A Logic Programming Formalization for Circumscription

Masahiko Arai

Dept. of Information and Communication Technology, Tokai University
317 Nishino, Numazu, Shizuoka, 410-0395, Japan
arai@wing.ncc.u-tokai.ac.jp

Abstract. This paper proposes a practical way for circumscription. The meanings of the practical way are 1) a goal-oriented prover is given for solving circumscription problems. A feature of the prover is that the priority and the variable predicates are suggested so as to prove the intended results whereas usual methods prove queries by giving the priority and the variable predicates in advance. This prover is an application of the one for DNF formulae in the object system T and is the set of meta-rules of a meta-predicate which represents a clause. Circumscription formulae are represented as the meta-rules. The SLD-resolution procedures for the meta-system are given and a circumscription problem is solved by showing an empty node in the SLD-tree if semi-decidable. 2) Based on the prover, formulae satisfying circumscription formulae are given for predicates with functions. Practically circumscription is applied to prove the negation of a predicate p. For the formulae given above to be false, the condition is needed that T made p false is consistent with T. 3) Variable predicates are generalized so as to prove T made p false. By the generalization it is shown that circumscription problems are practically solved by giving a model to show the consistency for the undecidable cases. By encapsulating the consistency check as an oracle a practical logic programming for circumscription is given for queries without universal quantifiers. The oracle for queries with universal quantifiers is also given.

1 Introduction

Nonmonotonic logics are important for inferences based on defeasible assumptions in AI [1]. Major nonmonotonic logics are circumscription [7], default logic, and autoepistemic logic. These logics deal with the concepts of plausibility and normality and the proof systems are complicated. As seen in the sequent calculi for nonmonotonic logics these proof systems include both the proof and disproof procedures [3, 5]. Default and autoepistemic logics need the disproof-procedures to generate the extensions and the expansions, respectively. Regarding the provers circumscription is attractive from the following two reasons. One is that in propositional case the complexity of circumscription is Π^p_2 of polynomial hierarchy whereas that of autoepistemic logic is Π^p_3-complete for skeptical reasoning [4]. The other is that circumscription is formalized in classical logic by a formula in second-order logic, and resolution procedures can be applied to the provers. The proof-

© A. Gelbukh, R. Monroy. (Eds.)
Advances in Artificial Intelligence Theory
Research on Computing Science 16, 2005, pp. 23-32

systems of circumscription are usually based on minimal modes, and disproof-procedures are needed to generate minimal models. In the tableau calculus it is needed to show that the branches which are not closed are not minimal models by showing that there are predicates not proved in the branches [8]. For a given theory T and a query the MILO-resolution needs to generate a tree which shows that T and the query are consistent, and to prove all the leaves in T [10].

There are several restrictions in these provers, 1) The selection priority of the predicates circumscribed and the variable predicates must be given a priori [6]. 2) There are no procedures for the cases where the predicates include functions, i.e., the ones undecidable. Originally nonmonotonic systems are required to realize the concepts of plausibility and normality needed. The priority and the variable predicates must be chosen such that the requirements are satisfied. Therefore these vary accordingly to the queries. In this meaning a goal-oriented prover is practically needed such that both the circumscribed (the priority) and the variable predicates can be suggested in the processes of proofs. Regarding 2) the circumscription formula includes at least Π^0_1 formulas of arithmetical hierarchy, and the system is undecidable. Therefore semi-decidable provers are impossible. However if the undecidability is confined it is possible to make provers semi-decidable except the confined undecidable procedures. This method gives a practical way to solve problems for circumscription. For example if the undecidability is due to the proof of the consistency then it can be resolved practically by giving a model though not formally.

This paper proposes a goal-oriented prover which solve practically the above two problems. The proof-system is an application of that for the set of formulae in the disjunctive normal forms (DNF) [2]. The system is the set of Horn clauses called *meta-rules* of a second-order predicate *Prov* called a *meta-predicate* which means a clause. The system is called a *meta-system*, and is generated from the DNF formulae in the object system T. SLD-resolution procedures are given for the meta-system, and the search space is an SLD-tree. The meta-system is sound and complete in the meaning that if Prov with the empty value is proved in the meta-system then the object system is inconsistent, and vice versa. Let T be the set of clauses. For a given predicate p the circumscription formula of p is that $\alpha \leftarrow p$ is proved from $p \leftarrow \alpha$ and $T(p/\alpha)$, where $T(p/\alpha)$ is given by substituting α for p in T. α is called a *circumscribing formula* of p. The circumscription formula is a DNF formula, since the body consists of $p \leftarrow \alpha$ and $T(p/\alpha)$, where $T(p/\alpha)$ is a conjunction of clauses due to the fact that T is the set of clauses. Therefore the circumscription formula is a meta-rule in the meta-system and is called *the circumscription meta-rule*. Generally (T+circumscription) is undecidable and so is the meta-system.

In Sec.2, the proof system is formulated. In Sec.3, based on the prover, circumscribing formulae are given without variable predicates for the set of typical clauses of predicates with functions. Practically circumscription is applied to prove the negation of p. For the formulae given above to be false, the condition must be satisfied that T made p false is consistent with T. When the condition is satisfied, variable predicates are generalized so as to prove T made p false and are called *generalized variable-predicates*. Generally the consistency check is undecidable. By encapsulating the consistency check as an oracle, a practical logic programming for circumscription is given for queries without universal quantifiers, i.e., circumscription

problems are practically solved by giving a model to show the consistency. The oracle for queries with universal quantifiers is also given.

2 Formalization

Let (Γ) be a set of formulas of first-order logic. Without loss of generality it is assumed that predicates are one variable except the special predicate =, the equality. Assuming Skolem's functions and the Henkin theory of (Γ), let Γ be the matrix of (Γ) represented as the set of DNF formulae. In the following each upper case letter, U, W, X, Y or Z, takes the values of disjunctions, conjunctions of positive predicates, or the empty symbol □, and each lower case letter except x and y represents a predicate in the formula the upper case letter denotes. x or y is for the variable of (Γ). X and Y are the variables of the meta-system called M whose values are the conjunctions and the disjunctions of the Herbrand base of Γ, respectively. Now introduce a meta-predicate Prov(X;Y) which means Y←X, where X and Y are a conjunction and a disjunction of positive predicates, respectively. When Y={Y1,Y2} and X=X1&X2, Prov(X;Y) is also represented as Prov(X1,X2;Y1,Y2), where & is the conjunction sign and {} means the disjunction of the elements. The meta-system for to Γ is defined as follows.

Definition 1 Let Γ be the set of the formulas, (Z←W)←(Z1←W1)&…&(Zm←Wm) and (Z'←W')←. The meta-system M is the set of meta-rules as follows.

M0 Prov(X,X1;Y1,Y) ← Prov(X,X1,X1;Y1,Y1,Y),
M1 Prov(X,u;u,Y) ←,
M2 Prov(X;Y) ← Prov(X;u1,Y)&…&Prov(X;uj,Y)&Prov(X,U;Y), U=u1&…&uj,
M3 Prov(X,W;Z,Y) ← Prov(X,W,W1;Z1, Z,Y)&…&Prov(X,W,Wm;Zm, Z,Y),
M4 Prov(X,W';Z',Y) ←,

where u, u1,…, and uj are predicates (x is not shown explicitly). The left and the right formulae of a meta-rule are called the head and the body, respectively. When the negation of a query is added to Γ, the query is represented as a conjunction in the normal form and the corresponding meta-rule is M3 with W and Z empty. The body of a meta-rule without the head is called a *goal clause* for simplicity. It is noted that *M is a proving system without the negation symbol ~.*

Definition 2 (The SLD-tree) The root node is Prov(□; □). Each node is labeled a conjunction of Prov. Let Prov(X1;Y1) be the left most of the conjunction labeled to a node. Then the conjunctions below the node are generated as follows. In the following 'X1 (Y1) includes W (Z)' means that every predicate in W (Z) matches with a predicate in X1 (Y1) with the most general unifier. X-X1 means the formula removed atoms in X1 from X.

P1: If Prov(X1;Y1) matches with the head of M1or M4 then remove it.
P2: If X1 and Y1 include W and Z in M3, respectively, then remove it and add Prov(X1,Wi;Zi,Y1), (1≤i≤m).
P3: If Y1 includes Z in M3 and does not include any ui, (1≤i≤j), where u1&…&uj=

W-X1, then remove Prov(X1;Y1) and add Prov(X1;ui,Y1), (1≤i≤j), and Prov(X1,W,Wi;Zi,Y1), (1≤i≤m).

P4: If Y1 includes Z' in M4 and does not include any ui, (1≤i≤j), where u1&…&uj= W'-X1, then remove Prov(X1;Y1) and add Prov(X1;ui,Y1), (1≤i≤j).

The generated meta-predicates obtained by applying the above procedures repeatedly are called *the descendants* of Prov(X1;Y1). It is noted that for P2, P3, or P4 to be applied to Prov(X1;Y1), Y1 must include Z or Z'. The branch whose leaf is the empty node is called a *success-branch*. Theorems 3 and 4 have been proved in [2].

Theorem 3 *If there is the empty node in the SLD-tree for M then Γ is inconsistent and vice versa.*

Theorem 4 *If the SLD-tree includes the empty node then there is a success-branch such that any meta-predicate in a node doesn't appear in the descendants, i.e., there are no loops such that a meta-predicate is expanded repeatedly.*

Example 5 To show the existence of x for p(x) from {p(a),p(b)}, where a and b are constant. Γ is {p(a),p(b)}← and the negation of the query is ←p(x). Corresponding meta-rules and the goal clause for Prov(□; □) are, respectively,

Prov(X; p(a),p(b),Y) ←, (1)

Prov(X;Y) ← Prov(X;p(x),Y), (2)

← Prov(□; □), (3)

where the other meta-rules, M0, M1, and M2 are omitted (so are hereafter). (2) is the meta-rule corresponding to the negation of the query in Γ. The SLD-tree is

Prov(□;□) - Prov(□;p(x)) - Prov(□;p(x),p(x')) - □.

The second is obtained by matching Prov(□;□) with the head of (2) and P2. The third is given by matching Prov(□;p(x)) with the head of (2) and P2. By Theorem 4, x and x' are different. The last is shown by (1) and P1 with x=a and x'=b.

Let T be the set of clauses. The formulation of circumscription is given in second-order logic with the universal quantifier. The universal quantifiers in first and second-order logics satisfy the same inference rules [9] and a circumscription formula is a DNF formula in T. Therefore the meta-rule for circumscription is in the form of M3 and is given as follows.

C1 Prov(X,p(x);α(x),Y) ← Prov(X,p(x),α(c);p(c),α(x),Y)&T(p/α,x/c':X,p(x);α(x),Y),

where c and c' are constant symbols not appearing in T and T(p/α,x/c':X,p(x); α(x),Y) is the conjunction of meta-predicates for clauses in T replaced p and x with α and c', respectively. For example when T is the set of q(x)←p(x) and r(x)←s(a),

T(p/α,x/c':X,p(x);α(x),Y)≡Prov(X,p(x),α(c');q(c'),α(x),Y)

& Prov(X,p(x),s(a);r(c'),α(x),Y).

However the second meta-predicate is removed since it is true in M.

The other meta-rules needed for (T+circumscription) relate to the equality (=).

C2 Prov(X,X1(x);Y1(x),Y) ← Prov(X,X1(x),X1(y),x=y;Y1(y),Y1(x),Y)
&Prov(X,X1(x);x=y,Y1(x),Y),

C3 Prov(X,x=y,x=y';Y) ← Prov(X,y=y';Y),

C4 Prov(X;x=x,Y) ←,

where the variables x and y in X1 and Y1 are explicitly shown. C2 is the meta-rule to show that if a meta-predicate is proved at x=y and for any values not equal to y then the meta-predicate is proved for any values of x.

It is noted that α in C1 is a variable in the meta-system. By definition α(x) in the head of C1 is a disjunction of positive predicates and α(c) in the first meta-predicate in the body is a conjunction of positive predicates. Moreover α is unified with any formula. This is understood by regarding α as a positive predicate with the following auxiliary meta-rules, respectively, corresponding to {α1(x),α2(x)}, α1(x)&α2(x), {~α1(x),α2(x)}, and ~α1(x)&α2(x) for α(x).

A1 Prov(X, α(x);Y) ← Prov(X,α1(x);Y)&Prov(X, α2(x);Y),

A2 Prov(X;α(x),Y) ← Prov(X;α1(x),Y)&Prov(X;α2(x),Y),

A3 Prov(X;α(x),Y) ← Prov(X,α1(x);α2(x),Y),

A4 Prov(X,α(x);Y) ← Prov(X,α2(x);α1(x),Y).

Definition 6 *The meta-system MC* for (T+circumscription) is the set of M0, M1, M2, C1, C2, C3, C4 and the meta-rules of M4 for the clauses in T, and the auxiliary meta-rules A1, A2, A3, and A4.

3 Practical Logic Programming for Circumscription

Example 7 T is {p(a),p(b)}←. Then p(x)={p(a)&x=a, p(b)&x=b} is proved in MC. p(x)←{p(a)&x=a, p(b)&x=b} is obvious. {p(a)&x=a, p(b)&x=b}←p(x) is shown as follows. Meta-rules are with α(x)≡{p(a)&x=a,p(b)&x=b} and a constant d not in MC,

Prov(X;p(a),p(b),Y) ← ,

Prov(X;Y) ← Prov(X,p(d);α(d),Y).

A success-branch is given as follows, with X1≡p(d)&α(c), and Y1≡{p(c),α(d)}. The root, Prov(□;□), is omitted (so is hereafter).

Prov(p(d);α(d)) - Prov(X1;Y1) - Prov(X1,α(a),c=a;p(a),Y1)&Prov(X1;c=a,Y1) -
- Prov(X1;c=a,Y1) - Prov(p(d),α(c);c=a,c=b,p(c),α(d)) - □.

The second node is given by matching with C1 and is the first meta-predicate in the body of C1 since T(p/α,x/c':X,p(x); α(x),Y) is true. The third is given by applying C2 with y=a. By applying A1 to the first meta-predicate Prov(X1,p(a),c=a;p(a),Y1) and Prov(X1,p(b),c=b,c=a;p(a),Y1) are obtained. Both are true by P1 and by C3 with a=b false, respectively, and the fourth node is given. Similarly the fifth node is obtained by applying C2 with y=b. The last is given by A1 and P1.

It is noted that {x=a,x=b}←p(x) is also proved in Example 7. Then the following two circumscription meta-rules corresponding to α(x)≡x=a and α(x)≡x=b are used.

Prov(X,p(x);x=a,Y) ← Prov(X,p(x),c=a;p(c),x=a,Y),

Prov(X,p(x);x=b,Y) ← Prov(X,p(x),c'=b;p(c'),x=b,Y).

Similarly the well-known solution, i.e., for all x and all y {(x=a←p(x)), (y=b←p(y))}, is also proved in MC by using the above two circumscription meta-rules with the following meta-rule for the query and constants, d and d', not in MC.

Prov(X;Y) ← Prov(X,p(d),p(d');d=a,d'=b,Y).

Example 8 (The priority and variable predicates) With A, S, E, p1, and p2 for adult, student, employed, abnormal1, and abnormal2, respectively, Let T be p1(x)←S(x)&E(x), {E(x),p2(x)}←A(x), A(x)←S(x), S(m)← and let the query be ~E(m), where m (Mary) is a constant. The corresponding meta-rules are, respectively,

Prov(X,S(x),E(x);p1(x),Y) ←, (4)

Prov(X,A(x);E(x),p2(x),Y) ←, (5)

Prov(X,S(x);A(x),Y) ←, (6)

Prov(X;S(m),Y) ←, (7)

Prov(X;Y) ← Prov(X,E(m);Y). (8)

The circumscription meta-rules for p1 and p2 are, respectively,

Prov(X,p1(x);α1(x),Y)
←Prov(X,p1(x),α1(c);p1(c),α1(x),Y)&Prov(X,p1(x),S(c'),E(c');α1(c'),α1(x),Y), (9)

Prov(X,p2(x);α2(x),Y)
←Prov(X,p2(x),α2(d);p2(d),α2(x),Y)&Prov(X,p2(x),A(d');E(d'),α2(d'),α2(x),Y).(10)

It is noted that the second meta-predicate in the body of (9) is T(p1/α1,x/c':X,p1(x); α1(x),Y) since from (5) to (8) the meta-predicates are not changed and are true. Therefore these meta-predicates in T(p/α,x/c':X,p(x); α(x),Y) are dropped. Similarly (10) is obtained. The first and the second nodes of a success-branch are given by making α1(x) empty for a value x' in (9),

Prov(E(m); □) -
- Prov(E(m);p1(x'))&Prov(p1(x'),α1(c);p1(c))&Prov(p1(x'),S(c'),E(c');α1(c')) -.

The second node is given by P3. The first meta-predicate of the second node is expanded into Prov(E(m);S(x'),p1(x'))&Prov(E(m);E(x'),p1(x')) by matching with (4) and by applying P4 . From (7) the first meta-predicate is removed with x'=m by P1, and the second meta-predicate is also removed by P1. Therefore the first meta-predicate in the second node is removed. The third and the last nodes are

- Prov(p1(m),α1(c);p1(c))&Prov(p1(m),S(c'),E(c');α1(c')) - □.

The first meta-predicate in the third node is matched with (4) by unifying x and α1(c) with c and S(c)&E(c), respectively, and is removed. By using the auxiliary meta-rule A2, it is shown that the second meta-predicate is also removed by P1 and the empty node is obtained. The condition that α1(m) is false is satisfied by requiring that S(x)&E(x) is false, i.e., there are no students employed. It is easily shown that the empty node is not obtained by making α2(m) empty. Therefore the priority of p1 is higher than that of p2 and it is required that S or E is the variable predicate.

Suppose that for another student k (Ken), E(k) is another plausible query. Adding

Prov(X;S(k),Y) ←, (7')

require that Prov(□;E(k)) is proved. A success-branch is by making α2(x) in (10) empty at x=x',

Prov(□;E(k)) -
- Prov(□;E(k),p2(x'))&Prov(p2(x'),α2(d);p2(d))&Prov(p2(x'),A(d');E(d'),α2(d')) -.

The first meta-predicate of the second node is replaced by Prov(□;A(k),E(k),p2(k)) by matching with (5) for x'=k and from P4. From (6) with P4 and (7') with P1 the first meta-predicate is removed. The third and the last nodes are

- Prov(p2(k),α2(d);p2(d))&Prov(p2(k),A(d');E(d'),α2(d')) - □.

The first meta-predicate of the third node is removed by applying A4 with α2(d)=~α21(d)&α22(d) and by matching with (5) by unifying α21(d) and α22(d) with E(d) and A(d), respectively. The second meta-predicate is also removed by applying A2 and A3. The priority of p2 is higher than that of p1 and the variable predicate is A or E. The condition that α2(k) is false is satisfied by requiring that ~E(k)&A(k) is false. It is easily shown that p1(k) and p2(m) are true, i.e., Ken is abnormal as a student and so is Mary as an adult. In this case the condition that S(x)&E(x) is false is not correct since α1(k) is true. An alternative is that α1(x)≡S(x)&E(x)≡x=k, i.e., only

Ken is abnormal as a student. Similarly $\alpha 2(x) \equiv \sim E(x) \& A(x) \equiv x=m$, i.e., only Mary is abnormal as an adult.

Similarly the following Example 9 is proved in MC. Let Q(x) be a conjunction of positive or negative predicates except p and if Q(x) includes p then p is positive with functions.

Example 9 Let T be $p(x) \leftarrow Q(x)$, or $\{p(x),p(a)\} \leftarrow Q(x)$, or $\{p(x),p(f(x))\} \leftarrow$. Then $p(x)=Q(x)$, or $p(x)=\{Q(x)\&\sim p(a),\ p(a)\&x=a\}$, or $p(x)=p(x)\&\{p(f(-2,x)),p(f(2,x))\}$, respectively, is proved in MC.

As seen in Examples 8 and 9, to prove $\sim p(t)$ for a given term t, a sufficient condition is that T made p(t) false is required. Generally this form of circumscription is obtained by using generalized variable-predicates defined below.

Definition 10 Let S be a subset of the *Herbrand universe* of T. Then *T([p/□,S])* is defined by the set of clauses of the form $Z \leftarrow W$ in T for which W doesn't include p(t) and Z includes p(t) and from which p(t) is removed for t in S. *Generalized variable-predicates* are defined such that T([p/□,S]) is satisfied when T([p/□,S]) is consistent with T.

Theorem 11 *If (T+T([p/□,S])) is consistent then ~p(t) for t in S is proved with the generalized variable-predicates.*

Proof: Let $\alpha(x)$ be false for x in S, and be p(x) for x not in S. Then $p(x) \leftarrow \alpha(x)$ is proved and $T(p/\alpha)$ is proved assuming T([p/□,S]).

It is noted that for c not in T, $\sim p(c)$ is not proved since α is a model of p, t is in the Herbrand universe of T. It is also noted that ~p is not proved from T([p/□,S]), but is proved by circumscription with generalized variable-predicates. Example 7 is also shown by Theorem 11. Because let S be the set of x such that $x \neq a,b$. Since T([p/□,S]) is empty and consistent with T, without generalized variable-predicates, $\alpha(x)$ is given by the one which is false for $x \neq a,b$ and is p(a) for x=a and p(b) for x=b.
Usually the consistency check is undecidable. By encapsulating the consistency check as an oracle the logic programming with oracles is given in the following.

Definition 12 For a term t, *O(p(t):S)* is the oracle answering true if T([p/□,S']) is consistent with T and false otherwise, where S' is the sum of S and t. The extended predicate $Prov(X;Y:\pi,\Sigma)$ is defined from Prov(X;Y) by adding two variables π and Σ for predicates and subsets of the region of x, respectively. The oracle and *Prov(X;Y:π,Σ)* satisfy the following meta-rules O1, O2, and G for the initial goal clause.

O1 $Prov(X,r(x);Y:r(x),\Sigma) \leftarrow Prov(X,r(x);O(r(x):\Sigma),Y:r(x),\Sigma)$,

O2 $Prov(X;Y:r(x),\Sigma) \leftarrow Prov(X;Y:\pi,\Sigma')$,

G $\leftarrow Prov(□;□:\pi,\Sigma)$,

where r is the variable for predicates and Σ' is defined by adding x to Σ.

O2 is for preserving the information regarding Σ obtained in the proof. The following Theorem 13 is immediately obtained.

Theorem 13 *Let Mo be the meta-system M for T with O1 and O2. If ~p(t) is proved by using C1 once then ~p(t) is proved in Mo, and Mo is decidable regarding circumscription. The converse is obtained for MC with generalized variable-predicates.*

Example 14 Let T be p(a)← and p(x)←p(f(x)). Suppose that there is not an n such that a=f(n,b). Let S be the set of (b, f(b), ..., f(n,b), ...). Then T([p/□,S]) is empty. Therefore ~p(b) is proved without generalized variable-predicates. In Mo there is the success-branch due to the oracle.

Prov(p(b);□) - Prov(p(b);O(p(b):S)) - □.

It is noted that the oracle in Definition 12 is for queries with the existential quantifier. For the query, α(x)≡f(n(x),x)=a, where n(x) is the Skolem's function for n, the query includes the universal quantifier regarding x. In this case the oracle requires the information about the region in which α is false and is more complicated than the one give above.

Definition 15 Let *O(X1,p;Y1:Sp)* is the oracle answering true if T([p/□,Sp]) is consistent with T and false otherwise, where Sp is the complement of the region in which Y1(x)←p(x)&X1(x) is proved. The oracle satisfies

Ou Prov(X,X1(x),r(x);Y1(x),Y:π,Σ) ← Prov(X;O(X1,r;Y1:Sr),Y:π,Σ),

which is proved by Theorem 11. By using Ou, with α(x)≡f(n(x),x)=a, α(x)←p(x) is proved since there is a success-branch such that

Prov(p(c);α(c):π,Σ) - Prov(□;O(p;α:Sp):π,Σ) - □,

where Sp is x such that f(n,x)≠a for all n. The empty node is given by the oracle since T([p/□,Sp]) is consistent with T.

As is easily seen when T is {p(a),p(b)}←, ~p(a) or ~p(b) can be inferred with generalized variable-predicates. Therefore there is a problem in the definition of generalized variable-predicates. To remove the problem one way is to require that T([p/□,S]) is empty. However it is well known that meaningful results aren't given under the condition. Another way is to restrict the inference regarding ~p(a) or ~p(b) as follows.

Definition 16 For a generalized variable-predicate p(t) let φ be a disjunction of positive or negative predicates. Consider the restriction that ~p(t) can't be inferred if there is a clause φ such that {p(t),φ} is proved but φ is not proved in T. *'Semi-general'* and *'restricted'* variable-predicates are the cases where φ is the disjunction of positive p and φ is any disjunction not including ~p(t), respectively.

It is easily shown that for the semi-general case the proof of ~p doesn't depend on S but depends on another circumscribed predicate q is used in the proof. For the restricted case it is also shown that the necessary and sufficient condition to prove ~p(t) is that there are no clauses in T including p(t). An easy way to implement the above restrictions is the use of oracles answering under the restrictions. Then the oracle for the semi-general case is the most undecidable among the three.

4 Conclusion

A goal-oriented prover for solving circumscription problems was presented. A feature of the prover is that the priority and the variable predicates are suggested so as to prove the intended results. Based on the prover, formulae were given which satisfy circumscription formulae without variable predicates for the set of typical clauses of predicates with functions. Practically circumscription is applied to prove the negation of a predicate. For the formulae given above to be false, the condition must be satisfied that T made p false is consistent with T. Variable predicates are generalized so as to prove T made p false. By the generalization it was shown that circumscription problems are practically solved by giving a model to show the consistency for the cases where predicates include functions for which the problems become undecidable. Generally consistency problems are undecidable. By encapsulating the consistency check as an oracle a decidable prover was presented for queries without universal quantifiers. The oracle with universal quantifiers was also given. This prover is practical in the meaning that the consistency is proved by giving a model of T. The restrictions, semi-general and restricted, were considered for generalized variable-predicates not to infer undesirable predicates. These restrictions are additive and more unified formulations are desired.

References

1. G. Antoniou. Nonmonotonic Reasoning. The MIT Press, 1997.
2. M. Arai. A Parallelism-Oriented Prover with a Meta-Predicate. In *Proceedings of Second IEEE International Conference on Intelligent Systems,* pages 138-143. IEEE, 2004.
3. P. A. Bonatti and N. Olivetti. Sequent Calculi for Propositional Nonmonotonic Logics. *Transactions on Computational Logic,* 3(2):226-278. ACM, 2002.
4. M. Cadoli and M. Schaerf. A Survey of Complexity Results for Non-Monotonic Logics. *The Journal of Logic Programming,* 17:127-160. Elsevier Science Publishing, 1993.
5. U. Egly and H. Tompits. Proof-Complexity Results for Nonmonotonic Reasoning. *Transactions on Computational Logic,* 2(3):340-387. ACM, 2001.
6. V. Lifschitz. Computing Circumscription. In *Proceedings of IJCAI-85,*121-127, 1985.
7. J. McCarthy. Applications of Circumscription to Formalize Commonsense Knowledge. *Artificial Intelligence,* 28:89-116, 1986.
8. I. Niemela. Implementing Circumscription Using a Tableau Method. In *Proceedings of ECAI 96.* pages 80-84, John Wiley & Sons,1996.
9. W. Pohlers. Subsystems of Set Theory and Second Order Number Theory. *Handbook of Proof Theory (S. R. Buss, Editor)*, pages 209-336, Elsevier Science B. V., 1998.
10. T. C. Przymusinski. An Algorithm to Compute Circumscription. *Artificial Intelligence,* 38:49-73, 1989.