ICOT Technical Memorandum: TM-0855

TM-0855

Extracting Answers in Circumscription

by
N. Helft, K. Inoue & D. Poole

February, 1990

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Extracting Answers in Circumscription

Nicolas Helft **Katsumi Inoue**
ICOT Research Center
1-4-28 Mita, Minato-ku, Tokyo 108, Japan

David Poole
University of British Columbia
Vancouver, B.C., Canada V6T 1W5

December 18, 1989

Abstract

This paper gives a solution to two problems that arise in recent implementations of theorem provers for circumscription. The first problem is that, when a query contains variables, they only produce yes/no answers, rather than returning the actual values of the variables for which the query holds. The problem can be attributed to these theorem provers finding all explanations for a goal, rather than finding a minimal disjunct of these explanations. As well as providing answer extraction, we show how this technique of finding only minimal disjuncts of explanations can save much redundant computation.

1 Introduction

It has been observed in the literature that an important property of circumscription [7] that other commonsense representation formalisms lack, is that it is based on first-order predicate logic. This property has allowed the recent development of theorem proving procedures for circumscriptive theories [10, 3] and membership in all extensions [9] [1].

However good these procedures might be, they suffer from a number of problems, to which a solution is proposed in this paper. We first show an important drawback of both the procedures [10, 3]: the algorithms give only yes/no answers, that is, if the query contains variables, the values for it are not returned. This is clearly a serious drawback if we wish to use circumscription as part of a knowledge representation system. We propose a solution to this problem, and show how this solution can help improve efficiency.

[1] Etherington [1, Theorem 8.3] has shown the equivalence of these for propositional theories.

2 Background

This section gives an very brief survey of theorem provers for circumscription.

These are based on the following definition and result.

Definition 2.1: Let T and F be formulae, and $\mathcal{P}$ a set of literals. Then E is an *explanation* of F with respect to T and $\mathcal{P}$ if

1. $T \cup E \models F$
2. $T \cup E$ is consistent
3. E is a conjunction of elements of $\mathcal{P}$

A formula is *unexplained* if there exist no explanation for it.

In circumscription, predicate symbols are divided in three disjoint sets: predicates in P are minimized, those in Q are fixed and those in Z are allowed to vary [7, 6]. $CIRC(T; P; Z)$ denotes the circumscription of the formula T with minimized predicates P and variable predicates Z. We also assume Uniqueness of Names, as this is needed for the following to hold.

Theorem 2.2: [2] $CIRC(T; P; Z) \models F$ if and only if there is a disjunct of explanations E for F with respect to T and $\mathcal{P} = P^- \cup Q$ [2] such that $\neg E$ is unexplained.

While it is not necessary E contains no two explanations one of which is a subset of the other, this should be avoided to save redundant computation while checking unexplainability.

To compute the explanations and check for unexplainability, we observe that if $T \cup E \models F$, then $T \cup \neg F \models \neg E$. We thus need theorems of $T \cup \neg F$ belonging to $P^+ \cup Q$ [3]. The negation of conjunctions of the formulae obtained are the explanation we are looking for. If no formula E can consistently be added to T in such a way that F is derivable, then F is unexplained.

For these computations to be done, the theorem provers use an algorithm which, given a background theory T, a formula q in clausal form, and a set of predicate symbols, computes theorems of $T \cup \{q\}$ that can be expressed using only the given predicate symbols. This set of all theorems can be infinite in the first-order case and a suitable canonical form has to be found.

The algorithm can be seen as linear resolution with two modifications which are:

1. It should be complete in the sense that every relevant theorem can be produced, instead of the most restrictive sense of completeness used in theorem proving, that is, producing the formula *false* if the set is inconsistent.

[2] P^+ (P^-) denotes the set of positive (negative) literals whose predicate symbol belongs to P; Q denotes all literals, positive and negative, whose predicate symbol is in Q.

[3] The predicates of P have their sign changed because we look for the negation of E.

2. It should focus on producing only theorems that can be expressed within the given set of predicate symbols.

Algorithms having the first property were investigated by Slage & al. [12], and Minicozzi & Reiter [8] which called them *consequence finding* algorithms. Siegel [11] has recently produced a procedure which both improves efficiency and has the second property as well.

Helft & Inoue [5] expand on this comparison of the linear resolution algorithms. In this paper, we are concerned with another, related problem. The theorem provers for circumscription use a *query answering procedure* that calls the linear resolution algorithms to produce the answer. The problems we attempt to solve here arise from this query answering procedure rather than from the linear resolution algorithm. In this paper, which complements [5], we assume this algorithm exists and returns explanations, and concentrate on the query answering procedure.

The following example, although trivial, may be useful to understand how all this works.

Let the theory be

$$T = \{\, bird(tweety),\ \forall X\ bird(X) \wedge \neg ab(X) \supset flies(X) \,\}.$$

In this well-known example, $P = \{ab\}$, $Q = \{bird\}$ and $Z = \{flies\}$, so that $\mathcal{P} = \{ab^-, bird^+, bird^-\}$. Now, let the query be $flies(tweety)$.

Then $E = \neg ab(tweety)$, is an explanation of $flies(tweety)$. Its negation $ab(tweety)$ is unexplained. Thus $flies(tweety)$ is in the circumscribed theory.

If the query is $\neg flies(tweety)$, the answer is "No": there is simply no explanation for this formula.

3 Extracting answers

Consider the following example. T contains the formulae

$\forall X\ p(X) \wedge \neg ab(X) \supset q(X)$

$p(a) \vee p(b)$

$p(c)$

$p(d) \vee p(e)$

$ab(a) \vee ab(c)$

The query is $q(X)$, that is, we want to know for which values of X does $q(X)$ hold. ab is the predicate whose extension is minimized, and both p and q are allowed to vary [4]. X is a variable and a, b, c, d and e are constants.

[4] We let p and q vary because the algorithm in [3] works only in the case all non-minimized predicated can vary, but we could fix p as it is allowed in [10] without affecting the example.

Let's see what the answers to the query are. All the P, Z-minimal models of the above set contain $\neg ab(d)$ and $\neg ab(e)$, and thus

$q(d) \vee q(e)$

is one of such answers. Moreover, these minimal models can be divided in two sets, those containing $\neg ab(a)$ and those containing $\neg ab(c)$. In the first of these sets of models $q(a) \vee q(b)$ holds, and so does $q(c)$ in the second. Thus

$q(a) \vee q(b) \vee q(c)$

hold in all the minimal models, and is another answer to the query.

However, none of the algorithms [10, 3] gives these answers. The reason is that they implicitly use the property that, if a certain disjunct of explanations whose negation is unexplained exist, then the maximal disjunct will also have its negation unexplained. This maximal disjunct is then tested for unexplainability.

These algorithms would thus find the three possible explanations for $q(X)$ which are

$E1 = \neg ab(a) \wedge \neg ab(b)$

$E2 = \neg ab(c)$

$E3 = \neg ab(d) \wedge \neg ab(e)$.

Now they would consider the disjunct $E1 \vee E2 \vee E3$, and compute its negation, which is the conjunction of the following clauses.

$\neg ab(a) \vee \neg ab(c) \vee \neg ab(d)$

$\neg ab(a) \vee \neg ab(c) \vee \neg ab(e)$

$\neg ab(b) \vee \neg ab(c) \vee \neg ab(d)$

$\neg ab(b) \vee \neg ab(c) \vee \neg ab(e)$.

Each of these formulae is unexplained, and thus the answer is "YES", that is, $\exists X\ p(X)$. This answer is of course logically correct, but it doesn't give us information about the values for which the query holds. The algorithm of [10] only gives yes/no answers; in [3] some examples are given in which a value is returned, which turns out to be the value for the current substitution computed by the answering algorithm. The above example shows that this would not work in general. The reason is that when the negation of the explanations are computed, the algorithm loses the substitution values for the correct answers. Intuitively, this is because both $\neg ab(d)$ and $\neg ab(e)$ are unexplained, and all four of the above clauses contain either one or the other.

We will now show how to correct this. We will not be concerned with knowing if the negation of *all* the explanations is unexplained, but rather with knowing *which* explanation or *which combination* of explanations has its negation unexplained. In other words, we look for *minimal* disjuncts of the explanations [5].

[5] By *minimal* here we mean a disjunct that contains as less explanations as possible. We require this in

We first note that each of the explanations can be associated with an answer. This can be done easily. For example, using Green's [4] technique of associating with a query $q(X)$ the formula $q(X) \vee answer(X)$. We would thus get the formulae

$ab(a) \vee ab(b) \vee answer(a) \vee answer(b)$,

$ab(c) \vee answer(c)$, and

$ab(d) \vee ab(e) \vee answer(d) \vee answer(e)$.

In other words,

$\neg ab(a) \wedge \neg ab(b)$ is an explanation for $q(a) \vee q(b)$,

$\neg ab(c)$ is an explanation for $q(c)$, and

$\neg ab(d) \wedge \neg ab(e)$ is an explanation for $q(d) \vee q(e)$.

At this point, instead of testing if the disjunction of these is unexplained, we can test each of these separately. This gives the following results.

$ab(a) \vee ab(b)$ is not unexplained, as $\neg ab(c)$ is an explanation for it.

$ab(c)$ is not unexplained, as $\neg ab(a)$ is an explanation for it.

$ab(d) \vee ab(e)$ is unexplained.

From this information we can now extract the desired answers.

The first one is easy. $q(d) \vee q(e)$ is such an answer because it has an explanation whose negation is unexplained. Neither $q(a) \vee q(b)$ nor $q(c)$ are answers, but we can see that their disjunction is, as the disjunction of the negation of their corresponding explanations is unexplained. Thus $q(a) \vee q(b) \vee q(c)$ is the second answer looked for.

Thus the general answering procedure modifies the ones of [10, 3] in the following way.

1. We associate with each explanation E_i the corresponding answer Q_i.
2. For each explanation E_i, we test if its negation is unexplained. If it is, the corresponding answer Q_i follows. If it is not, there exists an explanation for it. Call the negation of one such explanations E_i'. We then explore the lattice of disjuncts of these E_i'. When a certain disjunct follows from the theory, the corresponding disjunction of answers follows from the circumscribed theory. The lattice should obviously be explored from smaller disjuncts to bigger ones, in order to prune branches as soon as one disjunct follows from the theory.

In the above example, $E3$ is unexplained and thus we can output the corresponding answer $q(d) \vee q(e)$. To $E1$ is associated the new explanation $E1' = ab(c)$, and to $E2$ the new explanation $E2' = ab(a)$. As the disjunction of $E1'$ and $E2'$ follows from the original theory, we can output

addition of the minimality in the sense that *each* explanation should be as short as possible; the linear resolution algorithms that find explanations are responsible for this latter type of minimality, which is taken into account by [3, 10].

the disjunction of the corresponding answers. If a fourth explanation $E4$ existed and was explained by $E4'$, we would need to test for the disjunctions $E1' \vee E4'$ and $E2' \vee E4'$. However, $E1' \vee E2' \vee E4'$ would not need to be considered.

4 Improving Efficiency

This section presents some examples to show that the idea of testing for the minimal disjuncts of explanations can save much computation, as in the dialectical implementation of membership in all extensions [9].

Example 4.1

$p(0)$

$p(X) \supset p(s(X))$

$p(X) \wedge \neg ab(X) \supset q$

In this example, there is an infinite number of explanations of q: $\neg ab(0)$, $\neg ab(s(0))$, ... But any of these is enough to determine that q holds in all extensions. Take for example the first one, $\neg ab(0)$. It is very easy to verify that its negation $ab(0)$ is unexplained, and the computation can stop. An attempt to compute all explanations would give no answer.

Example 4.2

$looks\_like\_emu \wedge \neg ab1 \supset emu$

$looks\_like\_ostrich \wedge \neg ab2 \supset ostrich$

$\neg emu \vee \neg ostrich$

$emu \supset bird$

$ostrich \supset bird$

$looks\_like\_emu$

$looks\_like\_ostrich$

$looks\_like\_bird \wedge \neg ab3 \supset bird$

$long\_computation \supset looks\_like\_bird$

Here both $\neg ab1$ and $\neg ab2$ are explanations for *bird*. Their negations are unexplained, and so *bird* follows from the circumscription. At this point, there is no need to look for other explanations for *bird*; we can thus avoid the *long_computation* that would result from examining the remaining choice.

5 Conclusion

We have uncovered an important problem in theorem provers for circumscription, that of not being able to return the substitution values for a query. The reason is that these theorem provers are direct implementations of a Theorem that states the need for the existence of a certain explanation of the query, ignoring that many of such explanations may exist. We showed that a minimal, rather than a maximal disjunct of these explanations needed to be produced.

Examples were shown in which the proposed modification of the query answering procedure saves much unnecessary computation.

References

[1] Etherington, D. W., *Reasoning with Incomplete Information.* Pitman, 1988.

[2] Gelfond, M., Przymusinska, H. and Przymusinski, T., "On the Relationship between Circumscription and Negation as Failure", *Artificial Intelligence* **38** (1989), pp.75–94.

[3] Ginsberg, M., "A Circumscriptive Theorem Prover", *Artificial Intelligence* **39** (1989), pp.209–230.

[4] Green, C., "Theorem-Proving by Resolution as a Basis for Question-Answering Systems", in: B. Meltzer and D. Michie (Eds.), *Machine Intelligence* **4** (Edinburgh University Press, Edinburgh, 1969), pp.183–205.

[5] Helft, N. and Inoue, K., A Note On Computing Circumscription. ICOT TR 527/89, 1989.

[6] Lifschitz, V., "Computing Circumscription", *Proc. IJCAI-85* (1985), pp.121–127.

[7] McCarthy, J., "Applications of Circumscription to Formalizing Common-sense Knowledge", *Artificial Intelligence* **28** (1986), pp.89–116.

[8] Minicozzi, E., and Reiter, R., A Note On Linear Resolution Strategies in Consequence-Finding, in: *Artificial Intelligence* **3** 1972.

[9] Poole, D., "Explanation and Prediction: An Architecture for Default and abductive Reasoning", *Computational Intelligence* **5** (1989), pp.97–110.

[10] Przymusinski, T., An Algorithm to Compute Circumscription. *Artificial Intelligence* 38:49–73, 1989.

[11] Siegel, P., Représentation et Utilisation de la Connaissance en Calcul Propositionnel. Thèse d'État. Université d'Aix-Marseille II. 1987.

[12] Slagle, J., Chang, C. L., and Lee, R., Completeness Theorems for Semantic Resolution in Consequence Finding. *International Joint Conference on Artificial Intelligence*, Washington, D.C., 1969.