MAX-PLANCK-INSTITUT FÜR INFORMATIK

Quantifier Elimination
in Second–Order Predicate Logic

Dov Gabbay & Hans Jürgen Ohlbach

MPI–I–94–231 July 1992

INFORMATIK

Im Stadtwald
D 66123 Saarbrücken
Germany

Author's Address

Dov Gabbay
Imperial College, Dept. of Computing
Queens Gate
London SWZ 2AZ, England

Hans Jürgen Ohlbach
Max–Planck–Institut für Informatik
Im Stadtwald
D-6600 Saarbrücken 11
F. R. Germany
ohlbach@mpi-sb.mpg.de

Publication Notes

This report also appears in the proceedings of the
Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92).

Acknowledgements

This work was stimulated considerably by many discussions with Luis Fariñas del Cerro and Andreas Herzig from Toulouse. We are grateful to Andrzej Szałas from Warsaw university who has digged out Ackermann's paper.

It was supported by the ESPRIT project 3125 MEDLAR, by the "Sonderforschungsbereich" 314, "Künstliche Intelligenz und wissensbasierte Systeme" of the German Research Council (DFG) and by the BMFT funded project LOGO.

The first author is a SERC Senior Research Fellow.

Abstract

An algorithm is presented which eliminates second–order quantifiers over predicate variables in formulae of type $\exists P_1, \ldots, P_n \psi$ where ψ is an arbitrary formula of first–order predicate logic. The resulting formula is equivalent to the original formula – if the algorithm terminates. The algorithm can for example be applied to do interpolation, to eliminate the second–order quantifiers in circumscription, to compute the correlations between structures and power structures, to compute semantic properties corresponding to Hilbert axioms in non classical logics and to compute model theoretic semantics for new logics. An earlier version of the paper has been published in [GO92b].

Key Words: Quantifier Elimination, Second–Order Predicate Logic, Circumscription, Interpolation.

Contents

1 Introduction

Automating reasoning for second–order predicate logic is much more difficult than for first–order predicate logic. Procedures that transform a given second–order formula into an equivalent first–order formula are therefore extremely useful. This transformation is of course not always possible, but there are important problem classes where the second–order formulae have a characteristic structure which allows for a transformation into an equivalent first–order formula.

Several methods have been developed for computing from a given second–order formula an equivalent first–order formula. These methods basically fall into two classes. The first class of algorithms computes or guesses suitable instantiations for the second–order predicate variables that are guaranteed to preserve equivalence [Ack35a, vB84, Sza92, Sim92]. The idea of the second class of algorithms is to compute sufficiently many consequences from the formulae containing the second–order variables and then keeping from the resulting set of formulae only those without the second–order variables [Ack35a, KK66, BGW92]. The algorithm we are going to present falls into this second class.

The structure of the formulae our algorithm can handle is $\exists P_1,\ldots,P_n\psi$ where the P_i are predicate variables for n–place predicates and ψ is an arbitrary first–order formula. Our algorithm essentially normalizes ψ into clause form and generates all (constraint–) resolvents of the clauses with the predicates P_i. It is shown that the subset of the generated clauses not containing predicates P_i (which may be infinite) is equivalent to the original formula. Since $\forall P_1,\ldots,P_n\psi \Leftrightarrow \neg\exists P_1,\ldots,P_n\neg\psi$ ($\Leftrightarrow$ is the equivalence sign), the algorithm can of course also handle universal quantifiers by reducing this case to the case with existential quantifiers. This algorithm is simple and it can be realized easily with existing theorem provers, for example OTTER.

Let us illustrate the SCAN algorithm[1] with some simple examples.
It is easy to see that

$$\exists P \begin{pmatrix} P \vee Q \\ \neg P \vee R \end{pmatrix} \quad \begin{array}{l}\text{is logically}\\ \text{equivalent to}\end{array} \quad Q \vee R$$

where $Q \vee R$ is just the resolvent between the two clauses on the left hand side. The "$\Rightarrow$"–direction follows from the fact that $Q \vee R$ as a resolvent is of course implied by the original formula. To see the "$\Leftarrow$"–direction, suppose Q is true in an interpretation. In this case the assigment for P, which is existentially quantified, can be chosen to be true also, making the existentially quantified formula true as a whole. If instead, R is true then P must be chosen to be false and again the left hand side formula is true. Thus, the existentially quantified P can be eliminated by just taking the single resolvent with P.

A slightly more complex example illustrates that in fact *all* (not redundant) resolvents with the second–order predicate have to be generated.

$$\exists P \begin{pmatrix} P \vee Q \\ \neg P \vee R \\ \neg P \vee S \end{pmatrix} \quad \begin{array}{l}\text{is logically}\\ \text{equivalent to}\end{array} \quad \begin{pmatrix} Q \vee R \\ Q \vee S \end{pmatrix}$$

If Q is false in a model, it is necessary that R and S are both true in order to choose P such that all three clauses on the left hand side become true. Falsity of Q enforces truth of both R and S on the right hand side only if both resolvents are present.

In the presence of second–order predicates with arguments, the resolution rule has to be changed slightly, as the third example demonstrates.

$$\exists P \begin{pmatrix} P(a) \vee Q \\ \neg P(b) \vee R \end{pmatrix} \quad \begin{array}{l}\text{is logically}\\ \text{equivalent to}\end{array} \quad a = b \Rightarrow (Q \vee R)$$

Only in models of the right hand side where a and b are mapped to the same objects, it is necessary that one of Q or R must be true in order to satisfy the left hand side. If a and b are

[1]SCAN means 'Synthesizing Correspondence Axioms for Normal logics.' We developed this algorithm for a particular application and the name was chosen before we realized that it is applicable in a general context.The casual chain of the papers is the following: First SCAN was introduced in [GaOh92a]. This stimulated [Sza92]. Both papers then stimulated [Sim92].

interpreted differently, we may well choose both $P(a)$ and $\neg P(b)$ to be true. That means instead of unification, just a constraint for the arguments of the resolution literals has to be generated.

Further examples are given in the applications section. The SCAN algorithm is defined in the next section and its soundness is proved. In the third section we give a more detailed comparison with other approaches. Finally various applications of quantifier elimination are discussed. We conclude with a section on limitations of the algorithm and some implementation hints.

2 The SCAN Algorithm

The algorithm is defined and its soundness is proved.

Definition 2.1 (The SCAN Algorithm)
Input to SCAN is a formula $\alpha = \exists P_1, \ldots, P_n\, \psi$ with predicate variables $P_1, \ldots, P_n$ and an arbitrary first–order formula ψ.
Output of the SCAN — if it terminates — is a formula φ_α which is logically equivalent to α, but not containing the predicate variables $P_1, \ldots, P_n$.
SCAN performs the following three steps:

1. ψ is transformed into clause form using second order skolemization. That means the resulting formula has the form: $\exists P_1, \ldots, P_n \exists f_1, \ldots, f_n \psi'$ where the f_i are the Skolem functions and ψ' is a set of clauses. From the algorithm's point of view, the quantifier prefix can be ignored. Therefore ψ' is treated as an ordinary clause set with the usual Skolem constants and functions.

2. All C–resolvents and C–factors with the predicate variables $P_1, \ldots, P_n$ have to be generated. C–resolution ('C' for constraint) is defined as follows:

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

   and the C-factorization rule is defined analogously:

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

   Notice that only C-resolutions between different clauses are allowed (no self resolution). A C-resolution or C-factorization can be optimized by destructively resolving literals $x \neq t$ where the variable x does not occur in t with the reflexivity equation. C–resolution and C–factorization takes into account that second order quantifiers may well impose conditions on the interpretations which must be formulated in terms of equations and inequations.

   As soon as *all* resolvents and factors between a particular literal and the rest of the clause set have been generated (the literal is 'resolved away'), the clause containing this literal must be deleted (purity deletion). If all clauses are deleted this way, this means that α is a tautology.

   All equivalence preserving simplifications may be applied freely. These are for example:

   - Tautologous resolvents can be deleted.
   - Subsumed clauses can be deleted.
   - Subsumption factoring can be performed. Subsumption factoring means that a factor subsumes its parent clause. This may be realized by just deleting some literals. For example $Q(x), Q(a)$, where x is a variable, can be simplified to $Q(a)$.
   - Subsumption resolution can also be performed. Subsumption resolution means that a resolvent subsumes its parent clause, and this again may be realized by deleting some literals [OS91] (see also example 4.4). For example the resolvent between P, Q and $\neg P, Q, R$ is just Q, R such that $\neg P$ can be deleted from the clause. (An instance of this operation is realized as so called 'unit_deletion' in the OTTER theorem prover.)

If an empty clause is generated, this means that α is contradictory.

3. If the previous step terminates and there are still clauses left then reverse the skolemization. A method for reversing the skolemization in a set F of clauses is (1) to abstract all arguments of all occurrences of Skolem functions by variables, i.e. $f(s_1, \ldots, s_n)$ is replaced with $f(x_1, \ldots, x_n)$ and additional literals $x_i \neq s_i$ are added to the clause where the x_i are fresh variables and (2) to consistently rename all variables such that the arguments of all occurrences of the Skolem function are the same. If this is possible and $F[f(x_1, \ldots, x_n)]$ is the result then $\forall x_1, \ldots, x_n \exists y\ F[y]$ is the solution. This process is repeated for all Skolem functions.

   If it is not possible to rename the variables consistently, the only chance is to take parallel Henkin quantifiers [Hen61] (see example 4.5) or leave the second–order quantification. $\triangleleft$

The next example illustrates the different steps of the SCAN algorithm in detail. The input is: $\exists P\ \forall x, y\ \exists z\ (\neg P(a) \vee Q(x)) \wedge (P(y) \vee Q(a)) \wedge P(z)$. In the first step the clause form is to be 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. The second–order quantifier prefix is therefore $\exists P\ \exists f\ \forall x, y$. But this is only needed for the correctness proof below.

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 $C_4 = Q(x), Q(a)$ which is equivalent to $Q(a)$ (this is one of the equivalence preserving simplifications). The C-resolvent between C_1 and C_3 is $C_5 = (a \neq f(x,y), Q(x))$. There are no more resolvents with $\neg P(a)$. Therefore 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. Thus, C_2 and C_3 are simply to 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.)

Theorem 2.2 (Correctness of SCAN)
If SCAN terminates for a formula α then α is logically equivalent to $SCAN(\alpha)$

Proof: The formulae under consideration contain a prefix of second–order existential quantifiers over predicate and function variables as the only second–order component. In order to prove the equivalence we can therefore take a standard first–order Tarskian model theory augmented with assignments of n-ary relations to n-place predicate variables and n-ary functions to n-place function variables.

Since we use second order skolemization, clause form generation as well as reversing the skolemization are equivalence preserving. Adding a resolvent or a factor to a clause set is also equivalence preserving. Therefore the only critical step in the SCAN algorithm is the purity deletion rule.

Removing a clause cannot make (in an interpretation) true clause sets false. Therefore every interpretation satisfying the clause set before the deletion satisfies it also after the deletion.

What we are left with to prove is that an interpretation satisfying the clause set without the pure clause also satisfies the clause set with the pure clause. And this turns out to be the really hard part of the proof where we have to exploit the second order character of the problem. What has to be exploited is that the predicate P is existentially quantified and therefore its interpretation can be chosen appropriately.

Before we come to the proof for the general case, it is useful to make some conceptual simplifications.

- Exploiting the equivalence
  $(P(s_1,\ldots,s_n) \vee C) \Leftrightarrow (P(x_1,\ldots,x_n) \vee C \vee x_1 \neq s_1 \vee \ldots \vee x_n \neq s_n)$ it can be assumed that the predicate P which has been "resolved away" in the pure clause C has only variables as arguments.

- The proof for an n-place predicate is not different to the proof for a one–place predicate. Just read x in $P(x)$ as a vector of variables. W.l.o.g we assume therefore that P is a one–place predicate.

- Since purity deletion is done after all resolvents and factors with the pure literal are generated, it can be assumed w.l.o.g that there is only one resolution partner in the clause set. If there are n resolution partners in the clause set then all proof steps below can be repeated n times.

- The clauses containing no resolution partners do not contain complementary literals with the predicate P. They are not touched during the purity deletion process. In the sequel they can therefore be ignored.

- The variables in the clauses can be renamed such that different clauses *share* the same variables.

There are two cases which have to be distinguished. The first case is that the predicate P occurs in the pure clause C only with one sign, either positively or negatively. The second case is that P occurs with both signs, i.e. C is self resolving. This is the case where $SCAN$ may loop.
Let us now consider the first case and w.l.o.g assume the predicate P occurs only positively in C. Thus, the situation before and after purity deletion looks as follows:

$$K = \left\{ \begin{array}{l} P(x), C(x) \qquad (=_{def}\ A) \\ Factors(A) \\ \underline{\neg P(x_1),\ldots,\neg P(x_n), D(x_1,\ldots,x_n)} \\ S(x_1),\ldots,S(x_n), D(x_1,\ldots,x_n) \\ \quad \vdots \end{array} \right. \quad \rightarrow \ K' = \left\{ \begin{array}{l} \\ Factors(A) \\ \underline{\neg P(x_1),\ldots,\neg P(x_n), D(x_1,\ldots,x_n)} \\ S(x_1),\ldots,S(x_n), D(x_1,\ldots,x_n) \\ \quad \vdots \end{array} \right.$$

where C and D denote the remaining literals. These literals may well contain additional variables. For the purpose of this proof these variables can be ignored. C and D may also contain additional *positive* literals with the predicate symbol P. $S(x_1),\ldots,S(x_n), D(x_1,\ldots,x_n)$ stands for the $2^n - 1$ resolvents which are possible between these two clauses. S denotes either $\neg P$ or C. For example if $n = 2$ there are three resolvents:

$$\begin{array}{rl} \neg P(x_1), & C(x_2), D(x_1,x_2) \\ C(x_1), & \neg P(x_2), D(x_1,x_2) \\ C(x_1), & C(x_2), D(x_1,x_2) \end{array}$$

If tautologies are automatically eliminated those resolvents which are either themselves tautologies or which are derived from tautologies are not present. We shall see that the factors of A are needed to take over the role of those clauses which are derived from tautologies. Subsumed clauses, however, may be deleted without any effect on the proof.
Take any interpretation $\Im$ satisfying K' and mapping the symbol P to a predicate $\mathcal{P}$ and the variable x to a domain element a. If $\Im$ satisfies $C(x)$ or $P(x)$ then $\Im$ satisfies K and we are done. If $\Im$ falsifies both we move to an interpretation $\Im'$ by changing the assignment of P to a predicate $\mathcal{P}'$ which is like $\mathcal{P}$ except that $\mathcal{P}'(a)$ is true. Then $\Im'$ satisfies $P(x), C(x)$. We have to show that $\Im'$ still satisfies all the other clauses.
Assume $\Im'$ maps the variables x_i to some a_i. Let J be the set of variables which are mapped to a, i.e. $x_i \in J$ if $a_i = a$. For these variables, the truth value of $\neg P(x_i)$ has changed from $true$ under $\Im$ to $false$ under $\Im'$. For all other variables nothing has changed. Therefore if $\neg P(x_j), j \notin J$ is true under $\Im$, it is still true under $\Im'$. In this case all clauses containing this literal are still true. Now suppose $\neg P(x_j), j \notin J$ are all false under $\Im$. If for simplicity we assume $J = \{x_1,\ldots,x_j\}$, there is a

clause $M = C(x_1), \ldots, C(x_j), \neg P(x_{j+1}), \ldots, \neg P(x_n),\ D(x_1, \ldots, x_n)$ among the resolvents. In this clause, all literals with predicate C and $\neg P$ are false under $\Im$. Since $\Im$ satisfies K', it must satisfy $D(x_1, \ldots, x_n)$. The interpretation of D has not changed. Therefore $\Im'$ satisfies $D(x_1, \ldots, x_n)$ as well. Thus, $\Im'$ satisfies all clauses in K.
It remains to be shown that in case of automatic tautology deletion the critical clause M is *not* deleted. If M would either itself be a tautology or derived from a tautology, the clause A would look like $A = P(x), P(y), C'(x, y)$. In this case the structure of M would be
$M = P(y_1), C'(x_1, y_1), \ldots, P(y_1), C'(x_j, y_j),$
$\neg P(x_{j+1}), \ldots, \neg P(x_n), D(x_1, \ldots, x_n)$. That means for example $P(y_1), C'(x_1, y_1)$ would be false for all assignments of y_1, in particular for $\Im(y_1) = a$. This assigment would also falsify the factor $P(x), C'(x, x)$ of clause A, which cannot be the case[2].
From this we conclude that both $\Im'$ and $\Im$ satisfy $\exists P\ K$.

The remaining case to be considered is the case where the clause C is self resolving. Schematically the situation looks as follows:

$$K = \left\{ \begin{array}{l} P(x), \neg P(y), C(x,y) \\ \neg P(x), D(x) \\ \hline \neg P(y), D(x), C(x,y) \\ \neg P(y), D(x), C(x,x'), C(x',y) \\ \quad \vdots \quad \text{(possibly infinitely many resolvents)} \end{array} \right. \quad \rightarrow \quad K' = \left\{ \begin{array}{l} \\ \neg P(x), D(x) \\ \hline \neg P(y), D(x), C(x,y) \\ \neg P(y), D(x), C(x,x'), C(x',y) \\ \quad \vdots \end{array} \right.$$

To simplify things, let us assume, neither $C(x, y)$ nor $D(x)$ contain negative occurrences of P. If in a given interpretation $\Im$ which maps x to some a_0, $D(x)$ is true, we can choose $P(x)$ to be true without further conflicts. If $D(x)$ is false, but $C(x, y)$ is true, where y is mapped to some a_1, $\Im$ satisfies K. If both $D(x)$ and $C(x, y)$ are false then the first resolvent enforces $\neg P(y)$ to be true. That means, $P(a_0)$ to be false enforces $P(a_1)$ to be false and therefore it has to be checked whether the first clause still remains true under the assignment $x \mapsto a_1$ etc. With the same arguments as in the base case this is proved with induction on n using the n^{th} resolvent. That means that in this case $\Im$ again satisfies $\exists P\ K$.
The case that $C(x)$ contains further negative literals with P means that there is another recursion loop which generates new branches of resolvents. Induction on the number of these further literals proves the statement.
The case that $D(x, y)$ also contains negative literals with P requires the integration of the arguments we used to prove the first case into the proof for the second case. This is technically complicated, but there are no further proof ideas needed.

Collecting everything together we can finally conclude $\alpha \Leftrightarrow SCAN(\alpha)$. $\triangleleft$

If the formula given to SCAN contains a cycle in the P-literals, SCAN may keep on producing infinitely many clauses. In some cases the size of the clauses remains finite. According to a result of Ackermann [Ack35a, Ack35b] which can be adapted to our case, the (possibly infinite) conjunction of the P-literal free clauses is a solution. It is one of the advantages of SCAN that in this case a subsumption test on the resolvents may terminate the process and thus compute a finite result whereas otherwise only infinite junk would be produced.

As an example where this happens, apply SCAN to the formula:

$$\exists P\ \forall x, y\ (P(x, y) \Rightarrow P(y, x)) \wedge (P(x, y) \Leftrightarrow Q(x, y)).$$

Since the symmetry clause $\neg P(x, y), P(y, x)$ contains only a trivial cycle which can easily be recognized, SCAN stops and returns as expected the symmetry of Q.

There is, however, no proof that SCAN stops in all cases where there is a finite solution. If this were the case then it would be decidable whether a theorem $\exists x\ P(x)$ has finitely many different proofs or not.

[2]This argument does not imply that only binary factors are needed. Before the factor itself is operated on, its factor has to be generated. That means all factors are needed.

As mentioned in the introduction, formulae with universal quantifiers have to be negated before giving them to SCAN and the result has to be negated again. The question may arise whether there is a possibility to treat universally quantified variables directly. Eliminating P from formulae $\forall P\ F[P]$ in some sense means factoring out the tautological part of $F[P]$. For example $(\forall P\ (P \vee \neg P) \wedge Q)$ is equivalent to Q, i.e. the P–part is tautologous. SCAN uses resolution as the basic operation, and resolution is sensitive to contradictions and not to tautologies. Therefore it is resolution which requires negation of the formula and elimination of the contradictious part.

In applications like circumscription, the structure of the formulae is $\forall P^*\ Q[P^*] \Rightarrow R[P^*]$ with a large $Q[P^*]$ and a small $R[P^*]$. In this case it is much more convenient to negate the formula yielding $\exists P^*\ Q[P^*] \wedge \neg R[P^*]$ because the big $Q[P^*]$ remains untouched.

There is a corollary derived from the proof of SCAN (2.2) which may be of some interest. The proof says that deleting a clause as soon as one literal is resolved away preserves equivalence. This may be exploited to eliminate only certain unwanted formulae. As an example, consider a PROLOG program containing a binary predicate P which is *symmetric*. Adding the symmetry clause to the program clauses PROLOG to loop. The corollary allows to eliminate the symmetry clause by generating all non redundant resolvents with the other PROLOG clauses. That means in this case that all clauses containing some $P(s,t)$ are duplicated with $P(t,s)$ replacing $P(s,t)$ in the copy. For all queries not containing P, the new PROLOG program is equivalent to the old one together with the symmetry clause.

3 Comparison with other Methods

Wilhelm Ackermann gave two procedures for eliminating existential quantifiers. Both eliminate only one quantifier at a time. The first one requires to bring the formula into a form

$$\exists P\ \forall x\ (A(x) \vee P(x)) \wedge \Delta[\neg P]$$

where $\Delta[\neg P]$ is a formula containing only negative occurrences of $P(x)$. The result is then $\Delta[A]$, i.e. all occurrences of $\neg P(x)$ are replaced with $A(x)$ in Δ. This method has difficulties in handling problems with clauses containing several occurrences of P. For example

$$\exists P\ \forall x,y\ (P(x,a) \vee P(a,x) \vee C(x)) \wedge (\neg P(y,a) \vee \neg P(a,y) \vee D(y))$$

falls into this problematic class. The SCAN–solution for this case, however, is simply $C(a) \vee D(a)$.

In its kernel the second method of Ackermann is actually quite similar to SCAN. Although his notation is very different to ours, it amounts to generating the conjunction of all P–free resolvents[3].

It is, however, also restricted to one–place predicates. Literals $P(s_1,\ldots,s_n)$ have therefore to be written as $First(x,s_1) \wedge \ldots \wedge nth(x,s_n) \Rightarrow P'(x)$ before this method can be applied. This transformation blows up the formulae considerably. In fact, if SCAN is reduced to formulae with one-place predicates where all arguments are variables and no further simplifications of the resolvents are applied, you obtain Ackermann's second method.

Since Ackermann does not use resolution as we do, it is very difficult to integrate optimization steps like subsumption deletion etc. That means that his method would not terminate for the above example with the symmetry clause. Therefore SCAN is much easier to handle and it behaves better than Ackermann's method in such pathological cases.

The idea of generating consequences of the formulae with P and then taking the subset of P–free formulae is actually the kernel of other approaches to this problem. For example a theorem in [KK66] says that it is the set of *all* consequences you have to take. This is of course too much to be of practical value. A minimal subset free of redundancies should be sufficient. We showed that

[3] Historical note: On page 401 of [Ack35a] there is the definition an 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 step to full resolution as we know it now is not that big.

the set of resolvents without tautologies and subsumed clauses is sufficient. Bachmair, Ganzinger and Waldmann [BGW92] have gone even one stop further. Their "hierarchical theorem proving" approach allows the formulation of redundancy criteria based on term orderings. Furthermore they have incorporated equality reasoning by superposition principles. This mechanism can be used to get rid of existentially quantified predicate *and function symbols*.

4 Applications of Quantifier Elimination

As mentioned in the introduction, the applications we have in mind are classes of problems where formulae with the structure $\exists P_1, \ldots, P_n \psi$ or $\forall P_1, \ldots, P_n \psi$ occur. This need not be second–order formulae in the first place. Even in standard first–order logic there might be useful applications. Suppose there is an axiomatization of something in terms of a predicate P and maybe some other predicates, and by some reason it is known that only theorems not containing P are to be proved from these axioms. That means

$$\begin{array}{lll} & Axioms(P) & \Rightarrow Theorem \\ \text{iff} & \forall P\ (Axioms(P) & \Rightarrow Theorem) \\ \text{iff} & (\exists P\ Axioms(P)) & \Rightarrow Theorem \\ \text{iff} & SCAN(\exists P\ Axioms(P)) & \Rightarrow Theorem \end{array}$$

i.e. SCAN can optimize the axioms with respect to the particular class of theorems not containing P.

Actually the situation is a special case of *interpolation*. We have

$$\begin{array}{lll} & \forall Q, R\ \varphi(Q,P) & \Rightarrow \psi(P,R) \\ \text{iff} & (\exists Q\ \varphi(Q,P)) & \Rightarrow \forall R\ \psi(P,R) \\ \text{iff} & (SCAN(\exists Q\ \varphi(Q,P)) & \Rightarrow \neg SCAN(\exists R \neg\psi(P,R)) \\ \text{iff} & \varphi'(P) & \Rightarrow \psi'(P) \end{array}$$

i.e. SCAN does interpolation.

4.1 Circumscription

Circumscription is a transformation of formulae proposed by John McCarthy [McC80] for the purpose of formalizing non–monotonic aspects of commonsense reasoning. 'Circumscribing' a predicate means in semantic terms minimizing the extension of that predicate.

If for example the formula $Bird(Tweety)$ is circumscribed with respect to the predicate $Bird$, we want to formalize that $Tweety$ is the only bird at all, i.e.
$Circ(Bird(Tweety), Bird) \Leftrightarrow (Bird(Tweety) \wedge \forall x\ Bird(x) \Rightarrow x = Tweety)$.

Analogously $Circ(\forall x\ Bird(x) \Rightarrow fly(x), fly) \Leftrightarrow \forall x\ Bird(x) \Leftrightarrow fly(x)$,
i.e. if birds fly then minimizing the extension of fly adds the information that *only* birds fly.

Unfortunately circumscription in its general definition requires a second–order quantifier. For circumscribing a single unary predicate at a time, the definition is as follows:

$$Circ(F(P), P) = F(P) \wedge \forall P^*\ (F(P^*) \wedge \forall y\ P^*(y) \Rightarrow P(y)) \Rightarrow (\forall x\ P(x) \Rightarrow P^*(x)).$$

So far only in special cases equivalent first–order formulae could be computed [Lif85]. SCAN offers a uniform way to compute first–order circumscriptions in most of the cases where there is one at all.

Example 4.1 (For Circumscription)
$Circ(P(a), P) = P(a) \wedge \forall P^*\ (P^*(a) \wedge \forall y\ P^*(y) \Rightarrow P(y)) \Rightarrow (\forall x\ P(x) \Rightarrow P^*(x))$.
In order to get rid of the second–order quantification, the $\forall P^* \ldots$–formula is negated to obtain a version with existential quantifier $\exists P^* \ldots$ which is a suitable input to SCAN. Clausifying the

negated formula yields

$$\begin{array}{ll}\exists P^* \ \exists x \ \forall y & P^*(a) \\ & \neg P^*(y), P(y) \\ & P(x) \\ & \neg P^*(x)\end{array}$$

Fortunately there is no Skolem function. Therefore we need not worry about skolemization and unskolemization. We just keep in mind that x is existentially quantified and therefore to be treated as a constant symbol.
Resolving P^* away yields

$$\begin{array}{ll}\exists x & P(a) \\ & P(x) \\ & x \neq a\end{array}$$

The result has to be negated again such that the final version is as expected:

$$Circ(P(a), P) \Leftrightarrow P(a) \wedge \forall x \ P(x) \Rightarrow x = a.$$

◁

Example 4.2 (For Circumscription)
$Circ(\forall x \ P(x) \Rightarrow Q(x), Q) =$
$\forall x \ P(x) \Rightarrow Q(x) \wedge \forall Q^* \ (\forall y \ P(y) \Rightarrow Q^*(y) \wedge \forall y \ Q^*(y) \Rightarrow Q(y)) \Rightarrow (\forall x \ Q(x) \Rightarrow Q^*(x))$.
Negation of the $\forall P^* \ldots$–formula and clausification yields

$$\begin{array}{ll}\exists Q^* \ \exists x \ \forall y & \neg P(y), Q^*(y) \\ & \neg Q^*(y), Q(y) \\ & Q(x) \\ & \neg Q^*(x)\end{array}$$

Resolving Q^* away yields

$$\begin{array}{ll}\exists x \ \forall y & \neg P(x) \\ & Q(x) \\ & \neg P(y), Q(y)\end{array}$$

Negation again yields: $(\forall y \ P(y) \Rightarrow Q(y)) \Rightarrow (\forall x \ Q(x) \Rightarrow P(x))$
Together with $\forall y \ P(y) \Rightarrow Q(y)$ this simplifies to $\forall y \ P(y) \Leftrightarrow Q(y)$. ◁

4.2 Power Structures

An *algebra* is a set together with some functions operating on this set. A *structure* is an algebra plus some additional relations between the elements of the algebra's carrier set. A *power structure* of a structure is again a structure. Its carrier set is just the powerset of the structure's carrier set and its functions and relations are obtained by lifting the structure's functions and relations to operate on sets instead of elements [Bri92]. For example a binary relation R can be lifted to a one-place function F:

$$F(X) = \{z \mid \exists x \in X \ R(x, z)\}$$

The *duality problem* is now to find the correspondences between the properties of the relation and the properties of the lifted function. For example transitivity of the binary relation corresponds to the property $F(F(X)) \subseteq F(X)$. Quite a number of theories in mathematics, logic, and computer science turn out to be instances of power structure constructions, duality theory, or correspondence theory respectively, in logic, or power domains in denotational semantics of non deterministic programs, just to name two of them.

With SCAN it is possible to do one direction and compute from the properties of the lifted function the properties of the underlying relation. For example in order to obtain transitivity from $F(F(X)) \subseteq F(X)$ we proceed as follows:
Applying the above definition of F in terms of R, $\forall X \ \forall z \ z \in F(F(X)) \Rightarrow z \in F(X)$
is rewritten to

$\forall X\ (\exists x\ x \in F(X) \wedge R(x,z)) \Rightarrow (\exists x\ x \in X \wedge R(x,z))$
and further to
$\forall X\ \forall z\ (\exists x\ (\exists x'\ x' \in X \wedge R(x',x)) \wedge R(x,z)) \Rightarrow (\exists x\ x \in X \wedge R(x,z))$
which can also be written as
$\forall X\ \forall z\ (\exists x\ (\exists x'\ X(x) \wedge R(x',x)) \wedge R(x,z)) \Rightarrow (\exists x\ X(x) \wedge R(x,z))$
and this is now a typical quantifier elimination problem. Negation, application of SCAN and negation again yields the transitivity of R (c.f. example 4.3).

The other direction, computing from the properties of the structure the properties of the power structure, turns out to be a formula synthesizing problem that can be solved with resolution theorem provers by constructively proving certain existentially quantified theorems. We shall report on this in a subsequent publication.

4.3 Correspondence Theory

Many non classical logics are characterized by a basic Hilbert calculus which corresponds to a model theoretic semantics in terms of possible worlds. Different variants of the logic manifest themselves by additional Hilbert axioms which correspond to particular properties of the possible worlds structure.

For example normal modal logic [Che80] is characterized by the Hilbert axioms and rules for predicate logic plus the K-axiom

$$(\Box P \wedge \Box(P \Rightarrow Q)) \Rightarrow \Box Q$$

and the necessitation rule

$$\frac{P}{\Box P}.$$

The semantics of this logic is Kripke's well known possible worlds semantics:

$$w \models \Box P \quad \text{iff} \quad \forall v\ \mathcal{R}(w,v) \Rightarrow v \models P$$

where $\mathcal{R}$ is the accessibility relation.

The correspondence problem in these logics is to find for a given additional Hilbert axiom an equivalent property of the accessibility relation [vB84]. For example the Hilbert axiom $\forall P\ \Box P \Rightarrow P$ corresponds to the reflexivity of the accessibility relation. Actually this is one of the instances of power structure constructions. The predicates can be identified with the set of worlds where they are true and the modal operators can be seen as functions operating on these sets of worlds. For mechanizing deduction in non classical logics it is very important to find these correspondences [Ohl91]. So far the method for finding the correspondences was mostly by intuition and the verification required complex proofs [vB84].

SCAN is the first algorithm which offers a method for computing the correspondences fully automatically. Moreover, since SCAN preserves equivalences, the computed correspondence axioms are *guaranteed to be complete* in the sense that a formula is derivable in the Hilbert calculus if and only if it is valid in the frames which are models of the computed correspondence axiom. The idea is as follows: We take the model theoretic semantics for the operators and translate the Hilbert axioms into predicate logic (cf. [Ohl91]).

For example $\forall P\ \Box P \Rightarrow P$ is translated into $\forall P\ (\forall w, v\ \mathcal{R}(w,v) \Rightarrow P(v)) \Rightarrow P(w)$. Application of SCAN to the negation of this formula and negating the result again yields the expected reflexivity axiom $\forall w\ \mathcal{R}(w,w)$.

Notice that this method is applicable as long as the semantics of the operators can be formalized with first–order predicate logic axioms. Moreover since SCAN preserves equivalence we automatically get soundness *and completeness* if the basic semantics is complete. That means the Hilbert calculus and the semantics together with the correspondence axioms computed by SCAN define the same logic.

Let us illustrate this application of SCAN with some more examples from modal logic.

In the sequel $H(P,v)$ means P holds in world v. For atomic P it is possible to write $P(v)$ instead of $H(P,v)$. The semantics of the modal operators are written as an equivalence in terms of the H–predicate and the accessibility relation. This equivalence is used as a rewrite rule for eliminating the operators. As usual, the $\Diamond$–operator is seen as an abbreviation for $\neg\Box\neg$.

Example 4.3 (Modal K4-Axiom)

Hilbert axiom:	$\forall P\ \Box P \Rightarrow \Box\Box P$
H-Formulation:	$\forall P\ \forall a\ H(\Box P \Rightarrow \Box\Box P, a)$
Semantics:	$\forall P\ \forall w\ H(\Box P, w) \Leftrightarrow (\forall v\ \mathcal{R}(w,v) \Rightarrow H(P,v))$
negated axiom:	$\exists P\ \exists a\ \neg H(\Box P \wedge \Diamond\Diamond\neg P, a)$

translated (i.e. Semantics applied as rewrite rule).

$$\exists P\ \exists a\ (\forall v\ \mathcal{R}(a,v) \Rightarrow P(v)) \wedge \exists b\ \mathcal{R}(a,b) \wedge \exists c\ \mathcal{R}(b,c) \wedge P(c)$$

clause form:	$\neg\mathcal{R}(a,v), P(v)$	
	$\mathcal{R}(a,b)$	
	$\mathcal{R}(b,c)$	(Quantifier prefix: $\exists a,b,c\ \forall v$)
	$\neg P(c)$	
P resolved away:	$\neg\mathcal{R}(a,c)$	
	$\mathcal{R}(a,b)$	
	$\mathcal{R}(b,c)$	
unskolemized:	$\exists a,b,c\ \neg\mathcal{R}(a,c) \wedge \mathcal{R}(a,b) \wedge \mathcal{R}(b,c)$	
negated:	$\forall a,b,c\ \mathcal{R}(a,b) \wedge \mathcal{R}(b,c) \Rightarrow \mathcal{R}(a,c)$	(transitivity)

$\triangleleft$

SCAN can also work with quantified versions of modal logic. The predicate $exists(w,x)$ expresses that the object x exists in the domain of the world w.

Example 4.4 (Barcan Formula)

Hilbert axiom:	$\forall P\ \Box\forall x\ P(x) \Rightarrow \forall x\ \Box P(x)$	
H-Formulation:	$\forall P\ \forall a\ H(\Box\forall x\ P(x) \Rightarrow \forall x\ \Box P(x), a)$	
semantics of $\forall$:	$w \models \forall x\ P(x)$ iff	for all a in the domain of w $w[x/a] \models P(x)$.
negated axiom:	$\exists P\ \exists a\ \neg H(\Box\forall x\ P(x) \wedge \exists x\ \Diamond\neg P(x), a)$	
translated:	$\exists P\ \exists a\ \forall v\ \mathcal{R}(a,v) \Rightarrow (\forall x\ exists(v,x) \Rightarrow P(v,x)) \wedge$	
	$\exists x\ exists(a,x) \wedge \exists b\ \mathcal{R}(a,b) \wedge \neg P(b,x)$	
clause form:	$\neg\mathcal{R}(a,v), \neg exists(v,x), P(v,x)$	
	$exists(a,c)$	
	$\mathcal{R}(a,b)$	
	$\neg P(b,c)$	
P resolved away:	$\mathcal{R}(a,b)$	
	$exists(a,c)$	
	$\neg exists(b,c)$	Here we have resolved with $\mathcal{R}(a,b)$.

This is an equivalence preserving simplification (subsumption resolution.)

unskolemized:	$\exists a,b,c\ \mathcal{R}(a,b) \wedge \neg exists(b,c) \wedge exists(a,c)$	
negated:	$\forall a,b\ \mathcal{R}(a,b) \Rightarrow (\forall c\ exists(a,c) \Rightarrow exists(b,c))$	(Increasing Domain)

$\triangleleft$

The next example is the McKinsey axiom. It does not correspond to a first–order definable property of the accessibility relation.

Example 4.5 (McKinsey Axiom)

Hilbert axiom: $\forall P\ \Box\Diamond P \Rightarrow \Diamond\Box P$
H-Formulation: $\forall P\ \forall a\ H(\Box\Diamond P \Rightarrow \Diamond\Box P, a)$

semantics: $\forall P\ \forall w\ H(P,w) \Leftrightarrow (\forall v\ \mathcal{R}(w,v) \Rightarrow H(P,v))$

negated axiom: $\exists P\ \exists a\ H(\Box\Diamond P \wedge \Box\Diamond\neg P, a)$
translated:
$$\exists a\ \forall x\ \mathcal{R}(a,x) \Rightarrow \exists y\ (\mathcal{R}(x,y) \wedge P(y)) \wedge \forall x\ \mathcal{R}(a,x) \Rightarrow \exists y\ (\mathcal{R}(x,y) \wedge \neg P(y))$$

clause form:
$$\neg\mathcal{R}(a,x), \mathcal{R}(x,f(x))$$
$$\neg\mathcal{R}(a,x), P(f(x))$$
$$\neg\mathcal{R}(a,y), \mathcal{R}(y,g(y))$$
$$\neg\mathcal{R}(a,y), \neg P(g(y)) \qquad \text{(Quantifier prefix: } \exists f, g\ \forall x, y)$$

P resolved away:
$$\neg\mathcal{R}(a,x), \mathcal{R}(x,f(x))$$
$$\neg\mathcal{R}(a,y), \mathcal{R}(y,g(y))$$
$$\neg\mathcal{R}(a,x), \neg\mathcal{R}(a,y), f(x) \neq g(y)$$

unskolemized with second–order Henkin quantifiers:
$$\exists a \begin{pmatrix} \exists f\ \forall x \\ \exists g\ \forall y \end{pmatrix} \begin{array}{l} (\mathcal{R}(a,x) \Rightarrow \mathcal{R}(x,f(x))) \wedge \\ (\mathcal{R}(a,y) \Rightarrow \mathcal{R}(y,g(y))) \wedge \\ (\neg\mathcal{R}(a,x) \vee \neg\mathcal{R}(a,y) \vee f(x) \neq g(y)) \end{array}$$

negated:
$$\forall a \begin{pmatrix} \forall f\ \exists x \\ \forall g\ \exists y \end{pmatrix} \begin{array}{l} ((\mathcal{R}(a,x) \Rightarrow \mathcal{R}(x,f(x))) \wedge \\ (\mathcal{R}(a,y) \Rightarrow \mathcal{R}(y,g(y)))) \Rightarrow \\ (\mathcal{R}(a,x) \wedge \mathcal{R}(a,y) \wedge f(x) = g(y)) \end{array}$$

$\triangleleft$

The reader may verify with SCAN that the axiom $\forall P\ \Diamond\Box P \Rightarrow \Box\Diamond P$ is equivalent to confluence of $\mathcal{R}$: $\forall a,b,c\ \mathcal{R}(a,b) \wedge \mathcal{R}(a,c) \Rightarrow \exists d\ \mathcal{R}(b,d) \wedge \mathcal{R}(c,d)$. Compared to the McKinsey axiom, slight changes in the sequence of the modal operators obviously may have dramatic effects.

Based on Ackermann's first elimination method Andrzej Szałas has developed an alternative algorithm for computing correspondence axioms in modal logic [Sza92]. This algorithm terminates and reports failure in cases where SCAN loops. This may be an advantage. It may also be a disadvantage because sometimes it is possible to recognize and exploit regular structures in the loop.

Inspired by the SCAN algorithm, Harold Simmons has recently proposed another method for computing correspondence axioms for modal logics [Sim92]. His method extends an idea of van Benthem where it is necessary to guess appropriate instantiations for universally quantified predicate variables [vB84]. Simmons figured out how to determine these instantiations without guessing.

4.4 Semantics for Hilbert Calculi

There are more and more applications of logic in areas other than mathematics and classical computer science. In particular AI approaches to modelling intelligent agents are a very interesting area for applications of logic. Existing logics, however, usually do not cover all aspects which are needed there (formal notions of knowledge, belief, actions, causality, probability, time etc. as well as all kinds of combinations). Therefore there is a need for supporting the development of new logics and the mechanization of these logics. In [GO92a] we shall present a methodology for developing mechanizations of logics defined via Hilbert calculi. The main problem here is to find a model theoretic semantics which serves as a basis for a translation of the formulae of the new logic into predicate logic. This can be done by starting with a very general and therefore very weak neighbourhood semantics and by proving certain key lemmata which enable the transition form a weak semantics to a stronger semantics. A version of neighbourhood semantics for an n-place connective C is:

$$w \models C(P_1, \ldots, P_n, w) \text{ iff } \mathcal{N}(w, \zeta(v_1, \ldots, v_n)\Psi_C(v_1 \models P_1, \ldots, v_n \models P_n)$$

which means that the set of all n-tuples $v_1, \ldots, v_n$ for which $\Psi_C(v_1 \models P_1, \ldots, v_n \models P_n)$ holds forms a neighbourhood of w. Ψ is a propositional function such as $\Rightarrow$, $\wedge$ etc. (For a binary causality operator, for example, one would choose $\Rightarrow$.)

A relational semantics for n-place connectives is

$$w \models C(P_1, \ldots, P_n, w) \text{ iff}(\forall v_1, \ldots, v_n\ R(w, v_1, \ldots, v_n) \text{ implies } \Psi_C(v_1 \models P_1, \ldots, v_n \models P_n)$$

where R_C is an $n+1$–ary accessibility relation.

Semantics of these kind can be used to translate the formulae of the new logic into predicate logic. Applied to Hilbert axioms, second–order formulae are obtained which can be processed by SCAN. The key lemmas for strengthening the semantics are: Closedness under intersection and supersets allows the transition from neighbourhood semantics to relational semantics. If the accessibility relation collapses to a point relation, relational semantics can be strengthened to predicate logic semantics.

We illustrate the idea with an example from Łukasiewicz. The implicational fragment of propositional logic can be axiomatized by modus ponens: from P and $P \to Q$ derive Q, and one more axiom $((P \to Q) \to R) \to ((R \to P) \to (S \to P))$

Assume the strongest semantics of the $\to$–connective (material implication) is not yet known, but we have an approximation in terms of possible worlds and a ternary relation:

$$x \models P \to Q \quad \text{iff} \quad \forall y, z\ \mathcal{R}(x, y, z) \text{ implies } (y \models P \text{ implies } z \models Q)$$

This semantics can be used to translate the above Hilbert axiom and modus ponens into predicate logic. For example the translation of modus ponens yields

$$\forall P, Q\ \forall x\ (P(x) \wedge (\forall y, z\ \mathcal{R}(x, y, z) \Rightarrow P(y) \Rightarrow Q(z)) \Rightarrow Q(x)$$

If we do this for the above axiom also, we get two second–order formulae from which SCAN can eliminate the quantifiers. The result is:

$$\begin{array}{l} \text{SCAN(Modus Ponens)} = \forall z\ \exists x, y\ \mathcal{R}(x, y, z) \\ \text{SCAN(Axiom)} = \forall a, b, c, d, e, h, k\ (\mathcal{R}(a, b, c) \wedge \mathcal{R}(c, d, e) \wedge \mathcal{R}(e, h, k)) \Rightarrow \\ \quad (\exists u, v\ (\mathcal{R}(b, u, v) \wedge \mathcal{R}(d, v, k) \wedge (\forall x, y\ \mathcal{R}(u, x, y) \Rightarrow (k = x)))) \end{array}$$

It can now be proved with standard predicate logic means that the conjunction of these two formulae is equivalent to

$$\forall a\ \mathcal{R}(a, a, a) \wedge \forall a, b, c\ \mathcal{R}(a, b, c) \Rightarrow a = b \wedge a = c$$

That means the relation $\mathcal{R}$ collapses to a point relation. This fact is the key lemma from which it is trivial to show that the $\to$ connective is actually material implication.

The proof that the accessibility relation collapses to a point relation requires a lemma to be proved:

$$\forall x, y, z, u, v\ \mathcal{R}(x, y, z) \wedge \mathcal{R}(u, v, x) \Rightarrow v = z$$

A proof of this lemma has been found by Bill McCune and Larry Wos using the OTTER theorem prover with unit resolution resolution (UR-Resolution) [4]:

Axioms:
$all\ a, b, c, d, e, h, k\ (R(a, b, c)\ \&\ R(c, d, e)\ \&\ R(e, h, k))$
$\Rightarrow (exists\ u, v\ (R(b, u, v)\ \&\ R(d, v, k)\ \&\ (all\ x, y\ (R(u, x, y) \Rightarrow (k = x))))$.

[4] Dov Gabbay and Mark Reynolds found a different proof by hand. So far we have not been able to prove the theorem with an automated theorem prover in one go, i.e. without inventing the lemma.

all z exists x, y $R(x,y,z)$.

Axioms in clause form:

$\neg R(x_1,x_2,x_3)|\neg R(x_3,x_4,x_5)|\neg R(x_5,x_6,x_7)|R(x_2,f_4(x_1,x_2,x_3,x_4,x_5,x_6,x_7),f_3(x_1,x_2,x_3,x_4,x_5,x_6,x_7))$.
$\neg R(x_1,x_2,x_3)|\neg R(x_3,x_4,x_5)|\neg R(x_5,x_6,x_7)|R(x_4,f_3(x_1,x_2,x_3,x_4,x_5,x_6,x_7),x_7)$.
$\neg R(x_1,x_2,x_3)|\neg R(x_3,x_4,x_5)|\neg R(x_5,x_6,x_7)|\neg R(f_4(x_1,x_2,x_3,x_4,x_5,x_6,x_7),x,y)|(x=x_7)$.

$R(g(z),f(z),z)$.

Negated Lemma in clause form:
$R(e,d,c)$.
$R(b,a,e)$.
$(a\neq c)$.

Equality Axioms
$(x=x)$.
$(x_1\neq y_1)|(x_2\neq y_2)|(x_3\neq y_3)|\neg R(x_1,x_2,x_3)|R(y_1,y_2,y_3)$.
$(x\neq y)|(y=x)$.
$(x\neq y)|(y\neq z)|(x=z)$.

— UNIT CONFLICT at 1707.97 sec — 4702 [binary,4701,1487] .

Length of proof is 7.

————— PROOF —————

1	$\neg R(x,y,z)\|\neg R(z,u,v)\|\neg R(v,w,v_6)\|R(y,k(x,y,z,u,v,w,v_6),h(x,y,z,u,v,w,v_6))$.
2	$\neg R(x,y,z)\|\neg R(z,u,v)\|\neg R(v,w,v_6)\|R(u,h(x,y,z,u,v,w,v_6),v_6)$.
3	$\neg R(x,y,z)\|\neg R(z,u,v)\|\neg R(v,w,v_6)\|\neg R(k(x,y,z,u,v,w,v_6),v_7,v_8)\|(v_7=v_6)$.
4	$(x=x)$.
5	$(x\neq y)\|(z\neq u)\|(v\neq w)\|\neg R(x,z,v)\|R(y,u,w)$.
8	$R(g(x),f(x),x)$.
9	$R(e,d,c)$.
10	$R(b,a,e)$.
11	$(a\neq c)$.

Abbreviations:

l	$=_{def}\ g(b)$
m	$=_{def}\ g(l)=g(g(b))$
n	$=_{def}\ g(m)=g(g(g(b)))$
o	$=_{def}\ f(b)$
p	$=_{def}\ f(l)=f(g(b))$
q	$=_{def}\ f(m)=f(g(g(b)))$
r	$=_{def}\ g(p)=g(f(g(b)))$
s	$=_{def}\ f(p)=f(f(g(b)))$
t	$=_{def}\ k(n,q,l,p,l,o,b)=k(g(g(g(b))),f(g(g(b))),g(b),f(g(b)),f(b),b)$
u	$=_{def}\ h(n,q,l,p,l,o,b)=h(g(g(g(b))),f(g(g(b))),g(b),f(g(b)),f(b),b)$
v	$=_{def}\ h(r,s,p,u,b,a,e)=h(g(f(g(b))),f(f(g(b))),f(g(b)),h(n,q,l,p,l,o,b),b,a,e)$
w	$=_{def}\ k(q,t,u,v,e,d,c)=k(f(g(g(b))),k(n,q,l,p,l,o,b),h(n,q,l,p,l,o,b),h(r,s,p,u,b,a,e),e,d,c)$
g_x	$=_{def}\ g(x)$
g_x^2	$=_{def}\ g(g_x)=g(g(x))$
g_x^3	$=_{def}\ g(g_x^2)=g(g(g(x)))$

15	[ur, 2, 8, 8, 8]	$R(f(g_x),h(g(g_x^3),f(g_x^2),g_x^2,f(g_x),g_x,f(x),x),x)$.
19	[ur, 1, 8, 8, 8]	$R(f(g_x^2),k(g(g_x^3),f(g_x^2),g_x^2,f(g_x),g_x,f(x),x),$ $h(g(g_x^3),f(g_x^2),g_x^2,f(g_x),g_x,f(x),x))$.
96	[ur, 2, 8, 15, 10]	$R(u,v,e)$.
1477	[ur, 3, 19, 96, 9, 11]	$\neg R(w,a,x)$.
1487	[ur, 1, 19, 96, 9]	$R(t,w,h(q,t,u,v,e,d,c))$.

4680 [ur, 5, 4, 4, 10, 1477] $(w \neq b)$.
4701 [ur, 3, 8, 8, 8, 4680] $\neg R(t, w, x)$.
4702 [binary,4701,1487] Contradiction.

Using the above lemma, the proof of the main parts

$$\forall a\ \mathcal{R}(a,a,a) \quad \text{and}$$
$$\forall a,b,c\ \mathcal{R}(a,b,c) \Rightarrow a = b \wedge a = c$$

of the theorem is straightforward.

Proof of $\forall a\ \mathcal{R}(a,a,a)$ with OTTER.

Axioms:
$all\ x,y,z,u,v\ R(x,y,z)\ \&\ R(u,v,x) \Rightarrow v = z$

$all\ a,b,c,d,e,h,k\ ((R(a,b,c)\ \&\ R(c,d,e)\ \&\ R(e,h,k))$
$\Rightarrow (exists\ u,v\ (R(b,u,v)\ \&\ R(d,v,k)\ \&\ (all\ x,y\ (R(u,x,y) \Rightarrow (k = x)))))).$

$all\ z\ exists\ x,y\ R(x,y,z).$

Formulae in clause form:

$\neg R(x,y,z)|\neg R(u,v,x)|(v = z).$
$\neg R(x_1,x_2,x_3)|\neg R(x_3,x_4,x_5)|\neg R(x_5,x_6,x_7)|R(x_2, f_4(x_1,x_2,x_3,x_4,x_5,x_6,x_7), f_3(x_1,x_2,x_3,x_4,x_5,x_6,x_7)).$
$\neg R(x_1,x_2,x_3)|\neg R(x_3,x_4,x_5)|\neg R(x_5,x_6,x_7)|R(x_4, f_3(x_1,x_2,x_3,x_4,x_5,x_6,x_7), x_7).$
$\neg R(x_1,x_2,x_3)|\neg R(x_3,x_4,x_5)|\neg R(x_5,x_6,x_7)|\neg R(f_4(x_1,x_2,x_3,x_4,x_5,x_6,x_7), x, y)|(x = x_7).$
$R(g(z), f(z), z).$

$\neg R(c_1,c_1,c_1).$ (negated thoerem.)

———— PROOF ————

1 []	$\neg R(x,y,z)\|\neg R(u,v,x)\|(v = z).$
3 []	$\neg R(x,y,z)\|\neg R(z,u,v)\|\neg R(v,w,v6)\|R(u, f_1(x,y,z,u,v,w,v6), v6).$
5 []	$R(f_4(x), f_3(x), x).$
6 []	$\neg R(c_1,c_1,c_1).$
9 [ur,5,1,5]	$(f_3(f_4(x)) = x).$
11 [para_from,9,5]	$R(f_4(f_4(x)), x, f_4(x)).$
16 [ur,11,3,11,5]	$R(x, f_1(f_4(f_4(f_4(x))), f_4(x), f_4(f_4(x)), x, f_4(x), f_3(x), x), x).$
94 [ur,16,1,11]	$(x = f_4(x)).$
95 [ur,16,1,5]	$(f_3(x) = x).$
102 [para_from,94,5]	$R(x, f_3(x), x).$
281 [para_into,102,95]	$R(x,x,x).$
282 [binary,281,6]	Contradiction.

Proof of $\forall a,b,c\ \mathcal{R}(a,b,c) \Rightarrow a = b \wedge a = c$ with OTTER.

Axioms:
$all\ x,y,z,u,v\ R(x,y,z)\ \&\ R(u,v,x) \Rightarrow v = z$

$all\ x\ R(x,x,x).$

Formulae in clause form:

$\neg R(x,y,z)|\neg R(u,v,x)|(v = z).$
$R(x,x,x).$

$R(c_3,c_2,c_1).$ (negated theorem)
$(c_3 \neq c_1)|(c_3 \neq c_1).$

————— PROOF —————

1 []	$\neg R(x,y,z) \mid \neg R(u,v,x) \mid (v=z)$.
3 []	$R(x,x,x)$.
4 []	$R(c_3,c_2,c_1)$.
5 []	$(c_3 \neq c_2) \mid (c_3 \neq c_1)$.
6 [binary,4,1]	$\neg R(c_1,x,y) \mid (c_2 = y)$.
7 [binary,4,1]	$\neg R(x,y,c_3) \mid (y = c_1)$.
12 [binary,6,3]	$(c_2 = c_1)$.
20 [para_from,12,5]	$(c_3 \neq c_1)$.
31 [binary,7,3]	$(c_3 = c_1)$.
32 [binary,31,20]	Contradiction.

That means using SCAN and standard predicate logic theorem proving, both of which can be automated, it is possible to analyze unknown logics and to find the strongest semantics. This semantics (plus some further optimizations) in turn serve as a basis for a translation into predicate logic.

5 Limitations of the SCAN Algorithm

There is a strange phenomenon which requires further investigation. From modal logic we know cases where a Hilbert axiom has a semantic property which is only second–order axiomatizable. Together with a further axiom, both correspond to a first–order axiomatizable property of the accessibility relation. For example the McKinsey axiom (example 4.5) $\forall P\ \Box\Diamond P \Rightarrow \Diamond\Box P$ alone corresponds to a second–order property of the accessibility relation (reversing skolemization in the SCAN algorithm needs second–order Henkin quantifiers). Combined with the transitivity axiom $\Box P \Rightarrow \Box\Box P$ (ex. 4.3), however, these two define atomicity $\forall x\ \exists y\ (\mathcal{R}(x,y) \wedge \forall z\ \mathcal{R}(y,z) \Rightarrow z = y))$ [vB84, page203] which is obviously a first–order definable property.

Applied to the McKinsey axiom, SCAN actually computes this property if the critical clause which prevents reversing the skolemization in the normal way is replaced with its factor. Although we have some ideas, why transitivity might in this particular case enable this operation, we are far from having a general theory for processing combinations of axioms with these strange properties. Actually the proof that the McKinsey axiom together with the transitivity axiom correspond to atomicity requires the axiom of choice. Therefore no simple solution of this problem is to be expected.

The example, however, shows that SCAN is not complete in the sense that it computes always a first–order formula when there is one. It is, however, not clear whether this has to be the job of SCAN at all. Obviously the conjunction of two formulae, one or both of which are second–order can be equivalent to a first–order formula, even if one of them alone is not equivalent to a first–order formula. This has to be investigated further.

In cases where SCAN loops and produces infinitely many clauses it is sometimes possible to recognize some regularities. We can try to recognize these regularities automatically and to exploit them for finding a finite representation of the infinitely many clauses.

6 Conclusion

We have presented an algorithm for the elimination of second–order quantifiers over predicate variables. The algorithm is simple and easy to implement by a slight modification of a standard resolution theorem prover. In fact, for example the theorem prover OTTER can be used without any changes [McC90]. The trick is to encode those predicates which are not to be resolved upon as $ans–literals. For each 'empty' clause, OTTER finds, it prints the $ans–literals which are part of the empty clause. These literals make up one clause of the result. In order to simulate C-resolution by ordinary resolution the input literals have to be abstracted, i.e. in the clause form the literals $P(s)$ have to be replaced with $P(x) \mid x \neq s$.

Although this works, one would like to have some changes. For example subsumption must be disabled because the implemented subsumption algorithm does not consider the \$ans–literals and therefore removes clauses which actually are not subsumed. Furthermore there are no simplifications on the \$ans–literals. This may cause the output to be unnecessary long.

An even simpler way for realizing SCAN is to let a theorem prover exhaustively generate all resolvents and then to collect manually those clauses not containing the predicates to be eliminated. This permits the application of all simplifications. On the other hand, it sometimes does not prevent the theorem prover from looping in parts of the clause set which are not to be touched by the SCAN algorithm.

We have shown that the SCAN algorithm can be applied to really hard problems such as computation of interpolands or computation of first–order circumscription. It can also be used to transform Hilbert axioms for non classical logics into properties of the semantic structure. This is extremely useful for the mechanization of these logics because it allows for compiling formulae of these logics into first–order predicate logic and using standard predicate logic deduction systems.

Convinced that these are not the only applications of our new technique we would like to encourage everybody to join our work in this area.

References

[Ack35a] Wilhelm Ackermann.
Untersuchung über das Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 110:390–413, 1935.

[Ack35b] Wilhelm Ackermann.
Zum Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 111:61–63, 1935.

[BGW92] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Theorem proving for hierarchic first-order theories, 1992. To appear in Proc. ALP'92, Lecture Notes in Comp. Science.

[Bri92] Chris Brink. *Power Structures and Their Applications.* PhD thesis, Fac. of Science, Rand Afrikaans University, Johannesburg, South Afrika, 1992.

[Che80] B. F. Chellas. *Modal Logic: An Introduction.* Cambridge University Press, Cambridge, 1980.

[GO92a] Dov M. Gabbay and Hans Jürgen Ohlbach. From a Hilbert calculus to its model theoretic semantics, 1992. presented at UKALP, april 92, to be published.

[GO92b] Dov M. Gabbay and Hans Jürgen Ohlbach. Quantifier elimination in second–order predicate logic. *South African Computer Journal*, 7:35–43, July 1992.

[Hen61] L. Henkin. Some remarks on infinitely long formulas. In *Infinistic Methods*, pages 167–183. Pergamon Press, Oxford, 1961.

[KK66] G. Kreisel and J.L. Krivine. *Éléments de Logique Mathématique. Théorie des modèles.* Société Mathématique de France, 1966.

[Lif85] Vladimir Lifschitz. Computing circumscription. In *Proc. of IJCAI 85*, pages 121–127. University of, 1985.

[McC80] John McCarthy. Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence*, 13:295–323, 1980.

[McC90] William McCune. OTTER 2.0. In Mark Stickel, editor, *Proc. of* 10^{th} *Internation Conference on Automated Deduction, LNAI 449*, pages 663–664. Springer Verlag, 1990.

[Ohl91] Hans Jürgen Ohlbach. Semantics based translation methods for modal logics. *Journal of Logic and Computation*, 1(5):691–746, 1991.

[OS91] Hans Jürgen Ohlbach and Jörg H. Siekmann. The Markgraf Karl refutation procedure. In Jean Luis Lassez and Gordon Plotkin, editors, *Computational Logic, Essays in Honor of Alan Robinson*, pages 41–112. MIT Press, 1991.

[Sim92] Harold Simmons. An algorithm for eliminating predicate variables from π_1^1 sentences, 1992.

[Sza92] Andrzej Szałas. On correspondence between modal and classical logic: Automated approach. Technical Report MPI–I–92–209, Max Planck Institut für Informatik, Saarbrücken, march 1992.

[vB84] Johan van Benthem. Correspondence Theory in D. Gabbay, F. Guenthner: Handbook of Philosophical Logic, volume II, Extensions of Classical Logic of *Synthese Library Vo. 165*, pages 167–248. D. Reidel Publishing Company, Dordrecht, 1984.