Resolution Principle Theorem Proving (cont.'d)

The Resolution Principle along with paramodulation provides a simple basis for mechanized inference and question answering

The Resolution Principle

Review of terminology

atom: Predicate constant followed by a list of arguments; e.g., P(x,y,g(x,f(y)))

literal: atom or negated atom; e.g., Q(C); ¬P(x,y,g(x,f(y)));

clause: disjunction of literals; e.g., Q(C) ∨¬P(x,y,g(x,f(y)));
alternative notation { Q(C), ¬P(x,y,g(x,f(y)}.

Narrow resolution

We have seen a special case of "narrow" resolution before, namely the "cancellation" rule"

$$\frac{\phi \vee \psi,\ \neg\phi \vee \chi}{\psi \vee \chi}$$

Note that what we are doing here is to cancel the *complementary* literals $\phi, \neg\phi$ and inferring the disjunction of remaining literals. The resolution is narrow in that it involves only one literal from each clause.

The cancellation rule seems intuitively quite plausible, and we can make it even more plausible by relating it to an ancient Greek *syllogism*, namely the rule

$$\frac{P \Rightarrow Q,\ Q \Rightarrow R}{P \Rightarrow R}.$$

Clearly this expresses the transitivity of implication. We can make any substitution we like in this schema, so let's replace P by $\neg\psi$, Q by ϕ, and R by χ. So then the rule becomes

$$\frac{\neg\psi \Rightarrow \phi,\ \phi \Rightarrow \chi}{\neg\psi \Rightarrow \chi},$$

which we can transform by eliminating implications to

$$\frac{\psi \vee \phi,\ \neg\phi \vee \chi}{\psi \vee \chi}.$$

This differs only trivially from the earlier cancellation rule.

Of course, plausibility is not enough – we want to be sure the rule is in fact sound (truth-preserving). This is not hard to show in the case where ϕ, ψ, and χ are closed formulas. We'll sketch a more general proof of the soundness of resolution later.

More generally, narrow resolution applies to pairs of clauses each of which can have any number of literals. In addition – and this is the really crucial idea contributed by Robinson – the two literals we "cancel" need not be negations of each other; they do need to have the same predicate constant, and exactly one of the literals must be negative, but the terms in the two literals need not be the same, as long as they are *unifiable*.

Informally, **unification** of two argument lists consists of going through the two lists in parallel from left to right, and whenever nonidentical symbols are encountered, where at least one is a variable, substituting for the variable so as to make the symbols at that point identical. For example, to unify the argument lists of

P(x,f(x),z) and ¬P(A,y,w)

we begin by replacing x by A, written (x/A). In doing so, the first literal becomes

P(A,f(A),z)

(note the change in the second term!). Since the first terms of the two literals are now identical, we can proceed to the second term. We see that f is not the same as y, so we replace y by f(A), i.e., (y/f(A)). Proceeding to the third term, we can now replace either of the mismatched variables z, w by the other, say (z/w). At this point the unification is complete, and we have the *unifier* (x/A)(y/f(A))(z/w). You can think of this unifier as three equations constraining the variables involved, namely

x = A
y = f(A)
z = w

The unifier obtained in this way is called the *most general unifier*, or *mgu*, because we have not made any variables more particular than necessary. For example, we could have unified the third terms, z and w, by replacing both by A, or f(A), etc., but this would not have given us the mgu.

With this informal notion of unification in mind, let's look at an example of resolving. We will resolve a 4-literal clause with a 2-literal clause. So we will end up with a *resolvent* (conclusion) with (4 + 2 - 2) = 4 literals. The clauses are

1. ¬P(x,y,u) ∨ ¬P(y,z,v) ∨ ¬P(u,z,w) ∨ $\underline{\text{P(x,v,w)}}$
2. $\underline{\text{¬P(i(x),E,u)}}$ ∨ P(u,x,E)

The literals to be unified are underlined. A convenient notation is to call them literals 1d and 2a respectively, where we are using a, b, c, ... for "first literal", "second literal", "third literal", etc.

First we should rename variables to avoid confusion, so let's use s, t in place of x, u

in the second clause:

1. $\neg P(x,y,u) \vee \neg P(y,z,v) \vee \neg P(u,z,w) \vee \underline{P(x,v,w)}$
2. $\underline{\neg P(i(s),E,t)} \vee P(t,s,E)$

Now we unify the indicated literals, finding the mgu

$(x/i(s))(v/E)(t/w)$.

(E is a constant and I is a unary function.) We now make the substitution specified by the unifier *throughout both clauses*:

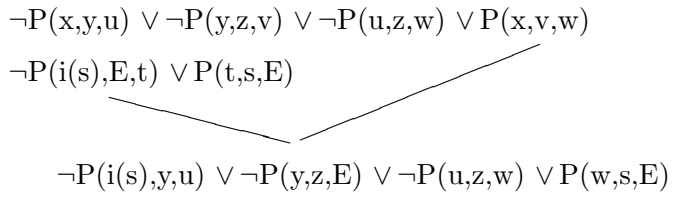
1'. $\neg P(i(s),y,u) \vee \neg P(y,z,E) \vee \neg P(u,z,w) \vee \underline{P(i(s),E,w)}$

2'. $\underline{\neg P(i(s),E,w)} \vee P(w,s,E)$

The underlined literals are now just negations of each other (that was the point of unification), so we can now "cancel" them and infer the disjunction of remaining literals, i.e., the *resolvent*:

3. $\neg P(i(s),y,u) \vee \neg P(y,z,E) \vee \neg P(u,z,w) \vee P(w,s,E)$ r[1d,2a]

Note the notation alongside the resolvent, indicating that this clause is the resolvent obtained by unifying the fourth literal of the first clause with the first literal of the second clause. Another, very perspicuous way of presenting resolution derivations is in a graphical form, where we draw lines from the literals being unified to the resolvent, like this:



Factoring

Factoring is an additional inference rule that is needed if we restrict ourselves to narrow resolution. (We'll see later why it's necessary.) Basically, a factor is a special case of a clause, obtained by unifying two of its literals, and replacing the unified literals by the single unified literal.

More exactly, we pick two literals (of one clause) with the same predicate and same 'sign' and (i) unify the literals, (ii) make the unifying substitution *throughout* the clause, and (iii) delete the extra copy of the two unified literals. For example, let's factor clause 1 above by unifying its first and third literals:

1. $\underline{\neg P(x,y,u)} \vee \neg P(y,z,v) \vee \underline{\neg P(u,z,w)} \vee P(x,v,w)$

$(u/x)(z/y)(w/x)$

f[1a,c]: $\neg P(x,y,x) \vee \neg P(y,y,v) \vee P(x,v,x)$

General procedure for resolution principle theorem proving

1. Add the denial of the conclusion to the premises
2. Convert the premises and denial of the conclusion to clause form
3. Systematically resolve pairs of clauses and form factors of clauses, adding inferred clauses to the set of clauses, until the empty clause is obtained. The particular strategies for systematic resolution vary, e.g., unit preference, set-of-support, etc. (discussed later)

The third step above is also referred to as a *refutation*, since we are refuting the claim that the clauses we are starting with can be simultaneously true.

No factoring is needed if we use a more general, 'wide' form of resolution discussed later. (Essentially this combines narrow resolution and factoring.) On the other hand, the above outline leaves out paramodulation, the extra inference rule needed if we wish to handle equality.

Example

The following 'paranoia' example is a little more interesting than the one shown at the beginning.[1]

Given:

1. Noone likes anyone of whom he is afraid
2. Noone likes anyone who does not like him
3. All paranoids are afraid of everyone except their friends
4. There is a paranoid, and all of his friends are also paranoid

Conclusion:

There is someone whom all non-paranoids dislike

We will interpret "dislikes" as "does not like" and use the following symbolism:

$A(x,y) \longleftrightarrow$ x is afraid of y
$L(x,y) \longleftrightarrow$ x likes y
$P(x) \longleftrightarrow$ x is paranoid
$H(x,y) \longleftrightarrow$ x has friend y

[1] Exam questions tend to be of about this order of complexity!

Premises and denial of conclusion in FOL:

	Strict version	Looser version
1.	$(\forall x\ (\forall y\ [A(x,y) \Rightarrow \neg L(x,y)]))$	$\forall\, x,y.\ A(x,y) \Rightarrow \neg L(x,y)$
2.	$(\forall x\ (\forall y\ [\neg L(y,x) \Rightarrow \neg L(x,y)]))$	$\forall\, x,y.\ \neg L(y,x) \Rightarrow \neg L(x,y)$
3.	$(\forall x\ (\forall y\ [[P(x) \wedge \neg H(x,y)] \Rightarrow A(x,y)]))$	$\forall\, x,y.\ (P(x) \wedge \neg H(x,y)) \Rightarrow A(x,y)$
4.	$(\exists x\ [P(x) \wedge (\forall y\ [H(x,y) \Rightarrow P(y)])])$	$\exists\, x.\ P(x) \wedge \forall\, y.\ H(x,y) \Rightarrow P(y)$
5.	$\neg(\exists x\ (\forall y\ [\neg P(y) \Rightarrow \neg L(y,x)]))$	$\neg\, \exists\, x \forall\, y.\ \neg P(y) \Rightarrow \neg L(y,x)$

Converting to clause form (note that 4, 5 yield 2 clauses each):

1. $\neg A(x,y) \vee \neg L(x,y)$
2. $L(y,x) \vee \neg L(x,y)$
3. $\neg P(x) \vee H(x,y) \vee A(x,y)$

4'. $P(C)$

4". $\neg H(C,y) \vee P(y)$

5'. $\neg P(f(x))$

5". $L(f(x),x)$

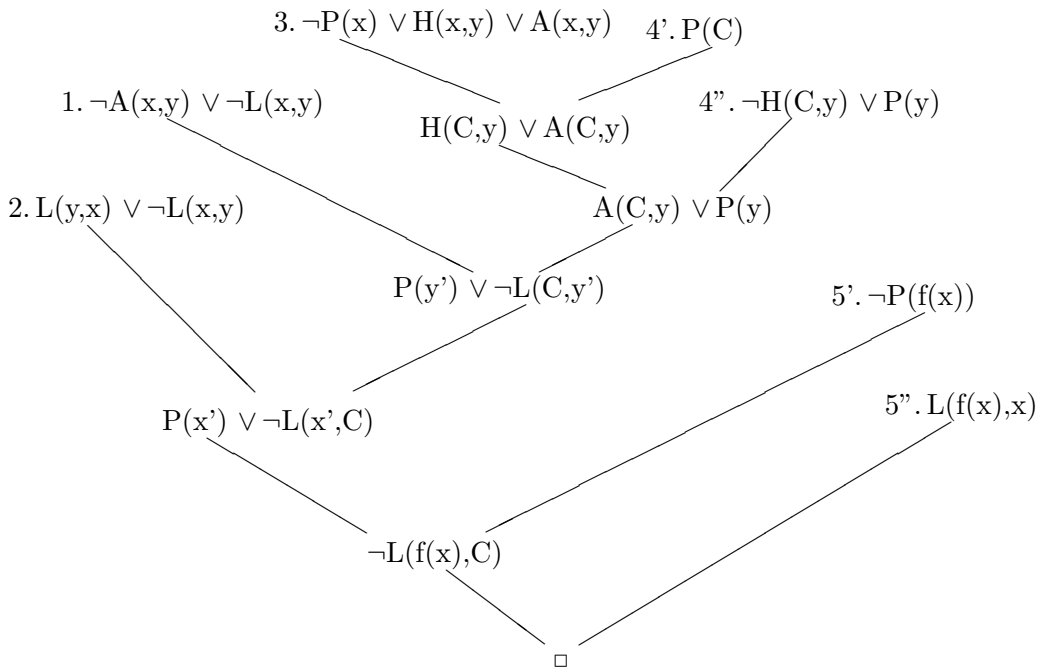
For nontrivial problems, it may be a good idea to sketch out a proof in words before commencing with a formal refutation. Then we can often model the formal proof more or less step-by-step on the intuitive proof. In particular, the informal proof can be a good guide as to what premise (or part of the conclusion) to use next. So here is an informal proof:

If there are no non-paranoids, the conclusion is trivially true.
Otherwise, let B be an arbitrary non-paranoid. (C is the paranoid).
Then by 4", C does not have B as friend.
Then by 4' and 3, C is afraid of B.
Then by 1, C dislikes B.
Then by 2, B dislikes C.
But since B is an arbitrary non-paranoid, the conclusion follows.

Resolution proof:

6.	$H(C,y) \vee A(C,y)$	r[3a,4']	(x/C)
7.	$A(C,y) \vee P(y)$	r[6a,4"a]	variables only
8.	$P(y') \vee \neg L(C,y')$	r[7a,1a]	(x/C)
9.	$P(x') \vee \neg L(x',C)$	r[8b,2a]	(y/C)(y'/x)
10.	$\neg L(f(x),C)$	r[9a,5']	(x'/f(x))
11.	$\square$	r[10,5"]	(x/C)

An even clearer presentation of the proof is as a *refutation graph*:



Mathematical example

(*primarily for grads*)

The following is a quite interesting example from group theory, taken from James Slagle's 1971 book *Artificial Intelligence: The Heuristic Programming Approach* (the first AI textbook!).

1. There exists a left solution, i.e., for all x, y in the group there is an element z such that

   $z \circ x = y$

   (The dot is some operator that associates an element w in the group with any pair of elements u, v in the group)

2. There exists a right solution, i.e., for all x,y there is a z such that

   $x \circ z = y$

3. Associativity, i.e., for all x, y, z

   $(x \circ y) \circ z = x \circ (y \circ z)$

Theorem. There exists a right identity element, i.e., there is an x such that for all y

$$y \circ x = y\,.$$

E.g., The positive reals form a group under multiplication; identity = 1;
e.g., all reals form a group under addition; identity = 0;
e.g., binary words form a group under equivalence; identity = 111...1;
e.g., binary words form a group under exclusive or; identity = 000...0.

We let P(x,y,z) denote x ∘ y = z. This avoids use of equality and paramodulation for the time being, but makes it a bit tricky to express (the relevant parts of) axioms 1-3. In axiom 3, we shall have to use variables to name the subexpressions of terms like (x ∘ y) ∘ z), namely (x ∘ y) and the entire term (x ∘ y) ∘ z), and relate these subexpressions to each other using the P-predicate.

1. $\forall\, x,y\ \exists\, z.\ P(z,x,y)$
2. $\forall\, x,y\ \exists\, z.\ P(x,z,y)$
3. Note the extra 'names' we introduce for subexpressions:

$$\underbrace{\underbrace{(x \circ y)}_{u} \circ z}_{w} = x \circ \underbrace{\underbrace{(y \circ z)}_{v}}_{w}$$

$$\forall\, x,y,z,u,v,w.\ [[(x \circ y = u) \wedge (y \circ z = v) \wedge (u \circ z = w)] \Rightarrow (x \circ v = w)]$$
$$\wedge\, [[(x \circ y = u) \wedge (y \circ z) = v) \wedge (x \circ v = w)] \Rightarrow (u \circ z = w)]$$

$$\forall\, x,y,z,u,v,w.\ [[P(x,y,u) \wedge P(y,z,v) \wedge P(u,z,w)] \Rightarrow P(x,v,w)]$$
$$\wedge\, [[P(x,y,u) \wedge P(y,z,v) \wedge P(x,v,w)] \Rightarrow P(u,z,w)]$$

4. Denial of conclusion:
   $\neg\exists\, x \forall\, y.\ P(y,x,y)$, i.e., $\forall\, x\, \exists\, y.\ \neg P(y,x,y)$

Converting to clause form:

1. P(g(x,y),x,y)
2. P(x,h(x,y),y)
3. We need only the second part of the conjunction:
   $\neg P(x,y,u) \vee \neg P(y,z,v) \vee \neg P(x,v,w) \vee P(u,z,w)$
4. $\neg P(k(x),x,k(x))$

Resolution proof:

5. ¬P(x,y,u) ∨ ¬P(y,z,y) ∨ P(u,z,u)	f[3a,c] (v/y)(w/u)
6. Rewrite 1 as P(g(v,w),v,w) ¬P(y,z,y) ∨ P(u,z,u)	r[1,5a] (x/g(v,w))(v/y)(w/u)
7. ¬P(y,x,y)	r[4,6b] (u/k(x)(z/x)
8. Rewrite 7 as ¬P(u,v,u) □ (Note that both atoms become P(x,h(x,x),x).)	r[2,7] (u/x)(v/h(x,y)(y/x)

Factoring and general (wide) resolution

It might be thought that factoring is convenient but inessential; but as the following example shows, as long as we are using narrow resolution, factoring is needed as well.

1. P(x) ∨ P(y)
2. ¬P(u) ∨ ¬P(w)

The first clause says that either every x has property P or every y has property P. But this is the same as saying every x has property P. Similarly, the second clause is just a redundant way of saying that nothing has property P. So the two clauses are clearly unsatisfiable, but when we try to use (narrow) resolution alone, we get nowhere:

3. P(y) ∨ ¬P(w)	r[1a,2a]
4. ¬P(w) ∨ ¬P(w')	r[2a,3a] (after renaming)

etc. But with factoring the refutation is easy:

3'. P(x)	f[1a,b]
4'. ¬P(u)	f[2a,b]
5. □	r[3',4']

General (wide) resolution actually allows us to collapse these 3 steps into one: Given two clauses we are allowed to unify any number of literals of form P(...) from one clause with any number of literals of form ¬P(...) from the other clause.

So above we simultaneously unify the arguments of P(x), P(y), ¬P(u), and ¬P(w), obtaining some unifier such as

(y/x)(u/x)(w/x),

and since the unification succeeded we immediately infer the empty clause, □.

The unification algorithm

We've already explained unification informally and given examples, but we should now look at a slightly more formal statement of the algorithm.

Algorithm for mgu of two lists of terms $(\sigma_1, ..., \sigma_n)$ *and* $(\tau_1, ..., \tau_n)$*:*

1. Make all variables in one list distinct from all variables in the other;
2. Initialize the mgu to ( );
3. Go through the two lists in parallel, symbol by symbol from left to right;
   (a) If we have exhausted the two lists, succeed with unifier mgu;
   (b) If the two symbols are the same, proceed to the next pair of symbols (for the purpose of this exposition, brackets and commas also count as symbols; of course, in a Lisp implementation there would be no commas separating the terms on the lists);
   (c) If neither symbol is a variable, return FAIL;
   (d) If one symbol is a variable ν and the other is a constant or variable τ, push (ν/τ) onto mgu, apply this substitution throughout the remainder of both lists and proceed to the next pair of symbols;
   (e) If one symbol is a variable ν and the other is a function symbol followed by its list of arguments, say $f(\tau_1, ..., \tau_m)$, *where* ν *does not occur in the terms* $\tau_1, ..., \tau_m$, then push $(\nu/f(\tau_1, ..., \tau_m))$ onto mgu, apply this substitution throughout the remainder of both lists, and proceed to the next pair of symbols (the one after $f(\tau_1, ..., \tau_m)$, which now heads the remainder of both lists);
   (f) Otherwise return FAIL.

Note that in step (3e), we do not allow substitution of a term for a variable if that term contains the variable. The reason for this can be seen from an example like the following.

We try to unify the lists

(x, f(x)) and (y, y).

First we replace (y/x). So the lists become (x, f(x)) and (x, x). Next we replace (x/f(x)) throughout both lists, obtaining (f(x), f(f(x)) and (f(x), f(x)). We now again would have to replace (x/f(x)), and so on *ad infinitum*. So this substitution cannot possibly succeed, and the same is true in the general case.

Slagle gave the following example of a unification;

P(s, g(s), t, h(s,t), u, k(s,t,u))

¬P(v, w, m(w), x, n(w,x), y)

(Working out the unifier here is a good exercise.) Slagle pointed out that the single-step refutation made possible by unification is in sharp contrast with the number of steps that would have been required by the original Herbrand procedure: 10^{256}! This is because the Herbrand procedure is based on making *ground* substitutions only.