A Resolution Decision Procedure for the 2-Variable Fragment

Hans de Nivelle

29.11.2000

General

The unsatisfiability problem for FOL is undecidable. There is no algorithm that always terminates, says 'yes' on unsatisfiable formulae, and 'no' on satisfiable formulae.

But the unsatisfiability problem is enumerable. There exists algorithms that terminate and say 'yes' if and exactly if the formula is unsatisfiable.

Resolution is such a enumeration procedure.

Decidable Fragments

For quite a few subclasses of first order logic, the unsatisfiability problem is decidable.

- The Bernays/Schönfinkel class: Formulae of the form $\exists^*\forall^* F$, with F quantifier free.

- The Ackermann class: $\exists^*\forall\exists^* F$, with F quantifier free.

- The Gödel class: $\exists^*\forall^2\exists^* F$, with F quantifier free.

- The 2-variable fragment: F can be written using only x and y.

- The guarded fragment: All quantifications have form $\forall\overline{x}(a \rightarrow F)$ or $\exists\overline{x}(a \wedge F)$, with a an atom, and Freevar(F) $\subseteq$ Freevar(a).

A Short History of the Fragment

Kurt Gödel: The fragment of first order logic, of the form $\exists^*\forall^2\exists^* F$, with F function free, is decidable.

Kurt Gödel: My decidability proof also goes through if one allows equality in F.

Dana Scott: The 2 variable fragment can be translated into the Gödel Class.

?: Gödel's second remark is wrong. The fragment becomes undecidable if one allows equality in F.

Mortimer: The 2 variable fragment with equality is decidable.

Resolution and Paramodulation

We define resolution and paramodulation, indexed by a *selection function*. The selection function selects literals in a clause, and it selects sides of a equality literals.

Resolution Let $A_1 \vee R_1$ and $\neg A_2 \vee R_2$ be clauses, s.t. A_1 and A_2 are unifiable, and selected. Let Θ be the most general unifier. Then $R_1\Theta \cup R_2\Theta$ is a resolvent.

Paramodulation Let $c_1 = t_1 \approx t_2 \vee R_1$ and $c_2 = A[u_1] \vee R_2$ be clauses, s.t. $t_1 \approx t_2$ is selected in clause c_1, t_1 is selected in the equality $t_1 \approx t_2$, $A[u_1]$ selected in c_2, u_1 is not a variable, u_1 and t_1 have most general unifier Θ. Then $A\Theta[t_2\Theta] \cup R_1\Theta \cup R_2\Theta$ is the result of paramodulating c_1 into c_2.

Factoring Let $c = A_1 \vee A_2 \vee R$ be a clause, s.t. A_1 is selected, and A_1, A_2 have most general unifier Θ. Then $A_1\Theta \vee R\Theta$ is a factor of c.

Equality Reflexivity Let $c = t_1 \not\approx t_2 \vee R$ be a clause, s.t. $t_1 \not\approx t_2$ is selected, and t_1, t_2 have most general unifier Θ. Then $R\Theta$ is obtained by equality reflexivity from c.

Equality Factoring Let $c = t_1 \approx t_2 \vee u_1 \approx u_2 \vee R$ be a clause, s.t. $t_1 \approx t_2$ is selected in c, t_1 is selected in $t_1 \approx t_2$, u_1 is selected in $u_1 \approx u_2$, and t_1, u_1 have most general unifier Θ. Then $t_2\Theta \not\approx u_2\Theta \vee u_1\Theta \approx u_2\Theta \vee R\Theta$ is an equality factor of c.

(This is an ugly rule, needed for technical reasons)

The following rules operate on the database as a whole:

Subsumption If for clauses c_1, c_2 there is a substitution Θ, such that $c_1\Theta \subseteq c_2$, then c_1 *subsumes* c_2. In that case c_2 can be deleted from the database.

Splitting The splitting rule can be applied when a clause c can be partitioned into two non-empty parts, that do not have overlapping variables. If c that can be partitioned as $R_1 \vee R_2$, then the prover tries to refute R_1 and R_2 independently.

Possible Selection Functions

Most widely used selection functions are based on orders.

1. Define some arbitrary order $\prec$ on the signature of the language.

2. Extend it to an order on all terms/atoms in some predetermined way.

3. The maximal terms are selected in equalities. The maximal literals are selected in clauses.

Possible selection functions are: *Lexicographic Path Order*, *Recursive Path Order*, or *Knuth Bendix Order*. We will be using LRPO. It has the following properties:

- If $A \prec B$, then $A\Theta \prec B\Theta$ for all A, B, Θ.

- It is well-founded.

- It is preserved in contexts: If $t_1 \prec t_2$, then $A[t_1] \prec A[t_2]$.

- If t is a proper subterm of $T[t]$, then $t \prec T[t]$.

- If symbol $s \succ$ all symbols in t. Then every term containing $s \succ t$.

The S^2 -class

We define the S^2 class. Let c be a clause. It is in S^2 if it satisfies the following conditions:

- c contains at most 2 variables.
- There is a literal that contains all variables in c. If c contains ground literals, then c is a ground clause.
- There are no nested function symbols.
- Each term, built by a function symbol, contains all variables that occur in c.

Strategy for the S^2 Class

In the initial clause set, put a special mark on the predicate symbols of the literals that contain two variables.

Then use an LRPO based on the following signature-order:
Function symbols $\succ$ marked predicate symbols $\succ$ the other symbols.

To obtain termination, it is sufficient to show that the clauses remain within the fragment. For this we need to show:

1. Each selected literal contains all variables of its clause.

2. Each selected literal contains the deepest occurrences of variables of its clause.

Lemma:
In function free clauses, containing 2 variables, the selected literals contain 2 variables.

Proof:
Marked literals become 1-variable in a 1- variable clause. Every clause derived from a 1-variable clause, is 1-variable.

Adding Equality to This

Adding equality to S^2 results in undecidability.

However the 2-variable fragment can be translated into a weaker fragment, in which every clause c needs to satisfy one of the following conditions:

- c contains 2 variables, and no function symbols.
- c contains 1 variable, and possible function symbols. Each term, built by a function symbol contains the variable of c.
- The clause is ground. It contains no nested function symbols.

Strategy

In the initial clause set, put a special mark on the predicate symbols of the literals that contain two variables. This marking includes the equality symbol.

Then use an LRPO based on the following signature-order:
Function symbols $\succ$ marked predicate symbols $\succ$ the marked equality symbol $\succ$ non-equality predicate symbols $\succ$ the equality symbol $\succ$ the other symbols.

Good News: Most equalites are directed in the right direction. Paramodulation into variables is restricted.

Problem: Clauses of the form $X \approx Y \vee R_1(X) \vee R_2(X)$ are problematic.

Solution: Decompose them into:

$$\begin{array}{l|l|l} X \approx c \vee R_1(X) & R_1(X) & R_2(Y) \\ Y \approx c \vee R_2(Y) & & \end{array}$$

This is possible because

$$\forall xy(x \approx y \vee R_1(x) \vee R_2(y))$$

is equivalent to

$$\exists c \forall xy((x \approx c \vee R_1(x)) \wedge (y \approx c \vee R_2(y))) \vee$$

$$\forall x \; R_1(x) \vee \forall y \; R_2(y).$$

Bad News: Clauses of the form

$$X \approx c_1 \vee \cdots X \approx c_n.$$

They trivialize the problem, but I don't know a good way to handle them.