Chapter 10
First-Order Logic: Resolution

Resolution is a sound and complete algorithm for propositional logic: a formula in clausal form is unsatisfiable if and only if the algorithm reports that it is unsatisfiable. For propositional logic, the algorithm is also a decision procedure for unsatisfiability because it is guaranteed to terminate. When generalized to first-order logic, resolution is still sound and complete, but it is not a decision procedure because the algorithm may not terminate.

The generalization of resolution to first-order logic will be done in two stages. First, we present *ground resolution* which works on ground literals as if they were propositional literals; then we present the *general resolution* procedure, which uses a highly efficient matching algorithm called *unification* to enable resolution on non-ground literals.

10.1 Ground Resolution

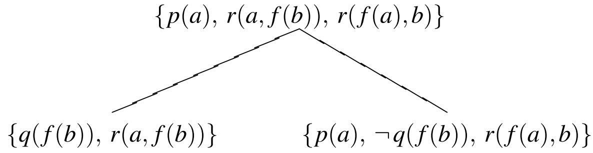
Rule 10.1 (Ground resolution rule) *Let* C_1, C_2 *be* ground *clauses such that* $l \in C_1$ *and* $l^c \in C_2$. C_1, C_2 *are said to be* clashing clauses *and to* clash *on the complementary literals* l, l^c. C, *the* resolvent *of* C_1 *and* C_2, *is the clause*:

$$Res(C_1, C_2) = (C_1 - \{l\}) \cup (C_2 - \{l^c\}).$$

C_1 *and* C_2 *are the* parent clauses *of* C. ■

M. Ben-Ari, *Mathematical Logic for Computer Science*,
DOI 10.1007/978-1-4471-4129-7_10,

Example 10.2 Here is a tree representation of the ground resolution of two clauses. They clash on the literal $q(f(b))$:



■

Theorem 10.3 *The resolvent C is satisfiable if and only if the parent clauses C_1 and C_2 are both satisfiable.*

Proof Let C_1 and C_2 be satisfiable clauses which clash on the literals l, l^c. By Theorem 9.24, they are satisfiable in an Herbrand interpretation $\mathscr{H}$. Let B be the subset of the Herbrand base that defines $\mathscr{H}$, that is,

$$B = \{p(c_1,\dots,c_k) \mid v_H(p(c_1,\dots,c_k)) = T\}$$

for ground terms c_i. Obviously, two complementary ground literals cannot both be elements of B. Suppose that $l \in B$. For C_2 to be satisfied in $\mathscr{H}$ there must be some *other* literal $l' \in C_2$ such that $l' \in B$. By construction of the resolvent C using the resolution rule, $l' \in C$, so $v_{\mathscr{H}}(C) = T$, that is, $\mathscr{H}$ is a model for C. A symmetric argument holds if $l^c \in B$.

Conversely, if C is satisfiable, it is satisfiable in an Herbrand interpretation $\mathscr{H}$ defined by a subset B of the Herbrand base. For some literal $l' \in C$, $l' \in B$. By the construction of the resolvent clause in the rule, $l' \in C_1$ or $l' \in C_2$ (or both). Suppose that $l' \in C_1$. We can extend the $\mathscr{H}$ to $\mathscr{H}'$ by defining $B' = B \cup \{l^c\}$. Again, by construction, $l \notin C$ and $l^c \notin C$, so $l \notin B$ and $l^c \notin B$ and therefore B' is well defined.

We need to show that C_1 and C_2 are both satisfied by $\mathscr{H}'$ defined by the Herbrand base B'. Clearly, since $l' \in C$, $l' \in B \subseteq B'$, so C_1 is satisfied in $\mathscr{H}'$. By definition, $l^c \in B'$, so C_2 is satisfied in $\mathscr{H}'$.

A symmetric argument holds if $l' \in C_2$. ■

The ground resolution procedure is defined like the resolution procedure for propositional logic. Given a set of ground clauses, the resolution step is performed repeatedly. The set of ground clauses is unsatisfiable iff some sequence of resolution steps produces the empty clause. We leave it as an exercise to show that ground resolution is a sound and complete refutation procedure for first-order logic.

Ground resolution is not a useful refutation procedure for first-order logic because the set of ground terms is infinite (assuming that there is even one function symbols). Robinson (1965) showed that how to perform resolution on clauses that are not ground by looking for substitutions that create clashing clauses. The definitions and algorithms are rather technical and are described in detail in the next two sections.

10.2 Substitution

We have been somewhat informal about the concept of substituting a term for a variable. In this section, the concept is formally defined.

Definition 10.4 A *substitution* of terms for variables is a set:

$$\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\},$$

where each x_i is a distinct variable and each t_i is a term which is not identical to the corresponding variable x_i. The *empty substitution* is the empty set. ■

Lower-case Greek letters $\{\lambda, \mu, \sigma, \theta\}$ will be used to denote substitutions. The empty substitution is denoted ε.

Definition 10.5 An *expression* is a term, a literal, a clause or a set of clauses. Let E be an expression and let $\theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ be a substitution. An *instance* $E\theta$ of E is obtained by *simultaneously* replacing each occurrence of x_i in E by t_i. ■

Example 10.6 Here is an expression (clause) $E = \{p(x), q(f(y))\}$ and a substitution $\theta = \{x \leftarrow y, y \leftarrow f(a)\}$, the instance obtained by performing the substitution is:

$$E\theta = \{p(y), q(f(f(a)))\}.$$

The word *simultaneously* in Definition 10.5 means that one does *not* substitute y for x in E to obtain:

$$\{p(y), q(f(y))\},$$

and *then* substitute $f(a)$ for y to obtain:

$$\{p(f(a)), q(f(f(a)))\}.$$

■

The result of a substitution need not be a ground expression; at the extreme, a substitution can simply rename variables: $\{x \leftarrow y, z \leftarrow w\}$. Therefore, it makes sense to apply a substitution to an instance, because the instance may still have variables. The following definition shows how substitutions can be composed.

Definition 10.7 Let:

$$\theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\},$$
$$\sigma = \{y_1 \leftarrow s_1, \ldots, y_k \leftarrow s_k\}$$

be two substitutions and let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_k\}$ be the sets of variables substituted for in θ and σ, respectively. $\theta\sigma$, the *composition of* θ *and* σ, is the substitution:

$$\theta\sigma = \{x_i \leftarrow t_i\sigma \mid x_i \in X, x_i \neq t_i\sigma\} \cup \{y_j \leftarrow s_j \mid y_j \in Y, y_j \notin X\}.$$

In words: apply the substitution σ to the terms t_i of θ (provided that the resulting substitutions do not collapse to $x_i \leftarrow x_i$) and then append the substitutions from σ whose variables do not already appear in θ. ■

Example 10.8 Let:

$$\begin{aligned} E &= p(u, v, x, y, z), \\ \theta &= \{x \leftarrow f(y),\ y \leftarrow f(a),\ z \leftarrow u\}, \\ \sigma &= \{y \leftarrow g(a),\ u \leftarrow z,\ v \leftarrow f(f(a))\}. \end{aligned}$$

Then:

$$\theta\sigma = \{x \leftarrow f(g(a)),\ y \leftarrow f(a),\ u \leftarrow z,\ v \leftarrow f(f(a))\}.$$

The vacuous substitution $z \leftarrow z = (z \leftarrow u)\sigma$ has been deleted. The substitution $y \leftarrow g(a) \in \sigma$ has also been deleted since y already appears in θ. Once the substitution $y \leftarrow f(a)$ is performed, no occurrences of y remain in the expression. The instance obtained from the composition is:

$$E(\theta\sigma) = p(z, f(f(a)), f(g(a)), f(a), z).$$

Alternatively, we could have performed the substitution in two stages:

$$\begin{aligned} E\theta &= p(u, v, f(y), f(a), u), \\ (E\theta)\sigma &= p(z, f(f(a)), f(g(a)), f(a), z). \end{aligned}$$

We see that $E(\theta\sigma) = (E\theta)\sigma$. ■

The result of performing two substitutions one after the other is the same as the result of computing the composition followed by a single substitution.

Lemma 10.9 *For any expression E and substitutions θ, σ, $E(\theta\sigma) = (E\theta)\sigma$.*

Proof Let E be a variable z. If z is not substituted for in θ or σ, the result is trivial. If $z = x_i$ for some $\{x_i \leftarrow t_i\}$ in θ, then $(z\theta)\sigma = t_i\sigma = z(\theta\sigma)$ by the definition of composition. If $z = y_j$ for some $\{y_j \leftarrow s_j\}$ in σ and $z \neq x_i$ for all i, then $(z\theta)\sigma = z\sigma = s_j = z(\theta\sigma)$.

The result follows by induction on the structure of E. ■

We leave it as an exercise to show that composition is associative.

Lemma 10.10 *For any substitutions θ, σ, λ, $\theta(\sigma\lambda) = (\theta\sigma)\lambda$.*

10.3 Unification

The two literals $p(f(x),\ g(y))$ and $\neg\, p(f(f(a)),\ g(z))$ do not clash. However, under the substitution:

$$\theta_1 = \{x \leftarrow f(a),\ y \leftarrow f(g(a)),\ z \leftarrow f(g(a))\},$$

they become clashing (ground) literals:

$$p(f(f(a)),\ g(f(g(a)))), \qquad \neg\, p(f(f(a)),\ g(f(g(a)))).$$

The following simpler substitution:

$$\theta_2 = \{x \leftarrow f(a),\ y \leftarrow a,\ z \leftarrow a\}$$

also makes these literals clash:

$$p(f(f(a)),\ g(a)), \qquad \neg\, p(f(f(a)),\ g(a)).$$

Consider now the substitution:

$$\mu = \{x \leftarrow f(a),\ z \leftarrow y\}.$$

The literals that result are:

$$p(f(f(a)),\ g(y)), \qquad \neg\, p(f(f(a)),\ g(y)).$$

Any further substitution of a ground term for y will produce clashing ground literals.

The general resolution algorithm allows resolution on clashing literals that contain variables. By finding the simplest substitution that makes two literals clash, the resolvent is the most general result of a resolution step and is more likely to clash with another clause after a suitable substitution.

Definition 10.11 Let $U = \{A_1, \ldots, A_n\}$ be a set of atoms. A *unifier* θ is a substitution such that:

$$A_1\theta = \cdots = A_n\theta.$$

A *most general unifier (mgu)* for U is a unifier μ such that any unifier θ of U can be expressed as:

$$\theta = \mu\lambda$$

for some substitution λ. ■

Example 10.12 The substitutions θ_1, θ_2, μ, above, are unifiers of the set of two atoms $\{p(f(x), g(y)),\ p(f(f(a)), g(z))\}$. The substitution μ is an mgu. The first two substitutions can be expressed as:

$$\theta_1 = \mu\{y \leftarrow f(g(a))\}, \qquad \theta_2 = \mu\{y \leftarrow a\}.$$

■

Not all atoms are unifiable. It is clearly impossible to unify atoms whose predicate symbols are different such as $p(x)$ and $q(x)$, as well as atoms with terms whose outer function symbols are different such as $p(f(x))$ and $p(g(y))$. A more tricky case is shown by the atoms $p(x)$ and $p(f(x))$. Since x *occurs* within the larger term $f(x)$, any substitution—which must substitute simultaneously in both atoms—cannot unify them. It turns out that as long as these conditions do not hold the atoms will be unifiable.

We now describe and prove the correctness of an algorithm for unification by Martelli and Montanari (1982). Robinson's original algorithm is presented briefly in Sect. 10.3.4.

10.3.1 The Unification Algorithm

Trivially, two atoms are unifiable only if they have the same predicate letter of the same arity. Thus the unifiability of atoms is more conveniently described in terms of the unifiability of the arguments, that is, the *unifiability of a set of terms*. The set of terms to be unified will be written as a set of term equations.

Example 10.13 The unifiability of $\{p(f(x), g(y)),\ p(f(f(a)), g(z))\}$ is expressed by the set of term equations:

$$\begin{aligned} f(x) &= f(f(a)), \\ g(y) &= g(z). \end{aligned}$$

■

Definition 10.14 A set of term equations is in *solved form* iff:

- All equations are of the form $x_i = t_i$ where x_i is a variable.
- Each variable x_i that appears on the left-hand side of an equation does not appear elsewhere in the set.

A set of equations in solved form defines a substitution:

$$\{x_1 \leftarrow t_1,\ \ldots,\ x_n \leftarrow t_n\}.$$

■

The following algorithm transforms a set of term equations into a set of equations in solved form, or reports if it is impossible to do so. In Sect. 10.3.3, we show that the substitution defined by the set in solved form is a most general unifier of the original set of term equations, and hence of the set of atoms from which the terms were taken.

Algorithm 10.15 (Unification algorithm)
Input: A set of term equations.
Output: A set of term equations in solved form or report *not unifiable*.

Perform the following transformations on the set of equations as long as any one of them is applicable:

1. Transform $t = x$, where t is not a variable, to $x = t$.
2. Erase the equation $x = x$.
3. Let $t' = t''$ be an equation where t', t'' are not variables.
   - If the outermost function symbols of t' and t'' are not identical, terminate the algorithm and report *not unifiable*.
   - Otherwise, replace the equation $f(t'_1, \ldots, t'_k) = f(t''_1, \ldots, t''_k)$ by the k equations $t'_1 = t''_1, \ldots, t'_k = t''_k$.
4. Let $x = t$ be an equation such that x has another occurrence in the set.
   - If x occurs in t and x differs from t, terminate the algorithm and report *not unifiable*.
   - Otherwise, transform the set by replacing all occurrences of x in other equations by t. ■

Example 10.16 Consider the following set of two equations:

$$\begin{aligned} g(y) &= x, \\ f(x, h(x), y) &= f(g(z), w, z). \end{aligned}$$

Apply rule 1 to the first equation and rule 3 to the second equation:

$$\begin{aligned} x &= g(y), \\ x &= g(z), \\ h(x) &= w, \\ y &= z. \end{aligned}$$

Apply rule 4 to the second equation by replacing occurrences of x in other equations by $g(z)$:

$$\begin{aligned} g(z) &= g(y), \\ x &= g(z), \\ h(g(z)) &= w, \\ y &= z. \end{aligned}$$

Apply rule 3 to the first equation:

$$\begin{aligned} z &= y, \\ x &= g(z), \\ h(g(z)) &= w, \\ y &= z. \end{aligned}$$

Apply rule 4 to the last equation by replacing y by z in the first equation; next, erase the result $z = z$ using rule 2:

$$\begin{aligned} x &= g(z), \\ h(g(z)) &= w, \\ y &= z. \end{aligned}$$

Finally, transform the second equation by rule 1:

$$\begin{aligned} x &= g(z), \\ w &= h(g(z)), \\ y &= z. \end{aligned}$$

This successfully terminates the algorithm. We claim that:

$$\mu = \{x \leftarrow g(z),\ w \leftarrow h(g(z)),\ y \leftarrow z\}$$

is a most general unifier of the original set of equations. We leave it to the reader to check that the substitution does in fact unify the original set of term equations and further to check that the unifier:

$$\theta = \{x \leftarrow g(f(a)),\ w \leftarrow h(g(f(a))),\ y \leftarrow f(a),\ z \leftarrow f(a)\}$$

can be expressed as $\theta = \mu\{z \leftarrow f(a)\}$. ■

10.3.2 The Occurs-Check

Algorithms for unification can be extremely inefficient because of the need to check the condition in rule 4, called the *occurs-check*.

Example 10.17 To unify the set of equations:

$$\begin{aligned} x_1 &= f(x_0, x_0), \\ x_2 &= f(x_1, x_1), \\ x_3 &= f(x_2, x_2), \\ &\cdots \end{aligned}$$

we successively create the equations:

$$\begin{aligned} x_2 &= f(f(x_0, x_0), f(x_0, x_0)), \\ x_3 &= f(f(f(x_0, x_0), f(x_0, x_0)), f(f(x_0, x_0), f(x_0, x_0))), \\ &\cdots \end{aligned}$$

The equation for x_i contains 2^i variables. ■

In the application of unification to logic programming (Chap. 11), the occurs-check is simply ignored and the risk of an illegal substitution is taken.

*10.3.3 The Correctness of the Unification Algorithm **

Theorem 10.18

- *Algorithm 10.15 terminates with the set of equations in solved form or it reports* not unifiable.
- *If the algorithm reports* not unifiable, *there is no unifier for the set of term equations.*
- *If the algorithm terminates successfully, the resulting set of equations is in solved form and defines the mgu*:

$$\mu = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}.$$

Proof Obviously, rules 1–3 can be used only finitely many times without using rule 4. Let m be the number of distinct variables in the set of equations. Rule 4 can be used at most m times since it removes all occurrences, except one, of a variable and can never be used twice on the same variable. Thus the algorithm terminates.

The algorithm terminates with failure in rule 3 if the function symbols are distinct, and in rule 4 if a variable occurs within a term in the same equation. In both cases there can be no unifier.

It is easy to see that if it terminates successfully, the set of equations is in solved form. It remains to show that μ is a most general unifier.

Define a transformation as an *equivalence transformation* if it preserves sets of unifiers of the equations. Obviously, rules 1 and 2 are equivalence transformations. Consider now an application of rule 3 for $t' = f(t'_1, \ldots, t'_k)$ and $t'' = f(t''_1, \ldots, t''_k)$. If $t'\sigma = t''\sigma$, by the inductive definition of a term this can only be true if $t'_i\sigma = t''_i\sigma$ for all i. Conversely, if some unifier σ makes $t'_i = t''_i$ for all i, then σ is a unifier for $t' = t''$. Thus rule 3 is an equivalence transformation.

Suppose now that $t_1 = t_2$ was transformed into $u_1 = u_2$ by rule 4 on $x = t$. After applying the rule, $x = t$ remains in the set. So any unifier σ for the set must make $x\sigma = t\sigma$. Then, for $i = 1, 2$:

$$u_i\sigma = (t_i\{x \leftarrow t\})\sigma = t_i(\{x \leftarrow t\}\sigma) = t_i\sigma$$

by the associativity of substitution and by the definition of composition of substitution using the fact that $x\sigma = t\sigma$. So if σ is a unifier of $t_1 = t_2$, then $u_1\sigma = t_1\sigma = t_2\sigma = u_2\sigma$ and σ is a unifier of $u_1 = u_2$; it follows that rule 4 is an equivalence transformation.

Finally, the substitution defined by the set is an mgu. We have just proved that the original set of equations and the solved set of equations have the *same* set of unifiers. But the solved set itself defines a substitution (replacements of terms for variables) which is a unifier. Since the transformations were equivalence transformations, no equation can be removed from the set without destroying the property that it is a unifier. Thus any unifier for the set can only substitute more complicated terms for the same variables or substitute for other variables. That is, if μ is:

$$\mu = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\},$$

any other unifier σ can be written:

$$\sigma = \{x_1 \leftarrow t_1', \ldots, x_n \leftarrow t_n'\} \cup \{y_1 \leftarrow s_1, \ldots, y_m \leftarrow s_m\},$$

which is $\sigma = \mu\lambda$ for some substitution λ by definition of composition. Therefore, μ is an mgu. ■

The algorithm is nondeterministic because we may choose to apply a rule to any equation to which it is applicable. A deterministic algorithm can be obtained by specifying the order in which to apply the rules. One such deterministic algorithm is obtained by considering the set of equations as a queue. A rule is applied to the first element of the queue and then that equation goes to the end of the queue. If new equations are created by rule 3, they are added to the beginning of the queue.

Example 10.19 Here is Example 10.16 expressed as a queue of equations:

$$\begin{array}{lllll}
\langle\, g(y) = x, & f(x, h(x), y) = f(g(z), w, z) & & & \rangle \\
\langle\, f(g(y), h(g(y)), y) = f(g(z), w, z), & x = g(y) & & & \rangle \\
\langle\, g(y) = g(z), & h(g(y)) = w, & y = z, & x = g(y) & \rangle \\
\langle\, y = z, & h(g(y)) = w, & y = z, & x = g(y) & \rangle \\
\langle\, h(g(z)) = w, & z = z, & x = g(z), & y = z & \rangle \\
\langle\, z = z, & x = g(z), & y = z, & w = h(g(z)) & \rangle \\
\langle\, x = g(z), & y = z, & w = h(g(z)) & & \rangle
\end{array}$$

■

10.3.4 Robinson's Unification Algorithm *

Robinson's algorithm appears in most other works on resolution so we present it here without proof (see Lloyd (1987, Sect. 1.4) for a proof).

Definition 10.20 Let A and A' be two atoms with the same predicate symbols. Considering them as sequences of symbols, let k be the leftmost position at which the sequences are different. The pair of terms $\{t, t'\}$ beginning at position k in A and A' is the *disagreement set* of the two atoms. ■

Algorithm 10.21 (Robinson's unification algorithm)
Input: Two atoms A and A' with the same predicate symbol.
Output: A most general unifier for A and A' or report *not unifiable*.

Initialize the algorithm by letting $A_0 = A$ and $A'_0 = A'$. Perform the following step repeatedly:

- Let $\{t, t'\}$ be the disagreement set of A_i, A'_i. If one term is a variable x_{i+1} and the other is a term t_{i+1} such that x_{i+1} does not occur in t_{i+1}, let $\sigma_{i+1} = \{x_{i+1} \leftarrow t_{i+1}\}$ and $A_{i+1} = A_i\sigma_{i+1}$, $A'_{i+1} = A'_i\sigma_{i+1}$.

If it is impossible to perform the step (because both elements of the disagreement set are compound terms or because the occurs-check fails), the atoms are not unifiable. If after some step $A_n = A'_n$, then A, A' are unifiable and the mgu is $\mu = \sigma_i \cdots \sigma_n$. ■

Example 10.22 Consider the pair of atoms:

$$A = p(g(y),\ f(x, h(x), y)), \qquad A' = p(x,\ f(g(z), w, z)).$$

The initial disagreement set is $\{x,\ g(y)\}$. One term is a variable which does not occur in the other so $\sigma_1 = \{x \leftarrow g(y)\}$, and:

$$\begin{aligned} A\sigma_1 &= p(g(y),\ f(g(y), h(g(y)), y)), \\ A'\sigma_1 &= p(g(y),\ f(g(z), w, z)). \end{aligned}$$

The next disagreement set is $\{y,\ z\}$ so $\sigma_2 = \{y \leftarrow z\}$, and:

$$\begin{aligned} A\sigma_1\sigma_2 &= p(g(z),\ f(g(z), h(g(z)), z)), \\ A'\sigma_1\sigma_2 &= p(g(z),\ f(g(z), w, z)). \end{aligned}$$

The third disagreement set is $\{w,\ h(g(z))\}$ so $\sigma_3 = \{w \leftarrow h(g(z))\}$, and:

$$\begin{aligned} A\sigma_1\sigma_2\sigma_3 &= p(g(z),\ f(g(z), h(g(z)), z)), \\ A'\sigma_1\sigma_2\sigma_3 &= p(g(z),\ f(g(z), h(g(z)), z)). \end{aligned}$$

Since $A\sigma_1\sigma_2\sigma_3 = A'\sigma_1\sigma_2\sigma_3$, the atoms are unifiable and the mgu is:

$$\mu = \sigma_1\sigma_2\sigma_3 = \{x \leftarrow g(z),\ y \leftarrow z,\ w \leftarrow h(g(z))\}.$$

■

10.4 General Resolution

The resolution rule can be applied directly to non-ground clauses by performing unification as an integral part of the rule.

Definition 10.23 Let $L = \{l_1, \ldots, l_n\}$ be a set of literals. Then $L^c = \{l_1^c, \ldots, l_n^c\}$. ■

Rule 10.24 (General resolution rule) *Let* C_1, C_2 *be clauses* with no variables in common. *Let* $L_1 = \{l^1_1, \ldots, l^1_{n_1}\} \subseteq C_1$ *and* $L_2 = \{l^2_1, \ldots, l^2_{n_2}\} \subseteq C_2$ *be subsets of literals such that* L_1 *and* L^c_2 *can be unified by an mgu* σ. C_1 *and* C_2 *are said to be* clashing clauses *and to* clash *on the sets of literals* L_1 *and* L_2. C, *the* resolvent *of* C_1 *and* C_2, *is the clause*:

$$Res(C_1, C_2) = (C_1\sigma - L_1\sigma) \cup (C_2\sigma - L_2\sigma).$$

■

Example 10.25 Given the two clauses:

$$\{p(f(x), g(y)),\ q(x, y)\}, \qquad \{\neg\, p(f(f(a)), g(z)),\ q(f(a), z)\},$$

an mgu for $L_1 = \{p(f(x), g(y))\}$ and $L^c_2 = \{p(f(f(a)), g(z))\}$ is:

$$\{x \leftarrow f(a), y \leftarrow z\}.$$

The clauses resolve to give:

$$\{q(f(a), z),\ q(f(a), z)\} \quad = \quad \{q(f(a), z)\}.$$

■

Clauses are *sets* of literals, so when taking the union of the clauses in the resolution rule, identical literals will be collapsed; this is called *factoring*.

The general resolution rule requires that the clauses have no variables in common. This is done by *standardizing apart*: renaming all the variables in one of the clauses before it is used in the resolution rule. All variables in a clause are implicitly universally quantified so renaming does not change satisfiability.

Example 10.26 To resolve the two clauses $p(f(x))$ and $\neg\, p(x)$, first rename the variable x of the second clause to x': $\neg\, p(x')$. An mgu is $\{x' \leftarrow f(x)\}$, and $p(f(x))$ and $\neg\, p(f(x))$ resolve to $\Box$.

The clauses represent the formulas $\forall x p(f(x))$ and $\forall x \neg\, p(x)$, and it is obvious that their conjunction $\forall x p(f(x)) \wedge \forall x \neg\, p(x)$ is unsatisfiable. ■

Example 10.27 Let $C_1 = \{p(x), p(y)\}$ and $C_2 = \{\neg\, p(x), \neg\, p(y)\}$. Standardize apart so that $C'_2 = \{\neg\, p(x'), \neg\, p(y')\}$. Let $L_1 = \{p(x), p(y)\}$ and let $L^c_2 = \{p(x'), p(y')\}$; these sets have an mgu:

$$\sigma = \{y \leftarrow x, x' \leftarrow x, y' \leftarrow x\}.$$

The resolution rule gives:

$$\begin{aligned} Res(C_1, C_2) &= (C_1\sigma - L_1\sigma) \cup (C'_2\sigma - L_2\sigma) \\ &= (\{p(x)\} - \{p(x)\}) \cup (\{\neg\, p(x)\} - \{\neg\, p(x)\}) \\ &= \Box. \end{aligned}$$

■

In this example, the empty clause cannot be obtained without factoring, but we will talk about clashing literals rather than clashing sets of literals when no confusion will result.

Algorithm 10.28 (General Resolution Procedure)
Input: A set of clauses S.
Output: If the algorithm terminates, report that the set of clauses is *satisfiable* or *unsatisfiable*.

Let $S_0 = S$. Assume that S_i has been constructed. *Choose* clashing clauses $C_1, C_2 \in S_i$ and let $C = Res(C_1, C_2)$. If $C = \square$, terminate and report that S is *unsatisfiable*. Otherwise, construct $S_{i+1} = S_i \cup \{C\}$. If $S_{i+1} = S_i$ for all possible pairs of clashing clauses, terminate and report S is *satisfiable*. ■

While an unsatisfiable set of clauses will eventually produce $\square$ under a suitable systematic execution of the procedure, the existence of infinite models means that the resolution procedure on a satisfiable set of clauses may never terminate, so general resolution is not a decision procedure.

Example 10.29 Lines 1–7 contain a set of clauses. The resolution refutation in lines 8–15 shows that the set of clauses is unsatisfiable. Each line contains the resolvent, the mgu and the numbers of the parent clauses.

1.	$\{\neg p(x),\ q(x),\ r(x, f(x))\}$		
2.	$\{\neg p(x),\ q(x),\ r'(f(x))\}$		
3.	$\{p'(a)\}$		
4.	$\{p(a)\}$		
5.	$\{\neg r(a, y),\ p'(y)\}$		
6.	$\{\neg p'(x),\ \neg q(x)\}$		
7.	$\{\neg p'(x),\ \neg r'(x)\}$		
8.	$\{\neg q(a)\}$	$x \leftarrow a$	3, 6
9.	$\{q(a),\ r'(f(a))\}$	$x \leftarrow a$	2, 4
10.	$\{r'(f(a))\}$		8, 9
11.	$\{q(a),\ r(a, f(a))\}$	$x \leftarrow a$	1, 4
12.	$\{r(a, f(a))\}$		8, 11
13.	$\{p'(f(a))\}$	$y \leftarrow f(a)$	5, 12
14.	$\{\neg r'(f(a))\}$	$x \leftarrow f(a)$	7, 13
15.	$\{\square\}$		10, 14

■

Example 10.30 Here is another example of a resolution refutation showing variable renaming and mgu's which do not produce ground clauses. The first four clauses form the set of clauses to be refuted.

1.	$\{\neg p(x,y),\ p(y,x)\}$		
2.	$\{\neg p(x,y),\ \neg p(y,z),\ p(x,z)\}$		
3.	$\{p(x,f(x))\}$		
4.	$\{\neg p(x,x)\}$		
3′.	$\{p(x',f(x'))\}$		Rename 3
5.	$\{p(f(x),x)\}$	$\sigma_1 = \{y \leftarrow f(x), x' \leftarrow x\}$	$1, 3'$
3″.	$\{p(x'',f(x''))\}$		Rename 3
6.	$\{\neg p(f(x),z),\ p(x,z)\}$	$\sigma_2 = \{y \leftarrow f(x), x'' \leftarrow x\}$	$2, 3''$
5‴.	$\{p(f(x'''),x''')\}$		Rename 5
7.	$\{p(x,x)\}$	$\sigma_3 = \{z \leftarrow x, x''' \leftarrow x\}$	$6, 5'''$
4⁗.	$\{\neg p(x'''',x'''')\}$		Rename 4
8.	$\{\Box\}$	$\sigma_4 = \{x'''' \leftarrow x\}$	$7, 4''''$

If we concatenate the substitutions, we get:

$$\sigma = \sigma_1\sigma_2\sigma_3\sigma_4 = \{y \leftarrow f(x), z \leftarrow x, x' \leftarrow x, x'' \leftarrow x, x''' \leftarrow x, x'''' \leftarrow x\}.$$

Restricted to the variables of the original clauses, $\sigma = \{y \leftarrow f(x), z \leftarrow x\}$. ■

10.5 Soundness and Completeness of General Resolution *

10.5.1 Proof of Soundness

We now show the soundness and completeness of resolution. The reader should review the proofs in Sect. 4.4 for propositional logic as we will just give the modifications that must be made to those proofs.

Theorem 10.31 (Soundness of resolution) *Let S be a set of clauses. If the empty clause $\Box$ is derived when the resolution procedure is applied to S, then S is unsatisfiable.*

Proof We need to show that if the parent clauses are (simultaneously) satisfiable, so is the resolvent; since $\Box$ is unsatisfiable, this implies that S must also be unsatisfiable. If parent clauses are satisfiable, there is an Herbrand interpretation $\mathscr{H}$ such that $v_{\mathscr{H}}(C_i) = T$ for $i = 1, 2$. The elements of the Herbrand base that satisfy C_1 and C_2 have the same form as ground atoms, so there must be a substitutions λ_i such that $C_i' = C_i\lambda_i$ are ground clauses and $v_{\mathscr{H}}(C_i') = T$.

Let C be the resolvent of C_1 and C_2. Then there is an mgu μ for C_1 and C_2 that was used to resolve the clauses. By definition of an mgu, there must substitutions θ_i such that $\lambda_i = \sigma\theta_i$. Then $C_i' = C_i\lambda_i = C_i(\sigma\theta_i) = (C_i\sigma)\theta_i$, which shows that $C_i\sigma$ is satisfiable in the same interpretation.

Let $l_1 \in C_1$ and $l_2^c \in C_2$ be the clashing literals used to derive C. Exactly one of $l_1\sigma, l_2^c\sigma$ is satisfiable in $\mathscr{H}$. Without loss of generality, suppose that $v_{\mathscr{H}}(l_1\sigma) = T$. Since $C_2\sigma$ is satisfiable, there must be a literal $l' \in C_2$ such that $l' \neq l_2^c$ and $v_{\mathscr{H}}(l'\sigma) = T$. But by the construction of the resolvent, $l' \in C$ so $v_{\mathscr{H}}(C) = T$. ■

10.5.2 *Proof of Completeness*

Using Herbrand's theorem and semantic trees, we can prove that there is a *ground resolution refutation* of an unsatisfiable set of clauses. However, this does not generalize into a proof for general resolution because the concept of semantic trees does not generalize since the variables give rise to a potentially infinite number of elements of the Herbrand base. The difficulty is overcome by taking a ground resolution refutation and lifting it into a more abstract general refutation.

The problem is that several literals in C_1 or C_2 might collapse into one literal under the substitutions that produce the ground instances C_1' and C_2' to be resolved.

Example 10.32 Consider the clauses:

$$\begin{aligned} C_1 &= \{p(x),\ p(f(y)),\ p(f(z)),\ q(x)\}, \\ C_2 &= \{\neg\, p(f(u)),\ \neg\, p(w),\ r(u)\} \end{aligned}$$

and the substitution:

$$\{x \leftarrow f(a),\ y \leftarrow a,\ z \leftarrow a,\ u \leftarrow a,\ w \leftarrow f(a)\}.$$

The substitution results in the ground clauses:

$$C_1' = \{p(f(a)),\ q(f(a))\}, \qquad C_2' = \{\neg\, p(f(a)),\ r(a)\},$$

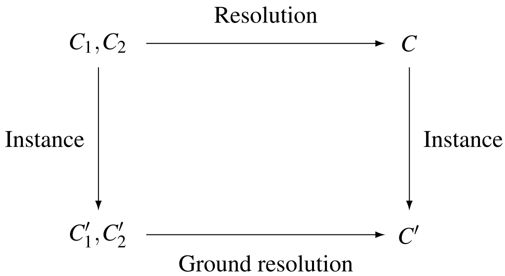
which resolve to: $C' = \{q(f(a)),\ r(a)\}$. The lifting lemma claims that there is a clause $C = \{q(f(u)),\ r(u)\}$ which is the resolvent of C_1 and C_2, such that C' is a ground instance of C. This can be seen by using the unification algorithm to obtain an mgu:

$$\{x \leftarrow f(u),\ y \leftarrow u,\ z \leftarrow u,\ w \leftarrow f(u)\}$$

of C_1 and C_2, which then resolve giving C. ■

Theorem 10.33 (Lifting Lemma) *Let C_1', C_2' be ground instances of C_1, C_2, respectively. Let C' be a ground resolvent of C_1' and C_2'. Then there is a resolvent C of C_1 and C_2 such that C' is a ground instance of C.*

The relationships among the clauses are displayed in the following diagram.



Proof The steps of the proof for Example 10.32 are shown in Fig. 10.1.

First, standardize apart so that the names of the variables in C_1 are different from those in C_2.

Let $l \in C'_1$, $l^c \in C'_2$ be the clashing literals in the ground resolution. Since C'_1 is an instance of C_1 and $l \in C'_1$, there must be a set of literals $L_1 \subseteq C_1$ such that l is an instance of each literal in L_1. Similarly, there must a set $L_2 \subseteq C_2$ such that l^c is an instance of each literal in L_2. Let λ_1 and λ_2 mgu's for L_1 and L_2, respectively, and let $\lambda = \lambda_1 \cup \lambda_2$. λ is a well-formed substitution since L_1 and L_2 have no variables in common.

By construction, $L_1\lambda$ and $L_2\lambda$ are sets which contain a single literal each. These literals have clashing ground instances, so they have a mgu σ. Since $L_i \subseteq C_i$, we have $L_i\lambda \subseteq C_i\lambda$. Therefore, $C_1\lambda$ and $C_2\lambda$ are clauses that can be made to clash under the mgu σ. It follows that they can be resolved to obtain clause C:

$$C = ((C_1\lambda)\sigma - (L_1\lambda)\sigma) \cup ((C_2\lambda)\sigma - (L_2\lambda)\sigma).$$

By the associativity of substitution (Theorem 10.10):

$$C = (C_1(\lambda\sigma) - L_1(\lambda\sigma)) \cup (C_2(\lambda\sigma) - (L_2(\lambda\sigma)).$$

C is a resolvent of C_1 and C_2 provided that $\lambda\sigma$ is an mgu of L_1 and L_2^c. But λ is already reduced to equations of the form $x \leftarrow t$ for distinct variables x and σ is constructed to be an mgu, so $\lambda\sigma$ is a reduced set of equations, all of which are necessary to unify L_1 and L_2^c. Hence $\lambda\sigma$ is an mgu.

Since C'_1 and C'_2 are ground instances of C_1 and C_2:

$$C'_1 = C_1\theta_1 = C_1\lambda\sigma\theta'_1 \qquad C'_2 = C_2\theta_2 = C_2\lambda\sigma\theta'_2$$

for some substitutions $\theta_1, \theta_2, \theta'_1, \theta'_2$. Let $\theta' = \theta'_1 \cup \theta'_2$. Then $C' = C\theta'$ and C' is a ground instance of C. ■

Theorem 10.34 (Completeness of resolution) *If a set of clauses is unsatisfiable, the empty clause* $\square$ *can be derived by the resolution procedure.*

Proof The proof is by induction on the semantic tree for the set of clauses S. The definition of semantic tree is modified as follows:

$$
\begin{array}{lcl}
C_1 & = & \{p(x),\ p(f(y)),\ p(f(z)),\ q(x)\} \\
C_2 & = & \{\neg\, p(f(u)),\ \neg\, p(w),\ r(u)\} \\
\\
\theta_1 & = & \{x \leftarrow f(a),\ y \leftarrow a,\ z \leftarrow a\} \\
\theta_2 & = & \{u \leftarrow a,\ w \leftarrow f(a)\} \\
\\
C_1' & = & C_1\theta_1 = \{p(f(a)),\ q(f(a))\} \\
C_2' & = & C_2\theta_2 = \{\neg\, p(f(a)),\ r(a)\} \\
C' & = & Res(C_1, C_2) = \{q(f(a)),\ r(a)\} \\
\\
L_1 & = & \{p(x),\ p(f(y)),\ p(f(z))\} \\
\lambda_1 & = & \{x \leftarrow f(y),\ z \leftarrow y\} \\
L_1\lambda_1 & = & \{p(f(y))\} \\
\\
L_2 & = & \{\neg\, p(f(u)),\ \neg\, p(w)\} \\
\lambda_2 & = & \{w \leftarrow f(u)\} \\
L_2\lambda_2 & = & \{\neg\, p(f(u))\} \\
\\
\lambda & = & \lambda_1 \cup \lambda_2 = \{x \leftarrow f(y), z \leftarrow y, w \leftarrow f(u)\} \\
L_1\lambda & = & \{p(f(y))\} \\
C_1\lambda & = & \{p(f(y)),\ q(f(y))\} \\
L_2\lambda & = & \{\neg\, p(f(u))\} \\
C_2\lambda & = & \{\neg\, p(f(u)),\ r(u)\} \\
\\
\sigma & = & \{u \leftarrow y\} \\
C & = & Res(C_1\lambda, C_2\lambda) = \{q(f(y)),\ r(y)\},\ \text{using}\ \sigma \\
\\
\lambda\sigma & = & \{x \leftarrow f(y),\ z \leftarrow y,\ w \leftarrow f(y),\ u \leftarrow y\} \\
C_1\lambda\sigma & = & \{p(f(y)),\ q(f(y))\} \\
C_2\lambda\sigma & = & \{\neg\, p(f(y)),\ r(y)\} \\
C & = & Res(C_1, C_2) = \{q(f(y)),\ r(y)\},\ \text{using}\ \lambda\sigma \\
\\
\theta_1' & = & \{y \leftarrow a\} \\
C_1' & = & C_1\theta_1 = \{p(f(a)),\ q(f(a))\} = C_1\lambda\sigma\theta_1 \\
\theta_2' & = & \{y \leftarrow a\} \\
C_2' & = & C_2\theta_2 = \{\neg\, p(f(a)),\ r(a)\} = C_2\lambda\sigma\theta_2 \\
\\
\theta' & = & \{y \leftarrow a\} \\
C' & = & Res(C_1', C_2') = \{q(f(a)),\ r(a)\}
\end{array}
$$

Fig. 10.1 Example for the lifting lemma

A node is a failure node if the (partial) interpretation defined by a branch falsifies some *ground instance* of a clause in S.

The critical step in the proof is showing that an inference node n can be associated with the resolvent of the clauses on the two failure nodes n_1, n_2 below it. Suppose that C_1, C_2 are associated with the failure nodes. Then there must be ground instances C_1', C_2' which are falsified at the nodes. By construction of the semantic tree, C_1' and C_2' are clashing clauses. Hence they can be resolved to give a clause C' which is falsified by the interpretation at n. By the Lifting Lemma, there is a clause C such that C is the resolvent of C_1' and C_2', and C' *is a ground instance of* C. Hence C is falsified at n and n (or an ancestor of n) is a failure node. ■

10.6 Summary

General resolution has proved to be a successful method for automated theorem proving in first-order logic. The key to its success is the unification algorithm. There is a large literature on strategies for choosing which clauses to resolve, but that is beyond the scope of this book. In Chap. 11 we present *logic programming*, in which programs are written as formulas in a restricted clausal form. In logic programming, unification is used to compose and decompose data structures, and computation is carried out by an appropriately restricted form of resolution that is very efficient.

10.7 Further Reading

Loveland (1978) is a classic book on resolution; a more modern one is Fitting (1996). Our presentation of the unification algorithm is taken from Martelli and Montanari (1982). Lloyd (1987) presents resolution in the context of logic programming that is the subject of the next chapter.

10.8 Exercises

10.1 Prove that ground resolution is sound and complete.

10.2 Let:

$$\begin{aligned}\theta &= \{x \leftarrow f(g(y)),\ y \leftarrow u,\ z \leftarrow f(y)\},\\ \sigma &= \{u \leftarrow y,\ y \leftarrow f(a),\ x \leftarrow g(u)\},\\ E &= p(x, f(y), g(u), z).\end{aligned}$$

Show that $E(\theta\sigma) = (E\theta)\sigma$.

10.3 Prove that the composition of substitutions is associative (Lemma 10.10).

10.4 Unify the following pairs of atomic formulas, if possible.

$$\begin{array}{ll} p(a, x, f(g(y))), & p(y, f(z), f(z)),\\ p(x, g(f(a)), f(x)), & p(f(a), y, y),\\ p(x, g(f(a)), f(x)), & p(f(y), z, y),\\ p(a, x, f(g(y))), & p(z, h(z, u), f(u)). \end{array}$$

10.5 A substitution $\theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ is *idempotent* iff $\theta = \theta\theta$. Let V be the set of variables occurring in the terms $\{t_1, \ldots, t_n\}$. Prove that θ is idempotent iff $V \cap \{x_1, \ldots, x_n\} = \emptyset$. Show that the mgu's produced by the unification algorithm is idempotent.

10.6 Try to unify the set of term equations:

$$x = f(y), \qquad y = g(x).$$

What happens?

10.7 Show that the composition of substitutions is not commutative: $\theta_1\theta_2 \neq \theta_2\theta_2$ for some θ_1, θ_2.

10.8 Unify the atoms in Example 10.13 using both term equations and Robinson's algorithm.

10.9 Let S be a finite set of expressions and θ a unifier of S. Prove that θ is an idempotent mgu iff for every unifier σ of S, $\sigma = \theta\sigma$.

10.10 Prove the validity of (some of) the equivalences in by resolution refutation of their negations.

References

M. Fitting. *First-Order Logic and Automated Theorem Proving (Second Edition)*. Springer, 1996.
J.W. Lloyd. *Foundations of Logic Programming (Second Edition)*. Springer, Berlin, 1987.
D.W. Loveland. *Automated Theorem Proving: A Logical Basis*. North-Holland, Amsterdam, 1978.
A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.
J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.