Chapter 8

Resolution In First-Order Logic

8.1 Introduction

In this chapter, the resolution method presented in Chapter 4 for propositional logic is extended to first-order logic without equality. The point of departure is the Skolem-Herbrand-Gödel theorem (theorem 7.6.1). Recall that this theorem says that a sentence A is unsatisfiable iff some compound instance C of the Skolem form B of A is unsatisfiable. This suggests the following procedure for checking unsatisfiability:

Enumerate the compound instances of B systematically one by one, testing each time a new compound instance C is generated, whether C is unsatisfiable.

If we are considering a first-order language without equality, there are algorithms for testing whether a quantifier-free formula is valid (for example, the *search* procedure) and, if B is unsatisfiable, this will be eventually discovered. Indeed, the *search* procedure halts for every compound instance, and for some compound instance C, $\neg C$ will be found valid.

If the logic contains equality, the situation is more complex. This is because the *search* procedure does not necessarily halt for quantifier-free formulae that are not valid. Hence, it is possible that the procedure for checking unsatisfiability will run forever even if B is unsatisfiable, because the *search* procedure can run forever for some compound instance that is not unsatisfiable. We can fix the problem as follows:

Interleave the generation of compound instances with the process of checking whether a compound instance is unsatisfiable, proceeding by rounds. A round consists in running the *search* procedure a fixed number of steps for each compound instance being tested, and then generating a new compound instance. The process is repeated with the new set of compound instances. In this fashion, at the end of each round, we have made progress in checking the unsatisfiability of all the activated compound instances, but we have also made progress in the number of compound instances being considered.

Needless to say, such a method is horribly inefficient. Actually, it is possible to design an algorithm for testing the unsatisfiability of a quantifier-free formula with equality by extending the congruence closure method of Oppen and Nelson (Nelson and Oppen, 1980). This extension is presented in Chapter 10.

In the case of a language without equality, any algorithm for deciding the unsatisfiability of a quantifier-free formula can be used. However, the choice of such an algorithm is constrained by the need for efficiency. Several methods have been proposed. The *search* procedure can be used, but this is probably the least efficient choice. If the compound instances C are in CNF, the resolution method of Chapter 4 is a possible candidate. Another method called the method of *matings* has also been proposed by Andrews (Andrews, 1981).

In this chapter, we are going to explore the method using resolution. Such a method is called *ground resolution*, because it is applied to quantifier-free clauses with no variables.

From the point of view of efficiency, there is an undesirable feature, which is the need for systematically generating compound instances. Unfortunately, there is no hope that the process of finding a refutation can be purely mechanical. Indeed, by Church's theorem (mentioned in the remark after the proof of theorem 5.5.1), there is no algorithm for deciding the unsatisfiability (validity) of a formula.

There is a way of avoiding the systematic generation of compound instances due to J. A. Robinson (Robinson, 1965). The idea is not to generate compound instances at all, but instead to generalize the resolution method so that it applies directly to the clauses in B, as opposed to the (ground) clauses in the compound instance C. The completeness of this method was shown by Robinson. The method is to show that every ground refutation can be lifted to a refutation operating on the original clauses, as opposed to the closed (or ground) substitution instances. In order to perform this lifting operation the process of *unification* must be introduced. We shall define these concepts in the following sections.

It is also possible to extend the resolution method to first-order languages with equality using the *paramodulation* method due to Robinson and Wos (Robinson and Wos, 1969, Loveland, 1978), but the completeness proof is rather delicate. Hence, we will restrict our attention to first-order languages without equality, and refer the interested reader to Loveland, 1978, for an exposition of paramodulation.

As in Chapter 4, the resolution method for first-order logic (without equality) is applied to special conjunctions of formulae called clauses. Hence, it is necessary to convert a sentence A into a sentence A' in clause form, such that A is unsatisfiable iff A' is unsatisfiable. The conversion process is defined below.

8.2 Formulae in Clause Form

First, we define the notion of a formula in clause form.

Definition 8.2.1 As in the propositional case, a *literal* is either an atomic formula B, or the negation $\neg B$ of an atomic formula. Given a literal L, its *conjugate* $\overline{L}$ is defined such that, if $L = B$ then $\overline{L} = \neg B$, else if $L = \neg B$ then $\overline{L} = B$. A sentence A is in *clause form* iff it is a conjunction of (prenex) sentences of the form $\forall x_1 ... \forall x_m C$, where C is a disjunction of literals, and the sets of bound variables $\{x_1, ..., x_m\}$ are disjoint for any two distinct clauses. Each sentence $\forall x_1 ... \forall x_m C$ is called a *clause*. If a clause in A has no quantifiers and does not contain any variables, we say that it is a *ground clause*.

For simplicity of notation, the universal quantifiers are usually omitted in writing clauses.

Lemma 8.2.1 For every (rectified) sentence A, a sentence B' in clause form such that A is valid iff B' is unsatisfiable can be constructed.

Proof: Given a sentence A, first $B = \neg A$ is converted to B_1 in NNF using lemma 6.4.1. Then B_1 is converted to B_2 in Skolem normal form using the method of definition 7.6.2. Next, by lemma 7.2.1, B_2 is converted to B_3 in prenex form. Next, the matrix of B_3 is converted to conjunctive normal form using theorem 3.4.2, yielding B_4. In this step, theorem 3.4.2 is applicable because the matrix is quantifier free. Finally, the quantifiers are distributed over each conjunct using the valid formula $\forall x(A \land B) \equiv \forall x A \land \forall x B$, and renamed apart using lemma 5.3.4.

Let the resulting sentence be called B'. The resulting formula B' is a conjunction of clauses.

By lemma 6.4.1, B is unsatisfiable iff B_1 is. By lemma 7.6.3, B_1 is unsatisfiable iff B_2 is. By lemma 7.2.1, B_2 is unsatisfiable iff B_3 is. By theorem 3.4.2 and lemma 5.3.7, B_3 is unsatisfiable iff B_4 is. Finally, by lemma 5.3.4 and lemma 5.3.7, B_4 is unsatisfiable iff B' is. Hence, B is unsatisfiable iff B' is. Since A is valid iff $B = \neg A$ is unsatisfiable, then A is valid iff B' is unsatisfiable. $\square$

EXAMPLE 8.2.1

Let

$$A = \neg\exists y\forall z(P(z,y) \equiv \neg\exists x(P(z,x) \land P(x,z))).$$

First, we negate A and eliminate $\equiv$. We obtain the sentence

$$\exists y\forall z[(\neg P(z,y) \lor \neg\exists x(P(z,x) \land P(x,z)))\land \\ (\exists x(P(z,x) \land P(x,z)) \lor P(z,y))].$$

Next, we put in this formula in NNF:

$$\exists y\forall z[(\neg P(z,y) \lor \forall x(\neg P(z,x) \lor \neg P(x,z)))\land \\ (\exists x(P(z,x) \land P(x,z)) \lor P(z,y))].$$

Next, we eliminate existential quantifiers, by the introduction of Skolem symbols:

$$\forall z[(\neg P(z,a) \lor \forall x(\neg P(z,x) \lor \neg P(x,z)))\land \\ ((P(z,f(z)) \land P(f(z),z)) \lor P(z,a))].$$

We now put in prenex form:

$$\forall z\forall x[(\neg P(z,a) \lor (\neg P(z,x) \lor \neg P(x,z)))\land \\ ((P(z,f(z)) \land P(f(z),z)) \lor P(z,a))].$$

We put in CNF by distributing $\land$ over $\lor$:

$$\forall z\forall x[(\neg P(z,a) \lor \neg P(z,x) \lor \neg P(x,z))\land \\ (P(z,f(z)) \lor P(z,a)) \land (P(f(z),z)) \lor P(z,a))].$$

Omitting universal quantifiers, we have the following three clauses:

$$C_1 = (\neg P(z_1,a) \lor \neg P(z_1,x) \lor \neg P(x,z_1)),$$
$$C_2 = (P(z_2,f(z_2)) \lor P(z_2,a)) \text{ and}$$
$$C_3 = (P(f(z_3),z_3) \lor P(z_3,a)).$$

We will now show that we can prove that $B = \neg A$ is unsatisfiable, by instantiating C_1, C_2, C_3 to ground clauses and use the resolution method of Chapter 4.

8.3 Ground Resolution

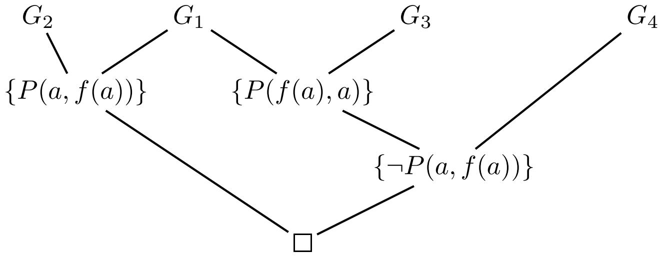
The *ground resolution method* is the resolution method applied to sets of ground clauses.

EXAMPLE 8.3.1

Consider the following ground clauses obtained by substitution from C_1, C_2 and C_3:

$$\begin{aligned}
&G_1 = (\neg P(a,a)) \text{ (from } C_1, \text{ substituting } a \text{ for } x \text{ and } z_1)\\
&G_2 = (P(a,f(a)) \lor P(a,a)) \text{ (from } C_2, \text{ substituting } a \text{ for } z_2)\\
&G_3 = (P(f(a),a)) \lor P(a,a)) \text{ (from } C_3, \text{ substituting } a \text{ for } z_3).\\
&G_4 = (\neg P(f(a),a) \lor \neg P(a,f(a))) \text{ (from } C_1, \text{ substituting } f(a)\\
&\qquad \text{for } z_1 \text{ and } a \text{ for } x).
\end{aligned}$$

The following is a refutation by (ground) resolution of the set of ground clauses G_1, G_2, G_3, G_4.



We have the following useful result.

Lemma 8.3.1 (Completeness of ground resolution) The ground resolution method is complete for ground clauses.

Proof: Observe that the systems G' and $GCNF'$ are complete for quantifier-free formulae of a first-order language without equality. Hence, by theorem 4.3.1, the resolution method is also complete for sets of ground clauses. $\square$

However, note that this is not the case for quantifier-free formulae with equality, due to the need for equality axioms and for inessential cuts, in order to retain completeness.

Since we have shown that a conjunction of ground instances of the clauses C_1, C_2, C_3 of example 8.2.1 is unsatisfiable, by the Skolem-Herbrand-Gödel theorem, the sentence A of example 8.2.1 is valid.

Summarizing the above, we have a method for finding whether a sentence B is unsatisfiable known as *ground resolution*. This method consists in converting the sentence B into a set of clauses B', instantiating these clauses to ground clauses, and applying the ground resolution method.

By the completeness of resolution for propositional logic (theorem 4.3.1), and the Skolem-Herbrand-Gödel theorem (actually the corollary to theorem 7.6.1 suffices, since the clauses are in CNF, and so in NNF), this method is complete.

However, we were lucky to find so easily the ground clauses G_1, G_2, G_3 and G_4. In general, all one can do is enumerate ground instances one by one, testing for the unsatisfiabiliy of the current set of ground clauses each time. This can be a very costly process, both in terms of time and space.

8.4 Unification and the Unification Algorithm

The fundamental concept that allows the lifting of the ground resolution method to the first-order case is that of a most general unifier.

8.4.1 Unifiers and Most General Unifiers

We have already mentioned that Robinson has generalized ground resolution to arbitrary clauses, so that the systematic generation of ground clauses is unnecessary.

The new ingredient in this new form of resolution is that in forming the resolvent, one is allowed to apply substitutions to the parent clauses.

For example, to obtain $\{P(a, f(a))\}$ from

$$\begin{aligned} &C_1 = (\neg P(z_1, a) \vee \neg P(z_1, x) \vee \neg P(x, z_1)) \quad \text{and} \\ &C_2 = (P(z_2, f(z_2)) \vee P(z_2, a)), \end{aligned}$$

first we substitute a for z_1, a for x, and a for z_2, obtaining

$$G_1 = (\neg P(a, a)) \quad \text{and} \quad G_2 = (P(a, f(a)) \vee P(a, a)),$$

and then we resolve on the literal $P(a, a)$.

Note that the two sets of literals $\{P(z_1, a), P(z_1, x), P(x, z_1)\}$ and $\{P(z_2, a)\}$ obtained by dropping the negation sign in C_1 have been "unified" by the substitution $(a/x, a/z_1, a/z_2)$.

In general, given two clauses B and C whose variables are disjoint, given a substitution σ having as support the union of the sets of variables in B and C, if $\sigma(B)$ and $\sigma(C)$ contain a literal Q and its conjugate, there must be a subset $\{B_1, ..., B_m\}$ of the sets of literals of B, and a subset $\{\overline{C_1}, ..., \overline{C_n}\}$ of the set of literals in C such that

$$\sigma(B_1) = ... = \sigma(B_m) = \sigma(C_1) = ... = \sigma(C_n).$$

We say that σ is a *unifier* for the set of literals $\{B_1, ..., B_m, C_1, ..., C_n\}$. Robinson showed that there is an algorithm called the *unification algorithm*, for deciding whether a set of literals is unifiable, and if so, the algorithm yields what is called a *most general unifier* (Robinson, 1965). We will now explain these concepts in detail.

Definition 8.4.1 Given a substitution σ, let $D(\sigma) = \{x \mid \sigma(x) \neq x\}$ denote the *support* of σ, and let $I(\sigma) = \bigcup_{x \in D(\sigma)} FV(\sigma(x))$. Given two substitutions σ and θ, their *composition* denoted $\sigma \circ \theta$ is the substitution $\sigma \circ \widehat{\theta}$ (recall that $\widehat{\theta}$ is the unique homomorphic extension of θ). It is easily shown that the substitution $\sigma \circ \theta$ is the restriction of $\widehat{\sigma} \circ \widehat{\theta}$ to $\mathbf{V}$. If σ has support $\{x_1, ..., x_m\}$ and $\sigma(x_i) = s_i$ for $i = 1, ..., m$, we also denote the substitution σ by $(s_1/x_1, ..., s_m/x_m)$.

The notions of a unifier and a most general unifier are defined for arbitrary trees over a ranked alphabet (see Subsection 2.2.6). Since terms and atomic formulae have an obvious representation as trees (rigorously, since they are freely generated, we could define a bijection recursively), it is perfectly suitable to deal with trees, and in fact, this is intuitively more appealing due to the graphical nature of trees.

Definition 8.4.2 Given a ranked alphabet Σ, given any set $S = \{t_1, ..., t_n\}$ of finite Σ-trees, we say that a substitution σ is a *unifier of S* iff

$$\sigma(t_1) = ... = \sigma(t_n).$$

We say that a substitution σ is a *most general unifier* of S iff it is a unifier of S, the support of σ is a subset of the set of variables occurring in the set S, and for any other unifier σ' of S, there is a substitution θ such that

$$\sigma' = \sigma \circ \theta.$$

The tree $t = \sigma(t_1) = ... = \sigma(t_n)$ is called a *most common instance* of t_1,...,t_n.

EXAMPLE 8.4.1

(i) Let $t_1 = f(x, g(y))$ and $t_2 = f(g(u), g(z))$. The substitution $(g(u)/x, y/z)$ is a most general unifier yielding the most common instance $f(g(u), g(y))$.

(ii) However, $t_1 = f(x, g(y))$ and $t_2 = f(g(u), h(z))$ are not unifiable since this requires $g = h$.

(iii) A slightly more devious case of non unifiability is the following:

Let $t_1 = f(x, g(x), x)$ and $t_2 = f(g(u), g(g(z)), z)$. To unify these two trees, we must have $x = g(u) = z$. But we also need $g(x) = g(g(z))$, that is, $x = g(z)$. This implies $z = g(z)$, which is impossible for finite trees.

This last example suggest that unifying trees is similar to solving systems of equations by variable elimination, and there is indeed such an analogy. This analogy is explicated in Gorn, 1984. First, we show that we can reduce the problem of unifying any set of trees to the problem of unifying two trees.

Lemma 8.4.1 Let t_1,...,t_m be any m trees, and let # be a symbol of rank m not occurring in any of these trees. A substitution σ is a unifier for the set $\{t_1, ..., t_m\}$ iff σ is a unifier for the set $\{\#(t_1, ..., t_m), \#(t_1, ..., t_1)\}$.

Proof: Since a substitution σ is a homomorphism (see definition 7.5.3),

$$\sigma(\#(t_1, ..., t_m)) = \#(\sigma(t_1), ..., \sigma(t_m)) \quad \text{and}$$
$$\sigma(\#(t_1, ..., t_1)) = \#(\sigma(t_1), ..., \sigma(t_1)).$$

Hence,

$$\sigma(\#(t_1, ..., t_m)) = \sigma(\#(t_1, ..., t_1)) \quad \text{iff}$$
$$\#(\sigma(t_1), ..., \sigma(t_m)) = \#(\sigma(t_1), ..., \sigma(t_1)) \quad \text{iff}$$
$$\sigma(t_1) = \sigma(t_1),\ \sigma(t_2) = \sigma(t_1), ...,\ \sigma(t_m) = \sigma(t_1) \quad \text{iff}$$
$$\sigma(t_1) = ... = \sigma(t_m). \quad \square$$

Before showing that if a set of trees is unifiable then it has a most general unifier, we note that most general unifiers are essentially unique when they exist. Lemma 8.4.2 holds even if the support of mgu's is not a subset of $FV(S)$.

Lemma 8.4.2 If a set of trees S is unifiable and σ and θ are any two most general unifiers for S, then there exists a substitution ρ such that $\theta = \sigma \circ \rho$, ρ is a bijection between $I(\sigma) \cup (D(\theta) - D(\sigma))$ and $I(\theta) \cup (D(\sigma) - D(\theta))$, and $D(\rho) = I(\sigma) \cup (D(\theta) - D(\sigma))$ and $\rho(x)$ is a variable for every $x \in D(\rho)$.

Proof: First, note that a bijective substitution must be a bijective renaming of variables. Let $f|_A$ denote the restriction of a function f to A. If ρ is bijective, there is a substitution ρ' such that $(\rho \circ \rho')|_{D(\rho)} = Id$ and $(\rho' \circ \rho)|_{D(\rho')} = Id$. But then, if $\rho(x)$ is not a variable for some x in the support of ρ, $\rho(x)$ is a constant or a tree t of depth ≥ 1. Since $(\rho \circ \rho')|_{D(\rho)} = Id$, we have $\rho'(t) = x$. Since a substitution is a homomorphism, if t is a constant c, $\rho'(c) = c \neq x$, and otherwise $\rho'(t)$ has depth at least 1, and so $\rho'(t) \neq x$. Hence, $\rho(x)$ must be a variable for every x (and similarly for ρ'). A reasoning similar to the above also shows that for any two substitutions σ and ρ, if $\sigma = \sigma \circ \rho$, then ρ is the identity on $I(\sigma)$. But then, if both σ and θ are most general unifiers, there exist σ' and θ' such that $\theta = \sigma \circ \theta'$ and $\sigma = \theta \circ \sigma'$. Thus, $D(\sigma') = I(\theta) \cup (D(\sigma) - D(\theta))$, $D(\theta') = I(\sigma) \cup (D(\theta) - D(\sigma))$, $\theta = \theta \circ (\sigma' \circ \theta')$, and $\sigma = \sigma \circ (\theta' \circ \sigma')$. We claim that $(\sigma' \circ \theta')|_{D(\sigma')} = Id$, and $(\theta' \circ \sigma')|_{D(\theta')} = Id$. We prove that $(\sigma' \circ \theta')|_{D(\sigma')} = Id$, the other case being similar. For $x \in I(\theta)$, $\sigma' \circ \theta'(x) = x$ follows from above. For $x \in D(\sigma) - D(\theta)$, then $x = \theta(x) = \theta'(\sigma(x))$, and so, $\sigma(x) = y$, and $\theta'(y) = x$, for some variable y. Also, $\sigma(x) = y = \sigma'(\theta(x)) = \sigma'(x)$. Hence, $\sigma' \circ \theta'(x) = x$. Since $D(\theta')$ and $D(\sigma')$ are finite, θ' is a bijection between $D(\theta')$ and $D(\sigma')$. Letting $\rho = \theta'$, the lemma holds. $\square$

We shall now present a version of Robinson's unification algorithm.

8.4.2 The Unification Algorithm

In view of lemma 8.4.1, we restrict our attention to pairs of trees. The main idea of the unification algorithm is to find how two trees "disagree," and try to force them to agree by substituting trees for variables, if possible. There are two types of disagreements:

(1) Fatal disagreements, which are of two kinds:

  (i) For some tree address u both in $dom(t_1)$ and $dom(t_2)$, the labels $t_1(u)$ and $t_2(u)$ are not variables and $t_1(u) \neq t_2(u)$. This is illustrated by case (ii) in example 8.4.1;

  (ii) For some tree address u in both $dom(t_1)$ and $dom(t_2)$, $t_1(u)$ is a variable say x, and the subtree t_2/u rooted at u in t_2 is not a variable and x occurs in t_2/u (or the symmetric case in which $t_2(u)$ is a variable and t_1/u isn't). This is illustrated in case (iii) of example 8.4.1.

(2) Repairable disagreements: For some tree address u both in $dom(t_1)$ and $dom(t_2)$, $t_1(u)$ is a variable and the subtree t_2/u rooted at u in t_2 does not contain the variable $t_1(u)$.

In case (1), unification is impossible (although if we allowed infinite trees, disagreements of type (1)(ii) could be fixed; see Gorn, 1984). In case (2), we force "local agreement" by substituting the subtree t_2/u for all occurrences of the variable x in both t_1 and t_2.

It is rather clear that we need a systematic method for finding disagreements in trees. Depending on the representation chosen for trees, the method will vary. In most presentations of unification, it is usually assumed that trees are represented as parenthesized expressions, and that the two strings are scanned from left to right until a disagreement is found. However, an actual method for doing so is usually not given explicitly. We believe that in order to give a clearer description of the unification algorithm, it is better to be more explicit about the method for finding disagreements, and that it is also better not to be tied to any string representation of trees. Hence, we will give a recursive algorithm inspired from J. A. Robinson's original algorithm, in which trees are defined in terms of tree domains (as in Section 2.2), and the disagreements are discovered by performing two parallel top-down traversals of the trees t_1 and t_2.

The type of traversal that we shall be using is a recursive traversal in which the root is visited first, and then, from left to right, the subtrees of the root are recursively visited (this kind of traversal is called a *preorder traversal*, see Knuth, 1968, Vol. 1). We define some useful functions on trees. (The reader is advised to review the definitions concerning trees given in Section 2.2.)

Definition 8.4.3 For any tree t, for any tree address $u \in dom(t)$:

$leaf(u) =$ *true* iff u is a leaf;

$variable(t(u)) =$ *true* iff $t(u)$ is a variable;

$left(u) =$ *if* $leaf(u)$ *then nil else* $u1$;

$right(ui) =$ *if* $u(i+1) \in dom(t)$ *then* $u(i+1)$ *else nil.*

We also assume that we have a function *dosubstitution*(t, σ), where t is a tree and σ is a substitution.

Definition 8.4.4 (A unification algorithm) The formal parameters of the algorithm *unification* are the two input trees t_1 and t_2, an output flag indicating whether the two trees are unifiable or not (*unifiable*), and a most general unifier (*unifier*) (if it exists).

The main program *unification* calls the recursive procedure *unify*, which performs the unification recursively and needs procedure *test-and-substitute* to repair disagreements found, as in case (2) discussed above. The variables *tree*1 and *tree*2 denote trees (of type *tree*), and the variables *node*, *newnode* are tree addresses (of type *treereference*). The variable *unifier* is used to build a most general unifier (if any), and the variable *newpair* is used to form a new substitution component (of the form (t/x), where t is a tree and x is a variable). The function *compose* is simply function composition, where *compose*(*unifier*, *newpair*) is the result of composing *unifier* and *newpair*, in this order. The variables *tree*1, *tree*2, and *node* are global variables to the procedure *unification*. Whenever a new disageement is resolved in *test-and-substitute*, we also apply the substitution *newpair* to *tree*1 and *tree*2 to remove the disagreement. This step is not really necessary, since at any time, *dosubstitution*$(t_1, unifier) = tree1$ and *dosubstitution*$(t_2, unifier) = tree2$, but it simplifies the algorithm.

Procedure to Unify Two Trees t_1 and t_2

```
procedure unification($t_1, t_2$ : tree; var unifiable : boolean;
                                         var unifier : substitution);
  var node : treereference; tree1, tree2 : tree;

  procedure test-and-substitute(var node : treereference;
    var tree1, tree2 : tree;
    var unifier : substitution; var unifiable : boolean);
  var newpair : substitution;

  {This procedure tests whether the variable tree1(node) belongs
  to the subtree of tree2 rooted at node. If it does, the
  unification fails. Otherwise, a new substitution newpair
  consisting of the subtree tree2/node and the variable
  tree1(node) is formed, the current unifier is composed with
  newpair, and the new pair is added to the unifier.
  To simplify the algorithm, we also apply newpair to tree1
  and tree2 to remove the disagreement}

  begin
  {test whether the variable tree1(node) belongs to the
   subtree tree2/node, known in the literature as "occur check"}
```

```
            if $tree1(node) \in tree2/node$ then
              $unifiable := \textbf{false}$
            else

              {create a new substitution pair consisting of the
              subtree $tree2/node$ at address $node$, and the
              variable $tree1(node)$ at $node$ in $tree1$}

              $newpair := ((tree2/node)/tree1(node))$;

              {compose the current partial unifier with
              the new pair $newpair$}

              $unifier := compose(unifier, newpair)$;

              {updates the two trees so that they now agree on
              the subtrees at $node$}

              $tree1 := dosubstitution(tree1, newpair)$;
              $tree2 := dosubstitution(tree2, newpair)$
            endif
          end test-and-substitute;

procedure unify(var $node : treereference$;
    var $unifiable$ : boolean; var $unifier : substitution$);
var $newnode : treereference$;

{Procedure unify recursively unifies the subtree
of $tree1$ at $node$ and the subtree of $tree2$ at $node$}

  begin
    if $tree1(node) <> tree2(node)$ then
      {the labels of $tree1(node)$ and $tree2(node)$ disagree}
      if $variable(tree1(node))$ or $variable(tree2(node))$ then
        {one of the two labels is a variable}
        if $variable(tree1(node))$ then
          $test\text{-}and\text{-}substitute(node, tree1, tree2, unifier, unifiable)$
        else
          $test\text{-}and\text{-}substitute(node, tree2, tree1, unifier, unifiable)$
        endif
      else
        {the labels of $tree1(node)$ and $tree2(node)$
        disagree and are not variables}
        $unifiable := \textbf{false}$
      endif
    endif;
```

```
        {At this point, if $unifiable$ = true, the labels at node
        agree. We recursively unify the immediate subtrees of node
        in tree1 and tree2 from left to right, if node is not a leaf}

        if ($left(node) <> nil$) and $unifiable$ then
          $newnode := left(node)$;
          while ($newnode <> nil$) and $unifiable$ do
            $unify(newnode, unifiable, unifier)$;
            if $unifiable$ then
              $newnode := right(newnode)$
            endif
          endwhile
        endif
      end unify;
```

Body of Procedure Unification

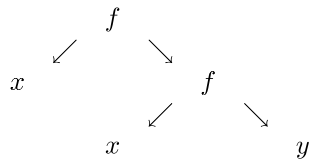
```
begin
  $tree1 := t_1$;
  $tree2 := t_2$;
  $unifiable :=$ true;
  $unifier := nil$;        {empty unification}
  $node := e$;             {start from the root}
  $unify(node, unifiable, unifier)$
end unification
```

Note that if successful, the algorithm could also return the tree *tree*1 (or *tree*2), which is a most common form of t_1 and t_2. As presented, the algorithm performs a single parallel traversal, but we also have the cost of the *occur check* in *test-and-substitute*, and the cost of the substitutions. Let us illustrate how the algorithm works.

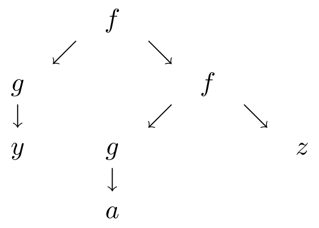
EXAMPLE 8.4.2

Let $t_1 = f(x, f(x, y))$ and $t_2 = f(g(y), f(g(a), z))$, which are represented as trees as follows:

Tree t_1

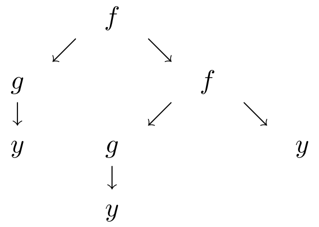


Tree t_2

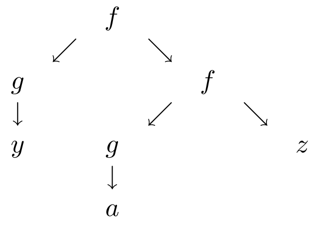


Initially, $tree1 = t_1$, $tree2 = t_2$ and $node = e$. The first disagreement is found for $node = 1$. We form $newpair = (g(y)/x)$, and $unifier = newpair$. After applying $newpair$ to $tree1$ and $tree2$, we have:

Tree $tree1$

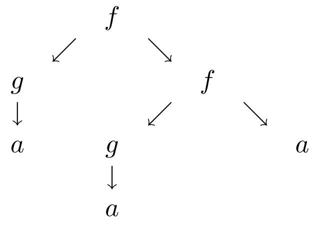


Tree $tree2$

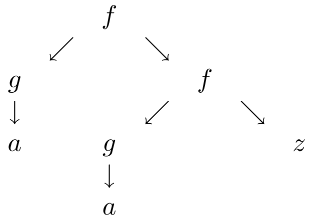


The next disagreement is found for $node = 211$. We find that $newpair = (a/y)$, and compose $unifier = (g(y)/x)$ with $newpair$, obtaining $(g(a)/x, a/y)$. After applying $newpair$ to $tree1$ and $tree2$, we have:

Tree $tree1$



Tree *tree*2



The last disagreement occurs for $node = 22$. We form $newpair = (a/z)$, and compose $unifier$ with $newpair$, obtaining

$$unifier = (g(a)/x, a/y, a/z).$$

The algorithm stops successfully with the most general unifier $(g(a)/x, a/y, a/z)$, and the trees are unified to the last value of $tree1$.

In order to prove the correctness of the unification algorithm, the following lemma will be needed.

Lemma 8.4.3 Let $\#$ be any constant. Given any two trees $f(s_1, ..., s_n)$ and $f(t_1, ..., t_n)$ the following properties hold:

(a) For any i, $1 \leq i \leq n$, if σ is a most general unifier for the trees

$$\begin{array}{rcll} f(s_1, ..., s_{i-1}, \#, ..., \#) & \text{and} & f(t_1, ..., t_{i-1}, \#, ..., \#), & \text{then} \\ f(s_1, ..., s_i, \#, ..., \#) & \text{and} & f(t_1, ..., t_i, \#, ..., \#) & \text{are unifiable iff} \\ \sigma(f(s_1, ..., s_i, \#, ..., \#)) & \text{and} & \sigma(f(t_1, ..., t_i, \#, ..., \#)) & \text{are unifiable.} \end{array}$$

(b) For any i, $1 \leq i \leq n$, if σ is a most general unifier for the trees $f(s_1, ..., s_{i-1}, \#, ..., \#)$ and $f(t_1, ..., t_{i-1}, \#, ..., \#)$, and θ is a most general unifier for the trees $\sigma(s_i)$ and $\sigma(t_i)$, then $\sigma \circ \theta$ is a most general unifier for the trees $f(s_1, ..., s_i, \#, ..., \#)$ and $f(t_1, ..., t_i, \#, ..., \#)$.

Proof: (a) The case $i = 1$ is trivial. Clearly, if σ is a most general unifier for the trees $f(s_1, ..., s_{i-1}, \#, ..., \#)$ and $f(t_1, ..., t_{i-1}, \#, ..., \#)$ and if the trees $\sigma(f(s_1, ..., s_i, \#, ..., \#))$ and $\sigma(f(t_1, ..., t_i, \#, ..., \#))$ are unifiable, then $f(s_1, ..., s_i, \#, ..., \#)$ and $f(t_1, ..., t_i, \#, ..., \#)$ are unifiable.

We now prove the other direction. Let θ be a unifier for

$$f(s_1, ..., s_i, \#, ..., \#) \quad \text{and} \quad f(t_1, ..., t_i, \#, ..., \#).$$

Then,

$$\theta(s_1) = \theta(t_1), ..., \theta(s_i) = \theta(t_i).$$

Hence, θ is a unifier for

$$f(s_1, ..., s_{i-1}, \#, ..., \#) \quad \text{and} \quad f(t_1, ..., t_{i-1}, \#, ..., \#).$$

Since σ is a most general unifier, there is some θ' such that $\theta = \sigma \circ \theta'$. Then,

$$\begin{aligned}&\theta'(\sigma(f(s_1, ..., s_i, \#, ..., \#))) = \theta(f(s_1, ..., s_i, \#, ..., \#))\\ &\quad = \theta(f(t_1, ..., t_i, \#, ..., \#)) = \theta'(\sigma(f(t_1, ..., t_i, \#, ..., \#))),\end{aligned}$$

which shows that θ' unifies

$$\sigma(f(s_1, ..., s_i, \#, ..., \#)) \quad \text{and} \quad \sigma(f(t_1, ..., t_i, \#, ..., \#)).$$

(b) Again, the case $i = 1$ is trivial. Otherwise, clearly,

$$\sigma(s_1) = \sigma(t_1), ..., \sigma(s_{i-1}) = \sigma(t_{i-1}) \text{ and } \theta(\sigma(s_i)) = \theta(\sigma(t_i))$$

implies that $\sigma \circ \theta$ is a unifier of

$$f(s_1, ..., s_i, \#, ..., \#) \quad \text{and} \quad f(t_1, ..., t_i, \#, ..., \#).$$

If λ unifies $f(s_1, ..., s_i, \#, ..., \#)$ and $f(t_1, ..., t_i, \#, ..., \#)$, then

$$\lambda(s_1) = \lambda(t_1), ..., \lambda(s_i) = \lambda(t_i).$$

Hence, λ unifies

$$f(s_1, ..., s_{i-1}, \#, ..., \#) \quad \text{and} \quad f(t_1, ..., t_{i-1}, \#, ..., \#).$$

Since σ is a most general unifier of these two trees, there is some σ' such that $\lambda = \sigma \circ \sigma'$. But then, since $\lambda(s_i) = \lambda(t_i)$, we have $\sigma'(\sigma(s_i)) = \sigma'(\sigma(t_i))$, and since θ is a most general unifier of $\sigma(s_i)$ and $\sigma(t_i)$, there is some θ' such that $\sigma' = \theta \circ \theta'$. Hence,

$$\lambda = \sigma \circ (\theta \circ \theta') = (\sigma \circ \theta) \circ \theta',$$

which proves that $\sigma \circ \theta$ is a most general unifier of $f(s_1, ..., s_i, \#, ..., \#)$ and $f(t_1, ..., t_i, \#, ..., \#)$. $\square$

We will now prove the correctness of the unification algorithm.

Theorem 8.4.1 (Correctness of the unification algorithm) (i) Given any two finite trees t_1 and t_2, the unification algorithm always halts. It halts with output $unifiable =$ **true** iff t_1 and t_2 are unifiable.

(ii) If t_1 and t_2 are unifiable, then they have a most general unifier and the output of procedure $unify$ is a most general unifier.

Proof: Clearly, the procedure *test-and-substitute* always terminates, and we only have to prove the termination of the $unify$ procedure. The difficulty in proving termination is that the trees $tree1$ and $tree2$ may grow. However, this can only happen if *test-and-substitute* is called, and in that case, since unifiable is not **false** iff the variable $x = tree1(node)$ does not belong to $t = tree2/node$, after the substitution of t for all occurrences of x in both $tree1$ and $tree2$, the variable x has been completely eliminated from both $tree1$ and $tree2$. This suggests to try a proof by induction over the well-founded lexicographic ordering $<<$ defined such that, for all pairs (m, t) and (m', t'), where m, m' are natural numbers and t, t' are finite trees,

$$(m, t) << (m', t') \quad \text{iff either } m < m',$$
$$\text{or } m = m' \text{ and } t \text{ is a proper subtree of } t'.$$

We shall actually prove the input-output correctness assertion stated below for the procedure $unify$.

Let s_0 and t_0 be two given finite trees, σ a substitution such that none of the variables in the support of σ is in $\sigma(s_0)$ or $\sigma(t_0)$, u any tree address in both $dom(\sigma(s_0))$ and $dom(\sigma(t_0))$, and let $s = \sigma(s_0)/u$ and $t = \sigma(t_0)/u$. Let $tree1_0$, $tree2_0$, $node_0$, $unifiable_0$ and $unifier_0$ be the input values of the variables $tree1$, $tree2$, $unifiable$, and $unifier$, and $tree1'$, $tree2'$, $node'$ $unifiable'$ and $unifier'$ be their output value (if any). Also, let m_0 be the sum of the number of variables in $\sigma(s_0)$ and $\sigma(t_0)$, and m' the sum of the number of variables in $tree1'$ and $tree2'$.

Correctness assertion:

$$\text{If } tree1_0 = \sigma(s_0), \quad tree2_0 = \sigma(t_0), \quad node_0 = u,$$
$$unifiable_0 = \textbf{true} \quad \text{and} \quad unifier_0 = \sigma, \text{ then}$$

the following holds:

(1) The procedure $unify$ always terminates;

(2) $unifiable' = \textbf{true}$ iff s and t are unifiable and, if $unifiable' = \textbf{true}$, then $unifier' = \sigma \circ \theta$, where θ is a most general unifier of s and t, $tree1' = unifier'(s_0)$, $tree2' = unifier'(t_0)$, and no variable in the support of $unifier'$ occurs in $tree1'$ or $tree2'$.

(3) If $tree1' \neq \sigma(s_0)$ or $tree2' \neq \sigma(t_0)$ then $m' < m_0$, else $m' = m_0$.

Proof of assertion: We proceed by complete induction on (m, s), where m is the sum of the number of variables in $tree1$ and $tree2$ and s is the subtree $tree1/node$.

(i) Assume that s is a constant and t is not a variable, the case in which t is a constant being similar. Then u is a leaf node in $\sigma(s_0)$. If $t \neq s$, the comparison of $tree1(node)$ and $tree2(node)$ fails, and $unifiable$ is set to **false**. The procedure terminates with failure. If $s = t$, since u is a leaf node in $\sigma(s_0)$ and $\sigma(t_0)$, the procedure terminates with success, $tree1' = \sigma(s_0)$,

$tree2' = \sigma(t_0)$, and $unifier' = \sigma$. Hence the assertion holds with the identity substitution for θ.

(ii) Assume that s is a variable say x, the case in which t is a variable being similar. Then u is a leaf node in $\sigma(s_0)$. If $t = s$, this case reduces to case (i). Otherwise, $t \neq x$ and the occur check is performed in *test-and-substitute*. If x occurs in t, then $unifiable$ is set to **false**, and the procedure terminates. In this case, it is clear that x and t are not unifiable, and the assertion holds. Otherwise, the substitution $\theta = (t/x)$ is created, $unifier' = \sigma \circ \theta$, and $tree1' = \theta(\sigma(s_0)) = unifier'(s_0)$, $tree2' = \theta(\sigma(t_0)) = unifier'(t_0)$. Clearly, θ is a most general unifier of x and t, and since x does not occur in t, since no variable in the support of σ occurs in $\sigma(s_0)$ or $\sigma(t_0)$, no variable in the support of $unifier'$ occurs in $tree1' = \theta(\sigma(s_0))$ or $tree2' = \theta(\sigma(s_0))$. Since the variable x does not occur in $tree1'$ and $tree2'$, (3) also holds. Hence, the assertion holds.

(iii) Both s and t have *depth* ≥ 1. Assume that $s = f(s_1, ..., s_m)$ and $t = f'(t_1, ..., t_n)$. If $f \neq f'$, the test $tree1(node) = tree2(node)$ fails, and $unify$ halts with failure. Clearly, s and t are not unifiable, and the claim holds. Otherwise, $s = f(s_1, ..., s_n)$ and $t = f(t_1, ..., t_n)$.

We shall prove the following claim by induction:

Claim: (1) For every i, $1 \leq i \leq n+1$, the first $i-1$ recursive calls in the while loop in $unify$ halt with success iff $f(s_1, ..., s_{i-1}, \#, ..., \#)$ and $f(t_1, ..., t_{i-1}, \#, ..., \#)$ are unifiable, and otherwise one of the calls halts with failure;

(2) If the first $i-1$ recursive calls halt with success, the input values at the end of the $(i-1)$-th iteration are:

$$node_i = ui, \quad unifiable_i = \textbf{true}, \quad unifier_i = \sigma \circ \theta_{i-1},$$

where θ_{i-1} is a most general unifier for the trees $f(s_1, ..., s_{i-1}, \#, ..., \#)$ and $f(t_1, ..., t_{i-1}, \#, ..., \#)$, (with $\theta_0 = Id$, the identity substitution),

$$tree1_i = unifier_i(s_0), \quad tree2_i = unifier_i(t_0),$$

and no variable in the support of $unifier_i$ occurs in $tree1_i$ or $tree2_i$.

(3) If $tree1_i \neq tree1_0$ or $tree2_i \neq tree2_0$, if m_i is the sum of the number of variables in $tree1_i$ and $tree2_i$, then $m_i < m_0$.

Proof of claim: For $i = 1$, the claim holds because before entering the while loop for the first time,

$$tree1_1 = s_0, \quad tree2_1 = t_0, \quad node_1 = u1,$$
$$unifier_1 = \sigma, \quad unifiable_1 = \textbf{true}.$$

Now, for the induction step. We only need to consider the case where the first $i-1$ recursive calls were successful. If we have $tree1_i = tree1_0$ and

$tree2_i = tree2_0$, then we can apply the induction hypothesis for the assertion to the address ui, since $tree1_0/ui$ is a proper subtree of $tree1_0/u$. Otherwise, $m_i < m_0$, and we can also also apply the induction hypothesis for the assertion to address ui. Note that

$$\begin{aligned} &tree1_i/u = \theta_{i-1}(f(s_1, ..., s_i, ..., s_n)) \quad \text{and} \\ &tree2_i/u = \theta_{i-1}(f(t_1, ..., t_i, ..., s_n)), \quad \text{since} \\ &unifier_i = \sigma \circ \theta_{i-1}. \end{aligned}$$

By lemma 8.4.3(a), since θ_{i-1} is a most general unifier for the trees

$$\begin{aligned} f(s_1, ..., s_{i-1}, \#, ..., \#) \quad &\text{and} \quad f(t_1, ..., t_{i-1}, \#, ..., \#), \quad \text{then} \\ f(s_1, ..., s_i, \#, ..., \#) \quad &\text{and} \quad f(t_1, ..., t_i, \#, ..., \#) \quad \text{are unifiable, iff} \\ \theta_{i-1}(f(s_1, ..., s_i, \#, ..., \#)) \quad &\text{and} \quad \theta_{i-1}f((t_1, ..., t_i, \#, ..., \#)) \quad \text{are unifiable.} \end{aligned}$$

Hence, $unify$ halts with success for this call for address ui, iff

$$f(s_1, ..., s_i, \#, ..., \#) \quad \text{and} \quad f(t_1, ..., t_i, \#, ..., \#) \quad \text{are unifiable.}$$

Otherwise, $unify$ halts with failure. This proves part (1) of the claim.

By part (2) of the assertion, the output value of the variable $unifier$ is of the form $unifier_i \circ \lambda_i$, where λ_i is a most general unifier for $\theta_{i-1}(s_i)$ and $\theta_{i-1}(t_i)$ (the subtrees at ui), and since θ_{i-1} is a most general unifier for

$$f(s_1, ..., s_{i-1}, \#, ..., \#) \quad \text{and} \quad f(t_1, ..., t_{i-1}, \#, ..., \#),$$

λ_i is a most general unifier for

$$\theta_{i-1}(f(s_1, ..., s_i, \#, ..., \#)) \quad \text{and} \quad \theta_{i-1}f((t_1, ..., t_i, \#, ..., \#)).$$

By lemma 8.4.3(b), $\theta_{i-1} \circ \lambda_i$ is a most general unifier for

$$f(s_1, ..., s_i, \#, ..., \#) \quad \text{and} \quad f(t_1, ..., t_i, \#, ..., \#).$$

Letting

$$\theta_i = \theta_{i-1} \circ \lambda_i,$$

it is easily seen that part (2) of the claim is satisfied. By part (3) of the assertion, part (3) of the claim also holds.

This concludes the proof of the claim. □

For $i = n + 1$, we see that all the recursive calls in the while loop halt successfully iff s and t are unifiable, and if s and t are unifiable, when the loop is exited, we have

$$unifier_{n+1} = \sigma \circ \theta_n,$$

where θ_n is a most general unifier of s and t,

$$tree1_{n+1} = unifier_{n+1}(s_0), \quad tree2_{n+1} = unifier_{n+1}(t_0),$$

and part (3) of the assertion also holds. This concludes the proof of the assertion. $\square$

But now, we can apply the assertion to the input trees t_1 and t_2, with $u = e$, and σ the identity substitution. The correctness assertion says that *unify* always halts, and if it halts with success, the output variable *unifier* is a most general unifier for t_1 and t_2. This concludes the correctness proof. $\square$

The subject of unification is the object of current research because fast unification is crucial for the efficiency of programming logic systems such as PROLOG. Some fast unification algorithms have been published such as Paterson and Wegman, 1978; Martelli and Montanari, 1982; and Huet, 1976. For a survey on unification, see the article by Siekmann in Shostak, 1984a. Huet, 1976, also contains a thorough study of unification, including higher-order unification.

PROBLEMS

8.4.1. Convert the following formulae to clause form:

$$\forall y(\exists x(P(y,x) \lor \neg Q(y,x)) \land \exists x(\neg P(x,y) \lor Q(x,y)))$$
$$\forall x(\exists y P(x,y) \land \neg Q(y,x)) \lor (\forall y \exists z(R(x,y,z) \land \neg Q(y,z)))$$
$$\neg(\forall x \exists y P(x,y) \supset (\forall y \exists z \neg Q(x,z) \land \forall y \neg \forall z R(y,z)))$$
$$\forall x \exists y \forall z(\exists w(Q(x,w) \lor R(x,y)) \equiv \neg \exists w \neg \exists u(Q(x,w) \land \neg R(x,u)))$$

8.4.2. Apply the unification algorithm to the following clauses:

$$\{P(x,y), P(y,f(z))\}$$
$$\{P(a,y,f(y)), P(z,z,u)\}$$
$$\{P(x,g(x)), P(y,y)\}$$
$$\{P(x,g(x),y), P(z,u,g(u))\}$$
$$\{P(g(x),y), P(y,y), P(u,f(w))\}$$

8.4.3. Let S and T be two finite sets of terms such that the set of variables occurring in S is disjoint from the set of variables occurring in T. Prove that if $S \cup T$ is unifiable, σ_S is a most general unifier of S, σ_T is a most general unifier of T, and $\sigma_{S,T}$ is a most general unifier of $\sigma_S(S) \cup \sigma_T(T)$, then

$$\sigma_S \circ \sigma_T \circ \sigma_{S,T}$$

is a most general unifier of $S \cup T$.

8.4.4. Show that the most general unifier of the following two trees contains a tree with 2^{n-1} occurrences of the variable x_1:

$$f(g(x_1, x_1), g(x_2, x_2), ..., g(x_{n-1}, x_{n-1})) \quad \text{and}$$
$$f(x_2, x_3, ..., x_n)$$

∗ **8.4.5.** Define the relation $\leq$ on terms as follows: Given any two terms t_1, t_2,

$$t_1 \leq t_2 \quad \text{iff} \quad \text{there is a substitution } \sigma \text{ such that } t_2 = \sigma(t_1).$$

Define the relation $\cong$ such that

$$t_1 \cong t_2 \quad \text{iff} \quad t_1 \leq t_2 \text{ and } t_2 \leq t_1.$$

(a) Prove that $\leq$ is reflexive and transitive and that $\cong$ is an equivalence relation.

(b) Prove that $t_1 \cong t_2$ iff there is a bijective renaming of variables ρ such that $t_1 = \rho(t_2)$. Show that the relation $\leq$ induces a partial ordering on the set of equivalence classes of terms modulo the equivalence relation $\cong$.

(c) Prove that two terms have a least upper bound iff they have a most general unifier (use a separating substitution, see Section 8.5).

(d) Prove that any two terms always have a greatest lower bound.

Remark: The structure of the set of equivalence classes of terms modulo $\cong$ under the partial ordering $\leq$ has been studied extensively in Huet, 1976. Huet has shown that this set is well founded, that every subset has a greatest lower bound, and that every bounded subset has a least upper bound.

8.5 The Resolution Method for First-Order Logic

Recall that we are considering first-order languages without equality. Also, recall that even though we usually omit quantifiers, clauses are universally quantified sentences. We extend the definition of a resolvent given in definition 4.3.2 to arbitrary clauses using the notion of a most general unifier.

8.5.1 Definition of the Method

First, we define the concept of a separating pair of substitutions.

Definition 8.5.1 Given two clauses A and A', a *separating pair of substitutions* is a pair of substitutions ρ and ρ' such that:

ρ has support $FV(A)$, ρ' has support $FV(A')$, for every variable x in A, $\rho(x)$ is a variable, for every variable y in A', $\rho'(y)$ is a variable, ρ and ρ' are bijections, and the range of ρ and the range of ρ' are disjoint.

Given a set S of literals, we say that S is *positive* if all literals in S are atomic formulae, and we say that S is *negative* if all literals in S are negations of atomic formulae. If a set S is positive or negative, we say that the literals in S are of the *same sign*. Given a set of literals $S = \{A_1, ..., A_m\}$, the *conjugate* of S is defined as the set

$$\overline{S} = \{\overline{A_1}, ..., \overline{A_m}\}$$

of conjugates of literals in S. If S is a positive set of literals we let $|S| = S$, and if S is a negative set of literals, we let $|S| = \overline{S}$.

Definition 8.5.2 Given two clauses A and B, a clause C is a *resolvent* of A and B iff the following holds:

(i) There is a subset $A' = \{A_1, ..., A_m\} \subseteq A$ of literals all of the same sign, a subset $B' = \{B_1, ..., B_n\} \subseteq B$ of literals all of the opposite sign of the set A', and a separating pair of substitutions (ρ, ρ') such that the set

$$|\rho(A') \cup \rho'(\overline{B'})|$$

is unifiable;

(ii) For some most general unifier σ of the set

$$|\rho(A') \cup \rho'(\overline{B'})|,$$

we have

$$C = \sigma(\rho(A - A') \cup \rho'(B - B')).$$

EXAMPLE 8.5.1

Let

$$A = \{\neg P(z, a), \neg P(z, x), \neg P(x, z)\} \quad \text{and}$$
$$B = \{P(z, f(z)), P(z, a)\}.$$

Let

$$A' = \{\neg P(z, a), \neg P(z, x)\} \quad \text{and} \quad B' = \{P(z, a)\},$$
$$\rho = (z_1/z), \quad \rho' = (z_2/z).$$

Then,

$$|\rho(A') \cup \rho'(\overline{B'})| = \{P(z_1, a), P(z_1, x), P(z_2, a)\}$$

is unifiable,

$$\sigma = (z_1/z_2, a/x)$$

is a most general unifier, and

$$C = \{\neg P(a, z_1), P(z_1, f(z_1))\}$$

is a resolvent of A and B.

If we take $A' = A$, $B' = \{P(z, a)\}$,

$$|\rho(A') \cup \rho'(\overline{B'})| = \{P(z_1, a), P(z_1, x), P(x, z_1), P(z_2, a)\}$$

is also unifiable,

$$\sigma = (a/z_1, a/z_2, a/x)$$

is the most general unifier, and

$$C = \{P(a, f(a))\}$$

is a resolvent.

Hence, two clauses may have several resolvents.

The generalization of definition 4.3.3 of a resolution DAG to the first-order case is now obvious.

Definition 8.5.3 Given a set $S = \{C_1, ..., C_n\}$ of first-order clauses, a *resolution DAG* for S is any finite set

$$G = \{(t_1, R_1), ..., (t_m, R_m)\}$$

of distinct DAGs labeled in the following way:

(1) The leaf nodes of each underlying tree t_i are labeled with clauses in S.

(2) For every DAG (t_i, R_i), every nonleaf node u in t_i is labeled with some triple $(C, (\rho, \rho'), \sigma)$, where C is a clause, (ρ, ρ') is a separating pair of substitutions, σ is a substitution and the following holds:

For every nonleaf node u in t_i, u has exactly two successors $u1$ and $u2$, and if $u1$ is labeled with a clause C_1 and u_2 is labeled with a clause C_2 (not necessarily distinct from C_1), then u is labeled with the triple $(C, (\rho, \rho'), \sigma)$, where (ρ, ρ') is a separating pair of substitutions for C_1 and C_2 and C is the resolvent of C_1 and C_2 obtained with the most general unifier σ.

A resolution DAG is a *resolution refutation* iff it consists of a single DAG (t, R) whose root is labeled with the empty clause. The nodes of a DAG that are not leaves are also called *resolution steps*.

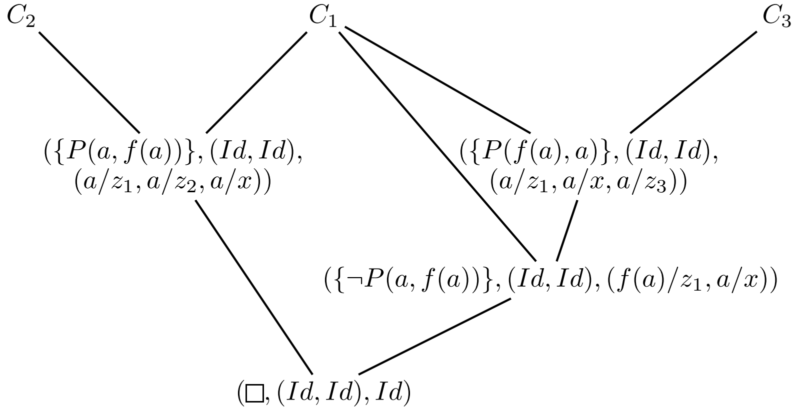
We will often use a simplified form of the above definition by dropping (ρ, ρ') and σ from the interior nodes, and consider that nodes are labeled with clauses. This has the effect that it is not always obvious how a resolvent is obtained.

EXAMPLE 8.5.2

Consider the following clauses:

$$\begin{aligned} &C_1 = \{\neg P(z_1, a), \neg P(z_1, x), \neg P(x, z_1)\}, \\ &C_2 = \{P(z_2, f(z_2)), P(z_2, a)\} \quad \text{and} \\ &C_3 = \{P(f(z_3), z_3), P(z_3, a)\}. \end{aligned}$$

The following is a resolution refutation:



8.5.2 Soundness of the Resolution Method

In order to prove the soundness of the resolution method, we prove the following lemma, analogous to lemma 4.3.1.

Lemma 8.5.1 Given two clauses A and B, let $C = \sigma(\rho(A - A') \cup \rho'(B - B'))$ be any resolvent of A and B, for some subset $A' \subseteq A$ of literals of A, subset $B' \subseteq B$ of literals of B, separating pair of substitutions (ρ, ρ'), with $\rho = (z_1/x_1, ..., z_m/x_m)$, $\rho' = (z_{m+1}/y_1, ..., z_{m+n}/y_n)$ and most general unifier $\sigma = (t_1/u_1, ..., t_k/u_k)$, where $\{u_1, ..., u_k\}$ is a subset of $\{z_1, ..., z_{m+n}\}$. Also, let $\{v_1, ..., v_p\} = FV(C)$. Then,

$$\models (\forall x_1 ... \forall x_m A \land \forall y_1 ... \forall y_n B) \supset \forall v_1 ... \forall v_p C.$$

Proof: We show that we can constuct a G-proof for

$$(\forall x_1 ... \forall x_m A \land \forall y_1 ... \forall y_n B) \rightarrow \forall v_1 ... \forall v_p C.$$

Note that $\{z_1, ..., z_{m+n}\} - \{u_1, ..., u_k\}$ is a subset of $\{v_1, ..., v_p\}$. First, we perform p $\forall : right$ steps using p entirely new variables w_1,...,w_p. Let

$$\sigma' = \sigma \circ (w_1/v_1, ..., w_p/v_p) = (t'_1/z_1, ..., t'_{m+n}/z_{m+n}),$$

be the substitution obtained by composing σ and the substitution replacing each occurrence of the variable v_i by the variable w_i. Then, note that the support of σ' is disjoint from the set $\{w_1, ..., w_p\}$, which means that for every tree t,

$$\sigma'(t) = t[t'_1/z_1]...[t'_{m+n}/z_{m+n}]$$

(the order being irrelevant). At this point, we have the sequent

$$(\forall x_1 ... \forall x_m A \wedge \forall y_1 ... \forall y_n B) \rightarrow \sigma'(\rho(A - A')), \sigma'(\rho'(B - B')).$$

Then apply the $\wedge : left$ rule, obtaining

$$\forall x_1 ... \forall x_m A, \forall y_1 ... \forall y_n B \rightarrow \sigma'(\rho(A - A')), \sigma'(\rho'(B - B')).$$

At this point, we apply $m+n$ $\forall : left$ rules as follows: If $\rho(x_i)$ is some variable u_j, do the substitution t'_j/x_i, else $\rho(x_i)$ is some variable v_j not in $\{u_1, .., u_k\}$, do the substitution w_j/v_j.

If $\rho'(y_i)$ is some variable u_j, do the substitution t'_j/y_i, else $\rho'(y_j)$ is some variable v_j not in $\{u_1, ..., u_k\}$, do the substitution w_j/v_j.

It is easy to verify that at the end of these steps, we have the sequent

$$(\sigma'(\rho(A - A')), Q), (\sigma'(\rho'(B - B')), \overline{Q}) \rightarrow \sigma'(\rho(A - A')), \sigma'(\rho'(B - B'))$$

where $Q = \sigma'(\rho(A'))$ and $\overline{Q} = \sigma'(\rho'(B'))$ are conjugate literals, because σ is a most general unifier of the set $|\rho(A') \cup \rho'(\overline{B'})|$.

Hence, we have a quantifier-free sequent of the form

$$(A_1 \vee Q), (A_2 \vee \neg Q) \rightarrow A_1, A_2,$$

and we conclude that this sequent is valid using the proof of lemma 4.3.1. □

As a consequence, we obtain the soundness of the resolution method.

Lemma 8.5.2 (Soundness of resolution without equality) If a set of clauses has a resolution refutation DAG, then S is unsatisfiable.

Proof: The proof is identical to the proof of lemma 4.3.2, but using lemma 8.5.1, as opposed to lemma 4.3.1. □

8.5.3 Completeness of the Resolution Method

In order to prove the completeness of the resolution method for first-order languages without equality, we shall prove the following lifting lemma.

Lemma 8.5.3 (Lifting lemma) Let A and B be two clauses, σ_1 and σ_2 two substitutions such that $\sigma_1(A)$ and $\sigma_2(B)$ are ground, and assume that D is a resolvent of the ground clauses $\sigma_1(A)$ and $\sigma_2(B)$. Then, there is a resolvent C of A and B and a substitution θ such that $D = \theta(C)$.

Proof: First, let (ρ, ρ') be a separating pair of substitutions for A and B. Since ρ and ρ' are bijections they have inverses ρ^{-1} and $\rho^{'-1}$. Let σ be the substitution formed by the union of $\rho^{-1} \circ \sigma_1$ and $\rho^{'-1} \circ \sigma_2$, which is well defined, since the supports of ρ^{-1} and $\rho^{'-1}$ are disjoint. It is clear that

$$\sigma(\rho(A)) = \sigma_1(A) \quad \text{and} \quad \sigma(\rho'(B)) = \sigma_2(B).$$

Hence, we can work with $\rho(A)$ and $\rho'(B)$, whose sets of variables are disjoint. If D is a resolvent of the clauses $\sigma_1(A)$ and $\sigma_2(B)$, there is a ground literal Q such that $\sigma(\rho(A))$ contains Q and $\sigma(\rho'(B))$ contains its conjugate. Assume that Q is positive, the case in which Q is negative being similar. Then, there must exist subsets $A' = \{A_1, ..., A_m\}$ of A and $B' = \{\neg B_1, ..., \neg B_n\}$ of B, such that

$$\sigma(\rho(A_1)) = ... = \sigma(\rho(A_m)) = \sigma(\rho'(B_1)) = ..., \sigma(\rho'(B_n)) = Q,$$

and σ is a unifier of $\rho(A') \cup \rho'(\overline{B'})$. By theorem 8.4.1, there is a most general unifier λ and a substitution θ such that

$$\sigma = \lambda \circ \theta.$$

Let C be the resolvent

$$C = \lambda(\rho(A - A') \cup \rho'(B - B')).$$

Clearly,

$$\begin{aligned} D &= (\sigma(\rho(A)) - \{Q\}) \cup (\sigma(\rho'(B)) - \{\neg Q\}) \\ &= (\sigma(\rho(A - A')) \cup \sigma(\rho'(B - B'))) \\ &= \theta(\lambda(\rho(A - A') \cup \rho'(B - B'))) = \theta(C). \end{aligned}$$

□

Using the above lemma, we can now prove the following lemma which shows that resolution DAGs of ground instances of clauses can be lifted to resolution DAGs using the original clauses.

Lemma 8.5.4 (Lifting lemma for resolution refutations) Let S be a finite set of clauses, and S_g be a set of ground instances of S, so that every clause in

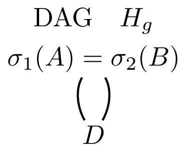
S_g is of the form $\sigma_i(C_i)$ for some clause C_i in S and some ground substitution σ_i.

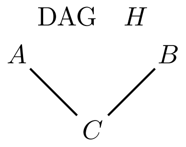
For any resolution DAG H_g for S_g, there is a resolution DAG H for S, such that the DAG H_g is a homomorphic image of the DAG H in the following sense:

There is a function $F : H \rightarrow H_g$ from the set of nodes of H to the set of nodes of H_g, such that, for every node u in H, if $u1$ and $u2$ are the immediate descendants of u, then $F(u1)$ and $F(u2)$ are the immediate descendants of $F(u)$, and if the clause C (not necessarily in S_g) is the label of u, then $F(u)$ is labeled by the clause $\theta(C)$, where θ is some ground substitution.

Proof: We prove the lemma by induction on the underlying tree of H_g.

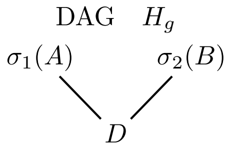
(i) If H_g has a single resolution step, we have clauses $\sigma_1(A)$, $\sigma_2(B)$ and their resolvent D. By lemma 8.5.3, there exists a resolvent C of A and B and a substitution θ such that $\theta(C) = D$. Note that it is possible that A and B are distinct, but $\sigma_1(A)$ and $\sigma_2(B)$ are not. In the first case, we have the following DAGs:

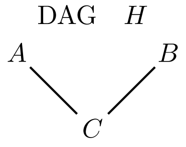




The homomorphism F is such that $F(e) = e$, $F(1) = 1$ and $F(2) = 1$.

In the second case, $\sigma_1(A) \neq \sigma_2(B)$, but we could have $A = B$. Whether or not $A = B$, we create the following DAG H with three distinct nodes, so that the homomorphism is well defined:

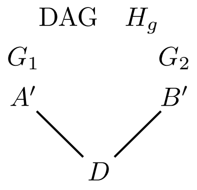




The homomorphism F is the identity on nodes.

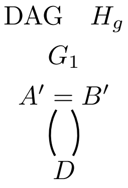
(ii) If H_g has more than one resolution step, it is of the form either

(ii)(a)



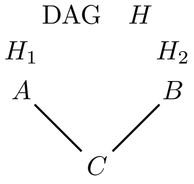
where A' and B' are distinct, or of the form

(ii)(b)



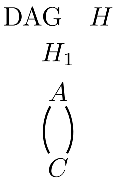
if $A' = B'$.

(a) In the first case, by the induction hypothesis, there are DAGs H_1 and H_2 and homomorphisms $F_1 : H_1 \rightarrow G_1$ and $F_2 : H_2 \rightarrow G_2$, where H_1 is rooted with some formula A and H_2 is rooted with some formula B, and for some ground substitutions θ_1 and θ_2, we have, $A' = \theta_1(A)$ and $B' = \theta_2(B)$. By lemma 8.5.3, there is a resolvent C of A and B and a substitution θ such that $\theta(C) = D$. We can construct H as the DAG obtained by making C as the root, and even if $A = B$, by creating two distinct nodes 1 and 2, with 1 labeled A and 2 labeled B:



The homomorphism $F : H \rightarrow H_g$ is defined such that $F(e) = e$, $F(1) = 1$, $F(2) = 2$, and it behaves like F_1 on H_1 and like F_2 on H_2. The root clause C is mapped to $\theta(C) = D$.

(b) In the second case, by the induction hypothesis, there is a DAG H_1 rooted with some formula A and a homomorphism $F_1 : H_1 \rightarrow G_1$, and for some ground substitution θ_1, we have $A' = \theta_1(A)$. By lemma 8.5.3, there is a resolvent C of A with itself, and a substitution θ such that $\theta(C) = D$. It is clear that we can form H so that C is a root node with two edges connected to A, and F is the homomorphism such that $F(e) = e$, $F(1) = 1$, and F behaves like F_1 on H_1.



The clause C is mapped onto $D = \theta(C)$. This concludes the proof. □

EXAMPLE 8.5.3

The following shows a lifting of the ground resolution of example 8.3.1 for the clauses:

$$
\begin{aligned}
C_1 &= \{\neg P(z_1, a), \neg P(z_1, x), \neg P(x, z_1)\}\\
C_2 &= \{P(z_2, f(z_2)), P(z_2, a)\}\\
C_3 &= \{P(f(z_3), z_3), P(z_3, a)\}.
\end{aligned}
$$

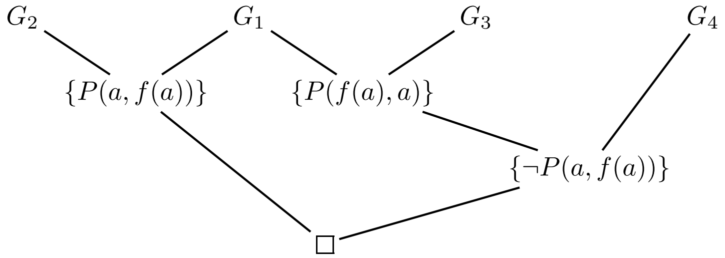
Recall that the ground instances are

$$
\begin{aligned}
G_1 &= \{\neg P(a, a)\}\\
G_2 &= \{P(a, f(a)), P(a, a)\}\\
G_3 &= \{P(f(a), a), P(a, a)\}\\
G_4 &= \{\neg P(f(a), a), \neg P(a, f(a))\},
\end{aligned}
$$

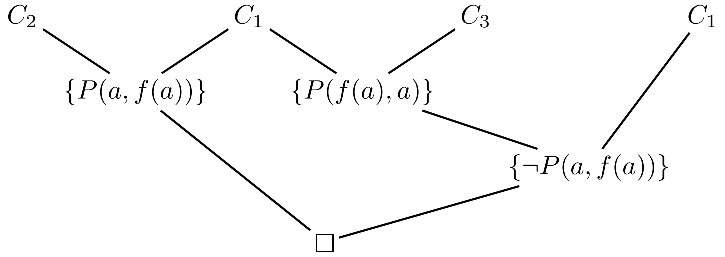
and the substitutions are

$$
\begin{aligned}
\sigma_1 &= (a/z_2)\\
\sigma_2 &= (a/z_1, a/x)\\
\sigma_3 &= (a/z_3)\\
\sigma_4 &= (f(a)/z_1, a/x).
\end{aligned}
$$

Ground resolution-refutation H_g
for the set of ground clauses G_1, G_2, G_3, G_4



Lifting H of the above resolution refutation
for the clauses C_1, C_2, C_3



The homomorphism is the identity on the nodes, and the substitutions are, (a/z_2) for node 11 labeled C_2, $(a/z_1, a/x)$ for node 12 labeled C_1,

(a/z_3) for node 212 labeled C_3, and $(f(a)/z_1, a/x)$ for node 22 labeled C_1.

Note that this DAG is not as concise as the DAG of example 8.5.1. This is because is has been designed so that there is a homomorphism from H to H_g.

As a consequence of the lifting theorem, we obtain the completeness of resolution.

Theorem 8.5.1 (Completeness of resolution, without equality) If a finite set S of clauses is unsatisfiable, then there is a resolution refutation for S.

Proof: By the Skolem-Herbrand-Gödel theorem (theorem 7.6.1, or its corollary), S is unsatisfiable iff a conjunction S_g of ground substitution instances of clauses in S is unsatisfiable. By the completeness of ground resolution (lemma 8.3.1), there is a ground resolution refutation H_g for S_g. By lemma 8.5.4, this resolution refutation can be lifted to a resolution refutation H for S. This concludes the proof. $\square$

Actually, we can also prove the following type of Herbrand theorem for the resolution method, using the constructive nature of lemma 7.6.2.

Theorem 8.5.2 (A Herbrand-like theorem for resolution) Consider a first-order language without equality. Given any prenex sentence A whose matrix is in CNF, if $A \rightarrow$ is LK-provable, then a resolution refutation of the clause form of A can be obtained constructively.

Proof: By lemma 7.6.2, a compound instance C of the Skolem form B of A can be obtained constructively. Observe that the Skolem form B of A is in fact a clause form of A, since A is in CNF. But C is in fact a conjunction of ground instances of the clauses in the clause form of A. Since $\neg C$ is provable, the *search* procedure will give a proof that can be converted to a $GCNF'$-proof. Since theorem 4.3.1 is constructive, we obtain a ground resolution refutation H_g. By the lifting lemma 8.5.4, a resolution refutation H can be constructively obtained for S_g. Hence, we have shown that a resolution refutation for the clause form of A can be constructively obtained from an LK-proof of $A \rightarrow$. $\square$

It is likely that theorem 8.5.2 has a converse, but we do not have a proof of such a result. A simpler result is to prove the converse of lemma 8.5.4, the lifting theorem. This would provide another proof of the soundness of resolution. It is indeed possible to show that given any resolution refutation H of a set S of clauses, a resolution refutation H_g for a certain set S_g of ground instances of S can be constructed. However, the homomorphism property does not hold directly, and one has to exercise care in the construction. The interested reader should consult the problems.

It should be noted that a Herbrand-like theorem for the resolution method and a certain Hilbert system has been proved by Joyner in his Ph.D thesis (Joyner, 1974). However, these considerations are somewhat beyond the scope of this text, and we will not pursue this matter any further.

PROBLEMS

8.5.1. Give separating pairs of substitutions for the following clauses:

$$\{P(x, y, f(z))\},\ \{P(y, z, f(z))\}$$
$$\{P(x, y), P(y, z)\},\ \{Q(y, z), P(z, f(y))\}$$
$$\{P(x, g(x))\},\ \{P(x, g(x))\}$$

8.5.2. Find all resolvents of the following pairs of clauses:

$$\{P(x, y), P(y, z)\},\ \{\neg P(u, f(u))\}$$
$$\{P(x, x), \neg R(x, f(x))\},\ \{R(x, y), Q(y, z)\}$$
$$\{P(x, y), \neg P(x, x), Q(x, f(x), z)\},\ \{\neg Q(f(x), x, z), P(x, z)\}$$
$$\{P(x, f(x), z), P(u, w, w)\},\ \{\neg P(x, y, z), \neg P(z, z, z)\}$$

8.5.3. Establish the unsatisfiability of each of the following formulae using the resolution method.

$$(\forall x \exists y P(x, y) \land \exists x \forall y \neg P(x, y))$$

$$(\forall x \exists y \exists z (L(x, y) \land L(y, z) \land Q(y) \land R(z) \land (P(z) \equiv R(x))) \land$$
$$\forall x \forall y \forall z ((L(x, y) \land L(y, z)) \supset L(x, z)) \land \exists x \forall y \neg (P(y) \land L(x, y)))$$

8.5.4. Consider the following formulae asserting that a binary relation is symmetric, transitive, and total:

$$S_1 : \forall x \forall y (P(x, y) \supset P(y, x))$$
$$S_2 : \forall x \forall y \forall z ((P(x, y) \land P(y, z)) \supset P(x, z))$$
$$S_3 : \forall x \exists y P(x, y)$$

Prove by resolution that

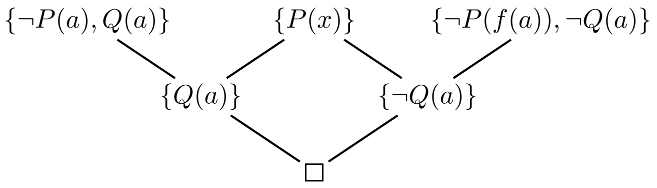
$$S_1 \land S_2 \land S_3 \supset \forall x P(x, x).$$

In other words, if P is symmetric, transitive and total, then P is reflexive.

8.5.5. Complete the details of the proof of lemma 8.5.1.

∗ **8.5.6.** (a) Prove that given a resolution refutation H of a set S of clauses, a resolution refutation H_g for a certain set S_g of ground instances of S can be constructed.

Apply the above construction to the following refutation:



(b) Using (a), give another proof of the soundness of the resolution method.

∗ **8.5.7.** As in the propositional case, another way of presenting the resolution method is as follows. Given a (finite) set S of clauses, let

$$R(S) = S \cup \{C \mid C \text{ is a resolvent of two clauses in } S\}.$$

Also, let

$$\begin{aligned} R^0(S) &= S, \\ R^{n+1}(S) &= R(R^n(S)), (n \geq 0), \text{ and let} \\ R^*(S) &= \bigcup_{n \geq 0} R^n(S). \end{aligned}$$

(a) Prove that S is unsatisfiable if and only if $R^*(S)$ is unsatisfiable.

(b) Prove that if S is finite, there is some $n \geq 0$ such that

$$R^*(S) = R^n(S).$$

(c) Prove that there is a resolution refutation for S if and only if the empty clause $\square$ is in $R^*(S)$.

(d) Prove that S is unsatisfiable if and only if $\square$ belongs to $R^*(S)$.

8.5.8. Prove that the resolution method is still complete if the resolution rule is restricted to clauses that are not tautologies (that is, clauses not containing both A and $\neg A$ for some atomic formula A).

∗ **8.5.9.** We say that a clause C_1 *subsumes* a clause C_2 if there is a substitution σ such that $\sigma(C_1)$ is a subset of C_2. In the version of the resolution method described in problem 8.5.7, let

$$R_1(S) = R(S) - \{C \mid C \text{ is subsumed by some clause in } R(S)\}.$$

$$\begin{aligned} \text{Let } R_1^0 &= S, \\ R_1^{n+1}(S) &= R_1(R_1^n(S)) \text{ and} \\ R_1^*(S) &= \bigcup_{n \geq 0} R_1^n(S). \end{aligned}$$

Prove that S is unsatisfiable if and only if $\square$ belongs to $R_1^*(S)$.

8.5.10. The resolution method described in problem 8.5.7 can be modified by introducing the concept of *factoring*. Given a clause C, if C' is any subset of C and C' is unifiable, the clause $\sigma(C)$ where σ is a most general unifier of C' is a *factor* of C. The *factoring rule* is the rule that allows any factor of a clause to be added to $R(S)$. Consider the simplification of the resolution rule in which a resolvent of two clauses A and B is obtained by resolving sets A' and B' consisting of a single literal. This restricted version of the resolution rule is sometimes called *binary resolution*.

(a) Show that binary resolution together with the factoring rule is complete.

(b) Show that the factoring rule can be restricted to sets C' consisting of a pair of literals.

(c) Show that binary resolution alone is not complete.

8.5.11. Prove that the resolution method is also complete for infinite sets of clauses.

8.5.12. Write a computer program implementing the resolution method.

8.6 A Glimpse at Paramodulation

As we have noted earlier, equality causes complications in automatic theorem proving. Several methods for handling equality with the resolution method have been proposed, including the *paramodulation method* (Robinson and Wos, 1969), and the *E-resolution method* (Morris, 1969; Anderson, 1970). Due to the lack of space, we will only define the *paramodulation rule*, but we will not give a full treatment of this method.

In order to define the paramodulation rule, it is convenient to assume that the *factoring rule* is added to the resolution method. Given a clause A, if A' is any subset of A and A' is unifiable, the clause $\sigma(A)$ where σ is a most general unifier of A' is a *factor* of A. Using the factoring rule, it is easy to see that the resolution rule can be simplified, so that a resolvent of two clauses A and B is obtained by resolving sets A' and B' consisting of a single literal. This restricted version of the resolution rule is sometimes called *binary resolution* (this is a poor choice of terminology since both this restricted rule and the general resolution rule take two clauses as arguments, but yet, it is used in the literature!). It can be shown that binary resolution alone is not complete, but it is easy to show that it is complete together with the factoring rule (see problem 8.5.10).

The *paramodulation rule* is a rule that treats an equation $s \doteq t$ as a (two way) *rewrite rule*, and allows the replacement of a subterm r unifiable with

s (or t) in an atomic formula Q, by the other side of the equation, modulo substitution by a most general unifier.

More precisely, let

$$A = ((s \doteq t) \vee C)$$

be a clause containing the equation $s \doteq t$, and

$$B = (Q \vee D)$$

be another clause containing some literal Q (of the form $Pt_1...t_n$ or $\neg Pt_1...t_n$, for some predicate symbol P of rank n, possibly the equality symbol $\doteq$, in which case $n = 2$), and assume that for some tree address u in Q, the subterm $r = Q/u$ is unifiable with s (or that r is unifiable with t). If σ is a most general unifier of s and r, then the clause

$$\sigma(C \vee Q[u \leftarrow t] \vee D)$$

(or $\sigma(C \vee Q[u \leftarrow s] \vee D)$, if r and t are unifiable) is a *paramodulant* of A and B. (Recall from Subsection 2.2.5, that $Q[u \leftarrow t]$ (or $Q[u \leftarrow s]$) is the result of replacing the subtree at address u in Q by t (or s)).

EXAMPLE 8.6.1

Let

$$A = \{f(x, h(y)) \doteq g(x, y), P(x)\}, \quad B = \{Q(h(f(h(x), h(a))))\}.$$

Then

$$\{Q(h(g(h(z), h(a)))), P(h(z))\}$$

is a paramodulant of A and B, in which the replacement is performed in B at address 11.

EXAMPLE 8.6.2

Let

$$A = \{f(g(x), x) \doteq h(a)\}, \quad B = \{f(x, y) \doteq h(y)\}.$$

Then,

$$\{h(z) \doteq h(a)\}$$

is a paramodulant of A and B, in which the replacement is performed in A at address e.

It can be shown that the resolution method using the (binary) resolution rule, the factoring rule, and the paramodulation rule, is complete for any finite set S of clauses, provided that the reflexity axiom and the functional reflexivity axioms are added to S. The *reflexivity axiom* is the clause

$$\{x \doteq x\},$$

and the *functional reflexivity axioms* are the clauses

$$\{f(x_1, ..., x_n) \doteq f(x_1, ..., x_n)\},$$

for each function symbol f occurring in S, of any rank $n > 0$.

The proof that this method is complete is more involved than the proof for the case of a first-language without equality, partly because the lifting lemma does not extend directly. It can also be shown that paramodulation is complete without the functional reflexivity axioms, but this is much harder. For details, the reader is referred to Loveland, 1978.

Notes and Suggestions for Further Reading

The resolution method has been studied extensively, and there are many refinements of this method. Some of the refinements are still complete for all clauses (linear resolution, model elimination), others are more efficient but only complete for special kinds of clauses (unit or input resolution). For a detailed exposition of these methods, the reader is referred to Loveland, 1978; Robinson, 1979, and to the collection of original papers compiled in Siekmann and Wrightson, 1983. One should also consult Boyer and Moore, 1979, for advanced techniques in automatic theorem proving, induction in particular. For a more introductory approach, the reader may consult Bundy, 1983, and Kowalski, 1979.

The resolution method has also been extended to higher-order logic by Andrews. The interested reader should consult Andrews, 1971; Pietrzykowski, 1973; and Huet, 1973.