Chapter 1

Resolution in Predicate Logic

In this chapter, we will demonstrate how the resolution method introduced in Chapter **??** can be adapted for predicate logic. This chapter, the last in the section on predicate logic, is quite extensive, so let's provide an overview of its structure:

- We begin with an informal introduction (Section **??**).

In the following three sections, we introduce the tools that allow us to deal with the specifics of predicate logic: quantifiers, variables, and terms.

- In Section **??**, we show how to remove quantifiers using *Skolemization*, to obtain open formulas that can be converted into CNF.
- In Section **??**, we explain that we could seek a resolution refutation 'at the propositional logic level' (so-called *grounding*), if we first substitute 'suitable' constant terms for the variables.
- In Section **??**, we show how to find such 'suitable' substitutions using the *unification algorithm*.

This will give us all the necessary tools to present the resolution method itself. The rest of the chapter follows a similar structure to Chapter **??**.

- The resolution rule, the resolution proof, and related concepts are described in Section **??**.
- Section **??** is dedicated to the proof of soundness and completeness.
- Finally, in Section **??**, we describe LI-resolution and its application in Prolog.

1.1 Introduction

Just as in propositional logic, the resolution method in predicate logic is based on proof by contradiction. To prove that a sentence φ holds in a theory T (i.e., $T \models \varphi$), we start with the theory $T \cup \{\neg\varphi\}$. We 'convert' this theory to CNF, and then reject the resulting set of clauses S by resolution (i.e., show that $S \vdash_R \square$), thereby showing that it is unsatisfiable.

What do we mean by conjunctive normal form? The role of a *literal* is played by an *atomic formula*[1] or its negation. A *clause* is (in set representation) a finite set of literals, and a *formula* is a set of clauses.[2] Otherwise, we use the same terminology, e.g., we talk about *positive*, *negative*, *opposite* literals, $\square$ denotes an empty clause (which is unsatisfiable), etc.

First, let's informally demonstrate the specifics of resolution in predicate logic with a few very simple examples.

Notice first that if the theory T and the sentence φ are *open* (do not contain quantifiers), we can easily construct a CNF formula S *equivalent* to the theory $T \cup \{\neg\varphi\}$ (i.e., having the same set of models). Even universal quantifiers at the beginning of the formula are not problematic; we can remove them without changing the meaning.[3]

Example 1.1.1. Let $T = \{(\forall x)P(x), (\forall x)(P(x) \to Q(x))\}$ and $\varphi = (\exists x)Q(x)$. It is easily seen that

$$T \sim \{P(x), P(x) \to Q(x)\} \sim \{P(x), \neg P(x) \vee Q(x)\}$$

and also:

$$\neg\varphi = \neg(\exists x)Q(x) \sim (\forall x)\neg Q(x) \sim \neg Q(x)$$

Therefore, we can convert the theory $T \cup \{\neg\varphi\}$ to an *equivalent* CNF formula

$$S = \{\{P(x)\}, \{\neg P(x), Q(x)\}, \{\neg Q(x)\}\}$$

which we can easily refute by resolution in two steps. (Imagine $P(x)$ as a propositional variable p and $Q(x)$ as a propositional variable q.)

Generally, this won't be possible, particularly with existential quantifiers. Unlike in propositional logic, *not* every theory is equivalent to a CNF formula. However, we can always find an *equisatisfiable* CNF formula, i.e., one that is unsatisfiable *if and only if* $T \cup \{\neg\varphi\}$ is unsatisfiable, which is sufficient for proof by contradiction. This construction is called *Skolemization* and involves replacing existentially quantified variables with newly added constant or function symbols.

For example, we replace the formula $(\exists x)\psi(x)$ with the formula $\psi(x/c)$, where c is a new constant symbol representing a *witness*, i.e., an element that satisfies the existential quantifier. Since there may be many such elements, we lose *equivalence* of theories, but it holds that if the original formula is satisfiable, then the new formula is also satisfiable, and vice versa.

Example 1.1.2. If $T = \{(\exists x)P(x), P(x) \leftrightarrow Q(x)\}$ and $\varphi = (\exists x)Q(x)$, then

$$\neg\varphi \sim (\forall x)\neg Q(x) \sim \neg Q(x)$$

and equivalence can be converted to CNF as usual, giving:

$$T \cup \{\neg\varphi\} \sim \{(\exists x)P(x), \neg P(x) \vee Q(x), \neg Q(x) \vee P(x), \neg Q(x)\}$$

Now we replace the formula $(\exists x)P(x)$ with $P(c)$, where c is a new constant symbol. This gives the CNF formula:

$$S = \{\{P(c)\}, \{\neg P(x), Q(x)\}, \{\neg Q(x), P(x)\}, \{\neg Q(x)\}\}$$

It is not equivalent to the theory $T \cup \{\neg\varphi\}$, but it is *equisatisfiable* (in this case, both are unsatisfiable).

[1] I.e., $R(t_1, \ldots, t_n)$ or $t_1 = t_2$, where t_i are L-terms and R is an n-ary relation symbol from L.

[2] As in propositional logic, we also allow infinite sets of clauses.

[3] Any formula is equivalent to its *general closure*, and the equivalence holds in both directions.

Skolemization can be more complex; sometimes a constant symbol is not enough. If we have a formula of the form $(\forall x)(\exists y)\psi(x,y)$, the chosen witness for y depends on the chosen value for x, so 'y is a function of x'. In this case, we must replace y with $f(x)$, where f is a new unary function symbol. This gives the formula $(\forall x)\psi(x, y/f(x))$, and we can now remove the universal quantifier and write only $\psi(x, y/f(x))$, which is now an open formula, albeit in a different language (extended by the symbol f). Skolemization is formally described, and the necessary properties are proved in Section **??**.

Now let's look at the *resolution rule*. In predicate logic, it is more complex. Again, we will show just a few examples; the formal definition will come later (Section **??**).

Example 1.1.3. In the previous example, we arrived at the following CNF formula S, which is unsatisfiable, and we want to refute it by resolution:

$$S = \{\{P(c)\}, \{\neg P(x), Q(x)\}, \{\neg Q(x), P(x)\}, \{\neg Q(x)\}\}$$

If we look at it 'at the propositional logic level' ('ground level') and replace each atomic formula with a new propositional variable, we get $\{\{r\}, \{\neg p, q\}, \{\neg q, p\}, \{\neg q\}\}$, which is not unsatisfiable. We need to use the fact that $P(c)$ and $P(x)$ have a 'similar structure' (they are *unifiable*).

Since the clause $\{\neg P(x), Q(x)\}$ holds, it also holds after performing *any substitution*, i.e., the clause $\{\neg P(x/t), Q(x/t)\}$ is a consequence of S for any term t. We might imagine that we 'add' all such clauses to S.[4] The resulting CNF formula, when converted to the 'propositional logic level', would be unsatisfiable.

The *unification algorithm* directly tells us that the correct substitution is x/c, and we include this already in the *resolution rule*, i.e., the *resolvent* of the clauses $\{P(c)\}$ and $\{\neg P(x), Q(x)\}$ will be the clause $\{Q(c)\}$.

Unification can be more complex, and we should also note another difference from propositional logic: we allow resolution over multiple literals at once if they are all *unifiable* together:

Example 1.1.4. From the clauses $\{R(x, f(x)), R(g(y), z)\}$ and $\{\neg R(g(c), u), P(u)\}$ (where R is binary relational, f and g are unary functional, and c is a constant symbol), it will be possible to derive the resolvent $\{P(f(g(c)))\}$ using the *substitution* (*unification*) $\{x/g(c), y/c, z/f(g(c)), u/f(g(c))\}$, where we choose *both* literals at once from the first clause.

Remark 1.1.5. The fact that variables have a 'local meaning' in individual clauses (i.e., we can substitute them in one clause without affecting other clauses) follows from the following simple tautology, which holds for any formulas ψ, χ (even if x is free in both):

$$\models (\forall x)(\psi \wedge \chi) \leftrightarrow (\forall x)\psi \wedge (\forall x)\chi$$

As seen in the previous example, we will also require that clauses in the resolution rule have disjoint sets of variables; this can be achieved by renaming variables, which is a special case of substitution.

1.2 Skolemization

In this section, we will show how to reduce the question of the satisfiability of a given theory T to the question of the satisfiability of an *open* theory T'. Recall that T and T' will generally not be equivalent but will be *equisatisfiable*:

[4] There are infinitely many such clauses; even just the *variants* of one clause, i.e., clauses obtained by simply renaming variables, are infinite in number. This does not matter; a CNF formula can, by definition, be infinite.

Definition 1.2.1 (Equisatisfiability)**.** Given a theory T in language L and a theory T' in not necessarily the same language L', we say that T and T' are *equisatisfiable* if:

$$T \text{ has a model } \Leftrightarrow T' \text{ has a model}$$

The entire construction consists of the following steps, which we will explain below:

1. Conversion to *prenex normal form* (bringing quantifiers to the front).
2. Replacing formulas with their general closures (to obtain sentences).
3. Removing existential quantifiers (replacing sentences with *Skolem forms*).
4. Removing remaining universal quantifiers (resulting in open formulas).

1.2.1 Prenex Normal Form

First, we will show the process by which any formula can have its quantifiers 'brought to the front', i.e., converted to the so-called *prenex normal form*, which begins with a sequence of quantifiers and continues with only the free formula.

Definition 1.2.2 (PNF)**.** A formula φ is in *prenex normal form (PNF)* if it is of the form

$$(Q_1 x_1) \dots (Q_n x_n) \varphi'$$

where Q_i is a quantifier (either $\forall$ or $\exists$), and the formula φ' is open. The formula φ' is called the *open kernel* of φ, and $(Q_1 x_1) \dots (Q_n x_n)$ is the *quantifier prefix*.

If φ is a formula in PNF and all the quantifiers are universal, then we say that φ is a *universal* formula.

The goal of this subsection is to show the following proposition:

Proposition 1.2.3 (Conversion to PNF)**.** *For each formula φ, there exists an equivalent formula in prenex normal form.*

The algorithm, like the conversion to CNF, will be based on replacing subformulas with *equivalent* subformulas, aiming to move quantifiers closer to the root of the formula tree. What do we mean by the equivalence of formulas $\varphi \sim \varphi'$? That they have the same meaning, i.e., they have the same truth value in every model and for every variable assignment. Equivalently, that $\models \varphi \leftrightarrow \varphi'$ holds. We will need the following simple observation:

Observation 1.2.4. *If we replace a subformula ψ of a formula φ with an equivalent formula ψ', then the resulting formula φ' is also equivalent to the formula φ.*

The conversion is based on repeated application of the following syntactic rules:

Lemma 1.2.5. *Let $\overline{Q}$ denote the quantifier opposite to Q. Let φ and ψ be formulas, and let the variable x be free in φ but not in ψ. Then the following hold:*

$$\begin{aligned}
\neg(Qx)\varphi &\sim (\overline{Q}x)\neg\varphi \\
(Qx)\varphi \wedge \psi &\sim (Qx)(\varphi \wedge \psi) \\
(Qx)\varphi \vee \psi &\sim (Qx)(\varphi \vee \psi) \\
(Qx)\varphi \to \psi &\sim (\overline{Q}x)(\varphi \to \psi) \\
\psi \to (Qx)\varphi &\sim (Qx)(\psi \to \varphi)
\end{aligned}$$

Proof. The rules can be easily verified semantically, or proved using the tableau method (if they are not sentences, we must replace them with their general closures). □

Note that in the rule $(Qx)\varphi \to \psi \sim (\overline{Q}x)(\varphi \to \psi)$ for bringing a quantifier out of the *antecedent* of an implication, we must change the quantifier (from $\forall$ to $\exists$ and vice versa), whereas when bringing it out of the *consequent*, the quantifier remains the same. Why is this so? This is best seen if we rewrite the implication using disjunction and negation:

$$(Qx)\varphi \to \psi \sim \neg(Qx)\varphi \vee \psi \sim (\overline{Q}x)(\neg\varphi) \vee \psi \sim (\overline{Q}x)(\neg\varphi \vee \psi) \sim (\overline{Q}x)(\varphi \to \psi)$$

Also note the assumption that x is not free in ψ. Without it, the rules would not work, e.g.:

$$(\exists x)P(x) \wedge Q(x) \;\not\sim\; (\exists x)(P(x) \wedge Q(x))$$

In such a situation, we replace the formula with a variant in which we rename the bound variable x to a new variable:

$$(\exists x)P(x) \wedge Q(x) \;\sim\; (\exists y)P(y) \wedge Q(x) \sim (\exists y)(P(y) \wedge Q(x))$$

Exercise 1.1. Prove Observation **??** and all the rules in Lemma **??**.

Let's demonstrate the process with an example:

Example 1.2.6. Convert the formula $((\forall z)P(x,z) \wedge P(y,z)) \to \neg(\exists x)P(x,y)$ to PNF. We will write only the individual steps. Note what rule was applied to which subformula (and also the renaming of the variable in the first step), and follow the process on the formula tree.

$$\begin{aligned}
&(\forall z)P(x,z) \wedge P(y,z) \;\to\; \neg(\exists x)P(x,y)\\
\sim\; &(\forall u)P(x,u) \wedge P(y,z) \;\to\; (\forall x)\neg P(x,y)\\
\sim\; &(\forall u)(P(x,u) \wedge P(y,z)) \;\to\; (\forall v)\neg P(v,y)\\
\sim\; &(\exists u)(P(x,u) \wedge P(y,z) \;\to\; (\forall v)\neg P(v,y))\\
\sim\; &(\exists u)(\forall v)(P(x,u) \wedge P(y,z) \;\to\; \neg P(v,y))
\end{aligned}$$

Now we are ready to prove Proposition **??**:

Proof of Proposition **??**. By induction on the structure of the formula φ, using Lemma **??** and Observation **??**. □

Since every formula $\varphi(x_1, \ldots, x_n)$ is equivalent to its *general closure*

$$(\forall x_1) \ldots (\forall x_n)\varphi(x_1, \ldots, x_n)$$

we can state Proposition **??** as follows:

Corollary 1.2.7. *For every formula φ, there exists an equivalent* sentence *in PNF.*

For example, in Example **??**, the resulting sentence is $(\forall x)(\forall y)(\forall z)(\exists u)(\forall v)(P(x,u) \wedge P(y,z) \to \neg P(v,y))$.

Remark 1.2.8. The prenex form is not unique; the rules for conversion can be applied in different orders. As we will see in the next subsection, it is advantageous to bring out quantifiers that become existential first: if we have a choice between $(\forall x)(\exists y)\varphi(x,y)$ and $(\exists y)(\forall x)\varphi(x,y)$, we choose the second option because, in the first case, 'y depends on x'.

1.2.2 Skolem Form

Now we have converted our axioms to equivalent sentences in prenex form. If any sentence contained only universal quantifiers, i.e., it was of the form

$$(\forall x_1)\dots(\forall x_n)\varphi(x_1,\dots,x_n)$$

where φ is open, we could simply replace it with its open kernel φ, which is equivalent in this case. But how do we deal with existential quantifiers, e.g., $(\exists x)\varphi(x)$, $(\forall x)(\exists y)\varphi(x,y)$, etc.? We first replace them with their *Skolem form.*

Definition 1.2.9 (Skolem Form)**.** Let φ be an L-sentence in PNF, and let all its bound variables be different. Let the existential quantifiers from the prefix of φ be $(\exists y_1),\dots,(\exists y_n)$ (in this order), and let for each i, $(\forall x_1),\dots,(\forall x_{n_i})$ be precisely all the universal quantifiers preceding the quantifier $(\exists y_i)$ in the prefix of φ.

Let L' be the extension of L with *new* n_i-ary function symbols $f_1,\dots,f_n$, where the symbol f_i has arity n_i for each i. The *Skolem form* of the sentence φ is the L'-sentence φ_S obtained from φ by, for each $i=1,\dots,n$:

- removing the quantifier $(\exists y_i)$ from the prefix, and
- substituting the variable y_i with the term $f_i(x_1,\dots,x_{n_i})$.

This process is also called *skolemization.*

Example 1.2.10. The Skolem form of the sentence

$$\varphi=(\exists y_1)(\forall x_1)(\forall x_2)(\exists y_2)(\forall x_3)R(y_1,x_1,x_2,y_2,x_3)$$

is the sentence

$$\varphi_S=(\forall x_1)(\forall x_2)(\forall x_3)R(f_1,x_1,x_2,f_2(x_1,x_2),x_3)$$

where f_1 is a new constant symbol, and f_2 is a new binary function symbol.

Remark 1.2.11. Note that when skolemizing, we must start from a sentence! For example, given the formula $(\exists y)E(x,y)$, $E(x,c)$ is not its Skolem form. We must first take the general closure $(\forall x)(\exists y)E(x,y)$, and then correctly skolemize it as $(\forall x)E(x,f(x))$, which is equivalent to the open formula $E(x,f(x))$ (which says something much weaker than $E(x,c)$).

It is also important that each symbol used in skolemization is genuinely new; its only 'role' in the whole theory must be to represent the 'existing' elements in this formula.

In the following lemma, we show the key property of the Skolem form:

Lemma 1.2.12. *Let $\varphi=(\forall x_1)\dots(\forall x_n)(\exists y)\psi$ be an L-sentence, and let φ' be the sentence*

$$(\forall x_1)\dots(\forall x_n)\psi(y/f(x_1,\dots,x_n))$$

where f is a new function symbol. Then:

(i) The L-reduct of every model of φ' is a model of φ, and

(ii) Every model of φ can be expanded to a model of φ'.

Proof. First, prove part (i): Let $\mathcal{A}' \models \varphi'$, and let $\mathcal{A}$ be its reduct to the language L. For each variable assignment e, $\mathcal{A} \models \psi[e(y/a)]$ holds for $a = (f(x_1, \ldots, x_n))^{\mathcal{A}'}[e]$, so $\mathcal{A} \models \varphi$.

Now, part (ii): Since $\mathcal{A} \models \varphi$, there exists a function $f^A : A^n \to A$ such that for every variable assignment e, $\mathcal{A} \models \psi[e(y/a)]$, where $a = f^A(e(x_1), \ldots, e(x_n))$. Thus, the expansion of the structure $\mathcal{A}$ obtained by adding the function f^A is a model of φ'. □

Remark 1.2.13. The expansion of the model in the second part of the lemma need not be (and typically is not) unique, unlike the extension by defining a new function symbol.

Applying the previous lemma repeatedly (for all existential quantifiers in sequence) gives us the following corollary:

Corollary 1.2.14. *A sentence φ and its Skolem form φ_S are equisatisfiable.*

1.2.3 Skolem's Theorem

In this subsection, we summarize the entire process described in the previous subsections. The key is the following theorem by Norwegian logician Thoralf Skolem:

Theorem 1.2.15 (Skolem's Theorem)**.** *Every theory has an open conservative extension.*

Proof. Let T be an L-theory. Replace each axiom with its general closure (if it is not already a sentence) and convert it to PNF, obtaining an equivalent theory T'. Now replace each axiom of the theory T' with its Skolem form. This gives us a theory T'' in the extended language L'. From Lemma **??**, it follows that the L-reduct of each model of T'' is a model of T', so T'' is an extension of T', and that each model of T' can be expanded to the language L' to be a model of T'', so it is a conservative extension. The theory T'' is axiomatized by universal sentences; removing the quantifier prefixes (i.e., taking the kernels of the axioms) gives us the open theory T''', which is equivalent to T'' and therefore also a conservative extension of T. □

From the semantic characterization of conservative extension, the following corollary easily follows:

Corollary 1.2.16. *For every theory, we can find an equisatisfiable open theory using skolemization.*

We can now easily convert an open theory to CNF (expressed as a *formula S* in set representation) using equivalent syntactic transformations, just as in propositional logic (see Section **??**).

1.3 Grounding

In this section, we show that if we have an open theory that is unsatisfiable, we can demonstrate its unsatisfiability 'on specific elements'. What do we mean by that? There exists a finite number of *ground (basic) instances* of the axioms (instances where we substitute constant terms for the variables) such that their conjunction (which contains no variables) is unsatisfiable.

Definition 1.3.1 (Ground Instance)**.** Let φ be an open formula in free variables $x_1, \ldots, x_n$. We say that the instance $\varphi(x_1/t_1, \ldots, x_n/t_n)$ is a *ground instance* if all terms $t_1, \ldots, t_n$ are ground (constant).

Example 1.3.2. The theory $T = \{P(x,y) \vee R(x,y), \neg P(c,y), \neg R(x,f(x))\}$ in the language $L = \langle P, R, f, c\rangle$ has no model. We can demonstrate this with the following conjunction of ground instances of the axioms, where we substitute the constant c for the variable x and the ground term $f(c)$ for y:

$$(P(c,f(c)) \vee R(c,f(c))) \ \wedge \ \neg P(c,f(c)) \ \wedge \ \neg R(c,f(c))$$

This sentence is clearly unsatisfiable. Moreover, the ground atomic sentences $(P(c,f(c))$ and $R(c,f(c))$ can be understood (because they contain no variables) as propositional variables p_1, p_2, where p_1 means 'it holds that $P(c,f(c))$' and p_2 means 'it holds that $R(c,f(c))$'. We then get the following proposition, which can be easily refuted by resolution:

$$(p_1 \vee p_2) \wedge \neg p_1 \wedge \neg p_2$$

This process of converting to ground instances (and thus to propositional logic) is called 'grounding'. We will formalize it shortly and prove *Herbrand's theorem*,[5] which states that such an unsatisfiable conjunction of ground instances of the axioms exists for every unsatisfiable theory.

1.3.1 Direct Reduction to Propositional Logic

Now realize that thanks to Herbrand's theorem, grounding allows the following (although inefficient) procedure for refuting formulas by resolution 'at the propositional logic level': In the input formula S, we replace each clause with the set of all its ground instances (if there are none, i.e., if the language does not contain a constant symbol, we add one constant symbol to the language). In the resulting set of clauses S', we treat atomic sentences as propositional variables, and we refute S' using propositional resolution (which we know is sound and complete).

The problem with this approach is that the number of clauses in S' (ground instances of the clauses from S) can be large, even infinite, e.g., whenever the language contains at least one functional (non-constant) symbol.

Example 1.3.3. If we have a CNF formula $S = \{\{P(x,y), R(x,y)\}, \{\neg P(c,y)\}, \{\neg R(x,f(x))\}\}$ in the language $L = \langle f, c\rangle$, we replace it with the following infinite formula S':

$$\begin{aligned} S' = \{&\{P(c,c), R(c,c)\}, \{P(c,f(c)), R(c,f(c))\}, \{P(f(c),c), R(f(c),c)\}, \ldots, \\ &\{\neg P(c,c)\}, \{\neg P(c,f(c))\}, \{\neg P(c,f(f(c)))\}, \{\neg P(c,f(f(f(c))))\}, \ldots, \\ &\{\neg R(c,f(c))\}, \{\neg R(f(c),f(f(c)))\}, \{\neg R(f(f(c)),f(f(f(c))))\}, \ldots\} \end{aligned}$$

It is unsatisfiable because it contains the following finite subset, which is unsatisfiable, as we can easily show by propositional resolution:

$$\{\{P(c,f(c)), R(c,f(c))\}, \{\neg P(c,f(c))\}, \{\neg R(c,f(c))\}\} \vdash_R \square$$

In Section **??**, we will show an efficient procedure for finding suitable ground instances of clauses using so-called *unification*.

[5] French mathematician Jacques Herbrand worked at the end of the 1920s. During his short career (he tragically died at the age of 23), he discovered several other important results, and among other things, formalized the concept of a recursive function.

1.3.2 Herbrand's Theorem

In this subsection, we state and prove Herbrand's theorem. We assume that the language contains some constant symbol: if the language does not contain any, we add one. We need a constant symbol so that there are ground terms and we can create the so-called *Herbrand model*. This is the construction of a semantic object (model) from syntactic objects (ground terms) very similar to the *canonical model* (Definition **??**).[6]

Definition 1.3.4 (Herbrand Model)**.** Let $L = \langle \mathcal{R}, \mathcal{F} \rangle$ be a language with at least one constant symbol. An L-structure $\mathcal{A} = \langle A, \mathcal{R}^{\mathcal{A}}, \mathcal{F}^{\mathcal{A}} \rangle$ is a *Herbrand model* if:

- A is the set of all ground L-terms (the so-called *Herbrand universe*), and
- for each n-ary function symbol $f \in \mathcal{F}$ and ground terms "t_1", ..., "t_n" $\in A$,

$$f^{\mathcal{A}}(\text{“}t_1\text{”}, \ldots, \text{“}t_n\text{”}) = \text{“}f(t_1, \ldots, t_n)\text{”}$$

- Specifically, for each constant symbol $c \in \mathcal{F}$, $c^{\mathcal{A}} = \text{“}c\text{”}$.

We do not impose any conditions on the interpretations of relation symbols.

Recall that we use quotes around terms informally to clearly distinguish terms as syntactic objects (strings of symbols) from their interpretations (functions).

Example 1.3.5. Let $L = \langle P, f, c \rangle$ be a language where P is a unary relation symbol, f is a binary function symbol, and c is a constant symbol. The Herbrand universe for this language is the set

$$A = \{\text{“}c\text{”}, \text{“}f(c,c)\text{”}, \text{“}f(c, f(c,c))\text{”}, \text{“}f(f(c,c),c)\text{”} \ldots\}$$

The structure $\mathcal{A} = \langle A, P^{\mathcal{A}}, f^{\mathcal{A}}, c^{\mathcal{A}} \rangle$ is a Herbrand model if $c^{\mathcal{A}} = \text{“}c\text{”}$ and the function $f^{\mathcal{A}}$ satisfies:

- $f^{\mathcal{A}}(\text{“}c\text{”}, \text{“}c\text{”}) = \text{“}f(c,c)\text{”}$,
- $f^{\mathcal{A}}(\text{“}c\text{”}, \text{“}f(c,c)\text{”}) = \text{“}f(c, f(c,c))\text{”}$,
- $f^{\mathcal{A}}(\text{“}f(c,c)\text{”}, \text{“}c\text{”}) = \text{“}f(f(c,c),c)\text{”}$, etc.

The relation $P^{\mathcal{A}}$ can be any subset of A.

We are now ready to state Herbrand's theorem. Informally, it states that if a theory is satisfiable, i.e., has a model, then it even has a Herbrand model, and otherwise, we can find an unsatisfiable conjunction of ground instances of the axioms, which can be used for resolution refutation 'at the propositional logic level'.

Theorem 1.3.6 (Herbrand's Theorem)**.** *Let T be an open theory in a language L without equality and with at least one constant symbol. Then T either has a* Herbrand model*, or there exist finitely many ground instances of the axioms of T whose conjunction is unsatisfiable.*

[6] The difference is that we do not add countably many new constant symbols (we only use the constant symbols already in the language), and we do not prescribe how the model relations should look.

Proof. Let T_{ground} denote the set of all ground instances of the axioms of the theory T. We construct a systematic[7] tableau from the theory T_{ground} with the entry $\mathrm{F}\bot$ at the root, but in the language L, without extending it with auxiliary constant symbols to the language L_C.[8]

If the tableau contains a non-contradictory branch, then the canonical model for this branch (again without adding auxiliary constant symbols) is a Herbrand model of T. Otherwise, we have a tableau proof of contradiction, so the theory T_{ground}, and therefore T, is unsatisfiable. Since the tableau proof is finite, we used only finitely many ground instances of the axioms $\alpha_{\text{ground}} \in T_{\text{ground}}$. Their conjunction is therefore unsatisfiable. □

Remark 1.3.7. If the language includes equality, we first extend the theory T with the axioms of equality to obtain the theory T^*, and if T^* has a Herbrand model $\mathcal{A}$, we factorize it according to the congruence $=^A$, just as in the case of the canonical model.

To conclude this section, we state two corollaries of Herbrand's theorem.

Corollary 1.3.8. *Let $\varphi(x_1, \ldots, x_n)$ be an open formula in a language L with at least one constant symbol. Then there exist ground L-terms t_{ij} $(1 \leq i \leq m, 1 \leq j \leq n)$ such that the sentence*

$$(\exists x_1) \ldots (\exists x_n)\varphi(x_1, \ldots, x_n)$$

is true if and only if the following formula (a propositional tautology) is true:

$$\varphi(x_1/t_{11}, \ldots, x_n/t_{1n}) \vee \cdots \vee \varphi(x_1/t_{m1}, \ldots, x_n/t_{mn})$$

Proof. The sentence $(\exists x_1) \ldots (\exists x_n)\varphi(x_1, \ldots, x_n)$ is true if and only if $(\forall x_1) \ldots (\forall x_n)\neg\varphi$ is unsatisfiable, i.e., if $\neg\varphi$ is unsatisfiable. This follows from Herbrand's theorem applied to the theory $T = \{\neg\varphi\}$. □

Corollary 1.3.9. *Let T be an open theory in a language with at least one constant symbol. The theory T has a model if and only if the theory T_{ground} consisting of all ground instances of the axioms of T has a model.*

Proof. In a model of the theory T, all the axioms hold, and so do all their ground instances. Therefore, it is also a model of T_{ground}. If T has no model, then by Herbrand's theorem, some finite subset of the theory T_{ground} is unsatisfiable. □

1.4 Unification

Instead of substituting *all* ground terms and working with this new, huge, and typically infinite set of clauses, it is better to find a 'suitable' substitution in a specific resolution step and work only with it. In this section, we will explain what 'suitable' means (so-called *unification*) and how to find it (using the *unification algorithm*).

[7] Or any complete tableau, but in such a way that we do not extend contradictory branches.

[8] Since there are no quantifiers in T_{ground}, auxiliary symbols are not used anywhere in the tableau.

1.4.1 Substitution

First, let's provide a few examples of 'suitable' substitutions:

Example 1.4.1.

- From the clauses $\{P(x), Q(x,a)\}$ and $\{\neg P(y), \neg Q(b,y)\}$, we obtain, using the substitution $\{x/b, y/a\}$, the clauses $\{P(b), Q(b,a)\}$ and $\{\neg P(a), \neg Q(b,a)\}$, and from them, we derive the clause $\{P(b), \neg P(a)\}$ by resolution. We could also use the substitution $\{x/y\}$ and derive the resolvent $\{Q(y,a), \neg Q(b,y)\}$ by resolving over $P(y)$.

- Given the clauses $\{P(x), Q(x,a), Q(b,y)\}$ and $\{\neg P(v), \neg Q(u,v)\}$, a suitable substitution is $\{x/b, y/a, u/b, v/a\}$; we get $\{P(b), Q(b,a)\}$ and $\{\neg P(a), \neg Q(b,a)\}$, whose resolvent is $\{P(b), \neg P(a)\}$.

- Consider the clauses $\{P(x), Q(x,z)\}$ and $\{\neg P(y), \neg Q(f(y),y)\}$. We could use the substitution $\{x/f(a), y/a, z/a\}$ to obtain the pair of clauses $\{P(f(a)), Q(f(a),a)\}$ and $\{\neg P(a), \neg Q(f(a),a)\}$, resolving to $\{P(f(a)), \neg P(a)\}$.

  However, it is better to use the substitution $\{x/f(z), y/z\}$, after which we have $\{P(f(z)), Q(f(z),z)\}$ and $\{\neg P(z), \neg Q(f(z),z)\}$, and the resolvent $\{P(f(z)), \neg P(z)\}$. This substitution is *more general*, and the resulting resolvent 'says more' than $\{P(f(a)), \neg P(a)\}$ (the latter is its consequence, but not vice versa).

We now introduce the necessary terminology related to substitutions. Substitutions will be applied to terms or literals (atomic formulas or their negations), collectively referred to as *expressions*.

Definition 1.4.2 (Substitution)**.** A *substitution* is a finite set $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$, where x_i are distinct variables and t_i are terms, with the requirement that term t_i is not equal to variable x_i. The substitution σ is

- *ground* if all terms t_i are constant,
- a *variable renaming* if all terms t_i are distinct variables.

An *instance* of an expression (term or literal) E *under the substitution* $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$ is the expression obtained from E by simultaneously replacing all occurrences of variables x_i with terms t_i, denoted $E\sigma$. If S is a set of expressions, we denote $S\sigma = \{E\sigma \mid E \in S\}$.

Since variables are replaced *simultaneously* for all variables at once, any occurrence of variable x_i in term t_j will not lead to a chain of substitutions.

Example 1.4.3. For example, for $S = \{P(x), R(y,z)\}$ and substitution $\sigma = \{x/f(y,z), y/x, z/c\}$, we have:

$$S\sigma = \{P(f(y,z)), R(x,c)\}$$

Substitutions can be naturally *composed*. The composition of substitutions σ and τ, where σ is applied first and then τ, will be denoted as $\sigma\tau$. Thus, $E(\sigma\tau) = (E\sigma)\tau$ holds for any expression E.

Example 1.4.4. Let's start with an example. Given the expression $E = P(x,w,u)$ and the substitutions

$$\sigma = \{x/f(y), w/v\}$$
$$\tau = \{x/a, y/g(x), v/w, u/c\}$$

we have $E\sigma = P(f(y), v, u)$ and $(E\sigma)\tau = P(f(g(x)), w, c)$. Therefore, it must hold:

$$\sigma\tau = \{x/f(g(x)), y/g(x), v/w, u/c\}$$

Now the formal definition:

Definition 1.4.5 (Composition of Substitutions)**.** Let $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$ and $\tau = \{y_1/s_1, \ldots, y_m/s_m\}$ be substitutions. The *composition of substitutions* σ *and* τ is the substitution

$$\sigma\tau = \{x_i/t_i\tau \mid x_i \in X, x_i \neq t_i\tau\} \cup \{y_j/s_j \mid y_j \in Y \setminus X\}$$

where $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$.

Note that the composition of substitutions is not commutative; $\sigma\tau$ is typically a completely different substitution from $\tau\sigma$.

Example 1.4.6. If σ and τ are as in Example **??**, then:

$$\tau\sigma = \{x/a, y/g(f(y)), u/c, w/v\} \neq \sigma\tau$$

We now show that this defined composition of substitutions satisfies the required properties and that it is also *associative*. From associativity, it follows that we do not need to (and will not) write parentheses in the composition $\sigma\tau\varrho$, $\sigma_1\sigma_2\cdots\sigma_n$, etc.

Proposition 1.4.7. *Let σ, τ, and ϱ be substitutions, and let E be any expression. Then the following holds:*

(i) $(E\sigma)\tau = E(\sigma\tau)$

(ii) $(\sigma\tau)\varrho = \sigma(\tau\varrho)$

Proof. Let $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$ and $\tau = \{y_1/s_1, \ldots, y_m/s_m\}$. It is sufficient to prove for the case when the expression E is a single variable, the rest easily follows by induction. (Substitutions do not change other symbols.) We divide into three cases:

- If $E = x_i$ for some i, then $E\sigma = t_i$ and $(E\sigma)\tau = t_i\tau = E(\sigma\tau)$, where the second equality is by definition of $\sigma\tau$.
- If $E = y_j$ for some j, where $y_j \notin \{x_1, \ldots, x_n\}$, then $E\sigma = E$ and $(E\sigma)\tau = E\tau = s_j = E(\sigma\tau)$ again by definition of $\sigma\tau$.
- If E is another variable, then $(E\sigma)\tau = E = E(\sigma\tau)$.

This proves (i). Associativity (ii) can be easily proved by repeated use of (i). The following holds for any expression E, hence also for any variable:

$$E((\sigma\tau)\varrho) = (E(\sigma\tau))\varrho = ((E\sigma)\tau)\varrho = (E\sigma)(\tau\varrho) = E(\sigma(\tau\varrho)).$$

It follows that $(\sigma\tau)\varrho$ and $\sigma(\tau\varrho)$ are the same substitution.[9] □

[9] In more detail: we use the obvious property that for a substitution π it holds $\pi = \{z_1/v_1, \ldots, z_k/v_k\}$ if and only if $E\pi = v_i$ for $E = z_i$ and $E\pi = E$ if E is a variable different from all z_i.

1.4.2 Unification Algorithm

Which substitutions are 'suitable'? Those that, after being performed, make the given expressions 'become the same', i.e., *unified* (see Example **??**).

Definition 1.4.8 (Unification)**.** Let $S = \{E_1, \ldots, E_n\}$ be a finite set of expressions. A substitution σ is a *unifier for* S if $E_1\sigma = E_2\sigma = \cdots = E_n\sigma$, i.e., $S\sigma$ contains a single expression. If it exists, we also say that S is *unifiable*.

A unifier for S is *most general* if for every unifier τ for S there exists a substitution λ such that $\tau = \sigma\lambda$. Note that there may be multiple most general unifiers for S, but they differ only by variable renaming.

Example 1.4.9. Consider the set of expressions $S = \{P(f(x), y), P(f(a), w)\}$. The most general unifier for S is $\sigma = \{x/a, y/w\}$. Another unifier is, for example, $\tau = \{x/a, y/b, w/b\}$, but it is not most general because it cannot yield, for example, the unifier $\varrho = \{x/a, y/c, w/c\}$. The unifier τ can be obtained from the most general unifier σ using the substitution $\lambda = \{w/b\}$: $\tau = \sigma\lambda$.

We now introduce the *unification algorithm*. Its input is a non-empty, finite set of expressions S, and its output is either the most general unifier for S or information that S is not unifiable. The algorithm proceeds from the beginning of the expressions, successively applying substitutions to make the expressions more similar. We need the following definition:

Let p be the first (leftmost) position where some two expressions from S differ. Then the *disagreement set in* S, denoted $D(S)$, is the set of all subexpressions starting at position p of expressions in S.

Example 1.4.10. For $S = \{P(x, y), P(f(x), z), P(z, f(x))\}$, $p = 3$ and $D(S) = \{x, f(x), z\}$.

Algorithm (Unification Algorithm)**.**

- **Input**: a finite set of expressions $S \neq \emptyset$,
- **Output:** the most general unifier σ for S or information that S is not unifiable

(0) set $S_0 := S$, $\sigma_0 := \emptyset$, $k := 0$

(1) if $|S_k| = 1$, return $\sigma = \sigma_0\sigma_1 \cdots \sigma_k$

(2) determine if there is a variable x and a term t *not containing* x in $D(S_k)$

(3) if yes, set $\sigma_{k+1} := \{x/t\}$, $S_{k+1} := S_k\sigma_{k+1}$, $k := k + 1$, and go to (1)

(4) if no, report that S is not unifiable

Remark 1.4.11. Finding a variable x and a term t in step (2) can be computationally intensive.

Before proving correctness, let's demonstrate the algorithm with an example.

Example 1.4.12. Let's apply the unification algorithm to the following set:

$$S = \{P(f(y, g(z)), h(b)),\ P(f(h(w), g(a)), t),\ P(f(h(b), g(z)), y)\}$$

($k = 0$) The set $S_0 = S$ is not a singleton, $D(S_0) = \{y, h(w), h(b)\}$ contains the term $h(w)$ and the variable y not occurring in $h(w)$. Set $\sigma_1 = \{y/h(w)\}$ and $S_1 = S_0\sigma_1$, thus we have:

$$S_1 = \{P(f(h(w), g(z)), h(b)),\ P(f(h(w), g(a)), t),\ P(f(h(b), g(z)), h(w))\}$$

$(k = 1)$ $D(S_1) = \{w, b\}$, $\sigma_2 = \{w/b\}$, $S_2 = S_1\sigma_2$, thus:

$$S_2 = \{P(f(h(b), g(z)), h(b)),\ P(f(h(b), g(a)), t)\}$$

$(k = 2)$ $D(S_2) = \{z, a\}$, $\sigma_3 = \{z/a\}$, $S_3 = S_2\sigma_3$, thus:

$$S_3 = \{P(f(h(b), g(a)), h(b)),\ P(f(h(b), g(a)), t)\}$$

$(k = 3)$ $D(S_3) = \{h(b), t\}$, $\sigma_4 = \{t/h(b)\}$, $S_4 = S_3\sigma_4$, thus:

$$S_4 = \{P(f(h(b), g(a)), h(b))\}$$

$(k = 4)$ S_4 is a singleton, the most general unifier for S is the following:

$$\sigma = \sigma_1\sigma_2\sigma_3\sigma_4 = \{y/h(w)\}\{w/b\}\{z/a\}\{t/h(b)\} = \{y/h(b), w/b, z/a, t/h(b)\}$$

Proposition 1.4.13. *The unification algorithm is correct. For any input S, it terminates in a finite number of steps, and if S is unifiable, it returns the most general unifier σ; otherwise, it reports that S is not unifiable.*

If S is unifiable, then for the constructed most general unifier σ, it additionally holds that if τ is any unifier, then $\tau = \sigma\tau$.

Proof. In each step k, we eliminate some variable, so the algorithm must terminate. If the algorithm terminates unsuccessfully in step k, then it is not possible to unify the set S_k. It is easy to see that in that case, it is not possible to unify S either.

If the algorithm returns $\sigma = \sigma_0\sigma_1 \cdots \sigma_k$, it is evidently a unifier. It remains to prove that it is the most general, for which it suffices to prove the stronger property ('additionally') described in the proposition.

Let τ be any unifier for S. We will show by induction that for every $0 \leq i \leq k$, it holds:

$$\tau = \sigma_0\sigma_1 \cdots \sigma_i\tau$$

For $i = 0$, $\sigma_0 = \emptyset$ and $\tau = \sigma_0\tau$ holds trivially. Assume it holds for some i, and prove it for $i + 1$. Let $\sigma_{i+1} = \{x/t\}$. It suffices to prove that for any variable u, it holds:

$$u\sigma_{i+1}\tau = u\tau$$

From this, $\tau = \sigma_0\sigma_1 \cdots \sigma_i\sigma_{i+1}\tau$ immediately follows.

If $u \neq x$, then $u\sigma_{i+1} = u$, so $u\sigma_{i+1}\tau = u\tau$. In the case $u = x$, we have $u\sigma_{i+1} = x\sigma_{i+1} = t$. Since τ unifies the set $S_i = S\sigma_0\sigma_1 \cdots \sigma_i$, and variable x and term t‘$areinthedisagreementset\mathrm{D(S}_i)$, τ must unify x‘$andt.Inotherwords$,$\mathrm{t}\tau = x\tau$‘,$oru\sigma_{i+1}\tau = u\tau$‘,$whichwewantedtoprove$. □

1.5 Resolution Method

To prove that $T \models \varphi$, we can find a CNF formula S using skolemization that is unsatisfiable precisely when the theory $T \cup \{\neg\varphi\}$ is unsatisfiable, i.e., precisely when $T \models \varphi$. We then just need to find a resolution refutation of S.

In this section, we describe the resolution method itself. Most concepts and theorems will be very similar to propositional logic. The only significant difference will be the *resolution rule*.

1.5.1 Resolution Rule

The resolvent of a pair of clauses will be a clause derived from them by applying *(most general) unification*. First, an example:

Example 1.5.1. Let $C_1 = \{P(x), Q(x,y), Q(x,f(z))\}$ and $C_2 = \{\neg P(u), \neg Q(f(u),u)\}$ be clauses. Select *both* positive literals starting with Q from the first clause and the negative literal starting with $\neg Q$ from the second. The set of expressions $S = \{Q(x,y), Q(x,f(z)), Q(f(u),u)\}$ can be unified using the most general unifier $\sigma = \{x/f(f(z)), y/f(z), u/f(z)\}$. After applying this unification, we get the clauses $C_1\sigma = \{P(f(f(z))), Q(f(f(z)),f(z))\}$ and $C_2\sigma = \{\neg P(f(z)), \neg Q(f(f(z)),f(z))\}$, from which we derive the clause $C = \{P(f(f(z))), \neg P(f(z))\}$. This will be called the *resolvent* of the original clauses C_1 and C_2.

Definition 1.5.2 (Resolution Rule)**.** Let C_1 and C_2 be clauses with disjoint sets of variables and let them be of the form

$$C_1 = C_1' \sqcup \{A_1, \ldots, A_n\}, \quad C_2 = C_2' \sqcup \{\neg B_1, \ldots, \neg B_m\}$$

where $n, m \geq 1$ and the set of expressions $S = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$ is unifiable.[10] Let σ be the most general unifier of S.[11] The *resolvent* of the clauses C_1 and C_2 is the following clause:

$$C = C_1'\sigma \cup C_2'\sigma$$

Remark 1.5.3. The condition of disjoint sets of variables can always be met by renaming the variables in one of the clauses. Why is this necessary? For example, from the clauses $\{\{P(x)\}, \{\neg P(f(x))\}\}$ we can obtain the empty clause $\square$ if we replace the clause $\{P(x)\}$ with the clause $\{P(y)\}$. However, the set of expressions $\{P(x), P(f(x))\}$ is not unifiable, so this would not work without renaming variables.

1.5.2 Resolution Proof

Once we have defined the resolution rule, we can introduce the *resolution proof* and related concepts. The definitions will be the same as in propositional logic, with one difference: we allow renaming of variables in the clauses, see Remark **??**.

Definition 1.5.4 (Resolution Proof)**.** A *resolution proof (derivation)* of a clause C from a formula S is a *finite* sequence of clauses $C_0, C_1, \ldots, C_n = C$ such that for each i,

- either $C_i = C_i'\sigma$ for some clause $C_i' \in S$ and variable renaming σ, or
- C_i is the resolvent of some C_j, C_k where $j < i$ and $k < i$.

If a resolution proof exists, we say that C is *resolutively derivable* from S, and we write $S \vdash_R C$. A *(resolution) refutation* of the formula S is a resolution proof of $\square$ from S; in that case, S is *(resolutively) refutable*.

Remark 1.5.5. Why do we need to eliminate multiple literals from one clause at once in a resolution step? Consider the formula $S = \{\{P(x), P(y)\}, \{\neg P(x), \neg P(y)\}\}$. It is resolutively refutable, but there is no refutation that eliminates only one literal in each step.

[10] The symbol $\sqcup$ denotes *disjoint union*.

[11] Recall that unification means that $A_1\sigma = A_2\sigma = \cdots = B_1\sigma = \cdots = B_m\sigma$.

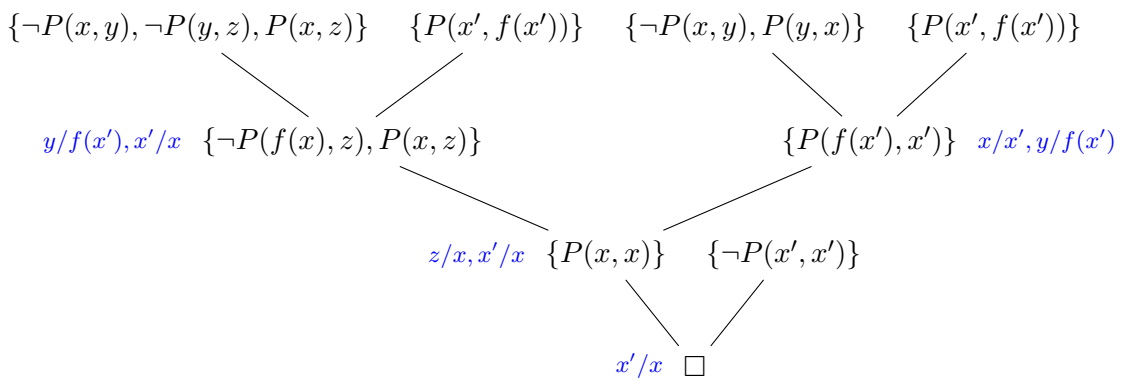


Figure 1.1: Resolution refutation of the formula S from Example **??**. The unification used in each resolution step is noted.

Now we will show an example of using the resolution method to prove the validity of a sentence.

Example 1.5.6. Let $T = \{\neg P(x,x), P(x,y) \to P(y,x), P(x,y) \wedge P(y,z) \to P(x,z)\}$ and let φ be the sentence $(\exists x)\neg P(x, f(x))$. We want to show that $T \models \varphi$. The theory $T \cup \{\neg\varphi\}$ is equisatisfiable (in this case even equivalent) to the following CNF formula:

$$S = \{\{\neg P(x,x)\}, \{\neg P(x,y), P(y,x)\}, \{\neg P(x,y), \neg P(y,z), P(x,z)\}, \{P(x, f(x))\}\}$$

We show that $S \vdash_R \square$. The resolution proof is, for example, the following sequence:

$$\{\neg P(x,y), \neg P(y,z), P(x,z)\}, \{P(x', f(x'))\}, \{\neg P(f(x), z), P(x,z)\}, \{\neg P(x,y), P(y,x)\},$$
$$\{P(x', f(x'))\}, \{P(f(x'), x')\}, \{P(x,x)\}, \{P(x', x')\}, \square$$

However, a resolution tree, as shown in Figure **??**, is more illustrative.

1.6 Soundness and Completeness

In this section, we will prove that the resolution method is both sound and complete in predicate logic.

1.6.1 Soundness Theorem

We begin with the proof of the soundness of the resolution rule. The principle is the same as the analogous observation in propositional logic. The proof is a bit more technical:

Proposition 1.6.1 (Soundness of the Resolution Step)**.** *Given clauses C_1, C_2, and let C be their resolvent. If clauses C_1 and C_2 hold in some structure $\mathcal{A}$, then C also holds in it.*

Proof. From the definition of the resolution rule, we know that the clauses and their resolvent can be expressed as $C_1 = C_1' \sqcup \{A_1, \ldots, A_n\}$, $C_2 = C_2' \sqcup \{\neg B_1, \ldots, \neg B_m\}$, and $C = C_1'\sigma \cup C_2'\sigma$, where σ is the most general unification of the set of expressions $S = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$, i.e., $S\sigma = \{A_1\sigma\}$.

Since clauses C_1 and C_2 are open formulas valid in $\mathcal{A}$, their instances after substitution σ are also valid in $\mathcal{A}$, i.e., we have $\mathcal{A} \models C_1\sigma$ and $\mathcal{A} \models C_2\sigma$. We also know that $C_1\sigma = C_1'\sigma \cup \{A_1\sigma\}$ and similarly $C_2\sigma = C_2'\sigma \cup \{\neg A_1\sigma\}$.

Our goal is to show that $\mathcal{A} \models C[e]$ for any variable assignment e. If $\mathcal{A} \models A_1\sigma[e]$, then $\mathcal{A} \not\models \neg A_1\sigma[e]$ and it must be that $\mathcal{A} \models C_2'\sigma[e]$. Therefore, $\mathcal{A} \models C[e]$. Conversely, if $\mathcal{A} \not\models A_1\sigma[e]$, then $\mathcal{A} \models C_1'\sigma[e]$, and again $\mathcal{A} \models C[e]$. □

The statement and proof of the Soundness Theorem are now the same as in propositional logic:

Theorem 1.6.2 (Soundness of Resolution)**.** *If a CNF formula S is refutable by resolution, then it is unsatisfiable.*

Proof. We know that $S \vdash_R \square$, so let's take some resolution proof of $\square$ from S. If there existed a model $\mathcal{A} \models S$, due to the soundness of the resolution rule, we could prove by induction on the length of the proof that $\mathcal{A} \models \square$, which is impossible. □

1.6.2 Completeness Theorem

The completeness theorem for resolution in predicate logic, i.e., that unsatisfiable formulas can be refuted by resolution, will be proved by reducing to the case of propositional logic. We will show that a resolution proof 'at the propositional logic level' can be 'lifted' to the predicate logic level.

The key is the following lemma, which ensures such 'lifting' in one resolution step. Its proof is somewhat technical.

Lemma 1.6.3 (Lifting Lemma)**.** *Given clauses C_1 and C_2 with disjoint sets of variables. If C_1^* and C_2^* are ground instances of clauses C_1 and C_2, and C^* is a resolvent of C_1^* and C_2^*, then there exists a resolvent C of clauses C_1 and C_2 such that C^* is a ground instance of C.*

Proof. Let $C_1^* = C_1\tau_1$ and $C_2^* = C_2\tau_2$, where τ_1 and τ_2 are ground substitutions that share no variables. We find a resolvent C such that $C^* = C\tau_1\tau_2$.

Let C^* be a resolvent of C_1^* and C_2^* over the literal $P(t_1, \ldots, t_k)$. We know that the clauses C_1 and C_2 can be expressed as $C_1 = C_1' \sqcup \{A_1, \ldots, A_n\}$ and $C_2 = C_2' \sqcup \{\neg B_1, \ldots, \neg B_m\}$, where $\{A_1, \ldots, A_n\}\tau_1 = \{P(t_1, \ldots, t_k)\}$ and $\{\neg B_1, \ldots, \neg B_m\}\tau_2 = \{\neg P(t_1, \ldots, t_k)\}$.

This means that $(\tau_1\tau_2)$ unifies the set of expressions $S = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$. Now, take the most general unification σ for S obtained using the Unification Algorithm. As C, choose the resolvent $C = C_1'\sigma \cup C_2'\sigma$.

It remains to show that $C^* = C\tau_1\tau_2$. Using the 'moreover' property from Proposition **??** on the correctness of the Unification Algorithm, we know that $(\tau_1\tau_2) = \sigma(\tau_1\tau_2)$, which we use in the third equality of the following calculation. In the fourth equality, we use the fact that $C_1'\tau_1\tau_2 = C_1'\tau_1$, and $C_2'\tau_1 = C_2'$, which follows from the fact that these are ground

substitutions that share no variables, and that $C_1'\tau_1$ and $C_2'\tau_2$ are ground instances:

$$\begin{aligned} C\tau_1\tau_2 &= (C_1'\sigma \cup C_2'\sigma)\tau_1\tau_2 \\ &= C_1'\sigma\tau_1\tau_2 \cup C_2'\sigma\tau_1\tau_2 \\ &= C_1'\tau_1\tau_2 \cup C_2'\tau_1\tau_2 \\ &= C_1'\tau_1 \cup C_2'\tau_2 \\ &= (C_1 \setminus \{A_1, \ldots, A_n\})\tau_1 \cup (C_2 \setminus \{\neg B_1, \ldots, B_m\})\tau_2 \\ &= (C_1^* \setminus \{P(t_1, \ldots, t_k)\}) \cup (C_2^* \setminus \{\neg P(t_1, \ldots, t_k)\}) = C^* \end{aligned}$$

□

By induction on the length of the resolution proof, we easily obtain the following corollary:

Corollary 1.6.4. *Given a CNF formula S, let S^* denote the set of all its ground instances. If $S^* \vdash_R C^*$ ('at the propositional logic level') for some ground clause C^*, then there exists a clause C and a ground substitution σ such that $C^* = C\sigma$ and $S \vdash_R C$ ('at the predicate logic level').*

Now it is easy to prove completeness:

Theorem 1.6.5 (Completeness of Resolution)**.** *If a CNF formula S is unsatisfiable, then it is refutable by resolution.*

Proof. Let S^* denote the set of all ground instances of the clauses from S. Since S is unsatisfiable, by Herbrand's theorem (specifically Corollary **??**), S^* is also unsatisfiable. From the completeness theorem of *propositional* resolution, we know that $S^* \vdash_R \square$ ('at the propositional logic level'). From the Lifting Lemma (or rather from Corollary **??**), we get a clause C and a ground substitution σ such that $C\sigma = \square$ and $S \vdash_R C$ ('at the predicate logic level'). But since the empty clause $\square$ is an instance of C, C must be $\square$. Thus, we have found a resolution refutation $S \vdash_R \square$. □

1.7 LI-resolution

In this section, we recall the concepts of *linear and linear-input proofs*, *LI-resolution*, and its completeness for Horn formulas. The definitions and theorems are the same as in propositional logic (the only difference is that in proofs we can use *variants* of clauses from S), and the proof can be done by reduction to propositional logic again using Herbrand's theorem and the Lifting Lemma.

Definition 1.7.1 (Linear and LI Proof)**.** A *linear proof* (by resolution) of a clause C from a formula S is a finite sequence

$$\begin{bmatrix} C_0 \\ B_0 \end{bmatrix}, \begin{bmatrix} C_1 \\ B_1 \end{bmatrix}, \ldots, \begin{bmatrix} C_n \\ B_n \end{bmatrix}, C_{n+1}$$

where C_i are called *central* clauses, C_0 is the *initial* clause, $C_{n+1} = C$ is the *final* clause, B_i are *side* clauses, and the following hold:

- C_0 is a variant of a clause from S, for $i \leq n$ C_{i+1} is the resolvent of C_i and B_i,

- B_0 is a variant of a clause from S, for $i \leq n$ B_i is a variant of a clause from S or $B_i = C_j$ for some $j < i$.

A *linear refutation* of S is a linear proof of $\square$ from S.

An *LI-proof* is a linear proof in which each side clause B_i is a variant of a clause from S. If there is an LI-proof, we say that C is *LI-provable* from S, and write $S \vdash_{LI} C$. If $S \vdash_{LI} \square$, then S is *LI-refutable*.

In Remark **??**, we noted that 'linear' resolution (based on linear proofs) is complete. The proof was left as an exercise. The same statement holds for predicate resolution:

Theorem 1.7.2 (Completeness of Linear Resolution)**.** *A clause C has a linear proof from a CNF formula S if and only if it has a resolution proof from S (i.e., $S \vdash_R C$).*

Proof. From a linear proof, we can easily construct a resolution tree. The converse implication follows from Remark **??** and the Lifting Lemma (whose application preserves the linearity of the resolution proof). $\square$

1.7.1 Completeness of LI-resolution for Horn Formulas

Recall the terminology related to Hornness and programs: A *Horn clause* is a clause containing at most one positive literal. A *Horn formula* is a (finite or infinite) set of Horn clauses. A *fact* is a positive unit (Horn) clause, a *rule* is a (Horn) clause with exactly one positive and at least one negative literal, and a *goal* is a non-empty (Horn) clause with no positive literals. Rules and facts are called *program clauses*.

As in propositional logic, LI-resolution is complete for Horn formulas:

Theorem 1.7.3 (Completeness of LI-resolution for Horn Formulas)**.** *If a Horn formula T is satisfiable, and $T \cup \{G\}$ is unsatisfiable for a goal G, then $T \cup \{G\} \vdash_{LI} \square$, by an LI-refutation that starts with the goal G.*

Proof. It follows from the analogous theorem in propositional logic, from Herbrand's theorem, and from the Lifting Lemma. $\square$

1.7.2 Resolution in Prolog

Finally, we show the application of LI-resolution in the Prolog programming language. A *program* in Prolog is a Horn formula containing only *program clauses*, i.e., *rules* and *facts*.

Example 1.7.4. As an example, consider a simple program describing the family relationships of three individuals, described in Table **??**. On the left side, we see the Prolog syntax, and on the right is the set notation of the corresponding clauses; the respective CNF formula will be denoted P.

The last line in the table is not part of the program; it is an *existential query*. We are interested in whether it holds in the Program P: $P \models (\exists X) son(charlie, X)$? Note that by negating the query, we obtain the *goal* $G = \{\neg son(charlie, X)\}$. Therefore, we want to *refute* the CNF formula $P \cup \{G\}$.

As in propositional logic (Corollary **??**), the following simple corollary of the completeness of LI-resolution for Horn formulas holds.

`son(X,Y):-father(Y,X),man(X).`	$\{son(X,Y), \neg father(Y,X), \neg man(X)\}$
`son(X,Y):-mother(Y,X),man(X).`	$\{son(X,Y), \neg mother(Y,X), \neg man(X)\}$
`man(charlie).`	$\{man(charlie)\}$
`father(bob,charlie).`	$\{father(bob,charlie)\}$
`mother(alice,charlie).`	$\{mother(alice,charlie)\}$
`?-son(charlie,X).`	$\{\neg son(charlie,X)\}$

Table 1.1: Example Prolog Program

Corollary 1.7.5. *For a program P and a goal $G = \{\neg A_1, \ldots, \neg A_k\}$ in variables $X_1, \ldots, X_n$, the following conditions are equivalent:*

- $P \models (\exists X_1) \ldots (\exists X_n)(A_1 \wedge \cdots \wedge A_k)$
- *$P \cup \{G\}$ has an LI-refutation starting with the goal G.*

Proof. It is not hard to see that the program P is always a satisfiable Horn formula. The first condition is equivalent to the unsatisfiability of $P \cup \{G\}$. The equivalence then follows from the completeness of LI-resolution for Horn formulas (Theorem **??**). □

If the answer to the query is positive, we want to know the *output substitution* σ, i.e., the composition of unifications from individual resolution steps, restricted to the variables in G. It holds that:

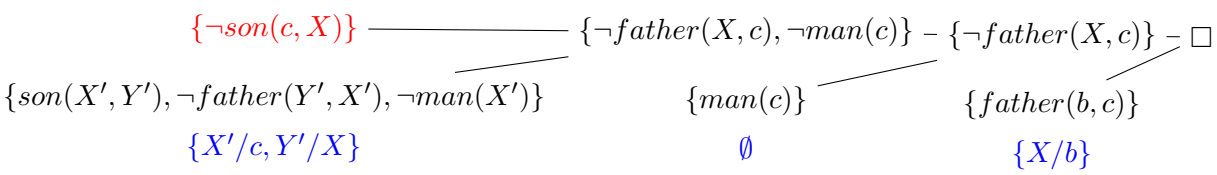
$$P \models (A_1 \wedge \cdots \wedge A_k)\sigma$$

Example 1.7.6. Continuing Example **??**, we find all output substitutions for our query:

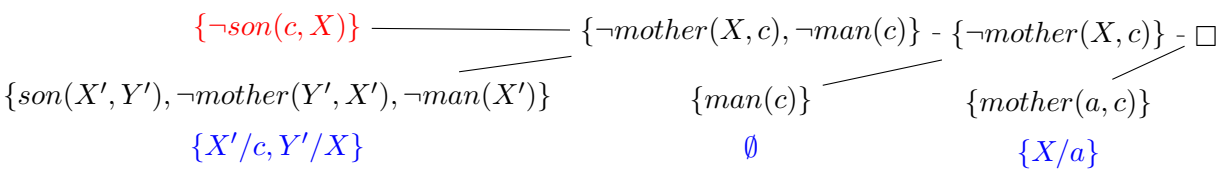
```
?-son(charlie,X).
X = bob ;
X = alice ;
No
```

It depends on which of the two rules we apply to the goal. The respective refutations are shown below. The output substitution is obtained by composing the substitutions from individual steps and restricting it to the variable X. (For lack of space, we have abbreviated the constant symbols to a, b, c.)

(a) Output substitution $\sigma = \{X/b\}$:



(b) Output substitution $\sigma = \{X/a\}$:



Bibliography