Computational Considerations in Reasoning about Action

Matthew L. Ginsberg
Computer Science Department
Stanford University
Stanford, CA 94305

Abstract

Any first-principles declarative planner will need to be constructed from an underlying declarative system that reasons about action. In this paper, we point out that if such a planning system is to be computationally viable, the associated declarative description of actions must satisfy at least two broad conditions. First, it will need to be *event-driven*, so that fluents that hold in a particular situation can be propagated into the future at reasonable computational cost. Second, it will need to be *anytime* in the sense that partial or approximate answers to queries can be provided in the presence of computational resource constraints. We suggest that the first these goals can be achieved by taking the truth values assigned to sentences to be functions from the temporal elements into a set of basic values, and that the second can be achieved by viewing temporal operators as functions on these functional truth values.

1 INTRODUCTION

Existing planning systems can be grouped into three broad categories: expert planners, general-purpose planners, and first-principles planners.

Expert planners, of which there are many, are essentially applications of expert-systems technology to planning problems. The situation in which a particular agent finds itself is classified to determine which of a predetermined set of actions is most likely to be effective in achieving the agent's goals. There has been some interest recently in constructing the expert decision rules automatically [Drummond, 1988], but the approach itself must inevitably be limited by the fact that the agent involved has no real idea what's going on – it is simply mindlessly applying rules that govern its behavior. The ensuing brittleness is typical of expert systems generally.

General-purpose planners, of which there are few, attempt to address this difficulty by working with a set of action descriptors that describe the possible actions in some domain, and then constructing a plan to achieve a particular goal using methods that are independent of the domain in which the agent finds itself. This work began with STRIPS [Fikes and Nilsson, 1971]; the most successful existing planner of this sort is arguably the SIPE system [Wilkins, 1988].

There are two difficulties with the general-purpose approach. The first is that the computational complexity of planning problems is typically very high, making it impractical to generate a complete plan that is guaranteed to achieve a particular goal. SIPE addresses this difficulty by restricting the form of the actions it can consider.

The nondeclarative restrictions placed on the form of the actions being considered unfortunately make these planners nonuniversal; there are domains for which any particular restriction is inappropriate. This is the essence of the second difficulty: General-purpose planners, by committing at a fundamental level to a specific description of actions, inherit some (but by no means all) of the brittleness of their expert-planning predecessors. "General-purpose" planners are only general-purpose within the bounds established by assumptions embodied in the form of the action descriptors.

First-principles planners (of which there are none) attempt to address these difficulties by viewing planning as a purely declarative activity. More specifically, first-principles planners view planning as theorem proving set against the background of a declarative system that describes actions in a particular domain.

This idea is an old one, dating back to the QA3 system [Green, 1969]; as work on declarative systems generally has advanced, the attractiveness of the approach has remained. With the development of nonmonotonic reasoning, for example, it was suggested that this general declarative notion could be applied to construct a planner that would be able to jump to conclusions while building its plans. It was later suggested that assumption-based truth maintenance [de Kleer, 1986], another general declarative technique, might bear on the problem of debugging plans that appear to be nonmonotonically sound but that closer inspection reveals to be flawed in some way [Ginsberg, 1991b].

The reason that there are no established planners of this type is that the underlying declarative descriptions of action are themselves lacking. The best-known reason for this is the infamous Yale shooting problem [Hanks and McDermott, 1987], although a variety of researchers have found solutions to this particular difficulty.

A more fundamental problem with declarative descriptions of action is that they are simply unsuitable for inclusion in planners. The approach suggested in [Green, 1969] and reiterated in [Genesereth and Nilsson, 1987] is still a valid one – given a monotonic description of a domain, it is indeed possible to view planning as theorem proving. The difficulty is that it is not practical to do so.

The reason for this can be seen by considering the frame axiom. Here is a typical rendering of it that might be included in a nonmonotonic axiomatization:

$$\mathtt{holds}(f, s) \wedge \neg \mathbf{ab}(a, f, s) \supset \mathtt{holds}(f, \mathtt{result}(a, s)) \tag{1}$$

Informally, this axiom says that if some fluent f holds in a situation s and the action a is not abnormal in that it reverses f when executed in the situation s, then f will continue to hold after the action is completed.

There are technical problems with this definition, but they can be avoided [Baker and Ginsberg, 1989, and many others]. But an overwhelming *computational* difficulty can be seen if we imagine using (1) to propagate a set of fluents through a long sequence of actions. The application of (1) for each action and to each fluent will result in a prohibitively large number of consistency checks, making the system unusable in practice.

This problem is avoided in general-purpose planning systems by using a nondeclarative description of action that has more attractive computational properties. In STRIPS, for example, actions are described in terms of add and delete lists, reducing the complexity of the reasoning enormously. The STRIPS formalism cannot deal with the inferred consequences of actions, however, as was observed in [Lifschitz, 1986]. (This is called the *ramification problem* in [Finger, 1987].) A partial solution to this difficulty can be found in [Ginsberg and Smith, 1988], but the approach presented there continues to describe actions in nondeclarative terms.

The intellectual foundation for the work described in this paper lies in an attempt to present a declarative description of the work in [Ginsberg and Smith, 1988]; we have tried to develop a formalization of action that will be computationally viable in the situations likely to arise in planning. The two specific heuristic commitments that we will make are the following:

First, we will assume that fluents typically survive long sequences of actions before being needed; a robot should be able to put a wrench in its toolbox, perform most of its day's activities, and conclude at a single stroke that the wrench is still in the toolbox. We will describe this by saying that our formalization of action needs to be *event-driven* in the sense that propagating fluent values through idle periods not incur significant computational expense.

Second, we will commit ourselves to a system that can reason about actions in an *anytime* fashion; the word appears to originate in [Dean and Boddy, 1988]. By this we mean that the system, when asked the value of a fluent in a specific situation, will produce some answer quickly, perhaps modifying that answer as necessary if allowed to consider more subtle features of the situation involved. It is generally recognized that planning problems are sufficiently difficult that approximate answers are inevitable; we are simply requiring that this sort of computational response be present in the description of action that underlies the planner itself.

The reason that we have chosen to discuss these two problems in this paper is not that there are no others (there are), but that the solutions to them are linked. Roughly speaking, both difficulties can be addressed by taking the truth value assigned to a sentence to be not a single value such as "true" or "false," but a function from a set of time points into such values.

The reason that this approach leads to an event-driven description is that it allows us to conveniently describe the expected future behavior of fluents in a compact fashion. Instead of saying, "The wrench is in the toolbox at 9:15," and, "Things in toolboxes tend to remain there," we can simply say, "The wrench is expected to be in the toolbox for the rest of the day," meaning that the truth value assigned to the sentence

$$\mathtt{in}(\mathtt{wrench}, \mathtt{toolbox})$$

is a function that maps the entire temporal interval from 9:15 to 5:00 to the value t (or perhaps dt – true by default – if we are prepared to admit the possibility of subsequent information reversing our conclusions). The problem of making our description event-driven now becomes essentially a matter of finding a data structure for the functional truth values that efficiently encodes the behavior of fluents that change only infrequently.

The idea of taking truth values to be temporal functions also bears on our requirement that the implementation of our formalism exhibit anytime behavior. As an example, consider a sentence such as, "One second after the valve is closed, the pressure will increase," which we will write somewhat schematically as

$$\texttt{delay(closed-valve,1)} \supset \texttt{pressure} \qquad (2)$$

where delay is an operator that we will use to push the temporal description of the valve into the future. The second argument to delay in (2) is a parameter indicating the amount by which the truth value is to be displaced.

From a formal point of view, the delay operator appearing in (2) is a *modal operator*, since one of its arguments is not an object in our language but a declarative sentence instead. It is shown in [Ginsberg, 1990] that when truth values are taken to be more descriptive than simply elements of the two-point set $\{t, f\}$ (true and false, respectively), it is possible to view modal operators as functions on the truth values of their propositional arguments. In (2), the modal operator delay corresponds to the *function* delay that is given by

$$[\texttt{delay}(f, \Delta)](t + \Delta) = f(t) \qquad (3)$$

Note that delay accepts a function as its first argument and returns a function as its result, since the truth values that we are using are themselves functional.

To see that this interpretation leads to anytime behavior, we need to make one more observation: The basic purpose of a deductive system is to determine what truth value should be assigned to a particular query. Now note that when considering a query q, we may well encounter a modal operator, requiring us to apply the corresponding function (as in (2) or (3)) to the truth values of the propositional arguments (closed-valve in (2)). But what are we to use for these truth values? We can use either the result of invoking the theorem prover recursively on the propositional arguments themselves, or use the values that can be obtained by simply searching for the given propositions in the database. Using these latter values as approximations to the former leads to a system that produces some answer quickly, but may modify that answer on further consideration. Perhaps there is a deductive demonstration that the valve in (2) will be open at some particular future time, and so on. If the analysis of the embedded sentences produces still further modal expressions, anytime behavior will result as the system makes and then examines assumptions about the truth values assigned to these embedded sentences. This observation is expanded upon in [Ginsberg, 1991a].

The remainder of this paper will consider each of these ideas in turn, and then show an example of an implementation of them being used to analyze a shooting scenario similar to that appearing in [Hanks and McDermott, 1987]. The implementation is built using the multivalued theorem proving system MVL [Ginsberg, 1988, Ginsberg, 1989, Ginsberg, 1991c].

2 TRUTH VALUES

We remarked in the introduction that we intend to label sentences in our declarative database with functional truth values that include information about the truth or falsity over time (or default truth/falsity, etc.) of the sentence involved. The reason that we are comfortable doing this is that the labels so constructed retain the mathematical structure of the original "instantaneous" labels, in that we can combine them, negate them, disjoin and conjoin them, and so on.

It is these operations of conjunction, disjunction and so forth that underlie the semantics of any declarative system. Specifically, if we have labels x and y for sentences p and q respectively, we need a way to construct a label $x \vee y$ for the disjunction of p and q, and so on. In [Ginsberg, 1988], a mathematical structure called a *bilattice* was introduced that consists of a set together with just these combining operations. Although the formal definition will not play a significant role in the remainder of this paper, we include it here in the interests of completeness:

Definition 2.1 *A* bilattice *is a sextuple* $(B, \wedge, \vee, \cdot, +, \neg)$ *such that:*

1. $(B, \wedge, \vee)$ *and* $(B, \cdot, +)$ *are both complete lattices, and*
2. $\neg : B \to B$ *is a mapping with* $\neg^2 = 1$ *such that* $\neg$ *is a lattice homomorphism from* $(B, \wedge, \vee)$ *to* $(B, \vee, \wedge)$ *and from* $(B, \cdot, +)$ *to itself.*

The bilattice operations $\wedge$, $\vee$ and $\neg$ all correspond to the usual logical notions, while $+$ corresponds to combination of evidence and is used to combine the truth values obtained from separate lines of reasoning to a single conclusion. Many more details of the bilattice work, together with a discussion of the philosophical ideas underlying the approach, can be found in [Ginsberg, 1988].

What is of interest to us here is the fact that if B is some bilattice, then B^2, the collection of ordered pairs of elements of B, inherits a bilattice structure from B where all of the bilattice operations are computed pointwise. (The construction is analogous to the construction of the Cartesian plane $\mathbb{R}^2$ as the product of two copies of the real line.) More generally, for any set S, the set B^S of functions from S into B inherits a bilattice structure from the set B.[1]

It follows that if we have some set T of time points, then the set B^T of functions from T into the "base" set of truth values B has the structure required of a set

[1]There is no real difference between viewing the set B^2 as the set of ordered pairs of elements of B, or as the set of functions from the two-point set $\{1, 2\}$ into B.

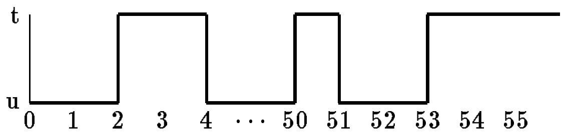


Figure 1: A fluent that is true three times

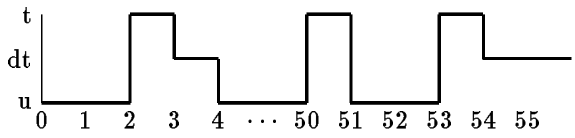


Figure 2: A default frame axiom

of truth values. As an example, if we take T to be the integers, then the graph in Figure 1 shows the truth value assigned to a fluent that is true for two units of time at $t = 2$, for one unit of time at $t = 50$, and for all time after $t = 53$.

Our event-based philosophy now corresponds simply to a data structure that represents these truth values by listing the points at which the value changes. In Figure 1, for example, we record the fact that the fluent is unknown at time 0, true at time 2, unknown at time 4, and so on; values at a total of six points are recorded. Determining the value of the fluent at any intermediate time t is a matter of walking along the graph until the next event is later than t, and taking the value at the last time encountered. Note that the computational effort required to determine the value of the fluent in Figure 1 is completely independent of the length of the gap between times 4 and 50.

2.1 EXTENSTIONS

The ontological shift that we are proposing does not in and of itself commit us to any specific computational or representational strategy. As an example, the simple representation scheme that we described in the previous paragraph can easily be extended in a variety of ways:

1. The set T of time points only needs the structure of a partial order in order for the above approach to work; to determine the value of a fluent f at some particular time t, we walk our way along the function until we find ourselves between two points t_0 and t_1 such that
$$t_0 \leq t < t_1,$$
so that t is no earlier than t_0 and t_1 is later than t. The value of f at t is then the value taken at t_0.
2. As an example, taking the above partial order to be the continuous real line allows us to avoid our earlier implicit assumption that time was discrete. This particular choice commits us to fluents being true over half-open intervals $[x, y)$ only, but this can be avoided by introducing auxiliary elements x^+ for each $x \in \mathbb{R}$ such that the half-open interval $[x, y^+)$ in fact denotes the closed interval $[x, y]$.
3. Another example involves taking the elements of the partial order to be action sequences, where an action sequence a_2 temporally follows a sequence a_1 whenever a_2 is an extension of a_1. Nonlinear action sequences can be handled by weakening the partial order to cater to possible linear action sequences consistent with a given nonlinear one. Alternatively, by using sequence variables to denote arbitrary sequences of actions following some particular one, it is possible to use the partial order given by variable instantiation to capture this temporal information. This idea appears to be central to the use of our ideas in a functioning planning system.
4. It is also possible to extend the scheme by introducing "decay functions" that describe how a fluent's truth value is expected to change as time goes by. In Figure 1, for example, the fluent's truth values do not change at all as time passes; a more realistic example might involve the truth value of the fluent falling from t to dt at times 3 and 54, as shown in Figure 2. Here, our confidence in the truth of the fluent decays as time passes, corresponding to the application of a nonmonotonic frame axiom. As before, information is recorded only when the truth value of the fluent changes from the expected one, so we still need to record information only about the "events" at times 0, 2, 4, 50, 51 and 53. By changing the set of base truth values to which the temporal functions map, this idea can be extended to include a wide variety of temporal behaviors, such as the probabilistic decay functions discussed in [Dean and Kanazawa, 1988]. A fuller discussion of this point can be found in [Ginsberg, 1991d].

The computational efficacy of the scheme that we have proposed is preserved in all of these cases.

2.2 THE FRAME AXIOM

The approach we have suggested encodes the information that is normally captured by a frame axiom such as (1) in two separate ways. First, the actual default behavior assigned to some particular fluent f is encoded in a truth value such as that appearing in Figure 2, which explicitly indicates the default truth of the fluent at times following times when it is known to be true with certainty.

We have thus far avoided the question of how we obtain truth values such as that appearing in Figure 2 in the first place. What we would like to do is to have a sentence in our database such as the following:

> If the robot puts a tool in its toolbox, then the tool is definitely in the toolbox at that time, and can be assumed by default to be in the toolbox at subsequent times.

Note first that the truth value to be assigned to the conclusion of the rule (that the tool is in the toolbox) is not the same as the truth value of the premise; the premise (that the tool is being put into the toolbox) holds only instantaneously, while the conclusion (that the tool is actually located in the toolbox) holds over a wide range of times. This is a technical issue that we will consider in the next section and is identical in principle to the appearance of the `delay` operator in the introduction.

More importantly, note that the future behavior of any particular fluent (in this case, the location of the tool) is determined not by applying some blanket frame axiom such as (1), but instead by an axiom describing this future behavior when the fluent is first asserted (i.e., when the robot puts the tool in the toolbox).

This is an important distinction between our approach and the conventional one. Computational issues aside, we prefer our approach on purely philosophical grounds, since it is *not* the case that the frame axiom applies to all, or even most of the fluents we encounter in everyday life.

As an example, consider the problem of entering a crosswalk when there is a car five feet away approaching at 60 MPH. Are we to apply the frame axiom to the fact that the car is five feet from the crosswalk, or to the fact that it is moving at a high speed? Clearly to the latter, although there is no information in (1) indicating that we should do so.

This is related to the well-known *problem of induction* [Skyrms, 1966]. How is it that we know to apply the frame axiom to a predicate such as "blue" or "green" but not to one such as "grue" (green until July 10th but blue subsequently) or "bleen?" Although we have not provided an answer to this question, we have indicated clearly the declarative point at which such an answer is used – in the description of the expected future behavior of newly established fluents. Similar observations have also been made in [Myers and Smith, 1988].[2]

[2][Haas, 1987] and [Schubert, 1990] have also argued against blanket frame axioms, suggesting instead that one should reason from specific fluent changes to the occurrence of an action of some sort. It seems to me, however, that this approach is unable to cope with either the ramification problem or with the moving automobile described earlier in this section.

3 MODAL EXPRESSIONS

Let us return to the observation made in the last section that the truth value to be assigned to the consequent of some rule is often not the same as the truth value of the antecedent. In the introduction, we handled a situation such as this in (2) by introducing a modal operator m and writing

$$m(a) \supset c$$

where a is the antecedent and c is the consequent. The modal operator m changes the truth value of a so that the truth value of c is modified correctly by the above rule. The example in the introduction involved a modal operator `delay` that delays the truth value of the antecedent by some amount of time; our current concerns can be dealt with by introducing an operator `propagate` that is responsible for inserting the consequent into the database with a complete "future history" if appropriate.

We will discuss these two operators in some detail shortly, but let us continue to examine general issues first. The idea that modal operators can be viewed truth-functionally (i.e., as functions on the truth values of the sentences on which they operate) is an old one in the philosophical community, but was discarded in favor of the possible-worlds approach [Kripke, 1971] when it was realized that there simply are not enough functions on the two-point set $\{t, f\}$ to correspond to all of the interesting modal expressions that one might wish to consider.

In [Ginsberg, 1990], however, it was pointed out that if truth values are taken from an arbitrary bilattice instead of from the set $\{t, f\}$, it becomes practical to view modal operators truth-functionally after all; in fact, the resulting construction is a generalization of Kripke's.

3.1 ANYTIME BEHAVIOR

An additional advantage of viewing modal operators truth-functionally is that the associated declarative systems naturally exhibit anytime properties; this can probably be made clearest by an example from PROLOG. Consider the following program:

```
landlubber(X) :- animal(X), not(fly(X)).
fly(X) :- bird(X), not(penguin(X)).
animal(X) :- bird(X).
penguin(X) :- bird(X), tuxedo(X).
bird(opus).
tuxedo(opus).
```

Animals that cannot fly are landlubbers, birds can fly unless they are penguins, and birds in tuxedos are penguins. Opus is a bird wearing a tuxedo. Is he a landlubber?

Ignoring inadequacies in our representation of the domain, the interpreter begins by noting that it can prove that Opus is an animal, and therefore that he is a landlubber unless he can be shown to fly. A new proof process is therefore begun with the intention being to prove that Opus can fly.

Since Opus is a bird, he can fly unless he can be shown to be a penguin. Yet another proof process is begun; since this one succeeds in showing Opus to be a penguin, he cannot be shown to fly and the original query (is Opus a landlubber?) succeeds.

[Ginsberg, 1990, Ginsberg, 1991a] propose that it be possible to interrupt this procedure at the points where new proof attempts are generated. Thus when creating the attempt to prove that Opus can fly, we note that since there is nothing in the database indicating explicitly that he can, we can tentatively label

`fly(opus)`

as unknown, and therefore assign `not(fly(opus))` the value of true using PROLOG's negation-as-failure rule. This allows us to tentatively confirm the original query.

Given more time, we can work on the goal `fly(opus)`, noting that this spawns the subgoal `penguin(opus)`. Once again, we break the inference process, using the fact that `penguin(opus)` is missing from our database to conclude tentatively that `not(penguin(opus))` is true and therefore that Opus can fly, so that the original query should fail. Finally, given still more time for reflection, we realize that Opus is a penguin and therefore a landlubber after all.

PROLOG's treatment of negation is as an operator that returns t unless the truth value of the argument is itself t; specifically, if some sentence p is unknown, negation-as-failure treats $\texttt{not}(p)$ as true. Viewed in this fashion, PROLOG's negation is a modal operator in our sense. We are proposing two extensions to this idea:

1. Extending the notion of a modal operator to include temporal operators such as those that arise when reasoning about action.
2. Using these modal operators as semantic markers for points at which the inference process can be suspended and an approximate answer computed.

The first of these is the focus of the current paper; the second is treated implicitly in [Ginsberg, 1990] and in more detail in [Ginsberg, 1991a].

3.2 TEMPORAL OPERATORS

Given that we take the view that temporal operators can be described by giving their functional behavior and then incorporating them into our declarative language, what operators are required in a system that reasons about action?

We will clearly need an operator `delay` to separate the occurrence of an action from the appearance of its effects, and another operator `propagate` that allows us to construct temporal truth functions such as the one appearing in Figure 2.

In fact, we appear to need nothing else; the axiomatization appearing in the appendix uses no modal operators other than these two. The description of `delay` is as appearing in (3), while `propagate` is defined recursively as[3]

$$[\texttt{propagate}(f)](t) = \begin{cases} f(t), & \text{if } t = 0 \text{ or } f(t) \neq u; \\ \texttt{decay}[\texttt{propagate}(f)(t-1)], & \text{otherwise.} \end{cases}$$

The function `decay` might be given by, for example:

x	$\texttt{decay}(x)$
t	dt
dt	dt
f	df
df	df
u	u

This decay function maps any truth value into its default version, corresponding to a nonmonotonic frame axiom. A monotonic frame axiom would simply take $\texttt{decay}(x) = x$. As an example, Figure 3 shows the result of applying `propagate` to a temporal function that changes from t to f. Figure 4 shows the result if the frame operator is chosen to be monotonic.

From these two operators, we can build a declarative description of action that has the desired properties of being event-driven and anytime. If a is an action that causes a fluent f to be true in a persistent way (such as putting a tool in the toolbox), we write[4]

$$\texttt{duration}(a,t) \wedge \texttt{propagate}(\texttt{delay}(a,t)) \supset f \quad (4)$$

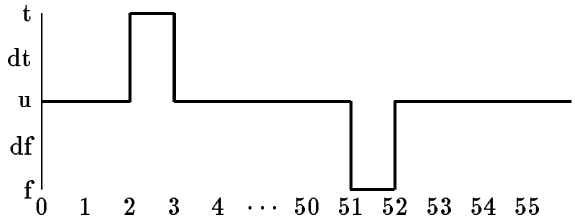
while if a causes f to be true only instantaneously (like striking a match causing a light), we write

$$\texttt{duration}(a,t) \wedge \texttt{delay}(a,t) \supset f$$

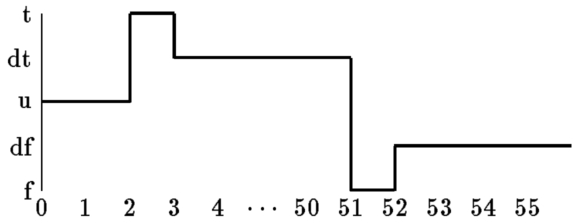
[3]The implementation of `propagate` does not follow this definition directly, since this would be horrendously inefficient. Instead, we use a monotonic propagation function (as described in the appendix), and simply drop information about any "events" at which a truth function becomes unknown.

[4]The axiom (4) is not quite satisfactory as it stands, because it is awkward to combine it with domain constraints describing ramifications of the action in question. This is handled in the appendix by reifying the fluents so that they can be treated as objects in our language, and replacing (4) with an axiom like

$$\texttt{causes-persistently}(a,f) \wedge \texttt{duration}(a,t) \wedge \texttt{propagate}(\texttt{delay}(\texttt{holds}(a),t)) \supset \texttt{holds}(f)$$

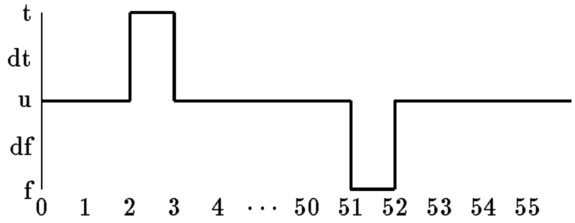


A temporal function f

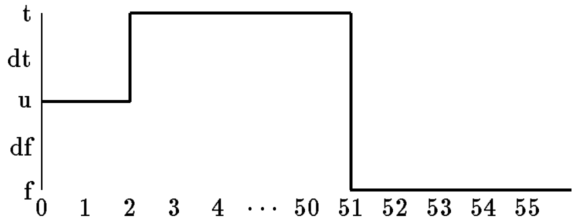


The function `propagate`(f)

Figure 3: Applying the frame axiom



f



`propagate`(f)

Figure 4: A monotonic frame axiom

The predicate `duration` indicates the amount of time taken by any particular action and is used to determine the value to be assigned to the parametric argument of `delay`.

3.3 EFFICIENCY CONSIDERATIONS

Suppose that we consider the example in Figures 3 and 4 a bit more closely. In the example in Figure 4, for example, note that there is no point to attempting to show that the fluent holds at time 4 – we already know this by virtue of the application of our monotonic propagation function. This implies the following:

> *When invoking the prover on a sentence p in order to evaluate a modal expression $m(p)$, one should only investigate proofs that will affect not only the truth value of p, but the truth value of $m(p)$ as well.*

In fact, the situation is a bit more subtle still; consider Figure 3.

Suppose that we are interested in showing that the fluent f holds at time 53, perhaps because f is a precondition to an action that we would like to take at that time, or perhaps because the restriction mentioned above implies that this is the only information about f that is of interest to us.

In the initial situation in which we know nothing, it follows that we should try to show that f holds at any time before $t = 53$, since this value will then be propagated to the time of interest. Although showing that f holds at time 54 does effect the value of `propagate`(f), it does not do so in an interesting way. This means that we should replace the above principle with the following stronger one:

> *When invoking the prover on a sentence p in order to evaluate a modal expression $m(p)$, one should only investigate proofs that will change the truth value of $m(p)$ in a way that will affect the response to the original query.*

Applying this idea can be fairly subtle. In the example we are considering, suppose that we succeed in showing that f holds at time 2, so that `propagate`(f) holds by default at time 53. Now there is no point in showing that f holds, but there *is* a reason to show that the negation of f holds at some time between 3 and 53, since $\neg f$ will block the propagation of f to the time that is of interest to us.

In the example in the figure, we can show that $\neg f$ holds at time 51; now we must once again change the focus of our proof efforts as we attempt to show that f is true either at $t = 52$ or at $t = 53$.

From a conceptual point of view, this is all quite straightforward. From an implementational point of view, however, it can be rather subtle, especially since we should preserve portions of the proof tree for a fluent f even if they appear not to be relevant to propagate(f). The reason for this is that subsequent developments may change this. In the example we have been considering, perhaps proving f has been reduced (after considerable effort) to proving g and h; when our focus changes to that of proving $\neg f$, we should retain this information in case (as happens in this example) we decide that we need to prove f after all.

4 AN EXAMPLE

The ideas that we have described have been implemented using the multivalued theorem prover described in [Ginsberg, 1989, Ginsberg, 1991c], which allows the user to select truth values from arbitrary bilattices and to include arbitrary modal operators in a declarative database. The precise axiomatization of actions generally and our domain in particular can be found in the appendix.

The domain we are considering involves a gun, which may or may not be loaded, and a victim (Fred), who may or may not be alive. At time 0, the gun is loaded and Fred is alive.

This domain has three actions: loading and unloading the gun (which take a single unit of time and always succeed), and shooting the gun at Fred. If the gun is loaded, firing it at Fred will kill him after a delay of two units of time.[5] All of the fluents persist in a nonmonotonic fashion except that once Fred dies, he is guaranteed to stay dead.

The course of events in this domain is as follows:

Time	Event
0	Fred is alive and the gun is loaded
1	The gun is unloaded
2	The shooting is attempted and the gun is reloaded
50	The shooting is attempted
52	The gun is reloaded
53	The shooting is attempted

The first shooting action should fail, since the gun has been unloaded at the previous time point. The second shooting action should succeed by default, since the gun has presumably remained loaded between times 3 and 50. The third shooting action will definitely succeed, since it immediately follows a `load` action.

Given this information, the system was asked to investigate the truth or falsity of the fluent `alive` (is Fred alive?) at all times; the results are shown in Figures 5 and 6.

The machine reasoned as follows:

a. Initially, it had no explicit information concerning the state of Fred's health.
b. The machine first realized that the shooting at time 50 was expected to succeed, so that at times 52 and subsequently, Fred was expected to be dead.
c. The machine also decided that the shooting at time 2 was likely to succeed, since the gun was loaded in the initial situation and that fact was expected to persist.
d. The machine next investigated the modal expression propagate(loaded) a bit more closely, since this modal expression was used in its conclusion that the shootings at times 2 and 50 succeed. It discovered that the gun is not expected to be loaded at time 2 after all, and therefore that the first shooting should fail.
e. The machine next realized that the shooting at time 53 was *guaranteed* to succeed, so that at times 55 and subsequently, Fred would certainly be dead.
f. The previous proof attempts involved attempts to find times at which Fred was expected to be dead; the machine now changed its focus to trying to prove that he was alive, and concluded that he was (by default) from time 0 until the success of the shooting at time 50.
g. Finally, the program realized that Fred was guaranteed to be alive in the initial situation, and updated its conclusions to reflect this.

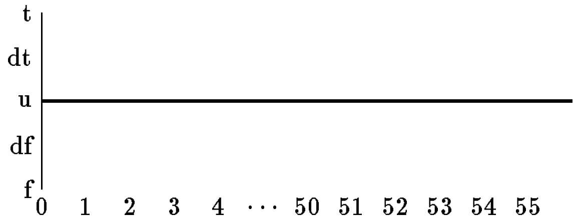
In this example, we can clearly see the two features that have been the focus of this paper – the event-driven nature of the description, evidenced by the lack of computational effort devoted to the "idle" time from $t = 4$ to $t = 50$, and the anytime nature of the analysis, shown in the shifting conclusions displayed in Figures 5 and 6.

5 CONCLUSION

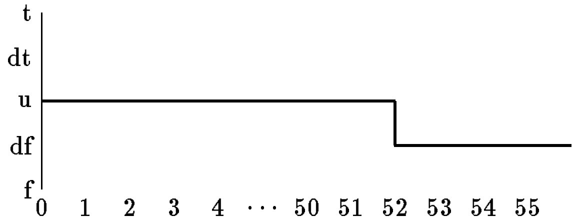
My intention in this paper has been to argue for two things: First, the use of truth values that directly capture the complete history of fluents that change over time; second, the manipulation of these truth values using truth-functional modal operators. We have presented an implementation of our ideas that correctly analyzes a simple example similar to the shooting scenario presented in [Hanks and McDermott, 1987], but the theoretical justifications for this approach are more compelling:

1. The approach embeds the action descriptions in a full declarative language, and investigates the consequences of actions by proving theorems against

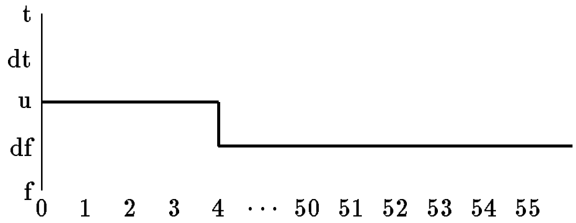
[5] This is the only consequence of the shooting action. Specifically, shooting does not cause the gun to become unloaded.



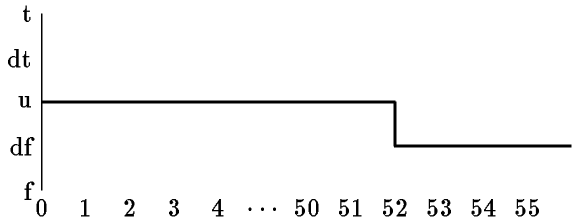
(a) Initial knowledge



(b) The second shooting succeeds by default

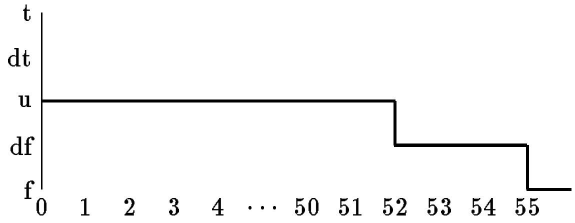


(c) The first shooting also succeeds by default

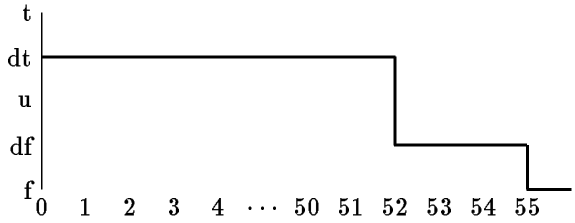


(d) Unload at $t = 1$ causes first shooting to fail

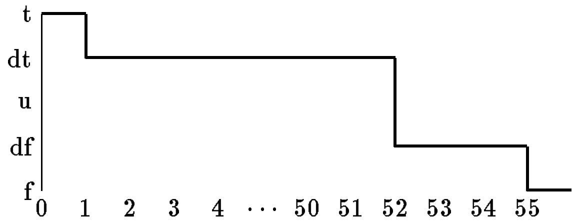
Figure 5: The shooting scenario



(e) The third shooting succeeds



(f) Fred is alive by default initially



(g) In the initial situation, Fred is alive for sure

Figure 6: The shooting scenario (ctd.)

this background. A system developed in this way will benefit from developments elsewhere in the theorem-proving community in a way that a more *ad hoc* approach cannot.

2. The event-driven nature of the approach allows us to reason in a computationally viable way about fluents that change value only infrequently. As discussed in Section 2.1, we do this without committing ourselves to any specific ontology of time or of action.
3. The natural implementation of our ideas exhibits an anytime behavior that we can expect to be present in the associated planning system as well. Furthermore, this ability to incrementally refine our conclusions is grounded in a solid formal foundation.
4. Finally, the approach we have described avoids the use for a blanket frame axiom such as (1), which is likely to fall prey to the problem of induction. Although we have presented no solution to this difficulty, our approach makes clear that such a solution will need be reflected in the declarative description of our domain, since the expected future history of any particular fluent needs to be asserted when the fluent itself is added to the database.

APPENDIX: AXIOMS USED IN SECTION 4

The axioms used in our description of action involve four separate causal predicates, as follows:

1. causes(a, f) means that the action a causes the fluent f to be true instantaneously.
2. causes-persistently(a, f) means that the action causes the fluent to be true in a way that is expected to persist into the future.
3. causes-forever(a, f) means that the action causes the fluent to be true in a way that is *guaranteed* to be true in the future.
4. Finally, causes-not(f_1, f_2) means that fluent f_1 implies the negation of fluent f_2. The fluents alive and dead are related in this fashion.

Fluents are reified using a holds predicate; the insertion of the reified fluents into the database is also reified using a triggers predicate, so that holds is the result of applying the modal operator propagate to triggers.

Here are the axioms associated with these predicates, expressed in a PROLOG-like style:

```
holds(F) :- causes(A,F), duration(A,T),
            delay(succeeds(A),T).
causes(A,F) :- causes-persistently(A,F).
holds(F) :- propagate(triggers(F)).
triggers(F) :- causes-persistently(A,F),
               duration(A,T),
               delay(succeeds(a),T),
               default.
triggers(F) :- causes-forever(A,F),
               duration(A,T)
               delay(succeeds(a),T).
not(holds(P)) :- causes-not(Q,P), holds(Q).
not(triggers(P)) :- causes-not(Q,P),
                    triggers(Q).
```

There are a couple of things to note here:

1. The MVL system has a true negation operator in addition to the modal negation-as-failure operator used in PROLOG. The not appearing in the heads of the last two of the above rules is true negation.
2. The sentence default is inserted into the database with a truth value indicating that it has value dt at all times. This allows us to use a monotonic propagate operator, with the inclusion of the default sentence serving to distinguish causes-persistently from causes-forever.

We also need axioms expressing conditions under which an action succeeds. We assume that there is a predicate prec(a, p) that holds whenever p is a precondition of the action a.

```
succeeds(A) :-
 occurs(A),
 forall(P,(holds(P) :- L (prec(A,P)))).
```

Note the appearance of the quantifier forall in the above rule, together with the modal operator of knowledge L. This rule says that the action A succeeds if it occurs and if every known precondition of A holds. The MVL system includes a modal operator of knowledge L and true quantifiers.

Finally, there is a dummy action init that takes place at time 0 and is used to construct the initial situation. This action has no preconditions and takes no time.

```
occurs(init).        ;true at time 0 only
duration(init,0).
```

To describe the shooting domain specifically, we first describe the initial situation as a consequence of the init action:

```
causes-persistently(init,alive).
causes-persistently(init,loaded).
```

These axioms say that Fred is alive (and expected to remain so) and that the gun is loaded (and also expected to remain so) in the initial situation.

We also need axioms describing the various actions. Here are load and unload:

```
causes-persistently(load,loaded).
causes-persistently(unload,unloaded).
duration(load,1).
duration(unload,1).
```

The fluents `load` and `unload` are negations of one another:

```
causes-not(loaded,unloaded).
causes-not(unloaded,loaded).
```

Shooting is similar. It causes Fred to be dead forever, has a precondition of the gun being loaded, and needs two units of time to take effect. The fluents `alive` and `dead` are negations of one another:

```
causes-forever(shoot,dead).
prec(shoot,loaded).
duration(shoot,2).
causes-not(alive,dead).
causes-not(dead,alive).
```

Finally, we need axioms saying what occurs when:

```
occurs(unload).   ;true at time 1
occurs(shoot).    ;true at times 2, 50
                  ;and 53
occurs(load).     ;true at times 2 and 52
```

When we indicate "true at time t" above, we mean that these facts are inserted into the database with truth values indicating that they are true at these times; there is no way to represent this using conventional PROLOG syntax. The three occurrences of the shooting action are combined to get a truth function similar to that shown in Figure 1 except for the fact that the occurrences are all of unit time duration.

Acknowledgement

This work has been supported by the Air Force Office of Scientific Research under grant number 90-0363 and by NSF under grant number IRI89-12188. I would like to thank Adnan Darwiche, Don Geddis, Drew McDermott and David Smith for various helpful discussions.

References

[Baker and Ginsberg, 1989] Andrew B. Baker and Matthew L. Ginsberg. Temporal projection and explanation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 906–911, 1989.

[de Kleer, 1986] Johan de Kleer. An assumption-based truth maintenance system. *Artificial Intelligence*, 28:127–162, 1986.

[Dean and Boddy, 1988] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, 1988.

[Dean and Kanazawa, 1988] Thomas Dean and Keiji Kanazawa. Probabilistic temporal reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 524–528, 1988.

[Drummond, 1988] Mark Drummond. Situated control rules. Technical report, NASA Ames Research Center, Moffett Field, CA, 1988.

[Fikes and Nilsson, 1971] R.E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Finger, 1987] Jeffrey J. Finger. *Exploiting Constraints in Design Synthesis.* PhD thesis, Stanford University, Stanford, CA, 1987.

[Genesereth and Nilsson, 1987] Michael R. Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, 1987.

[Ginsberg and Smith, 1988] Matthew L. Ginsberg and David E. Smith. Reasoning about action I: A possible worlds approach. *Artificial Intelligence*, 35:165–195, 1988.

[Ginsberg, 1988] Matthew L. Ginsberg. Multivalued logics: A uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

[Ginsberg, 1989] Matthew L. Ginsberg. User's guide to the MVL system. Technical report, Stanford University, 1989.

[Ginsberg, 1990] Matthew L. Ginsberg. Bilattices and modal operators. *Journal of Logic and Computation*, 1:41–69, 1990.

[Ginsberg, 1991a] Matthew L. Ginsberg. Anytime declarativism. Technical report, Stanford University, 1991.

[Ginsberg, 1991b] Matthew L. Ginsberg. The computational value of nonmonotonic reasoning. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, Boston, MA, 1991.

[Ginsberg, 1991c] Matthew L. Ginsberg. The MVL theorem proving system. *SIGART Bulletin*, 2(3):57–60, 1991.

[Ginsberg, 1991d] Matthew L. Ginsberg. What is the modal truth criterion? 1991.

[Green, 1969] C. C. Green. Theorem proving by resolution as a basis for question-answering systems. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence 4*, pages 183–205. American Elsevier, New York, 1969.

[Haas, 1987] A.R. Haas. The case for domain-specific frame axioms. In Frank M. Brown, editor, *The Frame Problem in Artificial Intelligence.* Morgan Kaufmann, San Mateo, CA, 1987.

[Hanks and McDermott, 1987] Steve Hanks and Drew McDermott. Nonmonotonic logics and temporal projection. *Artificial Intelligence*, 33:379–412, 1987.

[Kripke, 1971] Saul A. Kripke. Semantical considerations on modal logic. In L. Linsky, editor, *Reference and Modality*, pages 63–72. Oxford University Press, London, 1971.

[Lifschitz, 1986] Vladimir Lifschitz. On the semantics of STRIPS. In *Proceedings of the 1986 Workshop on Planning and Reasoning about Action*, Timberline, Oregon, 1986.

[Myers and Smith, 1988] Karen L. Myers and David E. Smith. On the persistence of derived beliefs. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.

[Schubert, 1990] Lenhart K. Schubert. Monotonic solution of the frame problem in the situation calculus. In Henry E. Kyburg, Jr., Ronald P. Loui, and Greg N. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer, Boston, 1990.

[Skyrms, 1966] Brian Skyrms. *Choice and Chance: An Introduction to Inductive Logic*. Dickerson, 1966.

[Wilkins, 1988] David E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann, San Mateo, CA, 1988.