SCAN–Elimination of Predicate Quantifiers
System Description

Hans Jürgen Ohlbach
Max-Planck-Institut für Informatik,
Im Stadtwald,
66123 Saarbrücken, Germany.
Email: `ohlbach@mpi-sb.mpg.de`.

August 2, 1995

Abstract

SCAN is an algorithm and a program package for computing first–order equivalent formulae for second–order predicate logic sentences. It is remotely accessible via WWW and can be used for general quantifier elimination as well as for specialized tasks, in particular computing first–order circumscription, and correspondence properties in non–classical logics. The general structure of the program and the services it provides are described.

Contents

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 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. Nevertheless, some of the proposed algorithms are quite powerful and can solve hard problems in the areas mentioned above.

In this paper the implementation of the SCAN algorithm[1] proposed by Gabbay and Ohlbach [GO92b, GO92c] is described. The system is accessible via World Wide Web (WWW). 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. So far, access to the program is not restricted.

Besides the basic quantifier elimination algorithm, interfaces for computing first–order circumscription and correspondence axioms in non–classical logics are available. The interfaces translate the given problem specification into a quantifier elimination problem and invoke the basic algorithm.

We describe the general structure of the program package. The technical details can be found in the WWW online documentation.

2 Quantifier Elimination with SCAN

The SCAN algorithm is based on an idea which already appeared in Ackermann [Ack35, Ack54]. SCAN takes as input only second-order formulae of the form

$$\alpha = \exists p_1, \ldots, \exists p_k\ \psi$$

with existentially quantified predicate variables p_i and a first-order formula ψ. If the predicate variables of the formula under consideration are universally quantified, we first negate the formula. This converts the universal quantifiers into existential quantifiers. Then we apply SCAN and negate the result.

The following three steps are performed by SCAN:

1. ψ is transformed into clause form.

2. All C-resolvents and C-factors with the predicate variables $p_1, \ldots, p_n$ 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: $\exists P\ \forall x, y\ \exists z\ (\neg P(a) \vee Q(x)) \wedge (P(y) \vee Q(a)) \wedge P(z)$. In the first step the clause form is computed:

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

f is a Skolem function.

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

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 $\neg P(a)$, 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).

3 The SCAN Program

The SCAN program is a modified version of the OTTER theorem prover devolved by Bill McCune at Argonne National Laboratory [McC90]. The modifications were implemented by Thorsten Engel at the Max-Planck-Institute in Saarbrücken. SCAN, like OTTER, is a C program with a very simple file based interface. An input file contains the specifications of the problem together with the particular settings of the various options. SCAN generates an output file with the result and optionally a protocol of its operations.

The formulae submitted to the system are separated into two parts. The main part is the so–called SOS (Set of Support). It contains the first–order part of the second–order formula to be simplified. If a formula φ and a list $p_1, \ldots, p_n$ of predicate symbols is specified as SOS then $\exists p_1, \ldots, p_n\ \varphi$ is in fact the second–order formula to be simplified. Should the predicate variables be interpreted as universally quantified then $\neg\varphi$ instead of φ must be defined as SOS. The second part is the so called USABLE list. It may consist of arbitrary other first–order formulae not containing the predicates to be eliminated. The USABLE part is used for simplification purposes and, sometimes, for terminating the main SCAN loop. If a formula ψ is put into the USABLE list, φ is put into the SOS list, and ξ is the output of SCAN, then the logical correlation between these formulae is

$$\psi \Rightarrow ((\exists p_1, \ldots, p_n\ \varphi) \Leftrightarrow \xi).$$

After reading the input file, 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 (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.
- 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 (extended purity deletion rule, cf. [DLS94]). 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.)

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 unskolemized, 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 unskolemization algorithm can be used as a quantifier elimination algorithm for function variables (there is a special device to tell the system which functions to eliminate.)

   There are three possible outcomes of the unskolemization routine:
   (i) it may generate a normal first–order formula,
   (ii) it may simply tell us that unskolemization is not possible, or
   (iii) it may generate an (again second–order) parallel Henkin quantifier. For example the unskolemized version of the clause $p(x, y, f(x)), q(x, y, g(y))$ is

$$\left[\begin{array}{cc} \forall x & \exists u \\ \forall y & \exists v \end{array}\right] p(x, y, u) \vee q(x, y, v).$$

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 his 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

$$\begin{array}{ll} C_1 & \neg p(x), p(f(x)), q(x) \\ C_2 & p(a). \\ C_3 & \neg q(x), q(f(x)). \\ C_4 & q(a). \end{array}$$

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, this clause is implied by C_3 and C_4 and can therefore be deleted. The loop is stopped if C_3 and C_4 are available.

4 The Circumscription Interface

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 allow 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 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:

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

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 $\forall x_1, \ldots, x_n$ $(P^*(x_1, \ldots, x_n) \rightarrow P(x_1, \ldots, x_n))$. 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.

Consider our little database above; $flies(Tweety)$. According to the definition of circumscription,

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

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

$$circ(flies(Tweety), flies) = flies(Tweety) \wedge (\forall x\ flies(x) \Rightarrow x = Tweety),$$

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

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

For more information about quantifier elimination and circumscription see [DLS94] and [KL95].

In the current implementation we realized this general version of circumscription by generating the circumscription formula according to the above schema and then applying SCAN to the second-order part. The generation of the circumscription formula is done with a Prolog program which computes the input file for the SCAN program.

5 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 $\Box p \Rightarrow p$ ($\Box$ is the modal box operator, $\Diamond$ is the diamond operator) 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. Jónnson and Tarksi 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.

Let us use the modal logic example to illustrate how SCAN can be used for computing the corresponding frame properties. Suppose we are given the Hilbert axiom $\Box p \Rightarrow p$ and the standard possible worlds semantics of the modal operator:

$$w \models \Box p \quad \text{iff} \quad \forall v \text{ if } r(w,v) \text{ then } v \models \Box p.$$

($\models$ is the satisfiability relation.) This semantics, together with the usual possible worlds semantics of the ordinary propositional connectives can be taken as a rewrite rules for translating the Hilbert axiom into predicate logic. For the above axiom we get:

$$\forall p\ \forall w\ (\forall v\ r(w,v) \Rightarrow p(v)) \Rightarrow p(w).$$

The two quantifiers express that Hilbert axioms are required to hold for all formulae and for all worlds. $(\forall v\ r(w,v) \Rightarrow p(v))$ is the translation of $\Box p$, where $p(v)$ means p is true in world v.

This is now a second-order predicate logic formula. Since we want to apply SCAN, we first negate it; $\exists p\ \exists w\ (\forall v\ r(w,v) \Rightarrow p(v)) \wedge \neg p(w)$, and give it as input to SCAN. SCAN answers with $\forall w\ r(w,w)$, and this is the desired frame property.

The procedure we have implemented for computing correspondences now works as follows:

- We specify a possible worlds semantics for all connectives.
- We use this specification as rewrite rules for translating the Hilbert axioms into predicate logic.
- Since the formula variables are taken to be universally quantified, we negate the formula.
- SCAN is applied to the negated formula and the result negated again.

Unfortunately this procedure is not sufficient for logics weaker than propositional logic, such as intuitionistic logic or relevance logic. In these logics there is an extra condition on the assignment of values to the predicates. In intuitionistic logic for example the requirement is that if a predicate P is true in a world w it remains true in all higher worlds:

$$\forall w\ P(w) \Rightarrow (\forall v\ r(w,v) \Rightarrow P(v)).$$

Nevertheless, we still can use SCAN for computing corresponding frame properties. We just add this assignment condition as a premise

$$\forall w\ \forall P\ \textit{assignment-condition}(P) \Rightarrow \ldots$$

to the translated Hilbert axioms. But this is still not enough. The assignment conditions, in particular those of intuitionistic logic, may be self resolving clauses which cause the resolution in SCAN to loop. This loop can only be stopped by showing that from a certain level onwards the new resolvents are are implied by other clauses and hence redundant. Unfortunately, in may cases this can be shown only with help of additional properties; in the intuitionistic logic case it is the transitivity of the accessibility relation which is essential for stopping the loop.

Therefore the input to our correspondence computation algorithm consists of four main parts:

1. the formula to be translated,
2. the semantics of the operators,
3. the 'predicate qualifications', for example the assignment conditions they become part of the resolution process,
4. the additional assumptions, for example the transitivity of the accessibility relation, these are only used for simplification purposes.

For modal logic and other logics above standard propositional logic, the first two parts are sufficient.

When using SCAN for computing the corresponding frame property for a given Hilbert axiom, four different things may happen.

- SCAN terminates and returns a first-order predicate logic formula as the corresponding frame property. The correctness proof of the SCAN algorithm itself then guarantees that the second-order formula which is given as input to SCAN is logically equivalent to the result.
- SCAN terminates and returns a second-order predicate logic formula with a parallel Henkin quantifier. For example the translation of the modal logic McKinsey axiom $\Box\Diamond p \Rightarrow \Diamond\Box p$ with SCAN yields

  $$\forall w\ \neg(H[\forall x_1 \exists y_1/\forall x_2 \exists y_2]\ \begin{array}{l}(r(w,x_1) \Rightarrow r(x_1,y_1)) \wedge \\ (r(w,x_2) \Rightarrow r(x_2,y_2)) \wedge \\ (r(w,x_1) \wedge r(w,x_2) \Rightarrow y_1 \neq y_2)\end{array}\ )$$

  where $H[\forall x_1\ \exists y_1/\forall x_2\ \exists y_2]$ is the parallel Henkin quantifier. The whole formula with the Henkin quantifier is negated. (Negation of a Henkin quantifier does NOT simply swap the quantifiers!).

  We cannot guarantee that there is no first-order formula which is equivalent to this second-order formula. (The unskolemization is a quite sophisticated search process which not always finds all unskolemization possibilities.)

  In modal logic it is known that the conjunction of the McKinsey axiom and the axiom $\Box p \Rightarrow \Box\Box p$ (which specifies transitivity of the accessibility relation) corresponds to the atomicity of the frames: $\forall x\ \exists y\ r(x,y) \wedge \forall z\ r(y,z) \Rightarrow y = z$. The current version of SCAN cannot manage this. SCAN would compute the McKinsey axiom and then transitivity axiom separately without relating the conjunction with atomicity.

  That means the computation of correspondence properties for conjunctions of Hilbert axioms may not give the optimal result.
- SCAN does not terminate, i.e. the resolution process loops. In this case we cannot guarantee the non–existence of a corresponding first-order property, but we can observe certain patterns. It may turn out that infinitely many resolvents are generated, but once in a while a clause drops out which does not contain the predicates to be eliminated drops out. Moreover, the sequence of these resolvents follows a certain pattern which can be finitely represented using infinite conjunctions and disjunctions. This occurs when applying SCAN to the Löb axiom $\Box(\Box p \Rightarrow p) \Rightarrow \Box p$. The resulting infinite p-free clause set in fact specifies the correspondence property, finiteness of the r-chains.
- The last thing which may happen is that SCAN does not terminate and none of the resolvents is without at least one of the predicates to be eliminated. For example this happens for the modal axiom $\Diamond\Box p \vee \Box(\Box(\Box q \Rightarrow q) \Rightarrow q)$ which is known to be frame incomplete, i.e. there is no class of frames characteristic of this axiom. Again, we cannot guarantee that if SCAN behaves like this it is a frame incomplete axiom, but it is a strong indication.

The correspondence theory interface to SCAN is realized as a Prolog program with an associated database of semantics definitions for operators. The Prolog program turns these semantics definitions into translation rules for Hilbert axioms. Given a Hilbert axiom it translates it into the input language for SCAN and generates the corresponding input file.

6 The WWW Interface

The various components of the SCAN system are linked together as a html document with corresponding forms. The html document contains online documentation of the system, and the forms allow everybody with access to the World Wide Web to use SCAN. Just fill out the form in a WWW browser and click the 'submit' button. The contents of the form are sent to a central server which activates the corresponding interface program and then the SCAN program itself. The answer is immediately sent back to the user. There is currently a time limit of 10 seconds for users without special rights. The URL of the html document is www.mpi-sb.mpg.de/guide/staff/ohlbach/scan/scan.html.

References

[Ack35] Wilhelm Ackermann. Untersuchung über das Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 110:390–413, 1935.

[Ack54] Wilhelm Ackermann. *Solvable Cases of the Decision Problem.* North–Holland, 1954.

[DLS94] Patrick Doherty, Witold Lukaszewics, and Andrzej Szalas. Computing circumscription revisited: A reduction algorithm. Technical Report LiTH-IDA-R-94-42, Institutionen för Datavetenskap, University of Linköping, 1994.

[GO92a] 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)*, pages 425–435. Morgan Kaufmann, 1992. Also published as a Technical Report MPI-I-92-231, Max-Planck-Institut für Informatik, Saarbrücken, and in the *South African Computer Journal*, 1992.

[GO92b] Dov M. Gabbay and Hans Jürgen Ohlbach. Quantifier elimination in second–order predicate logic. *South African Computer Journal*, 7:35–43, July 1992. also published in [GO92a].

[GO92c] Dov M. Gabbay and Hans Jürgen Ohlbach. Quantifier elimination in second–order predicate logic. Technical Report MPI-I-92-231, Max-Planck-Institut für Informatik, Saarbrücken, 1992. Published in Proc. of KR 92 and in the South African Computer Journal, 1992.

[KL95] G. Neelakantan Kartha and Vladimir Lifschitz. A simple formalization of actions using circumscription. In *Proceedings of IJCAI 95*, 1995.

[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.