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 $\perp$.

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, $\quad t \cdot \Theta$ 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 $\mathcal{E} = t_1 \equiv u_1, \ldots, t_n \equiv u_n$ 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 $f(a, X)$?

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 how a person would do it. 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 transition $(\mathcal{E}_i, \Theta_i) \vdash (\mathcal{E}_{i+1}, \Theta_{i+1})$ of the algorithm holds:

- For every substitution Σ_i, s.t. $\Theta_i \cdot \Sigma_i$ is a unifier of $\mathcal{E}_i$, there exists a substitution Σ_{i+1}, s.t. $\Theta_{i+1} \cdot \Sigma_{i+1}$ is a unifier of $\mathcal{E}_{i+1}$.
- For every substitution Σ_{i+1}, s.t. $\Theta_{i+1} \cdot \Sigma_{i+1}$ is a unifier of $\mathcal{E}_{i+1}$, there exists a substitution Σ_i, s.t. $\Theta_i \cdot \Sigma_i$ is a unifier of $\mathcal{E}_i$.

It starts in the state $(\mathcal{E}, \emptyset)$, and it ends in a state $(\emptyset, \Theta)$, or $(\perp, \Theta)$.

We use the notation $\perp$ to denote a system of equations that has no unifier.

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

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: $C_1, \ldots, C_n \vdash^*_{\mathrm{RES+FACT}} [\,]$ 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 clausaes. 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 L-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 $R_1\Theta \cup R_2\Theta$?

At least in the case there exist a ground instance Σ, s.t.

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

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 $[A_1\Theta] \cup R\Theta$?

At least in case there exists a ground substitution Σ, s.t. $A_1\Theta \succeq R\Theta$.

Lifting of Ordered Resolution and Factoring

How do we know this?

- Try to solve the ordering constraints. For some standard orders, this is NP-complete. For other orders, the complexity is unknown.

  Either way, the algorithms are complex, and the computational cost is high.

- Approximate the constraints.

  Look at the clause whether there are some literals that can never be maximal. Mark these literals as blocked.

  Attempt to construct resolvents with the non-blocked literals.

  When a resolvent is (almost) constructed. Check one more time, whether the constraint is solvable. If it is not, then don't construct the resolvent.

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

- Put the constraints explicitly in the resolvent. This approach seemed promissing < 2003. In the meantime some people have implemented it, and it appears that this is not a good approach.

Summary

We designed resolution for predicate logic in such a way that it simulates resolution for propositional logic.

After that, we have shown completeness of resolution for predicate logic, using the simulation.