write this as (T *s p*). ("T" suggests "true-in.") So, we have

```
Axiom 8:
  (T ?s always)
  (not (T ?s never))
```

We can think of facts as "propositions" in a Mooresque object language. In particular, we can combine them with connectives. For instance,

[1] I should point out that the logic I am developing is *not* intended as an analysis of the truth conditions of English or some other natural language. I doubt that this is at all a good way to think about natural language, and even if it is I see no reason why the internal representation should be constrained by the mere presence of words like "now" in natural language.

[2] It may be considered a predicate in an object language for which this temporal logic is a metalanguage; see below.

we can write (T *s* (& *p q*)), where the "&" is *not* part of our own logical notation; instead, it is simply syntactic sugar for set intersection; (& *p q*) is just the set of states that are elements of both *p* and *q*. Similarly, "V" and " – " in this context denote union and complement (with respect to the set "always"). Then we have things like

```
(iff (T ?s ?p) (not (T ?s (– ?p))))
(iff (T ?s (& ?p ?q))
    (and (T ?s ?p) (T ?s ?q)))
```

as trivial set-theory results, after syntactic desugaring.

Events are more difficult to handle than facts. An event is something happening. In the past, the only kind of events handled by AI researchers and most philosophers is what might be called a *fact change,* such as a block being moved from one place to another (McCarthy, 1968; Rescher, 1971). The defining feature of an event on this theory are the changes in facts that the event brings about. This approach suppresses some important features of events. For instance, they take time. A fact change is just a list of two facts; how long it took is not describable. Further, it is meaningless in fact-change formalisms to ask what happens in the middle of a fact change.

Consider the usual emphasis in studies based on McCarthy's situation calculus (McCarthy, 1968; Moore, 1980; Fikes, 1971). In this system, an action like "moving *x* to *y*" is reasoned about in terms of a function MOVE that maps a block, a place, and an old situation into a new situation; (MOVE *x y s*) is the situation resulting from moving *x* to *y* in *s*. The axioms of the calculus talk entirely about the different facts true in *s* and (MOVE *x y s*). There is no mention of the infinite number of states occurring during the move.

Some of these problems can be eliminated by simply shifting emphasis, as I will show shortly. But a deeper problem is that many events are simply not fact changes. An example due to Davidson (1967) is "John ran around the track 3 times." The only fact change that occurs is that John is more tired. The amount of fatigue is not terribly different from the amount ensuing on running around 4 times. Besides, surely no one would argue that the *definition* of "run around the track 3 times" is "be here tired." Of course, John might have a memory of having done it, but even "be here tired with a memory of having run around 3 times" is still not a plausible definition, if for no other reason than that John might have lost count. Also, this definition is circular, since John's memory must make reference to the concept to be analyzed, and hence can only mean "I remember [bringing it about that I am tired and have a memory of [bringing it about that I am tired and have a memory of [. . . .]]]"

If you still need to be convinced, consider the (large) class of actions that are done for their own sake, such as visiting Greece, eating a gourmet meal, or having sex. In all these cases, the fact changes are trivial, unappetizing, or only tangentially relevant. One could argue, I suppose, that these things are done only for the memory of having done them. It is true that doing them without remembering them would be a little pointless, but memory fades. Knowing you won't remember much of this trip, meal, or sexual activity 20 years from now is not much of a barrier to doing it now, and does not entail that doing it is logically impossible.

We need a fresh approach. One idea is that events be identified with a certain kind of fact, namely the fact that the event is taking place. Facts occupy time intervals, so we get the ability to talk about what happens during an event. This seems to be adequate for events that consist of some aimless thing happening for a while, such as a rooster crowing in the morning. The rooster-crowing event could just be defined to be the time during which the rooster is crowing. This event happens in a chronicle if any of its states are in that chronicle.

But most events do not fit this mold. Running around a track three times takes time, but cannot be identified with the states during which you are running on the track. The problem is that a given state may be part of a "3 times around" event in one chronicle, and a "2 times around" event in another. But the criterion would have the event happening in both.

We avoid this problem by identifying an event as a set of intervals, intuitively those intervals over which the event happens once, with no time "left over" on either side. An interval is a totally ordered, convex set of states. We can think of each interval as an event token, and the whole set as an event type. So "Fred running around a track 3 times" is the set of all intervals in which exactly that[3] happens.

Now we can indicate that an event happens between states *s1* and *s2* by writing (elt [*s1, s2*] *e*). As syntactic sugar for this, I will write (Occ *s1 s2 e*). Notice that I let variables beginning with "e" stand for events.

Can we always assume that an event occurs over a closed interval? Let us leave this question unanswered for the time being. In this paper, I will always used the Occ notation, and hence assume that they are closed, but it doesn't seem very important for most events whether they include two extra instants or not. Since we will want to allow for instantaneous events, at least some of them must be closed.[4] The notion of a fact being true over a period

[3] The phrase "exactly that" is intended to rule out "last Tuesday" as a token of this event if Fred ran around the track once on Tuesday (unless it took him 24 hr). But I do not mean to insist that an event happen over an interval only if it happens over no subinterval. When the event "Fred whistles" happens over an interval, it happens over an infinite number of subintervals. Incidentally, the idea of letting events be sets of intervals was stated by Montague (somewhat differently) in Montague (1960).

[4] Notice, by the way, that if we interpret event intervals consistently (as always closed, always half-open on the right, or whatever), then using them is equivalent to modifying McCarthy's situation calculus by letting actions be *relations* on situations (states) instead of functions.

of time is still valuable, even though it wouldn't carry the full load. This is written (subset [*s1, s2*] *p*), or, syntactically sugared, as (TT *s1 s2 p*).

Certain events and facts are closely related. For example, (sunion *S*), for any set *S* of sets, is the union of all its elements. The (sunion *e*) is a fact, true whenever *e* is "in progress," in the sense that in some possible chronicle *e* is in the process of occurring. I will use the syntactic sugaring (in-progress *e*) to mean the same thing as (sunion *e*), and (Tocc *s e*) to mean (T *s* (in-progress *e*)).

Given a fact, we can work our way back to events in more than one way. For instance, we can take the set of maximal intervals during which the fact is true, or the set of point intervals for all points where the fact is true.

Events can be related to each other in ways similar to those for facts. For instance, if *p* is a subset of *q*, then it is as if *p* implied *q*: at every state where *p* is true, so is *q*. For events, we write (subset *e1 e2*) as (one-way *e1 e2*): *e1* is one way *e2* can happen; every occurrence of *e1* is an occurrence of *e2*. For example, being squashed by a meteor is one way of being squashed.

We used boolean connectives like "&" to combine facts. These are not so useful with events. Instead, we need things like

```
(seq e1 e2 ... eN)
which stands for
{[s0, sN]: (exists (s1 ... sN-1)
           (and (Occ s0 s1 e1)
                (Occ s1 s2 e2)
                ...
                (Occ sN-1 sN eN)) )}
```

Corresponding to "never," the fact that is never true, there is an event that never happens. This will also be the empty set, so we can call it "never," too, making this the only thing that is both an event and a proposition. There does not seem to be any useful notion of the event that always happens.

More such constructs will be introduced as we go.

Remember that this logic takes an Olympian view of states of the universe. "Now" is not distinguished, so there is no question about representing what has already happened versus what may happen. I will talk about this more in Section 6, below. I should point out, though, that representing tokens of past or expected events as ordered pairs of states, like (s34, s107), is not adequate. A given interval is a token of many different events, which happened to occur at that point. So event tokens must be represented as ordered pairs of events and intervals, or something equivalent.

I want to stress at this point that devising ontologies like this is not an empty philosophical enterprise. On the contrary, I am interested in purely utilitarian ends; I want a way of thinking about time that is useful to a robot. I am not interested in expressing all possible ways of thinking about time, nor am I interested in calculating the truth values of English statements involving time. It may seem that logic and practicality have little to do with each other, that the problem for cognitive science is to build a computational model that reasons about time, and be done with it. Unfortunately, it is not so straightforward. Any program will be based on *some* ontology and assumptions about time. The wrong assumptions will mire us in a swamp of logical conundrums, which much be explicitly faced and conquered. The best way to do this is to make the logical machinery explicit (cf. McDermott, 1978a).

This is what I will be doing in the rest of this paper, examining three major problems that temporal reasoners will face: reasoning about causality and mechanism, reasoning about continuous change, and planning actions. There may be others, but these should suffice. They have been difficult in the past precisely because dangerous assumptions have been made about time, such as that there is a next moment, or that there is only one future. I will try to show that a program based on the logic I propose will have a better chance of avoiding these difficulties.

To illustrate how logical assumptions influence thought, I will try to prove a theorem about a mechanism, and show the power and weakness of what we have assumed so far. The theorem goes like this: Let DEV be a device with two states, DAY and NIGHT. DAY is always followed by NIGHT and NIGHT by DAY. DAY and NIGHT never overlap. Prove that if it is ever DAY or NIGHT, it will always be either DAY or NIGHT.

This may seem simple, but it is just the sort of inference that is beyond the capability of existing reasoning systems. Expressed in our notation, it is

```
DAY and NIGHT are mutually exclusive (except at boundaries):
  (if (and (Occ ?s1 ?s2 DAY) (Occ ?s3 ?s4 NIGHT))
     (forall (s)
       (if (and (= < ?s1 s ?s2) (= < ?s3 s ?s4))
          (or (= s ?s2 ?s3)
              (= s ?s1 ?s4))) ))

Each takes a nonzero amount of time:
  (if (or (Occ ?s1 ?s2 DAY) (Occ ?s1 ?s2 DAY))
     (< ?s1 ?s2))

and each follows the other
  (follows DAY NIGHT)
  (follows NIGHT DAY)
where
  (iff (follows ?e1 ?e2)
      (if (Occ ?s1 ?s2 ?e1)
         (forall (ch) (if (elt ?s2 ch)
                         (exists (s3)
                            (and (elt s3 ch)
                                 (Occ ?s2 s3 ?e2)))))))
```

That is, *e2* follows *el* if every occurrence of *el* is followed immediately by an occurrence of *e2* in every chronicle containing the occurrence of *el*, i.e., in every way events might proceed.

Now to prove

```
  (if (Occ S1 S2 DAY)
     (forall (s) (if (> s S2) (or (Tocc s DAY)
                                   (Tocc s NIGHT)))))
```

This theorem may seem trivial, but in fact it does not follow from what we have assumed so far. If each succeeding DAY or NIGHT interval is half as long as the previous one, then an infinite number of them could go by in a finite amount of time, after which the state of DEV could be something else. However, this is something we wish to rule out.

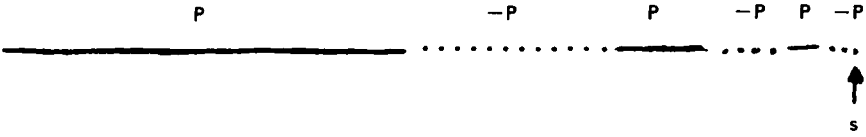
We do so with the axiom

```
Axiom 9:
  (forall (s p)
   (and (exists (s0)
              (or (TTopen s0 s p))
                  (TTopen s0 s (- p)) )
        (forall (ch)
              (if (elt s ch)
                 (exists (s1)
                  (and (elt s1 ch)
                       (or (TTopen s s1 p)
                           (TTopen s s1 (- p)))) )) )))
where
  (iff (TTopen s1 s2 p)
     (and (< s1 s2)
          (forall (s)
             (if (< s1 s s2) (T s p)) )))
```

This axiom, due to Ernie Davis, assures us that, for every fact and an arbitrary state, there is an interval preceding the state during which the fact is always true or always false; and another one following the state, in every chronicle containing it. (See Figure 2.)

The presence of this axiom rules out any super powerful axiom of "comprehension,"[5] which would allow us to infer that any set of states was a fact, such as the set of states during which the temperature in Cleveland is a rational number. This is *not* a fact because, assuming the temperature is smoothly changing, it will change truth value infinitely often in any finite interval.

[5] An axiom or axiom schema of comprehension states that for every property, there is a set of objects satisfying it. Stating this formally in a way that avoids paradoxes is a major preoccupation of set theorists (Mendelson, 1964).



P violates Axiom 9 at s

Figure 2. How a Well-Behaved Fact Does Not Behave

So we will need special-purpose comprehension axioms for well-behaved facts. I will just assume these along the way as obvious. For example, if *p* and *q* are facts, (& *p q*) is also.[6] When I introduce a function like "in-progress," and announce that its values are from a given domain, like facts, I am implicitly declaring an axiom like

```
Axiom 10:  (In-Progress Comprehension)
  (is FACT (in-progress ?e))
```

So you can take for granted that (in-progress *e*) satisfies Axiom 9. This axiom does away with any super powerful comprehension axiom for events, in case you were wondering.

You may now take it on faith that no further assumptions are required to prove that it will always be DAY or NIGHT, or you can bear with me through the following proof. (It is not as arbitrary as it seems; if anyone can find a simpler or clearer proof, I would like to hear about it.)

First, we need a few definitions. Letting sets of events be denoted by variables beginning with "ee," and integers be denoted by variables beginning with "n," we define

```
Axiom 11:
  (iff (chain ?ee 0 ?s1 ?s2) (= ?s1 ?s2))

  (iff (chain ?ee (+ ?n 1) ?s1 ?s3)
       (exists (e s2)
         (and (elt e ?ee)
              (Occ s2 ?s3 e)
              (chain ?ee ?n ?s1 ?s2)) )))
```

That is, there is an ee chain of length n from s1 to s2, if there is a sequence of abutting events from the set ee that reaches from s1 to s2.

[6] We could probably recast Axiom 9 as a biconditional and prove these axioms, but set-theoretic parsimony is not really important.

Now reachability is defined thus

```
Axiom 12:
  (iff (reachable ?ee ?ch ?s1 ?s)
      (exists (n s2)
         (and (> = n 0)
              (elt s2 ?ch)
              (chain ?ee n ?s1 s2)
              (= < ?s1 ?s ?s2)) ))
```

We read this "?s is ?ee-reachable in ?ch from ?s1."

Some corollaries of these definitions (using Peano arithmetic) are:

```
(if (reachable ?ee ?ch ?s1 ?s)
   (forall (s')
     (if (< ?s1 s' ?s)
        (reachable ?ee ?ch ?s1 s')) ))

(if (reachable ?ee ?ch ?s1 ?s)
   (or (= ?s1 ?s)
      (exists (e s2 s3 n)
        (and (elt e ?ee)
             (elt s3 ?ch)
             (Occ s2 s3 e)
             (= < s2 ?s s3)) )))
```

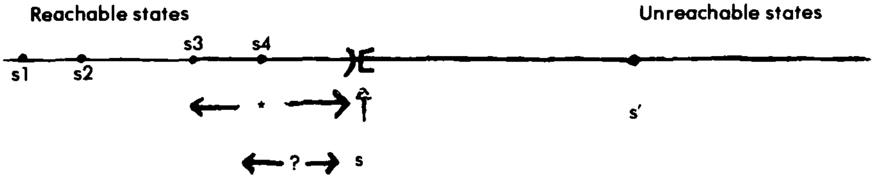
These state that if ?s is ?ee-reachable from ?s1, then every state between ?s1 and ?s is reachable, and ?s occurs in the middle of some event in ?ee.

Now the proof goes as follows: Assume that S' is a state such that (> S' S2) and (not (Tocc S' DAY)) and (not (Tocc S' NIGHT)). Then by Axiom 6, there is a chronicle CH1 containing S' and S2. Clearly, (not (reachable {DAY, NIGHT} CH1 S2 S')). So, by the properties of real numbers and the first corollary above, there must be a state S, (< S2 S) and (= < S S'), such that every state between S2 and S is {DAY, NIGHT}—reachable in CH1 from S2, and every state from S on is not {DAY, NIGHT}—reachable in CH1 from S2. But, by Axiom 9, there must be an SD3 before S such that either DAY is in progress for all states between SD3 and S, or it is not in progress for all those states. Similarly, there must be an SN3 before S such that it's NIGHT or it isn't from SN3 to S. Since it can't be neither or both, let S3 be the one for which either it is DAY from S3 to S or NIGHT from S3 to S. Clearly, (< S2 S3) because both DAY and NIGHT occur at least once after S2. Every state from S3 to S is {DAY, NIGHT}—reachable, so, by the second corollary, one of DAY or NIGHT is occurring from S3 to S, and this occurrence ends in some state S4 in CH1. (See Figure 3.)

```
        REACHABLE                      )[          NOT REACHABLE
      S1 S2                    S3   S4   S        S'
                              <-   *   ->
                                   <-  ?  ->
```

*either DAY or NIGHT in progress throughout this interval

S4 must come before S, or else S would be reachable, according to the definition, because S4 would end a chain from S2. But then starting at S4 NIGHT or DAY must occur, so DAY and NIGHT must coexist for more than an instant, which is impossible. So there is no such S, and all instants are reachable—QED.



*Either DAY or NIGHT is in progress throughout this interval

Figure 3. Proof of Eternal DAY or NIGHT

This may seem quite complicated. But it depended on only one new axiom, Axiom 9. Everything else came from definitions and arithmetic. Of course, this proof is much too complicated to expect a theorem prover to come up with it, but this was never my goal. My intent is similar to Hayes's: to express concepts in a form in which the intuitively plausible inferences are valid. If this is achieved, then we can start worrying about a practical inference program. In fact, I start worrying in Section 6, below. The only thing to point out here is that such a program has no hope of being complete.

I should also assure you that this paper is not crammed full of such long proofs of obvious results. The main purpose of showing you this was to let you get a feel for the generality of the ontological assumptions. They are so general that we have to tame them with Axioms like Axiom 9. But this is all the taming we will want to do.

Also, this result is not entirely academic. It is easily generalizable to a system with a finite number of mutually exclusive states which succeed each other the way DAY and NIGHT do. It gives us the ability to infer infinite loops in simple machines.

Now, as promised, I will examine three major problem areas from the point of view of this logic, before turning to implementation questions.

3. CAUSALITY

Causality is fundamental to a lot of problem solving. A problem solver brings things about by causing other things. What I mean by causality here is that one event (type) always follows another event (type). For example, if x is a loaded gun, pulling its trigger is followed by its firing.

Unfortunately, there must be more to it than that. For example, an exactly analogous case is, If a is approaching from the direction of the sun, the arrival of a's shadow is followed by the arrival of a. But we would not want to say that the arrival of a's shadow causes the arrival of a.

I assume that there is no way to get around this problem, and that there is no way to infer causality merely from correlation. So we will not try to define causality in terms of something more basic. Instead, we will assume whatever causal assertions we need, and infer events from them.

Events can cause two kinds of things: other events, and facts. The two cases are quite different, and the first is simpler.

When an event causes another, there is usually a delay. The scale of the date line attached to the chronicle tree is unknown in the logic, so we cannot use absolute time intervals. Instead, we assume that there are objects called *scales* which occupy some constant amount of time. If "hour" is such a scale, (* 5 hour) is a length of time equal to 5 times the size of hour (see McDermott 1980b for a fuller explanation). We will never be able to evaluate this, but we don't need to; we just need to be able to compare it to other things measured in hours or seconds. We can do the latter because we have as an axiom (= (3,600 second) hour). Note the elision of the * when it is clearly unnecessary.

With this out of the way, we introduce our basic predicate (ecause *p e1 e2 rf i*), which means that *e1* is always followed by *e2*, after a delay in the interval *i*, unless *p* becomes false before the delay is up. The delay is measured from a point *rf* through *e1*; if *rf*=0, this means from the start of *e1*; if *rf*=1, from the end.

```
Axiom 13:
  (if (ecause ?p ?e1 ?e2 ?rf ?i)
     (if (Occ ?s1 ?s2 ?e1)
        (forall (ch)
           (if (elt ?s2 ch)
              (exists (s3)
                (and (elt s3 ch)
                     (within-delay s3 ?rf ?i ?s1 ?s2)
                     (or (not (TT ?s2 s3 p))
                        (exists (s4)
                          (and (elt s4 ch)
                               (Occ s3 s4 ?e2)) ))) )) )))
```

```
where
  (iff (within-delay ?s ?rf ?i ?s1 ?s2)

     (elt { (d ?s) }
          { – (1 – ?rf)*(d ?s1) }
          { + ?rf*(d ?s2) } }

          ?i))
```

The (within-delay *s rf i s1 s2*) means that state *s* occurs after *s1* and *s2*, with delay *i*. An *rf* is a real number that says what point the delay is to be measured from. If it is 0, the delay is to be measured starting at *s1*; if 1, from *s2*; and so on for any number between 0 and 1. The *i* is a real interval, like <(3 min), (5 min)>, or [0, (5 hour)]. (An open side of an interval I denote by the usual angle bracket, as in <1, 3] or <1, 3>. A closed interval on the reals, while denoted with square brackets [...], is a completely different sort of thing from a state interval.)

As an example of ecause, we can express the idea that if a Republican is elected President, science will progress:

```
(ecause (POLPARTY ?x REPUBLICAN)
        (elected ?x)
        (INFLUX-MONEY-FOR-DESERVING-RESEARCHERS)
        1 [(1 year), (2 year)]))
```

In these examples, only the parts being illustrated are formalized in a reasonable way.

If the fuse on a powderkeg is lit, the keg will explode if the powder stays dry:

```
(ecause (& (DRY ?keg) (FUSE-OF ?fuse ?keg))
        (LIT ?fuse)
        (EXPLODE ?keg)
        1 [(30 sec), (2 min)]))
```

If a winch is rotated, an object gets hauled up:

```
(if (is WINCH ?x)
   (ecause (LOAD-OF ?x ?y)
           (ROTATE ?x)
           (RISE ?y)
           0
           [0, (2 sec)]))
```

Note that the object might not start rising for a second or two.

We also have the axiom:

```
Axiom 14:
  (if (ecause ?p ?el ?e2 ?rf ?i)
     (forall (s3 s4)
        (if (Occ s3 s4 ?e2)
           (exists (pc ec sl s2 rfc ic)
             (and (ecause pc ec ?e2 rfc ic)
                  (Occ sl s2 ec)
                  (within-delay s3 rfc ic sl s2)) )) ))
```

That is, if an event is ever caused, then each of its occurrences is preceded by one of its causes (with the appropriate delay). This might be called the Principle of Paranoia. Its chief virtue is in enabling us to infer that an event must have occurred when it is known to be the only cause of another event that occurred.

The second kind of causality is the causation of a fact by an event. For example, if a boulder falls to the bottom of a mountain, it will *be* at the bottom of the mountain. This is important in problem solving, where the goal is often to bring about some fact by causing one or more events.

One approach might be to say that e causes p if, in all chronicles, p is true for some period of time after e. We could do this, but it would be useless. In this sense, shooting a bullet past someone would be a way of achieving that it was near him.

I must digress here to talk about the speed at which facts change. The real world doesn't change fast most of the time. Many facts remain true for long enough to be depended on. For example, that boulder will probably stay at the bottom of the mountain for years (or centuries). We normally use such facts with confidence, for example, when planning to build a house on the boulder.

On the other hand, we cannot infer with certainty that the boulder will be there. If we could, then there would be no way to plan confidently to remove it. Confidence in the plan would just land us in a contradiction between our belief that the boulder will be gone by next year, and our certainty that it will be there for many years.

This is a classic example of a *non-monotonic* reasoning pattern (McDermott, 1980a; McDermott, 1981a; and Reiter, 1980). The inference that the boulder will be there is good until you find out that someone is planning to move it. I have resisted introducing non-monotonicity into the logic so far, because it is not that well understood, and what is well understood about it is not all that encouraging. But we are going to need it here.

The problem here is closely related to the *frame problem*. That was the problem that arose in McCarthy's situation calculus (McCarthy, 1968) of not being able to infer anything about a situation resulting from an action in a previous situation, without a large number of axioms of the form "p doesn't change in this transition." A typical axiom would say, "No block's color changes in the transition from s to (MOVE A B s)." The problem is even more acute for us, because almost anything could be happening in an interval. In McCarthy's calculus it was possible to pretend that a situation (MOVE A B S0) would persist until the next action, so that the situation after two actions could be denoted by something like (MOVE C A (MOVE A B S0)). Now the state of the world changes as the problem solver plans, so there is no term denoting the state of the world when the second action occurs. The frame problem becomes the problem of inferring what's true at the end of an arbitrary interval, given incomplete information about what happened during it.

Part of my expectation in developing a robust logic of time was that we could reason about facts "from the side," inferring that they were true for whole stretches of time. It's no loss that we can't work our way from one state to the "next" any more; that was always a bad idea. But now we find that in general you cannot infer that a fact is true for a period of time.

Let me distinguish this problem from another one that is often held to be solvable with non-monotonic notations. Every AI hacker knows that the example causality axioms I gave earlier are incomplete, and that there is no way to make them complete. For instance, the keg will not explode if the fuse is cut, or if all the oxygen is removed from the keg before the spark reaches it, or the keg is placed in an extremely strong box that can withstand the explosion, or .... But you see the point. It seems pointless to try to list all the ways the rule could fail.

This problem can be solved simply by letting our rules fail now and then. We can't hope to avoid errors, and it normally doesn't matter if a data base is "slightly" inconsistent. When it does matter, we can edit the rules to maintain consistency. So in a sense the theory is "approximately" true, and gets closer to the truth with every edit. Non-monotonic logic could play a role by letting rules "edit themselves" (McDermott, 1980a), but this hardly seems necessary.

The rule that a boulder stays put for years is not even approximately true in this sense. It would be approximately true only if it were used in a purely passive system. An astronomer observing an uninhabited planet might use the rule this way. He would simply live with errors caused by improbable occurrences like volcanic eruptions that moved boulders. But a problem solver knows full well *both* that it is counting on certain things to be true for a while, *and* that it could make them false any time it wanted to. (Other agents could also make them false, but we neglect this possibility.)

To capture these ideas in the logic, I introduce the notion of *persistence.* A fact p persists from s with a lifetime r, if in all chronicles it remains true until r has gone by or until it *ceases* to be true.

```
Axiom 15: (Definition of Persist)
  (iff (persist ?s ?p ?r)
       (and (T ?s ?p)
            (forall (s')
              (if (and (within-lifetime s' ?r ?s)
                       (not (T s'  ?p)))
                  (Occbetween ?s s' (cease ?p))) )))
```

where

```
  (iff (within-lifetime ?s2 ?r ?s1)
     (and (=< ?s1 ?s2)
          (< (- (d ?s2) (d ?s1)) ?r)))

  (iff (Occbetween ?s0 ?s3 ?e)
     (exists (s1 s2)
       (and (=< ?s0 s1 s2 ?s3)
            (Occ s1 s2 ?e)) ))
```

Ceasing does not mean merely that the fact goes from true to false. In fact, ceasing is so rare that it never happens unless we hear about it:

```
Axiom 16:  (Fundamental Property of Ceasing)
  (if (and (persist ?s ?p ?r)
           (within-lifetime ?s' ?r ?s)
           (M (nocease ?s ?p ?s')))
      (not (Occbetween ?s ?s' (cease ?p))))
```

M is a primitive sentence-forming operator, read "Consistent." Intuitively, if *Q* cannot be proven false, then (M *Q*) is true. The (nocease *s p s* ') means that no occurrence of (cease *p*) occurs between *s* and *s* '. To conclude that *p* actually does not cease, we require only that it be consistent that it not cease; positive information is necessary to override this. The overriding occurs when other rules allow us to infer (not (nocease *s p* S1)) for some S1 within the lifetime of the persistence; then the M fails, we cannot infer the *p* does not cease. But if no such rule applies, then we can make the inference.

I hope that this application of non-monotonic logic will not mess everything up. I am depending on a property of the logic of (McDermott, 1981a), namely, that from Axiom 16 and an occurrence of a ceasing within the lifetime of a persistence, we can deduce (not (nocease *s p s* ')). If a weaker logic is used, this should be made explicit in an axiom. However it is done, it is essential that (Occbetween *s s* ' (cease *p*)) kill off a persistence after *s* ', but leave the persistence "in force" for states between *s* and *s* '. Clearly, if (Occbetween *s* ?*s* ' (cease *p*)), then (Occbetween *s s* " (cease *p*)) for all states *s* " between *s* ' and the end of the persistence. Then we can infer (not (nocease *s p s* "))for all those states. We *cannot* infer such a thing for states *before s* '.

What this means is that a plan to remove the boulder five years from now cancels a persistence after that time, but leaves intact the inference that it will be there until then.

By the way, let me make a disclaimer. I try to appeal to non-monotonic deductions as seldom as possible. This is because the logics they are based on (McDermott, 1980a; McDermott, 1981a; Reiter, 1980; and McCarthy, 1980) are still rather unsatisfactory. For one thing, even some of the simple deductions in this paper may not be valid in any existing non-monotonic system. For example, the problem with existential quantifiers cited in (McDermott, 1981a) would probably block some of my proofs. (The system of (Reiter, 1980) avoids this problem, but has others.) For another thing, such logics do not distinguish between severities of contradictions; they use the same machinery for "rule edits" of the kind I described and for clipping off a persistent fact. In the usual terminology of such systems, this leads to unexpected "fixed points," or models, in which the wrong assumptions are retracted.

For the time being, we can view these not a problems with this paper, but as problems with non-monotonic logic. In attempting to represent things, it is helpful to be as formal as we can, but if the formal systems cannot keep up with the inferences we want to make, so much the worse for the formal systems. In the long run, I am confident that non-monotonic logics will be developed that capture the inferences we need. Probably the best way to see what inferences those are is to try to get along with the fewest possible non-monotonic inferences, but to feel free to use them when all else fails. If representation designers make it clear what they need, logicians will make it work.

Armed with the idea of persistence, we can make some progress on our original problem. First of all, it seems reasonable that most inferences of facts are actually about persistence of facts. For one thing, many facts have characteristic lifetimes. If x is a boulder, then (AT x *location*) has a lifetime measured in scores of years. If x is a cat, then (AT x *location*) has a lifetime measured in minutes (if the cat is sleeping) or seconds (if the cat is awake).

The senses actually tell you about persistences. I was driven to this by the following problem our logic appears to involve us in. At first blush, we might want an axiom to the effect that if a boulder is at a location loc in state S0, then (persist S0 (AT Boulder Loc) (50 year)). But then we can infer that the boulder will be there in 50 years, when another persistence will start, and so on. We can infer that the boulder will be there for any given time in the future. If this seems harmless, think about the cat instead.

The solution is to scrap such axioms. Instead, we normally start with a persistence and work our way to particular states, not vice versa. This requires that when we see a boulder, our eyes are telling our data base about a persistence, not about an instantaneous fact. Otherwise, as soon as we turned away, we would know nothing about the scene. Once you get used to this idea, it seems perfectly natural.

This brings us back to causation. Clearly, what events must cause directly is persistences, not the truth of facts. So our primitive predicate is (pcause *p e q rf i rl*), which means that event *e* is always followed by fact *q*, after a delay in the interval *i*, unless *p* becomes false before the delay is up. The delay is measured from a point *rf* through *e;* if *rf* = 0, this means from the start of *e;* if *rf* = 1, from the end. When *q* becomes true, it persists for lifetime *rl*. Formally,

```
Axiom 17:
  (if (pcause ?p ?e ?q ?rf ?i ?rl)
    (if (Occ ?s1 ?s2 ?e)
      (forall (ch)
        (if (elt ?s2 ch)
          (exists (s3)
            (and (elt s3 ch)
                 (within-delay s3 ?rf ?i ?s1 ?s2)
                 (or (not T s3 p))
                    (persist s3 ?q ?rl))) )) )))
```

And we have examples like:

```
(pcause always)
        (KILL ?x)
        (DEAD ?x)
        1 [0, 0]
        FOREVER)
```

We pick "FOREVER" to be a very long time, equal to the largest number that can be stored on the machine the universe is being simulated on, or the length of time until the Last Judgement, depending on your religion.

Another example is:

```
(if (is STOVE ?x)
   (pcause (- (BLACKOUT))
           (TURN-ON ?x)
           (HOT ?x)
           1 [(1 min), (2 min)]
           (24 hour)))
```

Notice that the persistence time is picked as the time interval over which it is reasonable to infer that the state will remain in existence, assuming you have no intention of changing it. I pick 24 hours here because within that time, either one's spouse will find the burner and turn it off, or the house will burn down. Of course, normally you plan to turn it off sooner. If for some reason you wanted the burner to stay on longer (say you were cooking something that took a really long time), you would just need axioms about putting signs up, or other special tactics. These would say, "If you put a sign up telling someone not to alter a state they won't worry about if they see a sign, then the state will remain as long as the sign is up."[7]

Notice that not all instances of inferring facts "from the side" are direct instances of persistence. Part of the power of the notion comes from the fact that persistent facts have consequences. For example, if everyone in the American embassy is audible (while the embassy is bugged), and the Henry is in the embassy for 15 minutes, then he is audible during that period. We don't have to come up with a general lifetime for audibility.

There is no Principle of Paranoia (Axiom 14) for pcause. This is because there are so many ways a fact can come about, including logical consequence, that it does not seem reasonable to look for a cause every time. Also, most true facts are "left-over" persistences. Most boulders in the world have been there longer than any lifetime you would use; the lifetime you can count on is much shorter than the times you observe. By the way, this should make it obvious that the logic does not imply that a persisting fact *stops* being true after its lifetime; we simply lose information after that point.

Since persistences, and not facts, are caused, and since there is usually no persistence that extends back to when a fact became true, there is really no cause for most facts, at least not in the technical senses I have been developing. Of course, many facts of interest are the result of observed or inferred events, and *these* will be caused. One interesting case is when an occurrence of (cease p) is inferred using Axiom 15. We can then infer that this ceasing was caused. In fact, we can call this the Special Principle of Paranoia:

```
Axiom 18:
  (if (Occ ?s3 ?s4 (cease ?p))
      (exists (pc ec s1 s2 rfc ic)
        (and (ecause pc ec (cease ?p) rfc ic)
             (Occ s1 s2 ec)
             (within-delay ?s3 rfc ic s1 s2)) )) ))
```

[7] Several people (notably Ernie Davis, James Allen, and Ken Forbus) have suggested that the idea of lifetime should be dropped from persistences. Even though a burner rarely stays on for more than 24 hr, it *would* if left unattended, and my notation obscures this fact. My main reason for sticking with limited persistences is to take into account the fact that in many cases, we simply lose information about a system for moments too far from our last observation.

pcause and ecause work hand in hand. Consider the event (PUTON A B) that occurs within minutes after the beginnings of persistences of (CLEARTOP A) and (CLEARTOP B), with lifetimes of several hours. Suppose we have axioms like:

```
(pcause (& (CLEARTOP ?x) (CLEARTOP ?y))
        (PUTON ?x ?y)
        (ON ?x ?y)
        1 [0, 0]
        (10 hour))
```

and, of course,

```
(iff (T ?s (CLEARTOP ?X))
   (not (exists (y) (T ?s (ON y ?x)))))
```

We can deduce from the persistence of (CLEARTOP A) and of (CLEARTOP B) that (& (CLEARTOP A) (CLEARTOP B)) will be true for several hours. Hence it will be true during the PUTON. (We will need an axiom about how long PUTONs take.) Hence (ON A B) will persist from the end of the PUTON. Hence (CLEARTOP B), no longer true, must have ceased, and the rule that it doesn't cease is inapplicable. However, it still can be inferred not to have ceased up to the end of the PUTON, so there is no contradiction. I will say more on the subject of reasoning about plans later.

Before going on to other topics, I should pause to review previous work on representing causality. Curiously, Hayes (1979a) argues that there is no isolated body of knowledge about causality. Every branch of "naive physics" has its own way of accounting for things happening. He also says he has found no need for non-monotonicity. I envy him. I think the reason for his good fortune is the "passive" character of his theory. It says how to reason about physical systems; it takes a Buddhist attitude of resignation toward bad things. For example, Hayes's Theory of Liquids (Hayes, 1979b) can be used to infer a flood, in such a way that it is plain contradictory to suppose the flood can be prevented. (This is a bit unfair, since he would presumably make the move of changing the axioms, i.e., the physical setup, as a reflection of the action of the planner. I don't know if this would amount to letting non-monotonicity in by the back door or not.)

The most obvious competitor to the theory I have presented is that of Rieger, who developed a graphical notation for what he calls "Common-Sense Algorithms" (CSA) (Rieger, 1976; Rieger, 1975). This notation included devices for representing concepts like continuous causality, "gated one-shots," thresholds, and much more. There are several problems with this notation, all stemming from Rieger's refusal to state precisely what the links and nodes of his networks *mean*. Apparently, networks representing physical devices and plans are written exactly the same, or are freely mixed. There is a systematic ambiguity about whether a link drawn on a page indicates that *if* something is done, something else will follow; or *that* the thing is actually done and the consequence actually occurs. For example, does the "threshold" link indicate that if the threshold is passed something will happen, or that the threshold is *supposed* to be passed eventually? It seems as if you need to be able to say both. Apparently in the CSA notation you can only say the latter. It seems somehow perverse to make algorithms more basic than physics. In my system, algorithms come in later, in a different form (see Section 5).

Besides this major flaw, there are lots of little places where the CSA notation fails to be precise. For instance, time delays and lifetimes are not mentioned. How is it possible to reason about a plan involving several parallel actions if they are completely unsynchronized?

On the other hand, there is substantial overlap in what he and I have done. His gated causality, and my provision of a gating fact as the first argument to ecause and pcause, are both due to realization of a key fact about causality, that events' behaviors are modified greatly by background facts.

4. FLOW

A system cannot reason about time realistically unless it can reason about continuous change. This has been neglected by all but a handful of people (Hendrix, 1973; Rieger, 1975). The assumption that actions are instantaneous state changes has made it hard to reason about any other kind. If I am filling a bathtub, how do I describe what happens to the water level during (MOVE A B S0)?

I will use the term *fluent* for things that change continuously over time. (The term is due to McCarthy (1968), who used it in almost exactly the same sense.) Actually, the notion of fluent is more general than that. It is intended to do the work that is done by "intensional objects" in other systems. The President of the United States is a typical intensional object. Unlike most people, he has lived in the same house for over 150 years. His age sometimes decreases suddenly. These may seem like strange properties, but they are necessary (on some theories) to provide the correct truth value for sentences like "The President lives in the White House" (true), or "In 1955, the President was a movie actor" (false).

In my logic, such objects correspond to fluents. A fluent is a thing whose *value* changes with time. The value of a fluent in a given state s is written (V *s v*). I will use "v" for variables ranging over fluents. So, we can express two different readings for the last example sentence above:

(T 1955 (ACTOR (V 1955 President)))
(T 1955 (ACTOR (V 1981 President)))

In the first, "President" is taken to mean "President at the time of the fact." In the second, it is taken to mean "President at the time of the utterance (1981). The rules of English make the first reading more likely, under which the sentence comes out false.

Fluents are valuable, not for this sort of playing around, but because physical quantities may be thought of as fluents. For example, "the temperature in Cleveland" is a fluent, which takes on values in temperature space. The changes of the fluent can then be reasoned about. In particular, the fluent's being in a certain region is a fact which might be helpful in causal reasoning. All of the fluents I will look at from now on will have numbers as values. Such fluents I will loosely call "quantities." Most quantities are real-valued and vary continuously as well.

At this point a certain abuse of notation will make life simpler. Strictly speaking, (> *v1 v2*) is meaningless, because > relates numbers or states, not fluents. What we really need is a function (>* *v1 v2*), which takes two fluents and returns the fact which is true in all states *s* for which (V *s v1*) is greater than (V *s v2*). Similarly, (>! *v1 r*) might take a fluent and a real number, and return the fact which is true just when the quantity's value is greater than the number. Clearly, to do this rigorously would be tedious. Instead, I will just assume that all of the red tape can be cleared away, and use (> *alpha beta*) freely, where *alpha* and/or *beta* is a fluent, integer, real number, etc. If either *alpha* or *beta* is "polluted" by being a fluent, the result is a fact; if both are numbers, the result is either true or false. Similarly, (+ *alpha beta*) will produce a new fluent, unless both alpha and beta are numbers, when the result is a number. For safety's sake, I will not do this for anything but simple arithmetic predicates and functions.

By the way, notice that since things like (> (− V1 V2) (* 5 V3)) are facts, they must obey Axiom 9. Ernie Davis has shown that this puts some fairly strong contraints on quantities. A quantity gives rise to a time function in every chronicle; given the time, the fluent delivers a unique value. Axiom 9 constrains this function not to jump around wildly, or ">" will chop it into pieces that disobey the axiom. For instance, we cannot have

$$(\mathrm{V}\ s\ \mathrm{V0}) = \sin \frac{1}{(\mathrm{d}\ s) - t0}$$

in some chronicle, since then (> V0 0) will change truth value infinitely often around t0. One way to rule this out is to require that any such function be "finitely piecewise analytic," i.e., that, over any closed interval, the function consist of finitely many fragments that are analytic when extended to the complex plane. ("Analytic" means "infinitely continuously differentiable.") This set of functions is closed under arithmetic operations and differentiation, and always produce well-behaved facts when compared. Restricting ourselves to this set allows for all the discontinuities that quantities exhibit in naive physics, and seems to capture the intuition that normal quantities jump a few times, but basically vary smoothly.

We won't look very hard at requirements like this. We simply let Axiom 9 take its course. But Davis's result is needed to justify the axiom, since otherwise there might not be interesting models satisfying it.

The fundamental event involving fluents is a "vtrans." A (vtrans *v r1 r2*) denotes the event consisting of all occasions when *v* changed from *r1* to *r2*.

```
Axiom 19:
  (= (vtrans ?v ?r1 ?r2)
     {[s1, s2]: (and (= (V s1 ?v) ?r1)
                     (= (V s2 ?v) ?r2))}{)
```

For example, a winch's rotating corresponds to a vtrans of its phase angle. An increase in inflation is a vtrans of INFLATION from one value to another. A change of Presidents is a vtrans of "President of the US" from one statesman to another.

Knowing that a vtrans occurred tells you nothing about *how* it occurred, unless the quantity involved is *continuous*, when we have an intermediate-value axiom:

```
Axiom 20:
  (if (continuous ?v)
     (if (Occ ?s1 ?s4 (vtrans ?v ?r1 ?r4))
        (forall (r2 r3)
          (exists (s2 s3)
             (and (=< ?s1 s2 s3 ?s4)
                  (if (=< ?r1 r2 r3 ?r4)
                     (and (Occ s2 s3 (vtrans ?v r2 r3))
                          (forall (s)
                             (if (=< s2 s s3)
                                (=< r2 (V s ?v) r3)) )))
                  (if (> = ?r1 r2 r3 ?r4)
                     (and (Occ s2 s3 (vtrans ?v r2 r3))
                          (forall (s)
                             (if (=< s2 s s3)
                                (> = r2 (V s ?v) r3)) )))) ))))
```

In English, if *v* changes continuously from *r1* to *r4*, and *r2* and *r3* lie between *r1* and *r4*, then there is a time interval in which *v* changes from *r2* to *r3* without going outside the bounds *r2* and *r3*. That is, it spends a certain period in every subinterval between *r1* and *r4*. (The conclusion of the axiom has two very similar conjuncts, one for the case when *v* is increasing, the other for when it is decreasing.)

Vtranses are normally inferred from "potranses." If (potrans *channel v r*) occurs, that means that "*v* was augmented through the given channel by an amount *r*." A potrans is a potential vtrans.

Potranses are intended to capture the way we reason about things like flows into tanks, and other more general changes. Often we know things like these:

> I just poured five gallons of reagent into the vat.
> I made 5 thousand dollars consulting today.
> Decontrolling oil will tend to increase inflation by 5%.

In all these cases, we are given a fact which all by itself would translate directly into a vtrans: the vat's contents increased by five gallons, my net worth increased by $5,000, inflation increased by 5%. But, as we all know, life is not so simple. If you know about a leak in the vat, then the increase is actually 5 gallons MINUS pouring time * rate of leak. The IRS will make sure that my net worth doesn't go up by the amount I made. The Reagan administration hopes that other measures will offset the decontrol of oil.

I adopt a very abstract model of this kind of situation. Many quantities may be thought of as fed by various "channels." These may correspond to physical entities, such as pipes into tanks, but they are never identified with anything physical. They are there almost as a pure technical device to enable us to count potranses. We could not have a potrans of *r* into *v* be an event by itself, since then pouring five gallons into the same vat by two different pipes simultaneously would be just one occurrence of one event.

However, we do assume certain things about channels (which we denote by variables starting with the letter "h"). First, there is the fact (channel-into *h v*), for which we have the axioms:

```
Axiom 21:
  (iff (exists (s)
          (and (= < ?s1 s ?s2)
               (T s (channel-into ?h ?v))) )
       (exists (r)
          (Occ ?s1 ?s2 (potrans ?h ?v r))))
   (if (and (Occ ?s1 ?s2 (potrans ?h ?v ?r1))
            (Occ ?s1 ?s2 (potrans ?h ?v ?r2)))
       (= ?r1 ?r2))
```

That is, that one unique amount "flows" through a given channel into a given quantity over any interval. No amount at all flows unless the channel actually "fed" the quantity at some time during the interval.

The fundamental fact about potranses and channels is then:

```
Axiom 22:
  (if (real-valued ?v)
        (iff (Occ ?s1 ?s2 (vtrans ?v ?r1 ?r2))
             (= (- ?r2 ?r1)
                  (sumpotrans ?s1 ?s2 ?v
                    {h: (exists (s)
                          (and (= < ?s1 s ?s2)
                               (T s (channel-into h ?v))) )}))))
```

where

```
  (= (sumpotrans ?s1 ?s2 ?v { }) 0)
```

and

```
  (if (and (= (sumpotrans ?s1 ?s2 ?v ?hh) ?sum)
           (Occ ?s1 ?s2 (potrans ?h ?v ?r)))
      (= (sumpotrans ?s1 ?s2 ?v
                     (union ?hh {?h}))
         (+ ? sum ?r)))
```

That is, the change in a real-valued fluent over an interval is the sum of the potential changes in it. (sumpotrans *s1 s2 v set-of-channels*) is the sum of all the potranses through the given channels into *v* from *s1* to *s2*.

Taken together, these two axioms enable us to count the contributions from all channels into a quantity over a given time interval.

Potranses are decomposable:

```
Axiom 23:
  (iff (Occ ?s1 ?s2 (potrans ?h ?v ?r))
       (forall (s)
         (if (= < ?s1 s ?s2)
             (exists (r1 r2)
               (and (Occ ?s1 s (potrans ?h ?v r1))
                    (Occ s ?s2 (potrans ?h ?v r2))
                    (= ?r (+ r1 r2))) )) ))
```

That is, the potrans through a channel over an interval is the sum of the potranses over each subinterval in a partition of it.

If a quantity is continuous, we can decompose potranses into it another way. If a certain amount "flows" into or out of a quantity, then for any smaller amount, the flow began with a sub-flow of this smaller amount. Formally:

```
Axiom 24: (An intermediate-value axiom)
  (if (continuious ?v)
      (if (Occ ?s1 ?s2 (potrans ?h ?v ?r))
         (forall (r ')
           (if (or (=< 0 r ' ?r)
                   (>= 0 r ' ?r))
               (exists (s)
                 (and (=< ?s1 s ?s2)
                      (Occ ?s1 s (potrans ?h ?v r '))
                      (Occ s ?s2 (potrans ?h ?v
                                  (- ?r r '))))) )) )))
```

Potranses are not instantaneous:

```
Axiom 25:
  (if (T ?s (channel-into ?h ?v))
     (Occ ?s ?s (potrans ?h ?v 0)))
```

For example, let's say that we had:

```
(continuous (WATER-VOL TANK1))

(persist S0 (= {h: (channel-into h (WATER-VOL TANK1))}
               {(INFLOW TANK1), (OVERFLOW TANK1)})
            (6 weeks))
```

That is, only two channels into TANK1 exist. Notice how casually I sneak new constructs into the fact notation. The {*x*: *p*} is the set of all *x* such that *p;* this is, of course, a fluent. So (= {*x*: *p*} {A, B}) is a fact, with an obvious meaning.

The channels have certain special properties. Nothing ever flows in through the overflow, and there is no flow out of it while the level is below some capacity.

```
(if (Occ ?s1 ?s2
        (potrans (OVERFLOW TANK1) (WATER-VOL TANK1) ?x))
   (=< ?x 0))

(if (TT ?s1 ?s2 (< (WATER-VOL TANK1) (CAP TANK1)))
   (Occ ?s1 ?s2
        (potrans (OVERFLOW TANK1) (WATER-VOL TANK1) 0)))
```

Notice that the notation allows us to be ambiguous about whether (CAP TANK1) is a fluent or a number. If we decided on the former, we would have to talk about its persistence, so let's pretend it's the latter, and the capacity cannot vary with time. We assume that (> (CAP TANK1) 0).

Nothing ever flows out through the inflow:

```
(if (Occ ?s1 ?s2
        (potrans (INFLOW TANK1) (WATER-VOL TANK1) ?x))
   (>= ?x 0))
```

The tank is built so that the capacity is never exceeded:

```
(=< (V ?s (WATER-VOL TANK1)) (CAP TANK1))
```

Now, let's say that for some S1 and S3 soon after S0, we have:

```
(= (V S1 (WATER-VOL TANK1)) 0)

(Occ S1 S3 (potrans (INFLOW TANK1) (WATER-VOL TANK1)
                    (+ (CAP TANK1) (5 gal))))
```

Then we can infer that there is a state S2, such that:

```
(Occ S1 S2 (potrans (INFLOW TANK1) (WATER-VOL TANK1)
                    (CAP TANK1)))

(Occ S2 S3 (potrans (INFLOW TANK1) (WATER-VOL TANK1)
                    (5 gal)))

(Occ S2 S3 (potrans (OVERFLOW TANK1) (WATER-VOL TANK1)
                    (-5 gal)))
```

Proof: By Axiom 24, there is a flow of (CAP TANK1) through (INFLOW TANK1), followed by a flow of (5 gal). But during this period, the flow through (OVERFLOW TANK1) must be zero, because it can't be positive, and if it were negative, then by Axiom 22 the volume would never get above (CAP TANK1) during this interval, so it would always be zero, a contradiction. Therefore, at the end of this period, the volume will be (CAP TANK1). This is state S2. Now 5 gal flow into the tank. At least 5 gal must flow out, or the tank capacity would be exceeded at S3. If more than 5 gal flowed out (i.e., less than −5 flowed in), then at the end the tank would be less than full. Then by Axiom 20 there must have been an interval between S2 and S3 during which the volume of water declined from (CAP TANK1) to the final value. But during this interval, either the flow into INFLOW would have had to be negative, or the flow into OUTFLOW would have had to be nonzero, both of which are impossible—QED.

Other examples would be possible, but they would mainly illustrate reasoning about continuous functions. My main goal is the exploration of a logical framework, so I will leave this for somebody else.

Continuous quantities do not in general persist at the same value for very long. For example, the quantity of water in a reservoir will change with rain, evaporation and use. We could try to handle this by indicating that the persistence of time (= WATER-LEVEL k) is (say) one day. But this is almost never right. The level is not likely to stay exactly the same for more than an instant, but it is not likely to double in one day, either, no matter how hard it rains.

We need to introduce the "rate" predicate:

```
Axiom 26:
  (if (> ?t 0)
     (iff (T ?s (rate ?v ?t ?i))
         (forall (s0 s1)
           (if (and (=<  s0 ?s s1)
                    (= (- (d s1) (d s0)) ?t))
                             ?t
               (elt ---------------------
                     (V s1 ?v)-(V s0 ?v)

                ?i)) )))
```

(rate *v t i*) means that the average rate of change of the quantity *v* over any interval of length *t* is within the given interval. The purpose of *t* is to smooth short-term fluctuations, and to allow us to talk of rates of change of non-continuous quantities. (*t* is not allowed to be 0, since then we would have to talk about derivatives, which are hard to define given multiple chronicles, and which don't seem to be necessary for "naive" reasoning about time.)

We also need to delimit rates of potransing:

```
Axiom 27:
  (if (> ?t 0)
     (iff (T ?s (porate ?h ?v ?t ?i))
         (forall (s0 s1 r)
           (if (and (=<  s0 ?s s1)
                    (= (- (d s1) (d s0)) ?t)
                    (Occ s0 s1 (potrans ?h ?v r)))
               (elt (?r / ?t)
                    ?i)) )))
```

Rather than infer persistences of values of numerical quantities, we can infer persistences of their rates of change. I will give an example of such an inference in the next section.

5. PLANNING

With what I have talked about so far, we can reason about causal situations, but only as a spectator. In this section, I will talk about how a program might reason about its own actions. Part of my motivation in defining time the way I did was to support reasoning about interesting actions, like preventing events. The flexibility in my event ontology now carries over to the world of actions: An *action* is in this theory an entity, the doing of which by an agent is an event. Formally, we just need a function (do *agent act*), which is used to name events consisting of an agent performing an action. In this paper, I will completely neglect multiple-agent situations, so the first argument to "do" will be dropped; it will simply map actions into events. Variables denoting actions will begin with "a."

In the first subsection below, I will see how far this takes us. Some actions, like preventing and allowing, just fall out of the ontology. Others, like protecting facts, are still problematical.

In the second subsection, I will explore the notion of "plan." A *plan* is a set of actions, often intended to carry out another action. In one form or another, this idea has been important to several AI researchers, from Sacerdoti (1977) to Schank (1977). I will show how the idea gets translated into my temporal-logic framework.

5.1. The Logic of Action

Many actions are auite straightforward, such as (PUTON x y), which is done whenever the problem solver, or "robot," actually puts x on y. These may correspond to primitive actions the hardware can execute. For each, there will be axioms giving their typical effects as persistences.

But many actions do not fit this mold, such as preventing, allowing, proving, observing, promising, maintaining, and avoiding.

Consider the action "Prevent e," where e is some event. To be concrete, let's have e be the event E1 = "Little Nell mashed by train TR1 in the 5 minutes after state S0." E1 will be prevented if it doesn't happen, assuming it was going to happen if not prevented. (You can't take credit for preventing an unlikely thing.)

This is the sort of thing that past problem solvers have neglected. In the present calculus, it is easy to do. First, we need a notion of event dependence.

```
Axiom 28:
  (iff (not-occur-if ?e1 ?e2)
       (and (forall (ch)
                (if (hap ch ?e1) (not (hap ch ?e2))) )
            (exists (ch) (hap ch ?e2) )
            (exists (ch) (not hap ch ?e2))) ))
```

```
where
  (iff (hap ?ch ?e)
       (exists (s1 s2)
         (and (subset [s1, s2] ?ch) (Occ s1 s2 ?e)) ))
```

For instance, (not-occur-if E1 E2), where E2 = "I move Little Nell in the 5 minutes after S0."

Now it is easy to define prevention:

Axiom 29:

```
  (= (prevent ?e) (one-of {a: (not-occur-if (do ?a) ?e)}))
```

So one may to prevent Little Nell from being mashed is to move her in the next 5 minutes.

In this axiom, I have used "event disjunction," written (one-of {e1 e2 ...}), although this is just syntactic sugar for "sunion." We extend the notation to actions, with the axiom

Axiom 30:

```
  (= (do (one-of {?a1 ... ?aN}))
     (one-of {(do ?a1) ... (do ?aN)}))
```

If there are no actions that E is negatively dependent on, then (do (prevent E)) is the empty set, "never." That is, (prevent E) never happens.

Axiom 31:

```
  (= (do (one-of { })) never)
```

A finer analysis of impossibility appears below.

As an example, let us take another look at TANK1. Suppose that at S0,

```
(= (V S0 (WATER-VOL TANK1)) 0),

and (persist S0 (porate (INFLOW TANK1) (WATER-VOL TANK1)
                        (1 sec) [R1, R2])
                T0)

where (> R2 R1 0) and (> T0 (/ (CAP TANK1) R1)).
```

A little more terminology: Let (anch *s e*) stand for {[*s, s2*]: (Occ *s s2 e*)}, the set of all occurrences of *e* starting in *s*. Let (culm *p e*) be

```
{[s1, s2]: (exists (s)
              (and (=< s1 s s2)
                   (TT s1 s2 p)
                   (Occ s s2 e)))},
```

the set of all intervals in which *p* is true and then *e* happens. Let (holds *p*) be {[s, s]: (T s *p*)}, the set of all point intervals at which p is true.

What we want to find is an action to prevent

```
E0 = (anch S0
          (culm (rate (INFLOW TANK1) WATER-VOL TANK1)
                      (1 sec) <0, infinity>)
                 (holds (= (WATER-VOL TANK1) (CAP TANK1)))))
```

That is, the overflow of the tank that will occur if the water is allowed to run.

We need the action (TURN-OFF (INFLOW TANK1)), defined by this axiom:

```
Axiom 32:
  (pcause (channel-into ?h ?v)
          (do (TURN-OFF ?h))
          (porate ?h ?v (1 msec) [0, 0])
          1 [0, 0]
          (1 day))
```

This says that turning off ?h causes the flow through it to become zero. (The fourth argument, 1, says that the effect begins when the action is done; the fifth, [0, 0], says it happens immediately; and the sixth says it persists for one day. See Axiom 17.)

What we want to prove is that if E1 =

```
  (do (within-time S0 (/ (CAP TANK1) R2)
          (TURN-OFF (INFLOW TANK1))))

then
  (not-occur-if E1 E0)

where
  (= (do (within-time ?s ?t ?a))
     {[s1, s2]: (and (Occ s1 s2 (do ?a))
                     (=<  ?s s2)
                     (=< (- (d s2) (d ?s)) ?t))})
```

The (within-time *s t act*) is the action *act* done within *t* of state *s*.

The proof that turning off the tank in time will prevent the tank from filling requires three steps: (1) showing that E0 is possible; (2) showing that E0 might not happen; and (3) showing that E0 doesn't happen if E1 does.

The first requirement is met by a proof similar to that of Section 4. But to make it go, we have to assume there is a chronicle in which the problem solver refrains from E1. It is tempting to devise an "Axiom of Free Will," which states that any action is avoidable; there is always a chronicle in which you don't do it. But there are counterexamples. If A1 = "snap your fingers within 1 minute of S0," and A2 = "keep from snapping your fingers for 1 minute after S0," then (one-of {A1, A2}) happens in every chronicle containing S0. I will call such an action *unavoidable* in S0. There is no easy way to tell if an action is avoidable or not, so we must just provide axioms to tell in every case, which drive:

```
Axiom 33:
  (iff (avoidable ?a ?s)
      (exists (ch)
         (and (elt ?s ch) (not (hap ch (do ?a)))) ))
```

In the present example, we have (avoidable (TURN-OFF (INFLOW TANK1)) S0). E1 is not exactly in this form, but we have the theorem

```
(if (avoidable ?a ?s)
   (avoidable (within-time ?s ?t ?a) ?s))
```

If you don't have to do ?a, you don't have to do it within some time. I won't spend any time on the theory of avoidable actions, since it is probably intricate and essentially trivial.

So there is a chronicle in which E1 does not occur, By Axiom 15 and Axiom 16, in this chronicle the water keeps running, so we can infer that the tank will reach capacity during the lifetime of the water's being on.

The third requirement is met by assuming that E1 happens in CH1, and showing that the fact required for "culm" will be cut off. This is pretty obvious.

The second requirement will follow from the third if we can show that the robot is able to turn off this channel. Clearly, we need an axiom to deduce this. For realism, it should be an axiom giving the exact circumstances under which a channel of this sort can be turned off. (You have to be near enough to the tap implementing the channel that you can reach it before (CAP TANK1) / R2.) But this is all tangential, so I won't give details.

To talk about allowing, I first introduce a notion complementary to not-occur-if:

```
Axiom 34:
  (iff (occur-if-not ?e1 ?e2)
      (and (forall (ch)
              (if (not (hap ch ?e1)) (hap ch ?e2)) )
           (exists (ch) (hap ch ?e2) )
           (exists (ch) (not (hap ch ?e2))) ))
```

In English, (occur-if-not e1 e2) means that e2 will occur if e1 does not, and e2 may or may not occur.

We need a little bit more. First, the concept of "event negation," written "nev":

```
Axiom 35:
  (iff (Occ ?s1 ?s2 (nev ?e))
      (and (T ?s1 (possible ?e))
           (T ?s2 (not possible ?e)))))

where
  (iff (T ?s (possible ?e))
      (exists (s1 s2 ch)
        (and (elt ?s ch) (elt s2 ch)
             (Occ s1 s2 ?e)) ))
```

That is, the negation of an event occurs if that event becomes impossible. For instance, the negation of "Capitalism collapses by the year 1900" occurred in the last half of the nineteenth century.

Now we can define a related operator on actions, "forgo":

```
Axiom 36:
  (iff (Occ ?s1 ?s2 (do (forgo ?a)))
      (Occ ?s1 ?s2 (nev (do ?a))))
```

Forgoing an action means doing something that makes doing the action impossible, which may mean just procrastinating until you have lost your chance. It is hard to forgo an action like "Whistling the Star-Spangled Banner" (except perhaps by having your lips removed), but easy to forgo an action like "Move Little Nell within 5 minutes after S0." If you don't move her within 5 minutes, you've forgone this action.

Now defining allow is straightforward:

```
Axiom 37:
  (iff (Occur-if-not (do ?a) ?e)
      (= (allow ?e) (forgo ?a)))
  (if (not (exists (a) (occur-if-not (do ?a) ?e) ))
     (= (do (allow ?e)) never))
```

Related to allowing and preventing are two other actions, forgoing preventing and forgoing allowing. To forgo preventing something is to make it impossible to prevent; this differs from allowing in that the thing might still fail to happen, whereas according to my definition an event that is allowed actually happens.

Forgoing allowing an event is more complex. Let A be an action such that not doing it would entail that the event (E) occurs. To forgo allowing it is to forgo forgoing A. This means doing something that makes it impossible not to do A. This differs from preventing in that E might still occur.

Of course, there is a more mundane notion than either preventing or allowing, in which you actively work to make something happen. I will call this "bring-about." It is described by the axiom:

```
Axiom 38:
  (= (bring-about ?e)
     (one-of {a: (exists (r i)
                        (ecause always (do a) ?e r i))}))
```

Bringing-about *e* is done by doing an action that (always) causes *e*.

James Allen (personal communication) has raised interesting objections to my analysis of allowing and preventing, which I will repeat here, since they are likely to seem weighty to many people:

> Since most things are possible, however improbably, ...every day I allow most of the events that happen in China. If someone was killed there, then since I did 'forgo' the action of boarding a plane, sneaking through customs, and throwing myself in front of the assailant's bullet, I allowed the killing. It gets even worse with prevent. There's probably no way I can ever prevent anything in this world. I have so little control over what happens that whatever I do, there is always some event (however improbable) that is possible and would nullify my efforts.

The first objection, that too many things get allowed, is no trouble for me; one does in fact allow an infinite number of things over any given day, without intending to allow most of them. (Note also that "I allow most of the events...in China" is ambiguous; one certainly does not allow the event consisting of the occurrence of all the events in China over a day.)

The second objection, that too few things get prevented, is more serious. Of course, there is a sense in which one could never *prove* that it is possible to prevent a given event, but this is just another case of excessive caution on the part of formal systems. A dose of non-monotonicity should cure it, one hopes. A deeper problem is that my analysis fails to take probabilities into account. We often plan to prevent something, *knowing* that the plan might not work because of improbable possibilities. But this point applies to all planning, not just prevention. In fact, it applies to a lot of reasoning. As far as I know, there is no theory combining formal logic with probability theory.

In McDermott (1978b), I discussed a classification scheme for actions, used in the NASL problem solver. An important distinction was between "primary" and "secondary" actions. A secondary action was one that was executed correctly when another action was executed in a particular way. For instance, in "Pick up this stick without moving any other stick," the subaction "Don't move any other stick" was a secondary modification of the primary action "Pick up this stick." Another word for a secondary action to which the system was committed was *policy*.

In the present calculus, the distinction does not get made this way. Secondary actions are no weirder than some intuitively primary actions. For instance, (avoid a), where a is an action, is simply an action which is done over any interval in which you don't do a. The key distinction now is between composing actions sequentially or in parallel. Before, I defined (seq e1 ... eN) to be an event consisting of all occurrences of e1, ... eN in order. We can define (seq a1 ... aN) in a similar way. For policies, we must define (par a1 ... aN):

```
Axiom 39:
  (= (par ?e1 ... ?eN)
     {[s1, s2]: (and (elt [s1, s2] ?e1)
                     (elt [s1, s2] ?e2)
                     ...
                     (elt [s1, s2] ?eN))})
```

So to do A1 while avoiding doing A2, we do (par A1 (avoid A2)). (Here and from now on, I extend notations defined over events to actions in the obvious way without comment.)

Another secondary action is "protection" Sussman (1975). Intuitively, a fact is protected by a problem solver during an interval if it stays true during that interval. However, I think there is more to protection than this, which I do not know how to formalize. There is a distinction between "restorable" and "unrestorable" protections. For instance, if you are protecting the fact, "The fuse (for some keg of dynamite) is not in contact with an open flame," then if the fact becomes false, you have failed. You might try to cut the fuse or run, but it is pointless to move the fuse away from the flame; the damage has been done. In most cases, though, it is worth it to reestablish the protected fact. If I am baby-sitting a child, I try to protect the fact that he is not out of my sight. I do not give up once he is invisible. So the act "Protect *p*" can often be successful, even if *p* lapsed a few times. I do not know how to formalize this. Perhaps you could put a time limit on how long the lapses are. So in the baby case, I have failed if he eludes me for more than 5 minutes, while in the dynamite case the maximum allowable lapse is zero. But this seems arbitrary. The only real criteria for success of actions like these are teleological. I am successful with the baby if he's around and in one piece when his parents arrive. I am successful with the dynamite if there's no explosion, and so forth.

I have already mentioned several ways of building new actions out of elementary actions, such as seq, par, forgo, and avoid. Another important class of action-building methods are the traditional programming constructs, like loops and conditionals. A complete study of how these composition methods fit together would start to resemble the study of programming-language semantics Milne (1976). I think we should avoid carrying this resemblance to an extreme. In particular, I think the ability to do simple reasoning about plans would go down the drain if variables and assignment were admitted into the plan language. In most loops that people execute, the outside world keeps all the state information. When a condition is no longer true, it will be false in the world, not in the robot's head.

5.2. The Logic of Problem Solving

So far in this section, I have analyzed actions, without ever introducing the concept of an action that "should be performed." A problem solver may be thought of as a program that takes an action that should be performed, a *task,* and performs it. Hence the notion is of some importance.

I tried once before, in McDermott (1977; and 1978b), to develop the logic of tasks. In that system, NASL, the fundamental predicate was (task *name act*) meaning, *name* denotes an action you should do, to wit, *act.* Unfortunately, the action of a task was usually underspecified. For instance, you might have the tasks:

```
(task T1 (PUTON A B))
(task T2 (PUTON B C))
(successor T1 T2)
```

This was where (PUTON B C) was an action that should be performed, but not at an arbitrary time; the "successor" formula constrained it to be after (PUTON A B).

The problem with this approach was that it distorted time relations, in three ways. First, the time dependence between the two actions was not part of their definition. This made it hard to say what "task" meant. If it meant "This action is to be done," then a task assertion didn't describe its action precisely, but only gave a generalization of it. (In the example, (PUTON B C) is a generalization of "Do (PUTON B C) after (PUTON A B).") Second, it wasn't made clear *when* something was a task. As with most previous AI representations, NASL lived only in the present; there was no way to talk about what had been a task or was going to be one. Third, to compensate for this, NASL changed the data base to reflect passing time. When something was no longer a task, it got erased. Unfortunately, when something had been assumed to be a task erroneously, it also got erased. There was no way to distinguish between these two (see Section 1).

In short, three notions of time were confused: the time of an action, the time of a task, and real-world time ("now"). Now that we have a good analysis of time, we can untangle these things.

The correct analysis of task and action seems to be this: A task is an action to which a problem solver is committed. The action must be well enough specified so that the time of commitment is not needed to know what it is the solver is committed to. Therefore, one may have a task like "Visit Greece," satisfiable any time, but usually the action must be more specified than that: "Put block A on block B within 5 minutes after . . . ."

A problem solver's being committed to an action is itself a fact. One may alternately have and not have the task of Visiting Greece. Entirely independently, one may actually visit Greece. There are several ways these might interact:

1. You might have the task and not have done the action yet: In this situation, a rational problem solver will devote resources to accomplishing the action, unless more urgent tasks intrude.
2. You might have the task and have already done the action: In this case, the task has succeeded, and nothing more need be done.
3. You might have done the action and not (now) have the task: This is quite common; an example would be insulting or cheering up someone yesterday without intending to now (or possibly then either).
4. You might have a task for an impossible action: This is quite common too. The action may have been possible when the task began; in this case, the task may be said to have *failed.* If it was never possible, it is wishful thinking. Philosophers have argued about whether it is ever rational to have such a task. I see no reason why not, in the case of task failure. It seems natural to say that I have a task of meeting that student at 2:30 yesterday as I promised, but I failed to do it.

The last two categories interact in an interesting way. Often when a task fails, there is some other action that was done *instead* of the intended one. For instance, you have a task of hitting the golf ball into the little hole, and you actually hit it into the big pond. Here is a combination of an action with no task and a task with no (possible) action. There should probably be a predicate relating the two: (did-instead act1 act2) would mean that act1 was a task and act2 was done instead (act1 never occurring). (In the example, act1 would be "Hit the ball into the hole on stroke 2 of hole 6 of the golf game played on Tuesday afternoon," and act2 would be "Hit the ball into the pond on stroke 2 of . . . .") Rather than examine this in detail, I will just point out an inadequacy in past representations of task networks. Problem solvers that maintain such networks (Sacerdoti, 1977; McDermott, 1978b) have failed to maintain a complementary *behavior network* that represents what actually happened (or is going to happen). If every task succeeds, the two networks would be isomorphic; where one had "Task a," the other would have "Did a." But if there was failure, there would be a link between the two networks, from "Task a," to "Did b." This would be more useful than simply recording that a task failed or succeeded, and would help the system in explaining its actions.

This is getting ahead of the story, into implementation and away from logic. We need to say more about logic first.

In both NOAH (Sacerdoti, 1977) and NASL (McDermott, 1978b), a key notion is that of one task being a *sub-task* of another. This means that the sub-task is part of the chosen plan for carrying out the super-task. Every task is either immediately executable or reducible to sub-tasks, which are executable or reducible, and so on.

A problem solver transforms a task into sub-tasks by *choosing* a plan for the task, and asserting that every element of the plan is a sub-task. This choice mechanism is probably not purely logical. That is, it seems that the solver probably doesn't *infer* a set of sub-tasks, but must actively choose them, whatever that means.[8]

The requirement that tasks be reduced to sub-tasks gives rise to a bug. In the current formalism, we can talk of reducing the action A1 to the action (seq B1 B2), but this reduces A to a single sub-task. Is there any sense in which B1 and B2 are sub-tasks of A? It makes sense for B1 to be thought of as a sub-task, but just any execution of B2 will cut no ice. We insist that B2 come after B1. To make B2 a sub-task would get us right back into the difficulty I raised at the beginning of this section, of tasks being underspecified.

The solution is to take as sub-tasks B1 and (just-after B1 B2), where just-after is defined as

```
Axiom 40:
  (= (just-after ?a1 ?a2)
     {[s2, s3]: (and (Occ s2 s3 ?a2)
                     (exists (s1) (Occ s1 s2 ?a1)))})
```

The set of actions {B1, (just-after B1 B2)} is such that A is executed in any chronicle in which they are executed; in other words, this set is a plan for A.

[8] Perhaps I'm wrong on this. But if this relationship really is inferential, it must be an inference of the form: if p is the best plan for a, then every element of p is a sub-task of a. Unfortunately, it can happen that there are two equally good plans for a. Since we need to introduce a pure choice here, we may as well accept it in general. Amazingly little has been done on the logic of choices in AI. The work on medical diagnosis (e.g., Shortliffe, 1976), and work on choices by problem solvers (e.g., McDermott, 1978b; Doyle, 1980) are two examples.

Of course, this is not really a solution to the problem, not until we provide rules for reducing tasks of the form (just-after a1 a2). But at least it suggests that what problem solvers do makes some logical sense. That is, in many cases, there is a well-defined plan for a task, each of whose elements must be done in order to carry out the task.

Another example is the classic plan for achieving a conjunction of facts. To analyze this, we need the action (achieve *prop until-prop*), which means "Bring it about that prop is true from the end of the achieve until the until-prop becomes true." This is clearly needed, for the reasons discussed in Section 3; if you were allowed to achieve things for a single instant, the achievement would usually be worthless. So we have:

```
Axiom 41:
  (iff (Occ ?s1 ?s2 (do (achieve ?p ?q)))
         (and (T ?s2 ?p)
              (forall (s4)
                 (if (and (< ?s2 s4) (not (T s4 ?p)))
                     (exists (s3)
                       (and (<  ?s2 s3) (=<  s3 s4)
                            (T s3 ?q)) )) )))
```

In English, doing (achieve ?p ?q) amounts to bringing it about that ?p, in such a way that if ?p ever becomes false thereafter, ?q must have become false first.

Historically, tasks of the form "achieve *p*" have been very important. Problem solvers like GPS (Ernst, 1969) concentrated on these (in the form of "difference reductions"), and this concentration has persisted. An especially interesting case is where *p* is a conjunction of facts (Sussman, 1975; Sacerdoti, 1977). The problem is, of course, that all the facts must be true at once, when the task is complete, and all too often the plan for one conjunct upsets another.

I will introduce the standard plan for achieving conjunctions after some useful definitions. First, we define "to-do" thus

```
Axiom 42:
  (iff (to-do ?a1 ?a2)
       (one-way (do ?a2) (do ?a1)))
```

That is, *a2* is a way to do *a1* if (do *a2*) is one way that (do *a1*) can happen. Next, we let (plan *aa*) be the action corresponding to the plan consisting of all the actions *aa*.

```
Axiom 43:
  (iff (Occ ?s1 ?s2 (do (plan ?aa)))
       (and (forall (a)
            (if (elt a ?aa)
               (Occbetween ?s1 ?s2 a)) )
         (exists (a s)
           (and (elt a ?aa) (Occ ?s1 s a)) )
         (exists (a s)
           (and (elt a ?aa) (Occ s ?s2 a)) )))
```

That is, a plan is done in any minimal time span in which all of its elements are done. And finally,

```
Axiom 44:
  (iff (T ?s (finished ?a))
      (and (exists (s1 s2)
              (and (Occ s1 s2 (do ?a))
                   (<  s2 ?s)) )
           (forall (s1 s2)
             (if (=< ?s s2)
                (not (Occ s1 s2 (do ?a)))) )))
```

That is, an action is finished when its last execution is past.

The following theorem then states that one way to achieve a conjunction is to achieve each of its conjuncts, in such a way that each conjunct remains true until the other is achieved.

```
(if (and (=  ?a1 (achieve ?p1 (& ?q (finished ?a2))))
         (= ?a2 (achieve ?p2 (& ?q (finished ?a1)))))
   (to-do (achieve (& ?p1 ?p2) ?q)
          (plan {?a1, ?a2})))
```

Proof: Assume that the task is to achieve P1 and P2 until Q, and let A1 and A2 be two actions that satisfy the antecedent. Assume that (do (plan {A1, A2})) occurs from S0 to S2. I will show that (do (achieve (& P1 P2) Q)) also occurs during that interval. According to Axiom 41, we must show that (T S2 (& P1 P2)), and that (& P1 P2) remains true until Q.

To show the first part, without loss of generality assume that A1 finishes before A2 (or no later). (See Figure 4.)

```
            ——A1 –>          (&P1 P2)     Q      (–P1)
SO                    S1        S2         S3       S4
               ———  A2       –>
```

Then A2 is not finished until at least S2, so P1 must be true from S1 to S2. But P2 is true at S2, so (T S2 (& P1 P2)).

To show the second part, let S4 be an arbitrary state after S2 in which (& P1 P2) is false. Then either P1 or P2 is false there. Assume without loss of generality that it is P1. By Axiom 41, there must be a state S3 after S1 in which (& Q (finished A2)) is true, and hence Q is true. This must come after S2, since until then A2 is not finished—QED.

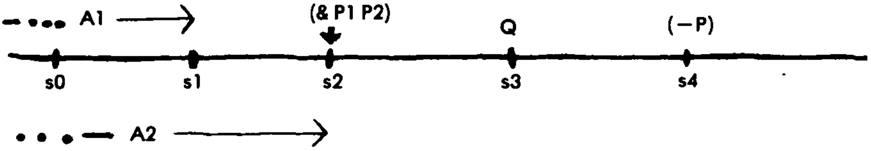


Figure 4. Conjunction Proof

There is one suspicious feature of this plan for conjunction achievement: there is no finite non-circular term for naming either sub-task. This rules out certain naive implementations of a problem solver based on this logic. A deeper problem is that there may be cases in which there are no actions satisfying the antecedent of my theorem, for instance if ?p1 and ?p2 contradict each other, or they require large amounts of a finite resource, or any of several other cases obtains. It is an open problem how one would go about proving the feasibility or unfeasibility of this plan.

The final problem to be examined in the light of this logic is the "plan decomposition" problem (pointed out by Eugene Charniak, personal communication). When you are writing a plan, say to paint something (Charniak, 1976), you have a choice whether to represent a step as "Dip brush in paint," or "Achieve (paint on brush)." The latter is the purpose of the former, but the former is the most common way of achieving this purpose, so common that it seems wrong not to make it part of the plan. But if the paint in the can is low, the usual step will not work. You will have to tilt the can and grope with the brush, or go buy more paint. On the other hand, it seems wasteful to make "Achieve (paint on brush)" the normal plan step, and rederive the normal sequence every time.

The solution seems to be to store two things: the usual plan, and a proof that it works. The proof in this case would have one fragment that said: "If you dip the brush in the paint, and there is paint deep enough to be dipped into, then you will get paint on the brush. If there is paint on the brush, then stroking the wall with it will get paint on the wall. . . ." The proof is not consulted until the plan fails, that is, until the "dip" step fails to bring about partial immersion of the brush. Then the proof would be consulted to see why this was done. The reason found would mention the bridging fact that "there is paint on the brush." The problem solver could then look for other ways to bring this about.

This sketch requires a lot of work to fill out, but I doubt that problem solvers will be robust and efficient until it is done. One big piece of work is to choose a format for these "proofs" that enables easy access to the relevant fragments. Presumably the proof would be broken into pieces festooned over the plan.

6. SKETCH OF AN IMPLEMENTATION

Up to now, I have avoided discussing data structures and algorithms. If Hayes (1979a) is right, I would be justified in avoiding them for a few more years yet. But I have not been able to keep from thinking about how these ideas would be expressed in a working program, and I think an occasional glimpse in that direction is necessary for even the most dedicated "AI logician." We are engaged in notational engineering, not philosophy.

An implementation must be able to do interesting, useful inferences. What is interesting and useful will vary from application to application. The one I am most interested in is problem solving. A problem solver must exploit the tree of possible chronicles, since it must reason about consequences of different courses of action. It must also be able to reason about the interactions between its actions and inanimate processes, and among its own actions. A typical interaction is the detection that a planned action will cause a persisting fact to cease.

Consider an example from (Sacerdoti, 1977). Say a problem solver has the tasks of painting the ladder and painting the ceiling. If it works on "paint the ceiling" first, it will notice that the ladder must be climbable, and that it is currently climbable. Therefore, it will persist in being climbable for years. The problem solver concludes that this state will last until the ceiling is painted, which will take a few hours. Now it turns to thinking about painting the ladder. It realizes that this will cause "ladder climbable" to cease, and remain untrue for a day (assuming paint dries this slowly). It then should see that it lacks sufficient information to decide if this is a problem, since it does not know whether it will paint the ladder before painting the ceiling. Since the situation is under its control, it imposes an order that didn't exist before, and decides to paint the ceiling first.

This is similar to Sacerdoti's algorithm, but with some important differences. First, the kind of retrieval that occurs is more generally applicable. If we found out someone was coming to repossess the ladder, exactly the same reasoning would go on, up to the point where we imposed extra order. A different response would be necessary if one of the events was outside our control. But the retrieval problem is the same.

Second, I do not model an action in terms of simple "addlists" and "deletelists," that is, lists of facts that change in truth value as a result of that action. Painting the ladder renders it unclimbable, but only for a while; we could always paint the ceiling tomorrow. In fact, there is no guarantee that we will catch the problem before we have already painted the ladder. Even if we do catch it, there may be some pressing reason why we should paint the ladder first; for instance, we may want to paint the ladder blue, and the ceiling green, and the only way to get green may be to mix our blue with some yellow we have lying around.

In fact, there will in general be many factors on each side of an ordering decision, and I am skeptical that one can casually decide on the basis of one of them how the plan ought to go. Instead, it seems more reasonable to try simulating the plan both ways whenever there is an important uncertainty as to order or outcome (Wilensky, 1980).

For this to work, the implementation must recognize the existence of multiple chronicles. It might seem that we want to keep a description of every relevant chronicle, but, of course, there are an infinite number of chronicles, each with an infinite description; what we really want is a partial description of the typical element of an interesting set of chronicles. For instance, the set of all chronicles in which I fail to prevent Little Nell from being mashed by the oncoming train would be of (somewhat morbid) interest, as would the set of chronicles in which I succeed. A partial description is just a data structure that supports information retrieval, like the action-conflict detection I described before. Let us call this kind of data structure a "time line," without reading too much into the phrase. Every set of chronicles will be represented by a data structure called a *chronset,* which consists of a defining characteristic of the chronicles in the set, plus a time line for accessing the events and facts that occur in those chronicles.

Chronsets are hierarchically organized. When the problem solver detects an important uncertainty in a chronset, it creates two (or more) new chronsets which represent the different outcomes. Almost everything true in the original chronset is true in the new ones; if I am on my way to visit Grandma when I hear Nell's cry for help, then the fact that I will see Grandma tonight is still true in both chronsets. Furthermore, the same chronset can be split more than one way. Before getting involved with Nell at all, I might have been speculating on whether nuclear war would occur by the year 2000, and what that would mean for civilization. The chronsets connected with this possibility have nothing to do with Nell.

Eventually, only one of a pair of alternative chronsets turns out to correspond to reality. This one becomes the new basis for further planning.

So, however time lines are implemented, they will have to be able to inherit properties from "superlines" belonging to higher chronsets. A flexible model of this kind of inheritance is the "data pool" model, developed in Sussman (1972) and McDermott (1981b). This allows a distributed data structure to be labeled so that different parts are "visible" in different data pools. Each data pool will correspond to a chronset. So, rather than have different time lines, we can have one big time line, with some parts invisible in some chronsets.

The next question is how time lines are implemented. The idea I am currently pursuing is that they are modeled "spatially," that is, using much the same machinery as in a spatial reasoner like that of McDermott (1980b), Davis (1981), and McDermott (1981c).

In our spatial reasoner, every entity is modeled as a "frob"—a frame of reference attached to an object. The frobs are arranged in a graph. If the position of an object is known (fairly precisely) with respect to another object, its position with respect to that object's frame is stored explicitly; otherwise, it is computed when needed. Questions such as, How far is it from A to B? are answered by computation on the coordinates of A and B. Questions such as, What object is near A? are answered by searching through a *discrimination tree* of objects stored with respect to A (McDermott, 1981c, d).

Our working hypothesis is that events and facts can be modeled as frobs. The reason this approach may fail is that the frob graphs may just be too complicated; however, it is hard to think of a more promising approach. (But see Allen, 1981.)

In general, a frob's position and other features are "fuzzy," that is, known only to within an interval. Hence we call the aggregation of frobs a *fuzzy map*. The fuzziness is entirely due to uncertainty. The position of the object in the real world is assumed definite. (Objects are not quantum mechanical.) If an event is to be thought of as a frob, there must be a sense in which it is a definite object with uncertain attributes. Of course, this is not what an event is at all. Instead, it is an infinite collection of time intervals. The time during which I sang the Star Spangled Banner is a meaningless quantity, unless you mean the fuzzy interval of all dates from my first singing of it to my last. But this interval will never be reduced to a point by further information.

On the other hand, there does seem to be a notion of temporary uncertainty that gets resolved. I am not sure what time the plumber is coming tomorrow; after she had been here, I am sure. This notion is completely outside the realm of the logic I developed in Sections 1 through 5. Consider a problem solver at state S0, with a time line including tomorrow, and an event "Plumber comes." It is simply wrong to say that there is uncertainty in what time the plumber is going to come in the day following S0, because *there are lots of 24-hour periods following S0,* one per chronicle. Twenty-four hours later, there will be an infinite number of problem solvers, in an infinite number of incomparable states following S0, each with a slightly different idea of when the plumber came.

So for its own sanity, a problem solver is going to need the notion of the *real* chronicle, the one that is actually going to happen. Actually, for completeness, we will have every chronset contain a unique *realest* chronicle, which must be the real chronicle if the chronset contains it. The uncertainty surrounding the exact time of an event in a given chronset is then the uncertainty about the occurrence of the event in the realest chronicle in the set. And this only makes sense for events that happen at most once in a chronicle: We will call an event or fact being modeled in a time line in this way an *occurrence.*

With these restrictions, it makes sense to apply techniques for mapping time. The existence of chronsets merely forces there to be many competing maps.

Before I discuss time lines in more detail, let me issue a warning about the "real" chronicle and its relatives. I am convinced that no hint of this concept must appear in the logic, because it would lead to some serious paradoxes and a breakdown of the system. (I thank Ernie Davis for discussions leading to this conclusion.) For instance, how do you represent that something is inevitable? In the logic so far, you must say that it will happen in all chronicles. It seems tempting to explore the alternative way of putting this, that the thing will happen in the real chronicle. After all, what can it matter that something happens in an unreal chronicle? But then everything that actually happens was inevitable.

The only conclusion is that the logic we use makes some extreme assumptions about time, which our implementation resolutely ignored. If this bothers you because you think logic ought to encompass everything that goes on in a robot, then this should convince you that it can't. If this bothers you because you want to know who is right, the logic or the implementation, my guess is that the implementation is right, but so what? Neither alternative is very palatable, but neither can be neglected. A system that accepted the idea of many futures would have no grounds for any decision; but neither would a system that accepted the idea of one future. The trick is to resonate between them, betting that there is one real future that matters, relying on a logic that presupposes the opposite.

One other topic falls under the "Implementation" heading. A *data dependency* is a note of the support that an assertion has, expressed as a list of other assertions (Doyle, 1979). In the implementation I am describing, there will be two kinds of dependency: the support for the contention *that* an occurrence will take place in a chronset; and the support for the time *when* it occurs in that chronset. The former is relatively straightforward. A cause will be linked to its effect. A bad occurrence will be linked to the task that prevents it. The only complication is that these links may have to cross chronset boundaries; for example, a task might be there because in another chronset, something bad will happen.

The second kind is more problematic. Times of occurrence are not asserted, but constrained. As constraints accumulate, they become more precisely known, just as in McDermott (1980b). How to erase such constraints is still an open problem in the spatial domain, and may also be a problem in the temporal domain.

Consider how this data-dependency system would solve the "Little Nell" problem I started with. Once the system (Dudley) has reasoned out the causal sequence involving the train and Nell, and sees that a bad event is going to happen, it looks for a plan to prevent it. Assuming it finds a candidate, it sets up an alternative chronset in which this plan is successfully executed. (See Figure 5.) It does an analysis of feasibility, and decides that this chronset probably corresponds to reality more closely than the one it started with. However, the data dependency supporting the assertion that "Move Nell" is a task specifies that the occurrence of "Nell is mashed" in the other, original chronset as the justification of the task. It is irrelevant that this chronset is not expected to be realized.

It *is* relevant that in the alternative chronset, she gets mashed. This assertion will be supported by a record of the causal argument (inv^lving the persistence of being tied up, the train schedule, and so on) that led to Dudley's alarm in the first place. If this argument is upset, say by a new assertion that Dick Daring is planning to free Nell in two minutes (thus terminating a crucial persistence), then it is no longer true, even in the alternative chronicle, that Nell is in danger, and the assertion "I have the task of moving Nell" will disappear from Dudley's data base.

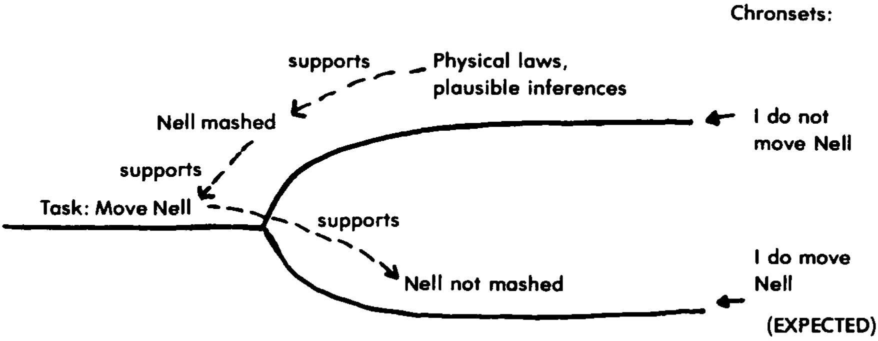


Figure 5. Tree of Chronsets for the Dudley-Nell Problem

As another illustration, let me sketch how this system would handle one straightforward kind of inference—system simulation. This kind of inference is the result of applying ecause and pcause rules to see how a system will behave. That is, starting in some state, we use these rules to predict future states, then start from there to predict more states, and so forth. Each application of a rule creates a new frob, corresponding to the caused effect. This frob will represent a persistence or event. It is also a frame of reference for further simulation; its effects will be frobs fuzzily located in its frame, and so on. Figure 6 shows how each occurrence is located more or less fuzziiy, at some offset in the frame of its cause. Each effect then serves as a frame for the next round. Once the structure is built, it can serve to answer queries, like "How soon after F1 will F3 occur?" This requires translating F3's fuzzy coordinates back into frame F0 for comparison. The more steps of translation, the fuzzier the coordinates get.

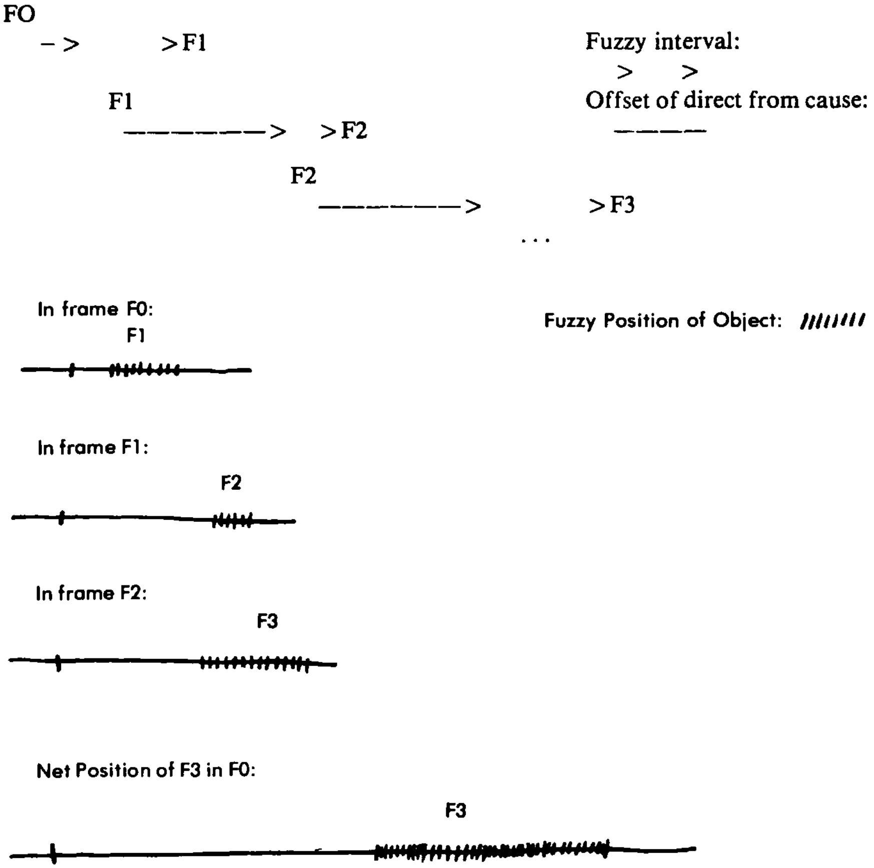


Figure 6. Each Frob is a Frame for the Next Occurrence

Just storing the coordinates does not suffice for answering questions such as "What's the first occurrence of. . .after F1?" This requires other sorts of indexing. (McDermott, 1981d)

This sketch is intended only to suggest what one might do. I feel that raw simulation of this sort is actually of little value, except for simple loop-free systems. If a loop is encountered, the unwary simulator will itself go into a loop. Instead, it should be on the lookout for "loopy" patterns, such as a quantity increasing twice, and try to step back and draw more interesting conclusions. I can only point at this problem here, and not hope to solve it.[9]

[9] I have implemented a preliminary version of a program for reasoning about simple mechanisms, including some with loops, and will report on it in a later paper.

7. CONCLUSIONS

I set out to develop a temporal logic which captured the openness and continuity of time. To this end, the basic ontology had to include states arranged into a branching set of continua, the "chronicle tree." Doing this enabled us to state facts about causes and effects, continuous change, and plans. In many cases, we could make useful deductions about the course of events. Here is a list of some of the situations considered:

- Causal sequences, including infinite loops
- Continuous change up to some threshold
- Actions taken to prevent the operation of causal systems
- Conflicts among actions done in the wrong order (cf. Sacerdoti, 1977)
- Changes in one's plans forced (or not forced) by changing circumstances

I look at some of these systems more formally than others, for which I emphasized implementational considerations.

I have found that logic and implementation fertilize each other. One often has a vague notion of what he wants a program to do, plus a pile of special cases that don't fit together too well. Sometimes one goes ahead and implements the special cases. I urge consideration of the alternative: temporarily to ignore hard-nosed programming issues, and try to achieve an elegant synthesis of the special cases in the logical domain. If you fail, it is likely that the logical bugs holding you up would have caused the program to exhibit bizarre behavior anyway. If you succeed, the results can often be transferred back to the programming domain. The ontology of the logic will be reflected in the data structures of the program (as chronicles gave rise to chronsets); the commonly encountered proofs will give rise to inference algorithms, and records of them become data dependencies, which help to make the program robust and less opaque. Of course, the program will fail to make inferences the logic allows (and hence, via non-monotonicity, jump to conclusions the logic forbids), but humans have these limitations too.

REFERENCES

Allen, J. *Maintaining knowledge about temporal intervals* (Technical Report TR86). University of Rochester, Department of Computer Science, 1981.

Charniak, E. *A framed PAINTING: the representation of a common sense knowledge fragment.* Working Paper 28, Fondazione Dalle Molle, 1976.

Charniak, E. A common representation for problem-solving and language-comprehension information. *Artificial Intelligence,* 1981 *16* (3), 225-255.

Davidson, D. In Rescher (Ed.), *The logical form of action sentences*. Pittsburgh, PA: Pittsburgh University Press, 1967.

Davis, E. *Organizing Spatial Knowledge* (Technical Report 193). Yale University, Computer Science Department, 1981.

Doyle, J. *A truth maintenance system* (Memo 521). MIT AI Laboratory, 1979.

Doyle, J. *A model for deliberation, action, and introspection* (TR 581). MIT AI Laboratory, 1980.

Ernst, G. W. & Newell, A. *GPS: A Case Study in Generality and Problem Solving*. New York: Academic Press, 1969.

Fikes, R. & Nilsson, N. J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence,* 1971 *2,* 189-208.

Goodman, N. The problem of counterfactual conditionals. *Journal of Philosophy,* 1947, *44,* 113-128.

Hayes, P. *The Naive Physics Manifesto*. Unpublished, 1979.

Hayes, P. *Ontology for Liquids*. Unpublished, 1979.

Hendrix, G. Modeling simultaneous actions and continuous processes. *Artificial Intelligence,* 1973, *4,* 145-180.

Hewitt, C. *Description and theoretical analysis (using schemata) of PLANNER: a language for proving theorems and manipulating models in a robot* (Technical Report 258). MIT, AI Laboratory, 1972.

Johnson-Laird, P. N. Mental models in cognitive science. *Cognitive Science,* 1980, *4* (1), 71-115.

Lewis, D. K. *Counterfactuals*. Oxford: Basil Blackwell, 1973.

McCarthy, J. *Programs with common sense*. In Minsky (Ed.), 1968.

McCarthy, J. Circumscription: a non-monotonic inference rule. *Artificial Intelligence,* 1980, *13* (1, 2).

McDermott, D. V. *Flexibility and efficiency in a computer program for designing circuits* (Technical Report 402). MIT, AI Laboratory, 1977.

McDermott, D. V. Tarskian semantics or, no notation without denotation!. *Cognitive Science,* 1978, *2* (3). (a)

McDermott, D. V. Planning and acting. *Cognitive Science,* 1978, *2* (2), 71-109. (b)

McDermott, D. V. & Doyle, J. Non-monotonic logic I. *Artificial Intelligence,* 1980, *13* (1, 2). (a)

McDermott, Drew V. *Spatial inferences with ground, metric formulas on simple objects* (Technical Report 173). Yale University, Computer Science Department, 1980. (b)

McDermott, D. V. Non-monotonic logic II: non-monotonic modal theories. *Journal of ACM,* 1981. (Also Yale CS TR 174). (a)

McDermott, D. V. *Contexts and data dependencies: a synthesis*. Submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence, 1981. (b)

McDermott, D. V. & Davis, E. Planning and executing routes through uncertain territory. Submitted to *Artificial Intelligence,* 1981. (c)

McDermott, D. V. *Finding objects with given spatial properties*. (Technical Report 195). Yale University, Computer Science Department, 1981. (d)

Mendelson, E. *Introduction to Mathematical Logic*. Van Nostrand, 1964.

Milne, R. & Strachey, C. *A Theory of Programming Language Semantics*. Halsted Press, 1976.

Minsky, M. (Ed.). *Semantic Information Processing*. Cambridge, MA: The MIT Press, 1968.

Montague, R. On the nature of certain philosophical entities. *The Monist,* 1960, *53,* 159-194. (Also in Montague, 1974.)

Montague, R. In R. Thomason, (Ed.), *Formal Philosophy*. New Haven, CT: Yale University Press, 1974.

Moore, R. *Reasoning about knowledge and action* (Technical Report 191). SRI AI Center, 1980.

Prior, A. *Past, Present, and Future.* Oxford University Press, 1967.

Reiter, R. A logic for default reasoning. *Artificial Intelligence,* 1980, *13* (1, 2).

Rescher, N. *The Logic of Decision and Action.* Pittsburgh, PA: Pittsburgh University Press, 1967.

Rescher, N. & Urquhart, A. *Temporal Logic.* New York: Springer-Verlag, 1971.

Rieger, C. The commonsense algorithm as a basis for computer models of human memory, inference, belief and contextual language comprehension. *Proc. Theoretical Issues in Nat. Lang. Processing Workshop.* Boston, MA, 1975.

Rieger, C. An organization of knowledge for problem solving and language comprehension. *Artificial Intelligence,* 1976, 1.

Sacerdoti, E. *A Structure for Plans and Behavior.* American Elsevier Publishing Company, Inc., 1977.

Schank, R. C. *Conceptual Information Processing.* American Elsevier Publishing Company, 1975.

Schank, R. C. & Abelson, R. P. *Scripts, Plans, Goals, and Understanding.* Hillsdale, NJ: Lawrence Erlbaum Associates, 1977.

Shortliffe, E. H. *Computer-Based Medical Consultations—MYCIN.* American Elsevier, 1976.

Sussman, G. J. & McDermott, D. V. From planning to conniving—a genetic approach. *Proc. FJCC 41,* 1171. IFIPS, 1972.

Sussman, G. J. *A Computer Model of Skill Acquisition.* American Elsevier Publishing Company, Inc., 1975.

Wilensky, R. *Metaplanning.* (Memo UCB/ERL M80/33). Berkeley Department of Computer Science, 1980.