Resolution for Predicate Logic

Hans de Nivelle and Marc Bezem.

We extend resolution and factoring for propositional logic to predicate logic. (clauses with variables) Our approach is based on lifting.

Lifting means that a clause with variables represents the set of its ground instances, and that the reasoning rules on variable clauses are designed in such a way that every reasoning step on ground clauses has a counter part on variable clauses.

Atoms/Literals/Terms/Clauses

Definition: We assume an infinite set of variables $\mathcal{V}$.

We also assume a set of functions $\mathcal{F}$. Each function $f \in \mathcal{F}$ has an arity $\#f$ attached to it, for which $\#f \geq 0$. A function c with $\#c = 0$ is called constant.

Definition: The set of terms is recursively defined as follows:

- A variable $V \in \mathcal{V}$ is a term.
- If $t_1, \ldots, t_n$ are terms, $f \in \mathcal{F}$ has arity n, then $f(t_1, \ldots, t_n)$ is a term.

Definition: We assume a set of predicate symbols $\mathcal{P}$. As with functions, each $p \in \mathcal{P}$ has an arity $\#p$.

An atom has the following form:

- $p(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms, and $p \in \mathcal{P}$ with $\#p = n$.

An atom is ground if it contains no variables.

A literal is an atom A or a negated atom $\neg A$. We will assume that $\neg\neg A = A$.

Definition: A clause is a finite multiset of literals

$$[A_1, \ldots, A_p]$$

.

The logical meaning of a clause $[A_1, \ldots, A_n]$ is the first-order formula $\forall \overline{X}\ (A_1 \vee \cdots \vee A_n)$. Here $\overline{X}$ are the variables that occur in $[A_1, \ldots, A_n]$.

The logical meaning of [ ] is $\bot$.

Substitutions/Unification

Definition: A substitution is a finite set of assignments of form $\{V_1 := t_1, \ldots, V_n := t_n\}$. The V_i are variables, the t_i are terms. It must be the case that $V_i = V_j$ implies $t_i = t_j$.

The application of a substitution Θ on a term t, $t \cdot \Theta$ is recursively defined as follows:

- If V equals one of the V_i then $V \cdot \Theta = t_i$.
- If V does not equal any of the V_i, then $V \cdot \Theta = V$.
- $f(t_1, \ldots, t_n) \cdot \Theta = f(t_1 \cdot \Theta, \ldots, t_n \cdot \Theta)$.

(We usually will not write the $\cdot$ in applications)

Composition

Definition: Let Θ_1 and Θ_2 be substitutions. The composition $\Theta_1 \cdot \Theta_2$ of Θ_1 and Θ_2 is defined as

$$\{V := t \mid t = (V \cdot \Theta_1) \cdot \Theta_2 \text{ and } V \neq t\}.$$

Theorem: For each term t, and substitutions Θ_1, Θ_2,

$$t \cdot (\Theta_1 \cdot \Theta_2) = (t \cdot \Theta_1) \cdot \Theta_2.$$

proof: We first prove the theorem for variables. Let V be a variable. If $(V \cdot \Theta_1) \cdot \Theta_2 \neq V$, then $\Theta_1 \cdot \Theta_2$ contains the assignment $V := (V \cdot \Theta_1) \cdot \Theta_2$, so that $V \cdot (\Theta_1 \cdot \Theta_2) = (V \cdot \Theta_1) \cdot \Theta_2$.

If $(V \cdot \Theta_1) \cdot \Theta_2 = V$, then there is no assignment $V := t$ in $\Theta_1 \cdot \Theta_2$. Hence $V \cdot (\Theta_1 \cdot \Theta_2) = V$.

In both cases, $V \cdot (\Theta_1 \cdot \Theta_2) = (V \cdot \Theta_1) \cdot \Theta_2$.

Composition (2)

For non-variable terms, we apply induction. Assume the theorem holds for all subterms of $f(t_1, \ldots, t_n)$. Then

$$f(t_1, \ldots, t_n) \cdot (\Theta_1 \cdot \Theta_2) = f(\ t_1 \cdot (\Theta_1 \cdot \Theta_2), \ldots, t_n \cdot (\Theta_1 \cdot \Theta_2)\ ).$$

By induction, this is equal to $f(\ (t_1 \cdot \Theta_1) \cdot \Theta_2, \ldots, (t_n \cdot \Theta_1) \cdot \Theta_2) =$

$$f(\ (t_1 \cdot \Theta_1), \ldots, (t_n \cdot \Theta_1)\ ) \cdot \Theta_2 = (f(\ t_1, \ldots, t_n) \cdot \Theta_1\ ) \cdot \Theta_2.$$

Unification

Definition: Let $\mathcal{E} = \quad t_1 \equiv u_1, \ldots, t_n \equiv u_n$ be a system of term equations.

A unifier of $\mathcal{E}$ is a substitution Θ s.t.
$t_1 \cdot \Theta = u_1 \cdot \Theta, \ldots, t_n \cdot \Theta = u_n \cdot \Theta$.

A most general unifier is a unifier Θ, such that for every unifier Θ', there exists a substitution Σ, s.t. $\Theta' = \Theta \cdot \Sigma$.

A unifier of two terms t_1, t_2 is a unifier of the equation $t_1 \equiv t_2$.
Similarly, a most general unifier of t_1, t_2 is a most general unifier of $t_1 \equiv t_2$.

Unification (2)

The terms $f(X, 1)$ and $f(0, Y)$ have unifier $\{X := 0,\ Y := 0\}$. This is also the most general unifier.

The terms $f(X, s(X))$ and $f(Y, Z)$ have unifier
$\{X := 0,\ Y := 0,\ Z := s(0)\}$. Is this a most general unifier?

Do $f(X, Y)$ and $f(a, Z)$ have a unifier?

And $f(X, Y)$ and $g(X, Y)$?

What about $f(X, Y)$ and $f(a, X)$?

And $f(X, s(X))$ and $f(s(Y), Y)$?

Most General Unifiers

G.A. Robinson showed the following (1965)

- If a system of equations has a unifier, then it has a most general unifier.
- There exists a simple algorithm that decides whether a system of equations has a most general unifier. It also computes the most general unifier.

Algorithm for MGU

The algorithm works pretty much everyone would do it. (The real contribution was inventing the notion of most general unifier) If you compare $p(X, Y)$ and $p(a, X)$ from left to right, you see that it is necessary to substitute $X := a$. Having done that, you obtain the terms $p(a, Y)$ and $p(a, a)$. Therefore, it is also necessary to $Y := a$.

The algorithm maintains a state $(\mathcal{E}, \Theta)$ consisting of a system of equations $\mathcal{E}$ and a substitution Θ.

For each pair of states $(\mathcal{E}_i, \Theta_i)$, $(\mathcal{E}_j, \Theta_j)$ of the algorithm holds:

- For every substitution Σ_i, s.t. Σ_i is a unifier of $\mathcal{E}_i$, there exists a substitution Σ_j, s.t. Σ_j is a unifier of $\mathcal{E}_j$, and

$$\Theta_i \cdot \Sigma_i = \Theta_j \cdot \Sigma_j.$$

Transitions of the Unification Algorithm

- $(\mathcal{E} + f(t_1, \ldots, t_n) \equiv f(u_1, \ldots, u_n),\ \Theta) \vdash$
  $(\mathcal{E}\ +\ t_1 \equiv u_1\ +\ \cdots\ +\ t_n \equiv u_n,\ \Theta)$.
- If $f \neq g$, then $(\mathcal{E} +\ f(t_1, \ldots, t_n) \equiv g(u_1, \ldots, u_m),\ \Theta) \vdash (\bot, \Theta)$.
- $(\mathcal{E}\ +\ X \equiv t,\ \Theta) \vdash (\mathcal{E} \cdot \{X := t\},\ \Theta \cdot \{X := t\})$ if $X \neq t$, and X does not occur in t.
- $(\mathcal{E}\ +\ X \equiv t,\ \Theta) \vdash (\bot, \Theta)$ if $X \neq t$, and X does occur in t.
- $(\mathcal{E}\ +\ X \equiv X,\ \Theta) \vdash (\mathcal{E}, \Theta)$.

For every Σ_1 which is a unifier of $\mathcal{E}_1$, there exists a substitution Σ_m, which is a unifier of $\mathcal{E}_m$, s.t. $\Theta_1 \cdot \Sigma_1 = \Theta_m \cdot \Sigma_m$.

This implies that there is no unifier when $\mathcal{E}_m = \bot$. When $\mathcal{E}_m = \{\ \}$, this reads: For every Σ_1 which is a unifier of $\mathcal{E}_1$, there exists a substitution Σ_m, s.t. $\Sigma_1 = \Theta_m \cdot \Sigma_m$.

And reversed: For every Σ_m, which is a unifier of $\mathcal{E}_m$, there exists a substitution Σ_1 which is a unifier of $\mathcal{E}_1$, s.t. $\Theta_m \cdot \Sigma_m = \Theta_1 \cdot \Sigma_1$.

If $\mathcal{E}_m = \{\}$, and one takes $\Sigma_m = \{\ \}$ this statement becomes: There exists a substitution Σ_1 which is a unifier of $\mathcal{E}_1$, s.t. $\Sigma_1 = \Theta_m$.

Resolution with Variables

Resolution: Let $[A_1] \cup R_1$ and $[\neg A_2] \cup R_2$ be clauses, which do not contain shared variables.

Assume that A_1 and A_2 are unifiable with mgu Θ. Then the clause $R_1\Theta \cup R_2\Theta$ is a resolvent of $[A_1] \cup R_1$ and $[\neg A_2] \cup R_2$.

Factoring: Let $[A_1, A_2] \cup R$ be a clause. Assume that A_1 and A_2 are unifiable with mgu Θ. Then the clause $[A_1\Theta] \cup R\Theta$ is a factor of $[A_1, A_2] \cup R$.

For propositional resolution, every factoring step is a simplification step. For predicate resolution, this is not the case anymore:

For example $[p(X, Y), p(Y, 0)]$ has factor $[p(0, 0)]$. Clearly, $[p(0, 0)]$ does not subsume $[p(X, Y), p(Y, 0)]$.

theorem:

Resolution with variables is a sound and complete calculus:

Let $C_1, \ldots, C_n$ be a sequence of clauses: $C_1, \ldots, C_n \vdash^*_{\text{RES+FACT}} [\,]$ iff $C_1, \ldots, C_n$ is unsatisfiable.

As with predicate resolution, soundness is a rule-wise property, while completeness is a global property.

Examples:

Try $[p(0)]$, $[\neg p(X), p(s(X))]$, $[\neg p(s^4(0))]$.

Or $[\neg p(X), \neg p(Y)]$, $[p(X), p(Y)]$.

Lifting

The original meaning of lifting is: Ground refutations can be lifted to predicate refutations.

(Apparently, clauses with variables are intuitively higher than propositional clauses)

We also want to lift redundancy and simplification.

Lifting of Saturated Sets

Our final aim is to be able prove the following theorem:

Lifting Theorem: Let S be a set of predicate clauses. Let $\overline{S}$ be the set of its ground instances.

If S is a saturated set, then $\overline{S}$ is a saturated set.

Remember the definition of saturated set:

We call S a saturated set if

- For every clause D that can be obtained by a forward reasoning rule from clauses $C_1, \ldots, C_n$ with $C_1, \ldots, C_n \in S$,
- there is a clause $D' \in S$, s.t. either $D' = D$ or D' subsumes D.

Sufficient Conditions for Lifting

Forward reasoning rules: Let $C_1, \ldots, C_n$ be a sequence of variable clauses. Let $C'_1, \ldots, C'_n$ be a sequence of ground clauses, s.t. each C'_i is an instance of C_i.

If there exists a forward reasoning rule, with which one can derive a clause D' from $C'_1, \ldots, C'_n$, then there must exist a forward rule for variable clauses with which it is possible to derive a clause D from $C_1, \ldots, C_n$ s.t. D' is an instance of D.

Subsumption : Assume that variable clause C_1 subsumes C_2. Then for every ground instance C'_2 of C_2 there must exist a ground instance C'_1 of C_1, s.t. C'_1 subsumes C'_2.

Let $C_1 = [A_1] \cup R_1$ and $C_2 = [\neg A_2] \cup R_2$ be clauses without shared variables.

Let $C_1\Sigma_1 = [A_1\Sigma_1] \cup R_1\Sigma_1$ and $C_2\Sigma_2 = [\neg A_2\Sigma_2] \cup R_2\Sigma_2$ be ground instances of C_1 and C_2, s.t. a resolvent $R_1\Sigma_1 \cup R_2\Sigma_2$ can be constructed.

Then A_1 and A_2 have an mgu Θ, so that the resolvent $R_1\Theta \cup R_2\Theta$ can be constructed, and $R_1\Sigma_1 \cup R_2\Sigma$ is an instance of the resolvent $R_1\Theta \cup R_2\Theta$.

proof

On the next slide.

From the fact that a resolvent can be constructed, it follows that $A_1\Sigma = A_2\Sigma_2$.

Because C_1 and C_2 have no common variables, when can define $\Sigma = \Sigma_1 \cup \Sigma_2$ without introducing conflicting assignments.

We have

$$A_1\Sigma = A_1\Sigma_1, \quad R_1\Sigma = R_1\Sigma_1,$$
$$A_2\Sigma = A_2\Sigma_2, \quad R_2\Sigma = R_2\Sigma_2.$$

Then Σ is a unifier of A_1 and A_2. It follows that there also exists a most general unifier Θ of A_1 and A_2.

By definition of mgu, there exists a substitution Θ', s.t. $\Theta \cdot \Theta' = \Sigma$.

We have $R_1\Sigma_1 \cup R_2\Sigma_2 = R_1\Sigma \cup R_2\Sigma = (R_1 \cup R_2)\Sigma =$

$$(R_1 \cup R_2)\Theta \cdot \Theta' = (\ (R_1 \cup R_2)\Theta)\Theta' = (\ (R_1\Theta) \cup (R_2\Theta)\ )\Theta'.$$

This completes the proof.

Let $C = [A_1, A_2] \cup R$ be a clause.

Let $C\Sigma$ be a ground instance of C for which $A_1\Sigma = A_2\Sigma$, so that the factor $[A_1\Sigma] \cup R$ can be constructed.

Then A_1 and A_2 have a mgu Θ, and $[A_1\Sigma] \cup R\Sigma$ is an instance of the factor $[A_1\Theta] \cup R\Theta$.

proof We have $A_1\Sigma = A_2\Sigma$. Therefore there exists an mgu Θ, and the factor $[A_1\Theta] \cup R\Theta$ can be constructed.

Because there exists a substitution Θ', s.t. $\Sigma = \Theta \cdot \Theta'$, we have $R\Sigma = (R\Theta)\Theta'$, so that $R\Sigma$ is an instance of $R\Theta$.

Lifting of Ordered Resolution and Factoring

Let $\succ$ be an A-order.

Resolution: Suppose that we have two variable disjoint clauses $[A_1] \cup R_1$ and $[\neg A_2] \cup R_2$ for which A_1 and A_2 are unifiable. When do we have to construct $R_1\Theta \cup R_2\Theta$?

$\Rightarrow$ When the resolution step has at least one instance that satisfies the ordering condition, i.e. when there exists a ground substitution Σ which makes the following constraint true:

$$A_1\Sigma \succ R_1\Sigma \text{ and } A_2\Sigma \succ R_2\Sigma \text{ and } A_1\Sigma = A_2\Sigma.$$

Factoring: Suppose that we have a clause $[A_1, A_2] \cup R$ in which A_1 and A_2 are unifiable. When do we have to construct $[A_1\Theta] \cup R\Theta$?

When there exists a ground substitution Σ that makes the following constraint true:

$$A_1\Theta \succeq R\Theta \text{ and } A_1\Theta = A_2\Theta.$$

Solving the Ordering Constraints

Dependent on the A-order $\succ$, complexity of solving the constraints varies from NP-hard to hopelessly impossible.

Many A-orders are based on complexity (weight) of terms, solving the constraints involves solving integer (diophantine) equations, possibly involving integers $>$ MAXINT.

If one accepts too many solutions, one constructs more resolvents/factor than necessary.

Trade off: Cost of constrant solving $\Leftrightarrow$ cost of constructing too many clauses.

Possible Compromises

- Approximate the constraints:

  When the clause is constructed, it can often be determined that some literals will not be maximal in any instance. These can be blocked in advance.

  When a resolvent is constructed, check one more time, whether the constraint is solvable. If it is not, then the resolvent is not constructed.

  As far as I know, all systems (Prover9, Spass, Vampire, E) follow this approach.

- Keep constraints explicitly in the resolvent:

  This approach seemed promising $<$ 2003. In the meantime, a few people have implemented it, and it has been established that this is not a good approach.

Examples

Define the following A-order $\prec$: Write A_1, A_2 in prefix notation. Compare A_1, A_2 from left-to-right, using alphabetic order.

Assume that the symbols are ordered by

$$p \prec q \prec f \prec g \prec a \prec b.$$

Then $p(a) \prec p(b), \quad p(f(a)) \prec p(a), \quad p(b) \prec q(a)$.

In clause $[p(X), q(Y)]$, the constraint $p(X) \succ q(Y)$ has no solution. Therefore, $p(X)$ can be blocked in advance.

In clause $[p(X), p(Y)]$, the constraint $p(X) \succ p(Y)$ has solution $\{X := b, Y := a\}$.

In clause $[\neg p(X), p(g(X))]$, both the constraints $p(X) \succ p(g(X))$ and $p(g(X)) \succ p(X)$ have solutions.

Can the clause resolve with itself? Solve the constraint:

$$p(X) \succ p(g(X)) \text{ and } p(X) = p(g(Y)) \text{ and } p(g(Y)) \succ p(Y).$$

First unify $p(X)$ and $p(g(Y))$.

$$p(g(Y)) \succ p(g(g(Y))) \text{ and } p(g(Y)) \succ p(Y).$$

Simplify into:

$$Y \succ g(Y) \text{ and } g(Y) \succ Y.$$

No solution!

Lifting of Redundancy

Also here the general rule is: The ground instances are the measure of everything:

Definition: A set of non-ground clauses S makes C redundant, if for every ground instance $C\Theta$ of C, either

1. there are ground instances $S_1, \ldots, S_n$ of clauses in S, s.t. $S_1, \ldots, S_n$ make $C\Theta$ redundant, or
2. $C\Theta$ is an instance of a clause in S.

For example $[p(X)]$ makes $[p(s(X))]$ redundant. (always through the second case)

The empty set $\{\}$ makes $[\neg p(X), p(X)]$ redundant. (because every instance is a tautology)

In case $s/1$ and $a/0$ are the only function symbols, $[p(a)]$ and $[p(s(X))]$ make $[p(X)]$ redundant.

$[p(X), q(X)]$ and $[\neg p(X)]$ make $[q(X), q(X)]$ redundant (Using the A-order from a few slides back)

Do, using the same A-order, $[\neg p(X)]$ and $[p(X), p(X)]$ make $[p(X), p(f(X))]$ redundant?

The examples show that testing redundancy in its full strenghth is very difficult. The dillemma is the same as with the ordering constraints. One has to weight cost of keeping unnecessary clauses against cost of checking redundancy.

Selection Functions

It is a pleasure to speak about something nice and easy after the previous slides. Since selection functions can be choosen completely free on the ground level, there are no restrictions on the predicate level.

One can define any selection function on the predicate level, and it will represent a valid selection function on the ground level.

Use of selection functions is the same as on the ground level.

(Of course, if you really want, you may define a complicated selection function on the ground level, and generate even more complicated constraints on the predicate level)

Summary

We defined the resolution calculus (with redundancy) for predicate logic in such a way that it simulates resolution for propositional logic.

One normally uses the word lifting for this: Ground refutations (and saturations) can be lifted to the predicate level.