A Circumscriptive Theorem Prover

Matthew L. Ginsberg

Computer Science Department
Stanford University
Stanford, California 94305

A Circumscriptive Theorem Prover

Abstract

In MVL, a generalization of first-order logic was introduced that led to the development of an effective theorem prover for some simple sorts of default reasoning. In this paper, we show that these ideas can also be used to construct a theorem prover for a wide class of circumscriptive theories.

The ideas to be discussed have been implemented, and the resulting system has been applied to the canonical birds flying example, to a non-separable circumscription [9], and to the Yale shooting problem. In all of these cases, the implementation returns the circumscriptively correct answer.

1 Introduction

Circumscription [11, 10] is probably the most thoroughly investigated of all of the approaches to nonmonotonic reasoning. Unfortunately, these investigations have not led to the development of effective algorithms for determining whether or not some potential conclusion follows from the circumscription axiom. There have been several attempts at this [9, 12], but none has been completely satisfactory.

This paper begins to address these difficulties. The approach we will be taking is a result of combining a variety of ideas relating to nonmonotonic inference:

1. The formalization of the connection between circumscription and minimal models. This appears to be due to Lifschitz [9], and is also present in McCarthy's original paper [10] and recent work of Shoham's [17].

2. A description of minimal models in terms of specific formulas describing them. This idea appears in Gelfond et. al.'s idea of a *free for negation* sentence [4], and in the concept of a *prime implicant* discussed by Reiter and de Kleer [15].

3. The observation that an assumption-based truth maintenance system (ATMS) [2] can be used to generate the formulae mentioned above. This observation appears in both [15] and MVL.

4. The construction of a backward-chaining ATMS and the implementation of a backward-chaining circumscriptive theorem prover using it. Construction of a backward-chaining ATMS, and examples of its use, are discussed in MVL. Przymusinski [12] has also presented an algorithm for computing circumscription, although his approach is different and the algorithm he develops appears to be substantially less efficient than the one we will propose.

The subsequent sections of this paper consider each of these points in turn. Since the implementation that is the subject of the paper is the result of a synthesis of a variety of ideas, each section of the paper will be devoted, in part, to a description of previous appearances of these ideas elsewhere in the literature.

In the next section, Section 2, we discuss the connection between the circumscription axiom and constructions involving minimal models.

In Section 3, we go on to show that these minimal models can be described in terms of a notion we will refer to as *confirmation.* Loosely speaking, a sentence is confirmed if it would follow from the closed world assumption [13] applied to some predicate. We will discuss circumscription in terms of sentences whose negations are *not* confirmed; Gelfond et. al., considering similar problems, refer to such sentences as *free for negation.*

Next, in Section 4, we describe the relationship between ATMS's and confirmation. The description we will give of circumscription involves the idea of a "weakest" confirming sentence; this is closely related to the notion of a *prime implicant* appearing in [15].

The implementation itself is discussed in Section 5. Here, we show that the notion of a *bilattice* MVL can be used to construct a backward-chaining ATMS of the sort needed by a backward-chaining circumscriptive theorem prover. The implementation is compared to Przymusinski's, and a variety of examples are presented in Section 6.

We conclude in Section 7 by discussing the limitations of our approach, and possible ways around these limitations.

The work in this paper is a natural outgrowth of ideas in MVL. It is intended, however, that this paper be principally self-contained, and results that depend crucially on the earlier work have been relegated to an appendix.

2 Circumscription and models

As remarked in the introduction, the description of circumscription that we will be using is one based on models, as opposed to the usual circumscription axiom appearing in [11].

The key result we need is this one, Proposition 1 of [9]:

Proposition 2.1 *Let T be a set of sentences, and p a predicate. Now define a partial order on the models of T, writing $M_1 \leq M_2$, if the following conditions hold:*

1. *M_1 and M_2 have the same universe and agree in their interpretations of constant symbols.*

2. *The extent of p in M_1 is no greater than the extent of p in M_2.*

Now if q is some sentence, then q is a consequence of circumscribing p in T while allowing all other predicates to vary if and only if q holds in all $\leq$-minimal models of T.

Similar results holds for more complex circumscriptions. If, for example, we wish to minimize several predicates in parallel, we need merely replace the partial order appearing in the proposition with one that defines $M_1 \leq M_2$ to mean that for each p_i being minimized, the extent of p_i in M_1 is no larger than the extent in M_2. Lifschitz goes on in [9] to discuss the use of other partial orders corresponding to prioritized circumscription or circumscription where not all predicates are allowed to vary. We will discuss these issues in Section 7.

We will have use for a slightly different form of Proposition 2.1. We begin by introducing some notation.

Suppose that D is some set of defaults. One might, for example, take D to be the set of all propositions of the form $\neg p(x)$, where p is a predicate being circumscribed and x is an instantiation for p's argument.[1] Given two models M_1 and M_2, we will write $M_1 \leq_D M_2$ just in case the set of elements of D that hold in M_1 is a subset of the set of elements of D that hold in M_2. Given a collection of models, we will refer to the ones that are maximal under the partial order $\leq_D$ as *D-maximal* elements of the collection. The following result should now be clear:

Corollary 2.2 *Let T be a set of sentences without function symbols, such that T includes domain closure and uniqueness of names assumptions.*[2] *Let p be a predicate, and let D be the set of all propositions of the form $\neg p(x)$, where x is a ground instantiation of p's argument. Now for any sentence q, q is a consequence of circumscribing p in T while allowing all other predicates to vary if and only if q holds in all D-maximal models of T.*

The approach we will take will be to work directly from the sets T and D, as opposed to working from the circumscription axiom itself.

Before proceeding, however, we should spend some time discussing the corollary. The assumptions made regarding T are essentially those needed to ensure both that we can enumerate all possible instantiations of p using constant symbols appearing in the original theory, and that these instantiations will be nonequivalent; these are quite strong assumptions.[3] It appears that both Corollary 2.2 and our implementation based on it will be valid in somewhat wider circumstances; this issue is currently under investigation by Gelfond and Lifschitz [3].

Even in the general case where the corollary may fail, however, the argument can be made that an algorithm to determine whether or not some sentence q holds in all D-maximal models of T is of comparable interest to one that determines whether or not q holds after circumscribing p in T: Recall that the original motivation for the circumscription axiom was to help characterize precisely the notion of predicate minimization appearing in the corollary. Shoham, for example, suggests in [17] that nonmonotonic theories should proceed by taking as primitive a partial order on models, with axioms such as that given by circumscription being secondary.

---

[1] The results we will present will generally require that D be finite. This will be true if the Herbrand universe of our theory is finite; it is not clear to what extent our results generalize to the case of non-finite D.

[2] Essentially, this assumption means that we have some finite set $\{x_1, \ldots, x_n\}$ and axioms enabling us to conclude that any object in our domain is equal to one and only one of the x_i.

[3] I am indebted to Vladimir Lifschitz to providing me with a sharp formulation of them (personal communication).

3 Confirmation

Technically, the reason we will work with Corollary 2.2 instead of the circumscription axiom is that it is possible to recast the corollary in a somewhat more useful form. We need the following definition:

Definition 3.1 *Let D be a set of sentences. We will say that a sentence p is* dnf with respect to D *if p is of the form*

$$\bigvee_i \bigwedge_j d_{ij}$$

for some collection of $d_{ij} \in D$. In other words, p is the disjunction of conjunctions of elements of D.

Now also fix a set T. Then we will say that some sentence q is confirmed by p for D and T *if the following conditions hold:*

1. *$T \cup \{p\}$ is satisfiable,*
2. *$T \cup \{p\} \models q$, and*
3. *p is dnf with respect to D.*

If no ambiguity is possible, we will simply say that q is confirmed by p*. If there is no p that confirms q, we will say that q is* unconfirmed*. If p confirms the negation of q, we will say that p* disconfirms q*.*

The reason the above definitions are useful is because of the following result:

Proposition 3.2 *Let T and D be sets of sentences, and q a sentence. Then q holds in all D-maximal models of T if only if there is some p that confirms q such that $\neg p$ is unconfirmed.*

Proof. The proof is in Appendix 1.

It is clear from this result and Corollary 2.2 that, given information about confirmation, we can determine whether or not some query q follows from a given circumscription.

Before proceeding, however, we present two examples of this notion. The first is the usual one involving birds flying:

$$b(T)$$
$$o(F)$$
$$\forall x.b(x) \wedge \neg a(x) \rightarrow f(x)$$
$$\forall x.o(x) \rightarrow b(x) \wedge \neg f(x)$$

We are using b for bird and o for ostrich, and T for Tweety and F for Fred. Tweety is a bird, and Fred is an ostrich. Birds that are not abnormal (a) can fly (f), and ostriches are birds that don't fly. We are now interested in the results of circumscribing a in this theory.

There is a unique a-minimal model of the above axioms, in which a holds for F but for no other object. It is not hard to see that $\neg a(T)$ is confirmed by $\neg a(T)$ itself, but that $\neg a(F)$ is unconfirmed, since it is inconsistent with the original theory. Similarly, $a(T)$ is unconfirmed, while $a(F)$ is trivially confirmed, since it is a consequence of the above axioms.

Next, we see that $f(T)$ is confirmed by $\neg a(T)$; since $a(T)$ is unconfirmed, it follows that $f(T)$ is a consequence of the circumscription: Tweety flies. Since $f(F)$ is unconfirmed, we cannot conclude that Fred does.

Here is a slightly more interesting example, due to Reiter [14]:

$$r(N)$$
$$q(N)$$
$$\forall x. r(x) \wedge \neg a(P, x) \rightarrow h(x)$$
$$\forall x. q(x) \wedge \neg a(R, x) \rightarrow \neg h(x)$$

Nixon is a Republican and a Quaker. Any Republican who is not politically abnormal is a hawk; any Quaker who is not religiously abnormal is not a hawk. As before, we circumscribe a.

Here, there are two a-minimal models. In one we have $a(P, N)$; in the other, we have $a(R, N)$. Because of the interaction between the conflicting default rules, we see that $a(P, N)$ is confirmed by $\neg a(R, N)$, and $a(R, N)$ is confirmed by $\neg a(P, N)$. As before, $\neg a(P, N)$ and $\neg a(R, N)$ each confirm themselves.

Now suppose we try to derive $h(N)$: Is Nixon a hawk? Although $h(N)$ is confirmed by $\neg a(P, N)$, the negation of this statement is not unconfirmed. It follows that $h(N)$ is not entailed by the circumscription. In Reiter's terms, the observation that the confirming fact has a confirmed negation is an indication that although the query $h(N)$ holds in *some* extension of the default theory, it does not hold in all of them. In a similar way, we see that $\neg h(N)$ also does not follow from the circumscription axiom in this case. This example is closely related to the disjunctive example we will consider in Section 6.3.

Our description of circumscription in terms of confirmation is closely related to a description developed by Gelfond et. al. in [4]. Their theorem 3 states that a sentence q follows from a circumscription if and only there exists a p consistent with T such that $T \models p \rightarrow q$ and the negation of p is *free for negation* (in our terms, unconfirmed). A sentence r is *not* free for negation (i.e., confirmed) if and only if there exists a conjunction c of elements of D that is consistent with T and such that $T \models c \rightarrow r$.[4]

The principal differences between our description and that appearing in the earlier paper are the following:

1. The description in terms of confirmation is somewhat more compact, leading to a simple description of both the relationship between p and q in Proposition 3.2 and of the constraints on q itself.

[4]We have modified the notation used in [4] somewhat in order to make the comparison clearer. I am indebted to Vladimir Lifschitz for drawing my attention to this.

2. As we will see in Section 4, the appearance of the disjunction in our definition of confirmation allows us to introduce the notion of a "weakest" confirming sentence. Construction of this sentence is crucial to the computational efficiency of our implementation.

3. The work in [4] discusses the possibility of not varying all predicates when the circumscription is performed. We will return to this point in Section 7.1.

4 Confirmation and truth maintenance

In order to use Proposition 3.2 effectively, we need some way to determine the various sentences that confirm some query q.

Suppose we consider the collection of all sentences that confirm q. Now it is fairly clear that the disjunction of all of these sentences also confirms q, and that the negation of this disjunction will be unconfirmed if and only if q satisfies the conditions of Proposition 3.2. Thus we can determine whether or not q holds in all D-maximal models of T by considering only the *weakest* of the sentences that confirm it.

This problem has in fact already been addressed in the AI literature. In [15], Reiter and de Kleer show that assumption-based truth maintenance systems (ATMS's) perform just this sort of calculation; similar remarks can be found in MVL.

Essentially, an ATMS takes a proposition such as q and determines the 'environment' in which q holds. One possible representation for this environment is as a list of contexts, where each context is described by a list of assumptions that must hold in it. If we take the elements of D as our possible assumptions, we see that the ATMS environments correspond to our dnf formulae. Since the ATMS is looking for a minimally specific environment in which q holds, we can think of it as looking for a weakest sentence p that confirms q. Similarly, the ATMS label assigned to $\neg p$ will tell us whether or not $\neg p$ is unconfirmed. A more precise statement of this idea is the content of Theorem 7 of [15] and Theorem A.1 of Appendix 2.

5 Implementation

There are some subtleties involved in actually implementing these ideas. Firstly, a (presumably backward-chaining) circumscriptive theorem prover will rely on a backward-chaining ATMS; de Kleer's published work [2] is based on forward-chaining methods. In addition, de Kleer's work only describes an ATMS for propositional calculus; a circumscriptive theorem prover will need to work with a fully first-order version. We now turn to the problem of constructing an ATMS with these properties.

5.1 Backward chaining

Construction of a backward chaining ATMS is discussed in MVL.[5] The essential idea is to construct a bilattice MVL that corresponds to deKleer's construction, and to then use the general-purpose algorithms also described in MVL to produce a backward chainer that uses this bilattice.

The ATMS bilattice is constructed using the fact that the environments described in the last section can be partially ordered by generality.[6] It is clear that the contexts themselves, viewed simply as subsets of the set of all possible assumptions, can be partially ordered by set inclusion. Thus if $c = c_1 \wedge \ldots \wedge c_m$ and $d = d_1 \wedge \ldots \wedge d_n$ are two contexts, we can take $c \leq d$ to mean that the set of c_i's contains the set of d_j's as a subset. In other words, $c \leq d$ if the context c is *less general* than the context d.

Using this partial order, we can construct a partial order on the environments themselves, saying that one environment e_1 is less general than another e_2 if *every* context in e_1 is less general than *some* context in e_2. If every context in e_1 contains some context in e_2 as a subset, then the environment e_1 is less general than the environment e_2.

The points in bilattices corresponding to ATMS's consist of pairs of environments $[\, e_1 \,.\, e_2 \,]$. If a sentence p has truth value $[\, e_1 \,.\, e_2 \,]$, this means that e_1 is the most general environment in which p is known to hold, and e_2 is the most general environment in which $\neg p$ is known to hold.

Theorem A.1 of Appendix 2 now tells us that closure in this bilattice corresponds to the evaluation of these most general environments.[7]

5.2 First-order ATMS's

Next, we discuss the construction of a first-order ATMS, as opposed to a propositional one. In general, we need to consider the possibility that the sentences appearing in the various ATMS contexts be quantified in some way.

We will assume that this quantification can be handled implicitly, as in PROLOG [1]. Existential quantification will be handled via Skolemization, and universal quantification will be handled by assuming that any free variables appearing in the database are universally quantified.[8]

We will now construct contexts in a fashion quite similar to that of the last section; from these contexts, we construct environments exactly as before. It follows that in order to extend our propositional ATMS to a first-order one, we need to extend the partial order we previously defined for contexts. Recall that this partial order defined a conjunctive context

[5]Reiter and de Kleer describe this as the "interpreted approach" to truth maintenance in [15]. No algorithm is presented, however.

[6]The following discussion is somewhat abbreviated. A more complete version can be found in MVL.

[7]Results from MVL that only make sense in light of this earlier work have been confined to Appendix 2 in the interests of maintaining the continuity of the main exposition.

[8]Ray Reiter has pointed out to me that not all first-order theories can be written in this fashion. It is not clear how difficult it will be to generalize the following construction to handle these situations.

c to be less general than a context d just in case every sentence in c was also in d.

In the first-order case, we need to modify the definition only slightly, saying that a context c is less general than another context d if and only if, for each sentence c_i appearing in c, there is some sentence d_j appearing in d such that d_j is an *instantiation* of c_i.

Thus, for example, the context consisting of the single sentence $p(x)$ is less general than the context consisting of the sentence $p(a)$, where x is a variable and a is a constant. This is as it should be, since $p(a)$ surely holds if $\forall x.p(x)$ does. (Recall that we are using $p(x)$ as shorthand for the quantified statement $\forall x.p(x)$.)

In what follows, we will represent contexts using propositions and binding lists, so that the context consisting of the sentence $p(a)$ might well be represented as

$$(p(x)\ .\ \{x = a\}).$$

The reason we do this is that we will frequently have assigned designators to the universally quantified propositions, and this representation is slightly more compact than restating the proposition involved. Thus an element of any particular context will generally have the form

$$(p\ .\ \sigma),$$

where p is an assumption (i.e., an element of D) and σ is a binding list indicating which variables in p are bound in the context. If D consists of the single sentence schema $\neg\mathsf{ab}(x)$, a context depending on $\neg\mathsf{ab}(a)$ and $\neg\mathsf{ab}(b)$ would be written as:

$$\{(\neg\mathsf{ab}(x)\ .\ \{x = a\}), (\neg\mathsf{ab}(x)\ .\ \{x = b\})\}.$$

In the LISP-like notation to be used in the next section, we would write:

$$((\neg\mathsf{ab}(x)\ .\ \{x = a\})(\neg\mathsf{ab}(x)\ .\ \{x = b\})).$$

Environments will be written as lists of contexts.

Once again, a formal description of inference using this bilattice appears in Appendix 2. It follows fairly easily from the result appearing there that if the value assigned to q after computing the bilattice closure is $[\ e\ .\ f\ ]$, where $e = \{c_1, \ldots, c_m\}$ and

$$c_i = \{(d_{ij}\ .\ \sigma_{ij})\},$$

then the weakest sentence confirming q is

$$\bigvee_i \bigwedge_j d_{ij}|_{\sigma_{ij}}.$$

5.3 Comparison to existing work

The construction of the weakest sentence confirming a query q results in our implementation having computational properties very different from those described by Przymusinski in [12].

Przymusinski's algorithm is constructed from the work we have already cited in [4]. Essentially, the algorithm requires that we examine the proof tree for the query, and for each conjunctive sentence p that confirms it, we check to see if $\neg p$ is unconfirmed. The algorithm is described in terms of what Przymusinski calls *MILO-resolution*, a variant of ordered resolution.

We see that the algorithm in [12] will need to determine if $\neg p$ is confirmed for every p confirming the query q, as opposed to the single check involved in applying Proposition 3.2 to the weakest sentence confirming q. In addition to the obvious computational savings to which this leads, it may be the case that some of the sentences confirming q subsume others. In this case, the ATMS construction we have proposed will avoid examining certain sentences unnecessarily.

Finally, we note that Przymusinski's algorithm is tied to a specific implementation of the resolution principle. This is to be contrasted with the work we have cited with regard to ATMS's [2, 7, 15], which allows the ATMS to construct its most general environments through the use of an arbitrary first-order theorem prover. As advances are made in the control of inference ([18] is typical), we can expect the approach we are suggesting to be capable of incorporating these advances more easily than one based on a specific resolution algorithm.

5.4 The multivalued theorem prover

In the next section, we will see examples of the prover being used to determine, in a variety of situations, whether or not some formula is a consequence of the circumscription axiom. Because the prover uses the multivalued description of ATMS's described in Sections 5.1 and 5.2, the implementation itself rests on the multivalued theorem prover described in MVL.

A complete discussion of the multivalued prover is outside the scope of this paper. Briefly, the implementation works via successive calls to a conventional first-order PROLOG-like theorem prover which has been modified to return a list of the database facts used in any particular proof. These facts are then accumulated using general bilattice-based constructions to produce the ATMS labelling for any particular query.

Because of the nature of the implementation, the prover will work efficiently to find legitimately new proofs of the proposition being considered, pruning its search space to avoid examining potential proofs that do not contain new justification information. Nevertheless, the multivalued prover must wait for the first-order engine to terminate before proceeding; since first-order inference is not decidable, this may never happen. Of course, this is to be expected, since the problem of circumscriptive entailment is itself nonsemidecidable.

6 Examples

In this section, we present four examples of the implemented system at work: the usual birds flying example, a non-separable circumscription, a simple example involving disjunction, and the Yale shooting problem. All of the 'output' shown is as actually produced by the program, except for trivial textual modifications. (For example, the database is maintained in disjunctive normal form, but is displayed below using a PROLOG-like representation.)

6.1 Birds flying

Here is the database for the usual birds flying example:

```
Bird(Tweety).
Ostrich(Fred).
Flies(x) :- Bird(x),Not(Ab(x)).
Bird(x) :- Ostrich(x).
Not(Flies(x)) :- Ostrich(x).
Not(Ab(x)).                          P4
```

As in Section 3, we are told that Tweety is a bird, and that Fred is an ostrich. Birds fly unless they are abnormal; ostriches are birds that don't fly. By default, nothing is abnormal. The `P4` labelling the statement that nothing is abnormal serves both to indicate that this is a default rule, and to give the ATMS a label for this rule.

We now ask the inference engine to find something that flies by giving it the query `Flies(x)`. Here is the result:

```
Flies(x)?
     Invoking multivalued prover on Flies(x).
     Value returned is:
        binding list: {x = Tweety}
        truth value:  (((P4 . {x = Tweety})))
     Checking circumscriptive context for truth value
        (((P4 . {x = Tweety}))).
     Checking confirmation for Not(Ab(Tweety)).
     Negation is unconfirmed.
Success!  x = Tweety.
```

The prover begins by invoking the multivalued prover on the query `Flies(x)` MVL. The prover succeeds, since it can find a proof of `Flies(x)` for x bound to Tweety. The truth value returned contains justification information. In this case, the proof that Tweety can fly used the proposition `P4` with x bound to Tweety.

As in Section 3, this means that `Flies(Tweety)` is confirmed by `Not(Ab(Tweety))`, so the prover next checks to see if it can find some confirmation for the negation of this statement. Again as in Section 3, since `Not(Not(Ab(Tweety)))` is unconfirmed, the prover succeeds and returns the answer $x =$ Tweety.

Alternatively, we can look for something that *doesn't* fly:

```
Not(Flies(x))?
     Invoking multivalued prover on Not(Flies(x)).
     Value returned is:
        binding list: {x = Fred}
        truth value:  TRUE
     Checking circumscriptive context for truth value TRUE.
     Negation is unconfirmed.
Success!  x = Fred.
```

The prover is able to show that Fred cannot fly without using any assumptions at all (recall that only the default rule about abnormality is a possible assumption; the other database facts are accepted unconditionally). Since TRUE cannot be disconfirmed, the prover informs us that Fred cannot fly.

6.2 A non-separable circumscription

We next turn to an example of a non-separable [9] circumscription. This example is due to Vladimir Lifschitz:

```
     P(a).
     P(f(x)) :- P(x).
     Not(P(x)).                P9
```

We are told that a satisfies the predicate P, and that $f(x)$ does if x does. We also are told that P is false by default, so that P is being circumscribed in this example.[9]

We first ask the system about $P(a)$ and $P(b)$:

```
P(b)?
     Invoking multivalued prover on P(b).
Fails.

P(a)?
     Invoking multivalued prover on P(a).
     Value returned is:
        binding list: {}
        truth value:  TRUE
     Checking circumscriptive context for truth value TRUE.
     Negation is unconfirmed.
Success!
```

[9]Recall [9] that a domain description is *separable* if and only if it is constructed from formulae that either contain no positive instance of P, or are of the form $P(x) \to Q(x)$, where Q is a predicate not involving P. Since the rule $P(x) \to P(f(x))$ is not of this form, this example is a non-separable one. The demonstration that our methods can deal with non-separable circumscriptions is of interest because the work in [9] is generally restricted to the separable case.

The query $P(b)$ fails immediately, since there is no possible proof of $P(b)$ from facts in the database.[10] The proof of $P(a)$ is similar to the conclusion in the last section that Fred could not fly.

Here are the results of the queries `Not(P(a))` and `Not(P(b))`:

```
Not(P(a))?
     Invoking multivalued prover on Not(P(a)).
     Value returned is:
        binding list: {}
        truth value:  (((P9 . {x = a})))
     Checking circumscriptive context for truth value (((P9 . {x = a}))).
     Checking confirmation for Not(P(a)).
     Negation confirmed based on truth value TRUE.
Fails.

Not(P(b))?
        Invoking multivalued prover on Not(P(b)).
        Value returned is:
           binding list: {}
           truth value:  (((P9 . {x = b)))
        Checking circumscriptive context for truth value
           (((P9 . {x = b))).
        Checking confirmation for Not(P(b)).
        Negation is unconfirmed.
Success!
```

When asked to prove `Not(P(a))`, the system notes that it can do this using `P9`, with x bound to a. Thus `Not(P(a))` is apparently confirmed by `Not(P(a))` itself. Unfortunately, the negation of this sentence is disconfirmed trivially, since $P(a)$ appears in the database. Thus the query `Not(P(a))` fails.

The query `Not(P(b))` is different. Here, the confirming statement `Not(P(b))` has a negation that is unconfirmed, so the prover succeeds.

6.3 A disjunctive example

We next turn to an example from [10]:

```
     Block(a) :- Not(Block(b)).
     Not(Block(x)).                   P15
```

[10]Note that we assume $b \neq f(a)$. Although T in this case has function symbols (violating the premises of Corollary 2.2), the results returned by the prover remain correct as long as applying the function always produces new constants. Note also that the assumption that $b \neq f(a)$ is made by the first-order prover itself, and is not implicit in the application of the algorithm we are using.

We are told that either a is a block or b is, and circumscribe the predicate `block`. The result of the circumscription should be that either a is the *only* block, or b is. It follows from this that

$$\neg[\mathtt{block}(a) \wedge \mathtt{block}(b)]$$

should be circumscriptively valid:

```
Not(And(Block(a),Block(b)))?
     Invoking multivalued prover on Not(And(Block(a),Block(b))).
     Value returned is:
        binding list: {}
        truth value:  (((P15 . {x = a})) ((P15 . {x = b})))
     Checking circumscriptive context for truth value
        (((P15 . {x = a})) ((P15 . {x = b}))).
     Checking confirmation for Or(Not(Block(a)),Not(Block(b))).
     Negation is unconfirmed.
Success!
```

This example is slightly more difficult than the preceding ones. The multivalued prover manages to prove the query by using either `P15` with x bound to a, or with x bound to b. Thus the query is confirmed by:

$$p \equiv \neg\mathtt{block}(a) \vee \neg\mathtt{block}(b).$$

The negation of p is $\mathtt{block}(a) \wedge \mathtt{block}(b)$, and this appears to itself be confirmed by `P15` applied to *both* of a and b (since applying `P15` to a allows us to conclude that b is a block, and similarly for applying it to b). Thus the negation of p is apparently confirmed by:

$$\neg\mathtt{block}(a) \wedge \neg\mathtt{block}(b).$$

This sentence is inconsistent with our theory, however, since we are assuming that either a or b is a block. Thus $\neg p$ is in fact unconfirmed, and the prover returns with success. This example is similar to the Reiter's Nixon-Quaker-Republican example discussed in Section 3.

6.4 The Yale shooting

Finally, we discuss the well known Yale shooting example from [8]:

```
     Not(Holds(alive,Do(shoot,s))) :- Holds(loaded,s).
     Holds(p,Do(a,s)) :- Holds(p,s), Not(Ab(a,p,s)).
     Holds(loaded,s0).
     Holds(alive,s0).
     Not(Ab(a,p,s)).                  P23
```

The example involves a gun and an intended victim (generally named Fred). If the gun is loaded in a state s, then Fred will not be alive in the state resulting from firing the gun. The second axiom is a frame axiom, telling us that a proposition p will hold after performing an action a in a state s if p held before performing the action, unless the triple (a, p, s) is abnormal. The gun is loaded and Fred is alive in the initial state s_0. Finally, actions are (by default) not abnormal.

The question is this: If we wait and then fire the gun, do we kill Fred? Surprisingly, the answer is no, since there are two different ways in which we might apply the default rules. In the first (the intended one), waiting has no effect, and the action of firing the gun is abnormal in the state `Do(wait,s0)` because Fred becomes not alive. The second possibility is one in which the waiting action is abnormal and the gun becomes mysteriously unloaded.

If we ask the circumscriptive theorem prover whether Fred is alive after we wait and fire the gun, here is the result:

```
Holds(alive,Do(shoot,Do(wait,s0)))?
      Invoking multivalued prover on Holds(alive,Do(shoot,Do(wait,s0))).
      Value returned is:
         binding list: {}
         truth value:  (((P23 . {a = wait, p = alive, s = s0})
                          (P23 . {a = shoot, p = alive, s = Do(wait,s0)})))
      Checking circumscriptive context for truth value
         (((P23 . {a = wait, p = alive, s = s0})
           (P23 . {a = shoot, p = alive, s = Do(Wait,s0)}))).
      Checking confirmation for
         And(Not(Ab(wait,alive,s0)),Not(Ab(shoot,alive,Do(wait,s0)))).
      Negation confirmed based on truth value
         (((P23 . {a = wait, p = loaded, s=s0}))).
Fails.
```

The system first notes that it can prove that Fred is alive simply by applying the frame axiom twice, first to conclude that he is alive after the waiting action, and then to conclude that he remains alive after the shooting action. Thus his being alive is confirmed by:

$$\neg \mathtt{ab}(\mathtt{wait}, \mathtt{alive}, s_0) \wedge \neg \mathtt{ab}(\mathtt{shoot}, \mathtt{alive}, \mathtt{do}(\mathtt{wait}, s_0)).$$

The negation of this sentence is confirmed by

$$\neg \mathtt{ab}(\mathtt{wait}, \mathtt{loaded}, s_0),$$

however. If `wait` is not abnormal in s_0, then the gun will be loaded after the waiting action, and Fred will necessarily be dead after the gun is fired. Since there is no proof of the negation of this sentence, the confirming sentence for the original query is itself disconfirmed, and the query does not follow from the circumscription.

Alternatively, we can try to show that Fred is *not* alive after the sequence of actions:

```
Not(Holds(alive,Do(shoot,Do(wait,s0))))?
     Invoking multivalued prover on
        Not(Holds(alive,Do(shoot,Do(wait,s0)))).
     Value returned is:
        binding list: {}
        truth value:  (((P23 . {a = wait, p = loaded, s = s0})))
     Checking circumscriptive context for truth value
        (((P23 . {a = wait, p = loaded, s = s0}))).
     Checking confirmation for Not(Ab(wait,loaded,s0)).
     Negation confirmed based on truth value
        (((P23 . {a = wait, p = alive, s = s0})
          (P23 . {a = shoot, p = alive, s = Do(wait,s0)}))).
Fails.
```

Now a proof is found indicating that Fred will not be alive after the gun is fired provided that waiting did not cause it to become unloaded. The negation of the confirming fact is $\mathtt{ab}(\mathtt{wait}, \mathtt{loaded}, s_0)$, but this follows from the conjunction

$$\neg\mathtt{ab}(\mathtt{wait}, \mathtt{alive}, s_0) \wedge \neg\mathtt{ab}(\mathtt{shoot}, \mathtt{alive}, \mathtt{do}(\mathtt{wait}, s_0)).$$

Since the negation of the above sentence cannot be proven, the confirmation of

$$\neg\mathtt{holds}(\mathtt{alive}, \mathtt{do}(\mathtt{shoot}, \mathtt{do}(\mathtt{wait}, s_0)))$$

is disconfirmed, and the prover fails.

In all of these examples, the prover returns the correct circumscriptive answer. In addition, the computational procedure used is reasonably efficient. The Yale shooting problem, for example, is solved in approximately one second on a Symbolics 3600.

7 Limitations and future work

In this section, we discuss some limitations on the approach we have proposed, and some suggestions for extensions of it.

7.1 Not varying all predicates

First, we have assumed throughout that all predicates not being circumscribed are varied when the circumscription is performed. This is required in order for Proposition 3.2 to be applicable.

As an example, consider the birds flying case. If all predicates are varied, we can actually conclude from the circumscription that Fred is the only ostrich, and this may not be an intended result. If we hold the `ostrich` predicate fixed, we avoid it.

We can no longer conclude that Tweety can fly, since Tweety might be an ostrich in this case. This is presumably acceptable; we *can* still conclude, however, that ostriches are the only nonflying birds. So there is still some power to the circumscription axiom even if not all predicates are varied.

One possible way around this problem is suggested by the work in [4]. Recall that in the definition of dnf formulae, we required that a dnf sentence p be constructed from negations of instantiations of the circumscribed predicate. Following ideas in [4], we can show that if we extend this definition to require that p be constructed from negations of instantiations of the circumscribed predicate, together with either instantiations of any non-varying predicate or negations of such instantiations, Proposition 3.2 can be applied to compute the result of a circumscription in which not every predicate is allowed to vary. It is clear that the methods of Section 5 can be used to construct a prover using the modified definition.

7.2 Equality

Another difficulty with the approach we have presented has to do with shortcomings of the first-order theorem prover underlying the implementation. Consider the following PROLOG fragment:

```
foo(x) :- not(x=a).
foo(a).
```

The query `foo(x)` should return success here without binding the variable x, since we know `foo(a)` and are able to derive `foo(x)` for $x \neq a$. Unfortunately, most PROLOG-type provers do not succeed here; the reason is that there are two different proof paths for the different instantiations of x, and the results cannot be combined in an effective fashion.

This problem can arise in subtle ways. Consider the birds flying database:

```
Bird(Tweety).
Ostrich(Fred).
Flies(x) :- Bird(x),Not(Ab(x)).
Bird(x) :- Ostrich(x).
Not(Flies(x)) :- Ostrich(x).
Not(Ab(x)).                        P4
```

Now the query `Flies(x):-Bird(x),Not(Ostrich(x))` (are ostriches the only nonflying birds?) in fact fails. It succeeds for x bound to Fred, since Fred is an ostrich. It succeeds for all other x, since they are not ostriches because of the circumscription axiom **P4**. But these two proofs cannot be combined to yield a proof of the universally quantified query.

This is a serious shortcoming, and reflects a problem with PROLOG (or, in the case of the actual implementation, with MRS [5, 16]). The only apparent solutions to it are either to substantially modify PROLOG or MRS, or to modify the ATMS bilattice being used by the circumscriptive theorem prover so that it includes instantiation information in the truth values. This approach is discussed in [6], and lies substantially outside the scope of the present paper.[11]

7.3 Prioritization

Finally, we discuss the issue of prioritized defaults. Lifschitz shows in [9] that by varying some predicates preferentially over others, it is possible to establish a prioritization among the predicates being minimized.

It should be possible to incorporate a version of this idea into the approach we have described. Proposition 3.2 cannot be modified in any obvious way, but by using this proposition twice, first with a high-priority set of defaults and then with defaults of a lower priority, essentially the same effect can be obtained. Another way to look at this is described in MVL, which discusses the possibility of introducing 'stratification' levels onto a bilattice. By splitting the ATMS of Section 5 into a hierarchy of stratification levels, a prioritization can be introduced onto the set of assumptions being considered.

Appendix 1: Proof of Proposition 3.2

Proposition 3.2. *Let T and D be sets of sentences, and q a sentence. Then q holds in all D-maximal models of T if and only if there is some p that confirms q such that $\neg p$ is unconfirmed.*

Proof. We begin by noting that the D-maximal models of T are clearly in 1-1 correspondence with maximal subsets of D consistent with T. It follows from this that given a D-maximal model M, we can associate with M a subset D_M of D given by:

$$D_M \equiv \{d \in D | d \text{ holds in } M\}.$$

We also associate with M a proposition p_M given by the conjunction of the elements of D_M. Since M is a model of T, it follows that $T \cup \{p_M\}$ is satisfiable; since M is D-maximal, it follows that

$$T \cup \{p_M\} \models \neg d$$

for any $d \in D - D_M$.

Now let p be the disjunction of all of the p_M for the various D-maximal models M. Since p holds in all of the D-maximal models, it is clear that $T \cup \{p\}$ is satisfiable. We also claim that $\neg p$ is unconfirmed.

Suppose that $\neg p$ were confirmed by some sentence r. Writing r as a disjunction, it is clear that $\neg p$ will in fact be confirmed by any consistent disjunct in r. Since there must be at least one such consistent disjunct, we can assume without loss of generality that r is in fact a conjunction of elements of D. Thus there exists some subset $E \subseteq D$ such that $T \cup E$

[11] Difficulties also arise in the treatment of equality because of the appearance of the unique names hypothesis in Corollary 2.2.

is consistent and $T \cup E \models \neg p$. But extending E to a maximal subset E' of D consistent with T gives us $T \cup E' \models \neg p$, a contradiction.

Next, we note that the D-maximal models of T are precisely the models of $T \cup \{p\}$. It follows from this that if q holds in every D-maximal model of T, then p confirms q.

For the converse, suppose that q does not hold in every D-maximal model of T, and specifically that q does not hold in one such model M. Now if r confirms q, we claim that p_M confirms $\neg r$.

To see this, suppose that $r = \vee r_i$. Now since $T \cup \{r\} \models q$, r must be false in M, so that each r_i must be false in M. But each r_i is a conjunction of elements of D, and we know from the D-maximality of M that, for any element d of D, either $T \cup \{p_M\} \models d$ or $T \cup \{p_M\} \models \neg d$. Since r_i is false in M, we must therefore have $T \cup \{p_M\} \models \neg r_i$, and thus $T \cup \{p_M\} \models \neg r$. It follows that p_M confirms $\neg r$, and the proposition is proved. $\square$

Appendix 2: Results from multivalued logic

The following results are included principally for completeness. In them, ϕ is a *truth assignment*, which is to say that it is a mapping that assigns a truth value to each sentence in our logical language. The partial order $\leq_k$ corresponds to completeness of information, so that $x \leq_k y$ for truth values x and y just in case y contains at least as much information as x does. Finally, cl is the multivalued analog of first-order deductive closure.

Theorem A.1. *Let D be the set of assumptions in our knowledge base, and suppose that ϕ is given by*

$$\phi(q) = \begin{cases} [\ (\{q\})\ .\ \text{nil}\ ], & \text{if } q \in D; \\ u, & \text{otherwise.} \end{cases}$$

Then if $\{q_1, \ldots, q_m\} \subseteq D$ and p is an arbitrary sentence,

$$[\ (\{q_1, \ldots, q_m\})\ .\ \text{nil}\ ] \leq_k \text{cl}(\phi)(p)$$

if and only if the q_i's entail p. $\square$

Intuitively, the theorem begins by labelling each sentence in D as 'self-justified'; for any assumption q, q holds in any context in which q holds. The conclusion tells us that if some collection of these q's justifies p, in that $q_1, \ldots, q_m \models p$, then the truth value of p in the closure will reflect this in that

$$[\ (\{q_1, \ldots, q_m\})\ .\ \text{nil}\ ] \leq_k \text{cl}(\phi)(p).$$

The justification for p in the closure requires no more premises than a justification depending simply on the q_i's.

Corollary A.2. *Let D be the set of assumptions in our knowledge base, and for some assumption $q \in D$, denote by $q|_\sigma$ the result of applying the instantiation σ to q. Now suppose that ϕ is given by*

$$\phi(q|_\sigma) = \begin{cases} [\ (\{(q\ .\ \sigma)\})\ .\ \text{nil}\ ], & \text{if } q \in D; \\ u, & \text{otherwise.} \end{cases}$$

Then if $\{q_1, \ldots, q_m\} \subseteq D$ *and* p *is an arbitrary sentence,*

$$[\ (\{q_1|_{\sigma_1}, \ldots, q_m|_{\sigma_m}\})\ .\ \mathrm{nil}\ ] \leq_k \mathrm{cl}(\phi)(p)$$

if and only if the $q_i|_{\sigma_i}$*'s entail* p. □

Proof. Apply Theorem A.1 with D' equal to the set of instantiations of D. □

Acknowledgement

This work has been supported by DARPA under grant number N00039-86-C-0033, by ONR under grant number N00014-81-K-0004, and by NSF under grant number DCR-8620059. I would like to thank the staff and students of the Logic Group at Stanford for providing so stimulating an environment in which to work, and would also like to thank Johan de Kleer, Benjamin Grosof, John McCarthy, Karen Myers, Ray Reiter and David Smith for many useful discussions and suggestions. I am especially grateful to Vladimir Lifschitz for the assistance he has given me in sharpening the results of Section 2, and in directing me to Gelfond's paper [4].

References

[1] W. F. Clocksin and C. S. Mellish. *Programming in Prolog.* Springer-Verlag, Berlin, 1981.

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

[3] M. Gelfond and V. Lifschitz. Compiling circumscriptive theories into logic programs. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.

[4] M. Gelfond, H. Przymusinska, and T. Przymusinski. The extended closed world assumption and its relationship to parallel circumscription. In *Proceedings of ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 133–139, 1986.

[5] M. R. Genesereth. An overview of meta-level architecture. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 119–124, 1983.

[6] M. L. Ginsberg. A new approach to implicit quantification. Technical Report 87-7, Logic Group, Stanford University, 1987.

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

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

[9] V. Lifschitz. Computing circumscription. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 121–127, 1985.

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

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

[12] T. C. Przymusinski. Query-answering in circumscriptive and closed-world theories. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 186–190, 1986.

[13] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 119–140. Plenum, New York, 1978.

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

[15] R. Reiter and J. de Kleer. Foundations of assumption-based truth maintenance systems: Preliminary report. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 183–188, 1987.

[16] S. Russell. The compleat guide to MRS. Technical Report STAN-CS-85-1080, Stanford University, June 1985.

[17] Y. Shoham. A semantical approach to nonmonotonic logics. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 388–393, 1987.

[18] D. E. Smith and M. R. Genesereth. Ordering conjunctive queries. *Artificial Intelligence*, 26(2):171–215, 1985.