Modality and Interrupts

Matthew L. Ginsberg

CIRL
1269 University of Oregon
Eugene, OR 97403
ginsberg@cs.uoregon.edu

Modality and Interrupts

Abstract

We suggest that modal operators, in addition to their well-understood semantic role in declarative systems, also mark points at which these systems can be interrupted. We use this idea to describe an interruptible declarative system that gradually refines its responses to queries. Although initial responses may be in error, a correct answer will be provided if arbitrarily large computational resources are available. The ideas presented generalize existing work on stratification of logic programs and the treatment of floundered subgoals.

1 Introduction

My aim in this paper is to address difficulties that arise if declarative methods are to be used in real-time or other systems that provide only limited computational resources. Conventional declarative systems are inappropriate for use in these applications because of the unpredictable amount of time needed for them to respond to queries. The two solutions that have been proposed are to abandon declarativism (as suggested in the work on situated automata theory and its variants [1, 8, 25, 33, 39, 40]) or to restrict the declarative language in a way that guarantees that inference is tractable [12, 42, and many others].

There are problems with both of these approaches. The arguments in favor of declarativism are strong enough that abandoning the methodology should surely be a last resort; the restricted languages considered by other authors are often too inexpressive to be applied to problems of practical interest.

Instead of abandoning the declarative methodology, we will modify it so that declarative systems can be applied to real-time problems; somewhat more precisely, we will present a declarative approach to inference that is what Dean and Boddy have called an *anytime* algorithm. The associated computation can be terminated at any time and an answer returned, and the quality of these answers improves in some well-behaved manner as a function of time (paraphrased from [7, page 52]). No declarative system has these properties: Interrupt a first-order theorem prover and the most useful information you are likely to recover is the current fringe of the proof tree.

I argue that modality provides the answer. Notions of groundedness that have arisen in work on nonmonotonic reasoning [26, 27, 31] suggest that one of the purposes of nonmonotonic modal operators in declarative systems is to insulate their arguments from other information in the database, so that the truth or falsity of a modal expression can be determined solely by an analysis of the sentential arguments it contains. We will develop from this the view that modal operators are markers for points at which inference can be suspended and partial answers returned.

As an example, consider the following story:

> Fred fell out of an airplane over a farmer's field. The good news is that he was wearing a parachute. The bad news is that the parachute didn't open. The good news is that there was a haystack in the field. The bad news is that there was a pitchfork in the haystack. The good news is that Fred missed the pitchfork. The bad news is that he also missed the haystack.

As we read this, we flip back and forth between conclusions about Fred's eventual well-being. We begin by concluding that he will be killed, using a nonmonotonic rule that people who fall out of airplanes typically die. The presence of the parachute gives us additional information that we incorporate to conclude that the poor fellow is probably OK after all; then we learn that the parachute didn't open, and so on.

In this particular case, the alternating conclusions reflect changes in our information about the situation. But there are examples of similar behavior where we simply haven't taken the time to analyze the information

```
dead(X) :- falling-onto(X,Y), not(good-parachute(X)), not(soft(Y)).
good-parachute(X) :- wearing-parachute(X,P), not(broken(P)).
soft(Y) :- haystack(Y,H), not(ouch(H)), not(missed(H)).
ouch(H) :- pitchfork(H,P), not(missed(P)).

falling-onto(fred,field).
wearing-parachute(fred,parachute).
broken(parachute).
haystack(field,haystack).
pitchfork(haystack,pitchfork).
missed(pitchfork).
missed(haystack).
```

Figure 1: Fred's voyage

that we do have. If you've ever walked away from your car and left the lights on, you know what I'm referring to.

If we were to write a PROLOG program for leaving a car, part of it might look like this:

```
ok-to-leave(Car) :- keys(Car,K), in-pocket(K), lights-out(Car).
lights-out(Car) :- daytime, not(abnormal).
abnormal :- recent-rainstorm.
abnormal :- drove-through-tunnel.
lights-out(Car) :- observed-lights-out(Car).
```

It's OK to leave the car if we've got the keys in our pocket and the lights are out; the lights are typically out if it's daytime. But not if there was a recent rainstorm or we just drove through a tunnel; in these cases we actually have to check that the lights are out.

The problem is that it is all too easy to be sloppy about assigning a value to `abnormal`; we know from long experience that things are usually normal, so we are tempted to assume that `not(abnormal)` is true and we wind up returning to a car with a dead battery.

From a formal point of view, the example about the parachutist is no different; an axiomatization might look like the one in Figure 1. Writing it this way doesn't do much for the joke, but makes clear the parallels between the two examples: If we are sloppy in our analysis of a clause such as `not(missed(haystack))`, we will conclude that this clause is true, and therefore that `soft(field)` is true and Fred survives the fall. Sloppiness regarding `not(soft(field))` will lead us to conclude correctly that Fred dies, although for the wrong reason. PROLOG's negation-as-failure operator serves to mark those points where we might be sloppy in this way; put more positively, the operator marks points at which the inference process can be suspended and a partial answer returned. More thought might alter the computed conclusion, but we may be unable or unwilling to devote the computational resources required.

We can think of this in terms of a recursive descent through a dependency graph corresponding to PROLOG's reasoning. Since we know that Fred is falling toward the field, we can decide if he will die by determining whether he's wearing a functioning parachute and whether the field is soft. A recursive analysis of `good-parachute`(`Fred`) leads us to the goal of `broken`(`parachute`); the recursion terminates here because `broken`(`parachute`) appears explicitly in the database. A full diagram of the recursive tree to which this process leads appears in Figure 2.

We will generalize this recursive descent model in two ways:

1. We will change the interpretation of the negation in Figure 1 from "negation as failure to prove" to "negation as failure to prove *so far*." By doing this, we will become able to respond to queries even if the subgoals they generate have not yet been analyzed completely.

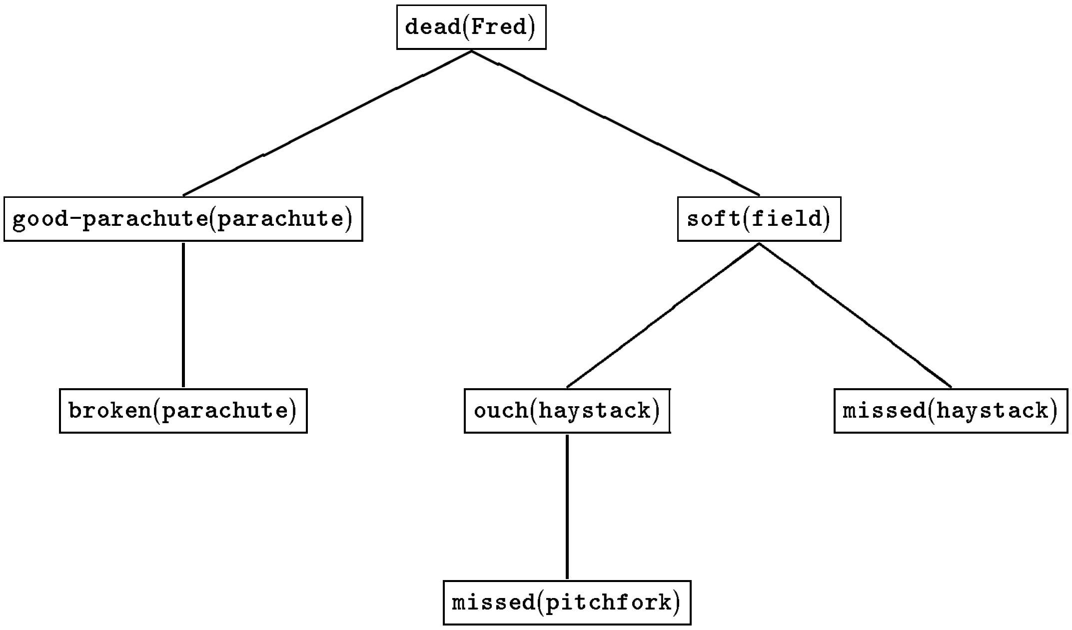

Figure 2: Modality as recursive descent

2. We need not restrict these ideas to negation as failure; any other operator that takes one or more sentential arguments can be treated similarly, provided that we can assign a value to the operator both before and after the investigation of the associated subgoals. A truth-functional semantics for modality such as that based on bilattices [17] will allow us to do this.

As an example, the translation mechanism that we will introduce in Section 2.2 will replace the first PROLOG axiom in Figure 1 with

$$\texttt{falling-onto}(x, y) \wedge \neg\Box\texttt{good-parachute}(x) \wedge \neg\Box\texttt{soft}(y) \supset \texttt{dead}(x)$$

where $\Box$ is a modal operator of necessity, serving to indicate that we are able to prove the argument from other information in the database. In this purely declarative formulation, $\Box$ continues to mark points at which new inference processes are spawned; these new processes analyze the argument to any particular appearance of $\Box$ itself. In terms of the recursive descent description, the modal operators mark points at which new nodes are formed. In terms of anytime reasoning, they mark points at which inference can be interrupted and an approximate answer computed.

The remainder of this paper makes these ideas precise. An outline is as follows:

1. In Section 2, we begin by describing the existing results underlying this work. We present a semantics for modal operators that allows us to evaluate them both before and after investigating their sentential arguments; the essential idea involves viewing the modalities truth-functionally [17].

2. In Sections 3 and 4, we use these results to describe an anytime procedure for computing the truth value assigned to a query. Approximate answers are computed quickly and the sequence of such answers converges to the correct one in the large-runtime limit.

3. In Section 5, we show that the resulting procedure generalizes existing work on stratification of logic programs [2, 14, 35, 44, 45], negative subgoals containing unbound variables [3, 4, 20], reasoning about action [18] and some informal remarks of Perlis' [34].

In the interests of maintaining the continuity of the exposition, proofs have been deferred to the appendix.

2 Preliminaries and previous work

2.1 Bilattices

We remarked in the introduction that our ideas depend on an ability to evaluate modal expressions both before and after considering their sentential arguments; this requirement is not supported by views of modality based on Kripke's possible world semantics [28]. Instead, we will use a *truth-functional* semantics for modality; the value assigned to a modal expression $m(p_1, \ldots, p_n)$ will be computed in terms of the values assigned to the sentential arguments p_1 through p_n. We can assign a preliminary value to the p_i by testing for their explicit appearance in the database; if a more complete analysis leads to a different value for some particular p_j, the truth-functional semantics assigned to m will lead to us to change the value of $m(p_1, \ldots, p_n)$.

As an example of a modal operator that admits a natural truth-functional semantics, consider the modal operator of necessity $\Box$. A natural interpretation for $\Box$ is given by

$$\Box(x) = \begin{cases} t, & \text{if } x = t; \\ f, & \text{if } x = f \text{ or } x = u \end{cases} \tag{1}$$

so that $\Box p$ is true if p itself is assigned the truth value true, and $\Box p$ is false if p is either false or unknown (i.e., those situations in which p is not necessarily true).

Note that the description given by (1) makes explicit use of the label u for a sentence that is not necessarily either true or false. This is a general property of our approach; if modal operators are to be assigned a truth-functional semantics, the truth values assigned to their sentential arguments must be taken from a set larger than the simple two-point set $\{t, f\}$.

Taking the values assigned to sentences from a set known as a *bilattice* [15] leads to a truth-functional description of modality that generalizes the conventional one [17]. This is the approach we take in this paper: Truth values are elements of a bilattice, and modal operators are viewed truth-functionally. Since the technical results we will present depend critically on the details of the bilattice interpretation of modality, we begin with a brief review of bilattice theory, summarizing material that has appeared elsewhere [15, 17]. Bear in mind that our interest in bilattices is only in that they provide a convenient formal framework in which modality can be viewed truth-functionally; results similar to ours could be developed in any other formal system that also treated modality truth-functionally.

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

Informally, a bilattice is a set of truth values equipped with two partial orders $\leq_t$ and $\leq_k$, corresponding to the partial orders associated with the lattices $(B, \wedge, \vee)$ and $(B, \cdot, +)$ respectively. The $\leq_t$ partial order indicates how true or false a particular value is, with f being the minimal element of this partial order (least true) and t the maximal element (most true). The $\leq_k$ partial order reflects how much is known about a particular sentence; the minimal element here is u (completely unknown) and the maximal element is $\perp$ (representing a contradictory state of knowledge where a sentence is known to be both true *and* false).[1]

The least upper bound and greatest lower bound operators for the $\leq_t$ partial order are $\vee$ and $\wedge$ respectively, and correspond to the usual logical notions. The operators for the $\leq_k$ partial order are denoted by $+$ and $\cdot$. The operator $+$ corresponds to the combination of evidence from separate lines of reasoning to a

[1] Other authors have used $\top$ for this contradictory truth value, since it is the maximal element under the k partial order [13]. We use $\perp$ in part to conform to AI usage of $\perp$ for a contradictory sentence, and in part to avoid any typographical confusion with T.

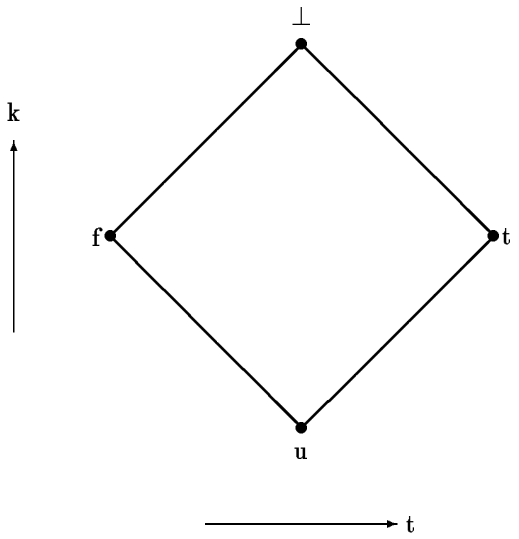

Figure 3: F, the smallest nontrivial bilattice

single conclusion, while $\cdot$ is a "consensus" operator that indicates the strongest conclusion that can still be strengthened further to each of a variety of values.

Finally, every bilattice is equipped with a negation operator $\neg$ that inverts the sense of the $\leq_t$ partial order (corresponding to de Morgan's laws) while leaving the $\leq_k$ partial order unchanged (since if you know nothing about a sentence p, you also know nothing about its negation $\neg p$). This corresponds to condition (2b) of the formal definition, where $\neg$ is required to reverse the order of $\wedge$ and $\vee$ while retaining the order of $\cdot$ and $+$.

The smallest bilattice is the trivial one, consisting of a single point and for which all of the bilattice operations map that point to itself. The smallest nontrivial bilattice appears in Figure 3 and corresponds to first-order logic. This bilattice F consists of only the four truth values $\{t, f, u, \perp\}$; as mentioned earlier, $\perp = t + f$ is used to label a sentence about which there is contradictory information. The two partial orders are as labelled in the figure. Although $t >_t f$, they are incomparable with respect to the $\geq_k$ partial order, since neither represents a more informed state than the other.

Larger bilattices contain other truth values; an example is shown in Figure 4. This is the bilattice corresponding to default logic and includes the values of dt (true by default), df (false by default) and $*$ (both true and false by default, as in the Nixon diamond [38, and later in this section]). The fact that $f >_k dt$ in this bilattice reflects the fact that additional knowledge can cause a change of heart about a default conclusion.

Within this framework, we now make the following definition:

Definition 2.2 *A* truth assignment *is a function $\phi : L \to B$, where L is our declarative language and B is a bilattice.*

A truth assignment assigns a truth value (possibly unknown) to each sentence in our declarative language. We will often refer to the sentences labelled with truth values other than unknown as the *database*.

Note that truth assignments assign truth values to all of the sentences in our language, not just the atomic ones. Thus we are free in principal to say that p and $\neg p$ are both true; although the consequences of such a truth assignment are contradictory, it is not disallowed by Definition 2.2 itself.

Definition 2.3 *For truth assignments ϕ and ψ, we will write $\psi \geq_k \phi$ to mean that $\psi(p) \geq_k \phi(p)$ for every sentence $p \in L$ and $\psi \geq_t \phi$ to mean that $\psi(p) \geq_t \phi(p)$ for every sentence $p \in L$. If $\psi \geq_k \phi$, we will say that ψ is an* extension *of ϕ; if $\psi \geq_t \phi$ as well, we will say that ψ is a* positive extension *of ϕ.*

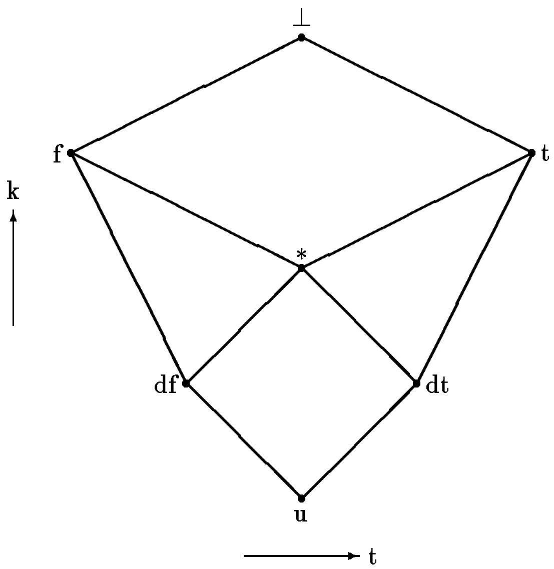

Figure 4: D, the bilattice corresponding to default logic

If ψ is an extension of ϕ, it means that the state of belief corresponding to ψ is uniformly "more informed" than the state of belief corresponding to ϕ. If ψ is a *positive* extension of ϕ, it means that all of the sentences about which additional information is available are more true under ψ than under ϕ.

The semantics of a bilattice system are given by a definition of *closure*. If ϕ is a truth assignment labelling each sentence p with explicit information about p, the closure of ϕ, $\mathrm{cl}(\phi)$, should be a truth assignment that labels p with the conclusions that this information entails. As an example, if ϕ labels both q and $q \supset r$ as true, then $\mathrm{cl}(\phi)$ should also label r as true since this is implied by the information in ϕ itself.

The formal definition of closure in this setting is given elsewhere [15]; our aim here is to give a vaguely constructive definition that appears in the earlier work only as a theorem. To understand the result we are about to present, suppose that $q \in L$ is some sentence in our language. What truth value should be assigned to q by $\mathrm{cl}(\phi)$?

If $S \subset L$ is a set of sentences entailing q, then the truth value to be assigned to the conjunction of the elements of S is

$$\bigwedge_{p \in S} \mathrm{cl}(\phi)(p)$$

If we denote this value by s, there should be a contribution to the truth value of q given by $u \vee s$. The reason that we disjoin with u can be seen if the conjunction is false, so that $s = f$. There should be no impact on the truth value of q in this case, since we are simply unable to draw useful conclusions from the implication $S \supset q$. The disjunction of s with u achieves just this effect. The reason we say that $u \vee s$ is only a *contribution* to the truth value of $\mathrm{cl}(\phi)(q)$, and not the truth value of $\mathrm{cl}(\phi)(q)$ itself, is because there may be many sets S that imply q and we want to combine the conclusions that can be drawn using each of them.

These arguments suggest that we take

$$\mathrm{cl}(\phi)(q) = \sum_{S \models q} u \bigvee \left[\bigwedge_{p \in S} \mathrm{cl}(\phi)(p) \right] \tag{2}$$

Once again, the motivation for (2) is as follows: There is a contribution for each set S that entails q. For each such set, we can compute the truth value of the conjunction of the elements of S by conjoining the truth values assigned to these elements individually. Since this conjunction only entails q and is not equivalent to it, we disjoin with unknown and then sum the results as S is allowed to vary.

As an example, suppose that our truth assignment ϕ labels the following statements as other than unknown:

$$\phi[\texttt{Quaker}(x) \wedge \neg\mathbf{ab}_r(x) \supset \texttt{pacifist}(x)] = t \tag{3}$$

$$\phi[\texttt{Quaker(Nixon)}] = t \tag{4}$$

$$\phi[\neg\mathbf{ab}_r(x)] = dt \tag{5}$$

$$\phi[\texttt{hippie}(x) \supset \texttt{pacifist}(x)] = t \tag{6}$$

$$\phi[\texttt{hippie(Nixon)}] = f \tag{7}$$

Quakers are pacifists unless they are (religiously) abnormal; hippies are definitely pacifists. Nixon is a Quaker but not a hippie. This example parallels one due to Reiter and Criscuolo [38]. What truth value should be assigned to cl(ϕ)[pacifist(Nixon)]?

There are two proofs that Nixon is a pacifist using information in the database. One uses (3)–(5); the other uses (6) and (7). If we assume that cl(ϕ) and ϕ agree on the sentences appearing explicitly in (3)–(7), the value to be assigned to cl(ϕ)[pacifist(Nixon)] is given by (2) as

$$\begin{aligned}
\mathrm{cl}(\phi)[\texttt{pacifist(Nixon)}] &= [u \vee (t \wedge t \wedge dt)] + [u \vee (t \wedge f)] \\
&= [u \vee dt] + [u \vee f] \\
&= dt + u \\
&= dt
\end{aligned}$$

The proof involving hippies has no impact on the overall truth value because Nixon is not known to be a hippie.

But now suppose we add the following to our database:

$$\phi[\texttt{Republican}(x) \wedge \neg\mathbf{ab}_p(x) \supset \neg\texttt{pacifist}(x)] = t \tag{8}$$

$$\phi[\texttt{Republican(Nixon)}] = t \tag{9}$$

$$\phi[\neg\mathbf{ab}_p(x)] = dt \tag{10}$$

Republicans who are not politically abnormal are not pacifists; Nixon is a Republican as well.

The new information does not lead to a new proof that Nixon is a pacifist, but because it leads to a default proof that Nixon is *not* a pacifist, the truth value of cl(ϕ)[pacifist(Nixon)] should be $*$ in this case. The expression (2) is not quite right; cl(ϕ)(q) should also include information about the negation of q. We add to (2) an additional term dealing with $\neg q$ and elevate the result to the status of a definition:

Definition 2.4 *The* closure *of a truth assignment ϕ is the k-minimal truth assignment $cl(\phi)$ satisfying*

$$cl(\phi)(q) = \sum_{S \models q} u \bigvee \left[\bigwedge_{p \in S} cl(\phi)(p)\right] + \neg \sum_{S \models \neg q} u \bigvee \left[\bigwedge_{p \in S} cl(\phi)(p)\right] \tag{11}$$

In our extended example, we can now compute the truth value assigned to cl(ϕ)[pacifist(Nixon)] to be:

$$\begin{aligned}
\mathrm{cl}(\phi)[\texttt{pacifist(Nixon)}] &= [u \vee (t \wedge t \wedge dt)] + [u \vee (t \wedge f)] + \neg[u \vee (t \wedge dt)] \\
&= [u \vee dt] + [u \vee f] + \neg[u \vee dt] \\
&= dt + u + df \\
&= *
\end{aligned}$$

The flexibility of bilattice-based knowledge representation has been described in a variety of papers; the original [15] showed that the ideas generalize first-order logic, some instances of default reasoning [36], and assumption-based truth maintenance [6]. These results have been extended [16] to include first-order ATMSs and circumscription [29, 30]. More recent work has applied the techniques to planning [22] and reasoning about action [18]. The bilattice-based theorem prover MVL [19] is available by anonymous ftp from t.uoregon.edu [128.223.56.46]. Documentation is also available [21].

The work in this paper will be concerned not with general bilattices, but with *distributive* ones:

Definition 2.5 *A bilattice* $(B, \wedge, \vee, \cdot, +, \neg)$ *will be called* distributive *if each of the bilattice operations distributes with respect to the others, so that* $x \wedge (y+z) = x \wedge y + x \wedge z$ *and so on [15].*

For distributive bilattices, $\mathrm{cl}(\phi)$ can be replaced with ϕ in the right hand side of the fixed-point equation (11):

Proposition 2.6 *Let* ϕ *be a truth assignment on a distributive bilattice. Then the closure of* ϕ *is given by:*

$$cl(\phi)(q) = \sum_{S \models q} u \bigvee \left[\bigwedge_{p \in S} \phi(p) \right] + \neg \sum_{S \models \neg q} u \bigvee \left[\bigwedge_{p \in S} \phi(p) \right] \tag{12}$$

where S *ranges over those subsets of* L *that entail* q *or* $\neg q$*.*

It is this proposition (and its modal generalization 2.12) that underlie the algorithms we will discuss. The bilattices corresponding to first-order logic, ATMSs and first-order ATMSs, planning, and reasoning about action are all distributive. The setting is sufficiently powerful to describe nonmonotonic reasoning based on autoepistemic logic [31], PROLOG's negation-as-failure operator, and Kripke-style possible worlds semantics.

2.2 Modal operators

What is a modal operator in this setting? We will simply take a modal operator to be any operator that is capable of manipulating the truth values of its sentential arguments:

Definition 2.7 *A* modal operator *is any* n*-ary function*

$$m : B^n \to B$$

from the bilattice of truth values to itself.

Given a modal operator m*, we extend our declarative language to include sentences of the form* $m(p_1, \ldots, p_n)$*, where* $p_i \in L$ *for each* i*. Such sentences will be called* modal.

As an example, we have already remarked that $\Box$ can be defined as the unary modal operator given by

$$\Box(x) = \begin{cases} t, & \text{if } x \geq_k t; \\ f, & \text{otherwise} \end{cases} \tag{13}$$

Thus $\Box p$ should be true if and only if p is known to be true. Note that $\Box(dt) = f$, since $t >_k dt$ and therefore $dt \not\geq_k t$. If p is labelled either false or unknown, then the truth value to be assigned to $\Box p$ should be false, since $\Box(f) = \Box(u) = f$ by virtue of (13).[2]

Another example is the modal operator **1**, corresponding to the identity function. Even conjunction can be viewed modally; the truth value assigned to $p \wedge q$ inherits a component from the conjunction (in the bilattice sense) of the truth values assigned to p and to q. These two modal operators are rather different from $\Box$, however:

[2]The $\Box$ that we have defined here is not the same as the usual modal operator of necessity, although the two are closely related. In fact, an examination of earlier results [17] shows that if we take as our bilattice the set F^W of functions from a set W of possible worlds into the four-point bilattice F, then the semantics of our $\Box$ match the S5 semantics that are typically used. A complete discussion of the fact that the bilattice approach generalizes Kripke's can be found elsewhere [17].

Definition 2.8 *A unary modal operator m will be called* monotonic *if it distributes with respect to $+$, so that $m(x+y) = m(x)+m(y)$ for all $x, y \in B$. A nonunary operator will be called modal if the result of fixing all but one of its arguments is modal.*

The modal operators $\mathbf{1}$ and $\wedge$ are monotonic on distributive bilattices; $\Box$ is not. That $\wedge$ is monotonic is simply a restatement of the distributivity axiom; to see that $\Box$ is not, note that

$$\Box(t+u) = \Box(t) = t$$

but

$$\Box(t) + \Box(u) = t + f = \bot$$

The distinction in Definition 2.8 is important because the semantics of monotonic and nonmonotonic modalities differ [17]. More specifically, it is generally assumed if m is a nonmonotonic modality, the truth value of $m(p_i)$ can be computed from the truth values of the p_i's alone. This is not true for monotonic modalities: $p \wedge \neg p$ (which uses the monotonic modality $\wedge$) is false even if both p and $\neg p$ are unknown. Put slightly different, the truth value assigned to an expression involving a monotonic modality can depend in subtle ways on semantic interactions among the modal arguments.

The recursive descent description of modality that we discussed in the introduction depends fundamentally on our ability to evaluate a modal expression $m(p_i)$ by assigning values to the various p_i. Because only nonmonotonic modalities can be evaluated in this fashion, it is only nonmonotonic modal operators that mark interruption points.

A consequence of this is that the locations of interruption points are typically fixed by the semantic contents of the database in question. One cannot add new interruption points by scattering the modal operator $\mathbf{1}$ throughout the database; this has no semantic impact but also introduces no new interruptions because $\mathbf{1}$ is monotonic.

It also appears that the points marked by nonmonotonic modal operators are the same as those where it is natural to interrupt the inference process. But perhaps this is to be expected: A modal operator such as "It is necessarily the case that" is indeed a natural point at which to break reasoning. In a practical system, "It is necessarily the case that p" might mean that p appears explicitly in the database or, if more time is available for computation, that p is a deductive consequence of the material in the database.

The following is proved elsewhere [17]:

Proposition 2.9 *Any modal operator on a distributive bilattice can be written in terms of the modal operator of necessity $\Box$ and monotonic modalities.*

Given the argument in the previous paragraph justifying interruption at the modality $\Box$, Proposition 2.9 provides philosophical justification for interruption at any nonmonotonic modal operator.

It should now be no surprise that we will evaluate a modal expression $m(p_1, \ldots, p_n)$ either by invoking the prover recursively on the p_i to determine their truth values or by simply searching for the p_i in the database and using whatever values we find. The first approach is correct, but the second may provide a sensible approximation if computation time is limited.

Perlis has made a similar but informal suggestion [34]. He suggests that in databases with a modal operator of knowledge, it should be possible to determine whether or not $\mathtt{knows}(q)$ at any point in time, independent of the state of the analysis of q. This idea is essentially what we are trying to formalize in this paper, working with a general description of modal operators as bilattice functions, and within a more precise formal setting.

What about closure in the presence of modality? Although (11) remains valid in this wider setting, the argument used to produce (12) by eliminating $\mathrm{cl}(\phi)$ from the right hand side of (11) cannot be applied to monotonic modalities.

For nonmonotonic modalities, there is no problem because it is impossible to work "backwards" through a nonmonotonic modality K to conclude (for example) p from Kp. The standard example is Moore's and arises in autoepistemic logic [31]. Imagine a database containing the single statement, "If I had a brother, I'd know it." The obvious conclusion here is that since you don't know you have a brother, you don't have one, but there is an additional possibility where you do know that you have a brother, and it follows *from that* that you have a brother after all. This is wrong – you shouldn't be able to work your way backwards from a hypothetical belief to a conclusion about the state of the world generally. A groundedness condition is the technical mechanism that ensures this:

Definition 2.10 *A modal operator m will be called* grounded *if for any ϕ and sentence $m(p_i)$,*

$$cl(\phi)(m(p_i)) = m(cl(\phi)(p_i))$$

Proposition 2.11 *Nonmonotonic modal operators are grounded.*

The groundedness conditions associated with nonmonotonic modalities provide the semantic isolation needed by the inference procedures we are about to describe. We therefore assume for the remainder of this paper that all modal operators are equipped with groundedness conditions, referring to such modal operators as *grounded* [17, Section 6]. This assumption might be valid because the operators are nonmonotonic, or because we are prepared to accept the limitations in expressiveness that the assumption entails. (For an example of such a limitation, see Section 5.3.)

In generalizing Proposition 2.6, we make use of the following notational convenience: If $S \subseteq L$ is a set of sentences, we denote by S_M the set of modal sentences in S and by S_0 the set of nonmodal sentences in S. We now have:

Proposition 2.12 *Let ϕ be a truth assignment and q a nonmodal sentence in our declarative language. Then*

$$cl(\phi)(q) = \sum_{S \models q} u \bigvee \left[\bigwedge_{p \in S_M} cl(\phi)(p) \wedge \bigwedge_{p \in S_0} \phi(p) \right] + \neg \sum_{S \models \neg q} u \bigvee \left[\bigwedge_{p \in S_M} cl(\phi)(p) \wedge \bigwedge_{p \in S_0} \phi(p) \right] \tag{14}$$

where S ranges over those subsets of L that entail q or $\neg q$.

Note that (14) is once again a fixed-point equation. However, the fact that cl(ϕ) is only applied to modal sentences in the right hand side of (14) means that the inherent circularity is restricted to situations where modal sentences are used in a proof of q. We will see shortly that this restriction has important practical implications.

At this point, suppose that we return to Fred and his ill-fated voyage; recall the PROLOG axiomatization appearing in Figure 1. Can we conclude `dead(Fred)`? We begin by presenting a method for translating PROLOG programs into our declarative language:

Definition 2.13 *Let s be the* PROLOG *statement*

```
h :- c1,...,cm,not(d1),...,not(dn).
```

By the declarative translation *of s, we will mean the sentence $D(s)$ given by*

$$c_1 \wedge \cdots \wedge c_m \wedge \neg\Box d_1 \wedge \cdots \wedge \neg\Box d_n \supset h$$

If P is a PROLOG *program, we will denote by $D(P)$ the conjunction of the declarative translations of each of the statements in P.*

The following is now an easy consequence of a variety of existing results [2, 17, 35, 45]:

Proposition 2.14 *Let P be a stratified* PROLOG *program. Then a positive atom* `a` *is a consequence of P if and only if $D(P) \models a$. A negative atom* `not(a)` *is a consequence of P if and only if $D(P) \models \neg\Box a$.* ■

In this particular example, the declarative translation of our PROLOG program is the following, where all of the variables are universally quantified:

$$\begin{aligned}
\texttt{falling-onto}(x,y) \wedge \neg\Box\texttt{good-parachute}(x) \wedge \neg\Box\texttt{soft}(y) &\supset \texttt{dead}(x) && (15)\\
\texttt{wearing-parachute}(x,p) \wedge \neg\Box\texttt{broken}(p) &\supset \texttt{good-parachute}(x) && (16)\\
\texttt{haystack}(y,h) \wedge \neg\Box\texttt{ouch}(h) \wedge \neg\Box\texttt{missed}(h) &\supset \texttt{soft}(y) && (17)\\
\texttt{pitchfork}(h,p) \wedge \neg\Box\texttt{missed}(p) &\supset \texttt{ouch}(h) && (18)
\end{aligned}$$

$$\begin{gathered}
\texttt{falling-onto}(\texttt{Fred},\texttt{field})\\
\texttt{wearing-parachute}(\texttt{Fred},\texttt{parachute})\\
\texttt{broken}(\texttt{parachute})\\
\texttt{haystack}(\texttt{field},\texttt{haystack})\\
\texttt{pitchfork}(\texttt{haystack},\texttt{pitchfork})\\
\texttt{missed}(\texttt{pitchfork})\\
\texttt{missed}(\texttt{haystack})
\end{gathered}$$

Our initial truth assignment ϕ labels all of the above sentences with t.

Since `missed(haystack)` is labelled as true by ϕ, $\mathrm{cl}(\phi)$ will need to label `missed(haystack)` and therefore $\Box$`missed(haystack)` as true as well. This means that $\neg\Box$`missed(haystack)` will be labelled false and there is no reason for us to label `soft(field)` as anything other than unknown because (17) cannot be applied.

With `soft(field)` labelled unknown, it follows that $\Box$`soft(field)` will be labelled false, so that $\neg\Box$`soft(field)` will be labelled t. We can conclude in a similar way that $\neg\Box$`good-parachute(Fred)` is also true, and can now use (15) to conclude that Fred does indeed die.[3]

We were able to assign a truth value first to `good-parachute(Fred)` and to `ouch(haystack)`, then to `soft(field)`, and finally to `dead(Fred)` because the database with which we are working is of a form that supports this sort of hierarchical reasoning. To formalize this, we make the following definition:

Definition 2.15 *Let ϕ be a truth assignment. By a* syntactic stratification *of ϕ we will mean a mapping s from the set of atoms into the nonnegative integers such that for any pair of atoms a_1 and a_2 appearing in a single sentence p with $\phi(p) \neq u$:*

1. *If both a_1 and a_2 appear outside the scope of all modal operators in p, then $s(a_1) = s(a_2)$.*

2. *If a_1 appears under the scope of a modal operator and a_2 does not, then $s(a_1) < s(a_2)$.*

If both a_1 and a_2 appear only under the scope of modal operators, no restriction is entailed. Furthermore, if a_1 or a_2 is an instance of an atom appearing in p, the same conditions apply.

A truth assignment ϕ will be called syntactically stratified *if there is some function s that is a syntactic stratification of ϕ.*

In the example we have been considering, Definition 2.15 applied to (15) – (18) gives us:

$$\begin{gathered}
s[\texttt{falling-onto}(x,y)] = s[\texttt{dead}(x)] > s[\texttt{good-parachute}(x)], s[\texttt{soft}(y)]\\
s[\texttt{wearing-parachute}(x,p)] = s[\texttt{good-parachute}(x)] > s[\texttt{broken}(p)]\\
s[\texttt{haystack}(y,h)] = s[\texttt{soft}(y)] > s[\texttt{ouch}(h)], s[\texttt{missed}(h)]\\
s[\texttt{pitchfork}(h,p)] = s[\texttt{ouch}(h)] > s[\texttt{missed}(p)]
\end{gathered}$$

[3] There are other possibilities in which, for example, `soft(field)` is labelled t and `dead(Fred)` remains unknown. These extensions are not grounded, however; details appear elsewhere [17].

for any x, y, h and p. We can therefore take

$$\begin{aligned}
s[\texttt{missed}(z)] = s[\texttt{broken}(p)] &= 0 \\
s[\texttt{pitchfork}(h,p)] = s[\texttt{ouch}(h)] &= 1 \\
s[\texttt{wearing-parachute}(x,p)] = s[\texttt{good-parachute}(x)] &= 1 \\
s[\texttt{haystack}(y,h)] = s[\texttt{soft}(y)] &= 2 \\
s[\texttt{falling-onto}(x,y)] = s[\texttt{dead}(x)] &= 3
\end{aligned}$$

This syntactic stratification matches Figure 2; the value assigned to any sentence is the distance of that sentence from the fringe of the modal tree. The figure only shows sentences appearing under the scope of modal operators; we have (for example)

$$s[\texttt{falling-onto}(x,y)] = s[\texttt{dead}(x)]$$

because they have identical status in (15).

We first determine truth values for sentences with $s = 0$, deciding that Fred misses the haystack and the pitchfork and that his parachute is broken. Next is $s = 1$: There is a pitchfork in the haystack but Fred misses it so it won't hurt to land there; Fred is wearing a parachute but it's not going to do him any good. At $s = 2$ we conclude that there is a haystack in the field but that it won't help to cushion Fred's fall. Finally, at $s = 3$ we realize that Fred is going to die because he's falling onto the field (and will miss the haystack).

The point of a syntactic stratification is that it allows us to compute the closure of a truth assignment incrementally, ignoring at step i any sentence involving a literal l with $s(l) > i$. We also have the following:

Proposition 2.16 *Let P be a* PROLOG *program. Then if $D(P)$ is syntactically stratified, P is stratified.*

Of course, this result is in the wrong direction, showing the PROLOG community's definition of stratification to be more general than ours. We generalize our definition of stratification substantially in Section 4.

Before moving on, let me summarize the results we have reviewed thus far:

1. It is possible to assign modal operators a truth-functional semantics. Furthermore, it can be shown [17] that in a bilattice setting, this semantics is a generalization of the standard Kripke semantics [28].

2. By determining the truth value of sentences before determining the truth values of the modal expressions to which they contribute, it is sometimes possible to stratify a declarative database in a way that allows incremental computation. Note that this stratification does not in and of itself imply the existence of an anytime algorithm; this issue is the contribution of the current paper and we turn to it now.

3 Anytime reasoning

In this section, we use Proposition 2.12 to develop an anytime approach to inference. Here is (14) again:

$$\mathrm{cl}(\phi)(q) = \sum_{S\models q} u \bigvee \left[\bigwedge_{p\in S_M} \mathrm{cl}(\phi)(p) \wedge \bigwedge_{p\in S_0} \phi(p) \right] + \neg \sum_{S\models\neg q} u \bigvee \left[\bigwedge_{p\in S_M} \mathrm{cl}(\phi)(p) \wedge \bigwedge_{p\in S_0} \phi(p) \right]$$

As we remarked earlier, $\mathrm{cl}(\phi)$ is only applied to modal sentences in the right hand side of the above expression; if ϕ is stratified, this enables us to reduce the problem of assigning a truth value to q to the problem of assigning truth values to sentences on lower stratification levels than q.

To make practical use of this idea, suppose that we are trying to determine the value assigned to q by $\mathrm{cl}(\phi)$. Using (14), we reduce the problem to that of assigning truth values to the sentences $p_1, \ldots, p_n$ appearing under the scope of modal operators used in proofs of q. Each p_i can now be analyzed similarly. If the database is syntactically stratified, this process is guaranteed to terminate and we can trickle the results back to determine the values assigned to each of the intermediate sentences and, eventually, to the original query q. We construct Figure 2 from the root downwards and then trickle the answer back to the original query.

The following procedure implements this. We first find possible proofs of the user's query q or of other sentences on which q depends, and then compute truth values that will be assigned by $\mathrm{cl}(\phi)$ to these sentences.

Procedure 3.1 Given a truth assignment ϕ, to respond to a nonmodal query q:

$J :=$ `find-proofs`(ϕ, q) (J contains justification information)
$\phi^J :=$ `update`(ϕ, J) (ϕ^J is a version of ϕ that takes this justification information into account)
if $\phi^J =$ failure **then** return failure **else** return $\phi^J(q)$

To find the proofs that bear on the query q, we maintain three lists. The first list U is a list of unexpanded nodes in the modal tree. This list thus contains propositions that still need to be considered by the system because they appear in modal expressions used to prove the original query q, its negation, or other sentences generated by the proof process. Expanded nodes correspond to propositions that have already been considered are retained in the second list E. The final list J associates, to each of these propositions, information about the sets S that entail it. Note also that (14) implies that the only information we need to retain about a particular set S is a list of the elements of S_M together with the truth value $\bigwedge_{p \in S_0} \phi(p)$.

Procedure 3.2 To compute `find-proofs`(ϕ, q), justification information relevant to q given the truth assignment ϕ:

$J := \emptyset$
$U := \{q, \neg q\}$ (U is a list of propositions still to be investigated)
$E := \emptyset$ (E is a list of sentences that have been considered)
repeat until $U = \emptyset$:
  $p :=$ an element of U
  $S :=$ the sentences used in a new proof of p
  if such an S can be found
    then $J := J \cup \{\langle p, \bigwedge_{w \in S_0} \phi(w), S_M \rangle\}$
      $U := U \cup [\{$sentences appearing inside elements of S_M or their negations$\}] - E$
    else remove p from U and add it to E
  end if
end repeat
return J

Having constructed the tree, we need to work through the entries in J and assign updated truth values to every sentence that appears there in accordance with (14); in the following procedure, R is a list of the nodes that have not yet been assigned values. At each step, we find a sentence all of whose modal dependencies (i.e., children) have been evaluated, assign it an appropriate truth value using (14), and then remove it from the list R. If we ever fail to find such a sentence, there must be a loop in the modal dependencies in the original database, and the procedure fails. (This can only happen for nonstratified databases.) The procedure returns a modified version of the original truth assignment ϕ.

Procedure 3.3 To compute $\mathtt{update}(\phi, J)$, the result of updating the truth assignment ϕ to incorporate information from the justifications in J:

$R := \{p | p$ or $\neg p$ appears as the leading element of a triple in $J\}$
(R contains sentences that need to have their truth values computed)
$\phi^J := \phi$
repeat until $R = \emptyset$:
  $C := \{p \in R |$for all $\langle p, v, S_M \rangle$ or $\langle \neg p, v, S_M \rangle$ in J, $S_M \cap R = \emptyset\}$
  if $C = \emptyset$ **then return** failure
  $p :=$ any element of C
  $\phi^J(p) := \sum_{\langle p,v,S_M \rangle \in J} u \bigvee \left[\bigwedge_{m(p_i) \in S_M} m(\phi^J(p_i)) \wedge v \right] +$
    $\neg \sum_{\langle \neg p,v,S_M \rangle \in J} u \bigvee \left[\bigwedge_{m(p_i) \in S_M} m(\phi^J(p_i)) \wedge v \right]$
  $\phi^J(\neg p) := \neg \phi^J(p)$
  $R := R - \{p, \neg p\}$
end repeat
return ϕ^J

As an example, suppose that we use the procedure to respond to our usual question about Fred. In order to make things a bit more compact, we rewrite our database as:[4]

$$\begin{gathered} j \wedge \neg \Box g \wedge \neg \Box s \supset d \\ w \wedge \neg \Box b \supset g \\ h \wedge \neg \Box o \wedge \neg \Box m_h \supset s \\ p \wedge \neg \Box m_p \supset o \\ j \wedge w \wedge b \wedge h \wedge p \wedge m_p \wedge m_h \end{gathered} \tag{19}$$

The first part of the analysis is summarized in the following table:

	U	E	J
1	d		
2	g, s	d	$\langle d, t, \{\neg \Box g, \neg \Box s\} \rangle$
3	b, s	g	$\langle g, t, \neg \Box b \rangle$
4	s	b	$\langle b, t, \emptyset \rangle$
5	o, m_h	s	$\langle s, t, \{\neg \Box o, \neg \Box m_h\} \rangle$
6	m_p, m_h	o	$\langle o, t, \neg \Box m_p \rangle$
7	m_h	m_p	$\langle m_p, t, \emptyset \rangle$
8	$\emptyset$	m_h	$\langle m_h, t, \emptyset \rangle$

(20)

We show only the atoms contained in U or E; as an example, U is initially set to $\{d, \neg d\}$ but we show only d. Since the set E of analyzed sentences and the set J of justifications grow monotonically as the procedure is executed, each entry above includes only that element, if any, that is newly added to E or to J. The entries in J themselves are of the form

$$\langle w, x, \{p_1, \ldots, p_n\} \rangle$$

where w can be proven from a set S with $x = \wedge_{w \in S_0} \phi(w)$ and $S_M = \{p_1, \ldots, p_n\}$.

In detail, what we have done is the following:

[4] The mnemonics are as follows: j is jump out of the plane, g is good parachute, s is a soft field, d is die, w is wear a parachute, b is a broken parachute, h is a haystack, o is ouch, m_x is miss x and p is a pitchfork.

1. Initially, E and J are set to $\emptyset$ and d (the query) is added to U. The negation of d is also added, but we don't show it.

2. The only proof of d or $\neg d$ from other information in the database is that d is a consequence of j (which itself is a consequence of (19) and so has truth value t), $\neg\Box g$ and $\neg\Box s$. We add g and s to U, move d from U to E, and add the indicated triple to J. The inference that we have drawn corresponds to the commonsense conclusion that people falling from airplanes typically die.

3. Searching for proofs of g or $\neg g$, we see that g follows from w and $\neg\Box b$, so b is added to U and a second entry is added to J. Fred's parachute will save him if it isn't broken.

4. Turning our attention to b, we see that b is a consequence of (19), so its truth value is t. This leads to the addition of an entry $\langle b, t, \emptyset\rangle$ to J; the third element of this entry is empty because no modal assumptions were used in the proof.

5. The remainder of (20) considers s (is the field soft?), o (is there a pitchfork around?), m_p (did Fred miss the pitchfork?) and m_h (did he miss the haystack?) in that order. U is now empty, and this phase of the analysis is complete.

A variety of choices were made as the procedure was executed. For example, we decided to consider g before s. The reason that we selected this particular order was to match the order of information in the original story, although the final answer is independent of the choices made (see Proposition 3.4). We will see shortly that the anytime version of this procedure may well be sensitive to the order in which proofs are discovered.

We now move to the second phase of the procedure – extracting the answer from the above table. This proceeds as follows:

	R	q	$\phi^J(q)$
1	d, g, b, s, o, m_p, m_h		
2	d, g, b, s, o, m_p	m_h	t
3	d, g, b, s, o	m_p	t
4	d, g, b, s	o	u
5	d, g, b	s	u
6	d, g	b	t
7	d	g	u
8	$\emptyset$	d	t

(21)

1. Initially, we set $\phi^J = \phi$.

2. Since m_h does not depend on any other modal sentences, we can compute the truth value assigned to m_h to be t. Fred misses the haystack. The sentence m_p is true similarly.

3. Continuing, o (ouch) depends on $\neg\Box m_p$ in order to be true. Since Fred does miss the pitchfork, o remains unknown.

4. s (the field is soft) will be true if the field contains a safe haystack (which it does, since $\neg\Box o$ is true because o is unknown) and if Fred hits the haystack (which he doesn't since $\neg\Box m_h$ is false because m_h is true). Since Fred misses the haystack, s remains unknown.

5. As for m_h and m_p, b is true because it has no other modal dependencies. g is unknown for reasons similar to s.

6. Finally, we evaluate the truth value to be assigned to d. Since $\phi^J(s) = \phi^J(g) = u$ (we couldn't prove either that Fred would land in a haystack or that he was wearing a working parachute), we assign f to both $\Box s$ and $\Box g$, and therefore t to both $\neg\Box s$ and $\neg\Box g$. This leads us to assign t to d as well, and we return t as a response to the original query. Fred dies.

Proposition 3.4 *If Procedure 3.1 terminates successfully, the value returned will be* $cl(\phi)(q)$.

Proposition 3.5 *Procedure 3.1 will terminate successfully provided that:*

1. *All calls to the underlying first-order theorem prover themselves terminate,*
2. *Only a finite number of proofs are found for any particular sentence in the database, and*
3. ϕ *is syntactically stratified.*

A consequence of Propositions 3.4 and 3.5 is that we have indeed managed to capture the meanings of PROLOG programs in our setting. And it might seem that we have done nothing more; Procedure 3.1 seems to be going to great lengths to do what PROLOG interpreters do already. In some sense this is true, although an advantage of working in our setting is that our results apply to modal operators generally as opposed to only the negation-as-failure operator used in PROLOG programs.

Far more interesting, however, is the following. Suppose that we change the termination criteria for the loop at the beginning of Procedure 3.2 to be:

repeat until either $U = \emptyset$ **or no time remains for additional analysis** ...[5]

As an example, suppose that we decide to terminate the analysis about Fred after step 7 of the inferential phase, so that we have not yet proved that Fred will miss the haystack. Now (21) becomes

	R	q	$\phi^J(q)$
1	d, g, b, s, o, m_p		
3	d, g, b, s, o	m_p	t
4	d, g, b, s	o	t
5	d, g, b	s	t
6	d, g	b	t
7	d	g	u
8	$\emptyset$	d	u

Since J doesn't contain any entries involving m_h, we assign m_h the truth value unknown. (Note that m_h does not appear *explicitly* in the database; it is only a consequence of (19).) We now conclude that s is true – since Fred misses the pitchfork, the field is a safe place to land. This causes us to label $\neg\Box s$ as false, and we no longer draw the conclusion that Fred dies.

Here are the conclusions that would be drawn if we were to terminate our analysis after each inference step:

	U	E	J	$\phi^J(d)$
1	d			u
2	g, s	d	$\langle d, t, \{\neg\Box g, \neg\Box s\}\rangle$	t
3	s, b	g	$\langle g, t, \neg\Box b\rangle$	u
4	s	b	$\langle b, t, \emptyset\rangle$	t
5	o, m_h	s	$\langle s, t, \{\neg\Box o, \neg\Box m_h\}\rangle$	u
6	m_p, m_h	o	$\langle o, t, \neg\Box m_p\rangle$	t
7	m_h	m_p	$\langle m_p, t, \emptyset\rangle$	u
8	$\emptyset$	m_h	$\langle m_h, t, \emptyset\rangle$	t

(22)

After the second step, we realize that people falling out of airplanes typically die. The third step introduces the parachute; since b does not appear explicitly in the database (it is only a consequence of (19)), we label $\Box b$ as false, $\neg\Box b$ as true, g as true, and therefore d as unknown. At step 4, we realize that b is true after all, and conclude that Fred dies.

[5] The "time remaining for additional analysis" need not include the time required by the second phase of the algorithm; this phase involves simply passing truth values up the tree and is effectively instantaneous. Alternatively, the two phases of the algorithm can be interleaved, so that the current estimate of $\mathrm{cl}(\phi)(q)$ is available at all times.

At step 5, we realize that the field contains a haystack; since m_h is not explicitly in the database, Fred's outlook once again improves. We observe the presence of the pitchfork in step 6 and realize that Fred will miss it in step 7. Finally, we realize in step 8 that Fred will miss the haystack and be killed after all. All of this is in keeping with the tenor of the original story.

Note that although we allow the inference procedure to be interrupted between invocations of the first-order prover, the first-order prover itself is not interrupted. The implementation works by modifying the database lookup mechanism so that it succeeds on any modal expression and then invoking the prover on p and allowing it to run until a proof is found.

Corollary 3.6 *Given unlimited computational resources, the value returned by the modified Procedure 3.1 in response to a query q is $cl(\phi)(q)$.*

Proof. This is a restatement of Proposition 3.4. ■

It is now reasonable to think of the modified procedure as anytime, although some slight subtleties are involved:

1. The modified procedure does not have an answer available at any time; we only compute the value of $\phi^J(q)$ when we have decided to terminate the first phase of the analysis. This is simple to change, however: It is easy to update ϕ^J whenever a new triple is added to J and thereby to ensure that an approximate answer is available at all times.

2. That the answer produced converges is guaranteed by Corollary 3.6. That the convergence is in some sense uniform is a bit harder to argue, since our conclusions change drastically from one step of the analysis to the next. Indeed, it is quite possible for the procedure to return the wrong answer at every point until the analysis is complete.

   The answers returned by our procedure are of uniformly improving quality in the sense that each considers more information than the previous one, since allowing more time for analysis monotonically extends the set J used by the procedure to cache intermediate conclusions. At any point in time, we make use of all information that has been determined to be relevant to the query.[6] We are not the only ones to view this sort of behavior as anytime; Drummond [9] makes similar remarks about a universal-planning algorithm that gradually considers more hypothetical situations in which an agent might find itself.

   In a fully declarative setting, this uniformity of information (but not of answer) is probably the best one can do. Agents embedded in real-time environments have no choice but to make mistakes; they will fail to consider some aspect of a problem that turns out to be relevant and draw conclusions that are simply wrong. Forming a conclusion, reversing it, and then reversing it again is something that we have all done; there is no reason to expect artificial systems to perform otherwise.

3. Although Corollary 3.6 guarantees that the final answer returned is independent of the syntactic structure of the database, the intermediate answers may well be sensitive to these syntactic features. As an example, if we had replaced (19) with seven individual axioms (one for each conjunct), we would have already been able to assign b the truth value t in step 3 of (22), thereby continuing to conclude that Fred dies. The intermediate answers are also sensitive to the order in which various subgoals are considered; had we noticed earlier that Fred missed the haystack, the subargument centered on the pitchfork could have been ignored.

4. Finally, although the eventual convergence of the procedure is guaranteed, we have presented no mechanism for estimating the computational resources needed to provide this convergence. It seems unlikely that there will be a sharp theoretical result to be had here, since such a result would depend on an ability to bound the number of proofs that will be found for any particular sentence in U.

[6] I am indebted to Becky Thomas for providing me with this argument.

4 Semantic stratification

Consider our example once again. When investigating the sentence $\texttt{good-parachute(Fred)}$, the details of Procedure 3.1 require us to look not only for proofs that Fred has a working parachute, but also proofs that he *doesn't* have one. To see why, suppose that (15) were replaced with

$$\texttt{falling-onto}(x, y) \wedge \neg M\texttt{good-parachute}(x) \wedge \neg M\texttt{soft}(y) \supset \texttt{dead}(x) \tag{23}$$

where the modal operator M ("maybe") indicates that x *might* have a working parachute. The above axiom says that people falling out of airplanes die if they are known *not* to have working parachutes and if they are known to be falling toward hard objects. This axiom would be suitable if people typically wore parachutes and the ground were made of Jello.

Now showing that $\texttt{good-parachute(Fred)}$ has truth value f instead of u will cause the truth value of $M\texttt{good-parachute(Fred)}$ to change from t to f, and this will impact our ability to use (23).

With the original axiom (15), however, there is no way that an explicit proof that Fred has no working parachute can affect our conclusion that he dies. This is because $\texttt{good-parachute(Fred)}$ affects our conclusions only via the modal sentence

$$\Box\texttt{good-parachute(Fred)} \tag{24}$$

and if the truth value of $\texttt{good-parachute(Fred)}$ changes from unknown to false there will be no impact on (24), since $\Box\texttt{good-parachute(Fred)}$ is false in either case. The next step in our development of Procedure 3.1 is to modify it to cater to this possibility.

Suppose that during our execution of Procedure 3.1, we have constructed a set J of justification triples that $\texttt{update}$ will use to respond to the original query q. Given an arbitrary set J of justifications and truth assignment ϕ, we continue to denote by ϕ^J the truth assignment $\texttt{update}(\phi, J)$ computed by Procedure 3.3.

Now suppose that we have a fixed sentence p; can a proof of p be relevant to the query q? If J is the set of justifications accumulated so far, then p will clearly be relevant to q if

$$\psi^J(q) \neq \phi^J(q) \tag{25}$$

where ψ agrees with ϕ except that $\psi(p) = t$, so that ψ is true at p. In informal terms, ψ and ψ^J correspond to what we would know if the attempt to prove p succeeded; ϕ and ϕ^J reflect what we know currently.

In fact, p can affect the truth value eventually assigned to q any time that p's truth value changes from its current value $\phi(p)$. To see why, suppose that we are working in the default bilattice and have a modal operator m with $m(u) = m(t) = u$ but $m(dt) = t$. Now if we have a sentence like

$$m(p) \supset q \tag{26}$$

it will certainly be important to consider proofs of p, since they might lead us to conclude that p is true by default and therefore that q is true. This leads us to the following definition:

Definition 4.1 *Given truth assignments ψ and ϕ and a set of sentences S, we will say that ψ* (positively) extends *ϕ at S if ψ is a (positive) extension of ϕ for which $\psi(x) = \phi(x)$ for all $x \notin S$.*

The intuition behind (25) corresponds to the observation that p will be relevant to q if there is some ψ that positively extends ϕ at p and such that

$$\psi^J(q) \neq \phi^J(q) \tag{27}$$

Overall, things aren't quite this simple. Consider (23); is

$$\neg\texttt{good-parachute(Fred)}$$

relevant to $\texttt{dead(Fred)}$? Not by virtue of (27), since changing the truth value of

$$\neg\texttt{good-parachute(Fred)}$$

from u to t isn't enough to change our belief that Fred survives the fall; we *also* have to know that the truth value of $\neg\texttt{soft(field)}$ changes from u to t.

Although we have to make assumptions about other sentences when determining the relevance of

$$\neg\texttt{good-parachute(Fred)}$$

to our original query, the sentences about which these assumptions are made are always elements of U. What we are doing is imagining that we in fact will succeed in our eventual search for a proof of some other sentence $w \in U$.

Definition 4.2 *Given a truth assignment ϕ, a query q, a set of sentences U and a set of justification triples J, we will say that p is* relevant *to q if there is some χ that positively extends ϕ at U and some ψ that positively extends χ at p such that $\psi^J(q) \neq \chi^J(q)$.*

In other words, p is relevant to q if, assuming that we learn enough to replace ϕ with χ, a change in p's truth value will affect the truth value to be assigned to q.

We are now in a position to rewrite Procedure 3.1. In the subroutine that computes the justification information, we require that the proposition being considered be relevant to the original query. We also include a trivial modification to deal with modal queries:

Procedure 4.3 Given a truth assignment ϕ, to respond to a query q:

> $J := \texttt{find-relevant-proofs}(\phi, q)$
> $\phi^J := \texttt{update}(\phi, J)$
> **if** $\phi^J =$ failure **then** return failure **else** return $\phi^J(q)$

Procedure 4.4 To compute $\texttt{find-relevant-proofs}(\phi, q)$, justification information relevant to U given the truth assignment ϕ:

> $J := \emptyset$
> **if** q is modal with $q = m(q_i)$, **then** $U := \{q_i, \neg q_i\}$ **else** $U := \{q, \neg q\}$ [1]
> $E := \emptyset$
> **repeat** until $U = \emptyset$ or no time remains for additional analysis: [2]
> $p :=$ an element of U that is relevant to q [3]
>   **if** no such p can be found **then** $U := \emptyset$
>   **else** $S :=$ the sentences used in a new proof of p
>   **if** such an S can be found
>   **then** $J := J \cup \{\langle p, \bigwedge_{w \in S_0} \phi(w), S_M \rangle\}$
> $U := U \cup [\{$sentences appearing inside elements of S_M or their negations$\}] - E$
>   **else** remove p from U and add it to E
>   **end if**
>   **end else**
>   **end if**
> **end repeat**
> **return** J

The changes are marked in the right margin:

1. If the query q is modal, we initially set U to contain the sentential arguments of the modal operator involved.

2. We terminate the execution of the algorithm if no time remains for further work.

3. We only analyze sentences that are relevant to the original query.

An implementation of this procedure needs some effective way to decide whether a particular sentence p or its negation is relevant to the original query q. This is achieved in practice by assuming that we can always take $\psi(p) = t$ in Definition 4.2, so that there is no effect such as that in the example surrounding (26). All naturally occurring modalities m appear to either preserve the t ordering or to invert it, so that if $x \geq_t y$ then uniformly either $m(x) \geq_t m(y)$ or $m(x) \leq_t m(y)$. This condition is enough to support the above assumption.[7]

We can now determine if p is relevant to q by considering p in combination with every subset of the set U of sentences currently under consideration. The structure of the modal dependencies can be used to prune substantially the number of possibilities considered, and many intermediate computations can be cached by keeping ϕ^J updated at all times and only computing differences between this truth assignment and a hypothetical one.

Corollary 4.5 *Given unlimited computational resources, the value returned by Procedure 4.3 in response to a query q is $cl(\phi)(q)$.*

The modification that produced Procedure 4.3 from Procedure 3.1 is important for a variety of reasons. The first is the one that led to the change in the first place; as an example, suppose that we include in our original axiomatization the sentence

$$\texttt{hard-to-prove} \supset \neg\texttt{good-parachute(Fred)}$$

so that if we can derive `hard-to-prove`, we will know for certain that Fred has no good parachute.

Procedure 3.1 would now have us attempt to derive `hard-to-prove`, since doing so will affect the truth value assigned to $\neg g$ and therefore potentially impact modal expressions depending on g. However, since the only modal expression involved is $\Box g$ and this expression is unaffected by a change in truth value of g from u to f, we need not consider `hard-to-prove` after all. Procedure 4.3 acts in accordance with this observation.

There are also examples where Procedure 4.3 succeeds but Procedure 3.1 doesn't. Here is one:

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

Animals are typically landlubbers; birds are typically flyers although penguins are an exception. Birds in tuxedos are penguins, and birds are animals. Opus is a bird in a tuxedo. Is he a landlubber?

The above axioms do not meet our narrow definition of stratification. The second axiom requires that we have

$$s[\texttt{bird}(x)] > s[\texttt{penguin}(x)]$$

while the third forces

$$s[\texttt{bird}(x)] = s[\texttt{penguin}(x)]$$

[7] As examples, $\Box$ preserves t and $\neg$ inverts it. Exceptions such as $\Box + \neg\Box$ seem to have no naturally occurring interpretations.

a contradiction. At this point, however, Procedure 3.1 still terminates successfully. Here is a trace similar to (20), where we have written l for `landlubber(Opus)`, *fly* for `fly(Opus)` and p for `penguin(Opus)`:

	U	E	J
1	l		
2	fly	l	$\langle l, t, \neg\Box fly\rangle$
3	p	fly	$\langle fly, t, \neg\Box p\rangle$
4	$\emptyset$	p	$\langle p, t, \emptyset\rangle$

Invoking the prover on `landlubber(Opus)`, we need to prove `animal(Opus)` and $\neg\Box$`fly(Opus)`. The last sentence is modal and will therefore be passed to subsequent steps of the algorithm; we can derive `animal(Opus)` from the fact that Opus is a bird. In other words, if $\neg\Box$`fly(Opus)` is true, Opus will be a landlubber. There is no other potential proof that Opus is a landlubber from information in the database.

To decide whether or not Opus can fly, we note that since he is a bird, he can fly unless he is known to be a penguin. We conclude that he is a penguin because he is a bird in a tuxedo.

We now modify our axiomatization to include the fact that penguins can't fly:

$$\texttt{penguin}(x) \supset \neg\texttt{fly}(x)$$

This seems innocuous enough, but leads to a proof that Opus is not a penguin. Since penguins can't fly, anything that can fly isn't a penguin. Since Opus is a bird, he can fly unless he is known to be a penguin. This leads to the conclusion that

$$\neg\Box\texttt{penguin(Opus)} \supset \neg\texttt{penguin(Opus)} \tag{28}$$

This makes sense: Since penguins are abnormal birds, we assume if possible that Opus is not a penguin.

Unfortunately, (28) causes us to add the following triple to J:

$$\langle \neg p, t, \neg\Box p\rangle \tag{29}$$

where p denotes the sentence `penguin(Opus)`. Since this triple makes it impossible to compute the truth value of p, Procedure 3.1 is doomed to fail.

Not so for Procedure 4.3. Since the effect of (29) is to change the truth value of p either from u to f (unknown to false) or from t to $\perp$ (true to a contradiction) and the value assigned to $\Box p$ is unchanged in either case (f in the first case and t in the second), $\neg p$ will never be relevant to l and we are able to conclude that Opus is a landlubber as previously.

The point here is that Procedure 4.3 can succeed for databases that are not syntactically stratified or for which Procedure 3.1 fails. Coupling this observation with Corollary 4.5, which tells us that the procedure is actually computing a semantic object (the truth value to be assigned to q in the closure), we are led to the following definition:

Definition 4.6 *A truth assignment ϕ will be called* semantically stratified *if Procedure 4.3 never fails when applied to ϕ.*

It might seem that this definition is too procedural to be of much use, but this is not the case; we will see in the next section, for example, that any stratified logic program is semantically stratified. We also immediately have:

Proposition 4.7 *Any syntactically stratified truth assignment is semantically stratified.*

Proof. This is Proposition 3.5 restated. ■

What we have shown thus far is that it is possible to view grounded modal operators as semantic markers for points at which a declarative inference procedure can be suspended, and that for semantically stratified databases, an anytime inference procedure can be built around this idea. The point that we have *not* addressed is to what extent interesting databases are semantically stratified.

5 Subsumption of existing work

We have presented an algorithm for anytime reasoning, shown that it terminates for a restricted class of declarative databases and shown that if it does terminate, it will return the answer supported by the multivalued semantics of modality.

Results such as these are quite parochial; we need to relate our techniques to existing work. We do that now, showing that our view of modality subsumes a variety of existing results from logic programming and knowledge representation.

5.1 Logic programs

The notion of stratification is not unique to us, of course; it originated in the logic programming community. Here is a stratified logic program:

```
a :- not(b).
a :- b.
```

From a logic programming point of view, the second axiom is "about" a and not b, so that we can stratify the program by determining the truth value of b before that of a.

From our point of view, however, the declarative translation of the second axiom is $b \supset a$, and is just as likely to allow us to conclude $\neg b$ from $\neg a$ as it is to allow us to conclude a from b. However, since the axiom $b \supset a$ can never be used to prove that b is *true*, we can still conclude that $\Box b$ is false and therefore that a follows in this example. The declarative translation of the above program fails to meet the requirements of Definition 2.15, but does satisfy Definition 4.6.

Here is a somewhat more formal way to look at it. The definition of stratification used by the logic programming community is the following [2]:

Definition 5.1 *Let P be a logic program. By an* LP-stratification *of P we will mean a mapping s from the set of atoms in P into the positive integers such that for every sentence*

```
a :- b1, ... bm, not(c1), ... not(cn).
```

we have $s(a) \geq s(b_i)$ for each i and $s(a) > s(c_j)$ for each j.

A logic program P will be called LP-stratified *if there is some function s that is an LP-stratification of P.*

Our definition of stratification requires $s(a) = s(b_i)$ for each b_i, since the declarative translation of (for example)

```
a :- b, not(c).
```

allows each of a and b to influence the other. Definition 5.1 only requires $s(a) \geq s(b_i)$ because it is realized that PROLOG cannot use information about a to draw conclusions about any of the b_i's. We can extend Proposition 2.16 to show that our definition is more general than that of the logic programming community:

Proposition 5.2 *If P is an LP-stratified logic program, then $D(P)$, the declarative translation of P, is a semantically stratified truth assignment.*

The converse of this result does not hold. Imagine, for example, that there is a strange genetic trait that skips generations. We might axiomatize this as:

```
strange(father(X)) :- not(strange(X)).        (30)
```

saying that if x lacks this genetic characteristic, x's father will have it. This axiom resembles an example of Van Gelder's [45].

An axiomatization including (30) can never be stratified because the same predicate occurs on both sides of the implication. In practice, however, investigation of

$$\texttt{strange}(\texttt{father}^k(x))$$

will lead us to consider $\texttt{strange}(\texttt{father}^{k-1}(x))$ so that no loop occurs. Members of the logic programming community have covered this sort of example by replacing (30) with all of its ground instances and modifying Definition 5.1 to deal with ground instances only [35, 45]. The resulting definition is that of *local* stratification, and a program including (30) can indeed be locally stratified. Such programs are typically semantically stratified for similar reasons.

Here is a PROLOG fragment that is not even locally stratified:

```
a :- not(b).
b :- not(a).
c :- not(d).
```

This fragment admits two minimal models. In one, a is true but b is not; in the other, b is true but a is not. Although this fragment is not semantically stratified either, Procedure 4.3 continues to terminate successfully when applied to c, since an investigation of c will never encounter the a-b loop.

Procedure 4.3 is not a panacea, however. Consider:

```
a :- not(b).
b :- not(a).
c :- a.
c :- b.
```

On an intuitive level, the query c should perhaps succeed, since c is a consequence of $a \vee b$ and this latter sentence holds in all minimal models of the above program. Although the formal semantics support this conclusion [17], Procedure 4.3 fails to return an answer.

5.2 Negative subgoals

We have focussed thus far on applying our ideas to a specific modal operator, the $\Box$ appearing in PROLOG programs. But the ideas can be applied to other modalities as well, allowing us to generalize work from a variety of other areas. One specific example involves the following program fragment:

```
g(Z) :- not(h(Z)).
h(a).
```

What should our response be to the query $g(z)$?

The problem is that the query generates the subgoal $\neg h(z)$ and conventional PROLOG interpreters are incapable of dealing with negative subgoals that contain unbound variables. (Such subgoals are called *floundered.*) Two solutions have been proposed.

The first, due to Chan [3], is to respond to the query $g(z)$ with the answer, "all z except for $z = a$." Chan proposes implementing this suggestion by maintaining and manipulating a list of possible exceptions to a particular query; we will not repeat the details here.

We can capture this in our setting by using as our set of truth values not simply the four-point bilattice $F = \{t, f, u, \bot\}$ but the set of *functions* from possible bindings into F. Thus the truth value to be assigned to our query $g(z)$ should be the function that maps all z except $z = a$ into true, and maps $z = a$ into unknown.

The set of functions from binding lists into F inherits a bilattice structure from the existing structure on F itself; the bilattice operations are all computed pointwise. Thus if f_1 and f_2 are two such functions and σ is a binding list,

$$(f_1 \wedge f_2)(\sigma) = f_1(\sigma) \wedge f_2(\sigma)$$

Given this, how does the analysis of our example proceed? The declarative translation of the logic program is

$$\neg\Box h(z) \supset g(z)$$
$$h(a)$$

and the first phase looks like this:

	U	E	J
1	$g(z)$		
2	$h(z)$	$g(z)$	$\langle g(z), t, \neg\Box h(z)\rangle$
3	$\emptyset$	$h(z)$	$\langle h(z), t \text{ if } z = a, \emptyset\rangle$

The second phase is:

	R	q	$\phi^J(q)$
1	$g(z), h(z)$		
2	$g(z)$	$h(z)$	t if $z = a$
3	$\emptyset$	$g(z)$	t unless $z = a$

In the final line above, we have that since the truth value assigned to $h(z)$ is true at $z = a$ and unknown elsewhere, the truth value assigned to $\Box h(z)$ is true at $z = a$ and false elsewhere. The truth value assigned to $\neg\Box h(z)$ is therefore false at $z = a$ and true elsewhere.

Proposition 5.3 *Using the functional truth values described above, the result returned in response to a query q is the same as that computed by Chan's algorithm.*

Chan needs to require that there are no cycles in the logic program being considered; our definition of stratification covers this restriction naturally.

I have proposed a different approach to this problem [20], arguing that the exceptions to $g(z)$ may be either difficult to compute or infinite in number. As a result, we may want to respond to the query $g(z)$ simply by reporting, "yes, except for specific choices for z." A formalization of this notion proposes, roughly speaking, that we should be prepared to return an answer without considering possible exceptions if we know that every exception involves binding some variable in the answer [20].

In the example we have been considering, this means that the empty binding is a legitimate response to our original query, since the only exception binds z to a. To see the value of this, suppose that we change the example to the following:

```
g(Z) :- not(h(Z)).
h(a) :- not(j).
```

where j is a sentence the truth value of which is difficult to determine. Investigation of the query $g(z)$ now forces us to decide whether or not j is true, although we may not be interested in this exception to our conclusion that g holds in general.

We formalize this by introducing a new modal operator X on our bilattice of functional truth values; X stands for "ignore exceptions." For a functional truth value f, here is the definition of X:

$$X(f)(\sigma) = \vee_{\chi \leq \sigma} f(\chi)$$

This probably needs some explanation. Since f, the bilattice argument of X, is a function on binding lists, $X(f)$ will be a function on binding lists as well. The value taken by $X(f)$ on a particular binding list σ is the disjunction of the values taken by f on all binding lists that are less specific than σ; we have written $\chi \leq \sigma$ to indicate that a specific binding list χ satisfies this criterion.[8]

[8] We are saying that one binding list is less specific than another if it binds fewer variables, or binds variables to less specific values.

In the example we are considering, the truth value eventually assigned to g is the following function, where we assume that the attempt to prove j fails:

$$\begin{array}{c|c} \sigma & g(\sigma) \\ \hline \emptyset & t \\ z=a & u \end{array} \tag{31}$$

In other words, the empty binding is mapped to true, since we can prove $g(z)$ for general z. But if z is bound to a, the proof fails because $\Box j$ is false and $h(a)$ is therefore true.

It follows from (31) that the value assigned to the modal expression $X(g)$ is given by:

$$\begin{array}{c|c|c} \sigma & g(\sigma) & X(g)(\sigma) \\ \hline \emptyset & t & t \\ z=a & u & t \end{array}$$

The reason for the change in the last row is that $z = a$ is more specific than the empty binding, and we disjoin the values taken at both bindings when computing $X(g)$. In other words, $\mathrm{cl}(\phi)(X(g))$ is simply true.

Proposition 5.4 *Suppose that we have a database ϕ and a query q. Then the value computed by Procedure 4.3 in response to the query $X(q)$ will be the same as the value computed by Ginsberg's algorithm [20] for dealing with floundered subgoals.*

In addition, there is a natural connection between the details of the earlier algorithm and the notion of relevance presented in Definition 4.2. In the example we are considering, investigation of $X(g)$ leads to the following:

$$\begin{array}{c||c|c|c} & U & E & J \\ \hline 1 & g(z) & & \\ 2 & h(z) & g(z) & \langle g(z), t, \neg\Box h(z)\rangle \\ 3 & j & h(z) & \langle h(z), t \text{ if } z=a, \neg\Box j\rangle \end{array} \tag{32}$$

But now j is not relevant to the original query $X(g)$ (although it would be relevant to the query g), and will therefore not be considered further. In fact, a careful examination of the previous algorithm [20] shows that the details of this algorithm are designed to achieve precisely the computational benefits that arise from the use of Definition 4.2.

5.3 Planning and reasoning about action

The final example I would like to consider involves the construction of a purely declarative system to plan or reason about action. The basic reason that planners need to be declarative is that things go wrong; at least on occasion, any intelligent system will need to pause and reassess its domain knowledge instead of simply invoking a planner and acting on the results.

One reason that declarative systems have not been applied to this domain is simply that planners need to be anytime; the ideas we have presented thus far go at least some way toward addressing this difficulty. A more serious reason, however, is that existing descriptions of action are simply inadequate; there is no computationally viable formulation of the frame or other temporal axioms.

In nondeclarative systems, these difficulties are addressed using what Chapman calls a *modal truth criterion* [5]. Roughly speaking, a modal truth criterion combines information about the validity of fluents at one time with with information about action occurrence to determine whether or not a particular fluent holds at some subsequent time.[9] The modal truth criterion is aptly named; as we will see, it really does correspond not to an axiom, but to a modality.

[9]By a *fluent*, we will mean something that can be true or false in a particular situation, such as `wearing(fred,parachute)` saying that Fred is wearing his parachute.

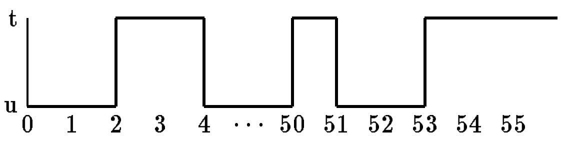

Figure 5: A fluent that is true three times

The approach we take views the truth value to be assigned to a particular fluent not as a simple value, but as a function from a set T of temporal elements into a base set of values B. The situation is like that in the last section, except the domain of the function is not the set of possible variable bindings, but is instead the set of times T. At this point, we remain uncommitted with regard to the question of whether the elements of T are integral times, real-valued times, plans (linear or nonlinear) or some other choice. We also remain uncommitted as to the choice of base bilattice B; it might be the four-point bilattice F, the default bilattice D, or some other bilattice incorporating probabilistic or other information. As an example, however, if we settle on T being the integers and B being the first-order bilattice, Figure 5 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 $t = 53$.

There are two modal operators needed to discuss action in this setting. The first is needed simply to separate the occurrence of an action from its effects; if an action occurs at time t_0, the effects will presumably not appear until time $t_0 + 1$. Perhaps one second after a valve is closed, the pressure in the associated pipe will increase.

Of course, the truth value assigned to the sentence, "The valve has just been closed," is a function, in this case the function mapping t_0 to true and all other times to unknown. The truth value of the sentence, "The pressure increased," should now be the function that is true at $t_0 + 1$ and unknown at other times. We accomplish this using a modal operator `delay`, given by

$$[\texttt{delay}(g)](t+1) = g(t) \tag{33}$$

As with the modal operator X of the previous section, `delay` accepts a function as its argument and returns a function as its result. Figure 6 shows the result of applying `delay` to a temporal function that changes from true to false.

The second modal operator we need corresponds directly to the modal truth criterion. If $g(t)$ indicates those time points at which a particular fluent λ is either established (i.e., made true) or disestablished (made false), then $\texttt{propagate}(g(t))$ will be the truth value of the fluent itself, which presumably persists from its points of establishment into the future by virtue of the frame axiom. Figure 6 also shows the result of applying `propagate` to the temporal function g. Between times 3 and 51, the result is true by default; after time 52, it is false by default.

From the two modal operators `delay` and `propagate`, we can build an anytime description of action. If a is an action that causes a fluent λ to be true in a persistent way (such as putting a tool in a toolbox), we write[10]

$$\texttt{propagate}(\texttt{delay}(a)) \supset \lambda \tag{34}$$

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

$$\texttt{delay}(a) \supset \lambda \tag{35}$$

[10] The axiom (34) 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 by reifying the fluents so that they can be treated as objects in our language, and replacing (34) with an axiom like

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

A complete axiomatization appears later in this section.

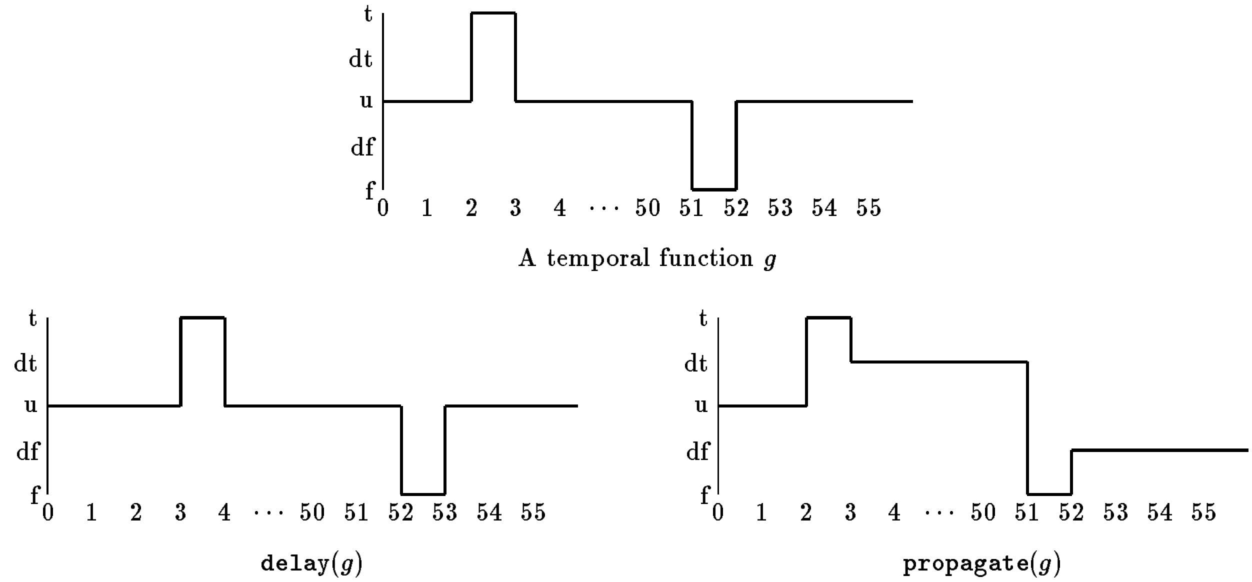

Figure 6: Delay and `propagate`

Note that the future behavior of any particular fluent is determined not by applying some blanket frame axiom, but instead by an axiom like (34) or (35) describing this future behavior when the fluent is first asserted (i.e., when the robot puts a tool in the toolbox or strikes a match). This is an important distinction between our approach and the conventional one; we prefer our ideas because 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 a blanket frame axiom indicating that we should do so.

This is related to the well-known *problem of induction* [43]. 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 by Myers and Smith [32].[11]

There are two other points to be made about this example. The first involves the modal operator `delay`; the second, the impact of relevance notions in this setting.

The point we would like to make about `delay` is that it is monotonic. From a formal point of view, this means that it cannot serve as an interruption marker unless it is equipped with artificial groundedness conditions. The impact of such groundedness conditions is that although we will remain able to reason from a sentence p (saying, for example, that an action occurred at some specific time) to the sentence `delay`(p) (giving the effects of the action), we will *not* be able to reason in the other direction. If we observe the effects of an action, we will not be able to conclude that the action occurred because this inference step is not grounded – one is not permitted to draw conclusions about sentential arguments from the truth of a grounded modal expression. There is no obvious solution to the conflict between a natural desire to view `delay` as an interruption marker and a desire to reason backwards through this particular modality.

With regard to relevance, the implications can be quite subtle. Imagine that our goal is to show that for

[11] Other authors 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 [23, 37, 41].

some sentence p, **propagate**(p) holds at time 54. There is clearly no point to considering arguments that lead to the conclusion that p is true at a time later than 54, since **propagate** will have no useful effect when applied to such a truth value.

Ignoring times later than 54, suppose that we manage to show that p was inserted into the database at time 2 (as in the figure), so that **propagate**(p) now holds at time 54 by virtue of the frame assumption. Now there is no reason to find other points at which p is inserted into the database; the information we already have suffices. But there *is* reason to search for times at which p is removed from the database, provided that these times are no later than time 54. This is the consistency check in our setting.

Continuing as in the diagram, suppose that we find that p is disestablished at time 52. Now there is no point in showing that it is reestablished at any time other than 53 or 54, and so on.

Axiomatization We now present a complete axiomatization of a STRIPS-like theory. Once again, the strength of our methods lies not only in our ability to formalize existing planners, but also in our ability to use declarative descriptions of action. We can easily modify our description to include actions of variable duration, conditional effects and so on.

We take the bilattice of base truth values to be the four-point bilattice F, and use three separate causal predicates:

1. **causes**(a, p) means that the action a causes the fluent p to be true instantaneously.

2. **adds**(a, p) means that the action causes the fluent to be true in a way that is expected to persist into the future.

3. **deletes**(a, p) means that the actions causes the fluent to be false in a way that is expected to persist into the future.

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(P) :- causes(A,P), delay(succeeds(A)).
holds(P) :- propagate(triggers(P)).
triggers(P) :- adds(A,P), delay(succeeds(A)).
not(triggers(P)) :- deletes(A,P), delay(succeeds(A)).
```

Our notation here involves a first-order negation operator as opposed to the negation-as-failure operator used in PROLOG. The `not` appearing in the head of the last of the above rules is first-order negation.

An action succeeds if it occurs and all its preconditions are satisfied:

```
succeeds(A) :- occurs(A), prec(A,P), all-hold(P).
all-hold([]).
all-hold([X|Y]) :- holds(X), all-hold(Y).
```

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.

```
occurs(init).          ; true at time 0 only
prec(init,[]).
```

To see our formal ideas at work, let us look at a simple domain where Fred, instead of falling out of an airplane, is the target of an assassination attempt. (The example bears a passing resemblance to the infamous Yale shooting problem [24].) This domain has three actions, `load`, `unload` and `shoot`, all of which involve the gun being used in the murder. Shooting Fred has as a precondition that the gun be loaded:

```
prec(load,[]).
adds(load,loaded).

prec(unload,[]).
deletes(load,loaded).

prec(shoot,[loaded]).
deletes(shoot,alive).
```

In the initial situation, Fred is alive and the gun is loaded; both facts are expected to persist into the future:

```
adds(init,alive).
adds(init,loaded).
```

Fred's security guard comes by at time 1 and unloads the gun, but it is reloaded at time 10. The assassin shows up at time 20:

```
occurs(unload).         ; true at time 1 only
occurs(load).           ; true at time 10
occurs(shoot).          ; true at time 20
```

When we indicate "true at time t," we mean that these facts are inserted into the database with truth values indicating that they are true at these times. Recall that our responsibility in providing the input theory is to provide a truth assignment in the sense of Definition 2.2, as opposed to simply a list of true sentences (i.e., sentences true at all times).

Here is how the analysis proceeds. The mnemonics are: a is the query, holds(alive); p is propagate and d is delay. We use subscripted s for actions' succeeding; s_i is init succeeding, s_s is shoot succeeding and s_l and s_u are load and unload succeeding. In a similar way, t_a is triggers(alive) and t_l is triggers(loaded). At each point in the following analysis, we also indicate the current value that would be taken by holds(alive) were we to compute ϕ^J:

	U	E	J	$\phi^J(a)$
1	a			u
2	t_a	a	$\langle a, t, p(t_a)\rangle$	u
3	s_i, t_a		$\langle t_a, t, d(s_i)\rangle$	u
4	t_a	s_i	$\langle s_i, t \text{ at } 0, \emptyset\rangle$	t
5	s_s	t_a	$\langle \neg t_a, t, d(s_s)\rangle$	t
6	t_l	s_s	$\langle s_s, t \text{ at } 20, p(t_l)\rangle$	t
7	t_l		$\langle t_l, t, d(s_i)\rangle$	until time 21
8	s_u, t_l		$\langle \neg t_l, t, d(s_u)\rangle$	until time 21
9	t_l	s_u	$\langle s_u, t \text{ at } 1, \emptyset\rangle$	t
10	s_l	t_l	$\langle t_l, t, d(s_l)\rangle$	t
11	$\emptyset$	s_l	$\langle s_l, t \text{ at } 10, \emptyset\rangle$	until time 21

The details are as follows:

1. Initially, the goal is simply a: What can we say about holds(alive)?

2. We can prove a by finding times at which Fred's being alive is triggered (i.e., times at which t_a is true), and propagating the result. So a is a consequence of $p(t_a)$. Since t_a itself is currently unknown, a remains unknown. There are no other ways to derive a, so it is moved to E.

3. We can derive triggers(alive) by using the fact that init triggers Fred's being alive; this is how we establish information about the initial state. We have not yet considered proofs of $\neg t_a$; a itself remains unknown because we have not yet assigned a truth value to succeeds(init).

4. The single proof of s_i (the success of the dummy action `init`) tells us that `init` succeeds at time 0; no modalities are used. We move s_i to E and can also conclude that

$$\texttt{delay(succeeds(init))}$$

is true at time 1, so that

$$\texttt{propagate[delay(succeeds(init))]}$$

is true at all times. Fred is always alive.

5. Now we look for proofs of $\neg t_a$. What can cause Fred to become not alive? The only proof found involves the assassination succeeding, so we add s_s to U.

6. The shooting will succeed at time 20, provided that the gun is loaded at that time. This is the information in the justification triple added in this step of the analysis; we add t_l to U and continue.

7. One way to trigger the gun's being loaded is to realize that `init` does so, since the gun is loaded in the initial situation. But we have already considered s_i, which is in E at this point. We conclude from this that the gun will be loaded at time 20, that the assassination attempt will succeed, and therefore that Fred is alive only until time 21.

8. We continue to look for new proofs of t_l or $\neg t_l$; in fact, we look for proofs of $\neg t_l$ only since a new proof of t_l would not be relevant to our overall conclusion. We see that $\neg t_l$ is a consequence of the `unload` action succeeding, so we add s_u to U.

9. We next note that s_u is true at time 1, so that the gun is presumably unloaded from time 2 on, the shooting now fails, and Fred is believed to survive after all.

10. At this point, t_l is relevant once again, and a new proof is found from the information in the database – instead of relying on the initial situation to load the gun, we look for an explicit `load` action.

11. There is a successful `load` action at time 10. The gun is loaded when the assassination is attempted and Fred dies at time 21. At this point U is empty and the analysis is complete.

Note the natural way in which our conclusions change. Initially, we don't know whether Fred is alive or not, and we begin by applying the frame assumption to conclude that he is and is likely to remain so. We then conclude that the assassin will succeed in killing him by a similar application of the frame axiom to the gun's being loaded in the initial situation.

Further consideration reverses this conclusion when we realize that the security guard's unloading the gun invalidates this reasoning. But the gun is reloaded at time 10, and we eventually conclude that the shooting succeeds after all. An alternative line of reasoning, more stable in this particular instance, would have us conclude initially that the gun was loaded not because of the initial `init` action, but because of the explicit loading action at time 10. This leads to an argument that is not subsequently defeated by other information in the database.

5.4 Step logics

Let me end this section not with another example, but with a comparison between our work and the most successful existing approach to anytime declarative reasoning, Elgot-Drapkin and Perlis' *step logic* [10, 11]. In step logic, one arranges to be able to interrupt a reasoning system after it has taken a certain number of inference steps en route to reaching a conclusion about a particular query.

The fundamental difference between this approach and ours is one of syntax as opposed to semantics. The interruption points used by step logics are necessarily syntactic objects; what constitutes a single inference using one rendering of a declarative database may correspond to many inferences if a different description is employed.

In the approach that we have described, the interruption markers are not inference steps but modal operators – objects that are fundamentally semantic as opposed to syntactic in nature. The definitions we have given (such as the original Definition 2.15 of stratification) appear to be appealing to syntactic features of our database, but this is an illusion; since the fundamental building blocks of our system are semantic, the overall description is also.

As an example, it is not difficult to show that Definition 4.6 will label a truth assignment as semantically stratified whenever it has a unique k-minimal closed and grounded extension. This is clearly a semantic notion and corresponds to the nonmonotonic reasoning community's idea of a declarative theory having a unique extension.

Granted, the truth-functional semantics we have given to modal operators is not the standard one. Nevertheless, modal operators remain semantic objects within our system, and the points at which we interrupt our reasoning are semantic as a result.

Is this a strength or a weakness? It might seem that the location of interruption points is really a metalevel issue; using either a syntactic description or an explicitly metalevel semantic description would allow us to discriminate between interruption markers and the base-level information already present in our system. But perhaps not: The argument can certainly be made that the points at which one wants to interrupt the conventional semantics of a declarative database should be the same as the points at which one also wants to interrupt the associated inference engine.

6 Conclusion and future work

My aim in this paper has been to suggest that modal operators can improve the usefulness of declarative systems by marking points at which inference can be suspended and an approximate answer returned. Viewing the modal operators truth-functionally, their correct evaluation requires a recursive call to the theorem prover on the sentential arguments of any particular modal expression; if the computational resources needed by this recursive call are not available, we can instead use the values found by simply searching the database for these sentences.

Using this idea, we developed an anytime declarative procedure that responds to queries by repeatedly calling a conventional first-order theorem prover and then accumulating the results. Approximate answers are computed quickly and gradually refined if time permits. Although the final answer returned by the system may appear to be changing in a nonsystematic way, the quality of the answer improves uniformly as additional information is considered.

Acknowledgement

This work has been supported by the Air Force Office of Scientific Research under contract 92-0693 and by ARPA/Rome Labs under contracts F30602-91-C-0036 and F30602-93-C-00031. I would like to thank David Etherington, Don Geddis, Alon Levy, Drew McDermott, Nils Nilsson, David Smith, Lynn Stein and Becky Thomas for many useful suggestions, as well as the anonymous reviewers of earlier drafts of this paper.

A Proofs

Proposition 2.6 *Let ϕ be a truth assignment on a distributive bilattice. Then the closure of ϕ is given by:*

$$cl(\phi)(q) = \sum_{S\models q} u \bigvee \left[\bigwedge_{p\in S} \phi(p) \right] + \neg \sum_{S\models\neg q} u \bigvee \left[\bigwedge_{p\in S} \phi(p) \right]$$

where S ranges over those subsets of L that entail q or $\neg q$.

Proof. This is Theorem 10.2 of [15]. ■

Proposition 2.11 *Nonmonotonic modal operators are grounded.*
Proof. This is an immediate consequence of Definitions 5.7 and 6.3 and Proposition 6.4 of [17]. ■

Proposition 2.12 *Let ϕ be a truth assignment and q a nonmodal sentence in our declarative language. Then*

$$cl(\phi)(q) = \sum_{S\models q} u \bigvee \left[\bigwedge_{p\in S_M} cl(\phi)(p) \wedge \bigwedge_{p\in S_0} \phi(p) \right] + \neg \sum_{S\models \neg q} u \bigvee \left[\bigwedge_{p\in S_M} cl(\phi)(p) \wedge \bigwedge_{p\in S_0} \phi(p) \right] \tag{36}$$

where S ranges over those subsets of L that entail q or $\neg q$.
Proof. We provide only a sketch here; a more formal derivation can be produced by modelling the proof of Theorem 10.2 in [15].

We know from Definition 2.4 that the closure of a truth assignment ϕ must satisfy

$$\mathrm{cl}(\phi)(q) = \sum_{S\models q} u \bigvee \left[\bigwedge_{p\in S} \mathrm{cl}(\phi)(p) \right] + \neg \sum_{S\models \neg q} u \bigvee \left[\bigwedge_{p\in S_M} \mathrm{cl}(\phi)(p) \right] \tag{37}$$

Let us deal only with the first term in (36); the second deals with proofs of $\neg q$ and is similar.

If a p used in a proof of q in (36) is a modal sentence, then since $p \in S_M$, (36) reduces to (37). So the case of interest involves situations in which q can be proven from other nonmodal sentences in the database.

Specifically, suppose that q can be proven from the sentence a together with the modal sentence m_a, and that a can be proven from b together with the modal sentence m_b. It follows that q is a consequence of the set $\{b, m_a, m_b\}$. We assume further that there are no derivations of b from other sentences in the database.

The relevant term in our original result (37) describing the closure of a truth assignment is now:

$$\sum_{S\models q} u \bigvee \bigwedge_{p\in S} \mathrm{cl}(\phi)(p) = u \vee [\mathrm{cl}(\phi)(a) \wedge \mathrm{cl}(\phi)(m_a)] + u \vee [\mathrm{cl}(\phi)(b) \wedge \mathrm{cl}(\phi)(m_a) \wedge \mathrm{cl}(\phi)(m_b)] \tag{38}$$

Since there are no proofs of b from other facts in the database, $\mathrm{cl}(\phi)(b) = \phi(b)$; since the only proof of a is from b,

$$u \vee \mathrm{cl}(\phi)(a) = u \vee [\phi(a) + \mathrm{cl}(\phi)(b) \wedge \mathrm{cl}(\phi)(m_b)]$$

Inserting these results into (38) gives us

$$\begin{aligned} & u \vee [(\phi(a) + \mathrm{cl}(\phi)(b) \wedge \mathrm{cl}(\phi)(m_b)) \wedge \mathrm{cl}(\phi)(m_a)] + u \vee [\mathrm{cl}(\phi)(b) \wedge \mathrm{cl}(\phi)(m_a) \wedge \mathrm{cl}(\phi)(m_b)] \\ &= u \vee [\phi(a) \wedge \mathrm{cl}(\phi)(m_a) + \mathrm{cl}(\phi)(b) \wedge \mathrm{cl}(\phi)(m_a) \wedge \mathrm{cl}(\phi)(m_b)] \\ &= u \vee [\phi(a) \wedge \mathrm{cl}(\phi)(m_a) + \phi(b) \wedge \mathrm{cl}(\phi)(m_a) \wedge \mathrm{cl}(\phi)(m_b)] \end{aligned}$$

as appears in (36). Roughly speaking, we can use the value assigned to a by ϕ instead of by $\mathrm{cl}(\phi)$ because any additional proofs of a will lead to proofs of q directly and appear in (36) in that guise. In the example we have considered, the proof of a from b is reflected in the direct proof of q from b, so that the dependence of q on $\phi(b)$ already appears in (36).

There is one possibility that we have not yet considered. It might be possible to prove b from a, so that a legitimate circularity appeared in (36). In this case, however, the ultimate proof of q would depend on a together with other assumptions (at least m_b in this example), and the original proof of q from a alone would necessarily use fewer assumptions and therefore be the only effective contributor to the truth value of q in the closure. ■

Proposition 2.16 *Let P be a* PROLOG *program. Then if $D(P)$ is syntactically stratified, P is stratified.*
Proof. This is in fact obvious, given the eventual definitions in Section 5.1. The definition of LP-stratification that appears there is strictly stronger than Definition 2.15 of syntactic stratification. ■

Proposition 3.4 *If Procedure 3.1 terminates successfully, the value returned will be $cl(\phi)(q)$.*
Proof. This is fairly obvious: We can show by induction that when any sentence p is removed from R in the **update** procedure, the value of $\phi^J(p)$ is set to $\mathrm{cl}(\phi)(p)$. The base case involves sentences that do not

depend on modal expressions for their proofs; the inductive step assumes that all of the modal expressions have already had their truth values correctly evaluated. It follows that $\phi^J(q) = \mathrm{cl}(\phi)(q)$. ■

Proposition 3.5 *Procedure 3.1 will terminate successfully provided that:*

1. *All calls to the underlying first-order theorem prover themselves terminate,*
2. *Only a finite number of proofs are found for any particular sentence in the database, and*
3. *ϕ is syntactically stratified.*

Proof. Consider the three procedures given by the original 3.1 and also the `find-proofs` and `update` computations. The only individual step that might fail to terminate is the invocation of the first-order prover in `find-proofs`. The only potentially infinite loop in the procedure is the loop in `find-proofs`, which can only be a problem if either a particular sentence is never removed from U (because an infinite number of proofs of it are found) or if an infinite number of sentences are added to U. If ϕ is syntactically stratified, an attempt to prove a sentence p will add to U only sentences w with $s(w) < s(p)$. Since $s(p)$ is finite and there are only finitely many sentences w generated by each $p \in U$ (since only finitely many proofs are found), it follows that only finitely many sentences are added to U over the course of the analysis.

The only way for the procedure to fail is if R fails to be empty when `update` concludes. But this implies that there is a circularity in the modal expressions being investigated, which is at odds with the assumption that ϕ be syntactically stratified. ■

Corollary 4.5 *Given unlimited computational resources, the value returned by Procedure 4.3 in response to a query q is $cl(\phi)(q)$.*
Proof. Suppose that

$$J' = \texttt{find-proofs}(\phi, q) - \texttt{find-relevant-proofs}(\phi, q)$$

so that J' is the difference between the relevant and "complete" justification sets. Then the eventual answer returned by Procedure 3.1 is $(\phi^{J'})^J(q)$ while that returned by Procedure 4.3 is $\phi^J(q)$. But $\phi^{J'}$ is an extension of ϕ, so the two answers are identical and the result follows easily from Proposition 3.4. ■

Proposition 5.2 *If P is an LP-stratified logic program, then $D(P)$, the declarative translation of P, is a semantically stratified truth assignment.*
Proof. This is a consequence of the facts that the only modal operator allowed in logic programs is $\neg\Box$ and that there is no other appearance of $\neg$.

Suppose that we view the sentences $\neg\Box p$ for various p as separate sentences in our language, so that truth values can be assigned to these sentences independent of the truth values assigned to the nonmodal sentences they contain. It now follows from the syntactic restrictions placed on logic programs that if M is a model of P, any other model that has more true positive literals than M does will also be a model of P.

It follows from this that the only way to show that a positive literal l is a consequence of a logic program is to use a rule of which l is the head; reasoning inductively on the length of the proof, we see that the only way to prove l is to use only a sequence of rules in the sense supported by logic programs, never using the contrapositive version to draw a conclusion about a literal appearing in the body of the rule.

Since the definition of LP-stratified guarantees that such a derivation can never produce a derivation in which l depends on $\neg\Box l$, it follows that no invocation of Procedure 4.3 can produce a loop in the relevant elements of J, so that the declarative translation of P will satisfy the conditions of Definition 4.6. ■

Proposition 5.3 *Using the functional truth values described, the result returned in response to a query q is the same as that computed by Chan's algorithm.*
Proof. Since Chan shows his algorithm to be correct [3], all we need do is to show that our approach is correct as well. This is once again a consequence of the syntactic restrictions on logic programs.

Suppose that we are responding to a query $g(x)$ that does not appear in the head of any rule. Since logic programs cannot contain sentences of the form

$$g(a) \vee g(b)$$

for constants a and b, it follows that the correct truth value can be assigned to $g(x)$ simply by looking for bindings σ such that $g(x)|_\sigma$ appears in the database and then returning a list of these bindings. This is well known and is simply a rephrasing of the usual argument that PROLOG responds correctly to the query $g(x)$, treating it as a Skolemized equivalent of the query $\exists x.g(x)$.

Given that our functional bilattice responds correctly to database lookups, we can use an inductive argument to show that it will respond correctly to inferential queries as well. The induction is on the length of the proof of the query being considered. ■

Proposition 5.4 *Suppose that we have a database ϕ and a query q. Then the value computed by Procedure 4.3 in response to the query $X(q)$ will be the same as the value computed by Ginsberg's algorithm for dealing with floundered subgoals [20].*

Proof. This is immediate: The semantics of the X operator exactly match the approximation proposed in [20], and we know from earlier results that Procedure 4.3 will give an accurate response to the query $X(q)$. ■

References

[1] P. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272, 1987.

[2] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–142. Morgan Kaufmann, 1987.

[3] D. Chan. Constructive negation based on the completed database. In *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, pages 111–125, 1988.

[4] D. Chan. An extension of constructive negation and its application in coroutining. In *Logic Programming: Proceedings of the Sixth International Conference and Symposium*, 1989.

[5] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

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

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

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

[9] M. Drummond and J. Bresina. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 138–144, 1990.

[10] J. J. Elgot-Drapkin. *Step-Logic: Reasoning Situated in Time.* PhD thesis, University of Maryland, College Park, MD, 1990.

[11] J. J. Elgot-Drapkin and D. Perlis. Reasoning in time I: Basic concepts. *J. Expt. Theor. Artif. Intell.*, 2:75–98, 1990.

[12] D. Etherington and J. Crawford. Toward efficient default reasoning. Unpublished manuscript, 1992.

[13] M. C. Fitting. Logic programming on a topological bilattice. *Fundamenta Informatica*, 11:209–218, 1988.

[14] M. Gelfond. On stratified autoepistemic theories. Technical report, University of Texas at El Paso, 1986.

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

[16] M. L. Ginsberg. A circumscriptive theorem prover. *Artificial Intelligence*, 39:209–230, 1989.

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

[18] M. L. Ginsberg. Computational considerations in reasoning about action. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, Boston, MA, 1991.

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

[20] M. L. Ginsberg. Negative subgoals with free variables. *Journal of Logic Programming*, 11:271–293, 1991.

[21] M. L. Ginsberg. User's guide to the MVL system. Technical report, University of Oregon, 1993.

[22] M. L. Ginsberg. Approximate planning. *Artificial Intelligence*, 1995.

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

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

[25] L. P. Kaelbling. Goals as parallel program specifications. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 60–65, 1988.

[26] K. Konolige. On the relation between default theories and autoepistemic logic. *Artificial Intelligence*, 35:343–382, 1988.

[27] K. Konolige. On the relation between default theories and autoepistemic logic (erratum). *Artificial Intelligence*, 41:115, 1990.

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

[29] J. McCarthy. Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[30] J. McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[31] R. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–94, 1985.

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

[33] N. J. Nilsson. Action networks. Technical report, Stanford University, 1988.

[34] D. Perlis. Non-monotonicity and real-time reasoning. In *Proceedings 1984 Non-monotonic Reasoning Workshop*, pages 363–372, New Paltz, NY, 1984. American Association for Artificial Intelligence.

[35] T. Przymusinski. On the declarative semantics of stratified deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, 1987.

[36] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[37] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, Boston, 1991.

[38] R. Reiter and G. Criscuolo. On interacting defaults. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 270–276, 1981.

[39] S. J. Rosenschein and L. P. Kaelbling. The synthesis of machines with provable epistemic properties. In J. Y. Halpern, editor, *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 83–98, Los Altos, CA, 1986. Morgan Kaufmann.

[40] M. J. Schoppers. Universal plans for reactive robots in unpredictable domains. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1039–1046, 1987.

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

[42] B. Selman and H. Kautz. Knowledge compilation using Horn approximations. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 904–909, 1991.

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

[44] A. Van Gelder. The alternating fixpoint of logic programs with negation: Extended abstract. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, 1989.

[45] A. Van Gelder. Negation as failure using tight derivations for general logic programs. *J. Logic Program.*, 6:109–133, 1989.