ELIMINATION OF PREDICATE QUANTIFIERS

ANDREAS NONNENGART
Max-Planck-Institut für Informatik
Im Stadtwald
D-66123 Saarbrücken

HANS JÜRGEN OHLBACH
Department of Computing
Imperial College of Science, Technology and Medicine
180 Queen's Gate
London SW7 2BZ

AND

ANDRZEJ SZALAS
Institute of Informatics
University of Warsaw
ul. Banacha 2
02-097 Warsaw

1 Introduction

Formulae of higher-order predicate logic are difficult to handle with automated inference systems. Some of these formulae, however, are equivalent to formulae of first-order predicate logic or even propositional logic. For example the formula of second-order predicate logic $\exists P\ (P \wedge \neg P)$ is trivially equivalent to the propositional constant *false*. In applications where formulae of higher-order predicate logic occur naturally it is very useful to determine whether the given formula is in fact equivalent to a simpler formula of first-order or propositional logic. Typical applications where this occurs are predicate minimization by circumscription, correspondence theory in non-classical logics, and simple versions of set theory. In these areas we are faced with formulae of second-order predicate logic with existentially or universally quantified predicate variables and we want to simplify them by computing equivalent first-order formulae.

In general this problem is not even semi-decidable, so no complete quantifier elimination algorithms for predicate quantifiers can exist. To illustrate the complexity of the problem, consider the following formula:

$$\exists C \left(\begin{array}{l} \forall x\ C(x,R) \vee C(x,Y) \vee C(x,G) \wedge \\ \forall x \forall y \forall z\ (C(x,y) \wedge C(x,z)) \Rightarrow y = z \wedge \\ \forall x \forall y\ E(x,y) \Rightarrow \neg \exists z\ (C(x,z) \wedge C(y,z)) \end{array} \right) \tag{1}$$

where R, Y, G are constant symbols. If $C(x,y)$ is interpreted as 'node x in a graph has colour y' and $E(x,y)$ as 'y is adjacent to x in a graph' then this formula expressess the graph 3-colourability problem. The first conjunct states that each node is coloured with one of the three colours. The second conjunct says that each node has at most one colour and finally the last conjunct requires adjacent nodes to have different colours. If there existed a sound algorithm which in this case could eliminate the predicate C and reduce the formula to something first-order we had a proof that $P = NP$. Namely, if the algorithm would eliminate the C and compute an equivalent first-order formula ϕ in terms of the predicates E and $=$, we had a polynomial algorithm[1] for deciding whether a graph can be coloured with three colours or not, and thus proved $P = NP$.

Although this shows that one's expectations should not be too high, some of the proposed algorithms are quite powerful and can solve hard problems in the areas mentioned in the beginning.

Wilhelm Ackermann has investigated the quantifier elimination problem in the following form: Given a formula $\exists P\ \Phi[P]$ with a predicate variable P and a first-order formula Φ containing P in predicate position, find a first-order formula Φ' not containing P anymore such that $(\exists P\ \Phi[P]) \equiv \Phi'$. The corresponding problem with a *universally* quantified predicate variable can be reduced to the problem with existentially quantified predicate variable by negating the formula, eliminating P and then negating the result. This way, formulae with arbitrary prefixes of either universally or existentially quantified predicates, but not mixed ones, can be treated.

Ackermann [1, 2] developed two different methods for finding such Φ', the first one is essentially a generalization of the Modus Ponens inference rule, and the second one can be seen as a simplified version of the resolution principle. This approach exploits the fact that a predicate P occurring in

[1] ϕ would specify a property of the graph which is necessary and sufficient for the graph to be colourable. Thus in order to decide this one would have to check whether the graph is a model for ϕ. An algorithm for doing this is: Compute the clause normal form for ϕ and test for each clause whether it is satisfied by the graph. If the clause has, say n variable symbols and the graph has m nodes then m^n instances of the clause need to be tested. Since the number of clauses is fixed and the maximal number of variables in the clauses is bounded, this is a polynomial algorithm. It is in fact linear in terms of the number m of nodes.

a subset Γ' of some set Γ of formulae is not needed anymore if Γ contains already all the consequences of the formulae with P [10]. This principle itself is not yet very useful because usually there are infinitely many consequences. It turns out, however, that a subset of the set of consequences of the formulae with P which allows us to derive the full set of consequences is sufficient. The set of resolvents with P is such a sufficient subset, and this is the kernel of Ackermann's second approach.

In recent years Ackermann's two approaches have been rediscovered. Dov Gabbay and various other authors have extended and refined the method and turned it into implementable and in fact implemented algorithms. In this chapter we give an overview on the different approaches and discuss some applications.

2 Ackermann's Quantifier Elimination Methods

We briefly sketch Ackermann's two quantifier elimination methods.

2.1 Direct Elimination of Quantifiers

We say that a formula Φ is *positive* w.r.t. a predicate P iff there is no occurrence of $\neg P$ in the negation normal form of Φ. Dually, we say that Φ is *negative* w.r.t. P iff every of its occurrences in the negation normal form of Φ is of the form $\neg P$.

The following lemma was proved by Ackermann in [1] and can also be found in Szałas [21].

LEMMA 2.1 *Let P be a predicate variable and let Φ and $\Psi(P)$ be first-order formulae such that $\Psi(P)$ is positive w.r.t. P and Φ contains no occurrences of P at all. Then*

$$\exists P\, \forall \overline{x}\ (\neg P(\overline{x}) \vee \Phi) \wedge \Psi(P) \quad \equiv \quad \Psi\left[P(\overline{\alpha}) \leftarrow [\Phi]^{\overline{x}}_{\overline{\alpha}}\right]$$

and similarly if the sign of P is switched and Ψ is negative w.r.t. P. The right hand formula is to be read as: Every occurrence of P in Ψ is to be replaced by Φ, where the actual argument of P, say $\overline{\alpha}$, replaces the variables of $\overline{x}$ in Φ (and the bound variables are renamed if necessary). ■

Hence, if the second-order formula under consideration has the syntactic form of the left-hand-side of the equivalence given in Lemma 2.1 then this lemma can immediately be applied for the elimination of P.

Let us illustrate the application of the method with a small example from modal logic. The standard translation of the modal logic T-axiom $\Box p \Rightarrow p$ is

$$\forall P\, \forall w\ ((\forall x\ R(w,x) \Rightarrow P(x)) \Rightarrow P(w)). \tag{2}$$

The negation of this formula is equivalent to

$$\exists w\ \exists P\ \forall x\ \ (P(x) \vee \neg R(w,x)) \wedge \neg P(w). \tag{3}$$

The $\exists P \ldots$ part of the formula has the right syntactic structure for lemma 2.1 with exchanged signs and with $\Phi \equiv \neg R(w,x)$ and $\Psi \equiv P(w)$. Replacing $P(w)$ with Φ^x_w yields $\neg R(w,w)$. Thus, (3) is equivalent to $\exists w\ \neg R(w,w)$ and therefore (2) is equivalent to $\forall w\ R(w,w)$.

As we have seen, the syntactic form required in Lemma 2.1 is usually not given initially and we are therefore forced to apply some well-known equivalence preserving transformations of classical logic in order to obtain this form. The following one was found particularly useful in [1]:

$$P(\overline{x})\ \equiv \forall \overline{y}\ (P(\overline{y}) \vee \overline{x} \neq \overline{y}).$$

This technique was substantially strengthened in Doherty, Łukaszewicz, Szałas [3] by applying the following equivalence:

$$P(\overline{x_1}) \vee \cdots \vee P(\overline{x_n})\ \equiv \exists \overline{z}\ [(\overline{z} = \overline{x_1} \vee \cdots \vee \overline{z} = \overline{x_n}) \wedge P(\overline{z})] \tag{4}$$

where $\overline{z} = \overline{x_n}$ denotes a componentwise conjunction. The following extended purity deletion rule, formulated in Szałas [21], is also useful:

> if there is a predicate Q among the list of predicates to be eliminated such that Q occurs with mixed sign in some clauses and either only with positive or only with negative sign in the other clauses then all clauses containing Q are deleted. For example in the two clauses $\neg Q(x), Q(f(x))$ and $Q(a)$ there is no clause containing Q only with negative sign. If these are the only clauses with Q, they can be deleted. (Since Q is existentially quantified, a model making Q true everywhere satisfies the clauses.)

Other quite well-known transformation techniques turn out to be useful as well as there are, e.g., the transformation into conjunctive normal form, renaming, and the following second-order Skolemization that preserves equivalence of formulae and allows us to eliminate existential quantifiers (similar formulation can be found, e.g., in van Benthem [23]):

$$\forall \overline{x}\ \exists y\ \Phi \quad \equiv \quad \exists f\ \forall \overline{x}\ \Phi^{\,y}_{f(\overline{x})}$$

Second-order Skolemization may make Lemma 2.1 applicable, thus eliminating P, but this way we end up again with a second-order formula. Sometimes, however, it is possible to do the transformation and then turn the Skolem-function again into an existential quantifier ("un-Skolemization").

An algorithm for eliminating predicate quantifiers based on Lemma 2.1 was defined in Szałas [21]. This algorithm was later developed and strengthened in [3]. The strengthened version of the algorithm, called DLS, is sketched in Section 2.3.

One of the major restrictions in Lemma 2.1 is that it does not allow disjunctions which contain both, positive and negative occurrences of the predicate symbol to be eliminated. Ackermann found a way to overcome this problem.

2.2 The Resolution Based Approach

On page 401 of Ackermann [1] there is the definition of a *contraction operation*

$$\mathcal{A}^{x_1,\ldots,x_n,z}_{y_1,\ldots,y_m} \wedge \mathcal{B}^{p_1,\ldots,p_n}_{q_1,\ldots,q_l,z} \rightarrow \mathcal{A}^{x_1,\ldots,x_n}_{y_1,\ldots,y_m} \vee \mathcal{B}^{p_1,\ldots,p_n}_{q_1,\ldots,q_l},$$

where the subscripts y_i stand for $P(y_i)$ and the superscripts x_i stand for $\neg P(x_i)$ in a clause containing also literals $\mathcal{A}$ or $\mathcal{B}$ respectively. Thus, contraction on z actually means resolution between $P(z)$ and $\neg P(z)$. The variables can and must of course be renamed appropriately for making a contraction step possible.

Ackermann proved that, given a formula $\exists P\ \Phi$, where P is a one-place predicate variable and Φ is a set of clauses written in the above form, the P-free subset of the conjunction of all contractions is equivalent to $\exists P\ \Phi$. This holds even if this conjunction is infinite. As an example, consider the formula

$$\exists P\ P(a) \wedge \neg P(b) \wedge \forall x \forall y\ \neg P(x) \vee P(y) \vee N(x,y).$$

In Ackermann's clause notation this is

$$(x \neq a)_x \wedge (x \neq b)^x \wedge N(x,y)^x_y.$$

We obtain an infinite set of contractions which simplifies to

$$\begin{gathered} a \neq b \wedge N(a,b) \wedge \\ (\forall v\ N(a,v) \vee N(v,b)) \wedge \\ (\forall v \forall w\ N(a,v) \vee N(v,w) \vee N(w,b)) \wedge \ldots \end{gathered}$$

Ackermann defined his resolution method for one-place predicate variables only. For n-place predicate variables he proposed a transformation which turns the n-place predicate variables into one-place predicate variables. This transformation, however, is complex and unnecessary. If one gives up the tensor notation Ackermann used and thinks in terms of resolution steps, a much more natural generalization to n-place predicate variables is possible (cf. Section 4).

2.3 The DLS Algorithm

The DLS algorithm was defined in Doherty, Łukaszewicz, Szałas [3]. It is a strengthened version of an algorithm given in Szałas [21]. The algorithm tries to transform the input formula into the form suitable for application of Lemma 2.1.

More precisely, the algorithm takes a formula of the form $\exists P\, \Phi$, where Φ is a first-order formula, as an input and returns its first-order equivalent or reports failure.[2] Of course, the algorithm can also be used for formulae of the form $\forall P\, \Phi$, since the latter formula is equivalent to $\neg\exists P\, \neg\Phi$. Thus, by repeating the algorithm one can deal with formulae containing many arbitrary second-order quantifiers.

The elimination algorithm consists of three phases: (1) preprocessing; (2) preparation for Lemma 2.1; (3) application of Lemma 2.1. 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 extended purity deletion rule (see Section 2.1) is applicable, it should be applied.

1. *Preprocessing.* The purpose of this phase is to transform the formula $\exists P\, \Phi$ into a form that separates positive and negative occurrences of the quantified predicate variable P. The form we want to obtain is

$$\exists \bar{x} \exists P[(\Phi_1(P) \wedge \Psi_1(P)) \vee \cdots \vee (\Phi_n(P) \wedge \Psi_n(P))],$$

where, for each $1 \leq i \leq n$, $\Phi_i(P)$ is positive w.r.t. P and $\Psi_i(P)$ is negative w.r.t. P. The steps of this phase are the following. (i) Eliminate the connectives $\Rightarrow$ 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 formulae. (ii) Move universal quantifiers to the right and existential quantifiers to the left, applying as long as possible the usual quantifier rules. (iii) In the matrix of the formula obtained so far, distribute all top-level conjunctions over the disjunctions that occur among their conjuncts. (iv) If the resulting formula is not in the required form, then report the failure of the algorithm. Otherwise replace the input formula by its equivalent given by

$$\exists \bar{x} (\exists P(\Phi_1(P) \wedge \Psi_1(P)) \vee \cdots \vee \exists P(\Phi_n(P) \wedge \Psi_n(P))).$$

[2] 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 formulae we consider is not an arithmetical notion (see, for instance van Benthem [23]).

Try to find first-order equivalent of the above formula by applying the next phases in the algorithm to each its disjunct 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 $\exists P(\Phi(P) \wedge \Psi(P))$, where $\Phi(P)$ (respectively, $\Psi(P)$) is positive (respectively, negative) w.r.t. P, into one of the forms required in Lemma 2.1. Both forms can always be obtained by using equivalences given in Section 2.1 and both transformations should be performed because none, one or both forms may require Skolemization. Un-Skolemization, 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.
3. *Application of the Ackermann Lemma.* The goal of this phase is to eliminate the second-order quantification over P, by applying Lemma 2.1, and then to un-Skolemize the function variables possibly introduced. This latter step employs the second-order Skolemization equivalence.
4. *Simplification.* Generally, application of Lemma 2.1 in step (3) often involves the use of equivalences mentioned in Section 2.1 in the left to right direction. If so, the same equivalences may often be used after application in the right to left direction, substantially shortening the resulting formula.

Observe that the above algorithm can be used in connection with both Lemma 2.1 and Lemma 3.1. Lemma 3.1 should be applied in case DLS reports failure in step 1(iv).

The DLS algorithm was implemented in the University of Linköping, IDA, by Joakim Gustafsson [7]. It is accessible via World Wide Web (URL: http://www.ida. liu.se/labs/kplab/projects/dls/). The program can be executed remotely by filling out a html form in a WWW browser and clicking the "submit" button. The contents of the html form is then sent to a central server, which activates the program and returns the answer to the user. The system provides the user with a form for eliminating predicate quantifiers and a separate form for computing circumscription.

3 A Fixpoint Approach to Quantifier Elimination

Recently (see Nonnengart, Szałas [16]), the idea that lead to Lemma 2.1 has been generalized along the lines of Ackermann's observations as described in Section 2.2, i.e. infinite formulae are allowed and in order to be able to finitely represent these infinite formulae the original syntax got extended by fixpoint operators.

As an example consider again the second-order formula

$$\exists P\ P(a) \wedge \neg P(b) \wedge \forall x \forall y\ \neg P(x) \vee P(y) \vee N(x,y).$$

The problem we have if we try to apply Lemma 2.1 is that we are not able to separate the positive from the negative occurrences of P such that the requirements for the lemma are fulfilled. This is certainly not too surprising for otherwise we would be able to find an equivalent first-order formula which is impossible as Ackermann's result shows. The idea is therefore to describe these many conjunctive elements in a finite language and that with the help of fixpoint operators as follows:

$$\Big[\nu P(x).\ x \neq a \wedge \forall y\ P(y) \vee N(y,x)\Big]_b^x$$

where $[\Phi]_b^x$ is meant to express that every occurrence of x in Φ is to be replaced by b. This fixpoint formula indeed represents the infinite result obtained by Ackermann and why this is so is described below.

3.1 Fixpoint Calculus

Let $\mathcal{L}_\mathcal{I}$ be the classical first-order logic. In order to define the *fixpoint calculus* $\mathcal{L}_\mathcal{F}$ we extend $\mathcal{L}_\mathcal{I}$ by allowing the least fixpoint operator $\mu P.\Phi(P)$, where Φ is positive w.r.t. P. We abbreviate a formula of the form $\neg(\mu P.\neg\Phi(P))$ by $\nu P.\Phi(P)$. Similarly, if Φ is negative w.r.t. P we consider the formulae $\mu\overline{P}.\Phi(\overline{P})$ and $\nu\overline{P}.\Phi(\overline{P})$ respectively. It is sometimes convenient to indicate the individual variables which are bound by the fixpoint operators μ and ν. We write $\mu P(\bar{x})$ and $\nu P(\bar{x})$ to indicate that the tuple $\bar{x}$ of variables is bound by a fixpoint operator.

Let us now recall some useful well-known facts. The formulation we give is adapted to the particular problems we deal with. In particular, the partial order we consider is the following:

- the carrier is the set of formulae of $\mathcal{L}_\mathcal{F}/_\equiv$, where we do not distinguish between logically equivalent formulae (formally the carrier is the quotient set $\mathcal{L}_\mathcal{F}/_\equiv$; in order to simplify the considerations the equivalence classes and formulae are identified)
- the formulae are ordered by implication, i.e. Φ is less or equal to Ψ iff $\Phi \Rightarrow \Psi$ is a tautology.

Note that fixpoint operator formulae are also formulae. Thus the partial order we consider is complete in the sense that for every formula $\Phi(P)$ which is positive w.r.t. P the set $\{\Phi^i(\bot) \mid i \in \omega\}$ has a least upper bound. Every such formula is monotone and therefore we have by the Knaster &

Tarski fixpoint theorem that the fixpoints we consider are well defined. Moreover, the fixpoints have the following nice characterization[3]:

$$\mu P.\Phi(P) \;\equiv\; \bigvee_{\beta\in\alpha} \Phi^{\beta}(\bot)$$

for some ordinal α (the least such ordinal is called the *closure ordinal for* $\Phi(P)$). In the case of fixpoint formulae we deal with the closure ordinal is always ω. In fact, we always have the following equivalences (see also Lemma 3.1):

$$\begin{array}{rcl} \mu P.\Phi(P) & \equiv & \bigvee_{\beta\in\omega} \Phi^{\beta}(\bot) \\ \nu P.\Phi(P) & \equiv & \bigwedge_{\beta\in\omega} \Phi^{\beta}(\top) \end{array}$$

Note that $\mu P(\bar{x}).\Phi(P)$ is the least (w.r.t. the partial order defined above) formula $\Psi(\bar{x})$ such that

$$\Psi(\bar{x}) \;\equiv\; \Phi(P \leftarrow \Psi(\bar{x})).$$

Now let us come back to the formula

$$\Big[\nu P(x).\; x \neq a \wedge \forall y\; P(y) \vee N(y,x)\Big]_b^x .$$

In this case we have that $\Phi(P) \equiv x \neq a \wedge \forall y\; P(y) \vee N(y,x)$ and therefore

$$\begin{array}{rcl} \Phi^0(\top) & \equiv & \top \\ \Phi^1(\top) & \equiv & x \neq a \\ \Phi^2(\top) & \equiv & x \neq a \wedge \forall y\; y \neq a \vee N(y,x) \\ & \equiv & x \neq a \wedge N(a,x) \\ \Phi^2(\top) & \equiv & x \neq a \wedge \forall y\; (y \neq a \wedge N(a,y) \vee N(y,x)) \\ & \equiv & x \neq a \wedge N(a,x) \wedge \forall y\; (N(a,y) \vee N(y,x)) \\ \ldots & & \ldots \end{array}$$

such that the above fixpoint formula – with x replaced by b – becomes

$$a \neq b \wedge N(a,b) \wedge \forall y\; (N(a,y) \vee N(y,b)) \wedge \ldots$$

just as desired.

[3]Given $\Phi(P)$ we write $\Phi(A)$ to indicate that we want to consider Φ with each occurrence of P replaced by A. Thus $\Phi(\bot)$ is Φ with P replaced by $\bot$ (false) and $\Phi^{\beta}(\bot) \equiv \Phi(\Phi^{\beta-1}(\bot))$.

3.2 The Fixpoint Lemma

Let us now show how a fixpoint formula can be obtained from a given second-order formula. The following lemma, proved in Nonnengart, Szałas [16], is a generalization of Lemma 2.1 and introduces fixpoint formulae.

LEMMA 3.1 *If Φ and Ψ are positive w.r.t. P then the closure ordinal for $\Phi(P)$ is less than or equal to ω and*

$$\exists P\,\forall\overline{x}\;(\neg P(\overline{x}) \vee \Phi(P)) \wedge \Psi(P) \;\equiv\; \Psi\left[P(\overline{\alpha}) \leftarrow \left[\nu P(\overline{x}).\Phi(P)\right]^{\overline{x}}_{\overline{\alpha}}\right]$$

and similarly for the case where the sign of P is switched and Φ and Ψ are negative w.r.t. P. ■

Note the strong similarities between Lemma 2.1 and Lemma 3.1. In fact, it can quite easily be observed that this fixpoint result subsumes the former result as described in Lemma 2.1 for in case that Φ does not contain any P at all we have that $\nu P(\overline{y}).\Phi$ is equivalent to Φ. Hence Lemma 3.1 is a proper generalization of Lemma 2.1.

Again it is usually necessary to apply some equivalence preserving transformations in order to obtain a formula in the form required for applying Lemma 3.1. This can be done by the initial phases of the DLS algorithm (see Section 2.3). Recall that the syntactic form required in Lemma 2.1 cannot always be obtained. This is not the case anymore for Lemma 3.1 for any formula can be transformed into the form required provided second-order Skolemization is allowed. This Skolemization evidently cannot be avoided in general for otherwise every second-order formula could be transformed into a (possibly infinite) first-order formula. Nevertheless, the lemma can always be applied and returns some result which is usually a fixpoint formula and sometimes another second-order formula. Such fixpoints can be tried to be simplified then and in particular in case where the fixpoint is bounded a final first-order result can be found (see [4]).

A prototypic implementation of the fixpoint algorithm developed by M.J. Gabbay has been finished, but it is not yet available.

4 Quantifier Elimination by the SCAN Algorithm

4.1 The SCAN Algorithm

The SCAN algorithm[4] was proposed by Gabbay and Ohlbach [6] as a refinement of Ackermann's resolution method[5]. SCAN takes as input second-order formulae of the form

$$\alpha \equiv \exists P_1 \ldots \exists P_k \ \Phi$$

with existentially quantified predicate variables P_i and a first-order formula Φ. SCAN eliminates all predicate variables at once. The following three steps are performed by SCAN:

1. Φ is transformed into clause form.
2. All C-resolvents and C-factors with the predicate variables $P_1, \ldots, P_k$ are generated. *C-resolution* ('C' is short for constraint) is defined as follows:

$$\begin{array}{ll} P(s_1, \ldots s_n) \vee C & P(\ldots) \text{ and } \neg P(\ldots) \\ \neg P(t_1, \ldots, t_n) \vee D & \text{are the } \textit{resolution literals} \\ \hline C \vee D \vee s_1 \neq t_1 \vee \ldots \vee s_n \neq t_n & \end{array}$$

   and the *C-factorization* rule is defined analogously:

$$\frac{P(s_1, \ldots, s_n) \vee P(t_1, \ldots, t_n) \vee C}{P(s_1, \ldots, s_n) \vee C \vee s_1 \neq t_1 \vee \ldots \vee s_n \neq t_n}.$$

   When *all* resolvents and factors between a particular literal and the rest of the clause set have been generated (the literal is then said to be 'resolved away'), the clause containing this literal is deleted (this is called 'purity deletion'). If all clauses have been deleted this way, we know α is a tautology. If an empty clause is generated, we know α is inconsistent.
3. If step 2 terminates and the set of clauses is non-empty then the quantifiers for the Skolem functions are reconstructed.

The next example illustrates the various steps of the SCAN algorithm in detail. The input is:

$$\exists P \ \forall x \forall y \ \exists z \ (\neg P(a) \vee Q(x)) \wedge (P(y) \vee Q(a)) \wedge P(z).$$

[4]SCAN means "Synthesizing Correspondence Axioms for Normal Logics. The name was chosen before the general nature of the procedure was recognized.

[5]The SCAN algorithm was discovered independently. Only afterwards Andrzej Szałas found Ackermann's paper.

In the first step the clause form is computed:

$$\begin{array}{ll} C_1 & \neg P(a), Q(x) \\ C_2 & \quad P(y), Q(a) \\ C_3 & \quad P(f(x,y)) \end{array}$$

f is a Skolem function.

In the second step of SCAN we begin by choosing $\neg P(a)$ to be resolved away. The resolvent between C_1 and C_2 is $C_4 \equiv Q(x), Q(a)$ which is equivalent to $Q(a)$ (this is one of the equivalence preserving simplifications). The C-resolvent between C_1 and C_3 is $C_5 \equiv (a \neq f(x,y), Q(x))$. There are no more resolvents with $\neg P(a)$, so C_1 is deleted. We are left with the clauses

$$\begin{array}{ll} C_2 & P(y), Q(a) \\ C_3 & P(f(x,y)) \\ C_4 & Q(a) \\ C_5 & a \neq f(x,y), Q(x). \end{array}$$

Selecting the next two P-literals to be resolved away yields no new resolvents, so C_2 and C_3 can be deleted as well. All P-literals have now been eliminated. Restoring the quantifiers, we then get

$$\forall x\ \exists z\ Q(a) \wedge (a \neq z \vee Q(x))$$

as the final result (y is no longer needed).

There are two critical steps in the algorithm. First of all the C-resolution loop need not always terminate. This may but need not indicate that there is no first-order equivalent for the input formula. If the resolution eventually terminates the next critical step is the un-Skolemization. Since this is essentially a quantifier elimination problem for existentially quantified function variables, there is also no complete solution. The algorithm we usually use is heuristics based.

Preventing C-resolution from looping is a difficult control issue. Some equivalence preserving transformations on clause sets turned out to be quite useful. In the algorithm we have implemented each new resolvent can be tested whether it is implied by the non-parent clauses. In the affirmative case it is deleted even if more resolutions are possible. Another technique is the application of (4) for collapsing multiple occurrences of the predicate P in the same clause. In the graph colouring axiomatization (1) the latter method can successfully be applied. The clause normal form of (1) is

$$\begin{array}{l} C(x,R), C(x,Y), C(x,G) \\ \neg C(x,y), \neg C(x,z), x = z \\ \neg C(x,z), \neg C(y,z), \neg E(x,y) \end{array}$$

Given this to SCAN and asking it to eliminate C the resolution loops. If we however replace the first clause by the equivalent formula $\forall x\ \exists c\ (c = R \vee c = Y \vee c = G) \wedge C(x, c)$, whose clause normal form is

$$\begin{array}{l} c(x) = R, c(x) = Y, c(x) = G \\ C(x, c(x)) \end{array},$$

there is no problem anymore. The two successive resolutions with the second clause yields a tautology. The two successive resolutions with the third clause yields $c(x) \neq c(y), \neg E(x, y)$. The result is now $\exists c\ (\forall x\ c(x) = R \vee c(x) = Y \vee c(x) = G) \wedge \forall x, y\ E(x, y) \Rightarrow c(x) \neq c(y)$ which is again second-order. It is just a reformulation of the original formula in terms of the colouring function c. As we have explained in the introduction, it would be quite surprising to get a better result.

4.2 The SCAN Program

The SCAN has been implemented as a modified version of the OTTER theorem prover devolved by Bill McCune at Argonne National Laboratory [15]. The modifications were implemented by Thorsten Engel at the Max-Planck-Institute in Saarbrücken [5].

SCAN maintains two lists of formulae, the SOS list and the USABLE list. The SOS list contains the formulae with the predicates to be eliminated. The USABLE list may contain extra information which may hold in the given domain and which may be used to simplify formulae. That means if α is in the usable list, β is in the SOS list and $P_1, \ldots, P_n$ are the predicates to be eliminated, SCAN tries to compute a formula γ such that

$$\alpha \Rightarrow ((\exists P_1 \ldots P_n\ \beta) \equiv \gamma)$$

holds.

SCAN performs the following steps (if not deactivated by corresponding options).

1. All formulae are converted into conjunctive normal form (clause form).
2. Certain simplifications are performed, in particular elimination of tautologies and subsumed clauses. If possible, unit-deletion is applied. Unit-deletion is a resolution step followed by a subsumption step that deletes one parent clause. The net effect is the deletion of a literal from a clause. As an example, resolution between the unit clause $p(x)$ and the non-unit clause $\neg p(a), q$ yields the resolvent q, which subsumes the non-unit parent clause. The same effect is achieved by simply deleting $\neg p(a)$ from the parent clause. (Sometimes this brings about surprising effects, but it is a very useful equivalence preserving transformation.)

3. For each predicate symbol P in the list of predicates to be eliminated:
   - A clause C containing P is chosen from the SOS list.
   - All C-factors are generated.
   - A literal L in C containing P is chosen and all C-resolvents with this literal and other clauses in the SOS list are created.

     Each resolvent (i) is simplified by means of the unit-deletion rule, (ii) is deleted if it is a tautology, (iii) is deleted if it is subsumed by other clauses, (iv) is deleted if it is implied by the other non-parent clauses and the USABLE list (in order to check this, the SCAN process is forked and operates for some time in the OTTER theorem proving mode), (v) causes the deletion of other subsumed clauses.
   - Once the literal L is 'worked off', the clause C is deleted.
   - Apply the extended purity deletion rule (see Section 2.1), if possible.
4. If the previous steps terminate (which cannot be guaranteed) then the resulting clause set is checked for redundancy. To this end, an attempt is made to prove this clause from the other clauses (again by forking the process and running it in a theorem proving mode). Clauses which can be proved from other clauses in a certain time are deleted.
5. Finally the remaining clause set is un-Skolemized, if possible. Reconstructing existential quantifiers for Skolem functions is in fact a quantifier elimination operation for second-order formulae with existentially quantified function variables, so the un-Skolemization algorithm can be used as a quantifier elimination algorithm for function variables. There are three possible outcomes of the un-Skolemization routine:
   (i) it may generate a normal first-order formula,
   (ii) it may simply tell us that un-Skolemization is not possible, or
   (iii) it may generate an (again second-order) parallel Henkin quantifier. For example the un-Skolemized version of the clause $P(x, y, f(x)), Q(x, y, g(y))$ is

$$\begin{bmatrix} \forall x & \exists u \\ \forall y & \exists v \end{bmatrix} P(x, y, u) \vee Q(x, y, v).$$

As already mentioned, SCAN may not terminate and go on generating C-resolvents forever. If this happens, two measures may help. Sometimes changing the order of the list of predicates to be eliminated may cause the system to terminate. As an example, consider the clauses

$$\begin{array}{ll} C_1 & \neg P(x), P(f(x)), Q(x) \\ C_2 & P(a) \\ C_3 & Q(b) \\ C_4 & -P(b). \end{array}$$

Starting by resolving P triggers an infinite loop due to the self-resolving clause C_1. If we instead start with Q then C_1 is replaced by its resolvent $\neg P(b), P(f(b))$, which is no longer self-resolving. Eliminating P is now possible in a finite number of steps. (In principle an algorithm could recognize such situations and control the resolution process more intelligently, but this has not been implemented.)

The second measure for terminating the resolution loop is application dependent. Sometimes extra information is available which make resolvents redundant. Checking this and deleting redundant resolvents may terminate a loop. As a simple example for the application of this method consider the clauses

$$\begin{array}{ll} C_1 & \neg P(x), P(f(x)), Q(x) \\ C_2 & P(a) \\ C_3 & Q(a). \end{array}$$

where P is to be eliminated. The self resolving clause C_1 triggers an infinite loop. The first resolvent is $P(f(a)), Q(a)$, and the second resolvent is $P(f(f(a))), Q(a), Q(f(a))$. Further resolutions with P are possible. However already the first resolvent is subsumed by C_3 and can therefore be deleted. The loop is stopped if C_3 is available.

There is no exact characterization of the formulae for which SCAN terminates. It very much depends on the selection heuristics for the resolution steps.

The system is accessible via World Wide Web (URL: http://www.mpi-sb.mpg.de/guide/staff/ohlbach/scan/scan.html). The program can be executed remotely by filling out a html form in a WWW browser and clicking the "submit" button. The contents of the html form are then sent to a central server, which activates the program and returns the answer to the user. Besides the basic functionality for eliminating predicate quantifiers it contains two preprocessors which allow you to compute circumscription and to automate certain aspects of correspondence theory as explained below. So far, access to the program is not restricted.

5 Applications

5.1 Circumscription

Circumscription was proposed by John McCarthy as a logically simple and clear means of doing default reasoning. As an example consider the database consisting of the single entry $flies(Tweety)$. From this database you can of course prove that Tweety flies, but if you ask $flies(Woodstock)$? the database either replies with 'don't know' or responds brutally with 'no'. If you have evidence that your database is complete then the answer 'no' is justified, but in this case you conclude $\neg flies(Woodstock)$ from the fact that $flies(Woodstock)$ is not provable from the database. Since predicate logic is only semi-decidable, this is not a complete procedure. Moreover, there is no clear semantics which allows one to justify this step.

McCarthy's circumscription idea solves this problem on the semantic level. The idea is to axiomatize in a certain sense the information that "this is all I know about a particular predicate P", i.e. I want to consider only those interpretations for P in which $P(x)$ is true only for the absolutely minimum number of x necessary to satisfy the database. This *minimization* of the extension of predicate symbols is called circumscription. Unfortunately the formula which axiomatizes the minimized predicate is second-order. In the simplest case it is as follows:

$$circ(\Phi[P], P) \equiv \Phi[P] \wedge \forall P^* (\Phi[P^*] \wedge (P^* \rightarrow P)) \Rightarrow (P \rightarrow P^*)$$

where $\Phi[P]$ is an arbitrary first-order formula containing the predicate P which is to be minimized. $\Phi[P^*]$ is like Φ, but all occurrences of P are replaced by P^*. $P^* \rightarrow P$ is short for $\forall x_1 \ldots \forall x_n\ (P^*(x_1, \ldots, x_n) \Rightarrow P(x_1, \ldots, x_n))$. You can also have a list of predicates to be minimized simultaneously, in which case $P^* \rightarrow P$ stands for the conjunction of all these implications. As an example consider our little database above with the entry $flies(Tweety)$.

EXAMPLE 5.1 According to the definition of circumscription,

$$\begin{aligned} &circ(flies(Tweety), flies) \equiv flies(Tweety) \wedge \\ &\quad \forall flies^* (flies^*(Tweety) \wedge (\forall x\ flies^*(x) \Rightarrow flies(x))) \\ &\qquad \Rightarrow (\forall x\ flies(x) \Rightarrow flies^*(x)). \end{aligned}$$

This calls for a quantifier elimination procedure to eliminate the predicate $flies^*$. If we do this, we find as a result

$$\begin{aligned} &circ(flies(Tweety), flies) \\ &\qquad \equiv flies(Tweety) \wedge (\forall x\ flies(x) \Rightarrow x = Tweety), \end{aligned}$$

i.e. $Tweety$ is the only thing that flies. ∎

In an extended version of circumscription one can minimize certain predicates at the cost of certain other predicates which are allowed to vary. That is, if P are the predicates to be minimized and Z are the predicates allowed to vary then $circ(\Phi[P,Z],P,Z)$ is a formula from which one might be able to prove additional positive facts about Z which are not provable from $\Phi[P,Z]$. The circumscription formula for this version is

$$circ(\Phi[P,Z],P,Z) \\ \equiv \Phi[P,Z] \wedge \forall P^*, Z^* \ (\Phi[P^*,Z^*] \wedge (P^* \rightarrow P)) \rightarrow (P \rightarrow P^*)$$

The current implementation of SCAN contains a module for realizing this general version of circumscription by generating the circumscription formula according to the above schema and then applying SCAN to the second-order part.

Similarly, the DLS algorithm, sketched in Section 2.3, appeared quite powerful when applied to elimination of predicate quantifiers from circumscription formulae. In particular, it subsumes most known results about reducing circumscription (see [12, 13, 14, 17]). Moreover, in [3] it is proved substantially stronger than most of those results. Let us show an example of application of the DLS algorithm in reducing circumscription (see also [3]).

EXAMPLE 5.2 This is an example considered in Doherty, Łukaszewicz, Szałas [3]. It is a variant of the Vancouver example of Reiter. Rather than using the function *city* as Reiter does, we will use a relation $C(x,y)$ with suitable axioms. The intention is that $C(x,y)$ holds iff y is the home town of x. Let $\Phi(Ab,C)$ be the theory

$$[\forall x \forall y \forall z (\neg Ab(x) \wedge C(x,y) \wedge C(wife(x),z)) \Rightarrow y = z] \wedge \\ [\forall x \forall y \forall z \ (C(x,y) \wedge C(x,z)) \Rightarrow y = z].$$

The circumscription of $\Phi(Ab,C)$ with Ab minimized and C varied is

$$circ(\Phi(Ab,C); Ab; C) \\ \equiv \Phi(Ab,C) \wedge \forall P^* \forall Z^* [\Phi(P^*,Z^*) \wedge [P^* \leq Ab] \Rightarrow [Ab \leq P^*]],$$

where

$$\Phi(P^*,Z^*) \\ \equiv [\forall x \forall y \forall z \neg P^*(x) \wedge Z^*(x,y) \wedge Z^*(wife(x),z) \Rightarrow y = z] \wedge \\ [\forall x \forall y \forall z \ Z^*(x,y) \wedge Z^*(x,z) \Rightarrow y = z].$$

The DLS algorithm reduces the second-order part of circumscription:

$$\forall P^* \forall Z^* [\Phi(P^*,Z^*) \wedge [P^* \leq Ab] \Rightarrow [Ab \leq P^*]].$$

After two iterations (the first for reducing P^* and the second for reducing Z^*) one obtains the result $\forall t \neg Ab(t)$. Consequently,

$$circ(\Phi(Ab,C); Ab; C) \equiv \Phi(Ab,C) \wedge \forall t \neg Ab(t). \qquad \blacksquare$$

For more information about quantifier elimination and circumscription see Doherty, Łukaszewicz, Szałas [3], Lifschitz, 1994 [14] and Kartha, Lifschitz [11].

5.2 Correspondence Theory in Non-Classical Logics

The correspondence problem comes in non-classical logics, in particular in modal logics, as well as in certain algebras. In modal logics it is the problem of finding for a given Hilbert axiom a corresponding characteristic property of the underlying possible worlds structure (*frame properties*). For example the modal axiom $\Box p \Rightarrow p$ corresponds to reflexivity of the accessibility relation $\forall x\ R(x,x)$. As another example, $\Box p \Rightarrow \Box\Box p$ corresponds to the transitivity of the accessibility relation.

An algebraic version of the same problem turns up when considering Boolean algebras with operators. Jónsson and Tarski [9] have shown that under certain conditions the operators with binary relations in the same way as modal operators can be represented with accessibility relations (at a certain level of abstraction there is no longer any difference). The 'correspondence problem' here is to find the correspondences between additional axioms in terms of the operators on the one side and the underlying relation on the other.

We explain briefly the general construction. We start with an axiomatic presentation of an extension of propositional logic or just a Boolean algebra with extra operators. In the first case the Boolean algebra is obtained as the Lindenbaum-Tarski algebra whose elements are equivalence classes of provably (from the axioms and rules) equivalent formulae. Stone's famous representation theorem for Boolean Algebras maps a Boolean algebra isomorphically to a field of sets. That means for an element x of the Boolean algebra there is an isomorphic image U_x consisting of all ultrafilters (or maximally consistent set of formulae in the logic case) containing x. The Boolean connectives $\wedge, \vee, \neg$ are mapped to the set functions $\cap, \cup, '$ defined for the complete and atomic set algebra consisting of the full powerset 2^U of the set U of all ultrafilters in the Boolean algebra.

The extra functions, for example a unary function f (think of it as the algebraic version of the modal $\Diamond$-operator) are so far only defined for those sets of ultrafilters U_x which are images of some element x in the Boolean algebra. In this case $f'(U_x) = U_{f(x)}$. How f' operates on arbitrary sets of ultrafilters is not defined. Jónsson and Tarski could show that in case f is *normal*, i.e. $f(0) = 0$ and f is *additive*, i.e. $f(x \vee y) = f(x) \vee f(y)$, which correspond to the necessitation rule and the K-axiom in modal logic, there is a proper extension f'' of f' to the full powerset. The definition is

$$w \in f''(U) \equiv \exists u\ u \in U \wedge \forall U_x\ u \in U_x \Rightarrow w \in U_f(x),$$

or, since $u \in U_x$ iff $x \in u$:

$$w \in f''(U) \equiv \exists u\ u \in U \wedge \forall x\ x \in u \Rightarrow f(x) \in w. \tag{5}$$

If we abbreviate $\forall x\ x \in u \Rightarrow f(x) \in w$ by $R(w,u)$ we get a shorter notation

$$w \in f''(U) \equiv \exists u\ R(w,u) \wedge u \in U$$

which is familiar from the Kripke semantics of the modal $\Diamond$-operator.

Although f'' is a proper extension of f' which means $f''(U_x) = f'(U_x)$ for the 'representable' sets U_x, this does not guarantee that f'' inherits all properties of f'. As a positive example, consider the property $\forall x\ x \leq f(x)$ which is an algebraic version of the modal T-axiom $P \Rightarrow \Diamond P$. In terms of the set representation this means $\forall U_x\ U_x \subseteq U_{f(x)}$, or alternatively

$$\forall x\ \forall w\ x \in w \Rightarrow f(x) \in w. \tag{6}$$

The question is now: Does this imply $\forall U\ U \subseteq f''(U)$? In fact it does. By the definition of f'' (5) we get $\forall U\ \forall w\ w \in U \Rightarrow \exists u\ u \in U \wedge \forall x\ x \in u \Rightarrow f(x) \in w$ which is implied by (6) (choose $u = w$).

Thus, $\forall x\ x \leq f(x)$ in the Boolean algebra implies $\forall U\ U \subseteq f''(U)$ in its set representation. So called *preservation theorems* give syntactic characterizations of properties which transfer to the full powerset algebra (see Sahlqvist [19] and Jónsson [8].)

Quantifier elimination comes into the play if we want to express the given property of f'' in terms of the accessibility relation R introduced above. For example $\forall U\ U \subseteq f''(U)$ can be written as $\forall U\ \forall w\ w \in U \Rightarrow \exists u\ R(w,u) \wedge u \in U$. Since $\forall U$ quantifies over the whole powerset of some basic set, this is equivalent to quantifying over a predicate variable:
$\forall P\ \forall w\ P(w) \Rightarrow \exists u\ R(w,u) \wedge P(u)$ and this is equivalent to $\forall w\ R(w,w)$.

As we have seen, correspondence theory rises essentially two problems, showing that the given axiom continues to hold in the full powerset structure of the set representation (which in the logic case is nothing else than the canonical model) and then finding for the formulation in terms of the second-order variables a hopefully but not necessarily first-oder formulation in terms of the accessibility relation only. Our quantifier elimination algorithms automate to a certain extend this second step.

The following example shows an application of Lemma 2.1 to a correspondence theory problem (see also Szałas [21]).

EXAMPLE 5.3 Consider the Hilbert axiom $\Box p \Rightarrow \Box\Box p$. Similarly to the previous example we get the following corresponding second-order formula:

$$\forall P\ \forall u\ \neg\forall v(\neg R(u,v) \vee P(v)) \vee \forall v(\neg R(u,v) \vee \forall w(\neg R(v,w) \vee P(w)))$$

After negation and transformation to a form required in Lemma 2.1 (e.g., using the DLS algorithm) we get:

$$\exists u \exists P \forall v[P(v) \vee \neg R(u,v)] \wedge [\exists v\ R(u,v) \wedge \exists w(R(v,w) \wedge \neg P(w))].$$

The application of Lemma 2.1 results in:

$$\exists u \exists v\ R(u,v) \wedge \exists w(R(v,w) \wedge \neg R(u,w)).$$

This formula has to be unnegated, i.e. we get:

$$\forall u \forall v \neg R(u,v) \vee \forall w(\neg R(v,w) \vee R(u,w)),$$

which is equivalent to

$$\forall u \forall v \forall w(R(u,v) \Rightarrow (R(v,w) \Rightarrow R(u,w)),$$

i.e. to the transitivity of R, which is the desired frame property. ■

Consider the following example of Nonnengart, Szałas [16], where SCAN loops and DLS fails, but Lemma 3.1 can successfully be applied.

EXAMPLE 5.4 Consider the temporal logic formula

$$\Box(p \Rightarrow \circ p) \ \Rightarrow \ (p \Rightarrow \Box p).$$

where $\Box$ should be interpreted as *always* or *henceforth* and $\circ$ as *at the next moment of time*. This formula corresponds to the following second-order formula, where $R_\Box$ and $R_\circ$ are accessibility relations for modalities $\Box$ and $\circ$, respectively:

$$\forall P\ \forall u\ [\forall v\ (R_\Box(u,v) \Rightarrow (P(v) \Rightarrow \forall w\ (R_\circ(v,w) \Rightarrow P(w))))] \Rightarrow \\ [P(u) \Rightarrow \forall x\ (R_\Box(u,x) \Rightarrow P(x))].$$

After negating and transforming this formula into a form required in Lemma 3.1 we obtain:[6]

$$\exists u \exists x\ \exists P \left[\begin{array}{c} \forall w\ (P(w) \vee \\ (u \neq w \wedge \forall v\ (\neg R_\Box(u,v) \vee \neg R_\circ(v,w) \vee \neg P(v)))) \\ \wedge \\ R_\Box(u,x) \wedge \neg P(x) \end{array} \right]$$

After application of Lemma 3.1 we get:

$$\exists u \exists x\ R_\Box(u,x) \wedge \nu \neg P(x).[u \neq x \wedge \forall v\ (\neg R_\Box(u,v) \vee \neg R_\circ(v,x) \vee \neg P(v))].$$

Unnegating the formula results in:

$$\forall u \forall x\ R_\Box(u,x) \Rightarrow \mu P(x).[u = x \vee \exists v\ (R_\Box(u,v) \wedge R_\circ(v,x) \wedge P(v))].$$

Thus the initial formula is equivalent to:

[6] Observe that the positive and negative occurrences of P are not separated, thus Lemma 2.1 cannot be applied

$$\begin{aligned}\forall u \forall x R_{\Box}(u,x) \Rightarrow \quad & \{R_{\Box}(u,u) \wedge [u = x \vee R_{\circ}(u,x) \vee \\ & \textstyle\bigvee_{i\in\omega} \exists v_0 \ldots \exists v_i (R_{\Box}(u,v_0) \wedge \ldots \wedge R_{\Box}(u,v_i) \wedge \\ & R_{\circ}(u,v_0) \wedge R_{\circ}(v_0,v_1) \wedge \ldots \wedge R_{\circ}(v_{i-1},v_i) \wedge R_{\circ}(v_i,x))]\}.\end{aligned}$$

I.e. this formula states that $R_{\Box}$ is the reflexive and transitive closure of $R_{\circ}$, a property which is not expressible by means of classical logic but expressible by means of fixpoint logic. ■

Other applications of Lemma 2.1 to correspondence theory are described in Szałas [21, 22]. For more application examples of Lemma 3.1 see Nonnengart, Szałas [16].

6 Discussion of Other Approaches

6.1 Lifschitz Results

In the last ten years V. Lifschitz published a number of results on second-order quantifier elimination techniques in the context of circumscription (see Lifschitz [14]). Most of these results are subsumed by the DLS algorithm. The only exception is formulated in the following theorem of [14].

THEOREM 6.1 *Let $\Phi_1(P)$, $\Phi_2(P)$ be any first-order formulae such that $\Phi_1(P)$ is positive w.r.t. P and $\Phi_2(P)$ is negative w.r.t. P.*
Then $circ(\Phi_1(P) \wedge \Phi_2(P), P)$ is equivalent to a first-order sentence. ■

Similarly, some formulae that are reducible by Theorem 6.1 are not reducible by SCAN. This indicates the necessity of combining general quantifier elimination algorithms with particular, specialized solutions, like the one formulated above.

6.2 The Sahlqvist–van Benthem Algorithm

The Sahlqvist–van Benthem algorithm was motivated by the modal correspondence theory (see [19, 23]). It is based on the idea of finding "minimal" substitutions for the eliminated predicates. The key role is played here by second-order Sahlqvist formulae that reflect a particular class of modal axioms (for a general definition see de Rijke [18]).

The Sahlqvist-van Benthem algorithm is based on the following theorem:

THEOREM 6.2 *Let Φ be a Sahlqvist formula. Then Φ reduces to a first-order formula via suitable substitutions. Moreover, these substitutions can be effectively obtained from Φ.* ■

It can now be observed that negated Sahlqvist formulae are of the form suitable for the DLS algorithm. Moreover, the substitutions mentioned in

Theorem 6.2 are obtained by the DLS algorithm (some of them during applications of the Ackermann lemma and some of them during applications of the extended purity deletion rule).

Thus the Sahlqvist-van Benthem algorithm is subsumed by the DLS algorithm. Moreover, the subsumption is strict. Also the SCAN algorithm extends the Sahlqvist-van Benthem algorithm (see de Rijke [18]).

6.3 The Simmons Algorithm

An algorithm for eliminating second-order quantifiers in the context of modal correspondence theory is also given in Simmons [20]. The main idea of this algorithm is similar to that of Sahlqvist and van Benthem algorithm. It depends on looking for first-order equivalents by finding suitable substitutions of the eliminated predicates. However, in addition to the substitution technique, Simmons applies second-order Skolemization (see Section 2.1), which strengthens the Sahlqvist–van Benthem algorithm.

7 Summary

The development of algorithms for eliminating predicate variables has become a small but quite active area of research. For particular applications like circumscription and correspondence theory a number of methods and results had been known, but for the general case not much happened after Ackermann's early papers. Only after Gabbay and Ohlbach's first paper in the KR92 conference a few people got interested in this problem and began exploring different alternatives. Since the problem is not even semi-decidable there is much room for special methods and heuristics. An ideal implementation of a quantifier elimination procedure seems to be a kind of expert system which analyses the formula first and then applies the most appropriate method. Since new ideas and methods are coming up quite frequently it might however still be too early to start developing such a complicated system.

There is some indication that such a system would be quite useful. In areas where quantifier elimination plays a role, for example in correspondence theory people so far have only investigated cases with quite small formulae (which nevertheless may be tricky). The method employed was more or less nothing else than guessing and verifying. A program which can deal with really big and complex formulae can open the door to the investigation of systems which are currently out of reach.

References

1. Wilhelm Ackermann. Untersuchung über das Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 110:390–413, 1935.
2. Wilhelm Ackermann. Zum Eliminationsproblem der Mathematischen Logik. *Mathematische Annalen*, 111:61–63, 1935.
3. Patrick Doherty, Witold Lukaszewicz, and Andrzej Szałas. Computing Circumscription Revisited: A Reduction Algorithm. Technical Report LiTH-IDA-R-94-42, Institutionen för Datavetenskap, University of Linköping, 1994. A preliminary report published in *Proceedings* 4^{th} *IJCAI*, Morgan Kaufmann Pub. Inc., pp 1502–1508, 1995. To appear in *Journal of Automated Reasoning.*
4. Patrick Doherty, Witold Lukaszewicz, and Andrzej Szałas. A Characterization Result for Circumscribed Normal Logic Programs. Technical Report LiTH-IDA-R-95-20, Institutionen för Datavetenskap, University of Linköping, 1995. To appear in *Fundamenta Informaticae.*
5. Thorsten Engel. Elimination of Predicate and Function Quantifiers. Diploma Thesis. Max-Planck-Institut für Informatik, Saarbrücken, 1996.
6. Dov M. Gabbay and Hans Jürgen Ohlbach. Quantifier Elimination in Second-Order Predicate Logic. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *Principles of Knowledge Representation and Reasoning (KR92)*, 425–435. Morgan Kaufmann, 1992. Also published in the *South African Computer Journal*, 7:35–43, 1992.
7. Joakim Gustafsson. An Implementation and Optimization of an Algorithm for Reducing Formulae in Second-Order Logic. Technical Report LiTH-MAT-R-96-04, Dept. of Mathematics, Linköping University, Sweden, 1996.
8. Bjarni Jónsson. A Survey of Boolean Algebras with Operators. In Rosenberg and Sabidussi: Algebra and Orders, pp. 239–286, 1994.
9. Bjarni Jónsson and Alfred Tarski. Boolean Algebras with Operators, Part I. American Journal of Mathematics, vol. 73, pp 891–939, 1951.
10. Georg Kreisel and Jean-Louis Krivine. *Éléments de Logique Mathématique. Théorie des modèles.* Société Mathématique de France, 1966.
11. G. Neelakantan Kartha and Vladimir Lifschitz. A Simple Formalization of Actions Using Circumscription. In *Proceedings of IJCAI 95*, 1995.
12. Phokion G. Kolaitis and Christos H. Papadimitriou. Some Computational Aspects of Circumscription. In *AAAI-88: Proceedings of the 7th National Conference on Artificial Intelligence*, 465–469, 1988.
13. Vladimir Lifschitz. Computing Circumscription. In *Proceedings of the 9th Int'l Joint Conference on Artificial Intelligence*, volume 1, pages 121–127, 1985.
14. Vladimir Lifschitz. Circumscription. In D.M. Gabbay, C.J. Hogger, J.A. Robinson editors, *Handbook of Logic in Artifficial Intelligence and Logic Programming*, vol. 3, Clarendon Press, Oxford, 297–352, 1994.
15. William McCune. OTTER 2.0. In Mark Stickel, editor, *Proc. of* 10^{th} *International Conference on Automated Deduction, LNAI 449*, 663–664. Springer Verlag, 1990.
16. Andreas Nonnengart and Andrzej Szałas. A Fixpoint Approach to Second-Order Quantifier Elimination with Applications to Correspondence Theory. Technical Report MPI-I-95-2-007, Max-Planck-Institut für Informatik, Saarbrücken, 1995. To appear in E. Orlowska (ed.), *Logic at Work. Essays Dedicated to the Memory of Helena Rasiowa*, Kluwer.
17. A. Rabinov. A Generalization of Collapsible Cases of Circumscription. *Artificial Intelligence*, 38:111–117, 1989.
18. Maarten de Rijke. Extending Modal Logic. Ph.D. Thesis, Institute for Logic, Language and Computation, University of Amsterdam, 1993.
19. Henrik Sahlqvist. Completeness and Correspondence in the First and Second Order Semantics for Modal Logic. In S. Kanger (ed.), *Proc. 3rd Scandinavian Logic*

Symposium, North Holland, 110–143, 1975.

20. Harold Simmons. The Monotonous Elimination of Predicate Variables. *Journal of Logic and Computation*, 4:23–68, 1994.
21. Andrzej Szałas. On the Correspondence between Modal and Classical Logic: an Automated Approach. Technical Report MPI-I-92-209, Max-Planck-Institut für Informatik, Saarbrücken, 1992. Also published in *Journal of Logic and Computation*, 3:605–620, 1993.
22. Andrzej Szałas. On an Automated Translation of Modal Proof Rules into Formulas of the Classical Logic. *Journal of Applied Non-Classical Logics*, 4:119–127, 1994.
23. Johan van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, Naples, 1983.