

# Computing Circumscription Revisited: Preliminary Report

Patrick Doherty\*

Dept. of Computer Science  
Linköping University  
S-581 83 Linköping, Sweden

Witold Łukaszewicz†

Andrzej Szalas†  
Institute of Informatics  
Warsaw University  
00-913 Warsaw 59, Poland

## Abstract

We provide a general method which can be used in an algorithmic manner to reduce certain classes of 2nd-order circumscription axioms to logically equivalent 1st-order formulas. The algorithm takes as input an arbitrary 2nd-order formula and either returns as output an equivalent 1st-order formula, or terminates with failure. In addition to demonstrating the algorithm by applying it to various circumscriptive theories, we analyze its strength and provide formal subsumption results based on comparison with existing approaches.

## 1 Introduction and Preliminaries

In recent years, a great deal of attention has been devoted to logics of “commonsense” reasoning. Among the candidates proposed, circumscription [Lifschitz, 1994], has been perceived as an elegant mathematical technique for modeling nonmonotonic reasoning, but difficult to apply in practice. Practical application of circumscription is made difficult due to two problems. The first concerns the difficulty in finding the proper circumscriptive policy for particular domains of interest. The second concerns the 2nd-order nature of circumscription axioms and the difficulty in finding proper substitutions of predicate expressions for predicate variables so the axioms can be used for making inferences. There have been a number of proposals for dealing with the second problem ranging from compiling circumscriptive theories into logic programs [Gelfond and Lifschitz, 1989], to developing specialized inference methods for such theories [Ginsberg, 1989; Przymusiński, 1991].

A third alternative is to focus on the more general problem of finding methods for reducing 2nd-order formulas to logically equivalent 1st-order formulas, where possible. Although some progress has been made using this approach, the class of 2nd-order circumscription formulas shown to be reducible is not as large as one might desire, the reduction methods proposed are somewhat isolated relative to each other and, most importantly,

the existing reduction theorems generally lack algorithmic procedures for doing the reductions.

In this article, we provide a general method which can be used in an algorithmic manner to reduce certain classes of 2nd-order circumscription axioms to logically equivalent 1st-order formulas. The algorithm takes as input an arbitrary 2nd-order formula and either returns as output an equivalent 1st-order formula, or terminates with failure. Of course, failure does not imply that there is no 1st-order equivalent for the input, only that the algorithm can not find one. The class of 2nd-order formulas, and analogously the class of circumscriptive theories which can be reduced, provably subsumes those covered by existing results. The algorithm can be applied successfully to circumscriptive theories which may include mixed quantifiers (some involving Skolemization), variable constants,  $n$ -ary tuples of minimized and varied predicates, separable, separated and in some cases, non-separated formulas, and formulas with  $n$ -ary predicate variables, among others. In addition to demonstrating the algorithm by applying it to some of these theories, we analyze its strength and provide formal subsumption results based on comparison with existing approaches.

Due to page limitations and the technical complexity of both circumscription and the algorithm we propose, we will be forced to remain brief with preliminaries. Consequently, we assume familiarity with the various types of circumscription and existing reduction results. In general, we will refer to the original articles for the relevant theorems and results. In addition, we provide only an informal description of the algorithm, but one that is adequate and sufficiently detailed for following the examples provided. For a detailed description of the algorithm and proofs of the subsumption results, we refer the reader to the technical report [Doherty *et al.*, 1994].

### 1.1 Notation

An  $n$ -ary *predicate expression* is any expression of the form  $\lambda\bar{x}. A(\bar{x})$ , where  $\bar{x}$  is a tuple of  $n$  individual variables and  $A(\bar{x})$  is any formula of first- or second-order classical logic. If  $U$  is an  $n$ -ary predicate expression of the form  $\lambda\bar{x}. A(\bar{x})$  and  $\bar{a}$  is a tuple of  $n$  terms, then  $U(\bar{a})$  stands for  $A(\bar{a})$ . As usual, a predicate constant  $P$  is identified with the predicate expression  $\lambda\bar{x}. P(\bar{x})$ . Similarly, a predicate variable  $\Phi$  is identified with the predicate expression  $\lambda\bar{x}. \Phi(\bar{x})$ .

\*Supported by Swedish TFR grant 93-270.

†Supported in part by KBN grant 3 P406 019 06 and ESPRIT BRA No. 6156-DRUMS II.

Truth values true and false are denoted by  $\top$  and  $\perp$ , respectively.

If  $U$  and  $V$  are predicate expressions of the same arity, then  $U \leq V$  stands for  $\forall \bar{x}. U(\bar{x}) \supset V(\bar{x})$ . If  $\bar{U} = (U_1, \dots, U_n)$  and  $\bar{V} = (V_1, \dots, V_n)$  are similar tuples of predicate expressions, i.e.  $U_i$  and  $V_i$  are of the same arity,  $1 \leq i \leq n$ , then  $\bar{U} \leq \bar{V}$  is an abbreviation for  $\bigwedge_{i=1}^n [U_i \leq V_i]$ . We write  $\bar{U} = \bar{V}$  for  $(\bar{U} \leq \bar{V}) \wedge (\bar{V} \leq \bar{U})$ , and  $\bar{U} < \bar{V}$  for  $(\bar{U} \leq \bar{V}) \wedge \neg(\bar{V} \leq \bar{U})$ .

If  $A$  is a formula,  $\bar{\sigma} = (\sigma_1, \dots, \sigma_n)$  and  $\bar{\delta} = (\delta_1, \dots, \delta_n)$  are tuples of any expressions, then  $A(\bar{\sigma} \leftarrow \bar{\delta})$  stands for the formula obtained from  $A$  by simultaneously replacing each occurrence of  $\sigma_i$  by  $\delta_i$  ( $1 \leq i \leq n$ ). For any tuple  $\bar{x} = (x_1, \dots, x_n)$  of individual variables and any tuple  $\bar{t} = (t_1, \dots, t_n)$  of terms, we write  $\bar{x} = \bar{t}$  to denote the formula  $x_1 = t_1 \wedge \dots \wedge x_n = t_n$ . We write  $\bar{x} \neq \bar{t}$  as an abbreviation for  $\neg(\bar{x} = \bar{t})$ .

## 1.2 Definitions

### Definition 1.1 (Second-Order Circumscription)

Let  $\bar{P}$  be a tuple of distinct predicate constants,  $\bar{S}$  be a tuple of distinct function and/or predicate constants disjoint from  $\bar{P}$ , and let  $T(\bar{P}, \bar{S})$  be a sentence. The *second-order circumscription* of  $\bar{P}$  in  $T(\bar{P}, \bar{S})$  with variable  $\bar{S}$ , written  $Circ_{SO}(T; \bar{P}; \bar{S})$ , is the sentence

$$T(\bar{P}, \bar{S}) \wedge \forall \bar{\Phi} \bar{\Psi}. \neg [T(\bar{\Phi}, \bar{\Psi}) \wedge \bar{\Phi} < \bar{P}] \quad (1)$$

where  $\bar{\Phi}$  and  $\bar{\Psi}$  are tuples of variables similar to  $\bar{P}$  and  $\bar{S}$ , respectively. ■

## 1.3 Useful Tautologies

Below we provide a list of some useful tautologies that are used throughout the paper.

**Proposition 1.1** The following pairs of formulas are equivalent. (Here  $Q$  stands for any quantifier and  $A, B, C$  are formulas such that  $C$  does not contain free occurrences of the variable  $x$ . In clauses (8), (9) and (11),  $\bar{t}, \bar{t}_1, \dots, \bar{t}_n$  are  $n$ -tuples of terms and it is assumed that neither  $C$  nor any term from  $\bar{t}, \bar{t}_1, \dots, \bar{t}_n$  contains variables from  $\bar{x}$ . In clause (10),  $f$  is a function variable which does not occur in  $A$ .)

- |                                                      |     |                                                                                                                            |
|------------------------------------------------------|-----|----------------------------------------------------------------------------------------------------------------------------|
| (1) $Qx(A(x)) \wedge C$                              | and | $Qx(A(x) \wedge C)$                                                                                                        |
| (2) $C \wedge Qx(A(x))$                              | and | $Qx(C \wedge A(x))$                                                                                                        |
| (3) $Qx(A(x)) \vee C$                                | and | $Qx(A(x) \vee C)$                                                                                                          |
| (4) $C \vee Qx(A(x))$                                | and | $Qx(C \vee A(x))$                                                                                                          |
| (5) $QxQyA$                                          | and | $QyQxA$                                                                                                                    |
| (6) $A \wedge (B \vee C)$                            | and | $(A \wedge B) \vee (A \wedge C)$                                                                                           |
| (7) $(A \vee B) \wedge C$                            | and | $(A \wedge C) \vee (B \wedge C)$                                                                                           |
| (8) $A(\bar{t})$                                     | and | $\forall \bar{x} (A(\bar{t} \leftarrow \bar{x}) \vee \bar{x} \neq \bar{t})$                                                |
| (9) $A(\bar{t}_1) \vee \dots \vee A(\bar{t}_n)$      | and | $\exists \bar{x} (\bar{x} = \bar{t}_1 \vee \dots \vee \bar{x} = \bar{t}_n) \wedge A(\bar{t}_1 \leftarrow \bar{x})$         |
| (10) $\forall \bar{x} \exists y A(\bar{x} \dots)$    | and | $\exists f \forall \bar{x} A(\bar{x}, y \leftarrow f(\bar{x}), \dots)$                                                     |
| (11) $A(\bar{t}_1) \wedge \dots \wedge A(\bar{t}_n)$ | and | $\forall \bar{x} (\bar{x} \neq \bar{t}_1 \wedge \dots \wedge \bar{x} \neq \bar{t}_n) \vee A(\bar{t}_1 \leftarrow \bar{x})$ |
- 

The equivalence (8) was found particularly useful by Ackermann [1935, 1954]. We extend the method by adding the equivalence (9). It makes the technique work in the case of clauses containing more than one positive (or negative) occurrence of the eliminated predicate.

This substantially generalizes the Ackermann technique. The equivalence (10) is a second-order formulation of the Skolem reduction (see [Bentham, 1983]). It allows us to perform Skolemization (i.e. elimination of existential quantifiers) and unskolemization (i.e. elimination of Skolem functions) in such a way that equivalence is preserved. We call this equivalence *second-order Skolemization*.

## 2 The Elimination Algorithm

In this section we discuss the elimination algorithm. Its complete formulation can be found in [Doherty et al., 1994]. The algorithm was originally formulated, in a weaker form, in [Szałas, 1993] in the context of modal logics. It is based on Ackermann's techniques developed in connection with the elimination problem. The elimination algorithm is based on the following lemma, proved by Ackermann in 1934 [Ackermann, 1935]. The proof can also be found in [Szałas, 1993].

**Lemma 2.1 (Ackermann Lemma)** Let  $\Phi$  be a predicate variable and  $A(\bar{x})$ ,  $B(\Phi)$  be formulas without second-order quantification. Let  $B(\Phi)$  be positive w.r.t.  $\Phi$  and let  $A$  contain no occurrences of  $\Phi$  at all. Then the following equivalences hold:

$$\exists \Phi \forall \bar{x} [\Phi(\bar{x}) \vee A(\bar{x}, \bar{z})] \wedge B(\Phi \leftarrow \neg \Phi) \equiv B(\Phi \leftarrow A(\bar{x}, \bar{z})) \quad (2)$$

$$\exists \Phi \forall \bar{x} [\neg \Phi(\bar{x}) \vee A(\bar{x}, \bar{z})] \wedge B(\Phi) \equiv B(\Phi \leftarrow A(\bar{x}, \bar{z})) \quad (3)$$

where in the righthand formulas the arguments  $\bar{x}$  of  $A$  are each time substituted by the respective actual arguments of  $\Phi$  (renaming the bound variables whenever necessary). ■

The following proposition together with the equivalences given in Proposition (1.1) is also used in the algorithm.

**Proposition 2.1** Let  $A$  be a formula of the form  $pref(A_1 \wedge \dots \wedge A_q)$ , where  $pref$  is a prefix of first-order quantifiers and  $A_1, \dots, A_q$  are disjunctions of literals. In addition, let  $\Phi$  be a predicate variable occurring in  $A$  and  $Conj(A)$  those conjuncts in  $A$  where  $\Phi$  occurs. Assume that for any conjunct in  $Conj(A)$ ,  $\Phi$  occurs either positively, or both positively and negatively (or analogously, negatively, or both negatively and positively). Then

$$\exists \Phi A \equiv pref(A_{i_1} \wedge \dots \wedge A_{i_r}) \quad (4)$$

where  $i_1, \dots, i_r \in \{1, \dots, q\}$  and  $A_{i_1}, \dots, A_{i_r}$  are all the conjuncts that do not contain occurrences of  $\Phi$  (the empty conjunction is regarded as being equivalent to  $\top$ ). ■

**Proof** See [Szałas, 1993]. ■

### 2.1 Outline of the Elimination Algorithm

We are now ready to outline the elimination algorithm. The algorithm takes a formula of the form  $\exists \Phi. A$ , where  $A$  is a first-order formula, as an input and returns its first-order equivalent or reports failure<sup>1</sup>. Of course, the

<sup>1</sup>The failure of the algorithm does not mean that the second-order formula at hand cannot be reduced to its first-order equivalent. The problem we are dealing with is not even partially decidable, for first-order definability of the formulas we consider is not an arithmetical notion (see, for instance, [Bentham, 1984]).

algorithm can also be used for formulas of the form  $\forall \Phi A$ , since the latter formula is equivalent to  $\neg \exists \Phi \neg A$ . Thus, by repeating the algorithm one can deal with formulas containing many arbitrary second-order quantifiers.

The elimination algorithm consists of four phases: (1) preprocessing; (2) preparation for the Ackermann lemma; (3) application of the Ackermann lemma; and (4) simplification. These phases are described below. It is always assumed that (1) whenever the goal specific for a current phase is reached, then the remaining steps of the phase are skipped, (2) every time the equivalence (4) of Proposition 2.1 is applicable, it should be applied.

- (1) **Preprocessing.** The purpose of this phase is to transform the formula  $\exists \Phi.A$  into a form that separates positive and negative occurrences of the quantified predicate variable  $\Phi$ . The form we want to obtain is<sup>2</sup>

$$\exists \bar{x} \exists \Phi [(A_1(\Phi) \wedge B_1(\Phi)) \vee \dots \vee (A_n(\Phi) \wedge B_n(\Phi))], \quad (5)$$

where, for each  $1 \leq i \leq n$ ,  $A_i(\Phi)$  is positive w.r.t.  $\Phi$  and  $B_i(\Phi)$  is negative w.r.t.  $\Phi$ .<sup>3</sup> The steps of this phase are the following. (i) Eliminate the connectives  $\supset$  and  $\equiv$  using the usual definitions. Remove redundant quantifiers. Rename individual variables until all quantified variables are different and no variable is both bound and free. Using the usual equivalences, move the negation connective to the right until all its occurrences immediately precede atomic formulas. (ii) Move universal quantifiers to the right and existential quantifiers to the left, applying as long as possible the equivalences (1) – (4) from Proposition 1.1. (iii) In the matrix of the formula obtained so far, distribute all top-level conjunctions over the disjunctions that occur among their conjuncts, applying the equivalences (6) – (7) from Proposition 1.1. (iv) If the resulting formula is not in the form (5), then report the failure of the algorithm. Otherwise replace (5) by its equivalent given by

$$\exists \bar{x} (\exists \Phi (A_1(\Phi) \wedge B_1(\Phi)) \vee \dots \vee \exists \Phi (A_n(\Phi) \wedge B_n(\Phi))). \quad (6)$$

Try to find Equation (6)'s first-order equivalent by applying the next phases in the algorithm to each disjunct in (6) separately. If the first-order equivalents of each disjunct are successfully obtained then return their disjunction, preceded by the prefix  $\exists \bar{x}$ , as the output of the algorithm.

- (2) **Preparation for the Ackermann lemma.** The goal of this phase is to transform a formula of the form  $\exists \Phi (A(\Phi) \wedge B(\Phi))$ , where  $A(\Phi)$  (resp.  $B(\Phi)$ ) is positive (resp. negative) w.r.t.  $\Phi$ , into one of the forms (2) or (3) given in Lemma 2.1. Both forms can always be obtained and both transformations should be performed because none, one or

<sup>2</sup>It should be emphasized that not every formula is reducible into this form.

<sup>3</sup>To increase the strength of the algorithm, it is essential to move as many existentially quantified variables as possible into the prefix of (5).

both forms may require Skolemization. Unskolemization, which occurs in the next phase, could fail in one form, but not the other. In addition, one form may be substantially smaller than the other. The steps of this phase are based on equivalences (6) – (10) from Proposition 1.1.

- (3) **Application of the Ackermann Lemma.** The goal of this phase is to eliminate the second-order quantification over  $\Phi$ , by applying the Ackermann lemma, and then to unskolemize the function variables possibly introduced. This latter step employs the equivalence (10) from Proposition 1.1.
- (4) **Simplification.** Generally, application of Ackermann's Lemma in step (3) often involves the use of equivalence (8) in Proposition 1.1 in the left to right direction. If so, the same equivalence, or its generalization (11), may often be used after application of the Lemma in the right to left direction, substantially shortening the resulting formula.

## 2.2 Discussion of the Algorithm

Assume we have a second-order formula  $A$  of the form

$$\exists \Phi. [(pref B) \wedge (pref' C)], \quad (7)$$

where,  $pref$  and  $pref'$  are sequences of first-order quantifiers,  $B$  and  $C$  are quantifier-free formulas in conjunctive normal forms,  $B$  is positive w.r.t.  $\Phi$ , and  $C$  is negative w.r.t.  $\Phi$ . Then, the following proposition holds.

**Proposition 2.2** Let  $A$  be an input formula of the form (7). Then, as a result, the algorithm returns a first-order formula provided that unskolemization (if necessary) succeeds. ■

Observe that Skolem functions are introduced in the second step of the algorithm whenever existential quantifiers are to be eliminated. These can appear in the input formula or may be introduced via application of the equivalence (9) of Proposition 1.1. In the following proposition, we formulate conditions under which no Skolem functions are introduced and the algorithm terminates successfully.

**Proposition 2.3** If one of the following conditions holds (1)  $B$  is universal and each conjunct of  $B$  contains at most one occurrence of  $\Phi$ , or (2)  $C$  is universal and each conjunct of  $C$  contains at most one occurrence of  $\neg \Phi$ , then the algorithm always returns a first-order formula as output. ■

If the input formula cannot be transformed into the form (7) then the algorithm fails.

## 3 On the Strength of the Algorithm

In this section we consider existing reduction results and their subsumption by our algorithm. A compilation of many of the existing reduction results can be found in [Lifschitz, 1994], in addition to other relevant results in earlier papers [Kolaitis and Papadimitriou, 1988; Lifschitz, 1985; 1988; Rabinov, 1989]. In [Doherty et al., 1994], we prove that the algorithm subsumes, and is even stronger than the results given in [Kolaitis and Papadimitriou, 1988; Lifschitz, 1985; 1988;

Rabinov, 1989]. We start with Rabinov's [1989] result which subsumes earlier results by Lifschitz regarding separability. In fact, the following theorem is stronger than the result of Rabinov.

Let  $D_i(P)$  denote  $N_i(P) \wedge M_i(P)$  such that the predicate constant  $P$  is positive in  $M_i$  and negative in  $N_i$ .  $D_i(P)$  is said to be *p-simple* if  $M_i(P)$  has the form  $U_i \leq P$ , where  $U_i$  is a predicate expression not containing  $P$ .  $D_i(P)$  is said to be *n-simple* if  $N_i(P)$  has the form  $P \leq U_i$ , where  $U_i$  is a predicate expression not containing  $P$ .

**Theorem 3.1** If  $T(P)$  is of the form

$$N_0(P) \wedge \bigvee_i D_i(P)$$

where each  $D_i(P)$  is either p-simple or contains no positive occurrences of  $P^4$  and  $N_0(P)$  is negative w.r.t.  $P$ , then the algorithm eliminates the second-order quantifiers from  $Circ_{SO}(T; P; ())$ . ■

The following theorem shows that the algorithm eliminates second-order quantification in the case of existential theories considered in [Kolaitis and Papadimitriou, 1988].

**Theorem 3.2** If  $T$  is a first-order existential sentence, then the algorithm eliminates second-order quantification from  $Circ_{SO}(T; \bar{P}; ())$ . ■

**Theorem 3.3** If  $T$  is a first-order monadic sentence, then the algorithm eliminates second-order quantification from  $Circ_{SO}(T; \bar{P}; \bar{S})$ . ■

The SCAN algorithm was introduced by Gabbay and Ohlbach [1992]. It is difficult to compare SCAN with our algorithm since no syntactic characterization of formulas accepted by SCAN is known. We conjecture that both approaches are successful for the same class of formulas. However, the additional advantage of our algorithm is that it always terminates, while SCAN may loop. For example, the formula

$$\forall \Phi[(\forall x \Phi(x) \supset \exists y \Phi(y) \wedge Q(x)) \supset \forall x \neg \Phi(x)]$$

when given as input to our algorithm does terminate, while for SCAN it does not.

Additional strengths and weaknesses are considered in the next section.

### 3.1 Comparison of Approaches

In comparing the different approaches and results concerning the reduction of circumscriptive theories, we will refer to Figure 1 below, which provides a pictorial view of the subsumption relation between the various theorems and types of theories reduced. DLS refers to our algorithm, MIXED refers to theories with mixed quantifiers, VC refers to theories which allow variable constants, and MONAD refers to theories with only monadic sentences. In addition,  $\forall$  and  $\exists$  refer to purely universal and existential theories, respectively, while  $\forall\exists$  refers to those theories where Skolemization is necessary, and  $\exists\forall$  refers to mixed theories not requiring Skolemization. The solid

arrows denote subsumption. In addition there are two broken solid arrows. The arrow pointing towards "Cor 3.3.3" is broken to signify that although the DLS algorithm in its general form does not fully subsume Corollary 3.3.3, when specialized appropriately, it does. We discuss this in a later section. The arrow pointing towards SKOLEM is broken to signify that the DLS algorithm works for those theories involving Skolemization when the unskolemization step is successful and the algorithm returns a first-order formula as output. Since, it may not be possible to unskolemize certain theories successfully, there is no complete subsumption of this class.

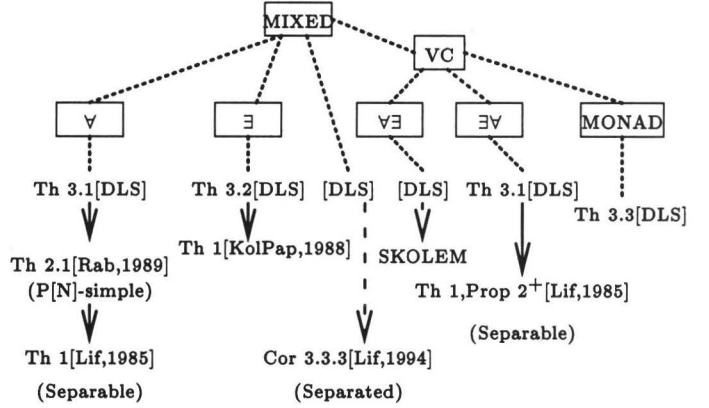


Figure 1: Subsumption Results.

### Positive Results

In addition to the results described in the previous section, observe that the method we propose is also stronger in regard to the following features:

- DLS provides us with a more general approach to existential quantification due to the possibility of allowing Skolemization. Thus it works for combinations of existential and universal quantifiers. On the other hand, Kolaitis and Papadimitriou consider pure existential formulas, while Lifschitz and Rabinov consider pure universal theories.
- DLS does not distinguish between theories with variable constants and those without. On the other hand both Rabinov, Kolaitis and Papadimitriou, (and Lifschitz to some extent), restrict their theories to those without variable constants. In some cases, Lifschitz's results can reduce theories with variable constants if the theories are separable and no Skolemization is involved. (See the next section for problems DLS has with separated theories).
- DLS permits as input circumscriptive theories with arbitrary numbers of minimized and varied predicates. This is not the case for Rabinov's result nor for Lifschitz's result pertaining to separated formulas.
- DLS describes how to constructively transform formulas into the required form.

<sup>4</sup>Rabinov requires n-simplicity here.

## Negative Results

Note that in the end of Section 2.2 we characterized the class of formulas for which the algorithm fails. Let us now discuss an additional source of weaknesses of the algorithm and a possible way of overcoming those weaknesses.

Observe that the elimination algorithm we deal with is independent of any particular theory. On the other hand, it is well known that second-order quantifiers can sometimes be eliminated when additional information is given.

One good illustrative example originates from the area of modal logics. Namely, McKinsey's axiom is not equivalent to any first-order formula. Accordingly, our algorithm fails (see [Szałas, 1993]). However, when one assumes that the accessibility relation is transitive, the elimination is possible, since McKinsey together with transitivity is first-order definable (see [Bentham, 1984]).

The same situation may occur when one computes circumscription. Consider the following theorem from [Lifschitz, 1994] (labeled "Cor 3.3.3" in Fig. 1):

**Theorem 3.4** If  $T(P)$  is a first-order sentence separated w.r.t.  $P$  then  $Circ_{SO}(T(P); P; ())$  is equivalent to a first-order sentence. ■

It permits one to deal with any sequences of first-order quantifiers provided that the formula is separated. The proof given by Lifschitz is based on a clever move which applies knowledge about the first-order theory one works with. Observe that in Theorem 3.4 the sentence  $T(P)$  is assumed to be separated, i.e. it is of the form  $T_1(P) \wedge T_2(P)$ , where  $T_1(P)$  is positive w.r.t.  $P$  and  $T_2(P)$  is negative w.r.t.  $P$ . Thus  $Circ_{SO}(T_1 \wedge T_2; P; ())$  is equivalent to

$$T_1(P) \wedge T_2(P) \wedge \neg \exists \Phi. \neg [T_1(\Phi) \wedge T_2(\Phi) \wedge \Phi < P].$$

Since  $T_2(P)$  is negative w.r.t.  $P$ ,  $T_2(P)$  together with  $\Phi < P$  imply  $T_2(\Phi)$ . Thus when  $T_2(P)$  is taken into consideration, one substantially simplifies the second order circumscription into the following second-order formula

$$T_1(P) \wedge T_2(P) \wedge \neg \exists \Phi. \neg [T_1(\Phi) \wedge \Phi < P].$$

The last formula is reducible to a first-order sentence (and is, in fact, in the scope of our algorithm).

The above examples show that the general algorithm we presented can (and should be) tuned to the particular situation it is applied to. Since circumscription is always defined over some first-order theory, moves similar to the method used by Lifschitz above, should be incorporated into the algorithm. If this is done for the case of separated theories by slightly modifying the preprocessing phase, then the specialized version of our algorithm subsumes all previous results concerning the reduction of circumscriptive theories.

## 4 Complexity of Reduction

Observe that the elimination algorithm we consider, terminates and is easily mechanizable. Let us now estimate its complexity.

First observe that during phase (3) of the algorithm, the form of the formula to be transformed is<sup>5</sup>

$$\exists \Phi \forall \bar{x} [\Phi(\bar{x}) \vee A(\bar{x}, \bar{z})] \wedge B(\Phi \leftarrow \neg \Phi) \quad (8)$$

and then its form is

$$B(\Phi \leftarrow A(\bar{x}, \bar{z})) \quad (9)$$

after application of the Ackermann lemma.

Thus, if the length of (8) is  $n$ , then the length of (9) is less than  $n^2$ . Observe, however, that this worst case occurs when  $\Phi$  has  $O(n)$  occurrences in (8). In practical examples, however, the length of (9) is usually  $O(n)$  (and often less than the length of (8)).

The worst case analysis of steps (1) and (2) shows that the size of the transformed formula can increase exponentially (due to possible transformations between disjunctive and conjunctive normal forms). This, however, is again a rare phenomenon – see examples below, in particular Section 5.2 concerning a Kolaitis and Papadimitriou example.

## 5 Applying the Algorithm to some Examples

The best way to understand how the algorithm works is to apply it to examples. Due to space limitations, we only apply the algorithm to two examples. A full catalogue of worked examples may be found in [Doherty et al., 1994]. We take a number of liberties in applying the algorithm so as not to drown in details. For example, step (2) in the previous section states that both forms of Ackermann's Lemma should be considered. In the examples, we choose one form and apply the algorithm. This saves considerable space. Also, the simplification phase is omitted unless it can be applied.

### 5.1 The Birthday Example

#### Example 5.1 (Birthday Example)

This example contains both existentially quantified and universal formulas. In addition, it contains both unary and binary predicates. Let  $\Gamma(Ab, G)$  be the theory

$$[\exists x \exists y (B(y) \wedge F(x, y) \wedge \neg G(x, y))] \wedge [\forall x \forall y (B(y) \wedge F(x, y) \wedge \neg Ab(x, y) \supset G(x, y))], \quad (10)$$

where  $B$ ,  $F$  and  $G$  are abbreviations for *Birthday*, *Friend* and *Gives Gift*, respectively. Here  $Ab(x, y)$  has the following intuitive interpretation: "x behaves abnormally w.r.t. y in the situation when y has a birthday and x is a friend of y". The circumscription of  $\Gamma(Ab, G)$  with  $Ab$  minimized and  $G$  varied is

$$Circ_{SO}(\Gamma(Ab, G); Ab; G) \equiv \Gamma(Ab, G) \wedge \forall \Phi \forall \Psi [\Gamma(\Phi, \Psi) \wedge [\Phi \leq Ab] \supset [Ab \leq \Phi]], \quad (11)$$

where

$$\Gamma(\Phi, \Psi) \equiv [\exists x \exists y (B(y) \wedge F(x, y) \wedge \neg \Psi(x, y))] \wedge [\forall x \forall y (B(y) \wedge F(x, y) \wedge \neg \Phi(x, y) \supset \Psi(x, y))] \quad (12)$$

$$\Phi \leq Ab \equiv \forall x \forall y [\Phi(x, y) \supset Ab(x, y)] \quad (13)$$

$$Ab \leq \Phi \equiv \forall x \forall y [Ab(x, y) \supset \Phi(x, y)]. \quad (14)$$

<sup>5</sup>The second form considered in lemma 2.1 is symmetric to the first one.

In the following, we will reduce

$$\forall \Phi \forall \Psi [\Gamma(\Phi, \Psi) \wedge [\Phi \leq Ab] \supset [Ab \leq \Phi]] \quad (15)$$

in (11). Negating (15), we obtain

$$\exists \Phi \exists \Psi [\Gamma(\Phi, \Psi) \wedge [\Phi \leq Ab] \wedge \neg [Ab \leq \Phi]]. \quad (16)$$

We remove  $\Psi$  first.

**Preprocessing.** Replacing  $\Gamma(\Phi, \Psi)$ ,  $\Phi \leq Ab$  and  $Ab \leq \Phi$  by their equivalents given by (12)–(14), eliminating  $\supset$ , renaming individual variables and moving existential quantifiers over individual variables to the left, we obtain

$$\begin{aligned} & \exists x \exists y \exists q \exists r \exists \Phi \exists \Psi [B(y) \wedge F(x, y) \wedge \neg \Psi(x, y) \\ & \wedge \forall u \forall z (\neg B(z) \vee \neg F(u, z) \vee \Phi(u, z) \vee \Psi(u, z)) \\ & \wedge \forall s \forall t (\neg \Phi(s, t) \vee Ab(s, t)) \wedge Ab(q, r) \wedge \neg \Phi(q, r)]. \end{aligned} \quad (17)$$

**Preparation for the Ackermann lemma.** (17) is in the form suitable for application of the Ackermann lemma. To see this, we rewrite it as

$$\begin{aligned} & \exists x \exists y \exists q \exists r \exists \Phi \exists \Psi \forall u \forall z [(\Psi(u, z) \vee \neg B(z) \vee \neg F(u, z) \\ & \vee \Phi(u, z)) \wedge \neg \Psi(x, y) \wedge B(y) \wedge F(x, y) \wedge \\ & \wedge \forall s \forall t (\neg \Phi(s, t) \vee Ab(s, t)) \wedge Ab(q, r) \wedge \neg \Phi(q, r)]. \end{aligned} \quad (18)$$

**Application of the Ackermann lemma.** Applying the Ackermann lemma to (18), we obtain

$$\begin{aligned} & \exists x \exists y \exists q \exists r \exists \Phi [(\neg B(y) \vee \neg F(x, y) \vee \Phi(x, y)) \wedge B(y) \wedge F(x, y) \\ & \wedge \forall s \forall t (\neg \Phi(s, t) \vee Ab(s, t)) \wedge Ab(q, r) \wedge \neg \Phi(q, r)]. \end{aligned} \quad (19)$$

We now remove  $\Phi$  in (19).

**Preprocessing.** (19) is in the form which is the goal of this phase.

**Preparation for the Ackermann lemma.** Using Proposition 1.1 (8), we replace (19) by

$$\begin{aligned} & \exists x \exists y \exists q \exists r \exists \Phi \forall v \forall w [(\Phi(v, w) \vee v \neq x \vee w \neq y \vee \neg B(y) \\ & \vee \neg F(x, y)) \wedge \forall s \forall t (\neg \Phi(s, t) \vee Ab(s, t)) \\ & \wedge \neg \Phi(q, r) \wedge B(y) \wedge F(x, y) \wedge Ab(q, r)]. \end{aligned} \quad (20)$$

**Application of the Ackermann lemma.** Applying the Ackermann lemma to (20), we obtain

$$\begin{aligned} & \exists x \exists y \exists q \exists r \forall s \forall t [(s \neq x \vee t \neq y \vee \neg B(y) \vee \neg F(x, y) \\ & \vee Ab(s, t)) \wedge (q \neq x \vee r \neq y \vee \neg B(y) \vee \neg F(x, y)) \\ & \wedge B(y) \wedge F(x, y) \wedge Ab(q, r)]. \end{aligned} \quad (21)$$

**Simplification.** We replace (21) by

$$\begin{aligned} & \exists x \exists y \exists q \exists r [(\neg B(y) \vee \neg F(x, y) \vee Ab(x, y)) \wedge (q \neq x \vee r \neq y \\ & \vee \neg B(y) \vee \neg F(x, y)) \wedge B(y) \wedge F(x, y) \wedge Ab(q, r)]. \end{aligned} \quad (22)$$

Negating (22), we obtain

$$\begin{aligned} & \forall x \forall y \forall q \forall r [(B(y) \wedge F(x, y) \wedge \neg Ab(x, y)) \vee (q = x \wedge r = y \\ & \wedge B(y) \wedge F(x, y)) \vee \neg B(y) \vee \neg F(x, y) \vee \neg Ab(q, r)]. \end{aligned} \quad (23)$$

(23) is logically equivalent to

$$\begin{aligned} & \forall x \forall y \forall q \forall r [\neg (B(y) \wedge F(x, y)) \vee ((B(y) \wedge F(x, y)) \\ & \wedge (\neg Ab(x, y) \vee (q = x \wedge r = y))) \vee \neg Ab(q, r)], \end{aligned} \quad (24)$$

which is equivalent to

$$\begin{aligned} & \forall x \forall y \forall q \forall r [\neg (B(y) \wedge F(x, y)) \\ & \vee \neg Ab(x, y) \vee (q = x \wedge r = y) \vee \neg Ab(q, r)]. \end{aligned} \quad (25)$$

The first-order formula (25) is logically equivalent to the second-order formula (15). Consequently,

$$\begin{aligned} & Circ_{SO}(\Gamma(Ab, G); Ab; G) \equiv \\ & \Gamma(Ab, G) \wedge \forall x \forall y \forall q \forall r [\neg (B(y) \wedge F(x, y)) \\ & \vee \neg Ab(x, y) \vee (q = x \wedge r = y) \vee \neg Ab(q, r)]. \end{aligned} \quad (26)$$

A more informative sentence, equivalent to (25), is

$$\begin{aligned} & \forall x \forall y \forall q \forall r [Ab(x, y) \wedge Ab(q, r) \wedge B(y) \\ & \wedge F(x, y) \supset (q = x \wedge r = y)]. \end{aligned} \quad (27)$$

(27), together with the theory  $\Gamma(Ab, G)$ , states that there is exactly one pair of individuals,  $x$  and  $y$ , such that  $y$  has a birthday,  $x$  is a friend of  $y$  and  $x$  does not give a gift to  $y$ .

## 5.2 An Existential Example

[Kolaitis and Papadimitriou, 1988] state

We notice that computing a first-order sentence equivalent to the circumscription of  $P$  in an existential first-order formula  $\psi(P)$  seems to increase the size of  $\psi(P)$  exponentially, a phenomenon not observed in the other known cases of first-order circumscription studied in [Lif85]. It would be interesting to determine whether this is inherent to existential first-order formulas, or a particular creation of our proof.

**Example 5.2 (Existential Example)** In light of the quote above, we take the example used by Kolaitis and Papadimitriou and compare the resulting first-order formula with that generated by our algorithm. Kolaitis and Papadimitriou apply their reduction technique to the theory

$$\exists x_1 \exists x_2 [R(x_1, x_2) \wedge P(x_1) \wedge P(x_2)] \quad (28)$$

and circumscribe  $P$  without varying predicates. The first-order equivalent they obtain is

$$\begin{aligned} & \exists x_1 (R(x_1, x_1) \wedge P(x_1) \wedge (\forall y (P(y) \equiv y = x_1)) \vee \\ & [\exists x_1 \exists x_2 (R(x_1, x_2) \wedge P(x_1) \wedge P(x_2) \wedge (x_1 \neq x_2) \wedge (\forall y (P(y) \\ & \equiv (y = x_1 \vee y = x_2))) \wedge \neg R(x_1, x_1) \wedge \neg R(x_2, x_2))]). \end{aligned} \quad (29)$$

We apply our reduction algorithm to the same theory and compare the results.

Let  $\Gamma(P)$  be the theory

$$\exists x_1 \exists x_2 [R(x_1, x_2) \wedge P(x_1) \wedge P(x_2)]. \quad (30)$$

The circumscription of  $\Gamma(P)$  with  $P$  minimized without variable predicates is

$$\begin{aligned} & Circ_{SO}(\Gamma(P); P; ()) \equiv \\ & \Gamma(P) \wedge \forall \Phi [\Gamma(\Phi) \wedge [\Phi \leq P] \supset [P \leq \Phi]], \end{aligned} \quad (31)$$

where

$$\Gamma(\Phi) \equiv \exists x_1 \exists x_2 [R(x_1, x_2) \wedge \Phi(x_1) \wedge \Phi(x_2)] \quad (32)$$

$$\Phi \leq P \equiv \forall x. \Phi(x) \supset P(x) \quad (33)$$

$$P \leq \Phi \equiv \forall x. P(x) \supset \Phi(x). \quad (34)$$

In the following, we will reduce

$$\forall\Phi[\Gamma(\Phi)\wedge[\Phi\leq P]\supset[P\leq\Phi]] \quad (35)$$

in (31). Negating (35), we obtain

$$\exists\Phi[\Gamma(\Phi)\wedge[\Phi\leq P]\wedge\neg[P\leq\Phi]]. \quad (36)$$

**Preprocessing.** Replacing  $\Gamma(\Phi)$ ,  $\Phi\leq P$  and  $P\leq\Phi$  by their equivalents given by (32)–(34), eliminating  $\supset$  and renaming individual variables, we obtain

$$\begin{aligned} &\exists\Phi[\exists x_1\exists x_2(R(x_1, x_2)\wedge\Phi(x_1)\wedge\Phi(x_2))\wedge \\ &\quad \forall y(\neg\Phi(y)\vee P(y))\wedge\exists z(P(z)\wedge\neg\Phi(z))]. \end{aligned} \quad (37)$$

We next move  $\exists x_1\exists x_2\exists z$  to the left, obtaining

$$\begin{aligned} &\exists x_1\exists x_2\exists z\exists\Phi[R(x_1, x_2)\wedge\Phi(x_1)\wedge\Phi(x_2)\wedge \\ &\quad \forall y(\neg\Phi(y)\vee P(y))\wedge P(z)\wedge\neg\Phi(z)]. \end{aligned} \quad (38)$$

**Preparation for the Ackermann lemma.** Applying Proposition 1.1 (8) and some standard equivalences, we replace (38) by

$$\begin{aligned} &\exists x_1\exists x_2\exists z\exists\Phi[\Phi(q)\vee(q\neq x_1\wedge q\neq x_2)) \\ &\quad \wedge(R(x_1, x_2)\wedge\forall y(\neg\Phi(y)\vee P(y))\wedge P(z)\wedge\neg\Phi(z))]. \end{aligned} \quad (39)$$

**Application of the Ackermann lemma.** The Ackermann lemma can now be applied to (39) resulting in

$$\begin{aligned} &\exists x_1\exists x_2\exists z[R(x_1, x_2)\wedge\forall y((y\neq x_1\wedge y\neq x_2)\vee P(y))\wedge \\ &\quad P(z)\wedge z\neq x_1\wedge z\neq x_2]. \end{aligned} \quad (40)$$

**Simplification.** Applying Proposition 1.1(11) to (39) results in

$$\begin{aligned} &\exists x_1\exists x_2\exists z[R(x_1, x_2)\wedge P(x_1)\wedge P(x_2) \\ &\quad \wedge P(z)\wedge z\neq x_1\wedge z\neq x_2]. \end{aligned} \quad (41)$$

Negating (41), we obtain

$$\begin{aligned} &\forall x_1\forall x_2\forall z[\neg R(x_1, x_2)\vee\neg P(x_1)\vee\neg P(x_2) \\ &\quad \vee\neg P(z)\vee z=x_1\vee z=x_2]. \end{aligned} \quad (42)$$

The first-order formula (42) is logically equivalent to the second-order formula (35). Consequently,

$$\begin{aligned} &Circ_{SO}(\Gamma; P) \equiv \Gamma(P)\wedge\forall x_1\forall x_2\forall z[\neg R(x_1, x_2) \\ &\quad \vee\neg P(x_1)\vee\neg P(x_2)\vee\neg P(z)\vee z=x_1\vee z=x_2]. \end{aligned} \quad (43)$$

Comparing (43) with (29), it is easily observed that not only is there a difference in the size of the formulas, but the output appears to make more sense relative to the minimization policy.

## 6 Conclusion

In this paper, we have presented a general algorithm which transforms second-order formulas into logically equivalent first-order formulas for a large class of second-order formulas. The algorithm has been shown to have a number of attractive properties, including a potentially wide area for practical application. To support this claim, we have provided a detailed description of the algorithms application to the reduction of circumscription axioms. In addition, we have shown that the algorithm, in its general form, provably subsumes nearly all existing results concerning the reduction of circumscription axioms. In contrast to previous results, the algorithm is more constructive in the sense that it provides a step-by-step method for transforming a formula and is guaranteed to terminate.

## References

- [Ackermann, 1935] W. Ackermann. Untersuchungen über das eliminationsproblem der mathematischen logik. *Mathematische Annalen*, 110:390–413, 1935.
- [Ackermann, 1954] W. Ackermann. *Solvable Cases of the Decision Problem*. North-Holland, Amsterdam, 1954.
- [Bentham, 1983] J. Van Bentham. *Modal Logic and Classical Logic*. Bibliopolis, Napoli, 1983.
- [Bentham, 1984] J. Van Bentham. Correspondence theory. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, volume 2, pages 167–247. D. Reidel Pub. Co., 1984.
- [Doherty et al., 1994] P. Doherty, W. Łukaszewicz, and A. Szalas. Computing circumscription revisited: A reduction algorithm. Technical Report LiTH-IDA-R-94-42, Linköping University, 1994. URL: <http://www.ida.liu.se/labs/rkllab/people/patdo/>.
- [Gabbay and Ohlbach, 1992] D. Gabbay and H. J. Ohlbach. Quantifier elimination in second-order predicate logic. Technical Report MPI-I-92-231, Max-Planck Institut für Informatik, 1992.
- [Gelfond and Lifschitz, 1989] M. Gelfond and V. Lifschitz. Compiling circumscriptive theories into logic programs. In *Proc. 2nd Int'l Workshop on Non-Monotonic Reasoning*, volume 346 of *Lecture Notes in Artificial Intelligence*, pages 74–99, 1989.
- [Ginsberg, 1989] M. L. Ginsberg. A circumscriptive theorem prover. *Artificial Intelligence*, 39:209–230, 1989.
- [Kolaitis and Papadimitriou, 1988] P. Kolaitis and C. Papadimitriou. Some computational aspects of circumscription. In *Proc. AAAI-88, St. Paul, MN*, pages 465–469, 1988.
- [Lifschitz, 1985] V. Lifschitz. Computing circumscription. In *Proc. 9th IJCAI, Los Angeles, CA*, pages 121–127, 1985.
- [Lifschitz, 1988] V. Lifschitz. Pointwise circumscription. In M. Ginsberg, editor, *Readings in Non-Monotonic Reasoning*, pages 179–193. Morgan Kaufmann Publishers, Palo Alto, CA, 1988.
- [Lifschitz, 1994] V. Lifschitz. Circumscription. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 3, pages 297–352. Clarendon Press, Oxford, 1994.
- [Przymusiński, 1991] T. Przymusiński. An algorithm to compute circumscription. *Artificial Intelligence*, 38:49–73, 1991.
- [Rabinov, 1989] A. Rabinov. A generalization of collapsible cases of circumscription. *Artificial Intelligence*, 38:111–117, 1989.
- [Szalas, 1993] A. Szalas. On the correspondence between modal and classical logic: an automated approach. *Journal of Logic and Computation*, pages 605–620, 1993.