Logic for Computer Science

April 1, 2011

Chapter 1

Propositional Logic

The intent of this book is to familiarize the computer science students with the concepts and the methods of logic. Logic studies reasoning, i.e. the rules of inferring new statements from an existing set of statements. We use it in our daily routine, many times without being aware of it. We use logic when we make plans, set goals, and solve problems. If John wants a college degree, then he must go to college. If he plans to get married, he must find a wife. If Mary has financial problems, then she must reduce the expenses, get extra money, or both. In the preceding 3 sentences, the second clause is inferred from the first.

This book deals with symbolic logic, also called mathematical or formal logic, that formalizes reasoning by using mathematical concepts. The advantage of this approach is increased precision; the disadvantage is that it requires some knowledge of mathematics. At this point, the skeptics may wonder about the benefits of studying symbolic logic or any logic for that matter. They may say "I can reason without studying logic, so why should I learn it?", or more to the point "Is this book going to make me a better programmer?". We answer the first question with "We hope that by studying it, you will sharpen your reasoning. That is not to imply that your reasoning is faulty, but that you will become more precise and efficient in making assertions and in drawing conclusions."

The answer to the second question is more complex. First of all, programmers must write *correct* programs. But how do we define correctness? In most cases, correctness means that the program satisfies a given set of *requirements*. These requirements are specified as a set of statements. We can use logic, particularly symbolic logic, to write down these statements. This way we make the requirements more precise, and check them for redundancies and inconsistencies. Logic also provides a methodology, called *program verification*, for checking program correctness. A program is *correct* if it terminates and, when finished, it accomplishes the task it was expected to perform. Program verification guarantees correctness. Many of the algorithms given in this book are accompanied by correctness proofs.

Second, many large programs include a reasoning component. Expert sys-

tems that diagnose diseases, find mineral deposits, or analyze newspaper articles must draw conclusions from the input data. Reasoning components also occur in many game and problem solving programs. The symbolic logic is needed both for designing and understanding such programs.

Logic programming is a growing field of computer science and its purpose is to design and implement systems that process logical assertions. The best known of these systems is *prolog*, one of the main languages of artificial intelligence. We provide a prolog description in Chapter 3. Logic programs tell the computer *what* to do rather than *how* to do it. So, a logic program provides a description of the task rather than an algorithm for doing it.

Logic is also useful in understanding and designing query languages for databases, particularly for relational databases. In fact, prolog is frequently used for databases.

Many persons associate mathematics with long, boring and tedious proofs. For this reason, computer science tries to automate proofs. Just like the computers freed the man from the boring task of performing arithmetic operations, the goal of *automatic theorem proving* is to have the computers do the proofs. For this field of computer science, symbolic logic is a must.

We hope that we convinced the skeptic to keep reading. Now, we must point out that humans have several methods of reasoning. One method is by *analogy*. By analogy, the sentence *If I study hard, I will pass the exam* is a consequence of the statement *John studied hard; he passed the exam*. Our purpose is not to describe how humans think, but to present a formal method of reasoning. This method is used in mathematics and sciences; it is rarely used in everyday life. Mark may say "I will marry Becky next summer." . He will not do a formal reasoning unless he is wrong. Then he will check which ones of his assumptions and inference rules did not work. He presumes that both he and Becky will be alive next summer, be willing to marry each other, and the marriage requirements will be satisfied. He also assumes that there will be a place where they can get married and an authorized person willing to do it. All these hidden assumptions are ignored in everyday reasoning unless the conclusion is false.

This book presents the classical boolean logic. It is called boolean because each sentence has only 2 truth values, true or false. Moreover, every sentence must be either true or false, a law known in logic as *the excluded middle*. There are formal logics that remove these restrictions, but they are more complicated and harder to use. In any case, after mastering this book, the reader should have no problem learning them. We recommend Nerode and Shore for modal and intuitionistic logic.

The book presents propositional logic, first order logic, and prolog. First order logic is a generalization of propositional logic, and prolog is a computer language that uses first order logic.

This chapter presents the propositional logic, the simplest of the logics described in the book. Many of the concepts and the methods introduced here are used in other logics, no matter how complex. The basic entity of these logics is *the formula*. We use the term formula instead of statement or sentence, because we define it in precise, mathematical way. Every logic, no matter how complex,

has a syntax that defines the formulas, and a semantics that assigns meanings to the formulas. In all logics we are interested in finding methods that determine the meanings of formulas.

The chapter has 10 sections, each one dealing with a different topic from propositional logic. The first section presents *the syntax* of the formulas. Section 2 formalizes *the structural induction*, a method for verifying formula properties. The next section introduces the notion of *truth assignment*. The truth assignment gives meaning to the formulas. Since there are many truth assignments, the formulas may have different meanings. In Section 4 we identify formulas that are always true, sometimes true and sometimes false, and always false. The next section introduces the notion of *semantic equivalence*. Two formulas are semantically equivalent if they have the same meaning. The section shows an algorithm for determining if two formulas are semantically equivalent and presents two applications. The remaining sections serves to study *the resolution method*. This procedure is used to determine if a formula is always false, i.e. every truth assignment gives it the value false. The method is used by many automatic theorem provers, like ITP (). Section 6 deals with the properties of two special classes of formulas, called *conjunctions* and *disjunctions*. In Section 7 we define a set of formulas, formed from disjunctions and conjunctions, called *conjunctive normal forms*. The section shows that every formula has an equivalent conjunctive normal form and gives an algorithm for finding it. The next section introduces the notion of resolution. This method tells us whether a conjunctive normal form is *unsatisfiable* or not. Section 9 discusses the computational complexity of the resolution method and presents several improvements of the method. The last section defines the SLD resolution, that will be used in the chapter on prolog. It also shows that this resolution is complete for Horn clauses.

1.1 Formulas

This section presents the *syntax* of the propositional logic. It starts with the fundamental principle of any logic, the separation of *the object language* from the *meta-language*. Next, we use the analogy with English to give a feeling for the notions of syntax and semantics. Then, we present the alphabet of the language and the syntax of the formulas. We follow it with an algorithm that recognizes formulas. Next, we introduce the notion of subformula, and present an algorithm that finds the subformulas of a formula. Later, we give the tree representations of formulas and algorithms that convert formulas to trees and trees to formulas.

It is useful to think of propositional logic (or any other logic for that matter) as a language. We call the language that we study *the object language*. We talk about the object language in another language, called *the meta-language*. For example, we may study the French grammar in English. In this case, the object language is French and the meta-language is English. Or, we may talk about the programming language Java in English. Here the object language is Java and

the meta-language English. So, the object language is the *observed language* and the meta-language is *the observer' s language*.

To avoid confusion, we require that the object language and the meta-language are *distinct*, i.e. at all times we know whether a string or a sentence belongs to the object language or to the meta-language. The task is easy when the two languages use different alphabets, but is poses a problem when we study English grammar in English. In this case we can use quotation marks, or a different script, to differentiate between the two languages. The journalists employ similar techniques to separate their statements from those of the persons they interview or cite. We make the distinction easier by using the mathematical script over a reduced alphabet for the object language. Most of the time, the meta-language uses a different script. If a meta-language string is written in the mathematical script, then it contains characters that are not in the object language, like brackets.

The propositional logic deals with the relations between the larger formulas and the smaller formulas that are their components. For a better understanding of the scope of this language, let us compare it to English, a spoken language. In English we use *sentences* to assert facts, ask questions, or express sentiments. A sentence is formed with *clauses*. The simple sentences are clauses, while the complex or compound sentences[1] have more than one clause.

The 3 sentences below are simple because they are clauses.

- Bill has false teeth.
- Mark is bald.
- Mike curses.

A clause cannot be broken down into smaller units without loosing its meaning. The complex sentences are formed by joining several clauses. The three statements below are complex or compound.

- Jay likes money and Mary likes Jay.
- If Mike works hard, Mike's wife is happy.
- Susan is smart or Bob is a monkey's uncle.

The first sentence is obtained by joining the clauses *Jay likes money* and *Mary likes Jay* with the conjunction *and*. The compound sentence, *If Mike works hard, Mike's wife is happy*, is formed by joining the simple sentences *Mike works hard* and *Mike's wife is happy* with the conjunction *if*. We add a comma after *hard* to separate the two simple sentences. We get the last complex sentence by joining the clauses *Susan is smart* and *Bob is a monkey's uncle* with the conjunction *or*.

Propositional logic studies sentences. It is not concerned with the grammatical structure of the clauses and regards them as symbols over a simple alphabet.

[1] The linguists distinguish them according to the number of independent clauses.

It investigates how we build sentences from clauses and how we determine the meaning of a sentence from the meaning of its component clauses.

In this section we discuss the *syntax* of the language. Its *semantics* is presented in the third section, and it is important to distinguish the two notions. The syntax consists of grammar rules. These rules operate on strings and are not concerned with the meaning of the strings. Syntactically, *cat* is a 3 letter string. It is the English semantics that associates with *cat* the small mammal that eats mice.

In English there are many syntactic rules for forming complex sentences from simpler sentences. One such rule is the *and* rule. It joins two sentences by inserting the word *and* between them. So, the sentence *Bill has false teeth and Mark is bald* is obtained by applying the *and* rule to the sentences *Bill has false teeth* and *Mark is bald*.

In propositional logic we call sentences **formulas**, because their syntax and semantics are defined by rigorous, unambiguous rules. This makes the object language precise and easy to manage.

Definition 1.1.1 (the alphabet) *The alphabet of the propositional language consists of:*
1. An infinite set of symbols $P_0, P_1, \ldots, P_n, \ldots$. *We call these symbols,* atomic formulas

2. The symbols $\neg$, $\vee$, $\wedge$, $\longrightarrow$, *and* $\longleftrightarrow$. *These characters are called* connectives*;* $\neg$ *is the* negation *sign,* $\vee$ *the* or *sign,* $\wedge$ *the* and *sign,* $\longrightarrow$ *the* if *sign, and* $\longleftrightarrow$ *the* if and only if *sign.*

3. The left and the right parentheses, (*and*).

Some people might object to infinite alphabets. After all, all spoken languages use only a finite number of symbols. We can easily overcome this objection by representing the symbols P_i by the letter P followed by the decimal representation of i. So, each formula is written as a string using the symbols $P, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$. Then the alphabet of the language is finite. However, for the purpose of our study, it is convenient to look at these strings as symbols.

In the meta-language we will try to avoid the use of the round parentheses (and). Instead, we shall use the square brackets [and] and the curly brackets { and }.

Now, that we have the alphabet, we can start building *formulas*. Just like the English sentences are formed from clauses by means of connectives (conjunctions, relative pronouns, conjunctive adverbs), the formulas are formed from *atomic formulas* and connectives.

Definition 1.1.2 (formulas) *1. The atomic formulas are formulas.*

2. If F *is a formula, so is* $\neg F$*;* $\neg F$ *is called the* negation *of* F.

3. If F *and* G *are formulas then* $(F \vee G)$ *is a formula, called* F or G.

4. If F *and* G *are formulas then* $(F \wedge G)$ *is a formula, called* F and G.

5. If F *and* G *are formulas then* $(F \longrightarrow G)$ *is a formula, called* if F then G.

6. If F and G are formulas then $(F \longleftrightarrow G)$ is a formula, called F if and only if *G.*

The formulas are the *strings of the object language* that are produced by the above grammar. These rules are a *generative grammar* because they tell us how to produce complex formulas from simpler formulas. The parentheses tell us how to *parse* the formula, i.e. how to break it down into atomic formulas. For example, the formula $(P_1 \longrightarrow \neg P_7)$ is obtained by applying rule 5 of Definition 1.1.2 to the formulas P_1 and $\neg P_7$. We will prove, in the next section, that the parsing can be done in one and only one way. So, we do not have the ambiguity of parsing the string *big man eating tiger* where we do not know if the tiger is big, or just eats big men.

The strings generated by the rules 1-6 of Definition 1.1.2 have the following 3 properties:

1. If a string $\neg F$ is a formula then F is a formula.

2. If (S) is a formula then there are formulas F and G such that at least one of the equalities $S = F \wedge G$, $S = F \vee G$, $S = F \longrightarrow G$, $S = F \longleftrightarrow G$ holds. We abbreviate this sentence by saying that whenever (S) is formula, there are formulas F and G and connective C in $\{\wedge, \vee, \longrightarrow, \longleftrightarrow\}$ such that $S = FCG$. We remember that FCG is the string obtained by appending C at the end of the string F and then appending G at the end the resulting string.

3. Every formula F is either atomic, the negation of a formula, or has the form (FCG) where F and G are formulas and C is one of the connectives $\wedge$, $\vee$, $\longrightarrow$, $\longleftrightarrow$.

The symbols F and G are meta-language variables (meta-variables) that denote formulas in the object language. The meta-variable C stands for one of the connectives $\wedge$, $\vee$, $\longrightarrow$, $\longleftrightarrow$, and the meta-variable S is a string, that may or may not be a formula.

Since this book deals with formulas, it is important to determine whether a string is or is not a formula. So, let us now use Definition 1.1.2 to construct a formula recognition algorithm. The algorithm[2] keeps scanning the input string underlining more and more symbols until one of the following conditions is met: the whole string is underlined, the last scan produced no lines, or a collision occurred. A collision occurs when two underlined strings run into each other. We say that a character is *unmarked* if it is not underlined.

Step 1 of Figure 1.1 implements the first rule of the definition of formulas, while the body of the loop takes care of the (inductive) rules 2, 3, 4, 5, and 6. We have 4 stop with failure conditions at Steps 1,2, and 3. The first 3 are due to collisions and the last one occurs because S has unmarked symbols. In the next section we will prove the correctness of the algorithm. For now, let us run it for three strings, a formula and two non-formula.

[2]There are faster algorithms, but they require formal language knowledge, and we do not cover this subject.

```
Step 1. while [scanning S from left to right] do
        if [the scanned symbol is an atomic formula] then
              if [the preceding symbol is underlined] then
                    stop with failure;
              else
                    underline the atomic formula;
Step 2. while [ [S is not fully underlined] and [the previous scan produced new
            underlines]] do
        while [scanning S from left to right] do
        {
              if [the scanned symbol is an unmarked negation succeeded
                        by an underlined string] then
                    if [the symbol preceding ¬ is underlined] then
                          stop with failure;
                    else
                          underline ¬ ;
              if [the scanned symbol is an unmarked ( succeeded, in this order,
                   by an underlined string G, an unmarked binary connective C,
                        an underlined string H, and an unmarked ) ] then
                    if [the symbol that precedes ( or the one that succeeds ) is
                              underlined] then
                          stop with failure;
                    else
                          underline (, C, and );
        }
Step 3. if [S is fully underlined] then stop with success;
        else stop with failure;
```

Figure 1.1: The Formula Recognition Algorithm

Examples 1.1.3 1. Let $S = \neg(P_1 \wedge ((P_2 \longrightarrow P_3) \vee (P_4 \longleftrightarrow P_5)))$. At Step 1 we underline all atomic formulas in S and get the string below.

$\neg(\underline{P_1} \wedge ((\underline{P_2} \longrightarrow \underline{P_3}) \vee (\underline{P_4} \longleftrightarrow \underline{P_5})))$

After Step 1 we have new underlines and no collisions. Since S is not underlined, we execute the loop.

$\neg(\underline{P_1} \wedge \underline{((P_2 \longrightarrow P_3) \vee (P_4 \longleftrightarrow P_5))})$

Step 2 produced new underlines, no collisions, and the string S is not (fully) underlined, so we repeat the loop.

$\neg(\underline{P_1} \wedge \underline{((P_2 \longrightarrow P_3) \vee (P_4 \longleftrightarrow P_5))})$

Again, Step 2 underlined new symbols, caused no collisions, and S is not underlined. So, we execute the loop.

$\underline{\neg(P_1 \wedge ((P_2 \longrightarrow P_3) \vee (P_4 \longleftrightarrow P_5)))}$

Once more Step 2 underlined more symbols, had no collisions, and S is not underlined, so we repeat the loop.

$\underline{\neg(P_1 \wedge ((P_2 \longrightarrow P_3) \vee (P_4 \longleftrightarrow P_5)))}$

At this point the loop repetition test fails because S is fully underlined. We go to step 3. There, the algorithm accepts S since S is (fully) underlined.

2. Now let us trace the algorithm for the string $S = (P_2 \vee (P_3 \longrightarrow P_4)) \longleftrightarrow P_1)$.

We apply Step 1 and underline all atomic formulas.

$(\underline{P_2} \vee (\underline{P_3} \longrightarrow \underline{P_4})) \longleftrightarrow \underline{P_1})$

Step 1 produced new underlines and had no collisions, so we execute the body of the loop.

$(\underline{P_2} \vee \underline{(P_3 \longrightarrow P_4)}) \longleftrightarrow \underline{P_1})$

Step 2 underlined new symbols, caused no collisions, and S is not fully underlined, so we repeat the loop.

$\underline{(P_2 \vee (P_3 \longrightarrow P_4))} \longleftrightarrow \underline{P_1})$

Again, Step 2 produced new underlines, had no collisions, and S is not fully underlined. Hence, we execute the loop.

$\underline{(P_2 \vee (P_3 \longrightarrow P_4))} \longleftrightarrow \underline{P_1})$

Now Step 2 produced no collisions and no underlines. So, we go to Step 3. Since S is not fully underlined, we reject the string.

3. Let the input string be $S = P_2P_1$. At Step 1, we underline P_2 and then we scan P_1. Here we have a collision, so we reject the string.

Definition 1.1.4 (subformula) *A subformula of F is a substring of F that is also a formula.*

For example, the subformulas of $((\neg P_4 \vee P_2) \longleftrightarrow (P_3 \wedge P_6))$ are:

P_4, P_2, P_3, P_6, $\neg P_4$, $(P_3 \wedge P_6)$, $(\neg P_4 \vee P_2)$, and $((\neg P_4 \vee P_2) \longleftrightarrow (P_3 \wedge P_6))$.

Observation 1.1.5 *1. F is a subformula of itself.*

2. A subformula can occur multiple times in the formula. For example, P_2 occurs twice in the formula $(P_2 \vee \neg P_2)$, at position (index) 1 and at index 4.

Now let us modify the algorithm from Figure 1.1 to generate a list of all substrings of F that are formulas. We assume that F is a formula, so we delete the collision and blank symbol tests. Since the list $Inst$ contains all occurrences of subformulas, it may have duplications.

Step 1. Initialize $Inst$ to empty; scan S from left to right, underline each atomic formula, and add it at the end of $Inst$;

Step 2. while [F is not fully underlined] do

{

while [scanning F] do

{

if [$\neg G$ is a substring of F, G is underlined, and $\neg$ is not] then

{ underline $\neg$; add $\neg G$ at the end of $Inst$; }

if [(GCH) is a substring of F, G and H are underlined,

$C \in \{\vee, \wedge, \longrightarrow, \longleftrightarrow\}$, and (, C,) are unmarked] then

{ underline (, C,); add (GCH) at the end of $Inst$; }

}

}

Figure 1.2: The algorithm for listing subformulas

Example 1.1.6 Let us apply this algorithm to find the subformulas of $F = ((\neg P_1 \longrightarrow (\neg P_3 \vee P_4)) \wedge (\neg P_4 \longleftrightarrow P_1))$.

Initially $Inst = []$. We execute Step 1 and get the underlines

$((\neg \underline{P_1} \longrightarrow (\neg \underline{P_3} \vee \underline{P_4})) \wedge (\neg \underline{P_4} \longleftrightarrow \underline{P_1}))$

and $Inst = [P_1, P_3, P_4, P_4, P_1]$. We listed P_4 and P_1 twice because they occur twice in F. Since F is not underlined we execute the body of the while loop and get the markings

$((\underline{\neg P_1} \longrightarrow (\underline{\neg P_3} \vee \underline{P_4})) \wedge (\underline{\neg P_4} \longleftrightarrow \underline{P_1}))$

and $Inst = [P_1, P_3, P_4, P_4, P_1, \neg P_1, \neg P_3, \neg P_4]$. We pass the repetition test because F is not underlined. We repeat the loop body and obtain the string

$((\underline{\neg P_1} \longrightarrow \underline{(\neg P_3 \vee P_4)}) \wedge \underline{(\neg P_4 \longleftrightarrow P_1)})$

and $Inst = [P_1, P_3, P_4, P_4, P_1, \neg P_1, \neg P_3, \neg P_4, (\neg P_3 \vee P_4), (\neg P_4 \longleftrightarrow P_1)]$. We go to Step 2 again and check if F is underlined. It is not, so we repeat the loop and have the markings

$\underline{((\neg P_1 \longrightarrow (\neg P_3 \vee P_4)) \wedge (\neg P_4 \longleftrightarrow P_1))}$

and $Inst = [P_1, P_3, P_4, P_4, P_1, \neg P_1, \neg P_3, \neg P_4, (\neg P_3 \vee P_4), (\neg P_4 \longleftrightarrow P_1), (\neg P_1 \longrightarrow (\neg P_3 \vee P_4))]$. The repetition test is true again, so we execute the loop and get the underlines

$\underline{((\neg P_1 \longrightarrow (\neg P_3 \vee P_4)) \wedge (\neg P_4 \longleftrightarrow P_1))}$

and $Inst = [P_1, P_3, P_4, P_4, P_1, \neg P_1, \neg P_3, \neg P_4, (\neg P_3 \vee P_4), (\neg P_4 \longleftrightarrow P_1), (\neg P_1 \longrightarrow (\neg P_3 \vee P_4)), ((\neg P_1 \longrightarrow (\neg P_3 \vee P_4)) \wedge (\neg P_4 \longleftrightarrow P_1))]$.

Now we fail the repetition test because F is fully underlined.

The list of subformulas of F is $Inst = [P_1, P_3, P_4, P_4, P_1, \neg P_1, \neg P_3, \neg P_4, (\neg P_3 \vee P_4), (\neg P_4 \longleftrightarrow P_1), (\neg P_1 \longrightarrow (\neg P_3 \vee P_4)), ((\neg P_1 \longrightarrow (\neg P_3 \vee P_4)) \wedge (\neg P_4 \longleftrightarrow P_1))]$.

We can find the *set* of subformulas Set of F by deleting the duplicate elements from $Inst$. We get $Set = \{P_1, P_3, P_4, \neg P_1, \neg P_3, \neg P_4, (\neg P_3 \vee P_4), (\neg P_4 \longleftrightarrow P_1), (\neg P_1 \longrightarrow (\neg P_3 \vee P_4)), ((\neg P_1 \longrightarrow (\neg P_3 \vee P_4)) \wedge (\neg P_4 \longleftrightarrow P_1))\}$.

Sometimes it is more convenient to represent formulas as trees.

Step 1. Scan F and underline all atomic formulas of F; For each underlined atom create a leaf labeled with that atom;
Step 2. while [F is not fully underlined] do
{
while [scanning F] do
{
if [$\neg G$ is a substring of F, G is underlined, and $\neg$ is not] then
{ underline $\neg$; create a new node labeled $\neg$; Make $\neg$
the root of the tree with left child G; }
if [(GCH) is a substring of F having G, H underlined,
$C \in \{\vee, \wedge, \longrightarrow, \longleftrightarrow\}$, and (, C,) are unmarked] then
{ underline (GCH); Create a new node labeled C ; Make
the new node the root of a tree with left child G
and right child H;}
}
}

Figure 1.3: The Formula-to-Tree Algorithm

Figure 1.4: The leaves of the tree from Example 1.1.8

Definition 1.1.7 (formula tree) *A formula tree is a labeled binary tree where*
1. the leaves are labeled with atomic formulas, and
2. the branches are labeled with the connectives $\neg$, $\vee$, $\wedge$, $\vee$. If the label is $\neg$ the tree has one child; otherwise it has two.
The set of the labels of the leaves is called the support of the tree.

We modify the algorithm from Figure 1.2 to transform a formula into a tree.

Example 1.1.8 Let us apply the algorithm from Figure 1.3 to the formula $F = ((P_2 \longrightarrow \neg P_3) \vee \neg P_6)$. At Step 1 we underline the atomic formulas and construct the leaves of the tree. We get the underlines $((\underline{P_2} \longrightarrow \neg \underline{P_3}) \vee \neg \underline{P_6})$ and the trees from Figure 1.4. At Step 2 we underline $\neg P_3$ and $\neg P_6$ and get the markings $((\underline{P_2} \longrightarrow \underline{\neg P_3}) \vee \underline{\neg P_6})$ and the trees from Figure 1.5. We repeat the loop body and get the underlines $(\underline{(P_2 \longrightarrow \neg P_3)} \vee \underline{\neg P_6})$ and the trees from Figure 1.6. The string is not completely underlined, so we repeat the loop. Now the whole string is underlined $\underline{((P_2 \longrightarrow \neg P_3) \vee \neg P_6)}$ and we have the tree from Figure 1.7.

We can also go from formula trees to strings by using the algorithm from Figure 1.8. The procedure *convert* is a recursive top-down algorithm that starts

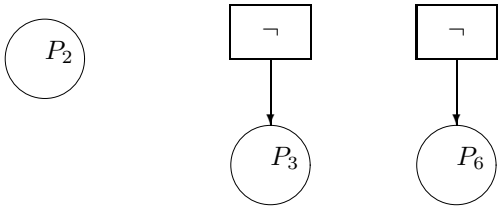


Figure 1.5: The trees from Example 1.1.8 after one run

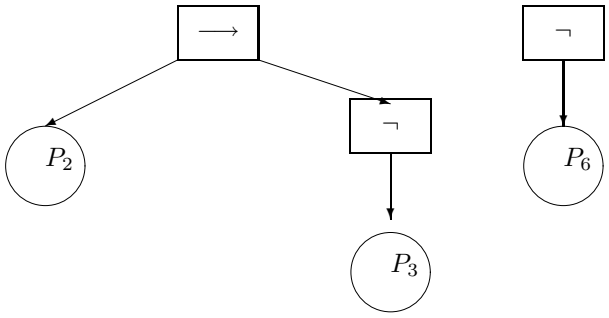


Figure 1.6: The trees from Example 1.1.8 after two runs

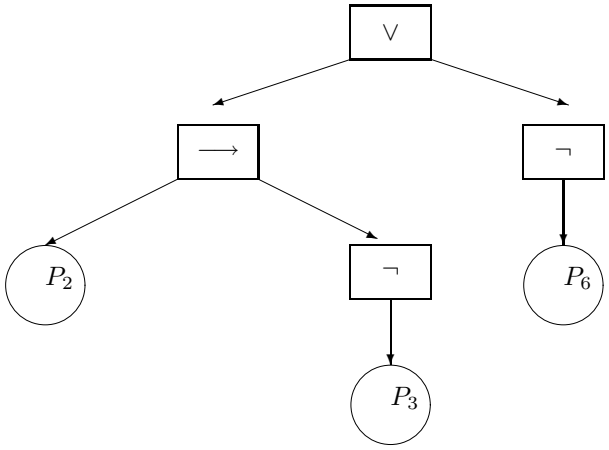


Figure 1.7: The formula tree for Example 1.1.8

```
procedure convert[Root]
{
      if [label[Root] is an atomic formula] then
            write[label[Root]];
       else if [label[Root] ∈ {∨, ∧, ⟶, ⟷}] then
       {
             write[(]; convert[Root.0]; write[label[Root]];
             convert[Root.1]; write[)];
       }
      else /* the label of the node is ¬ */
            { write[¬]; convert[Root.0]; }
 }
```

Figure 1.8: The tree to formula algorithm

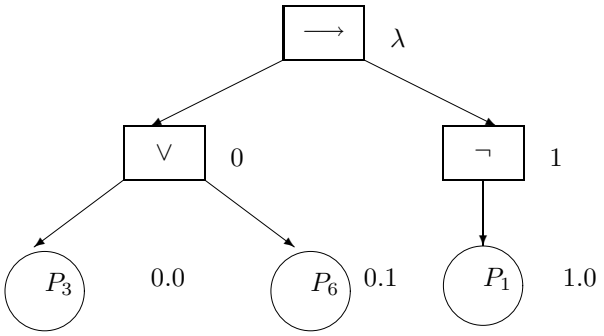


Figure 1.9: The input to Example 1.1.9

with Root, the address of the root. First, convert examines the label of Root and performs one of the following 3 actions:

If the label is an atomic formula, then it writes the label and returns.

If the label is one of the connectives $\vee$, $\wedge$, $\longrightarrow$, or $\longleftrightarrow$ then *convert* writes, in this order, a left parenthesis, converts the subtree Root.0, writes the label of Root, converts the subtree Root.1, and writes a right parenthesis.

If the label is $\neg$ then *convert* writes $\neg$ and processes the subtree Root.0.

Example 1.1.9 Let us apply this algorithm to the formula tree from Figure 1.9. The domain of the tree contains the addresses λ, 0, 1, 0.0, and 0.1. Now, let us trace the algorithm for Root=λ. The call convert[λ] produces the string

convert[λ] = (convert[0]$\longrightarrow$convert[1])

where convert[0] is the output of the first recursive call, $\longrightarrow$ is the label of λ, and convert[1] is the string produced by the second call.

The label of 0 is $\vee$, and its children are 0.0 and 0.1. So,

convert[0] = (convert[0.0]∨convert[0.1]).

The labels of 0.0 and 0.1 are P_3 and P_6, so convert[0.0] = P_3 and convert[0.1] =P_6. Then the above formula becomes

convert[0]= $(P_3 \vee P_6)$.

Now let's find convert[1]. The label of 1 is ¬ and its child is 1.0, so we get the relation

convert[1]=¬convert[1.0].

The label of 1.0 is the atomic formula P_1 so, convert[1.0]= P_1. We replace convert[1.0] in the above formula and get convert[1] = $\neg P_1$.

Now we substitute the values of convert[0] and convert[1] in the equation of convert[λ] and get

convert[λ] = $((P_3 \vee P_6) \longrightarrow \neg P_1)$.

In this book we talk about the object language strings called formulas. We talk about them in the meta-language. The meta-language must be rich enough to describe the entities from the object language. It must be able to characterize the set of formulas as well as many of its subsets. For this reason we need the notion of *meta-formula*.

Definition 1.1.10 (meta-formula) *1. The meta-variables A, B, F, G, H, I, J, K, L, Q, R, with or without subscripts, are formulas.*

2. The atomic formulas P_i are meta-formulas.

3. If U is a meta-formula, then ¬U is a meta-formula.

4. If U and V are meta-formulas, and C is one of the binary connectives $\{\vee, \wedge, \longrightarrow, \longleftrightarrow\}$, then (UCV) is a meta-formula.

Now let us compare the rules that define the meta-formulas with those of the formulas. Rules 1 and 2 of Definition 1.1.2 are the same as the rules 2 and 3 listed above. The remaining 4 rules of Definition 1.1.2 are covered by the rule 4 above. So, every string generated by Definition 1.1.2 is also produced by Definition 1.1.10. So, every formula is a meta-formula.

On the other side, not all meta-formulas are formulas. The strings $(F \vee G)$, $\neg(F \wedge P_1)$ are not formulas because the letter F is not in the object language.

The meta-formulas are useful in characterizing classes of formulas. We notice that the only difference between the formulas and the meta-formulas is that the meta-formulas contain meta-variables. If we replace all meta-variables of a meta-formula by formulas, then we obtain a formula. For example, let us replace F by $\neg P_3$ in the meta-formula $U = (F \longrightarrow P_1)$. We get the formula $(\neg P_3 \longrightarrow P_1)$. If we replace F by the formula $(P_{11} \wedge \neg(P_4 \longleftrightarrow P_7))$, U becomes $((P_{11} \wedge \neg(P_4 \longleftrightarrow P_7)) \longrightarrow P_1)$. We say that the formulas $(\neg P_3 \longrightarrow P_1)$ and $((P_{11} \wedge \neg(P_4 \longleftrightarrow P_7)) \longrightarrow P_1)$ are *instances* of the meta-formula U. The set of all instances of U is a subset of the set of formulas.

In the next sections we will be using the notations $F = \neg G$, $F = (G \vee H)$, $F = (G \wedge H)$, $F = (G \longrightarrow H)$, $F = (G \longleftrightarrow H)$, and in general F =some meta-formula. The meaning of these statements is that F is a formula belonging to the class defined by the meta-formula to the right of the equal sign.

Now, the attentive reader may ask in what language we wrote Definition 1.1.10. Recall that we describe the object language in the meta-language. This definition characterizes a meta-language concept, so it must be in the meta-meta-language. By the same reasoning the characters U and V of the definition must be meta-meta-variables, belonging to meta-meta-language. By analogy, we talk about the meta-meta-language in meta-meta-meta-language, and so on. This careful approach to logic was taken by A. Tarsky(). For our purpose, we will not differentiate among these languages; we will include all of them in the meta-language.

Now we will make several notational conventions that will simplify our presentation.

1. We will omit the outer parentheses of the expressions.

   So, we will write $(P_2 \vee P_3)$ as $P_2 \vee P_3$.

2. Let $F_0,\ldots, F_n$ be formulas and $m \geq 0$. We define $\bigwedge_{i=m}^{n} F_i$, called *big and from m to n*, as

$$\bigwedge_{i=m}^{n} F_i = \begin{cases} (P_0 \vee \neg P_0) & \text{if } m > n \\ F_m & \text{if } m = n \\ (\bigwedge_{i=m}^{n-1} F_i \wedge F_n) & \text{if } m < n \end{cases}$$

   For example, let us compute $\bigwedge_{i=1}^{3} F_i$.

   $\bigwedge_{i=1}^{3} F_i = \bigwedge_{i=1}^{2} F_i \wedge F_3$

   $= (\bigwedge_{i=1}^{1} F_i \wedge F_2) \wedge F_3$

   $= (F_1 \wedge F_2) \wedge F_3$.

   The formulas are always parenthesized to the left, i.e. all left parentheses occur before the first F.

3. Let $F_0,\ldots, F_n$ be formulas and m be a whole number. We define the formula $\bigvee_{i=m}^{n} F_i$, called *big or from m to n*, as

$$\bigvee_{i=m}^{n} F_i = \begin{cases} (P_0 \wedge \neg P_0) & \text{if } m > n \\ F_m & \text{if } m = n \\ (\bigvee_{i=m}^{n-1} F_i \vee F_n) & \text{if } m < n \end{cases}$$

   For example, $\bigvee_{i=3}^{4} F_i = \bigvee_{i=3}^{3} F_i \vee F_4 = F_3 \vee F_4$. The big or formulas are also parenthesized to the left.

Exercises

Exercise 1.1.1 *Show that the set of formulas is countably infinite.*

Exercise 1.1.2 *Use the algorithm from Figure 1.1 to find out if the strings below are formulas in the object language.*

1. $((\neg P_1 \vee P_7) \longleftrightarrow \neg(P_6 \wedge \neg P_2))$
2. $(\neg P_6 \longrightarrow \neg P_5) \wedge P_9$
3. $(Q \wedge \neg P_2)$
4. $\neg\neg(P_{-1} \vee P_9)$
5. $(P_8 \vee (P_3 \longrightarrow \neg P_9))$
6. $(\neg P_9 \longleftrightarrow \neg P_8)\neg$
7. $(\neg P_6 \wedge (F \vee P_4))$
8. $(P_1 \vee P_9)) \wedge P_4)$
9. $(P_1 \vee \neg P_3)(P_1 \longrightarrow P_7)$

Exercise 1.1.3 *Use the algorithm from Figure 1.3 to draw the tree representations of the formulas below.*

1. $\neg((\neg P_1 \longrightarrow P_2) \wedge \neg P_3)$
2. $((P_1 \vee \neg P_3) \longleftrightarrow \neg\neg(P_2 \vee P_6))$
3. $((P_4 \longrightarrow (\neg P_8 \wedge P_7)) \longleftrightarrow (P_8 \wedge P_2))$.

Exercise 1.1.4 *Use the algorithm from Figure 1.2 to find the set of subformulas of the following formulas:*

1. $(\neg(P_1 \longrightarrow P_2) \wedge \neg(\neg P_3 \longleftrightarrow P_1))$
2. $\neg(\neg(P_4 \longleftrightarrow P_3) \wedge \neg\neg(P_4 \vee \neg P_1))$
3. $(P_5 \longleftrightarrow ((P_7 \vee \neg P_8) \wedge P_3))$

Exercise 1.1.5 *Find meta-formulas that characterize the following sets of formulas:*

1. *The set of all formulas.*
2. *The set of formulas whose trees have the root labeled* $\wedge$.
3. *The set of formulas whose trees have the root labeled* $\neg$ *and the node at address* 0 *labeled* $\longrightarrow$.

Exercise 1.1.6 *Define a meta-formula whose instances have at least 5 atoms.*

Exercise 1.1.7 *For each of the following lines, determine if the formula* F *is an instance of the meta-formula* U. *So, find the values of the meta-variables of* U *that make* $U = F$, *or show that this is not possible. For example,* $F = ((P_2 \longrightarrow P_1) \wedge \neg P_3)$ *belongs to* $U = (G \wedge H)$ *because for* $G = (P_2 \longrightarrow P_1)$ *and* $H = \neg P_3$, $U = F$. *At the same* $F = (P_2 \vee (P_1 \wedge P_5)$ *does not belong to* $U = (G \wedge H)$ *because the tree representation of* F *has the root labeled* $\vee$ *while the tree* U *has the root labeled* $\wedge$. *So, it does not matter what value we give to the meta-variables of* U, *the root will always be* $\wedge$.

1. $F = (P_1 \vee \neg P_2)$, $U = (P_1 \vee G)$
2. $F = (P_1 \wedge P_2)$, $U = (G \wedge H)$
3. $F = (\neg P_2 \longrightarrow P_3)$, $U = (G \vee H)$.

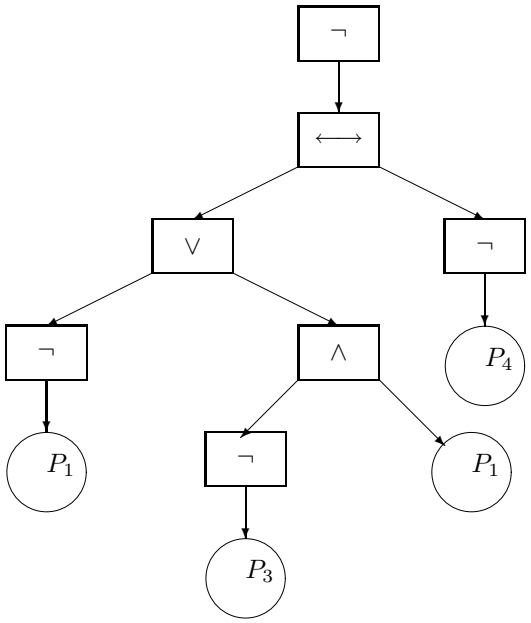


Figure 1.10: Tree for Exercise 1.1.9

Exercise 1.1.8 *Write the object language formulas that correspond to the following notations*

1. $\bigwedge_{i=1}^{5} P_i$
2. $\bigvee_{i=10}^{15} P_i$
3. $\bigwedge_{i=3}^{1} P_i$
4. $\bigvee_{i=1}^{1} P_i$
5. $\bigwedge_{i=10}^{10} P_i$
6. $\bigvee_{i=9}^{6} P_i$.

Exercise 1.1.9 *Use the algorithm from Figure 1.8 to find the formulas that correspond to the trees from Figures 1.10 and 1.11.*

1.2 Structural Induction

The main topic of this section is *the structural induction*, a formal method for proving statements about formulas. The structural induction is so widely used in logic and algebra that it is hard to imagine a rigorous presentation of the subject without using it. The method also goes by the name of induction on

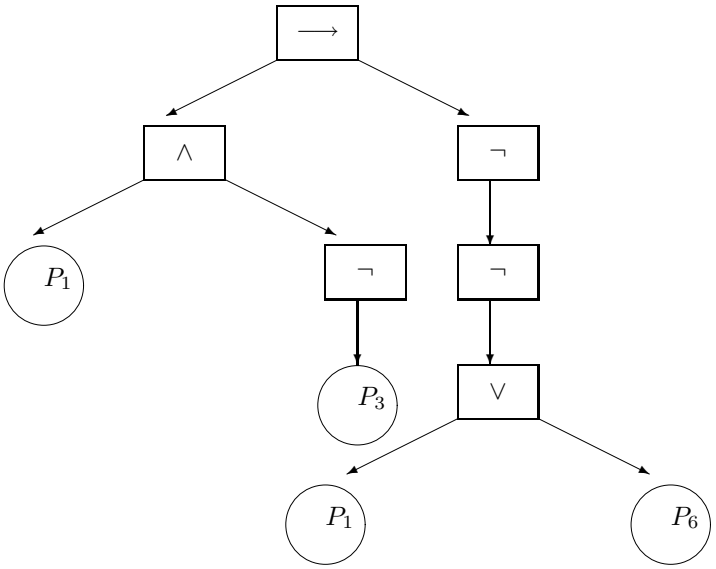


Figure 1.11: Tree for Exercise 1.1.9

1. $P[0]$ is true.
2. if $\mathcal{P}[n]$ is true then $\mathcal{P}[n+1]$ is also true.

Figure 1.12: The induction rules

Step 1. $\mathcal{P}[0]$ is true by rule 1.
Step 2. In rule 2 we make $n = 0$ and get the instance
if $\mathcal{P}[0]$ is true then $\mathcal{P}[1]$ is true.
Step 3. From Steps 1 and 2 we deduce that $\mathcal{P}[1]$ is true.
Step 4. In rule 2 we make $n = 1$ and we get the instance
if $\mathcal{P}[1]$ is true then $\mathcal{P}[2]$ is true.
Step 5. We deduce that $\mathcal{P}[2]$ is true from Steps 3 and 4.

Figure 1.13: A proof for $\mathcal{P}[2]$

formulas, induction on the complexity of the formulas, and induction on terms. As the names implies, it is an extension of the induction on natural numbers.

The section revisits the induction on the set of natural numbers N and then extends it to the set of formulas. Then it proves several lemmas that lead to The Unique Readability Theorem, which says that every non-atomic formula can be decomposed into smaller formulas in a unique way. The reader who is not interested in proof techniques can stop there, because the rest of the section proves many of the assertions made in the preceding section.

Let us remember the induction on the set of natural numbers, $\mathrm{N} = \{0, 1, 2, ...\}$. The induction is a formal method for verifying that a statement $\mathcal{P}$, called *calligraphic P*, is true on the set N. We write $\mathcal{P}[n]$ to emphasize that n is a meta-variable in the statement $\mathcal{P}$. The induction tells us that $\mathcal{P}$ is true for all n if and only if the meta-rules 1 and 2 of Figure 1.12 hold. The second line of Figure 1.12 is an implication. Its condition, $\mathcal{P}[n]$ is true, is called *the induction hypothesis*. It also contains the meta-variable n that denotes a natural number. But which one? The answer is that n can take any value from the set $N = \{0, 1, 2, ...\}$.

If we replace n by 0 we get the rule
2.0. if $\mathcal{P}[0]$ is true then $\mathcal{P}[1]$ is true.

If we replace n by 1 we obtain the rule
2.1 if $\mathcal{P}[1]$ is true then $\mathcal{P}[2]$ is true.

If we replace n by 1000 we get the rule
2.1000 if $\mathcal{P}[1000]$ is true then $\mathcal{P}[1001]$ is true.

So, meta-rule 2 is shorthand for an infinity of rules obtained by replacing n by the numbers 0,1,2, etc. When we replace n by 0 we get rule 2.0. When we replace n by 1 we get rule 2.1 and so on. The rules 2.0, 2.1, 2.1000, are called *instances* of meta-rule 2.

The two meta-rules of Figure 1.12 are sufficient to prove that $\mathcal{P}$ is true for any given number n. Figure 1.13 presents a proof that $\mathcal{P}[2]$ is true.

We can skip the details and present the proof as the sequence from Figure 1.14.

1. $\mathcal{P}[0]$ by meta-rule 1.
2. $\mathcal{P}[1]$, from line 1, by meta-rule 2.
3. $\mathcal{P}[2]$, from line 2, by meta-rule 2.

Figure 1.14: A shorter proof for $\mathcal{P}[2]$

Here we abbreviated $\mathcal{P}[n]$ is true to $\mathcal{P}[n]$. We can even skip the words *line* and *meta-rule* and write the proof as
1. $\mathcal{P}[0]$ by 1.
2. $\mathcal{P}[1]$, from 1, by 2.
3. $\mathcal{P}[2]$, from 2, by 2.

The induction theorem says that we need not bother with proofs like Figure 1.14.

Theorem 1.2.1 (induction on N) *If $\mathcal{P}$ satisfies conditions 1 and 2 of Figure 1.12 then $\mathcal{P}$ is true for any natural number n.*

Proof: We use the fact that N is *well ordered* by the relation $>$. This means that *every nonempty subset of N has a smallest number.* We use this property to get a contradiction.

So, assume that $\mathcal{P}$ satisfies conditions 1 and 2 of Figure 1.12 but $\mathcal{P}[n]$ is false for some n's. Then, by the well ordering of N, there is a *smallest* number m that makes $\mathcal{P}$ false. Since P satisfies condition 1, m cannot be 0. Then $m-1$ is a whole number. Now we ask if $\mathcal{P}[m-1]$ is true.

Case 1: $\mathcal{P}[m-1]$ is true. Then, by meta-rule 2, $\mathcal{P}[m]$ is true, contradicting the assumption that $\mathcal{P}[m]$ is false.

Case 2: $\mathcal{P}[m-1]$ is false. Then, the number $m-1$ is smaller than m and makes $\mathcal{P}$ false. This contradicts our assumption that m is the smallest number that makes $\mathcal{P}$ false.

In both cases we get a contradiction. So, our assumption that $\mathcal{P}$ is false for some natural numbers is false.**Q.E.D.**

Theorem 1.2.1 gives us a method for showing that $\mathcal{P}$ is true; it suggests that we prove that $\mathcal{P}$ satisfies the conditions listed in Figure 1.12.

Example 1.2.2 Let us show that for every natural number n, $\mathcal{P}[n] : n^2 \geq n$. First, we check that P satisfies condition 1. So, we replace n by 0, and get

$\mathcal{P}[0] : 0^2 \geq 0$.

We evaluate 0^2 and get $0 \geq 0$ which is true. So, $\mathcal{P}$ satisfies condition 1.
Now, let's see if P satisfies condition 2. The second condition is the implication

if $\mathcal{P}[n]$ then $\mathcal{P}[n+1]$.

So, we assume $\mathcal{P}[n]$, i.e.

(1) $n^2 \geq n$

and we need to prove

(2) $(n+1)^2 \geq (n+1)$.

Now we remember that

(3) $(n+1)^2 = n^2 + 2n + 1$.

We also know that
(4) $2n \geq 0$,
and
(5) $1 \geq 1$.
So, we add the inequalities (1), (4) and (5) and get
(6) $n^2 + 2n + 1 \geq n + 0 + 1$.
We use (3) to evaluate (6) and obtain
$(n+1)^2 \geq (n+1)$
which is exactly the formula (2) that we needed to prove.

Example 1.2.3 Let us recall that the **power set** of a set A is the set of all subsets of A. For example, the power set of the empty set ϕ is $\{\phi\}$ because the only subset of ϕ is ϕ. We write $|X|$ for the number of elements in the finite set X, and $P(X)$ for the power set of X. We will prove that whenever A has n elements, its power set has 2^n elements. So, our proposition $\mathcal{P}$ is
if $|A| = n$ then $|P(A)| = 2^n$.
Proof: We'll first check that $\mathcal{P}[0]$ is true.

If $|A| = 0$ then A= ϕ, since A has no elements. Then the set of subsets of ϕ is $\{\phi\}$, so $|P(A)| = 1$. But, $2^0 = 1$, so $|P(A)| = 2^0$.

Now let us assume that $\mathcal{P}$ is true for all sets with n-elements, i.e. if $|A| = n$, $|P(A)| = 2^n$. Let $B = \{b_1, b_2, ..., b_n, b_{n+1}\}$ be a set with $n+1$ elements. Now, let's look at the subsets of B. They fall into two disjoint categories.
Category 1: The subset does *not* contain b_{n+1}.

Then, the subset is also a subset of $A = \{b_1, ..., b_n\}$, a set with n elements. By the induction hypothesis A has 2^n subsets, so B has 2^n subsets that do not contain b_{n+1}.
Category 2: The subset contains the element b_{n+1}.

How many subsets do we have in this category? We can answer this question by noticing that every subset in this category is obtained by adding b_{n+1} to a subset of $A = \{b_1, ..., b_n\}$. For example, $\{b_{n+1}\}$ is obtained by adding b_{n+1} to ϕ, $\{b_1, b_{n+1}\}$ is obtained by adding b_{n+1} to $\{b_1\}$, and so on. Moreover, when added b_{n+1}, different subsets of A yield different subsets of B. So, this category has as many subsets as A. But we know from the induction hypothesis that A has 2^n subsets. So, $|P(B)| = |Category1| + |Category2| = 2^n + 2^n = 2 \cdot 2^n = 2^{n+1}$.**Q.E.D.**

Since the book deals with formulas, it is useful to have a set of rules, like the ones in Figure 1.12, for verifying assertions about formulas. The induction on natural numbers is not always satisfactory, since it is not clear how the integer is tied to the formula. We can do induction on the number of atomic formulas, on the number of connectives ($\neg$, $\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$), on the length of the formula, on the height of the tree representation of the formula, and so on. Of course, a math genius will (almost) always know how to attach numbers to formulas, but what about the rest of us?

Fortunately, there is a natural way to do it. First of all, let's look at the way we defined the natural numbers N. The sequence 0, 1, ..., n, ...starts with 0

1. 0 is in N.
2. if n is in N, then $n+1$ is in N.

Figure 1.15: The set N

1. $\mathcal{P}[F]$ is true when F is an atomic formula.
2. If $\mathcal{P}[F]$ is true then so is $\mathcal{P}[\neg F]$.
3. If $\mathcal{P}[F]$ and $\mathcal{P}[G]$ are true, so is $\mathcal{P}[(F \vee G)]$.
4. If $\mathcal{P}[F]$ and $\mathcal{P}[G]$ are true, so is $\mathcal{P}[(F \wedge G)]$.
5. If $\mathcal{P}[F]$ and $\mathcal{P}[G]$ are true, so is $\mathcal{P}[(F \longrightarrow G)]$.
6. If $\mathcal{P}[F]$ and $\mathcal{P}[G]$ are true, so is $\mathcal{P}[(F \longleftrightarrow G)]$.

Figure 1.16: The rules for structural induction

and every other number is obtained by adding 1 to its predecessor. So, N is generated by the inductive rules from Figure 1.15. Rule 1 gives us a starting number and rule 2 generates new numbers from the existing ones. The statement $\mathcal{P}$ is true if it is true for the starting number and the generating rule preserves the truth value, i.e. if $\mathcal{P}[n]$ is true so is $\mathcal{P}[n+1]$. This way, the generating rules from Figure 1.15 give us the inductive rules from Figure 1.12. Now, let us look at Definition 1.1.2 that defines the formulas. Rule 1, that states that an atomic formula is a formula, is a starting (base) rule; the other 5 are generative rules. Now let us rewrite these 6 rules, such that that $\mathcal{P}$ is true for the base rules, and the truth is preserved by the generating rules. We get the rules from Figure 1.16.

Let us see how these rules operate by looking at an example.

Example 1.2.4 Assume that $\mathcal{P}$ is a statement satisfying the 6 rules from Figure 1.16. We will show that $\mathcal{P}[(((P_1 \vee P_2) \longrightarrow \neg P_3) \longleftrightarrow (P_2 \wedge \neg P_4))]$ true. For a better illustration of what is going on, we represent the formula as a tree.

Now let's apply the structural induction rules to the tree from Figure 1.17. We can use rule 1 only, since all the others require that $\mathcal{P}$ is already true for some subformula. Rule 1 makes all atomic formulas, represented by circles, true. So, we write $\mathcal{P}$ to the right of these nodes to show that the assertion holds for these nodes. We get the tree from Figure 1.18. Now we can apply rules 2 and 3, since all children of the nodes labeled $\vee$ and $\neg$ have the $\mathcal{P}$ label. We obtain the tree from Figure 1.19. Next, we can apply rules 4 and 5 because all children of the nodes labeled $\wedge$ and $\longrightarrow$ have the $\mathcal{P}$ label. We get the tree from Figure 1.20. Finally, we apply rule 6 to the Figure 1.20 tree to mark the root. Now we have the fully marked tree from Figure 1.21. Since the root is labeled the formula is true for P.

Let us assume that the statement $\mathcal{P}$ satisfies the rules from Figure 1.16. Then, Example 1.2.4 gives us a procedure for verifying that $\mathcal{P}$ satisfies a formula tree F. This procedure works bottom-up, i.e. it starts by assigning the $\mathcal{P}$ label to the leaves of the tree and then it propagates the $\mathcal{P}$ label towards the root by using the meta-rules 2 - 6. Figure 1.22 displays the meta-rules 1-6 as *rewrite*

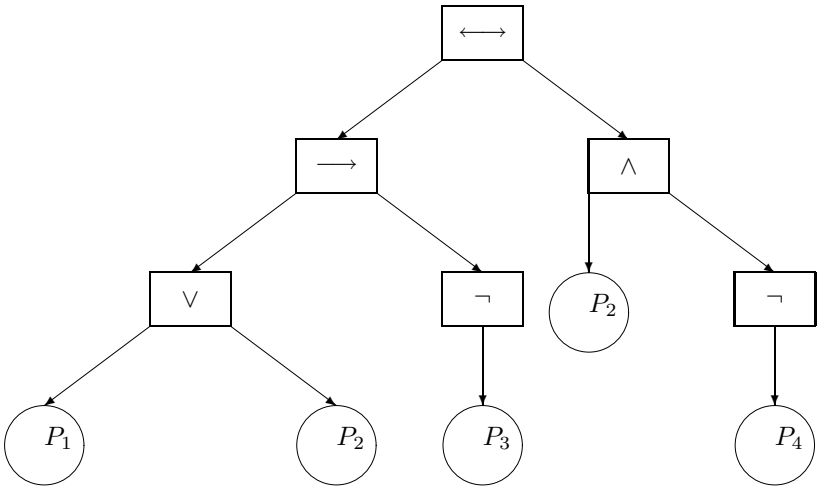


Figure 1.17: The tree from Example 1.2.4

rules that advance the $\mathcal{P}$ label towards the root. The left-hand-sides of the rules tell us that the node is either a leaf or all its children are labeled. We apply the meta-rule and get the labeling shown at the right of the rule. Now the top node of the left-hand-side is labeled. We did not bother writing $\mathcal{P}$ next to the children because they are irrelevant to the further advancement of $\mathcal{P}$. For the meta-rules 2-6, the children can be either leaves or branches. Now, a short observation: the order in which we apply these rewrites does not affect the end result. All these rewrites require is that we evaluate the children before we evaluate the parent, and the label inside the parent box determines which rewrite applies.

Theorem 1.2.5 (The Structural Induction Theorem) *If the statement P satisfies the rules listed in Figure 1.16, then P is true for any formula F.*

Proof: We assume that the theorem is false. Then, we use the fact that every non-empty set of formulas has a formula of minimum length, to derive a contradiction.

So, let us assume that $\mathcal{P}$ satisfies the 6 conditions of Figure 1.16, but it is false for some formulas. Among these formulas there are some of minimal length. Let F be one of them. Definition 1.1.2 tells us that F has one of the following 6 syntactic forms: $F = P_i$, $F = \neg G$, $F = (G \vee H)$, $F = (G \wedge H)$, $F = (G \longrightarrow H)$, $F = (G \longleftrightarrow H)$. We will show that in all 6 cases we get a

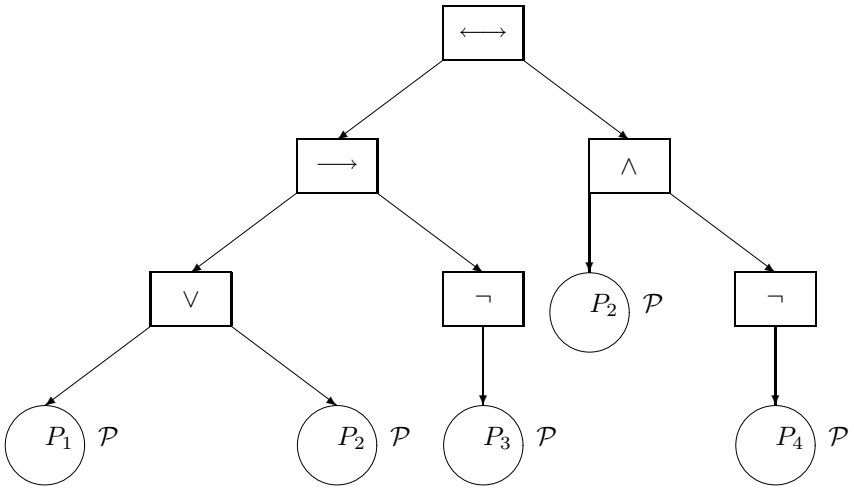


Figure 1.18: The Example 1.2.4 tree with marked leaves

contradiction.
Case 1: $F = P_i$ for some atom P_i. By condition 1 all atoms have the property $\mathcal{P}$. So, F cannot be an atom.
Case 2: $F = \neg G$. Now we look at $\mathcal{P}[G]$.

If $\mathcal{P}[G]$ is true then $\mathcal{P}[F]$ must be true by condition 2. Contradiction!

If $\mathcal{P}[G]$ is false then we have a formula shorter than F that makes P false. This contradicts the fact that F has minimal length.

So, F cannot be of the form $F = \neg G$.
Cases $F = (G \vee H)$, $F = (G \wedge H)$, $F = (G \longrightarrow H)$, $F = (G \longleftrightarrow H)$:

If $\mathcal{P}[G]$ and $\mathcal{P}[H]$ are both true then $\mathcal{P}[F]$ is true by one of the conditions 3 - 6 (condition 3 for $\vee$, 4 for $\wedge$, 5 for $\longrightarrow$ and 6 for $\longrightarrow$). Contradiction!

If $\mathcal{P}$ is false for at least one of the subformulas G, H then we get a formula shorter than F that makes $\mathcal{P}$ false. This contradicts the fact that F has minimal length.

So, F cannot have one of forms $F = (G \vee H)$, $F = (G \wedge H)$, $F = (G \longrightarrow H)$, $F = (G \longleftrightarrow H)$.

Since F cannot fit into any of the 6 cases, we conclude that there is no such F. This means that our assumption that there are formulas that make $\mathcal{P}$ false is wrong. **Q.E.D.**

Remark 1.2.6 When we prove the implications

1. if $\mathcal{P}[G]$ is true then $\mathcal{P}[\neg G]$ is true, and

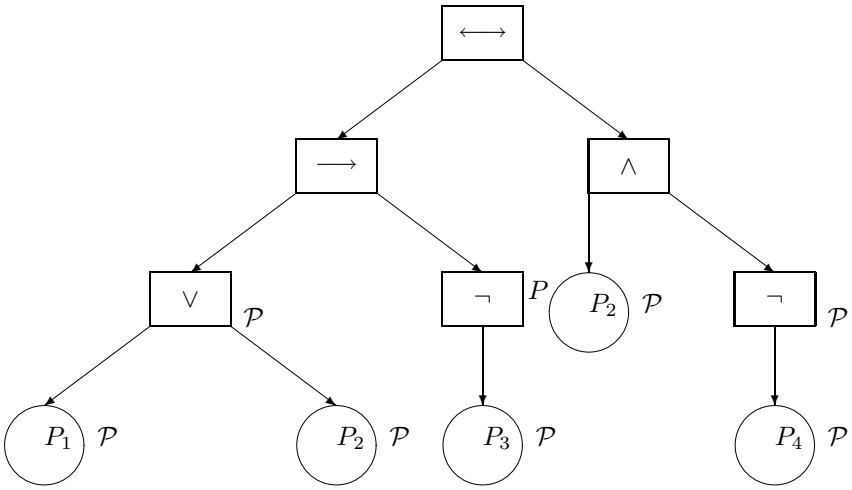


Figure 1.19: The Example 1.2.4 tree with more marked nodes

2. if $\mathcal{P}[G]$ and $\mathcal{P}[H]$ are true then $\mathcal{P}[GCH]$ is true, with $C \in \{\vee, \wedge, \longrightarrow, \longleftrightarrow\}$,

we call the statements $\mathcal{P}[G]$ and $\mathcal{P}[H]$ *induction hypotheses* and we label them with the letters IH.

The Structural Induction Theorem provides a powerful method for proving formula properties. All we have to do is to show that $\mathcal{P}$ satisfies the 6 conditions from Figure 1.16. Before we apply this theorem, let us introduce the notation $n[s, S]$. This expression is the number of times the symbol s occurs in the string S. For example, $n[a, babaca(] = 3$, because the string $babaca($ has 3 a's. We write $n[atom, F]$ to represent the number of atomic formula occurrences in F and $n[con, F]$ for the number of binary connectives ($\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$) occurrences in F.

Lemma 1.2.7 *Let F be a formula and S a prefix of F. Then $n[), S] \leq n[con, S] \leq n[(, S]$, i.e. every prefix of a formula has at least as many left parentheses as binary connectives, and at least as many binary connectives as right parentheses.*

Proof: We use The Structural Induction Theorem.
Case 1: F is an atomic formula. Then $F = P_i$, where i is one of the numbers 0, 1, 2, ... The only prefixes of P_i are λ and P_i. Both strings have no parentheses and no binary connectives, so the lemma is true for atomic formulas.

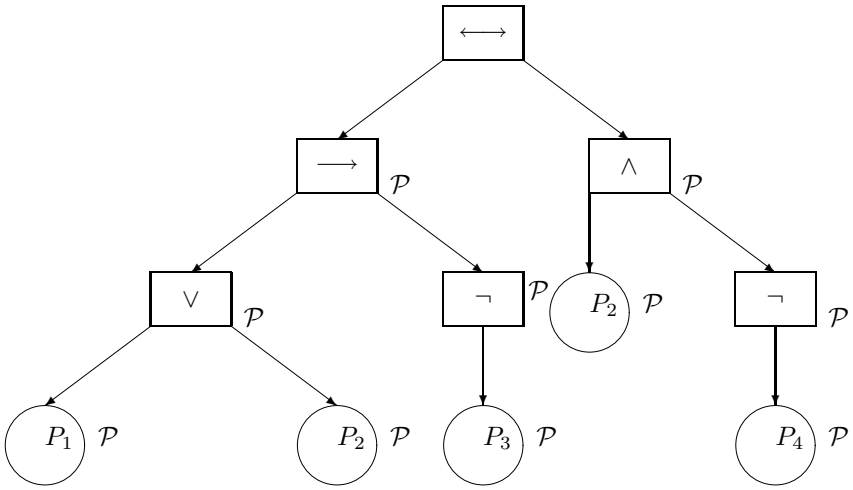


Figure 1.20: Almost all nodes of the Example 1.2.4 tree are marked

Case 2: $F = \neg G$. We assume that the lemma is true for G. We need to show that it is also true for F. We have a little problem here because the induction hypothesis tells us about the prefixes of G but we have to show the double inequality for the prefixes of F. But how do we relate the prefixes of F to the prefixes of G?

Let's look at the example $F = \neg\neg P_1$. Here $G = \neg P_1$. The prefixes of F are λ, $\neg$, $\neg\neg$, and $\neg\neg P_1$. We notice one thing. Except for the prefix λ, all the other prefixes of F contain the first $\neg$ followed by a prefix, possibly empty, of G !!! So, the prefix $\neg$ of F is the symbol $\neg$ followed by the prefix λ of G, $\neg\neg$ is $\neg$ followed by the prefix $\neg$ of G, and $\neg\neg P_1$ is $\neg$ followed by the prefix $\neg P_1$ of G!

Now let's return to the proof. Let S be a prefix of F. Then either S is empty or S is of the form $\neg I$, where I is a prefix of G.
The empty string poses no problems since it has no symbols, so $n[), \lambda] = n[con, \lambda = n[(, \lambda] = 0$.
Now let's solve the case $S = \neg I$, with I is a prefix of G.
By induction hypothesis,

(IH) $n[), I] \leq n[con, I] \leq n[(, I]$.

All the parentheses and the binary connectives that occur in S must be in I, so

$n[(, S] = n[(, I]$,
$n[con, S] = n[con, I]$, and
$n[), S] = n[), I]$.

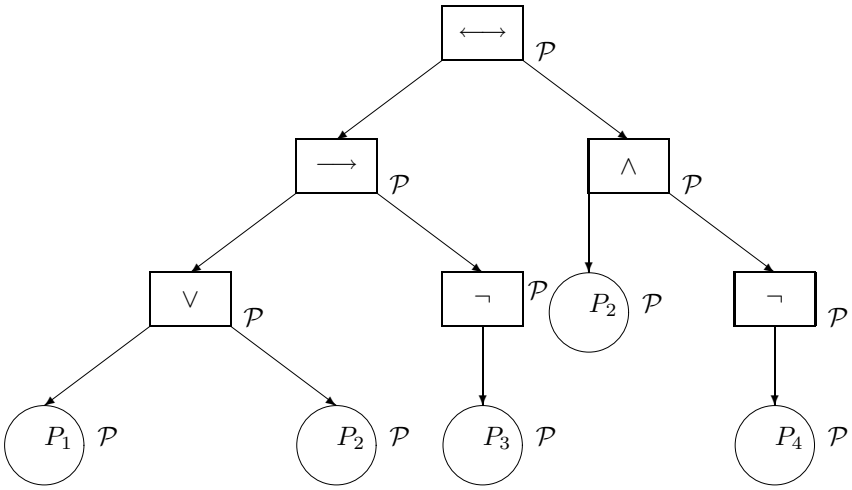


Figure 1.21: The Example 1.2.4 tree with marked root

We use these equations to replace $n[(, I]$, $n[con, I]$, and $n[), I]$ in the induction hypothesis and we get

$n[), S] \leq n[con, S] \leq n[(, S]$,

i.e. S satisfies the lemma.

Cases 3,4,5,6: $F = (GCH)$, where C is one of the symbols $\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$, and G and H are formulas. Let S be a prefix of F. We look at the position (index) of the last character of S. That index can be -1 if S is the empty string, on the first left parentheses or somewhere inside G, on the connective C or somewhere inside H, or on the last parenthesis of F. These 4 categories are displayed in Figure 1.23. We need to show that the lemma holds for each category.

Category 1:$S = \lambda$. We already showed, in Case 2, that λ satisfies the lemma.

Category 2: $S = (I$ where I is a prefix of G. Here we use the induction hypothesis for G, i.e.

(IH) $n[), I] \leq n[con, I] \leq n[(, I]$.

All the right parentheses and all the connectives of S must be in I; the left parentheses can either be in I or the parenthesis in front I.

(1) $n[), S] = n[), I]$

(2) $n[con, S] = n[con, I]$

(3) $n[(, S] = n[(, I] + 1$

Now we use (1), (2), and (3) to replace $n[), I]$, $n[con, I]$, and $n[(, I]$ in (IH) and get

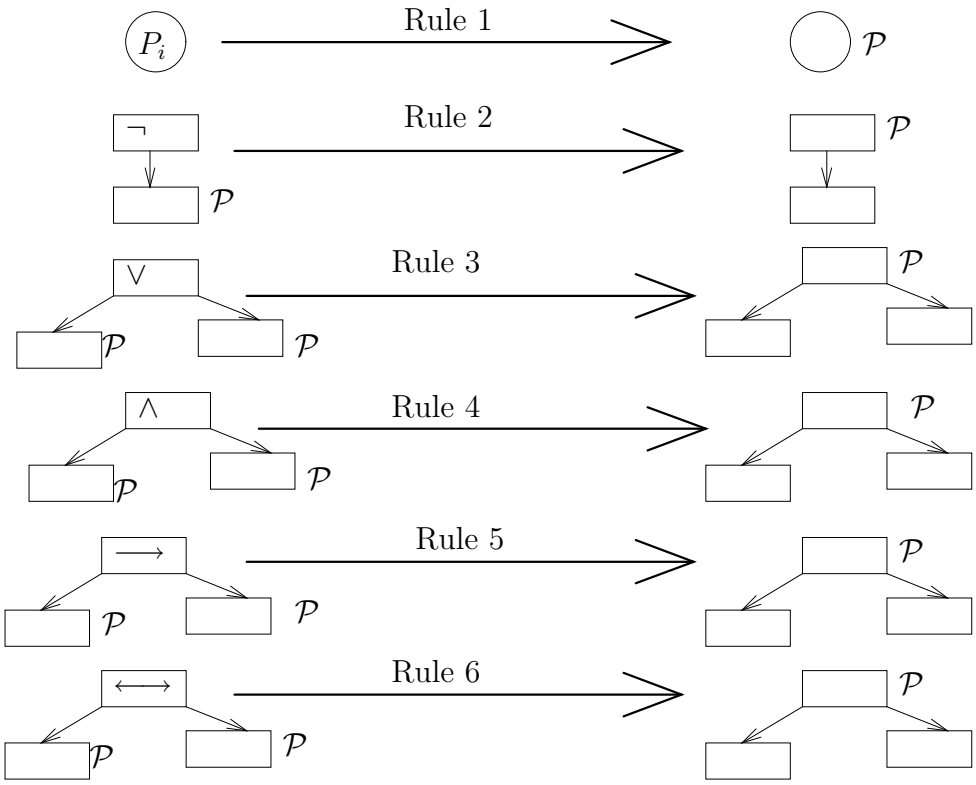


Figure 1.22: The rewrite rules for structural induction

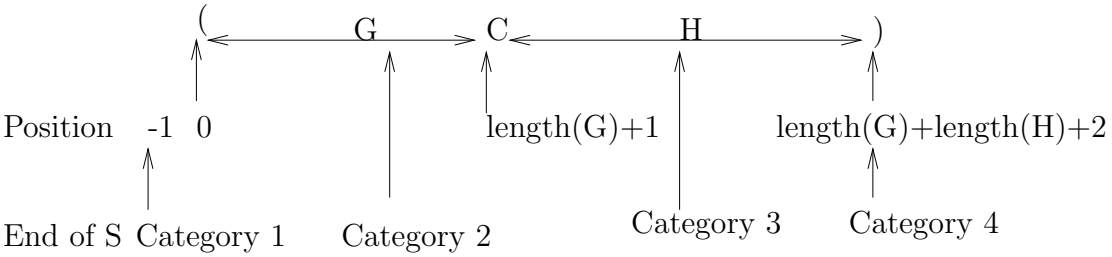


Figure 1.23: The 4 categories of Cases 3-6 of Lemma 1.2.7

(4) $n[), S] \leq n[con, S] \leq n[(, S] - 1$

From (4) we get that $n[), S] \leq n[con, S] < n[(, S]$ i.e. S satisfies the lemma.
Category 3:$S = (GCI$, where I is a prefix of H

In this case we have 2 induction hypotheses, one that applies to G (Recall that G is a prefix of G !) and one that applies to I.

(IH1) $n[), G] \leq n[con, G] \leq n[(, G]$

(IH2) $n[), I] \leq n[con, I] \leq n[(, I]$

Now let us express $n[(, S]$ as an expression with $n[(, G]$ and $n[(, I]$. We see that all the left parentheses of S are either the first parenthesis, are in G, or are in I. So,

(5) $n[(, S] = 1 + n[(, G] + n[(, I]$.

The binary connectives of S are either in G, or in I, or the C separating G from I. So,

(6) $n[con, S] = n[con, G] + 1 + n[con, I]$.

The right parentheses of S are either in G or in I, so

(7) $n[), S] = n[), G] + n[), I]$.

We add (IH1) and (IH2) and get

(8) $n[), G] + n[), I] \leq n[con, G] + n[con, I] \leq n[(, G] + n[(, I]$,

Now we use the equations (5)-(7) to replace the sums in (8) by the S counts.

(9) $n[(, S] \leq n[con, S] - 1 \leq n[(, S] - 1$

By adding 1 to the last 2 expressions we get

(10) $n[(, S] < n[con, S] \leq n[(, S]$, i.e.

S satisfies the proposition.

Category 4:$F = S$ This case is left as exercise. **Q.E.D.**

As expected, if S is a suffix of a formula, the inequalities from Lemma 1.2.7 are reversed.

Lemma 1.2.8 *Let F be a formula and S be a suffix of S. Then $n[(, S] \leq n[con, S] \leq n[), S]$.*

The proof of this lemma is left as exercise. (Exercise 1.2.14)

There are some important corollaries to these 2 lemmas.

Corollary 1.2.9 *For every formula F, $n[(, F] = n[con, F] = n[), F]$.*

Proof: Since F is a prefix of itself,

(1) $n[), F] \leq n[con, F] \leq n[(, F]$.

The formula F is also a suffix of itself, so

(2) $n[(, F] \leq n[con, F] \leq n[), F]$.

From $n[(, F] \leq n[con, F]$ and $n[con, F] \leq n[(, F]$ we get $n[(, F] = n[con, F]$. From $n[), F] \leq n[con, F]$ and $n[con, F] \leq n[), F]$ we get $n[con, F] = n[), F]$. **Q.E.D.**

Corollary 1.2.10 *Let G, H be two formulas, I a suffix G, J a prefix of H and C a binary connective. Then, the string $K = ICJ$ is not a formula.*

Proof: Assume that $K = ICJ$ is a formula. By Lemma 1.2.7,

(1) $n[), I] \leq n[con, I] \leq n[(, I]$.

At the same time, I is a suffix of G, so by Lemma 1.2.8,

(2) $n[(, I] \leq n[con, I] \leq n[), I]$.

From (1) and (2) we get

(3)$n[), I] = n[con, I] = n[(, I]$.

The string J is both a prefix of the formula H and a suffix of the formula K, so we have

(4) $n[), J] = n[con, J] = n[(, J]$.

Now we count the (binary) connectives of K. They are either in I, or in J, or the middle C. So,

(5) $n[con, K] = n[con, I] + 1 + n[con, J]$.

The left parentheses of K are either in I or in J.

(6) $n[(, K] = n[(, I] + n[(, J]$

But, then

$n[con, K] = n[con, I] + 1 + n[con, J]$ by (5)

$= n[(, I] + 1 + n[(, J]$ by (3) and (4)

$= n[(, I] + n[(, J] + 1$

$= n[(, K] + 1$ by (6)

$> n[(, K]$.

The inequality $n[con, K] > n[(, K]$ contradicts Corollary 1.2.9 that says that a formula must have the same number of left parentheses and binary connectives. **Q.E.D.**

Lemma 1.2.11 *If S is a prefix of a formula F and S is a formula, then $S = F$.*

Proof: We do it by structural induction on F.

Case 1: F is an atomic formula. Then, the prefixes of F are $S = \lambda$ and $S = F$. Since λ is not a formula, we are left with $S = F$, and F is a formula.

Case 2: $F = \neg G$. Again the prefixes of F are $S = \lambda$ and $S = \neg I$, where I is a prefix of G. We discard $S = \lambda$, so let $S = \neg I$. If S is a formula, so is I. But I is a prefix of G, so we can apply the induction hypothesis to G and I and get that $I = G$. Then $S = \neg I = \neg G = F$.

Cases 3,4,5,6: $F = (GCH)$. We have the 4 categories from Figure 1.23. We discard $S = \lambda$ since λ is not a formula.

Category 2: $S = (I$, where I is a prefix of G. We count the left and the right parenthesis of S.

$n[(, S] = 1 + n[(, I]$ because $S = (I$

$\geq 1 + n[(, I]$ by Lemma 1.2.7, $n[(, I] \geq n[), I]$

$> n[(, I]$

$= n[(, S]$ because $S = (I$

So, S has more left parentheses than right parentheses, contradicting Corollary 1.2.9.

Category 3: $S = (GCJ$, here J is a prefix of H. We count the number of parentheses of S and get that it has more left parentheses than right parentheses, contradicting Corollary 1.2.9.

$n[(, S] = 1 + n[(, G] + n[(, J]$ because $S = (GCJ$
$= 1 + n[), G] + n[(, J]$ by Corollary 1.2.10, $n[(, G] = n[), G]$
$\geq 1 + n[), G] + n[), J]$ by Lemma 1.2.7, $n[(, I] \geq n[), I]$
$> n[(, G] + n[(, J]$
$= n[(, S]$ because $S = (GCJ$

For the last category we have nothing to prove since $S = F$.

So, the only prefixes that are formulas are in Category 4, and those are the formula itself. **Q.E.D.**

This lemma has two important consequences, The Unique Readability Theorem, and Corollary 1.2.13. The Unique Readability Theorem states a key result, that our language contains no ambiguity.

Theorem 1.2.12 (The Unique Readability Theorem) *Every formula F belongs to only one of the following 6 categories: atoms, $\neg I$, $(I \vee J)$, $(I \wedge J)$, $(I \longrightarrow J)$, $(I \longleftrightarrow J)$ and the decomposition of F is unique.*

Proof: Let us assume that a formula has two decompositions, F and G. The strings F and G are equal, but they may be decomposed into non-equal subformulas.

If F is an atom, then it has length 1. So, G has also length 1, and is equal to F.

Assume now that $F = \neg I$. Since $F = G$, G must also start with $\neg$, i.e. $G = \neg J$ for some string J. From the equality $\neg I = \neg J$ we get that $I = J$. Again, the decomposition is unique.

Now let $F = (ICJ)$. Since $G = F$, the first symbol of G must also be a left parenthesis, i.e. $G = (KC_1L)$ for some formulas K and L and some binary connective C_1. Now, I and K are both substrings of $F = G$ that start at position 1. We claim that $I = K$. Assume that $I \neq K$. Then one of them must be a (proper) substring of the other. Since both I and K are formulas, we apply Lemma 1.2.11 and get that $I = K$, contradicting our assumption. Since $I = K$ and $(ICJ) = (KC_1L)$ we get that $CJ) = C_1L)$. The last equality yields $C = C_1$ and $L = J$. So, the connective C and the subformulas I and J are uniquely defined. **Q.E.D.**

Corollary 1.2.13 *1. The subformulas of $\neg G$ are $\neg G$ and the subformulas of G.*

2. The subformulas of (GCH) are (GCH), the subformulas of G, and the subformulas of H.

Proof: The subformulas are substrings that are also formulas. They are nonempty so, they have an index i that points to the first symbol and an index j that points to the last symbol.

1. Let S be a subformula of $\neg G$. The index i can be 0 or greater than 0, as shown in Figure 1.24. If $i = 0$, S is prefix of $\neg G$. Then $S = F$ by Lemma 1.2.11.

If $i > 0$ then S is a substring of G, so S is a subformula of G.

2. Let S be a subformula of (GCH). Then we have the following 5 cases: $i = 0$, $0 < i < length(G) + 1$, $i = length(G) + 1$, $length(G) + 1 < i <$

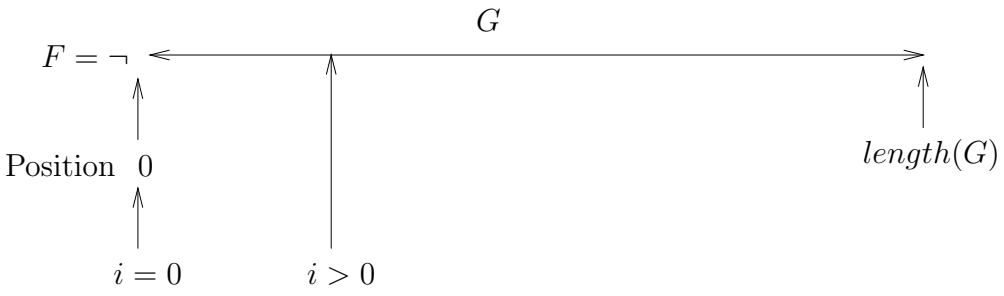


Figure 1.24: The subformulas of $\neg G$ from Corollary 1.2.13

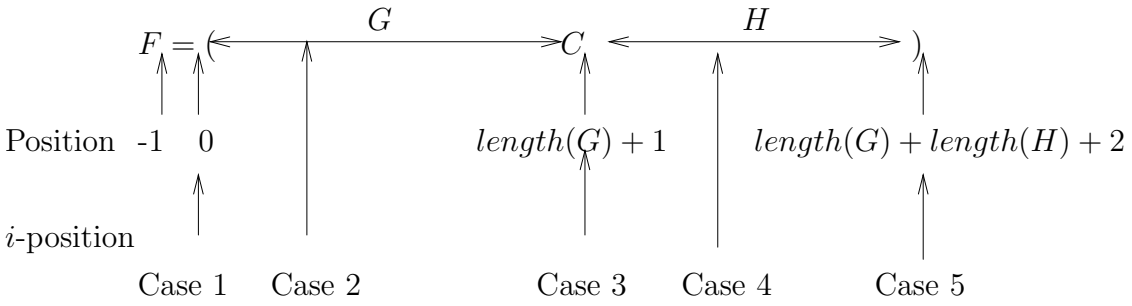


Figure 1.25: The subformulas of (GCH) from Corollary 1.2.13

$length(G) + length(H) + 2$, and $i = length(G) + length(H) + 2$. Figure 1.25 shows the 5 cases. We can discard Cases 3 and 5 because no formula starts with a binary connective or a right parenthesis.
Case 1: $i = 0$. Then S is a prefix of (GCH), so $S = (GCH)$ by Lemma 1.2.11.
Case 2: $0 < i < length(G) + 1$. Then S starts in G. We have 4 subcases according to the position of j (Figure 1.26).

We can discard Subcase 2.2 because no formula ends with a binary connective.

Subcase 2.1: $j < length(G) + 1$. Then S is a substring of G, hence a subformula of G.

Subcase 2.3: $length(G) + 1 < j < length(G) + length(H) + 2$ i.e. S ends in H. Then $S = ICJ$ with I a suffix of G and J a prefix of H. Then S is not a formula by Corollary 1.2.10.

Subcase 2.4: $j = length(G)+length(H)+2$. Then $S = JCH)$ with J a suffix of G. We will show that S has more right parentheses than left parentheses, so it is not a formula.

$n[), S] = n[), J] + n[), H] + 1$ because $S = JCH)$
$\geq n[(, J] + n[(, H] + 1$ we apply Lemma 1.2.11 to J and H
$> n[(, J] + n[(, H]$

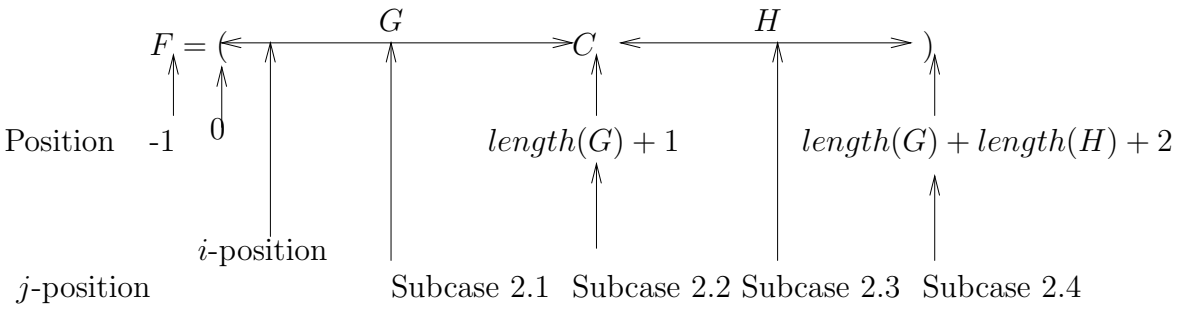


Figure 1.26: Case 2 of Corollary 1.2.13

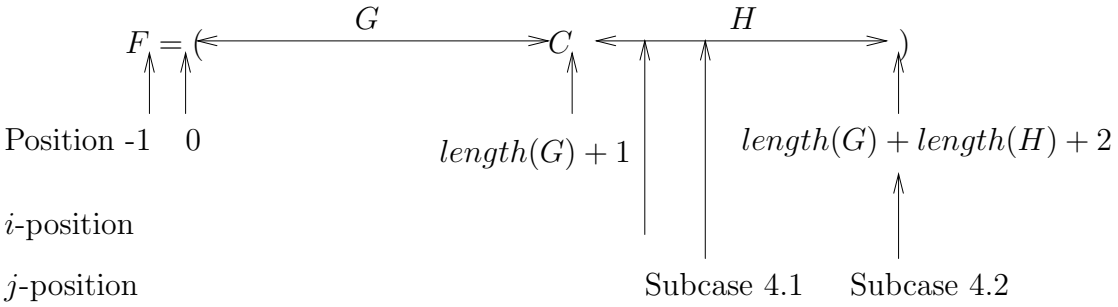


Figure 1.27: Case 4 of Corollary 1.2.13

$= n[(, S]$ because $S = JCH$)

Case 4: $length(G) + 1 < i < length(G) + length(H) + 2$. Then S starts in H. We have 2 subcases, depending on whether S ends in H or on the last symbol of (GCH) (Figure 1.27).

Subcase 4.1: S ends in H. Then S is a subformula of H.

Subcase 4.2: S ends in H on the last symbol of (GCH). Then $S = J)$ with J a suffix of H. We will show that S has more right parentheses than left parentheses, so it cannot be a formula.

$n[), S] = n[), J] + 1$ because $S = J)$

$\geq n[(, J] + 1$ we apply Lemma 1.2.11 to J

$> n[(, J]$

$= n[(, S]$ because $S = J)$

Q.E.D.

Now let us prove that the algorithm from Figure 1.1 is correct, i.e. it terminates, and accepts a string if and only if the string is a formula.

Lemma 1.2.14 *The algorithm from Figure 1.1 terminates.*

Proof: Assume that it does not terminate for some input. Since the scanning

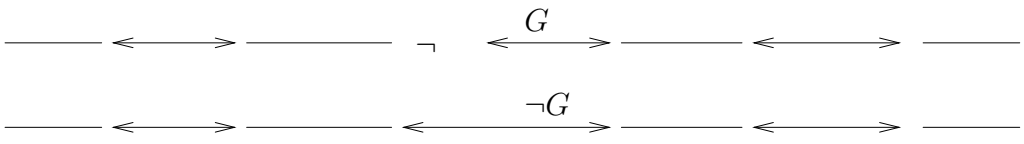


Figure 1.28: What happens when we underline a negation

loops terminate, the while loop at Step 2 must run forever. This means that at each iteration, the algorithm underlines additional input symbols. This is impossible, since the input has finitely many characters. **Q.E.D.**

During the execution of the algorithm, the symbols of the input S are either underlined or unmarked. We say that the occurrence T that begins at i is a *max-occurrence* if it is fully underlined and the symbols that precede T and succeed T do not exist or are unmarked. So, the max-occurrences are the largest underlined occurrences of S. For example, the max-occurrences of $S = ((\underline{\neg P_1} \vee \underline{P_2}) \wedge (\underline{P_1} \longrightarrow \underline{(P_2 \longleftrightarrow P_3)}))$ are the occurrence of $\neg P_1$ at position 2, the occurrence of P_2 at 5, the occurrence of P_1 at 9, and the occurrence of $(P_2 \longleftrightarrow P_3)$ at 11.

Every time we underline a symbol we change the set of max-occurrences.

Lemma 1.2.15 *The max-occurrences are always formulas.*

Proof: We need to show that whenever the algorithm underlines a new symbol, the newly created max-occurrences are formulas. We prove the lemma by induction n, the number of underline statements executed so far.

Basis: $n = 0$. Since there are no occurrences, we have nothing to prove.

Inductive Step: $n > 0$. Assume that the statement is true for the first n executions. Let us assume that we underline a new symbol. This can be done by at Step 1, or by the two underline statements from Step 2.

If the underlining is done in Step 1, the marked symbol is an atomic formula and the symbol that precedes it, if it exists, is unmarked. Moreover, the character succeeding the underlined symbol, if it exists, is unmarked because the algorithm hasn't processed it yet. So, the underlining produces only one new max-occurrence, the marked symbol. Since that is an atomic formula, the max-occurrence is a formula.

Now let us assume that we underline a negation. We can do this only when $\neg$ is succeeded by an underlined string and the symbol in front of the negation is either non-existent or unmarked. The underlined string is part of a max-occurrence, say G, as shown at the top of Figure 1.28. There, the max-occurrences are represented by arrow ended segments and the unmarked substrings by the usual segments. The effect of the underline is that the max-occurrence G disappears, and we get a new max-occurrence, $\neg G$. All the other max-occurrences of S remain the same. By IH, G is a formula. Then, so is $\neg G$. The new max occurrences are shown at the bottom of Figure 1.28.

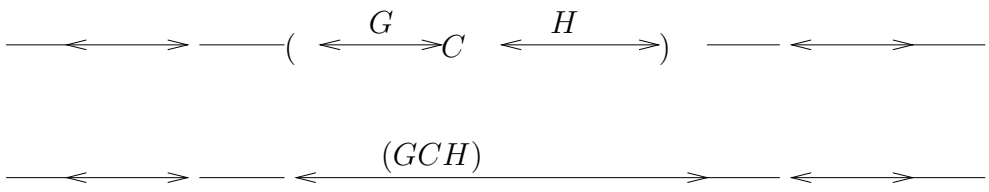


Figure 1.29: What happens when we underline (GCH)

Assume that we underline a left parenthesis at Step 2. Before we do it, we have the situation shown at the top of Figure 1.29 where $C \in \{\vee, \wedge, \longrightarrow, \longleftrightarrow\}$, and G, H are max-occurrences. We underline the parentheses and the connective C only when the symbol in front of (and the symbol that succeeds) are unmarked, so the effect of the underline is that the max-occurrences G and H disappear and we get a new max occurrence, (GCH), as shown at the bottom of Figure 1.29. All the other max-occurrences stay the same. The new max-occurrence is also a formula because we assumed that G and H are formulas. **Q.E.D.**

Corollary 1.2.16 *The formula recognition algorithm from Figure 1.1 accepts only formulas.*

Proof: Assume that S is accepted by the algorithm. Then it stops when S is fully underlined. But this means that S is a max-occurrence. By Lemma 1.2.15, S is formula.**Q.E.D.**

Lemma 1.2.17 *If the algorithm accepts F, the first symbol of F is underlined last.*

Proof: We prove it by contradiction. Assume that an underline statement marks position 0 but leaves some positions unmarked. Let I be the max-occurrence generated by the underline. By Lemma 1.2.15, I is a formula. At the same time, I is a prefix of F and I is not equal to F. This contradicts Lemma 1.2.11. **Q.E.D.**

Lemma 1.2.18 *The Formula Recognition Algorithm accepts all formulas.*

Proof: The proof is by structural induction on F.
Case 1: F is an atom. Then it is underlined at Step 1, it fails the loop test and is accepted at Step 3.
Case 2: $F = \neg G$. By IH, the algorithm accepts G. Let us run the algorithm with input F. We claim that the first negation is not underlined until the whole string G is underlined. Assume that this is not true, i.e. at some time during the execution, the algorithm underlines the negation while G is not fuly marked. In order to underline $\neg$ there must be a max-occurrence I that succeeds it. This means that I is a prefix of G. By Lemma 1.2.15, I is a formula. Then, by

Lemma 1.2.11, $I = G$, i.e. G is fully underlined. So, we get a contradiction, thus proving our claim.

The claim tells us that the algorithm ignores the negation as long as G is not underlined. Since $\neg$ does not interfere with the underlining of G, we can apply the induction hypothesis that says that G will be fully underlined. Then, at the next execution of Step 2, so will be F.
Cases 3,4,5,6: $F = (GCH)$, where C is a binary connective. We assume that G and H are accepted and we run the algorithm with input (GCH). We claim that the algorithm will not underline the end parentheses and the connective until both G and H are underlined. Assume that this is not the case and, at some time during the execution, there is a max-occurrence $J \neq F$ that contains one of these 3 symbols. By Lemma 1.2.15, J is a formula. By Corollary 1.2.13, J must be equal to either F, a subformula of G, or a subformula of H. The only subformula that contains any of these 3 symbols is F, so $J = F$. But this contradicts the hypothesis that $J \neq F$, thus proving our claim.

The claim tells us that the algorithm will keep the underlines inside G and inside H, as long as either one of them has unmarked symbols. So, it will ignore the 3 symbols. Since the connective C is unmarked, the underlinings of G and H do not interfere with each other. Then we can apply the induction hypothesis, that says that G and H will eventually be marked. At the next step, F will also be underlined. **Q.E.D.**

Figure 1.3 presents an algorithm that converts a formula into a formula tree. The loop of the algorithm is controlled by the same underline mechanism as the formula recognition algorithm from Figure 1.1. We showed, in Lemma 1.2.14, that the recognition algorithm terminates for all formulas. Then, the algorithm from Figure 1.3 also terminates for all formulas. Let α be the function computed by this algorithm[3]. The procedure convert, from Figure 1.8, also terminates for all input trees, the reason being that it makes at most two recursive calls, and every call decreases the height of the input tree[4]. We will show that the functions α and convert are inverses, i.e. $\alpha[convert[t]] = t$ for all formula trees t, and convert$[\alpha[F]] = F$ for all formulas F.

Lemma 1.2.19 *For all formulas F, convert$[\alpha[F]] = F$ and for all formulas trees t, $\alpha[convert[t]] = t$.*

Proof: First, we show that convert$[\alpha[F]] = F$. The proof is by structural induction on F.
Case 1: If F is an atom, the algorithm underlines F, and creates a leaf with label F. It does not go through the loop and outputs the tree t having the leaf as its root. Then convert$[t]$ outputs the label of the root, i.e. F.
Case 2: $F = \neg G$. From Lemma 1.2.17 we know that the algorithm does not underline $\neg$ until G is fully underlined. The tree $t = \alpha[\neg G]$ has the root labeled $\neg$ that has one child, $\alpha[G]$. When convert transforms t, it outputs the label of the root and then converts the subtree at address 0. So, convert$[t] = \neg$convert$[\alpha[G]]$. By IH, convert$[\alpha[G]] = G$. So, convert$[\alpha[F]] = F$.

[3]The interested reader can prove, by structural induction on F, that $\alpha[F]$ is a formula tree.
[4]The reader can prove, by induction on the height of the tree, that convert[t] is a formula.

Cases 3,4, 5, 6: $F = (GCH)$. Lemma 1.2.18 tells us that the algorithm accepts F, so F is in the domain of α. From Lemma 1.2.17, we know that the leftmost (of F is underlined during the last sweep. At that time $F = (G_1C_1H_1)$ with the parentheses and C_1 unmarked and G_1, H_1 underlined. By Lemma 1.2.15, G_1 and H_1 are formulas. By The Unique Readability Theorem, $G = G_1$, $C = C_1$ and $H = H_1$. So, the tree t generated by the algorithm has the root labeled C, $t/0 = \alpha[G]$ and $t/1 = \alpha[H]$. The procedure convert$[t]$ writes a left parenthesis, converts $t/0$, writes the label of the root, converts $t/1$, and writes a right parenthesis. So, convert$[t]$ = (convert$[t/0]C$convert$[t/1])$ = (convert$[\alpha[G]]C$convert$[\alpha H])$.

By IH, convert$[\alpha[G]] = G$ and convert$[\alpha[H] = H$, so
convert$[t]$ = (convert$[\alpha[G]]C$convert$[\alpha[H]]) = (GCH) = F$.

Now let us show the second part, that $\alpha[\text{convert}[t]] = t$ for all formula trees t. The proof is by induction on the height of the tree.
Basis: The height is 0. Then the tree is a leaf and its label is an atomic formula, say P_i. Then convert$[t] = P_i$ and $\alpha[P_i]$ is a tree having one node with the label P_i. So, $\alpha[\text{convert}[t]] = t$.
Inductive Step: The height is greater than 0. Then the root of t is labeled with the negation symbol or a binary connective. We will prove the latter case and leave the other one to the reader. Let C be the label of the root. Then convert$[t]$ = (convert$[t/0]C$convert$[t/1])]$. The strings convert$[t/0]$ and convert$[t/1]$ are formulas, so α will first build formula trees for them and then will create a root labeled C that has α[convert$[t/0]$ as its first child and α[convert$[t/1]$ as the second. The trees $t/0$ and $t/1$ have smaller heights than t, so we apply the IH and get that α[convert$[t/0] = t/0$ and α[convert$[t/1] = t/1$. So, α[convert$[t]]$, has $t/0$ as its first child, the root labeled C, and $t/1$ as its second child. Then $\alpha[\text{convert}[t]] = t$. **Q.E.D.**
Exercises

Exercise 1.2.1 *Prove by induction on n that $2^n > n$.*

Exercise 1.2.2 *Prove by induction that $3|n(n+1)(n+2)$ i.e. that 3 divides evenly the expression $n(n+1)(n+2)$.*

Exercise 1.2.3 *Prove by structural induction that $n[(, F] = n[), F]$, i.e. every formula F has the same number of ('s and)'s.*

Exercise 1.2.4 *Prove, by structural induction, that $n[con, F] = n[atom, F] - 1$.*

Exercise 1.2.5 *In the proof of The Structural Induction Theorem we assumed that F is a formula of minimal length that makes P false. Would the proof work if we take F to be a formula with the minimal number of atoms? Would it work if we take F to be a formula with the minimal number of connectives?*

Exercise 1.2.6 *Look at the proof of Lemma 1.2.7. In what categories fall the prefixes $(G$, $(GC$, $(GCH$? In all 3 cases specify the value of the prefix I.*

Exercise 1.2.7 *Finish the proof of Lemma 1.2.7 by taking care of Category 4.*

Exercise 1.2.8 *Let $|S|$ be the* length *(the number of symbols) of the string S. Prove by structural induction that for every formula F, $|F| = n[\neg, F] + 4n[con, F] + 1$. Remember that $n[con, F]$ is the number of occurrences of the binary connectives, $\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$.*

Exercise 1.2.9 *Let F be a formula that contains occurrences of $\neg$ and G the string obtained by deleting one occurrence of $\neg$ from F. Prove, by structural induction on F, that G is a formula.*

Exercise 1.2.10 *Show that there are no strings X and Y such that both X and Y are non-empty and both XY and YX are formulas.*

Exercise 1.2.11 *Let F, G,H be 3 formulas such that H contains occurrences of F. Let I be the result of replacing one occurrence of F in H by G. Prove, by structural induction on H, that I is a formula.*

Exercise 1.2.12 *Show that there do not exist non-empty string X, Y, Z such that both XY and YZ are formulas.*

Exercise 1.2.13 *Prove by structural induction on F that every non-empty suffix S of F with $n[(, S] = n[), S]$ is a formula.*

Exercise 1.2.14 *Prove Lemma 1.2.8.*

Exercise 1.2.15 *Prove that the functions α and convert are one to one and onto.*

Exercise 1.2.16 *Use the fact that α is one-to-one to prove The Unique Readability Lemma.*

Exercise 1.2.17 *Show that the algorithm below accepts all formulas, and some strings that are not formulas.*
Step 1. Scan S from left to right and underline all symbols in it that are atomic formulas.
Step 2. while [[S is not fully underlined] and [the previous scan produced new underlines]] do
{
while [scanning S from left to right] do
{
1. if [$\neg F$ is a substring of S and F is underlined] then underline $\neg$;
2. if [(FCG) is a substring of S, F and G are underlined and $C \in \{\vee, \wedge, \longrightarrow, \longleftrightarrow\}$] then
underline (FCG);
}
}
Step 3. Stop with success if S is (fully) underlined and stop with failure if it is not.

Exercise 1.2.18 *We define the formulas by the following rules:*

1. *the symbols* $P_0, P_1, \ldots P_n, \ldots$ *are formulas,*
2. *if* F *is a formula, so is* $\neg F$*,*
3. *if* F *and* G *are formulas, so is* $F \vee G$*.*

Show that the language defined by these rules does not have the unique readability property.

Exercise 1.2.19 *Write The Structural Induction Theorem for meta-formulas.*

Exercise 1.2.20 *Prove that the meta-formulas have the unique readability property*

1.3 Truth Assignments

Up until now we have viewed formulas as strings and characterized them in terms of their substrings. In this section we introduce the semantics of formulas, i.e. we give meanings to formulas. We define truth assignments, that attach to each atomic formula a truth value. In keeping up with the computer science tradition we use 1 and 0 instead of *true* and *false*. Then, we use The Unique Readability Theorem to extend the truth assignment to the set of all formulas. Next, we give a procedure for finding the truth value of a formula. The section ends with The Agreement Theorem that says that the truth value of a formula is determined by the truth values of its atoms.

Before we go on to the truth assignments, let us make clear the difference between *syntax* and *semantics* by looking at some English sentences.

Syntactically, the English sentence *Michelle loves Mark and Mark loves Mary* is formed by joining the clauses *Michelle loves Mark* and *Mark loves Mary* with the conjunction *and*.

The syntax looks at sentences as strings that can be used to form larger sentences using well defined string operations. One such operation is the *and* rule that takes two sentences *S1* and *S2* and forms the sentence *S1 and S2*.

The semantics is concerned with the truth of the sentences. So, *Mike likes bananas* is true if Mike likes bananas, and it is false if Mike is not fond of this fruit. The semantics assigns to the sentence one of the two truth values, true or false.

In this chapter clauses like *Mike likes bananas* are regarded as atomic formulas, so we are not going to get into their gramatical structure. Instead, the focus is on how we determine the truth value of a complex sentence from the truth values of its component sentences.

The sentence *Mary likes cheese balls and Mark likes Mary* consists of two clauses, *Mary likes cheese balls* and *Mark likes Mary*, formed with the *and* rule. The complex sentence is true if both clauses, *Mary likes cheese balls* and *Mark likes Mary*, are true.

We are also interested in semantic *equivalences*, i.e. sentences that have the same meaning. In English, the sentence *Bill likes apples and Mike likes pasta* has the same meaning as *Mike likes pasta and Bill likes apples*.

Now, that we have a feel for what we are going to do, let us return to defining a formal semantics for formulas.

Definition 1.3.1 (truth assignments) *A truth assignment is a function that assigns to each atomic formula P_i a truth value, i.e. a 0 or a 1.*

We will use the calligraphic capital letters $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, ..., $\mathcal{Z}$ for truth assignments. If these are not sufficient then we will add subscripts to these symbols. Now let's look at some truth assignments.

Examples 1.3.2

$$\mathcal{A}[P_i] = \begin{cases} 0 & \text{if } i \text{ is odd} \\ 1 & \text{if } i \text{ is even} \end{cases}$$

$$\mathcal{B}[P_i] = \begin{cases} 0 & \text{if } i \geq 1,000,000 \\ 1 & \text{if } i < 1,000,000 \end{cases}$$

$$\mathcal{C}[P_i] = \begin{cases} 0 & \text{if } i \text{ is prime} \\ 1 & \text{if } i \text{ is not prime} \end{cases}$$

We say that i is prime if $i > 1$ and i is divisible only by 1 and itself.
$\mathcal{D}[P_i] = 1$ for all i.
$\mathcal{E}[P_i] = 0$ for all i.

A truth assignment $\mathcal{T}$ gives values to atomic formulas. We can it to a function $\mathcal{T}^{ext}$, read $\mathcal{T}$ extension, that assigns values to all formulas.

Definition 1.3.3 (Interpretation of formulas) *Rule 1. If F is an atomic formula then $\mathcal{T}^{ext}[F] = \mathcal{T}[F]$.*

Rule 2. If $F = \neg G$, then

$$\mathcal{T}^{ext}[F] = \begin{cases} 0 & \textit{if } \mathcal{T}^{ext}[G] = 1 \\ 1 & \textit{if } \mathcal{T}^{ext}[G] = 0 \end{cases}$$

Rule 3. If $F = (G \vee H)$ then

$$\mathcal{T}^{ext}[F] = \begin{cases} 0 & \textit{if both } \mathcal{T}^{ext}[G] = 0 \textit{ and } \mathcal{T}^{ext}[H] = 0 \\ 1 & \textit{if } \mathcal{T}^{ext}[G] = 1 \textit{ or } \mathcal{T}^{ext}[H] = 1 \end{cases}$$

Rule 4. If $F = (G \wedge H)$ then

$$\mathcal{T}^{ext}[F] = \begin{cases} 0 & \textit{if at least one of } \mathcal{T}^{ext}[G], \mathcal{T}^{ext}[H] \textit{ is equal to 0} \\ 1 & \textit{if both } \mathcal{T}^{ext}[G] = 1 \textit{ and } \mathcal{T}^{ext}[H] = 1 \end{cases}$$

Rule 5. If $F = (G \longrightarrow H)$ then

$$\mathcal{T}^{ext}[F] = \begin{cases} 0 & \textit{if } \mathcal{T}^{ext}[G] = 1 \textit{ and } \mathcal{T}^{ext}[H] = 0 \\ 1 & \textit{if } \mathcal{T}^{ext}[G] = 0 \textit{ or } \mathcal{T}^{ext}[H] = 1 \end{cases}$$

Rule 6. If $F = (G \longleftrightarrow H)$

$$\mathcal{T}^{ext}[F] = \begin{cases} 0 & \textit{if } \mathcal{T}^{ext}[G] \neq \mathcal{T}^{ext}[H] \\ 1 & \textit{if } \mathcal{T}^{ext}[G] = \mathcal{T}^{ext}[H] \end{cases}$$

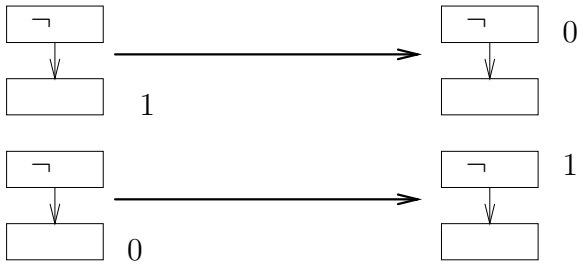


Figure 1.30: The ¬ rewrite rules

The function T^{ext} is well defined because The Unique Readability Theorem from Section 1.2 tells us that only one of the 6 rules apply and the main subformulas, G and H, are unique.

Let us compute $\mathcal{A}^{ext}[(\neg P_3 \wedge P_4)]$, $\mathcal{C}^{ext}[((P_3 \longrightarrow P_5) \longleftrightarrow P_{17})]$, $\mathcal{D}^{ext}[(\neg P_3 \vee P_7)]$, where $\mathcal{A}$, $\mathcal{C}$ and $\mathcal{D}$ are the truth assignments from Examples 1.3.2.

1. $\mathcal{A}^{ext}[P_3] = \mathcal{A}[P_3] = 0$, because 3 is odd. So,
   $\mathcal{A}^{ext}[\neg P_3] = 1$, by rule 2. Also,
   $\mathcal{A}^{ext}[P_4] = \mathcal{A}[P_4] = 1$, because 4 is even. Then,
   $\mathcal{A}^{ext}[(\neg P_3 \wedge P_4)] = 1$, by rule 4.
2. $\mathcal{C}^{ext}[P_3] = \mathcal{C}[P_3] = 0$ because 3 is prime. Then
   $\mathcal{C}^{ext}[(P_3 \longrightarrow P_5)] = 1$ by rule 5. Now
   $\mathcal{C}^{ext}[P_{17}] = \mathcal{C}[P_{17}] = 0$, because 17 is prime. So,
   $\mathcal{C}^{ext}[((P_3 \longrightarrow P_5) \longleftrightarrow P_{17})] = 0$ by rule 6, because
   $\mathcal{C}^{ext}[(P_3 \longrightarrow P_5)] \neq \mathcal{C}^{ext}[P_{17}]$.
3. $\mathcal{D}^{ext}[P_7] = \mathcal{D}[P_7] = 1$, so
   $\mathcal{D}^{ext}[(\neg P_3 \vee P_7)] = 1$ by rule 3.

We see that the process of finding the truth value of a formula is a *bottom up* process, starting atomic formulas and computing truth values for larger and larger subformulas. For this reason, it is easier to work with formula trees. Then, the rules 2-6 of Definition 1.3.3, become the rewrite rules from Figures 1.30-1.34.

Some nodes of the left-hand-side trees of Figures 1.30-1.34 have no labels because their truth values are irrelevant to the evaluation of the root. For the same reason, the children of right-hand-side trees are left unlabeled. These rules allow us to do a *lazy evaluation* of a formula, by computing only some truth assignments.

Example 1.3.4 Let's find the value of the formula from Figure 1.35 for the truth assignment $\mathcal{A}$ from Examples 1.3.2.

Since we need to specify where the rules are applied, we inserted the Dewey addresses to the left of the nodes. First, $\mathcal{A}$ assigns values to the atomic formulas, 0 if the index is odd and 1 if the index is even. We get the tree from Figure 1.36, with the truth value to the *right* of the node.

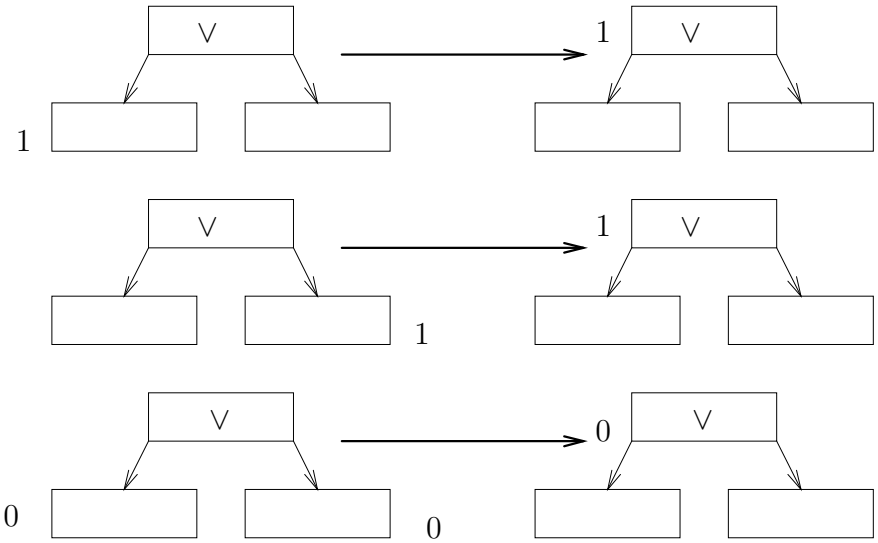


Figure 1.31: The $\vee$ rewrite rules

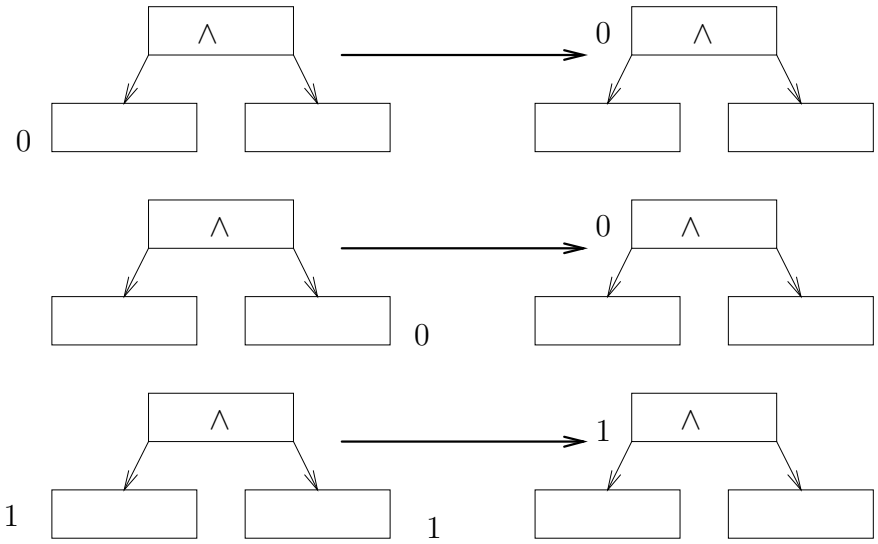


Figure 1.32: The $\wedge$ rewrite rules

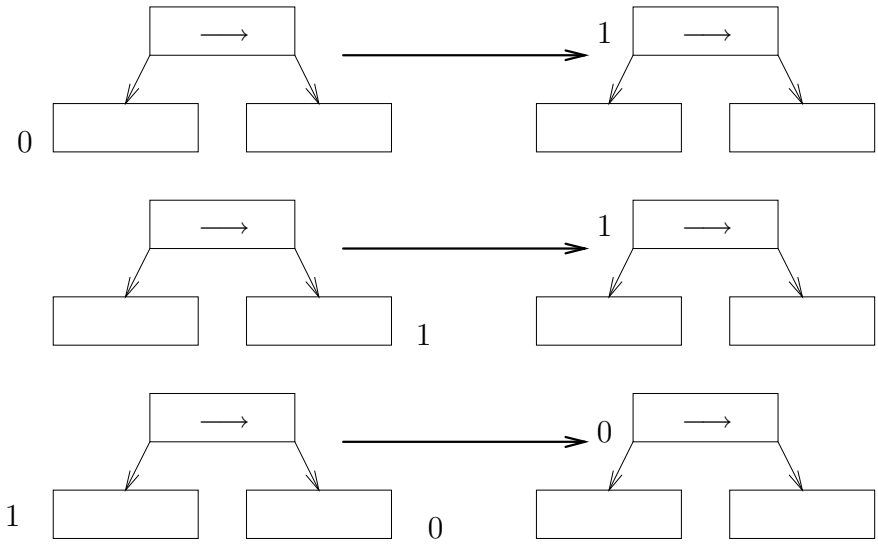


Figure 1.33: The ⟶ rewrite rules

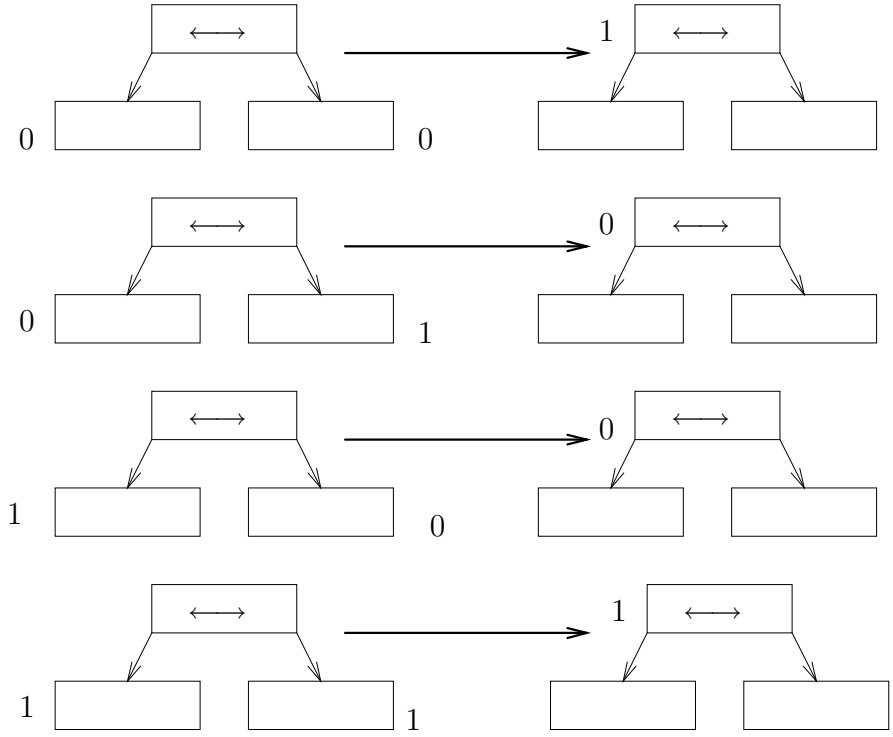


Figure 1.34: The ⟷ rewrite rules

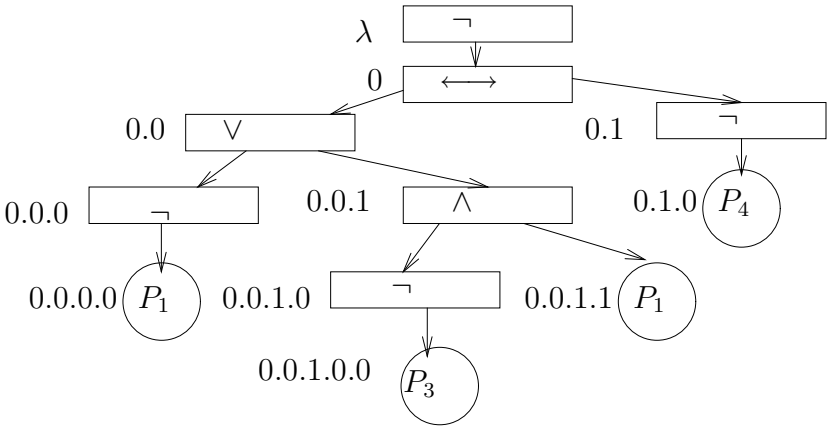


Figure 1.35: The formula tree for Example 1.3.4

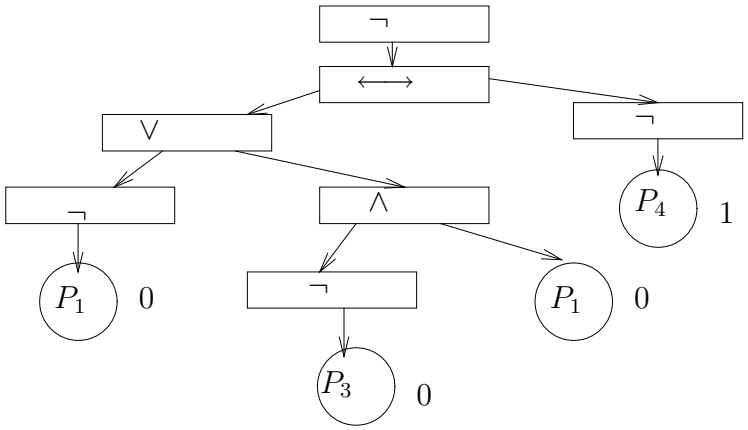


Figure 1.36: The atoms of the Tree 1.3.4 with truth values

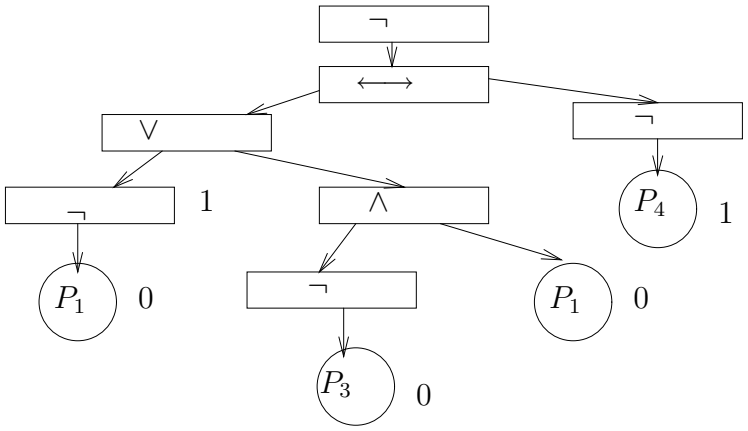


Figure 1.37: The Tree 1.3.4 after evaluating 0.0.0

Now let us apply rule 2, to address 0.0.0. We get the tree from Figure 1.37. Figure 1.38 shows the tree after we used rule 3 (the $\vee$ rule) at address 0.0.

Now we apply rule 2,the $\neg$ rule, at address 0.1 and get Figure 1.39. Further on, we use rule 6, the $\longleftrightarrow$ rule, to obtain Figure 1.40. Finally, we apply the negation rule, 2, at λ and we label the root, as shown in Figure 1.41. The truth value of this formula for $\mathcal{A}$ is 1. We did a lazy evaluation since we did not compute the truth values of all nodes.

What happens if we want to compute the truth value of a formula without converting it to a tree? The examples that followed Definition 1.3.3, showed that the evaluation process is cumbersome, so we need a better tool. For this reason we introduce the operations $\boxed{\neg}$, $\boxed{\vee}$, $\boxed{\wedge}$, $\boxed{\longrightarrow}$, $\boxed{\longleftrightarrow}$ that define the semantics of the connectives $\neg$, $\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$. We box the connectives to show that they are operations on the set $\{0,1\}$. We use Definition 1.3.3 to construct the tables for the box operations.

x	$\boxed{\neg}$x
0	1
1	0

The table for $\boxed{\neg}$

x	y	x $\boxed{\vee}$ y	x $\boxed{\wedge}$ y	x $\boxed{\longrightarrow}$ y	x $\boxed{\longleftrightarrow}$ y
0	0	0	0	1	1
0	1	1	0	1	0
1	0	1	0	0	0
1	1	1	1	1	1

The tables for $\boxed{\vee}$, $\boxed{\wedge}$, $\boxed{\longrightarrow}$, and $\boxed{\longleftrightarrow}$

Now we rewrite Definition 1.3.5 using the box operations.

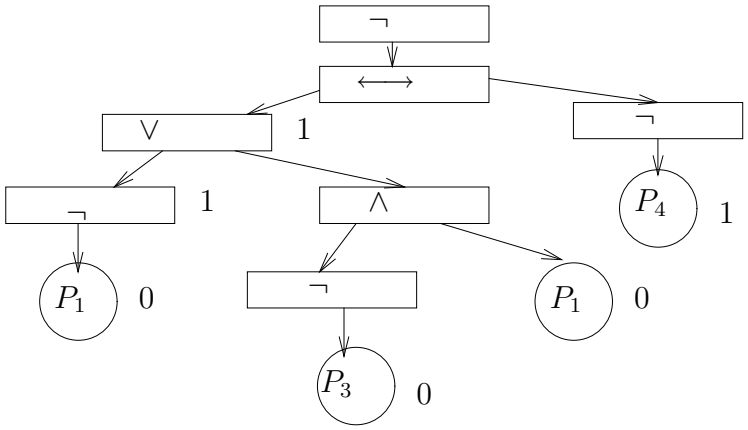


Figure 1.38: The Tree 1.3.4 after evaluating 0.0

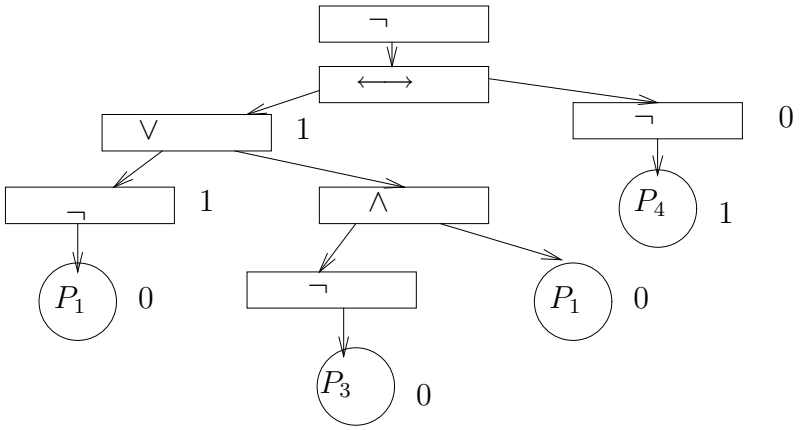


Figure 1.39: The Tree 1.3.4 after evaluating 0.1

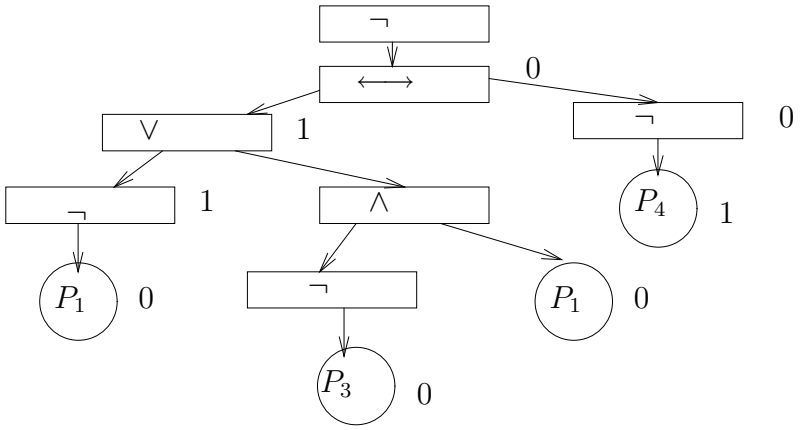


Figure 1.40: The Tree 1.3.4 after evaluating 0

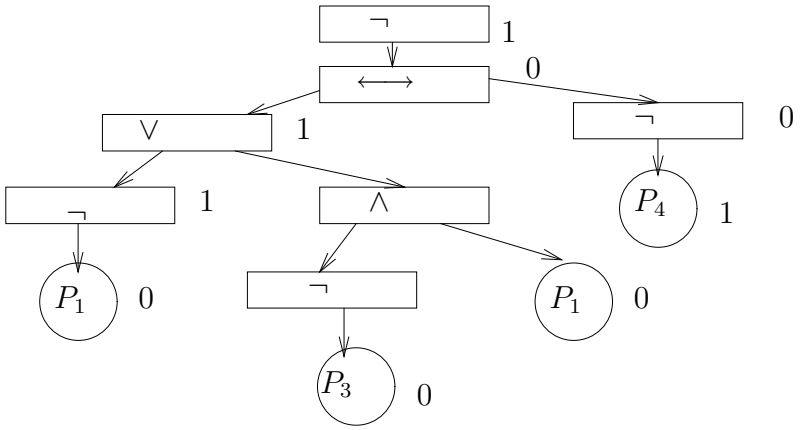


Figure 1.41: The Tree 1.3.4 after evaluating λ

Figure 1.42: The alternation of marked and unmarked substrings

Definition 1.3.5 (the truth assignment extension) *Let $\mathcal{T}$ be a truth assignment. Then $\mathcal{T}^{ext}$ is defined as follows:*
1. If F is an atomic formula then $\mathcal{T}^{ext}[F] = \mathcal{T}[F]$.
2. If $F = \neg G$ then $\mathcal{T}^{ext}[F] = \boxed{\neg}\mathcal{T}^{ext}[G]$.
3. If $F = GCH$ where C is one of the connectives $\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$, then $\mathcal{T}^{ext}[F] = \mathcal{T}^{ext}[G]\boxed{C}\,\mathcal{T}^{ext}[H]$.

Definition 1.3.6 (agreement on F) *We say that the truth assignments $\mathcal{A}$ and $\mathcal{B}$ agree on a formula F , if for all atoms P in F, $\mathcal{A}[P] = \mathcal{B}[P]$.*

Theorem 1.3.7 (The Agreement Theorem) *If $\mathcal{A}$ and $\mathcal{B}$ agree on F then $\mathcal{A}^{ext}[F] = \mathcal{B}^{ext}[F]$.*

Proof: The theorem says that this property is true for *all* truth values and for *all* formulas. So, we'll prove it by structural induction.
Case 1: F is an atom, say Q. Let $\mathcal{A}$ and $\mathcal{B}$ be two truth assignments that agree on F. Then $\mathcal{A}[Q] = \mathcal{B}[Q]$ since $\mathcal{A}$ and $\mathcal{B}$ agree on the atoms of F. Since $\mathcal{A}^{ext}[Q] = \mathcal{A}[Q]$ and $\mathcal{B}^{ext}[Q] = \mathcal{B}[Q]$, we get $\mathcal{A}^{ext}[F] = \mathcal{B}^{ext}[F]$.
Case 2: $F = \neg G$. Let $\mathcal{A}$ and $\mathcal{B}$ be truth assignments that agree on F. Then they agree on G, since G has the same atoms as F. By the induction hypothesis, $\mathcal{A}^{ext}[G] = \mathcal{B}^{ext}[G]$. Then $\mathcal{A}^{ext}[F] = \boxed{\neg}[\mathcal{A}^{ext}[G]] = \boxed{\neg}[\mathcal{B}^{ext}[G]] = \mathcal{B}^{ext}[F]$.
Cases 3, 4, 5, 6: $F = GCH$, where G, H are formulas and C is one of the connectives $\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$.

Let $\mathcal{A}$, $\mathcal{B}$ be two truth assignments that agree on F. Then, they agree on both G and H. By induction hypothesis, $\mathcal{A}^{ext}[G] = \mathcal{B}^{ext}[G]$ and $\mathcal{A}^{ext}[H] = \mathcal{B}^{ext}[H]$.

But,
$\mathcal{A}^{ext}[F]$
$= \mathcal{A}^{ext}[G]\boxed{C}\mathcal{A}^{ext}[H]$ by the interpretation of C
$= \mathcal{B}^{ext}[G]\boxed{C}\mathcal{B}^{ext}[H]$ by the induction hypotheses
$= \mathcal{B}^{ext}[F]$ by the interpretation of C. **Q.E.D.**

Figure 1.43 shows an algorithm for finding $\mathcal{T}^{ext}[F]$. It is based on Algorithm 1.1 that recognizes formulas. However, instead of underlining the symbols of F, we mark them with underlines, 0, 1, underlined 0, and underlined 1. Only the atoms and the connectives are marked with 0's and 1's; the parentheses are only underlined. Now, let us explain why some truth values are underlined.

During the execution of the algorithm, the formula F consists of an alternation of marked and unmarked substrings, as shown in Figure 1.42. There the marked substrings are represented by arrow ended segments.

In the preceding section we called the marked substrings max-occurrences and we showed that they are formulas. A max-occurrence can have several symbols marked with 0's and 1's, but only one of them is underlined. That

Step 1. Make a table having as headers the symbols of F. Under each atom P insert the underlined value of the atom.
Step 2. loop until F is fully marked
for each index i in $\{0, \ldots, |S|-1\}$ do
{
if ($S[i] = \neg$ and $S[i+1]$ is marked) then
{
find V, the first underlined truth value to the right of $\neg$;
erase the underline under V;
write the underlined value $\boxed{\neg}[V]$ under $\neg$;
}
if ((GCH) is an occurrence at i, G and H are marked and
the parentheses and C are unmarked) then
{
find U, the underlined value that precedes C;
erase the U-underline;
find V, the underlined value that succeeds C;
erase the V-underline;
underline the end parentheses of (GCH);
write $U\boxed{C}V$ under C and underline it;
}
}

Figure 1.43: Truth table for one assignment

number is the value of the max-occurrence. When the algorithm underlines a negation or a binary connective, the max-occurrences change as shown in Figure 1.28 and Figure 1.29. The value of $\neg G$ is computed from the value of G using the $\boxed{\neg}$ table. It is put under $\neg$ and underlined, while the value of G looses its underline. The value of (GCH) is computed from the underlined values of G and H. It is put under C and underlined while the values of G and H loose their underlines. This way we ensure that every max-occurrence has one and only one underlined truth value. In the end the whole formula is marked and its value is the underlined truth value. Let's apply Algorithm 1.43 to find $\mathcal{A}^{ext}[(\neg(P_1 \longrightarrow P_2) \longleftrightarrow (P_3 \wedge (P_1 \vee P_4)))]$, where $\mathcal{A}$ is the assignment from Examples 1.3.2.

At Step 1 we put the truth values under the atoms, 0 if the index is odd and 1 if the index if even, and we underline them.

(	¬	(	P_1	$\longrightarrow$	P_2	)	$\longleftrightarrow$	(	P_3	$\wedge$	(	P_1	$\vee$	P_4	)	)	)
			$\underline{0}$		$\underline{1}$				$\underline{0}$			$\underline{0}$		$\underline{1}$			

The next scan assigns values to the connectives $\longrightarrow$ and $\vee$:

(	¬	(	P_1	$\longrightarrow$	P_2	)	$\longleftrightarrow$	(	P_3	$\wedge$	(	P_1	$\vee$	P_4	)	)	)
		$\underline{\ }$	0	$\underline{1}$	1	$\underline{\ }$			$\underline{0}$		$\underline{\ }$	0	$\underline{1}$	1	$\underline{\ }$		

The formula is not marked, so we repeat Step 2. We assign values to the subformulas with root $\neg$ and $\wedge$.

(	¬	(	P_1	$\longrightarrow$	P_2	)	$\longleftrightarrow$	(	P_3	$\wedge$	(	P_1	$\vee$	P_4	)	)	)
	$\underline{0}$	_	0	1	1	_		_	0	$\underline{0}$	_	0	1	1	_	_	

Next, we assign a truth value to $\longleftrightarrow$.

(	¬	(	P_1	$\longrightarrow$	P_2	)	$\longleftrightarrow$	(	P_3	$\wedge$	(	P_1	$\vee$	P_4	)	)	)
_	0	_	0	1	1	_	$\underline{1}$	_	0	0	_	0	1	1	_	_	_

Now the whole string is underlined. The truth value of the formula is the underlined number, in our case 1.

Exercises

Exercise 1.3.1 *Show that the set of truth assignments is not countable. This means that no procedure can enumerate all truth assignments.*

Exercise 1.3.2 *Compute the following truth values:*

1. $\mathcal{A}^{ext}[(P_2 \longrightarrow P_3) \vee (P_5 \wedge P_8)]$
2. $\mathcal{B}^{ext}[P_{1000000} \longleftrightarrow (P_{100000000} \wedge P_3)]$
3. $\mathcal{C}^{ext}[(\neg P_5 \wedge P_8) \longrightarrow P_{11}]$
4. $\mathcal{D}^{ext}[P_{11} \longleftrightarrow \neg P_{13}]$
5. $\mathcal{E}^{ext}[\neg(P_6 \longleftrightarrow P_5)]$

where $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$ are the truth assignments from Examples 1.3.2.

Exercise 1.3.3 *Construct the tree representation for the formulas listed below. Then use the rewrite rules described in this section to find their truth value under the truth assignment $\mathcal{A}$ from Examples 1.3.2.*

1. $\neg(P_3 \longleftrightarrow P_2) \longleftrightarrow (P_3 \vee (P_{10} \wedge P_3))$
2. $\neg(P_{12} \wedge ((P_3 \longleftrightarrow P_5) \longrightarrow (P_3 \vee P_{11})))$
3. $(((P_{10} \longrightarrow P_9) \longrightarrow (P_6 \wedge P_7)) \wedge (P_8 \longrightarrow P_{11}))$

Exercise 1.3.4 *Prove, by structural induction on F, that Definitions 1.3.3 and 1.3.5 define the same function.*

Exercise 1.3.5 *Use the operations $\boxed{\neg}$, $\boxed{\vee}$, $\boxed{\wedge}$, $\boxed{\longrightarrow}$, $\boxed{\longleftrightarrow}$ to evaluate the following truth values:*

1. $\mathcal{A}^{ext}[(P_{11} \vee P_{14}) \longrightarrow (P_{12} \wedge (P_3 \longleftrightarrow P_{12}))]$
2. $\mathcal{C}^{ext}[\neg(P_{12} \longrightarrow P_3) \longrightarrow (P_{12} \vee \neg P_4)]$
3. $\mathcal{C}^{ext}[(P_4 \longrightarrow \neg(P_4 \vee P_5)) \longleftrightarrow (P_4 \vee P_7)]$

Here, $\mathcal{A}$ and $\mathcal{C}$ are the truth assignments defined in Examples 1.3.2.

Exercise 1.3.6 *Let F be a formula. On the set of truth assignments we define the relation $\equiv_F$ as*

$\mathcal{A} \equiv_F \mathcal{B}$ iff $\mathcal{A}[F] = \mathcal{B}[F]$.

a. Show that $\equiv_F$ is an equivalence relation.

b. Prove that if $\mathcal{A} \equiv_F \mathcal{B}$ then $\mathcal{A} \equiv_{\neg F} \mathcal{B}$.

c. Prove that if $\mathcal{A} \equiv_G \mathcal{B}$ and $\mathcal{A} \equiv_H \mathcal{B}$ then $\mathcal{A} \equiv_{(GCH)} \mathcal{B}$, where C is one of the connectives $\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$.

Exercise 1.3.7 *Redo Exercise 1.3.3 using truth tables.*

1.4 Models, Satisfiability, Tautology, Consequence

This section deals with the behaviour of a formula in the set of all truth assignments. Some formulas are always true (the tautologies), some formulas are sometimes true and sometimes false, and some formulas are always false (the unsatisfiable formulas). The section gives algorithms for testing the properties of satisfiability, tautology, and unsatisfiability. It also presents the notion of consequence and relates this notion to tautologies.

Starting with this section we will not differentiate between the truth assignment $\mathcal{T}$ and its extension $\mathcal{T}^{ext}$. So, we will simplify the notation and write $\mathcal{T}[F]$ instead of $\mathcal{T}^{ext}[F]$.

Definition 1.4.1 (model, countermodel, contradiction, tautology) *1. Let $\mathcal{T}$ be a truth assignment. If $\mathcal{T}[F] = 1$ then we say that $\mathcal{T}$ is a model of/for F, and write $\models_{\mathcal{T}} F$. If $\mathcal{T}[F] = 0$ then we say that $\mathcal{T}$ is a countermodel of/for F and we write $\not\models_{\mathcal{T}} F$.*

2. If F has a model, we say that F is satisfiable.

3. If F has no model, F is called a contradiction or an unsatisfiable formula. We write $\not\models F$ to show that F is unsatisfiable.

4. If every truth assignment is a model for F, we say that F is a tautology. We write $\models F$ to indicate that F is a tautology.

How can we find out if a formula F is satisfiable?

The off-the-cuff answer is to try the first truth assignment. If it is a model for the formula then we stop. If not, we try the second truth assignment and check if it is a model. If yes, we stop with success. Otherwise we get the next truth assignment and repeat the checking. If we go through all truth assignments and none is a model, the formula is unsatisfiable.

This algorithm has more holes than a colander. First of all, each truth assignment is a function with an infinite domain (the set of all atomic formulas), so we have the problem of specifying the truth assignments. Then, the set of truth assignments is infinite, so we have the second problem, of listing the truth assignments.

However, we are rescued by The Agreement Theorem. This theorem says that we can find the value of the formula F when we know the values of its atoms. So, all that we need are the restrictions of the truth assignments to the set of atoms. But how many different restrictions do we have?

The answer is simple. If F has n atoms, there are 2^n restrictions because each atom can be either 0 or 1. So, all truth assignments fall into one of these 2^n cases, according to the restriction of the truth assignment to the atoms of F.

This observation leads us to the algorithm shown in Figure 1.44. Now let's do an example.

Example 1.4.2 Let's see if $F = (P_1 \wedge P_2)$ is satisfiable.
At Step 1 we form the header.

P_1	P_2	(	P_1	$\wedge$	P_2	)

Step 1. identify the atoms of F, say P_1, ..., P_n; put each atom as a column head; insert a double bar, ||, after the last atom and write the formula F the way we did it in Section 1.3;
Step 2. initialize row-number to 0;
Step 3. while (row-number $< 2^n$) do
{
write row-number in binary under the columns P_1, P_2, ..., P_n;
find the value of F using the algorithm given in Section 1.3;
if (F has the value 1) then exit with success;
else increase row-number by 1;
}
Step 4. exit with failure;

Figure 1.44: The Satisfiability Algorithm

At Step 2 we initialize row-number to 0.
At Step 3 we test if 0 is less than $2^2 = 4$. It is, so we execute the loop body. We write 0 in binary as 00. So both P_1 and P_2 get the truth value 0. We evaluate $P_1 \wedge P_2$ using the truth table and get the table below.

P_1	P_2	(	P_1	$\wedge$	P_2	)
0	0	_	0	0	0	_

F evaluates to 0, so we increase row-number to 1 and check if it is less than $2^2 = 4$. It is, so we repeat the loop. We write 1 in binary as 01. Then P_1 gets the first digit of 01, and P_2 gets the second digit of 01. We evaluate F for this truth assignment and get the second line of the table.

P_1	P_2	(	P_1	$\wedge$	P_2	)
0	0	_	0	0	0	_
0	1	_	0	0	1	_

Again F evaluates to 0. So, we increase row-number to 2 and check if it is less than 4. It is, so we write 2 in binary. We get 10. Then P_1 gets 1 and P_2 0. We construct the 3rd line of the truth table.

P_1	P_2	(	P_1	$\wedge$	P_2	)
0	0	_	0	0	0	_
0	1	_	0	0	1	_
1	0	_	1	0	0	_

Again the truth value of F is 0, so we increase row-number to 3. 3 is less than 4 so we execute the loop body. We get the last line of the table below.

P_1	P_2	(	P_1	$\wedge$	P_2	)
0	0	_	0	0	0	_
0	1	_	0	0	1	_
1	0	_	1	0	0	_
1	1	_	1	1	1	_

At this time F evaluates to 1, so we stop with success. F is satisfiable and any truth assignment $\mathcal{T}$ with $\mathcal{T}[P_1] = 1$ and $\mathcal{T}[P_2] = 1$ is a model for F.

Note 1.4.3 Since the parentheses will always be marked with underscores, we

Step 1. identify the atoms of F, say P_1, ..., P_n; put each atom as a column head; insert a double bar, ||, after the last atom and write the formula F the way we did it in Section 1.3;
Step 2. initialize row-number to 0;
Step 3. while (row-number $< 2^n$) do
{
write row-number in binary under the columns P_1, P_2, ..., P_n;
find the value of F using the algorithm given in Section 1.3;
if (F has the value 1) then increase row-number by 1;
else exit with failure;
}
Step 4. exit with success;

Figure 1.45: The Tautology Recognizer

will not write them in separate columns. We will also omit the outer parentheses. The ('s will be in the column of the first atom or connective that succeeds them and the)'s will be joined to the preceding atom. This way we can focus on the way the truth values of the components determine the truth value of the formula.

We can modify Algorithm 1.44 to get the tautology recognizing procedure from Figure 1.45. Let us do an example.

Example 1.4.4 We want to check if $P_1 \vee \neg P_2$ is a tautology. We follow the conventions from Note 1.4.3 and omit the outer parentheses.
At Step 1 we form the header.

P_1	P_2	P_1	$\vee$	$\neg$	P_2

At Step 2 we initialize row-number to 0.
At step 3 we check if row number is $< 2^2 = 4$. It is, so we execute the loop body. We write 0 in binary as 00. Then, both P_1 and P_2 get the value 0. We evaluate $P_1 \vee \neg P_2$ and get the table below.

P_1	P_2	P_1	$\vee$	$\neg$	P_2
0	0	0	$\underline{1}$	1	0

F evaluates to 1, so we increase row-number to 1 and go to Step 3. Since row-number is less than 4, we execute the loop body. We write the row number 1 in binary as 01. So, P_1 gets 0 and P_2 gets 1. We evaluate $P_1 \vee \neg P_2$ and enter its value in the second row of the table.

P_1	P_2	P_1	$\vee$	$\neg$	P_2
0	0	0	$\underline{1}$	1	0
0	1	0	$\underline{0}$	0	1

Now F evaluates to 0, so we stop with failure. F is not a tautology and any truth assignment $\mathcal{T}$ with $\mathcal{T}[P_1] = 0$ and $\mathcal{T}[P_2] = 1$ is a countermodel of F.

We can go a step further and modify Algorithm 1.44 to get an unsatisfiability testing procedure. All we have to do is to exchange the words success and failure in Algorithm 1.44 and we get Figure 1.46.

Step 1. identify the atoms of F, say P_1, ..., P_n; put each atom as a column head; insert a double bar, ||, after the last atom and write the formula F the way we did it in Section 1.3;
Step 2. initialize row-number to 0;
Step 3. while (row-number $< 2^n$) do
{
write the binary digits of row-number under the columns P_1, P_2, ..., P_n;
find the value of F using the algorithm given in Section 1.3;
if (F has the value 1) then exit with failure;
else increase row-number by 1;
}
Step 4. exit with success;

Figure 1.46: The Unsatisfiability Algorithm

Definition 1.4.5 (consequence) *1. We say that a truth assignment $\mathcal{T}$ satisfies a set of formulas if it satisfies every formula in the set.*

2. We say that a formula G is a consequence of a set of formulas S, written $S \models G$, if every model of S is a model of G. If S is the finite set $\{F_1, \ldots, F_n\}$ we omit the braces and write $F_1 \ldots F_n \models G$ instead of $\{F_1, \ldots, F_n\} \models G$.

Note 1.4.6 The symbol $\models$ can mean both *tautology* and *consequence*. If there is no symbol in front of the sign then its meaning is *tautology*; otherwise it reads *consequence*.

Examples 1.4.7 1. Let $S = \{P_0, P_1 \longrightarrow P_0, P_0 \vee \neg P_1\}$ and $\mathcal{A}$ be a truth assignment with $\mathcal{A}[P_0] = 1$. Then $\mathcal{A}$ is a model of the set S, because it assigns 1 to every formula in the set.

2. Let us check that $F \vee G$ is a consequence of F. Assume that the truth assignment $\mathcal{T}$ is model of F. Then $\mathcal{T}[F] = 1$. Let us find $\mathcal{T}[F \vee G]$.

$\mathcal{T}[F \vee G] = \mathcal{T}[F] \boxed{\vee} \mathcal{T}[G] = 1 \boxed{\vee} \mathcal{T}[G] = 1$.

So, $\mathcal{T}$ is a model for $F \vee G$.

Then, by Definition 1.4.5, $F \models (F \vee G)$.

Definition 1.4.8 (Con[S],Mod[S]) *Let S be a set of formulas. Con[S] is the set of consequences of S, and Mod[S] is the set of models of S.*

Theorem 1.4.9 *1. If $S \subseteq T$ then Con[S] $\subseteq$Con[T].*
2. If $S \subseteq T$ then Mod[T] $\subseteq$Mod[S].
3. $S \subseteq$Con[S]
4. If T is a tautology, $T \in$Con[S].
5. Con[Con[S]] =Con[S].
6. Mod[S] =Mod[Con[S]].
7. $F_1, \ldots F_n \models F$ iff $\models \bigwedge_{i=1}^{n} F_i \longrightarrow F$.

Proof: We will prove 7 and leave the rest as exercises. Since 7 is an if and only if statement we will prove the two implications,

if $F_1, \ldots, F_n \models F$ then $\models (F_1 \wedge \ldots \wedge F_n) \longrightarrow F$, and
if $\models (F_1 \wedge \ldots \wedge F_n) \longrightarrow F$ then $F_1, \ldots, F_n \models F$.
$\Longrightarrow$ Assume that F is a consequence of $F_1, F_2, \ldots, F_n$. Let $\mathcal{T}$ be a truth assignment. We need to show that $\mathcal{T}[\bigwedge_{i=1}^n F_i \longrightarrow F] = 1$. We do a proof by cases.

Case 1: $\mathcal{T}[\bigwedge_{i=1}^n F_i] = 0$. Then
$\mathcal{T}[\bigwedge_{i=1}^n F_i \longrightarrow F] = \mathcal{T}[\bigwedge_{i=1}^n F_i] \boxed{\longrightarrow} \mathcal{T}[F] = 0 \boxed{\longrightarrow} \mathcal{T}[F] = 1$.

Case 2: $\mathcal{T}[\bigwedge_{i=1}^n F_i] = 1$. We recall that $\bigwedge_{i=1}^n F_i$ stands for $(\ldots((F_1 \wedge F_2) \wedge F_3) \wedge \ldots F_n)$. We can prove by induction on n that $\mathcal{T}[(\ldots((F_1 \wedge F_2) \wedge F_3) \wedge \ldots F_n)] = 1$ iff $\mathcal{T}[F_i] = 1$ for *every* $1 \leq i \leq n$.

So, $\mathcal{T}$ is a model of $F_1, \ldots, F_n$. Since F is a consequence of $F_1, F_2, \ldots, F_n$, $\mathcal{T}$ is a model of F, i. e. $\mathcal{T}[F] = 1$. Then $\mathcal{T}[\bigwedge_{i=1}^n F_i \longrightarrow F] = 1$ from the truth table of $\boxed{\longrightarrow}$.

Since in both cases we got $\mathcal{T}[\bigwedge_{i=1}^n F_i \longrightarrow F] = 1$ we are done.
$\Longleftarrow$ Assume that $\models \bigwedge_{i=1}^n F_i \longrightarrow F$. We need to show that $F_1, \ldots, F_n \models F$.

Let $\mathcal{T}$ be a model of $\{F_1, \ldots, F_n\}$. Then $\mathcal{T}[F_1] = \ldots = \mathcal{T}[F_n] = 1$. By the previous discussion,
(1) $\mathcal{T}[\bigwedge_{i=1}^n F_i] = 1$.
Since $\bigwedge_{i=1}^n F_i \longrightarrow F$ is a tautology,
(2) $\mathcal{T}[\bigwedge_{i=1}^n F_i \longrightarrow F] = 1$.
Now, $1 = \mathcal{T}[\bigwedge_{i=1}^n F_i \longrightarrow F]$
$= \mathcal{T}[\bigwedge_{i=1}^n F_i] \boxed{\longrightarrow} \mathcal{T}[F]$ by the interpretation of $\longrightarrow$
$= 1 \boxed{\longrightarrow} \mathcal{T}[F]$ by (1)
$= \mathcal{T}[F]$ from the table of $\longrightarrow$

So, $\mathcal{T}[F] = 1$. Since $\mathcal{T}$ is any truth assignment, $F_1, \ldots, F_n \models F$. **Q.E.D.**
Part 7 of Theorem 1.4.9 provides a method for checking consequences; all we have to do is to check that $\bigwedge_{i=1}^n F_i \longrightarrow F$ is a tautology.

Example 1.4.10 Let us check that $F \longrightarrow G, F \longrightarrow \neg G \models \neg F$.

We have to check that $((F \longrightarrow G) \wedge (F \longrightarrow \neg G)) \longrightarrow \neg F$ is a tautology. We apply Algorithm 1.45 and treat the meta-variables F and G as atomic formulas. We go through the algorithm and get the table below.

F	G	$((F$	$\longrightarrow$	$G)$	$\wedge$	$(F$	$\longrightarrow$	$\neg$	$G))$	$\longrightarrow$	$\neg$	F
0	0	0	1	0	1	0	1	1	0	$\underline{1}$	1	0
0	1	0	1	1	1	0	1	0	1	$\underline{1}$	1	0
1	0	1	0	0	0	1	1	1	0	$\underline{1}$	0	1
1	1	1	1	1	0	1	0	0	1	$\underline{1}$	0	1

For all truth assignments, the truth value of the formula is 1, so, the implication is a tautology. Then $F \longrightarrow G, F \longrightarrow \neg G \models \neg F$.

Now, let us see if the properties satisfiability, tautology, and unsatisfiability are preserved by the connectives.

Example 1.4.11 Let us check whether the following statements are true or false:

1. If F and G are tautologies, then so is $F \wedge G$.
2. If F and G are satisfiable, then $F \wedge G$ is satisfiable.

3. If F and G are unsatisfiable, then $F \vee G$ is unsatisfiable.

Solution: The general idea is to give a proof if the statement is true and a counter-example when the statement is false.

1. The statement 1 is true.

Let $\mathcal{T}$ be a truth assignment. We need to show that $\mathcal{T}[F \wedge G]$ is true.

Since F and G are tautologies, $\mathcal{T}[F] = \mathcal{T}[G] = 1$. So, $\mathcal{T}[F \wedge G] = \mathcal{T}[F] \boxed{\wedge} \mathcal{T}[G] = 1 \boxed{\wedge} 1 = 1$.

2. The statement 2 is false.

Let's choose $F = P_1$ and $G = \neg P_1$. Both F and G are satisfiable because every truth assignment $\mathcal{A}$ with $\mathcal{A}[P_1] = 1$ is a model for F, and any truth assignment $\mathcal{B}$ with $\mathcal{B}[P_1] = 0$ is a model for G.

But let us look at the truth table for $P_1 \wedge \neg P_1$.

P_1	P_1	$\wedge$	$\neg$	P_1
0	0	$\underline{0}$	1	0
1	1	$\underline{0}$	0	1

We see that the values of $P_1 \wedge \neg P_1$ (the underlined numbers in the table) are all 0's, so $P_1 \wedge \neg P_1$ is a contradiction.

3. The statement 3 is true.

Let $\mathcal{T}$ be a truth assignment. We need to show that $\mathcal{T}[F \vee G] = 0$.

We know that $\mathcal{T}[F \vee G] = \mathcal{T}[F] \boxed{\vee} \mathcal{T}[G]$. Since F and G are unsatisfiable, $\mathcal{T}[F] = \mathcal{T}[G] = 0$. So,

$\mathcal{T}[F] \boxed{\vee} \mathcal{T}[G] = 0 \boxed{\vee} 0 = 0$.

Exercises

Exercise 1.4.1 *Use the algorithms given in this section to clasify the formulas below as tautologies, unsatisfiable, and satisfiable but not tautologies.*

1. $\neg(P_1 \vee \neg P_1)$
2. $(P_1 \vee P_2) \vee (\neg P_1 \wedge \neg P_2)$
3. $(P_1 \longrightarrow (P_2 \longrightarrow P_3)) \longrightarrow ((P_1 \longrightarrow P_2) \longrightarrow (P_1 \longrightarrow P_3))$
4. $(P_1 \longleftrightarrow P_2) \wedge (P_1 \longleftrightarrow \neg P_2)$
5. $(\neg P_1 \vee \neg P_2) \wedge (P_1 \vee \neg P_2)$

Exercise 1.4.2 *Prove that if F is a tautology then $\neg F$ is unsatisfiable.*

Exercise 1.4.3 *Provide proofs for the first 6 parts of Theorem 1.4.9.*

Exercise 1.4.4 *Using Example 1.4.7 as a model, prove the following consequences:*

1. $F \longrightarrow G, G \longrightarrow H \models (F \longrightarrow H)$
2. $F \vee G, \neg G \models F$
3. $F \longrightarrow G, F \models G$

Exercise 1.4.5 *We define the new connectives $\downarrow$, $\uparrow$, and $\updownarrow$ as follows:*

$F \downarrow G = \neg(F \vee G)$

$F \uparrow G = \neg(F \wedge G)$

$F \updownarrow G = \neg F \wedge G$

Use the $\boxed{\neg}$, $\boxed{\vee}$, $\boxed{\wedge}$ *tables, to construct tables* $\boxed{\uparrow}$, $\boxed{\downarrow}$, $\boxed{\updownarrow}$. *The connective* $\downarrow$ *is called nor (not or) and the connective* $\uparrow$ *is called nand (not and).*

Exercise 1.4.6 *Prove that if* F *is a consequence of* $\bigwedge_{i=1}^{n} F_i$ *then* $\bigwedge_{i=1}^{n} F_i \wedge \neg F$ *is unsatisfiable.*

Exercise 1.4.7 *Prove that if* F *and* G *are satisfiable and* F *and* G *have no atoms in common then* $F \wedge G$ *is satisfiable.*

Exercise 1.4.8 *Let* S *be a set of formulas and* F *be a formula such that both* $S \cup \{F\}$ *and* $S \cup \{\neg F\}$ *are unsatisfiable. Show that* S *is unsatisfiable.*

Exercise 1.4.9 *If* $F \longrightarrow G$ *is unsatisfiable what can you say about* F *and* G*?*

Exercise 1.4.10 *Prove, by induction on* n*, that* $\mathcal{A}[\bigwedge_{i=1}^{n} F_i] = 1$ *implies that for all* $1 \leq i \leq n$ $\mathcal{A}[F_i] = 1$.

Exercise 1.4.11 *Prove or disprove: If* $F \longrightarrow G$ *and* $G \longrightarrow H$ *are tautologies then* $F \longrightarrow H$ *is a tautology.*

Exercise 1.4.12 *Prove or disprove: If* $F \longrightarrow G$ *and* $G \longrightarrow H$ *are satisfiable,* $F \longrightarrow H$ *is satisfiable.*

Exercise 1.4.13 *Prove or disprove: If* $F \longrightarrow G$ *is a tautology and* H *is a formula then* $(F \wedge H) \longrightarrow (G \wedge H)$ *is a tautology.*

Exercise 1.4.14 *Prove or disprove: If* $F \longrightarrow G$ *and* $H \longrightarrow I$ *are satisfiable then* $(F \wedge H) \longrightarrow (G \wedge I)$ *is satisfiable.*

Exercise 1.4.15 *Prove or disprove: If* $F \longrightarrow G$ *is satisfiable, then* $(F \vee H) \longrightarrow (G \vee H)$ *is satisfiable.*

Exercise 1.4.16 *Prove or disprove: If* $F \vee G$ *is a tautology and* F *and* G *have no atoms in common, at least one of them must be a tautology.*

Exercise 1.4.17 *Prove or disprove: If* $F \wedge G$, $G \wedge H$, *and* $H \wedge F$ *are satisfiable, then* $F \wedge (G \wedge H)$ *is satisfiable.*

Exercise 1.4.18 *Prove or disprove: If* $F \vee G$ *and* $\neg G$ *are tautologies, then* F *is tautology.*

Exercise 1.4.19 *Prove or disprove: If* $F \vee G$ *and* $\neg G$ *are satisfiable and* F *and* G *have no atoms in common, then* F *is satisfiable.*

Exercise 1.4.20 *Prove or disprove: Let* F,G,H,I *be 4 formulas such that* $F \wedge (G \wedge H)$, $F \wedge (G \wedge I)$, $F \wedge (H \wedge I)$, *and* $G \wedge (H \wedge I)$ *are satisfiable. Then* $(F \wedge G) \wedge (H \wedge I)$ *is satisfiable.*

Exercise 1.4.21 *Prove or disprove: If* $F \longrightarrow H$ *and* $G \longrightarrow H$ *are satisfiable, then* $(F \vee G) \longrightarrow H$ *is satisfiable.*

Exercise 1.4.22 *Prove or disprove: If $F \longrightarrow H$ and $G \longrightarrow H$ are tautologies, then $(F \vee G) \longrightarrow H$ is a tautology.*

Exercise 1.4.23 *If F is unsatisfiable, what are $Con[\{F\}]$ and $Mod[\{F\}]$? Prove your answer.*

Exercise 1.4.24 *Show that for every truth assignment $\mathcal{T}$ there is an infinite set of formulas $S_{\mathcal{T}}$ such that $Mod[S_{\mathcal{T}}] = \{\mathcal{T}\}$.*

1.5 Semantic Equivalences

This section deals with equivalent formulas, i.e. formulas that have the same meaning. It begins with the definition of the equivalence, followed by an equivalence table, and an algorithm for deciding the equivalence of two formulas. Then we prove The Substitution Theorem, which says that the meaning does not change when we replace a subformula by an equivalent subformula. The theorem allows us to treat formulas as algebraic expressions. Then, we introduce the notion of adequate systems of connectives, and we present two such systems.

Definition 1.5.1 (semantic equivalence) *Two formulas F and G are (semantically) equivalent, written $F \equiv G$, if for every truth assignment $\mathcal{T}$, $\mathcal{T}[F] = \mathcal{T}[G]$.*

Table 1.47 presents a list of equivalences.

Note 1.5.2 ($FORM$) We write $FORM$ for the set of all formulas of the propositional logic.

Note 1.5.3 Each line of Table 1.47 consists of a meta-formula, the equivalence sign, and another meta-formula. The meaning of the equivalences is the following: if we assign a formula to each meta-variable of the equivalence, the left side instance is equivalent to the right side instance. For example, line 3a contains the meta-variable F, that can take any value in the set of formulas $FORM$. If $F = P_3$, we get the equivalence $P_3 \vee P_3 \equiv P_3$; if $F = P_1 \vee \neg P_4$ we get the equivalence $(P_1 \vee \neg P_4) \vee (P_1 \vee \neg P_4) \equiv P_1 \vee \neg P_4$, and so on. Since $FORM$ is infinite, each line represents an infinite set of equivalences. So, the lines of Table 1.47 are actually *meta-equivalences*, that describe an infinite set of equivalences.

The next proposition characterizes the equivalences in terms of tautologies.

Proposition 1.5.4 *$F \equiv G$ iff $\models (F \longleftrightarrow G)$.*

Proof: We need to show the implications
$F \equiv G$ implies $\models (F \longleftrightarrow G)$, and
$\models (F \longleftrightarrow G)$ implies $F \equiv G$.
$\Longrightarrow$: Assume that $F \equiv G$. Let $\mathcal{T}$ be a truth assignment. Then $\mathcal{T}[F] = \mathcal{T}[G]$. Then , from the table of $\boxed{\longleftrightarrow}$, we get that $\mathcal{T}[F \longleftrightarrow G] = 1$. Since the truth assignment $\mathcal{T}$ is arbitrary, $(F \longleftrightarrow G)$ is a tautology.

1. $F \longleftrightarrow G \equiv (F \longrightarrow G) \wedge (G \longrightarrow F)$
2. $F \longrightarrow G \equiv \neg F \vee G$
3. a. $F \vee F \equiv F$ (idempotency of $\vee$)
3. b. $F \wedge F \equiv F$ (idempotency of $\wedge$)
4. a. $F \vee G \equiv G \vee F$ (commutativity of $\vee$)
4. b. $F \wedge G \equiv G \wedge F$ (commutativity of $\wedge$)
5. a. $(F \vee G) \vee H \equiv F \vee (G \vee H)$ (associativity of $\vee$)
5. b. $(F \wedge G) \wedge H \equiv F \wedge (G \wedge H)$ (associativity of $\wedge$)
6. a. $F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$ (distributivity of $\vee$ over $\wedge$)
6. b. $F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$ (distributivity of $\wedge$ over $\vee$)
7. a. $F \vee (F \wedge G) \equiv F$ (absorption)
7. b. $F \wedge (F \vee G) \equiv F$ (absorption)
8. $\neg\neg F \equiv F$ (double negation elimination)
9. a. $\neg(F \vee G) \equiv \neg F \wedge \neg G$ (DeMorgan's law)
9. b. $\neg(F \wedge G) \equiv \neg F \vee \neg G$ (DeMorgan's law)
10. a. $F \vee G \equiv F$ if F is tautology
10. b. $F \vee G \equiv G$ if F is a contradiction
11. a. $F \wedge G \equiv G$ if F is a tautology
11. b. $F \wedge G \equiv F$ if F is a contradiction

Figure 1.47: An equivalence table

$\Longleftarrow$: Assume $\models (F \longleftrightarrow G)$. Now, let $\mathcal{T}$ be a truth assignment. Since $F \longleftrightarrow G$ is a tautology, $\mathcal{T}[F \longleftrightarrow G] = 1$. We also know, from the interpretation of $\longleftrightarrow$, that $\mathcal{T}[F \longleftrightarrow G] = 1$ iff $\mathcal{T}[F] = \mathcal{T}[G]$. So, $\mathcal{T}[F] = \mathcal{T}[G]$. Since $\mathcal{T}$ is any assignment, $F \equiv G$ by Definition 1.5.1. **Q.E.D.**

Proposition 1.5.4 reduces equivalences to tautologies. So, we can modify the tautology recognition algorithm from Figure 1.45 to obtain the Algorithm from Figure 1.48.

Example 1.5.5 Now let's apply the algorithm from Figure 1.48 to verify equivalence 7.b. of Table 1.47.
Step 1. We have only two atoms, F and G. So, we form the header below[5].

F	G	F	$\wedge$	$(F$	$\vee$	$G)$	F

Step 2. We initialize row-number to 0.
Step 3. Since row-number is less than $2^n = 4$, we execute the loop body. We write 0 in binary as 00 and we assign the truth value 0 to both F and G. We evaluate the formulas for these truth values.

F	G	F	$\wedge$	$(F$	$\vee$	$G)$	F
0	0	0	$\underline{0}$	0	0	0	$\underline{0}$

The truth values for the two formulas (the underlined values) are the same, so, we increase row-number to 1.
Step 3. Because row-number is still less than 4, we repeat the loop body. We write the row number 1 as 01 and we assign 0 to F and 1 to G. Then we evaluate

[5] We follow Note 1.4.3 by omitting the outer parenthesis and lumping the other ones with connectives and atoms.

Step 1. We first gather all the atoms of the formulas (we treat the meta-variables as atoms) and put them as column headers. We write the double bar || after the last atom, and then the two formulas, making a column for each symbol and separating the formulas by a double bar. Let $P_1, ..., P_n$ be the atoms of the two formulas.
Step 2. initialize row-number to 0;
Step 3. while (row-number $< 2^n$) do
{
write row-number in binary and assign its digits as truth values to $P_1, ..., P_n$.
evaluate F and G for these values, as done in Section 1.3;
if the truth-values of F and G differ then exit with failure;
else increase row-number by 1;
}
Step 4. exit with success;

Figure 1.48: An equivalence recognizer

the two formulas. We get the table below.

F	G	F	$\wedge$	$(F$	$\vee$	$G)$	F
0	0	0	$\underline{0}$	0	0	0	$\underline{0}$
0	1	0	$\underline{0}$	0	1	1	$\underline{0}$

The two formulas have the same value, so we increase row-number to 2.
Step 3. The value of row-number is still less than 4, so we repeat the loop body. The number 2 is written in binary as 10, so we assign 1 to F and 0 to G. Then we evaluate the formulas.

F	G	F	$\wedge$	$(F$	$\vee$	$G)$	F
0	0	0	$\underline{0}$	0	0	0	$\underline{0}$
0	1	0	$\underline{0}$	0	1	1	$\underline{0}$
1	0	1	$\underline{1}$	1	1	0	$\underline{1}$

The formulas have the same value, so we increase row-number to 3.
Step 3. Since row-number is less than 4, we repeat the loop body. The binary representation of 3 is 11, so we give the value 1 to both F and G. Then we do the evaluation.

F	G	F	$\wedge$	$(F$	$\vee$	$G)$	F
0	0	0	$\underline{0}$	0	0	0	$\underline{0}$
0	1	0	$\underline{0}$	0	1	1	$\underline{0}$
1	0	1	$\underline{1}$	1	1	0	$\underline{1}$
1	1	1	$\underline{1}$	1	1	1	$\underline{1}$

The formulas are the same, so we increase the row number to 4.
Step 3. Now row-number is not less than 4. We go to step 4.
Step 4. We exit with success. The two formulas are equivalent.

Lemma 1.5.6 (The Congruence Lemma) *The semantic equivalence is a congruence for the operations $\neg$, $\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$, i.e. the equivalences below hold for all formulas F,G, H.*

1. $F \equiv G$ *implies* $\neg F \equiv \neg G$
2a. $F \equiv G$ *implies* $F \vee H \equiv G \vee H$
2b. $F \equiv G$ *implies* $H \vee F \equiv H \vee G$
3a. $F \equiv G$ *implies* $F \wedge H \equiv G \wedge H$
3b. $F \equiv G$ *implies* $H \wedge F \equiv H \wedge G$
4a. $F \equiv G$ *implies* $F \longrightarrow H \equiv G \longrightarrow H$
4b. $F \equiv G$ *implies* $H \longrightarrow F \equiv H \longrightarrow G$
5a. $F \equiv G$ *implies* $F \longleftrightarrow H \equiv G \longleftrightarrow H$
5b. $F \equiv G$ *implies* $H \longleftrightarrow F \equiv H \longleftrightarrow G$

Proof: All 9 implications are easy to show. We will show 5a and leave the rest as exercises. Assume that $F \equiv G$ and let $\mathcal{T}$ be a truth assignment. Then

$\mathcal{T}[F \longleftrightarrow H]$
$= \mathcal{T}[F] \boxed{\longleftrightarrow} \mathcal{T}[H]$ by the interpretation of $\longleftrightarrow$
$= \mathcal{T}[G] \boxed{\longleftrightarrow} \mathcal{T}[H]$ since $F \equiv G$, $\mathcal{T}[F] = \mathcal{T}[G]$
$= \mathcal{T}[G \longleftrightarrow H]$ by the interpretation of $\longleftrightarrow$.
Since $\mathcal{T}$ is any assignment, $F \longleftrightarrow H \equiv G \longleftrightarrow H$. **Q.E.D.**

Theorem 1.5.7 (The Substitution Theorem) *Let F, G and H be three formulas such that F contains occurrences of G and $G \equiv H$. Let I be the formula obtained from F by replacing an occurrence of G by H. Then $F \equiv I$.*

Before we go on to prove the theorem let us figure out what it says. Let $F = ((B \vee A) \vee C) \wedge ((B \vee A) \vee \neg C)$, $G = B \vee A$ and $H = A \vee B$.

We know, from equivalence 4.a of Table 1.47, that $G \equiv H$. We underline the two occurrences of G in F.

$(\underline{(B \vee A)} \vee C) \wedge (\underline{(B \vee A)} \vee \neg C)$. We replace the first occurrence by H and get

$I = ((A \vee B) \vee C) \wedge ((B \vee A) \vee \neg C)$.

The theorem tells us that F is equivalent to I, i.e.

$((B \vee A) \vee C) \wedge ((B \vee A) \vee \neg C \equiv ((A \vee B) \vee C) \wedge ((B \vee A) \vee \neg C)$.

If we replace the second occurrence of G by H, the theorem generates the equivalence

$((B \vee A) \vee C) \wedge ((B \vee A) \vee \neg C \equiv ((B \vee A) \vee C) \wedge ((A \vee B) \vee \neg C)$.

Proof: The proof is by structural induction on F. We assume that G occurs in F and that $G \equiv H$.

Case 1: F is an atom. Then $G = F$, since F has a single subformula. Replacing G by H in F yields $I = H$. Since $G \equiv H$, $F \equiv I$.

Case 2: $F = \neg J$. The induction hypothesis says that the theorem holds for J. The occurrence of G that is being replaced is a subformula of F. By Corollary 1.2.13 the occurrence is either F itself, or an occurrence of G in J.

Subcase 2.1: $G = F$. This is the situation described in Case 1, so $F \equiv I$.

Subcase 2.2: The occurrence is in J. Let K be the result of replacing G in J by H. Then $I = \neg K$. By the induction hypothesis, $J \equiv K$, and by The Congruence Lemma, $\neg J \equiv \neg K$. So, $F \equiv I$.

Case 3: $F = (JCK)$ where C is one of the connectives $\vee, \wedge, \longrightarrow, \longleftrightarrow$. By Corollary 1.2.13, the occurrence that is being replaced is either F itself, an occurrence of J, or an occurrence of K.

Subcase 3.1: $F = G$. This is the situation described in Case 1, so $F \equiv I$.

Subcase 3.2: The occurrence is in J. Let L be formula obtained by replacing the occurrence of G in J by H. By the induction hypothesis on J, $J \equiv L$. By The Congruence Lemma, $(JCK) \equiv (LCK)$. Since $I = (LCK)$, $F \equiv I$.

Subcase 3.3: G occurs in K. This subcase, similar to Subcase 3.2, is left as exercise. **Q.E.D.**

For many applications, we need to enlarge the notion of connective. So, let U be a meta-formula that contains the meta-variables $F_1, \ldots, F_n$. Then U defines a function $\underbrace{FORM \times \ldots \times FORM}_{ntimes} \longrightarrow FORM$ by mapping every n-tuple of formulas $< G_1, \ldots, G_n >$ into the formula $U[G_1, \ldots, G_n]$, obtained from U by replacing F_1 by G_1, F_2 by G_2, $\ldots$, F_n by G_n. We call U *a connective of arity* n. For example, the functions $G \downarrow H = \neg(G \vee H)$, and $G \uparrow H = \neg(G \wedge H)$ are the connectives of arity 2 that map every pair of formulas $< I, J >$ into the formula $\neg(I \vee J)$, respectively $\neg(I \wedge J)$. These two connectives are called *nor*, respectively *nand*. We notice that the meta-formulas $\neg G$, $G \vee H$, $G \wedge H$, $G \longrightarrow H$, $G \longleftrightarrow H$ define the connectives $\neg$, $\vee$, $\wedge$, $\longrightarrow$, and $\longleftrightarrow$.

A 0-ary connective has no meta-variables. For example, $\Box = P_0 \wedge \neg P_0$, and $\mathbf{T}= P_0 \vee \neg P_0$ are 0-ary connectives. The first formula, called *the box* is an unsatisfiable formula, and the second is a tautology.

Definition 1.5.8 (S-formulas) *Let S be a set of connectives. The S-formulas are inductively defined below.*

1. The atomic formulas are S-formulas.

2. The 0-ary connectives of S are S-formulas.

3. If $F \in S$ has arity $n > 0$ and $F_1, \ldots F_n$ are S-formulas, then $F[F_1, \ldots F_n]$ is an S-formula.

Example 1.5.9 Let us take $S = \{\neg, \vee\}$. The S-formulas are generated by the 3 rules, one for the atoms and one for each connective.

1. The atomic formulas are S-formulas.
2. If F is an S-formula, so is $\neg F$.
3. If F and G are S-formulas, so is $(F \vee G)$.

So, the S-formulas contain only atoms and the connectives $\neg$ and $\vee$.

Definition 1.5.10 (adequate set of connectives) *The set of connectives S is said to be adequate if every formula has an equivalent S-formula.*

The engineers implement formulas as networks that consists of *wires* and *gates*. The wires represent formulas and the gates are the connectives of arity greater than 0. A connective of arity n is a gate with n input wires and one output wire. For example, the formula $(F \wedge G) \vee \neg H$ is represented by the network from Figure 1.49. The shape of the gate identifies the connective. The top-left box is an *and* gate, the bottom left is a *not* and box at the right is an *or*. The value of

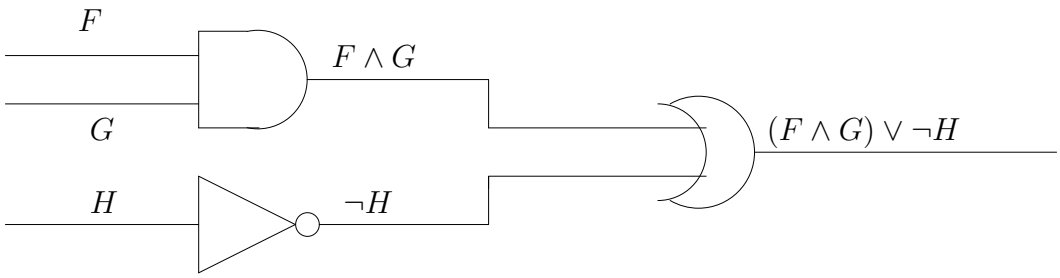


Figure 1.49: The network for $(F \wedge G) \vee \neg H$

the formula is determined by the presence or the absence of the current in the wire; 1 when it is conducting and 0 when not. The connective $\Box$ is implemented as a *cold* wire, that never has current. The tautology **T** is a *hot* wire, that always conducts.

The engineers are not interested in implementing a particular formula, but an equivalent one that optimizes such parameters as the number of gates, the maximum number of gates that the current must pass before the formula is evaluated (for our example the number is 2), the maximum arity of the connectives, and so on.

If a set of connectives is adequate, then every formula can be implemented using only the gate types that belong to the set. The adequacy of the set is also important in computing the S-normal forms.

Proposition 1.5.11 tells us that S is adequate when it can implement the connectives $\neg$, $\vee$, $\wedge$, $\longrightarrow$, and $\longleftrightarrow$.

Proposition 1.5.11 *If there are S-formulas, $\sigma_\neg[F]$, $\sigma_\vee[F,G]$,$\sigma_\wedge[F,G]$, $\sigma_\longrightarrow[F,G]$, $\sigma_\longleftrightarrow[F,G]$ such that*

1. $\neg F \equiv \sigma_\neg[F]$,
2. $(F \vee G) \equiv \sigma_\vee[F,G]$
3. $(F \wedge G) \equiv \sigma_\wedge[F,G]$
4. $(F \longrightarrow G) \equiv \sigma_\longrightarrow[F,G]$
5. $(F \longleftrightarrow G) \equiv \sigma_\longleftrightarrow[F,G]$

then S is adequate.

Proof: We show that G has an equivalent S-formula by structural induction on G.

Case 1: G is an atom. Then G is an S-formula.

Case 2. $G = \neg H$. By IH, H has an S-normal form, H_1. Then

$G = \neg H$

$\equiv \neg H_1$ by the IH and The Substitution Theorem

$\equiv \sigma_\neg[H_1]$ by 1

Since $\sigma_\neg[H_1]$ is an S-formula, we are done.

Cases 3,4,5,6: $G = (HCI)$, where C is one of the connectives $\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$. By IH, there are S-formulas H_1 and I_1 such that $H \equiv H_1$ and $I \equiv I_1$. We have the derivation

$G = (HCI)$

$\equiv (H_1CI_1)$ by The Substitution Theorem and the IH

$\equiv \sigma_C[H_1, I_1]$ by the connective rule that corresponds to C

The last formula is an S-formula. So, G has an S-equivalent formula. **Q.E.D.**

We will use Proposition 1.5.11 to show that several sets of connectives are adequate.

Proposition 1.5.12 *The sets* $S_1 = \{\neg, \vee, \wedge\}$, $S_2 = \{\neg, \vee\}$, $S_3 = \{\neg, \wedge\}$, $S_4 = \{\neg, \longrightarrow\}$, $S_5 = \{\uparrow\}$, $S_6 = \{\downarrow\}$, $S_7 = \{\Box, \longrightarrow\}$ *are adequate.*

Proof: We will show that the sets S_2, S_5, and S_7 are adequate and leave the rest as exercises. By Proposition 1.5.11 it suffices to show that the operations $\neg, \vee, \wedge, \longrightarrow, \longleftrightarrow$ can be implemented by the connectives that belong to the sets. The Substitution Theorem will be applied in almost all steps of the derivation, so we will not bother mentioning it.

1. Since $\neg$ and $\vee$ are already in S_2, we need to implement $\wedge$, $\longrightarrow$, and $\longleftrightarrow$.

1.1. $F \wedge G$

$\equiv \neg\neg(F \wedge G)$ by equivalence 8 of Table 1.47

$\equiv \neg(\neg F \vee \neg G)$ by equivalence 9b of Table 1.47

The connectives of the last formula are in S_2, so we implemented $\wedge$.

1.2. $F \longrightarrow G$

$\equiv \neg F \vee G$ by equivalence 2 of Table 1.47

Again, this formula has only S_2 connectives.

1.3 $F \longleftrightarrow G$

$\equiv (F \longrightarrow G) \wedge (G \longrightarrow F)$ by equivalence 1 of Table 1.47

At this point we can stop, since we have formulas that implement $\wedge$ and $\longrightarrow$. We will continue for the sake of displaying the S formula.

$\equiv (\neg F \vee G) \wedge (\neg G \vee F)$ by 1.2

$\equiv \neg(\neg(\neg F \vee G) \vee \neg(\neg G \vee F))$ by 1.1

2. We will show that all connectives can be implemented with $\uparrow$, where $F \uparrow G = \neg(F \wedge G)$.

2.1. $\neg F \equiv \neg(F \wedge F)$ by 3b of Table 1.47

$= F \uparrow F$ by the definition of $\uparrow$

2.2. $F \vee G \equiv \neg\neg(F \vee G)$ by equivalence 8 of Table 1.47

$\equiv \neg(\neg F \wedge \neg G)$ by equivalence 9a of Table 1.47

$= \neg F \uparrow \neg G$ by the definition of $\uparrow$

$= (F \uparrow F) \uparrow (G \uparrow G)$ by 2.1

2.3 $F \wedge G \equiv \neg\neg(F \wedge G)$ by equivalence 8 of Table 1.47

$= \neg(F \uparrow G)$ by the definition of $\uparrow$

$\equiv (F \uparrow G) \uparrow (F \uparrow G)$ by 2.1

2.4. $F \longrightarrow G \equiv \neg F \vee G$ by equivalence 2 of Table 1.47

Now we can apply 2.3 and 2.2, in this order, to obtain a $\uparrow$-formula.

$\equiv (\neg F \uparrow \neg F) \uparrow (G \uparrow G)$ by 2.2
$\equiv ((F \uparrow F) \uparrow (F \uparrow F)) \uparrow (G \uparrow G)$ by 2.1
2.5.$F \longleftrightarrow G$
$\equiv (F \longrightarrow G) \wedge (G \longrightarrow F)$ by equivalence 1 of Table 1.47

Since we know how to implement $\longrightarrow$ and $\wedge$ with $\uparrow$, $\longleftrightarrow$ has a $\uparrow$ formula equivalent to it.

3. We will show that $\neg$, $\vee$, $\wedge$ and $\longleftrightarrow$ can be implemented with $\Box$ and $\longrightarrow$, $\Box$ being an unsatisfiable formula (a contradiction).

3.1. $\neg F \equiv \Box \vee \neg F$ by equivalence 10b of Table 1.47
$\equiv \neg F \vee \Box$ by equivalence 4a of Table 1.47
$\equiv F \longrightarrow \Box$ by equivalence 2 of Table 1.47
3.2. $F \vee G \equiv \neg\neg F \vee G$ by equivalence 8 of Table 1.47
$\equiv (\neg F \longrightarrow G)$ by equivalence 2 of Table 1.47
$\equiv ((F \longrightarrow \Box) \longrightarrow G)$ by 3.1
3.3. $F \wedge G \equiv \neg\neg(F \wedge G)$ by equivalence 10b of Table 1.47
$\equiv \neg(\neg G \vee \neg H)$ by equivalence 9b of Table 1.47

Since we know how to implement $\neg$ and $\vee$ we have an S_7 formula for $\wedge$.

3.4. $F \longleftrightarrow G \equiv (F \longrightarrow G) \wedge (G \longrightarrow F)$ equivalence 9b of Table 1.47

Now we can apply the formula from 3.3 to get an S_7 formula for $F \longrightarrow G$.

In the preceding proofs we used both the equivalence sign $\equiv$ and the equality $=$. $F \equiv G$ means that the formulas represented by F and G are equivalent while $F = G$ signifies that the meta-formulas F and G denote the same formula. For example, $F \uparrow G = \neg(F \wedge G)$ means that $F \uparrow G$ represents the same formula as $\neg(F \wedge G)$. Since every equivalence contains the equality, it is not a mistake to replace $=$'s by $\equiv$'s, but we try to be as precise as possible. **Q.E.D.**

Note 1.5.13 We apply the equivalences from Table 1.47 by rewriting an instance of the left-hand-side of an equivalence by the corresponding instance of the right-hand-side, or vice versa. We give special names to these rewrites. When use the name $\longleftrightarrow$-*elimination* when we apply equivalence 1 from left to right and $\longleftrightarrow$-*introduction* when we apply it from right to left.

The applications of the second equivalence are called $\longrightarrow$-*eliminations* when we replace the left-hand-side and $\longrightarrow$-*introduction* when they replace the right-hand-side.

Equivalence 8 is called *double negation elimination* when it is applied left to right and *double negation introduction* when applied right to left.

The rest of the equivalences will be identified by their names, without specifying the direction. So, we will write

$A \vee B \equiv B \vee A$ by the commutativity of $\vee$

or simply,

$A \vee B \equiv B \vee A$ by commutativity.

These notations will free us from remembering the order in which the equivalences are listed in the table.

Exercises

Exercise 1.5.1 *Use the truth tables to verify the equivalences given in Table 1.47.*

Exercise 1.5.2 *Show that each line of Table 1.47 describes a countably infinite set of formula equivalences.*

Exercise 1.5.3 *Use truth tables to verify the equivalence* $F \longrightarrow G \equiv (\neg G \longrightarrow \neg F)$.

Exercise 1.5.4 *We say that two sets* S *and* T *are equivalent when they have the same set of models. Let* S *and* T *be two equivalent sets. Show that* $Con[S] = Con[T]$.

Exercise 1.5.5 *Check whether the connectives* $\longrightarrow$ *and* $\longleftrightarrow$ *are associative, commutative, idempotent. Remember that a binary connective* C *is associative when* $(FCG)CH \equiv FC(GCH)$, *commutative when* $FCH \equiv GCF$, *and idempotent when* $FCF \equiv F$.

Exercise 1.5.6 *Prove the remaining implications of Lemma 1.5.6.*

Exercise 1.5.7 *Check whether the connective* $\uparrow$ *defined by* $(F \uparrow G) = \neg(F \wedge G)$ *is associative, commutative, idempotent.*

Exercise 1.5.8 *Write the structural induction theorem that corresponds to the inductive definition of the* $\uparrow$*-formulas. Then, prove it.*

Exercise 1.5.9 *Write the object language formulas that correspond to the metalanguage expressions:* $(\neg P_1 \uparrow (P_3 \longrightarrow P_2)) \uparrow ((P_4 \uparrow P_6) \uparrow P_3)$, $(P_2 \uparrow (P_3 \uparrow P_4)) \uparrow ((P_2 \uparrow P_5) \uparrow P_1)$.

Exercise 1.5.10 *Prove that the sets* S_1,S_3,S_4, S_6 *of Proposition 1.5.12 are adequate.*

Exercise 1.5.11 *Let* $S = \{\longrightarrow\}$. *Write and prove The Structural Induction Theorem for* S.

Exercise 1.5.12 *Show that the* $S = \{\longrightarrow\}$ *is not adequate. Hint: Show that all* S*-formulas are satisfiable.*

Exercise 1.5.13 *We define the operation* $\diamond$ *by* $F \diamond G = \neg F \wedge G$ *and let* $S = \{\diamond\}$.

a. Prove that there is no tautology among the S*-formulas.*

b. Show that $S' = \{\diamond, \mathbf{T}\}$, *where* $\mathbf{T}$ *is a tautology, is adequate.*

1.6 Disjunctions and Conjunctions

This section takes a closer look at two special classes of formulas, the *disjunctions* and the *conjunctions* that will be widely used in the remainder of the book. First, we define them using meta-formulas and then we prove several theorems that relate the satisfiability and the equivalence of these formulas to their sets of components. Later on, we focus on conjunctions and disjunctions of literals and characterize their semantics in terms of their sets of literals.

Definition 1.6.1 (disjunctive form) *Let $F_1, F_2, \ldots, F_n$ be n distinct meta-variables. A disjunctive form of $F_1, F_2, \ldots, F_n$ is a meta-formula U that satisfies the following conditions:*

1. U contains exactly one occurrence of each of $F_1, F_2, \ldots, F_n$.

2. If $1 \leq i < j \leq n$, F_i precedes F_j in U.

3. U contains only the meta-variables $F_1, \ldots, F_n$, the connective $\vee$, left and right parentheses.

If $n = 0$ then $U = \Box$, the unsatisfiable formula.

Example 1.6.2 1. Let us find all disjunctive forms of F_1, F_2, F_3.

We have only two, $((F_1 \vee F_2) \vee F_3)$ and $(F_1 \vee (F_2 \vee F_3))$.

2. Let us find all disjunctive forms of F_1, F_2, F_3, F_4. We have 5 meta-formulas,

$(((F_1 \vee F_2) \vee F_3) \vee F_4)$,
$((F_1 \vee (F_2 \vee F_3)) \vee F_4)$,
$((F_1 \vee F_2) \vee (F_3 \vee F_4))$,
$(F_1 \vee ((F_2 \vee F_3) \vee F_4))$,
$(F_1 \vee (F_2 \vee (F_3 \vee F_4)))$.

Definition 1.6.3 (disjunction) *Let U be a disjunctive form of $F_1, \ldots, F_n$ and $G_1, \ldots, G_n$ be n formulas, not necessarily distinct. A disjunction of $G_1, \ldots, G_n$ is the instance $U[G_1, \ldots, G_n]$ obtained from U by replacing F_1 by G_1, F_2 by G_2,...,F_n by G_n. The sequence $G_1, \ldots, G_n$ is called the support of the disjunction and the formulas G_i, the disjuncts.*

Example 1.6.4 Let us find all disjunctions with support $P_1, \neg P_2, (P_1 \vee P_2)$. The sequence has 3 formulas, so we generate all disjunctive forms of F_1, F_2, F_3. They are $U = ((F_1 \vee F_2) \vee F_3)$ and $V = (F_1 \vee (F_2 \vee F_3))$. We compute the instances $U[P_1, \neg P_2, (P_1 \vee P_2)]$ and $V[P_1, \neg P_2, (P_1 \vee P_2)]$ and get the disjunctions $((P_1 \vee \neg P_2) \vee (P_1 \vee P_2))$ and $(P_1 \vee (\neg P_2 \vee (P_1 \vee P_2)))$.

Observation 1.6.5 1. The same disjunction can be generated by several disjunctive forms. The formula $((P_1 \vee (P_2 \wedge \neg P_3)) \vee \neg P_4)$ is generated by

the disjunctive form F_1 with support $(P_1 \vee (P_2 \wedge \neg P_3)) \vee \neg P_4$,
the disjunctive form $(F_1 \vee F_2)$ with support $(P_1 \vee (P_2 \wedge \neg P_3))$, $\neg P_4$, and
the disjunctive form $((F_1 \vee F_2) \vee F_3)$ with support P_1, $(P_2 \wedge \neg P_3)$, $\neg P_4$.

We will prove, in Theorem 1.6.8, that two disjunctions with the same support are equivalent.

Definition 1.6.6 (the prefix form for disjunctions) *We call the formula* $\bigvee_{i=n}^{m} F_i$ *the prefix form of* $F_n, F_{n+1}, \ldots, F_m$.

Lemma 1.6.7 relates the prefix form of $F_1, \ldots, F_n, \ldots, F_{n+m}$ to the prefix forms of the sequences $F_1, \ldots, F_n$ and $F_{m+1}, \ldots, F_{n+m}$.

Lemma 1.6.7 $(\bigvee_{i=1}^{n} F_i \vee \bigvee_{i=n+1}^{n+m} F_i) \equiv \bigvee_{i=1}^{n+m} F_i$.

Proof: By induction on m.
Basis: $m = 0$. Then,
$(\bigvee_{i=1}^{n} F_i \vee \bigvee_{i=n+1}^{n+m} F_i)$
$= (\bigvee_{i=1}^{n} F_i \vee \Box)$ since $n+1 > n+m$, $\bigvee_{i=n+1}^{n+m} F_i = \Box$
$\equiv (\Box \vee \bigvee_{i=1}^{n} F_i)$ by the commutativity of $\vee$, 4a of Table 1.47
$\equiv \bigvee_{i=1}^{n} F_i$ by the contradiction law 10b of Table 1.47.
Inductive Step: Assume that the equivalence holds for m. We will prove it for $m+1$. Due to the definition of $\bigvee_{i=n+1}^{n+m+1} F_i$, we need to consider two cases, $m = 0$ and $m > 0$.
Subcase 2.1: $m = 0$.
$(\bigvee_{i=1}^{n} F_i \vee \bigvee_{i=n+1}^{n+m+1} F_i)$
$= (\bigvee_{i=1}^{n} F_i \vee F_{n+1})$ since $m = 0$, $\bigvee_{i=n+1}^{n+m+1} F_i = F_{n+1}$
$= \bigvee_{i=1}^{n+1} F_i$ from the definition of $\bigvee_{i=1}^{n+m+1} F_i$
$= \bigvee_{i=1}^{n+m+1} F_i$ since $m = 0$.
Subcase 2.2: $m > 0$.
$(\bigvee_{i=1}^{n} F_i \vee \bigvee_{i=n+1}^{n+m+1} F_i)$
$= (\bigvee_{i=1}^{n} F_i \vee (\bigvee_{i=n+1}^{n+m} F_i \vee F_{n+m+1}))$ from the definition of $\bigvee_{i=n+1}^{n+m+1} F_i$
$\equiv ((\bigvee_{i=1}^{n} F_i \vee \bigvee_{i=n+1}^{n+m} F_i) \vee F_{n+m+1})$ from the associativity of $\vee$
$\equiv (\bigvee_{i=1}^{n+m} F_i \vee F_{n+m+1})$ by the induction hypothesis
$= \bigvee_{i=1}^{n+m+1} F_i$ from the definition of $\bigvee_{i=1}^{n+m+1} F_i$. **Q.E.D.**

Theorem 1.6.8 (The Prefix Form Theorem for Disjunctions) *All disjunctions with support* $G_1, G_2, G_3, \ldots, G_n$ *are equivalent to* $\bigvee_{i=1}^{n} G_i$.

Proof: By induction on n, the size of the support sequence.
Basis: $n = 0$. Then the disjunctive form is the box, and so is the disjunction. The prefix form of the empty sequence is also $\bigvee_{i=1}^{0} F_i = \Box$.
Inductive Step: We need to treat the subcases $n = 1$ and $n > 1$.

Subcase 2.1: $n = 1$. Let G be a formula. By Exercise 1.6.1, the disjunctive form of the meta-variable F is F. So, the disjunction with support G is G itself. The prefix form of the sequence G is also G. So, again, we have equality.

Subcase 2.2: Let us assume that the theorem is true for all sequences of length k, where $n \geq 1$ and $0 \leq k \leq n$. Let H be a disjunction with support $G_1, \ldots, G_n, G_{n+1}$. By Exercise 1.6.3, $H = (I \vee J)$, and there is some l, $1 \geq l < n+1$, such that the support of I is $G_1, \ldots, G_l$ and the support of J is $G_{l+1}, \ldots, G_{n+1}$. By the induction hypotheses for I and for J, we get equivalences (1) and (2).
(1) $I \equiv \bigvee_{i=1}^{l} G_i$

(2) $J \equiv \bigvee_{i=l+1}^{n+1} G_i$.

Now we can show that F is equivalent to the prefix form of its sequences.

$H = (I \vee J) \equiv (\bigvee_{i=1}^{l} F_i \vee \bigvee_{i=l+1}^{n+1} F_i)$ from (1) and (2) by substitution

$\equiv \bigvee_{i=1}^{n+1} F_i$ by Lemma 1.6.7 **Q.E.D.**

Theorem 1.6.8 tells us that two formulas with the same support are equivalent, because both are equivalent to $\bigvee_{i=1}^{n} F_i$. However, many formulas with different supports are equivalent. For example, the formula $F = P_1 \vee P_1$ with support P_1, P_1 is equivalent to $G = P_1$ with support P_1. We want a theorem that requires a weaker condition[6], than Theorem 1.6.8. For this purpose we define the *element-wise equivalence* of two sets of formulas.

Definition 1.6.9 (element-wise equivalence) *We say that two sets of formulas S and T are element-wise equivalent when*

1. *for every formula $F \in S$ there is a formula $G \in T$ such that $G \equiv F$, and*
2. *For every formula $G \in T$ there is a formula $F \in S$ such that $F \equiv G$.*

Examples 1.6.10 1. The sets of formulas $\{P_3, P_1, P_2, P_3, P_2\}$ and $\{P_1, P_2, P_3\}$ are element-wise equivalent because they satisfy Definition 1.6.9.

- All members of $\{P_3, P_1, P_2, P_3, P_2\}$ are in the second set, $\{P_1, P_2, P_3\}$.
- All members of $\{P_1, P_2, P_3\}$ are also in $\{P_3, P_1, P_2, P_3, P_2\}$.

2. The sets $\{P_1, (P_2 \vee P_2)\}$ and $\{(P_1 \wedge (P_1 \vee P_2)), P_2\}$ are element-wise equivalent because $P_1 \equiv (P_1 \wedge (P_1 \vee P_2))$ and $(P_2 \vee P_2) \equiv P_2$.

Theorem 1.6.11 *Let F be a disjunction with support $F_1, F_2, \ldots, F_n$ and G a disjunction with support $G_1, G_2, ..., G_m$. If the sets $\{F_1, \ldots, F_n\}$ and $\{G_1, \ldots G_m\}$ are element-wise equivalent, the formulas F and G are equivalent.*

The proof of the theorem relies on Lemma 1.6.12.

Lemma 1.6.12 $\models_{\mathcal{A}} \bigvee_{i=1}^{n} F_i$ *iff there exists some j, $1 \leq j \leq n$, such that* $\models_{\mathcal{A}} F_j$.

Proof: The proof is by induction on n.
Basis: $n = 0$. Then $\bigvee_{i=1}^{0} F_i = \Box$ has no model and there is no j between 1 and 0. So both statements, $\models_{\mathcal{A}} \Box$ and *for some j, $1 \leq j \leq n$,* $\models_{\mathcal{A}} F_j$, are false and the iff holds.
Inductive Step: Again we have the subcases $n = 1$ and $n > 1$.

Subcase 2.1: $n = 1$. Then $\bigvee_{i=1}^{n} F_i = F_1$ and the lemma becomes

$\models_{\mathcal{A}} F_1$ iff there exists some j, $1 \leq j \leq 1$, such that $\models_{\mathcal{A}} F_j$.

Since the only j greater than or equal to 1 and less than or equal to 1 is 1 itself, the above statement reduces to the assertion

$\models_{\mathcal{A}} F_1$ iff $\models_{\mathcal{A}} F_1$,

which is always true.

Subcase 2.2: Assume that the lemma is true for $n \geq 1$. Then $\bigvee_{i=1}^{n+1} F_i = (\bigvee_{i=1}^{n} F_i \vee F_{n+1})$.

[6] F is weaker than G if $G \models F$ but $F \not\models G$.

Assume that $\models_{\mathcal{A}} \bigvee_{i=1}^{n+1} F_i$. Since $\mathcal{A}[\bigvee_{i=1}^{n+1} F_i] = \mathcal{A}[\bigvee_{i=1}^{n} F_i] \boxed{\vee} \mathcal{A}[F_{n+1}]$, either $\models_{\mathcal{A}} \bigvee_{i=1}^{n} F_i$, or $\models_{\mathcal{A}} F_{n+1}$, or both. If $\models_{\mathcal{A}} \bigvee_{i=1}^{n} F_i$, we apply the IH and get that there is some j, $1 \leq j \leq n$, such that $\mathcal{A} \models F_j$. So, in all 3 cases, $\models_{\mathcal{A}} F_j$ for some $1 \leq j \leq n+1$.

Now let us go the other way and assume that $\models_{\mathcal{A}} F_j$ for some $1 \leq j \leq n+1$. If $j = n+1$,

$\mathcal{A}[\bigvee_{i=1}^{n} F_i] \boxed{\vee} \mathcal{A}[F_{n+1}] = \mathcal{A}[\bigvee_{i=1}^{n} F_i] \boxed{\vee} 1 = 1$.

If $j \leq n$ then, by IH, $\mathcal{A}[\bigvee_{i=1}^{n} F_i] = 1$. So,

$\mathcal{A}[\bigvee_{i=1}^{n} F_i] \boxed{\vee} \mathcal{A}[F_{n+1}] = 1 \boxed{\vee} \mathcal{A}[F_{n+1}] = 1$.

In either case $\models_{\mathcal{A}} \bigvee_{i=1}^{n+1} F_i$.**Q.E.D.**

Corollary 1.6.13 *Let F be a disjunction with support $F_1, \ldots, F_n$. Then $\models_{\mathcal{A}} F$ iff $\models_{\mathcal{A}} F_i$ for some $1 \leq i \leq n$.*

Proof: By Theorem 1.6.8, $F \equiv \bigvee_{i=1}^{n} F_i$, so $\models_{\mathcal{A}} F$ iff $\models_{\mathcal{A}} \bigvee_{i=1}^{n} F_i$.**Q.E.D.**

Corollary 1.6.14 *Let F be a disjunction with support $F_1, \ldots, F_n$. Then $\not\models_{\mathcal{A}} F$ iff for all i, $1 \leq i \leq n$, $\not\models_{\mathcal{A}} F_i$.*

Now we can prove Theorem 1.6.11.

Proof: Let us assume that the support sets $\{F_1, \ldots, F_n\}$ and $\{G_1, \ldots, G_m\}$ of F, respectively G, are element-wise equivalent. Let $\mathcal{A}$ be a truth assignment. We'll show that

$\mathcal{A}[F] = 1$ iff $\mathcal{A}[G] = 1$.

$\models_{\mathcal{A}} F$ iff

$\models_{\mathcal{A}} \bigvee_{i=1}^{n} F_i$ by Theorem 1.6.8

iff for some $1 \leq j \leq n$, $\models_{\mathcal{A}} F_j$ by Lemma 1.6.12

iff for some $1 \leq i \leq m$, $\models_{\mathcal{A}} G_i$ by Definition 1.6.9

iff $\models_{\mathcal{A}} \bigvee_{i=1}^{m} G_i$ by Lemma 1.6.12

iff $\models_{\mathcal{A}} G$ by Theorem 1.6.8 **Q.E.D.**

Observation 1.6.15 1. The converse of Theorem 1.6.11 is false. We can check, by truth tables, that $P_1 \vee P_2 \equiv ((P_1 \wedge P_2) \vee (\neg P_1 \wedge P_2)) \vee (P_1 \wedge \neg P_2)$, but the support sets of the two disjunctions, $\{P_1, P_2\}$ and $\{P_1 \wedge P_2, \neg P_1 \wedge P_2, P_1 \wedge \neg P_2\}$ are not elementwise equivalent because neither $P_1 \wedge P_2$, nor $\neg P_1 \wedge P_2$, nor $P_1 \wedge \neg P_2$ is equivalent to P_1.

2. Theorem 1.6.11 tells us that the truth value of a disjunction depends on the set of disjuncts, and not on the support sequence. If the disjunctions F and G have the same support set, i.e. $\{G_1, \ldots, G_m\} = \{F_1, \ldots, F_n\}$ then $F \equiv G$. So, from now on, we will talk about sets of disjuncts instead of support sequences.

We are particularly interested in the sets of disjuncts that contain only atoms and negations of atoms.

Definition 1.6.16 (literal, complement, clause) *1. A literal is an atom or the negation of an atom.*

2. The complement of the literal L , written $\overline{L}$, is defined as

$$\overline{L} = \begin{cases} \neg P_i & \text{if } L = P_i \\ P_i & \text{if } L = \neg P_i \end{cases}$$

3. A clause is a disjunction of literals.

We notice that the complement of a literal is also a literal. We will prove that two clauses that are not tautologies are equivalent iff they have the same set of disjuncts. But first we give a syntactic characterization of the tautology clauses.

Lemma 1.6.17 *A clause is a tautology iff, for some atom Q, both Q and $\neg Q$ are among its disjuncts.*

Proof: Let C be a clause.
$\Longleftarrow$: Assume that both Q and $\neg Q$ are among the disjuncts of C and let $\mathcal{A}$ be a truth assignment. Then either $\models_{\mathcal{A}} Q$ or $\models_{\mathcal{A}} \neg Q$, so $\mathcal{A}$ models a disjunct of C. By Corollary 1.6.13, $\models_{\mathcal{A}} C$.
$\Longrightarrow$: Assume that the disjuncts do not include a pair Q, $\overline{Q}$. We will show that C is not a tautology. Let $\mathcal{A}$ be the truth assignment defined below.

$$\mathcal{A}[P_i] = \begin{cases} 0 & \text{if } P_i \text{ is a disjunct of } C \\ 1 & \text{if } P_i \text{ is not a disjunct of } C \end{cases}$$

Now we will show that $\mathcal{A}$ is a countermodel for all literals L of C. The literal L is either an atom or the negation of an atom. If L is an atom, then $\mathcal{A}[L] = 0$ from the definition of $\mathcal{A}$.

Now let L be the negation of an atom, i.e. $L = \neg P_j$ for some index j. Since $\overline{L} = P_j$ is not in C, $\mathcal{A}[P_j] = 1$. This means that $\mathcal{A}[L] = \mathcal{A}[\neg P_j] = 0$.

So, $\mathcal{A}$ is a countermodel to all disjuncts of C. By the second corollary to Lemma 1.6.12, $\not\models_{\mathcal{A}} C$, i.e. C is not a tautology. **Q.E.D.**

Theorem 1.6.18 *Let F and G be two clauses that are not tautologies. Then $F \equiv G$ iff their sets of disjuncts are equal.*

Proof: $\Longleftarrow$ If the sets of disjuncts of F and G are equal, they are element-wise equivalent. So, by Theorem 1.6.11, $F \equiv G$.
$\Longrightarrow$: We will show that when the sets of disjuncts are not equal, $F \not\equiv G$.

Let $\{F_1, ..., F_n\}$ and $\{G_1, ..., G_m\}$ be the sets of disjuncts of F, respectively G. Let us assume that they are not equal. Then, there is an element that belongs to one set, but not to the other. Let's assume that F_1 is not a disjunct of G. Since F_1 is a literal, there is some j such that $F_1 = P_j$ or $F_1 = \neg P_j$. Because G is not a tautology, it has a countermodel, say $\mathcal{A}$. We define the truth assignment $\mathcal{B}$ that is exactly like $\mathcal{A}$, except that $\mathcal{B}[F_1] = 1$.

$$\mathcal{B}[P_i] = \begin{cases} 1 & \text{if } i = j \text{ and } F_1 = P_i \\ 0 & \text{if } i = j \text{ and } F_1 = \neg P_i \\ \mathcal{A}[P_i] & \text{if } i \neq j \end{cases}$$

Now we will show that $\mathcal{A}$ and $\mathcal{B}$ agree on the atoms of G. The only place where they may disagree is P_j.

If neither P_j nor $\neg P_j$ is a literal of G, then $\mathcal{A}$ and $\mathcal{B}$ agree on G.

If one of P_j, $\neg P_j$ is in G, then it must be $\overline{F_1}$, because F_1 is not in G. Then, from the construction of $\mathcal{B}$, $\mathcal{B}[\overline{F_1}] = 0$. Since $\mathcal{A}$ is a counter-model of G and $\overline{F_1}$ is a disjunct of G, $\mathcal{A}[\overline{F_1}]$ must also be 0. So, both $\mathcal{A}$ and $\mathcal{B}$ assign 0 to $\overline{F_1}$. Since they also agree on all the other atoms of G, they agree on G.

In both cases, $\mathcal{A}$ and $\mathcal{B}$ agree on G. Since $\mathcal{A}$ is a countermodel of G, so is $\mathcal{B}$.

However, $\models_{\mathcal{B}} F_1$, so $\models_{\mathcal{A}} F$. Then $\mathcal{B}[G] = \mathcal{A}[G] = 0$ and $\mathcal{B}[F] = 1$ which tells us that $F \not\equiv G$. **Q.E.D.**

Example 1.6.19 (the number of non-equivalent clauses with n atoms) Let us find out how many different, i.e. non-equivalent clauses, can be formed with n atoms, P_1, P_2, ...,P_n. By Theorem 1.6.18, a clause that is not a tautology is identified by its set of literals. Since duplications do not count in sets, let us take only the sets that have no repetitions of literals. There are only $2n$ possible disjuncts, P_1, $\neg P_1$, P_2, $\neg P_2$,..., P_n, $\neg P_n$, so, the sets can contain at most $2n$ elements. Moreover, every set that has more than n elements must have a pair Q, $\neg Q$ for some Q in $\{P_1, P_2, \ldots, P_n\}$. By Lemma 1.6.17, all these sets are tautologies. So, all non-tautologies have at most n literals. For each $Q \in \{P_1, P_2, \ldots, P_n\}$ the non-tautology clause offers the following distinct choices:

1. Q is in the set,
2. $\neg Q$ is in the set,
3. neither Q nor $\neg Q$ are in the set.

The fourth choice, both Q and $\neg Q$ are in the set, is not possible since the clause is not a tautology. Since for each Q we have 3 choices, we have n different Q's, and the choices are independent, we get 3^n different sets. By Theorem 1.6.18, the clauses that correspond to these sets are not equivalent. So, we have 3^n such clauses. We add the tautology for a total of $3^n + 1$ clauses.

Now we will study the properties of *conjunctions*. The presentation parallels the one we made for disjunctions.

Definition 1.6.20 (conjunctive form) *Let $F_1, F_2, \ldots, F_n$ be n distinct meta-variables. A conjunctive form of $F_1, F_2, \ldots, F_n$ is a meta-formula U that satisfies the following conditions:*

1. U contains exactly one occurrence of each of $F_1, F_2, \ldots, F_n$.

2. If $1 \leq i < j \leq n$, F_i precedes F_j.

3. U contains only the meta-variables $F_1, \ldots, F_n$, the connective $\wedge$, left and right parentheses.

If $n = 0$ then $U = \mathbf{T}$, a tautology.

Examples 1.6.21 1. Let us find the conjunctive forms of F_1, F_2, F_3, F_4.

We have 5 meta-formulas,

1. $(((F_1 \wedge F_2) \wedge F_3) \wedge F_4)$,
2. $((F_1 \wedge (F_2 \wedge F_3)) \wedge F_4)$,

3. $((F_1 \wedge F_2) \wedge (F_3 \wedge F_4))$,
4. $(F_1 \wedge ((F_2 \wedge F_3) \wedge F_4))$,
5. $(F_1 \wedge ((F_2 \wedge (F_3 \wedge F_4)))$.

The conjunctive forms are almost identical to the 5 disjunctive forms presented in Example 1.6.2, except that the $\vee$'s are replaced by $\wedge$'s.

Definition 1.6.22 (conjunction) *Let U be a conjunctive form of $F_1, \ldots, F_n$ and $G_1, \ldots, G_n$ be n formulas, not necessarily distinct. A conjunction of $G_1, \ldots, G_n$ is the instance $U[G_1, \ldots, G_n]$ obtained from U by replacing F_1 by G_1, F_2 by G_2,...,F_n by G_n. The sequence $G_1, \ldots, G_n$ is called the support of the conjunction and the formulas G_i, the conjuncts.*

Observation 1.6.23 We make the same observation for conjunctions that we did for disjunctions, that a conjunction can be generated by several conjunctive forms. The formula $((P_1 \wedge \neg P_3) \wedge (\neg P_4 \wedge P_5))$ can be generated by the following conjunctive forms:

1. $U = F_1$, with $F_1 = ((P_1 \wedge \neg P_3) \wedge (\neg P_4 \wedge P_5))$,
2. $U = (V \wedge W)$ with $V = (P_1 \wedge \neg P_3)$ and $W = (\neg P_4 \wedge P_5)$,
3. $U = ((X \wedge Y) \wedge W)$ with $X = P_1$, $Y = \neg P_3$, $W = (\neg P_4 \wedge P_5)$,
4. $U = (V \wedge (X \wedge Y)$ with $V = (P_1 \wedge \neg P_3)$, $X = \neg P_4$, $Y = P_5$, and
5. $U = ((X \wedge Y) \wedge (Z \wedge V)$ with $X = P_1$, $Y = \neg P_3$, $Z = \neg P_4$, $V = P_5$.

We will show that, just like the disjunctions, all formulas with the same support are equivalent.

Definition 1.6.24 (prefix forms for conjunctions) *The formula $\bigwedge_{i=n}^{m} G_i$ is the prefix form for a conjunction with support $G_1, \ldots, G_m$.*

Lemma 1.6.25 $(\bigwedge_{i=1}^{n} G_i \wedge \bigwedge_{i=n+1}^{n+m} G_i) \equiv \bigwedge_{i=1}^{n+m} G_i$.

The proof is by induction on m. It is similar to the proof of Lemma 1.6.7 and is left as exercise.

Now we can state the analogue of Theorem 1.6.8, which states that two conjunctions with the same support are equivalent.

Theorem 1.6.26 *[The Prefix Normal Form Theorem for Conjunctions] All conjunctions with support $G_1, G_2, \ldots, G_n$ are equivalent to $\bigwedge_{i=1}^{n} G_i$.*

Proof: By induction on n. Since the definition of $\bigwedge_{i=1}^{n} G_i$ has 3 cases, our proof will also have 3 cases, $n = 0$, $n = 1$ and the inductive step.
Basis: $n = 0$. The conjunctive form for the empty sequence of meta-variables is **T**, so the conjunction of the empty sequence of formulas is also **T**. At the same time $\bigwedge_{i=1}^{0} G_i = \mathbf{T}$. So both formulas are equal to **T**.
Basis: $n = 1$. The conjunctive form of F_1 is F_1, and the conjunction with support G_1 is obtained by replacing F by G_1. So, the conjunction is G_1. At the same time, $\bigwedge_{i=1}^{1} G_i = G_1$. Again, the two formulas are equal.
Inductive Step: We assume that the theorem is true for all supports of length less than or equal to $n \geq 1$. Let G be a conjunction with support $G_1, \ldots, G_n, G_{n+1}$.

The property listed in Exercise 1.6.3 is also valid for conjunctions, so $G = (I \wedge J)$, and the supports of I and J are the sequences $G_1, \ldots, G_l$, respectively $G_{l+1}, \ldots, G_{n+1}$, for some l between 1 and n. We can apply the induction hypothesis to I and to J, since both of them have n conjuncts or less. We get the equivalences (1) and (2).

(1) $I \equiv \bigwedge_{i=1}^{l} G_i$

(2) $J \equiv \bigwedge_{i=l+1}^{n+1} G_i$

Then

$G = (I \wedge J)$

$\equiv (\bigwedge_{i=1}^{l} G_i \wedge \bigwedge_{i=l+1}^{n+1} G_i)$ from (1) and (2)

$\equiv \bigwedge_{i=1}^{n+1} G_i$ by Lemma 1.6.25 **Q.E.D.**

We will prove a stronger result[7] than Theorem 1.6.26, that two conjunctions with element-wise equivalent sets of conjuncts are equivalent. First, we will prove the analogue of Lemma 1.6.12.

Lemma 1.6.27 $\models_{\mathcal{A}} \bigwedge_{i=1}^{n} G_i$ *iff for all* i, $1 \leq i \leq n$, $\models_{\mathcal{A}} G_i$.

Proof: By induction on n, the length of the support. Since the definition of $\bigwedge_{i=1}^{n} G_i$ has 3 cases, we also have 3 cases, $n = 0$, $n = 1$, and the inductive case.
Basis: $n = 0$. Then the conjunction is the tautology **T**, and it has no conjuncts. So, the statement

for all i, $1 \leq i \leq n$, $\mathcal{A}[F_i] = 1$

is vacuously true. Since $\mathcal{A}$ is a model of **T**, the iff holds.

Basis: $n = 1$. Then the conjunction is $G = G_1$ and the theorem statement becomes

$\models_{\mathcal{A}} G_1$ iff for all i, $1 \leq i \leq 1$, $\models_{\mathcal{A}} G_i$.

Since the inequality $1 \leq i \leq 1$ has only one solution, $i = 1$, the above statement reduces to

$\models_{\mathcal{A}} G_1$ iff $\models_{\mathcal{A}} G_1$

which is true.

Inductive Step: Assume that the lemma is true for $n \geq 1$. Then,

(1) $\bigwedge_{i=1}^{n+1} G_i = \bigwedge_{i=1}^{n} G_i \wedge G_{n+1}$

from the definition of $\bigwedge_{i=1}^{n+1} G_i$. Now we have the derivation below.

$\models_{\mathcal{A}} \bigwedge_{i=1}^{n+1} G_i$

iff $\models_{\mathcal{A}} (\bigwedge_{i=1}^{n} G_i \wedge G_{n+1})$ by (1)

iff $\models_{\mathcal{A}} \bigwedge_{i=1}^{n} G_i$ and $\models_{\mathcal{A}} G_{n+1}$ by the interpretation of $\wedge$

iff for all $1 \leq i \leq n$, $\models_{\mathcal{A}} G_i$ and $\models_{\mathcal{A}} G_{n+1}$ by the IH

iff for all $1 \leq i \leq n+1$, $\models_{\mathcal{A}} G_i$ **Q.E.D.**

Corollary 1.6.28 *Let G be a conjunction with support $G_1, \ldots, G_n$. Then $\models_{\mathcal{A}} G$ iff for all i, $1 \leq i \leq n$, $\models_{\mathcal{A}} G_i$.*

Proof: By Theorem 1.6.26, $G \equiv \bigwedge_{i=1}^{n} G_i$. **Q.E.D.**

Corollary 1.6.29 *Let G be a conjunction with support $G_1, \ldots, G_n$. Then $\not\models_{\mathcal{A}} G$ iff there exists some i, $1 \leq i \leq n$, such that $\not\models_{\mathcal{A}} G_i$.*

[7] The condition of the new result is weaker than that of the theorem.

Now we can prove that two conjunctions with element-wise sets of conjuncts are equivalent.

Theorem 1.6.30 *Let F and G be conjunctions with the sets of conjuncts $\{F_1, \ldots, F_n\}$, respectively $\{G_1, \ldots, G_m\}$. If the sets of conjuncts are element-wise equivalent, then $F \equiv G$.*

Proof: We will show that $\mathcal{A}$ is countermodel of F iff it is a countermodel of G.

$\not\models_{\mathcal{A}} F$

iff there is some i, $1 \leq i \leq n$, such that $\not\models_{\mathcal{A}} F_i$ by Corollary 1.6.29

iff there is some j, $1 \leq j \leq n$, such that $\not\models_{\mathcal{A}} G_j$ the disjunct sets are element-wise equivalent

iff $\not\models_{\mathcal{A}} G$ by Corollary 1.6.29 **Q.E.D.**

Observation 1.6.31 1. Theorem 1.6.30 allows us to talk about *sets of conjuncts*, because whenever the conjunct sets are equal, they are element-wise equivalent and the conjunctions are equivalent.

2. At this point we might ask: if F and G are conjunctions and $F \equiv G$, are the conjunct sets of F and G element-wise equivalent?

The answer is no. The conjunctions $F = P_1 \wedge \neg P_1$ and $G = P_2 \wedge \neg P_2$ are equivalent because both are unsatisfiable. However, the conjunct sets $\{P_1, \neg P_1\}$, and $\{P_2, \neg P_2\}$ are not element-wise equivalent.

Now let us concern ourselves with conjunctions of **literals**. For them we can prove a statement similar to Lemma 1.6.17.

Lemma 1.6.32 *Let F be conjunction of literals. Then F is unsatisfiable iff F contains both a literal and its complement.*

Proof: $\Longleftarrow$. Assume that F contains both P_i and $\neg P_i$. Let $\mathcal{A}$ be a truth assignment. Then $\mathcal{A}$ assigns 0 to one of them. By Corollary 1.6.29, $\mathcal{A}[F] = 0$.

$\Longrightarrow$: We will show that whenever the conjunct set of F does not contain pairs $P_i, \neg P_i$, F is satisfiable. Let F be such a set. We define $\mathcal{A}$ as

$$\mathcal{A}[P_i] = \begin{cases} 1 & \text{if } P_i \text{ is a conjunct of } F \\ 0 & \text{if } P_i \text{ is not a conjunct of } F \end{cases}$$

We will show that $\mathcal{A}$ models every conjunct $L \in F$. If $L = P_i$ is in F, then $\models_{\mathcal{A}} L$ from the construction of $\mathcal{A}$. If $L = \neg P_i$, P_i is not a conjunct because F does not contain the pair P_i, $\neg P_i$. So, $\mathcal{A}[P_i] = 0$, and $\mathcal{A}[L] = 1$. In either case $\models_{\mathcal{A}} L$. So, by Corollary 1.6.28, $\models_{\mathcal{A}} F$. **Q.E.D.**

The next theorem tells us that two satisfiable conjunctions of literals are equivalent if and only if they have the same set of conjuncts.

Theorem 1.6.33 *Let F and G be two conjunctions of literals that do not contain a pair Q, $\neg Q$. Then $F \equiv G$ if and only if their sets of conjuncts are equal.*

Proof: $\Longleftarrow$If the sets of conjuncts are equal then they are element-wise equivalent, so $F \equiv G$ by Theorem 1.6.30.

$\Longrightarrow$: Now we will prove that whenever the conjunct sets are not equal, $F \not\equiv G$. If the conjunct sets of F and G are different, one set must contain a literal that is not in the other. Let us assume, without loss of generality, that the literal F_1 from the first set is not a conjunct of G. Since F_1 is a literal, there is some index j such that either $F_1 = P_j$ or $F_1 = \neg P_j$.

The formula G does not contain pairs $Q, \neg Q$, so it is satisfiable by Lemma 1.6.32. Let $\mathcal{A}$ be a model of G. We will construct a truth assignment $\mathcal{B}$ that models G but is a countermodel of F.

$$\mathcal{B}[P_i] = \begin{cases} 0 & \text{if } i = j \text{ and } F_1 = P_j \\ 1 & \text{if } i = j \text{ and } F_1 = \neg P_j \\ \mathcal{A}[P_i] & \text{if } i \neq j \end{cases}$$

The above construction tells us that $\mathcal{B}[F_1] = 0$, so $\not\models_{\mathcal{B}} F$. Now let L be a conjunct of G. If the atom of L is different from P_j, then $\mathcal{A}$ and $\mathcal{B}$ agree on L, so $\models_{\mathcal{B}} L$. If the atom of L is P_j, then $L = \overline{F_1}$, because F_1 is not in G. Since $\mathcal{B}[F_1] = 0$, $\models_{\mathcal{A}} L$.

Then, $\mathcal{B}$ is a model for all conjuncts of G, so, it is a model of G. Since it is not a model of F, $F \not\equiv G$.**Q.E.D.**

Exercises

Exercise 1.6.1 *Prove that the only disjunctive form of F is F itself.*

Hint: Use exercises 1.2.3, 1.2.4.

Exercise 1.6.2 *Let U be a disjunctive form of $F_1, \ldots, F_k$ and V be a disjunctive form of $F_{k+1}, \ldots, F_n$, where $1 \leq k < n$. Then $(U \vee V)$ is a disjunctive form of $F_1 \ldots, F_n$.*

Exercise 1.6.3 *Let U be a disjunctive form of $F_1, \ldots, F_n$, for some $n \geq 2$. Prove that $U = (V \vee W)$, and there is an integer k such that $1 \leq k < n$, V is a disjunctive form of $F_1 \ldots, F_k$, and W is a disjunctive form of $F_{k+1}, \ldots F_n$.*

Hint: Represent the meta-formula U as a tree.

Exercise 1.6.4 (Hard) *Use the preceding exercise to compute the number of disjunctive forms of $F_1, \ldots, F_n$.*

Exercise 1.6.5 *Let $n > 1$ and let $F_1, F_2, \ldots, F_n$ be n meta-variables. Show that any disjunctive form of $F_1, F_2, \ldots, F_n$ is a string of the form $S_0 F_1 S_1 F_2 \ldots S_{n-1} F_n S_n$, where S_0 is a non-empty string of left parentheses, S_n is a non-empty string of right parentheses, and the other S_i's ($1 \leq i \leq n-1$ consist of (possibly empty) string of right parentheses, a $\vee$ sign and a (possibly empty) string of left parentheses.*

Exercise 1.6.6 *List all disjunctions with support G_1, G_1, G_2, G_1.*

Exercise 1.6.7 *List all disjunctive forms that generate the disjunction* $((P_1 \wedge \neg P_2) \vee ((\neg P_3 \vee P_2) \vee P_4))$.

Exercise 1.6.8 *A clause is said to be Horn if it contains at most one positive literal , i.e. literal that is an atom. For example,* $\neg P_1$, $\neg P_1 \vee P_2 \vee \neg P_4$, P_6 *are Horn clauses.* $P_1 \vee P_2$ *is not a Horn clause because it contains two positive literals,* P_1 *and* P_2*. How many nonequivalent Horn clauses can be defined with* n *atoms?*

Exercise 1.6.9 *For each of the following pairs of clauses, find a truth assignment that is a model for one and a counter-model for the other.*

1. $C_1 = (\neg P \vee Q) \vee \neg R$ *and* $C_2 = (P \vee Q) \vee \neg R$.
2. $C_1 = (P \vee \neg Q) \vee \neg R$ *and* $C_2 = P \vee \neg R$.
3. $C_1 = (P \vee Q) \vee \neg P$ *and* $C_2 = (P \vee Q) \vee \neg R$.

Exercise 1.6.10 *List all conjunctions with support* P_1, P_2, P_3, P_4, P_5.

Exercise 1.6.11 *List all possible supports of the conjunction* $((P_1 \wedge P_3) \wedge ((P_3 \longrightarrow \neg P_4) \wedge (P_1 \wedge \neg P_3)))$.

Exercise 1.6.12 *Prove Lemma 1.6.25.*

Exercise 1.6.13 *How many nonequivalent conjunctions of literals can we form with* n *atoms (and their complements)?*

Exercise 1.6.14 *Let* C_1 *and* C_2 *be two conjunctions. Prove that if the conjunct set of* C_1 *is a subset of the conjunct set of* C_2, $C_1 \vee C_2 \equiv C_1$.

Exercise 1.6.15 *Let* D_1 *and* D_2 *be two disjunctions. Prove that if the set of disjuncts of* D_1 *is a subset of the set of disjuncts of* D_2 *then* $D_1 \wedge D_2 \equiv D_1$.

1.7 Normal Forms

One of the most important problems of logic is deciding if a formula is or is not a tautology. Since F is a tautology iff $\neg F$ is unsatisfiable, this question reduces to the *decision problem*, i.e. determining if a formula is unsatisfiable. For propositional logic, we have an algorithm that solves the decision problem: we construct a truth table and look at the column that stores the values of the formula. If all of them are 0, the formula is unsatisfiable, otherwise it is not. There are several problems with this approach. First of all it is inefficient since the entries in the table increase exponentially in the number of atoms. Second, and, more important, it does not carry over to higher order logics.

For this reason, we introduce another method, *the resolution method*, that also works for first order logic. This method will be discussed in the next section. The resolution method applies only to a special set of formulas, called conjunctive normal forms (CNF's).

In this section we define the CNF's as well as the related class of disjunctive normal forms (DNF's). We prove that for every formula there are CNF's and DNF's equivalent to it, by constructing algorithms that compute the normal forms and then proving that the algorithms are correct. The section ends with a theorem that relates the DNF's and the CNF's.

Definition 1.7.1 (conjunctive normal forms) *1. A conjunctive normal form, abbreviated CNF, is a formula* $F = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n_i} L_{ij}$ *where* L_{ij}, $1 \leq i \leq n$, $1 \leq j \leq n_i$, *are literals. If* $n = 0$, F *is a tautology.*

2. A conjunctive normal form of the formula G *is a conjunctive normal form equivalent to* G.

Definition 1.7.1 tells us that a CNF is a conjunction of clauses, and both the conjunction and the clauses are in prefix form. The formula has n conjuncts, the i-th clause has n_i literals and L_{ij} is the j-th disjunct of the i-th conjunct. If $n_i = 0$, the i-th conjunct is the unsatisfiable clause $\Box$.

Example 1.7.2 The formula $((((P_1 \vee \neg P_2) \vee P_3) \wedge (\neg P_3 \vee P_2)) \wedge P_3)$ is a conjunctive normal form.

It has $n = 3$ conjuncts, $((P_1 \vee \neg P_2) \vee P_3)$, $(\neg P_3 \vee P_2)$, and P_3. Each conjunct is a disjunction of literals, more precisely a clause in prefix form. The clause $((P_1 \vee \neg P_2) \vee P_3)$ has $n_1 = 3$ literals, $L_{11} = P_1$, $L_{12} = \neg P_2$, and $L_{13} = P_3$. The second clause, $(\neg P_3 \vee P_2)$, has $n_2 = 2$ literals, $L_{21} = \neg P_3$ and $L_{22} = P_2$. The last clause, P_3, has $n_3 = 1$ literals, $L_{31} = P_3$.

We can characterize the normal forms better if we understand the notion of **scope** of a connective.

Definition 1.7.3 (the scope of a connective occurrence) *Let* G *be a formula occurrence in* F. *If* $G = \neg H$, *the* **scope** *of the connective* $\neg$ *is the occurrence of* H.

If $G = (HCI)$ *where* C *is one of the connectives* $\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$, *the scope of* C *is the occurrences of* G *and* H.

Remark 1.7.4 We remember that an occurrence is a substring *plus* the starting address of the substring. The scopes are defined for occurrences of connectives, not for connectives. For example, the formula $F = (\neg P_1 \vee \neg P_2)$ has two occurrences of $\neg$, at addresses 1 and 4. The scope of the first occurrence of $\neg$ is the substring P_1 and the scope of the second, P_2.

Address	0	1	2	3	4	5	6
F Symbol	(	$\neg$	P_1	$\vee$	$\neg$	P_2	)

The scope is also one or two occurrences. For example, the formula $G = ((P_1 \wedge P_2) \vee (P_1 \wedge P_2))$ has two occurrences of $\wedge$, at addresses 3 and 9. Both occurrences have as scope the same 2 subformulas, P_1 and P_2. However, the scope of the first are the occurrence of P_1 at 2 and the occurrence of P_2 at 4, while the scope of the second connective occurrence consists of the occurrence of P_1 at 8 and the occurrence of P_2 at 10.

Address	0	1	2	3	4	5	6	8	9	10	11	12
G Symbol	(	(	P_1	$\wedge P_2$	)	$\vee$	(	P_1	$\wedge$	P_2	)	)

When the scope has only one occurrence, we will not specify the address at which it occurs.

Example 1.7.5 Let us find the scopes of the connectives from the formula $F = ((P_1 \vee \neg P_3) \vee \neg(P_4 \wedge P_5))$.

The scope of the first negation is P_3, and the scope of the second is $(P_4 \wedge P_5)$. The scope of the first $\vee$ consists of the strings P_1, $\neg P_3$, and the scope of the second, the strings $(P_1 \vee \neg P_3)$ and $\neg(P_4 \wedge P_5))$. The scope of $\wedge$ consists of the sets P_4 and P_5. We notice that the first occurrence of $\vee$ is *within* the scope of the second occurrence of $\vee$.

Now we can introduce the notion of *dominance*.

Definition 1.7.6 (the dominance relation) *Let C and D be two occurrences of the operators $\neg$, $\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$ in the formula F. We say that the operator occurrence C* **dominates** *the operator occurrence D if D is within the scope of C.*

Examples 1.7.7 1. In the formula $F = (\neg(P_1 \wedge \neg P_2) \vee \neg P_4)$ the first occurrence of $\neg$ dominates the occurrence of $\wedge$ as well as the second occurrence of $\neg$, because its scope, $(P_1 \wedge \neg P_2)$, includes these occurrences.

2. In the formula $(F \vee \neg G) \wedge H$, the scope of $\wedge$ consists of the formulas $(F \vee \neg G)$ and H. The scope includes the $\vee$ and the $\neg$ connectives, so $\wedge$ dominates them.

3. In the formula $(F \vee G) \wedge (F \vee H)$ the scope of the first $\vee$ consists of the first occurrence of F and G. The scope of the second $\vee$ consists of the second occurrence of F and H. Neither scope includes the other operator, so neither $\vee$ dominates the other. On the other side, $\wedge$ dominates both occurrences of $\vee$ because its scope is $(F \vee G)$ and $(F \vee H)$.

Example 1.7.8 It is easy to see the dominance relation on formula trees. Let us find the dominance relation on the set of connectives of the formula $F = \neg((\neg P_1 \vee (\neg P_3 \wedge P_1)) \longleftrightarrow \neg P_4)$. Figure 1.50 shows the tree of this formula. The first $\neg$ of F has address λ, the second one 0.0.0, the third 0.0.1.0 and the fourth 0.1.

Now the scope of the operator at address A consists of the descendents of A. Since A is a *prefix* of all its descendent addresses, the operator at A dominates the operator at B iff A is a prefix of B and $A \neq B$!

In Figure 1.50 the first occurrence of $\neg$ dominates all other connective occurrences; $\longleftrightarrow$ dominates $\vee$, $\wedge$, and the second, the third and fourth occurrence of $\neg$. The occurrence of $\vee$ dominates the second and the third occurrence of $\neg$ and the occurrence of $\wedge$. Finally, the occurrence of $\wedge$ dominates the third occurrence of $\neg$.

We will use the dominance relation to characterize the conjunctive normal forms. First, let us draw the formula tree for $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{n_i} L_{ij}$. We can verify that the tree from Figure 1.51 satisfies the 5 conditions listed in Figure 1.52. We will prove that any formula that satisfies the conditions listed in Figure 1.52 is a

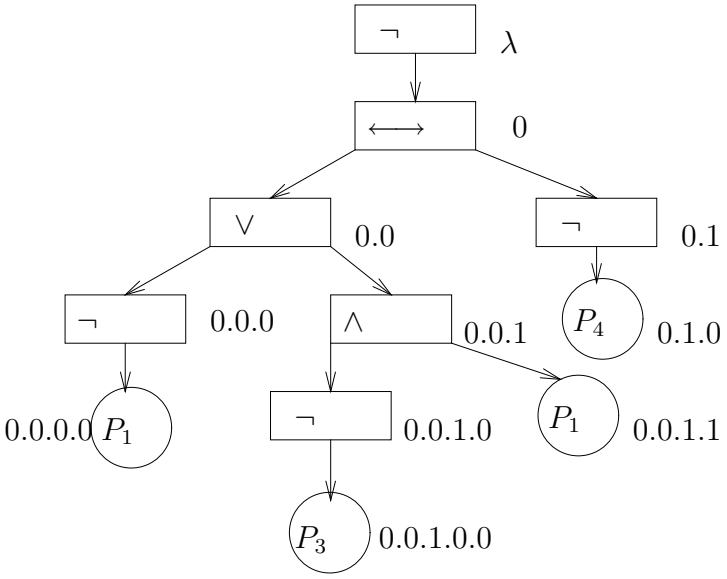


Figure 1.50: The formula tree for Example 1.7.8

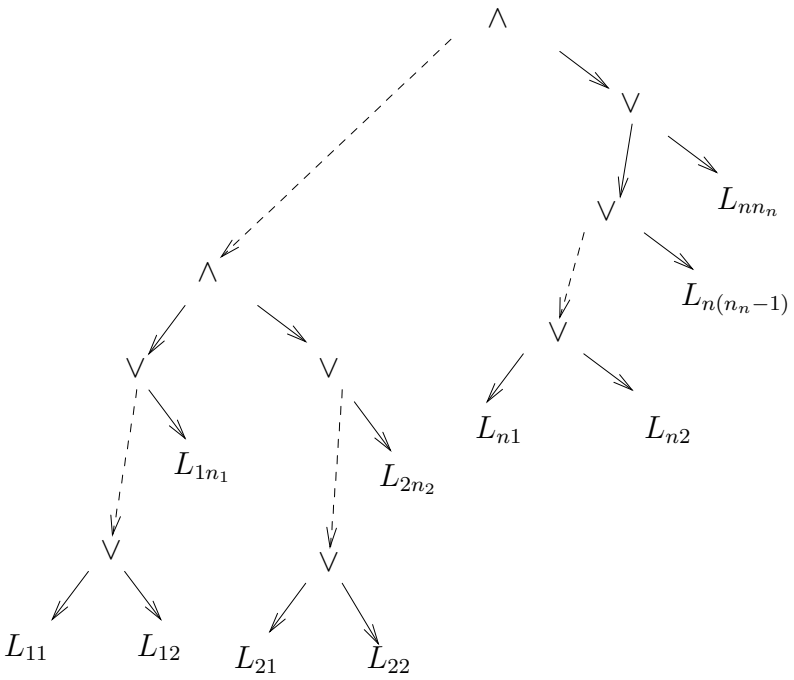


Figure 1.51: The formula tree for a CNF

1. The formula has no $\longleftrightarrow$ connectives.
2. The formula has no $\longrightarrow$ connectives.
3. No $\neg$ occurrence dominates a $\vee$, a $\wedge$ or another $\neg$ occurrence.
4. No $\vee$ occurrence dominates a $\wedge$ occurrence.
5. All conjunctions and disjunctions are in prefix form.

Figure 1.52: The properties of a CNF

conjunctive normal form. First we need some preliminary lemmas (or lemmata if one follows the Greek grammar).

Lemma 1.7.9 *Let F be a formula that satisfies the conditions given in Figure 1.52. If $n[\wedge, F] > 0$ then $F = G \wedge H$, G and H satisfy the conditions of Figure 1.52 and $n[\wedge, H] = 0$.*

Proof: Since F contains a $\wedge$ sign, it is not an atom. Then it has one of the following 5 forms: $F = \neg I$, $F = I \vee J$, $F = I \wedge J$, $F = I \longleftrightarrow J$, $F = I \longrightarrow J$.
Case 1: $F = \neg I$. Since F has $\wedge$ connectives, they must be inside I. This means that a $\neg$ dominates a $\wedge$, a contradiction to condition 3.
Case 2: $F = I \longleftrightarrow J$. This case contradicts condition 1.
Case 3: $F = I \longrightarrow J$. This case contradicts condition 2.
Case 4: $F = I \vee J$. Since F contains $\wedge$ symbols, then at least one of the subformulas I and J must have some. In either case, $\vee$ dominates a $\wedge$, a contradiction to condition 4.

So, $F = G \wedge H$ for some G and H. Since F satisfies the conditions listed in Figure 1.52, so do its subformulas. So, G and H satisfy the conditions.

If $n[\wedge, H] > 0$ then, by an analysis similar to the one we made for F, $H = I \wedge J$ for some formulas I and J. But then $F = G \wedge (I \wedge J)$ is no longer a prefix form, contradicting condition 5. **Q.E.D.**

Now let us look at the formulas F that satisfy the properties of a CNF and have no $\wedge$'s, i.e. $n[\wedge, F] = 0$. These formulas contain only the connectives $\vee$ and $\neg$. The next lemma characterizes them.

Lemma 1.7.10 *If F satisfies Figure 1.52 and $n[\wedge, F] = 0$, F is a clause in prefix form.*

Proof: We prove it by induction on the number of $\vee$'s in F.
Basis: $n[\vee, F] = 0$. Then F has only $\neg$ connectives and atoms. Since an occurrence of $\neg$ cannot dominate another occurrence of $\neg$, we can have at most one $\neg$ in front of the atoms. Since F is either an atom or the negation of an atom, F is a literal. Then F is a prefix form.
Inductive Step: Assume that the statement is true for all formulas with n or fewer $\vee$ connectives. Let G be a clause with $n + 1$ $\vee$'s. If $G = \neg H$ then $\neg$ dominates a $\vee$, contrary to condition 3 of the CNF properties. So, $G = H \vee I$. If $n[\vee, I] > 0$, reasoning the same way we did with G, we get that $I = J \vee K$. But then $G = H \vee (J \vee K)$ is no longer a prefix form, contradicting condition 5 of the CNF properties.

So, $G = H \vee I$ where H and I satisfy the conditions of the lemma, H has n $\vee$'s and I has none. By the induction hypothesis, G is a clause in prefix form. By the basis of the proof, I is literal. Then $G = H \vee I$ is a clause in prefix form. **Q.E.D.**

Theorem 1.7.11 *If F satisfies the CNF properties (Figure 1.52), then it is a CNF.*

Proof: We prove the proposition by induction on n, the number of $\wedge$'s in the formula.
Basis: $n = 0$. Since F satisfies Lemma 1.7.10, F is a clause in prefix form. Then F is a CNF of the type $\bigwedge_{i=1}^{1} \bigvee_{j=1}^{n_1} F_{ij}$, $1 \leq j \leq n_1$.
Inductive Step: Assume that the theorem holds for all formulas F with n $\wedge$'s. Let G be a formula with $n+1$ $\wedge$'s. By Lemma 1.7.9, $G = H \wedge I$, H and I satisfy the CNF-properties and $n[\wedge, I] = 0$. From the basis of the theorem, I is a clause in prefix form, i.e. $I = \bigvee_{k=1}^{l} N_k$. The formula H has n $\wedge$'s so, by IH, $H = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n_i} L_{ij}$. Then

$G = H \wedge I = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n_i} L_{ij} \wedge \bigvee_{k=1}^{l} N_k$.

The above formula can be written as

$\bigwedge_{i=1}^{n+1} \bigvee_{j=1}^{n_i} L_{ij}$,

where $n_{n+1} = l$ and for all $1 \leq j \leq l$, $L_{n+1,j} = N_j$. So, G is a CNF. **Q.E.D.**

Theorem 1.7.11 and the observation that all CNF's satisfy the CNF properties tell us that the **CNF**'s are precisely the formulas that satisfy the CNF-properties. We will use the CNF properties to construct an algorithm that takes as input a formula and computes a CNF for the formula.

Algorithm 1.7.12 (The CNF Algorithm) Step 1. Eliminate $\longleftrightarrow$.

We keep applying the simplification

$G \longleftrightarrow H \Longrightarrow (G \longrightarrow H) \wedge (H \longrightarrow G)$

to the formulas $G \longleftrightarrow H$ that have no $\longleftrightarrow$'s inside, until we cannot do it.
Step 2. Eliminate $\longrightarrow$.

We simplify $G \longrightarrow H$ to $\neg G \vee H$, as long as we have occurrences of $\longrightarrow$.
Step 3. Push the negation all the way inward until it rests on atoms.

We keep applying the 3 simplifications below until there are no more occurrences of the left-hand-side of the rules.

1. $\neg(G \vee H) \Longrightarrow \neg G \wedge \neg H$
2. $\neg(G \wedge H) \Longrightarrow \neg G \vee \neg H$
3. $\neg\neg H \Longrightarrow H$.

Step 4. Distribute $\vee$ over $\wedge$.

We use the simplifications below as long as we have occurrences of the left-hand-sides of the rules.

$G \vee (H \wedge I) \Longrightarrow (G \vee H) \wedge (G \vee I)$

$(G \wedge H) \vee I \Longrightarrow (G \vee I) \wedge (H \vee I)$

Step 5. Simplify.

We use the following simplifications:

1. If a conjunction contains an atom Q and its negation $\neg Q$, or if it has $\Box$ as a conjunct, the conjunction reduces to $\Box$.

2. Eliminate tautology clauses.

3. If a conjunction contains the clauses C and $C \vee D$, eliminate $C \vee D$.

Step 6. Put the formula in prefix form.

Before we go on to prove the correctness of The CNF Algorithm, let us use it to compute the CNF of a formula.

Example 1.7.13 Let us compute a **CNF** of $F = \neg((A \wedge B) \longleftrightarrow (B \vee C))$.

Step 1. Eliminate $\longleftrightarrow$.

We replace $(A \wedge B) \longleftrightarrow (B \vee C)$ by $((A \wedge B) \longrightarrow (B \vee C)) \wedge ((B \vee C) \longrightarrow (A \wedge B))$. We get

$F_1 = \neg(((A \wedge B) \longrightarrow (B \vee C)) \wedge ((B \vee C) \longrightarrow (A \wedge B)))$.

Step 2. Eliminate $\longrightarrow$.

In F_1 we replace $(A \wedge B) \longrightarrow (B \vee C)$ by $\neg(A \wedge B) \vee (B \vee C)$ and $(B \vee C) \longrightarrow (A \wedge B)$ by $\neg(B \vee C) \vee (A \wedge B)$.

We get $F_2 = \neg((\neg(A \wedge B) \vee (B \vee C)) \wedge (\neg(B \vee C) \vee (A \wedge B)))$.

Step 3. Push the negation inward.

$\neg((\neg(A \wedge B) \vee (B \vee C)) \wedge (\neg(B \vee C) \vee (A \wedge B)))$

$\equiv \neg(\neg(A \wedge B) \vee (B \vee C)) \vee \neg(\neg(B \vee C) \vee (A \wedge B))$ by DeMorgan's law

$\equiv (\neg\neg(A \wedge B) \wedge \neg(B \vee C)) \vee (\neg\neg(B \vee C) \wedge \neg(A \wedge B))$ DeMorgan's twice

$\equiv ((A \wedge B) \wedge \neg(B \vee C)) \vee ((B \vee C) \wedge \neg(A \wedge B))$ $\neg\neg$-elimination

$\equiv ((A \wedge B) \wedge (\neg B \wedge \neg C)) \vee ((B \vee C) \wedge (\neg A \vee \neg B)) = F_3$, DeMorgan's twice

Step 4. Distribute $\vee$ over $\wedge$.

In the formula $F_3 = ((A \wedge B) \wedge (\neg B \wedge \neg C))\underline{\vee}((B \vee C)\underline{\wedge}(\neg A \vee \neg B))$ the underlined $\vee$ dominates the underlined $\wedge$. So let us apply the distributivity equivalence $G \vee (H \wedge I) \equiv (G \vee H) \wedge (G \vee I)$. We get

$(((A \wedge B) \wedge (\neg B \wedge \neg C)) \vee (B \vee C)) \wedge (((A \wedge B) \wedge (\neg B \wedge \neg C)) \vee (\neg A \vee \neg B))$

But the underlined occurrences of $\vee$ still dominate the underlined occurrences of $\wedge$.

$(((A \wedge B)\underline{\wedge}(\neg B \wedge \neg C))\underline{\vee}(B \vee C)) \wedge (((A \wedge B)\underline{\wedge}(\neg B \wedge \neg C))\underline{\vee}(\neg A \vee \neg B))$

So, let us apply the distributivity equivalence again and get

$(((A \wedge B) \vee (B \vee C)) \wedge ((\neg B \wedge \neg C) \vee (B \vee C))) \wedge (((A \wedge B) \vee (\neg A \vee \neg B)) \wedge ((\neg B \wedge \neg C) \vee (\neg A \vee \neg B)))$.

The underlined $\vee$'s still dominate the underlined $\wedge$'s.

$(((A\underline{\wedge}B)\underline{\vee}(B \vee C)) \wedge ((\neg B\underline{\wedge}\neg C)\underline{\vee}(B \vee C))) \wedge (((A\underline{\wedge}B)\underline{\vee}(\neg A \vee \neg B)) \wedge ((\neg B\underline{\wedge}\neg C)\underline{\vee}(\neg A \wedge \neg B)))$

We apply the distributivity 4 times and get

$F_4 = (((A \vee (B \vee C)) \wedge (B \vee (B \vee C))) \wedge ((\neg B \vee (B \vee C)) \wedge (\neg C \vee (B \vee C)))) \wedge (((A \vee (\neg A \vee \neg B)) \wedge (B \vee (\neg A \vee \neg B))) \wedge ((\neg B \vee (\neg A \vee \neg B)) \wedge (\neg C \vee (\neg A \vee \neg B))))$.

Step 5. Simplify.

Let us look at the conjuncts of F_4. We will write the clauses as sets of literals.

$C_1 = \{A, B, C\}, C_2 = \{B, B, C\}, C_3 = \{\neg B, B, C\}, C_4 = \{\neg C, B, C\}, C_5 = \{A, \neg A, \neg B\}, C_6 = \{B, \neg A, \neg B\}, C_7 = \{\neg B, \neg A\}, C_8 = \{\neg C, \neg A, \neg B\}$.

It is good practice to list the atoms of the sets in alphabetical order, so A is before $\neg A$, $\neg A$ is before B, B is before $\neg B$ and so on. This way we do not list

the same element twice and it is easier to check if two clauses are equal, or if one clause absorbs the other. We do not change the sets since listing the same element several times and changing the order of the elements do not change the set. So, our clauses are

$C_1 = \{A, B, C\}, C_2 = \{B, C\}, C_3 = \{B, \neg B, C\}, C_4 = \{B, C, \neg C\}, C_5 = \{A, \neg A, \neg B\}, C_6 = \{\neg A, B, \neg B\}, C_7 = \{\neg A, \neg B\}, C_8 = \{\neg A, \neg B, \neg C\}$.

First we remove the tautologies. We remember from Section 1.6 that a clause is a tautology if and only if it contains both an atom and its negation. Since the elements of the set are ordered, we need only to check if an atom of the set is immediately followed by its negation. We see that C_3, C_4, C_5, C_6 are tautologies, so we remove them and we are left with $C_1 = \{A, B, C\}, C_2 = \{B, C\}, C_7 = \{\neg A, \neg B\}, C_8 = \{\neg A, \neg B, \neg C\}$.

Next we use the absorption properties to delete the clauses that are proper supersets of other clauses. We say that a clause C is a proper superset of clause D if every literal of D is in C, but some literals of C are not in D. Since the clauses are ordered we need only to check that D are embedded in E, i.e. the atoms of D occur in C in the same order in which they occur D.

Since C_1 is a proper superset of C_2, we eliminate C_1; C_8 is a proper superset of C_7 so we eliminate C_8.

We are left with the clauses $C_2 = \{B, C\}$ and $C_7 = \{\neg A, \neg B\}$.

Step 6. Compute the prefix form.

We use left parenthesization for clauses and for the conjunction. We get the conjunctive normal form

$F_6 = (B \vee C) \wedge (\neg A \vee \neg B)$.

Note 1.7.14 We could have used a simplification at step 4, by observing that $((A \wedge B) \wedge (\neg B \wedge \neg C))$ is unsatisfiable. Then $F_3 \equiv (B \vee C) \wedge (\neg A \vee \neg B)$ a CNF.

This way we avoid doing a lot of unnecessary work. The moral is that we should apply the simplifications whenever we can, not only in the last step.

Theorem 1.7.15 (The CNF Theorem) *Algorithm 1.7.12 is correct, i.e. every formula has a CNF.*

Proof: For any input F, the algorithm computes the output as a sequence

$F = F_0 \overset{Step1}{\Longrightarrow} F_1 \overset{Step2}{\Longrightarrow} F_2 \overset{Step3}{\Longrightarrow} F_3 \overset{Step4}{\Longrightarrow} F_4 \overset{Step5}{\Longrightarrow} F_5 \overset{Step6}{\Longrightarrow} F_6$.

We will show that each step terminates, F_1 satisfies the first condition of the CNF properties, F_2 the first 2, F_3 the first 3, F_4 the first 4, and F_6 all 5.

Step 1 is correct.

If G and H contain no $\longleftrightarrow$'s, the simplification

$G \longleftrightarrow H \Longrightarrow (G \longrightarrow H) \wedge (H \longrightarrow G)$

decreases the number of $\longleftrightarrow$'s, since the right-hand-side has no $\longleftrightarrow$'s. So, Step 1 terminates. Now let us show that the output F_1 has no $\longleftrightarrow$'s. Assume that it does. Then, it has a $G_0 \longleftrightarrow H_0$ subformula of minimal length. If G_0 or H_0 contain $\longleftrightarrow$'s then F_1 has a shorter subformula $G \longleftrightarrow H$, than $G_0 \longleftrightarrow H_0$, contradicting the fact that $G_0 \longleftrightarrow H_0$ has minimal length. So, $G_0 \longleftrightarrow H_0$ has

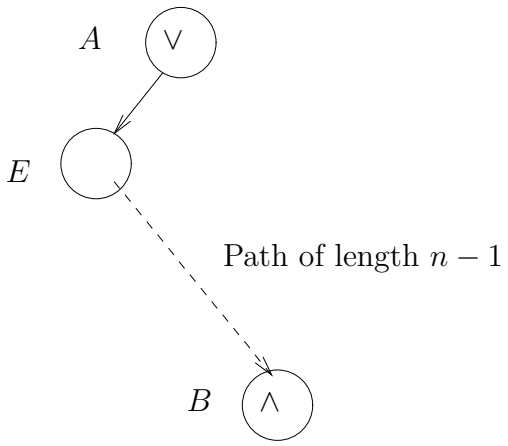


Figure 1.53: The formula tree for the claim of Theorem 1.7.15

no $\longleftrightarrow$'s, and will be reduced. This contradicts the fact that F_1 is the output of Step 1. Since all simplifications are equivalences, $F_1 \equiv F_0$.
Step 2 is correct.

The simplification $G \longrightarrow H \Longrightarrow \neg G \vee H$ decreases the number of $\longrightarrow$'s of F_1, so Step 2 terminates and its output F_2 has no $\longrightarrow$'s. It has no $\longleftrightarrow$'s because F_1 has none and the step did not introduce any. The simplification is derived from an equivalence, so $F_2 \equiv F_1$.
Step 3 is correct.

The termination of the step will be proved later on, when we discuss the *complexity functions*. Since F_2 has $\longleftrightarrow$'s and no $\longrightarrow$'s neither does F_3. Now let us assume that a $\neg$ occurrence of F_3 dominates another connective. Then, F_3 has a subformula $\neg G$ that contains one of the connectives $\neg$, $\vee$, or $\wedge$. Since G cannot be an atom, it must be equal, for some I and J, to $\neg J$, $(I \vee J)$, or $(I \wedge J)$. Then, $\neg G = \neg\neg I$, or $\neg G = \neg(I \vee J)$, or $\neg G = \neg(I \wedge J)$. But these formulas can be simplified by the step reductions, contradicting the fact that F_3 is the output of Step 3.
Step 4 is correct.

Again, the termination of the simplification will be proved with complexity functions. Let us show that in F_4 no $\vee$ dominates a $\wedge$. Assume that it does. Then, in the tree representation of F_4, the address A of $\vee$ is a prefix of the address B of $\wedge$. We claim that there is a pair of addresses C, D such that D is a child of C, the label of C is $\vee$ and the label of D is $\wedge$.

The proof of the claim is by induction on the length of the path from A to B. If the length is 1, then $C = A$ and $D = B$.

If the length is $n > 1$, we have the situation shown in Figure 1.53. Let E be the address of the child of A that is on the path to B. The label of E cannot be $\neg$ because $\neg$ would dominate the $\wedge$ at B. So, it must be $\vee$ or $\wedge$. If it is $\wedge$, then the pair A, E satisfies the claim. If not, the length of the path from E to

B is $n-1$ and the IH tells us that we can find a pair C, D that satisfies the claim. **Q.E.D. claim**

The claim tells us that the subformula of F_4 at address C has the form $I \vee (J \wedge K)$ or $(J \wedge K) \vee I$, depending on whether $D = C.0$ or $D = C.1$. In either case, it can be simplified, contradicting the assumption that F_4 is the output of Step 4.

The Step 4 simplifications rearrange the $\vee$ and the $\wedge$ connectives; if the left-hand-side has no $\longleftrightarrow$'s, no $\longrightarrow$, and the negation does not dominate another connective then neither does the right-hand-side. So, F_4 satisfies conditions 1-4 of the CNF properties.

Again, the simplifications of this step are oriented equivalences, so $F_4 \equiv F_3$.
Step 5 is correct.

The termination is easy because every simplification reduces the length of the formula. At the same time, the reductions are eliminating clauses, so they do not modify the dominance relations between the remaining connectives, nor do they reintroduce $\longleftrightarrow$'s or $\longrightarrow$'s. So, F_5 also satisfies the first 4 conditions of Figure 1.52.

All simplifications of this step are equivalences, so $F_5 \equiv F_4$.
Step 6 is correct[8].

We will show that the simplification

$I \wedge (J \wedge K) \Longrightarrow (I \wedge J) \wedge K$

computes the prefix form of the conjunction and the simplification

$I \vee (J \vee K) \Longrightarrow (I \vee J) \vee K$

does the same thing for disjunctions. We apply these simplifications, in any order, until we cannot do anymore. The termination of these reductions will be proved with complexity functions. For now, let us show that the output of this step is correct. These simplifications change only the dominance between $\vee$'s and between $\wedge$'s, so F_6 satisfies the first 4 conditions of the CNF properties.

Now let us assume that the output of these reductions is not a CNF. Then either a disjunct, or the conjunction are not in prefix form. Assume that the clause C of F_6 is not in prefix form. Let $L_1, \ldots, L_n$ be the literals of C in the order in which they occurs in C. If $n = 1$ or $n = 2$ then C is a prefix form, so n must be greater than 2. Let $C = D_{n-1} \vee E$. If $E = M \vee N$ then we can apply the simplification

$D_n \vee (M \vee N) \longrightarrow (D \vee M) \vee N$

to F_6, contradicting the fact that F_6 is the output of these 2 reductions. So, $E = L_n$. If D_{n-1} is a prefix form, so is $D_{n-1} \vee L_n$. Since we assumed that C is not a prefix form, D_{n-1} must not be a prefix form. The literals of D_{n-1} are $L_1, \ldots, L_{n-1}$. Now we apply the same reasoning to D_{n-1} as we did to C, and get that $D_{n-1} = D_{n-2} \vee L_{n-1}$, D_{n-2} is a disjunction of $L_1 \ldots, L_{n-2}$, and D_{n-2} is not a prefix form. We repeat this procedure until we get to D_2. Then D_2 is a prefix form, because it has only two literals. So, we have a contradiction.

[8] We can also prove the correctness by using the results from the preceding section, that tells us that the conjunctions and the disjunctions have prefix forms.

1. The formulas have no $\longleftrightarrow$ connectives.
2. The formulas have no $\longrightarrow$ connectives.
3. No $\neg$ occurrence dominates an occurrence of $\vee$, $\wedge$ or $\neg$.
4. No $\wedge$ occurrence dominates a $\vee$ occurrence.
5. All the conjunctions and the disjunctions are in prefix form.

Figure 1.54: The properties of a DNF

The proof that the conjunction is a prefix form is proved the same way. The 2 simplifications are equivalences, so $F_6 \equiv F_5$. Then $F_1 \equiv F_2 \equiv F_3 \equiv F_4 \equiv F_5 \equiv F_6$.**Q.E.D.**

Observation 1.7.16 We will show that every formula has a countably infinite set of CNF's. Let $G = (\ldots((C_1 \wedge C_2) \wedge \ldots \wedge C_{n-1}) \wedge C_n$ be a CNF of F. Now let Q be an atom that is not in C. We can check, by truth tables, that $C \equiv (C \vee Q) \wedge (C \vee \neg Q)$. This equivalence implies that $G_Q = ((\ldots((C_1 \wedge C_2) \wedge \ldots \wedge C_{n-1}) \wedge (C_n \vee Q)) \wedge (C_n \vee \neg Q)$ is equivalent to G. Since C is in prefix form, so are $C \vee Q$ and $C \vee \neg Q$. So all clauses of G_Q are in prefix form. The conjunction is also in prefix form, so G_Q is a CNF of F. Since the number of Q's is countably infinite, the set of CNF's of F is at least countably infinite. We recall that the set of all formulas, $FORM$, is countably infinite, so the number of CNF's of F is countably infinite by the Cantor-Bernstein Theorem.

The development of the disjunctive normal forms is almost identical to that of the conjunctive forms. The only difference is that we switch the places of $\vee$'s and $\wedge$'s, $\Box$ and $\mathbf{T}$. For this reason, we will not repeat the proofs, nor state the lemmas. We will give only the definitions, the main results, and the examples.

Definition 1.7.17 (DNF) *A disjunctive normal form, abbreviated* **DNF***, is a formula* $F = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} L_{ij}$ *where* L_{ij}, $1 \leq i \leq n$, $1 \leq j \leq n_i$, *are literals. If* $n = 0$ *then* $F = \Box$, *the unsatisfiable formula.*

A disjunctive normal form of G *is a disjunctive normal form equivalent to* G.

In Definition 1.7.17, n is the number of disjuncts, n_i is the number of literals of the i-th disjunct, and L_{ij} is the j-th conjunct of disjunct i. If $n_i = 0$ then the disjunct i is the tautology $\mathbf{T}$. The normal forms use the prefix forms for both conjunctions and disjunctions.

Example 1.7.18 The formula $((P_3 \vee ((\neg P_1 \wedge \neg P_3) \wedge P_4)) \vee (P_2 \wedge \neg P_3)) \vee (\neg P_1 \wedge P_4)$ is a disjunctive normal form with $n = 4$, $n_1 = 1$, $n_2 = 3$, $n_3 = 2$, $n_4 = 2$ and

$L_{11} = P_3$, $L_{21} = \neg P_1$, $L_{22} = \neg P_3$, $L_{23} = P_4$, $L_{31} = P_2$, $L_{32} = \neg P_3$, $L_{41} = \neg P_1$, $L_{42} = P_4$.

We can verify that the DNF's $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} L_{ij}$ satisfy the 5 DNF properties listed in Figure 1.54. So, we state the analogue of Theorem 1.7.11.

Theorem 1.7.19 *If* F *satisfies the conditions listed in Figure 1.54, then it is a DNF.*

We use the DNF properties to generate Algorithm 1.7.20.

Algorithm 1.7.20 (The DNF algorithm) Step 1: Eliminate $\longleftrightarrow$.

We keep applying the simplification

$G \longleftrightarrow H \Longrightarrow (G \longrightarrow H) \wedge (H \longrightarrow G)$

to the formulas $G \longleftrightarrow H$ that have no $\longleftrightarrow$'s inside, until we cannot do it.

Step 2: Eliminate $\longrightarrow$.

We simplify $G \longrightarrow H$ to $\neg G \vee H$, as long as we have occurrences of $\longrightarrow$.

Step 3: Push the negation all the way inward until it rests on atoms.

We keep applying the 3 simplifications below until there are no more occurrences of the left-hand-side of the rules.

1. $\neg(G \vee H) \Longrightarrow \neg G \wedge \neg H$
2. $\neg(G \wedge H) \Longrightarrow \neg G \vee \neg H$
3. $\neg\neg H \Longrightarrow H$.

Step 4: Distribute $\wedge$ over $\vee$.

Apply the two simplifications below while there are occurrences of the left-hand-sides of the rules.

$G \wedge (H \vee I) \Longrightarrow (G \wedge H) \vee (G \wedge I)$

$(G \wedge H) \vee I \Longrightarrow (G \wedge I) \vee (H \wedge I)$

Step 5: Simplify.

We use the following reductions:

1. Reduce to $\mathbf{T}$ any disjunction that contains a formula G and its negation $\neg G$ and any disjunction that contains $\mathbf{T}$.

2. Eliminate the conjuncts that are unsatisfiable, i.e. they contain both an atom Q and its negation $\neg Q$.

3. If the set of disjuncts contains both C and $C \wedge D$, then eliminate $C \wedge D$.

Step 6: Put the formula in prefix form.

Example 1.7.21 Let us apply Algorithm 1.7.20 to compute a disjunctive normal form for $F = (A \longrightarrow B) \longleftrightarrow \neg(B \vee C)$.

Step 1: Eliminate $\longleftrightarrow$.

We replace $(A \longrightarrow B) \longleftrightarrow \neg(B \vee C)$ by

$((A \longrightarrow B) \longrightarrow \neg(B \vee C)) \wedge (\neg(B \vee C) \longrightarrow (A \longrightarrow B))$

and get

$F_1 = ((A \longrightarrow B) \longrightarrow \neg(B \vee C)) \wedge (\neg(B \vee C) \longrightarrow (A \longrightarrow B))$.

Step 2: Eliminate $\longrightarrow$.

In F_1 we replace $A \longrightarrow B$ by $\neg A \vee B$ and get

$((\neg A \vee B) \longrightarrow \neg(B \vee C)) \wedge (\neg\neg(B \vee C) \longrightarrow (\neg A \vee B))$.

Now we replace $(\neg A \vee B) \longrightarrow \neg(B \vee C)$ by $\neg(\neg A \vee B) \vee \neg(B \vee C)$ and $\neg(B \vee C) \longrightarrow (\neg A \vee B)$ by $\neg\neg(B \vee C) \vee (\neg A \vee B)$.

We get $F_2 = (\neg(\neg A \vee B) \vee \neg(B \vee C)) \wedge (\neg\neg(B \vee C) \vee (\neg A \vee B))$.

Step 3: Push $\neg$ inward.

We replace $\neg(\neg A \vee B)$ by $\neg\neg A \wedge \neg B$, $\neg(B \vee C)$ by $\neg B \wedge \neg C$,

$\neg\neg(B \vee C)$ by $(B \vee C)$ and we get

$((\neg\neg A \wedge \neg B) \vee (\neg B \wedge \neg C)) \wedge ((B \vee C) \vee (\neg A \vee B))$.

Now we eliminate the double negation and we get

$F_3 = ((A \wedge \neg B) \vee (\neg B \wedge \neg C)) \wedge ((B \vee C) \vee (\neg A \vee B))$.
Step 4: Distribute $\wedge$ over $\vee$.

In F_3 the underlined $\wedge$ dominates the underlined $\vee$.

$F_3 = ((A \wedge \neg B) \vee (\neg B \wedge \neg C))\underline{\wedge}((B \vee C)\underline{\vee}(\neg A \vee B))$

We apply the distributivity and get

$(((A \wedge \neg B) \vee (\neg B \wedge \neg C)) \wedge (B \vee C)) \vee ((A \wedge \neg B) \vee (\neg B \wedge \neg C)) \wedge (\neg A \vee B))$.

Now the underlined $\wedge$'s still dominate the underlined $\vee$'s in

$(((A \wedge \neg B)\underline{\vee}(\neg B \wedge \neg C))\underline{\wedge}(B \vee C)) \vee ((A \wedge \neg B)\underline{\vee}(\neg B \wedge \neg C))\underline{\wedge}(\neg A \vee B))$,

so we apply the distributivity twice to get

$(((A \wedge \neg B) \wedge (B \vee C)) \vee ((\neg B \wedge \neg C) \wedge (B \vee C))) \vee (((A \wedge \neg B) \wedge (\neg A \vee B)) \vee ((\neg B \wedge \neg C) \wedge (\neg A \vee B)))$.

The underlined $\wedge$'s still dominate the underlined $\vee$'s in

$(((A\wedge\neg B)\underline{\wedge}(B\underline{\vee}C))\vee((\neg B\wedge\neg C)\underline{\wedge}(B\underline{\vee}C)))\vee(((A\wedge\neg B)\underline{\wedge}(\neg A\underline{\vee}B))\vee((\neg B\wedge\neg C)\underline{\wedge}(\neg A\underline{\vee}B)))$,

so we apply the distributivity 4 times and get

$F_4 = ((((A\wedge\neg B)\wedge B)\vee((A\wedge\neg B)\wedge C))\vee(((\neg B\wedge\neg C)\wedge B)\vee((\neg B\wedge\neg C)\wedge C)))\vee((((A\wedge\neg B)\wedge\neg A)\vee((A\wedge\neg B)\wedge B))\vee(((\neg B\wedge\neg C)\wedge\neg A)\vee((\neg B\wedge\neg C)\wedge B)))$.

Step 5: Simplify.

The disjunction F_4 has the disjuncts $D_1 = (A \wedge \neg B) \wedge B, D_2 = (A \wedge \neg B) \wedge C, D_3 = (\neg B \wedge \neg C) \wedge B, D_4 = (\neg B \wedge \neg C) \wedge C, D_5 = (A \wedge \neg B) \wedge \neg A, D_6 = (A \wedge \neg B) \wedge B, D_7 = (\neg B \wedge \neg C) \wedge \neg A, D_8 = (\neg B \wedge \neg C) \wedge B$

The conjunctions D_1, D_3, D_4, D_5, D_6, D_8 contain both an atom and its complement, hence they are unsatisfiable. The remaining conjunctions have the sets of literals $\{A, \neg B, C\}$ and $\{\neg A, \neg B, \neg C\}$.

We get $F_5 = D_2 \vee D_7 = ((A \wedge \neg B) \wedge C) \vee ((\neg A \wedge \neg B)\neg C)$
Step 6. Find the prefix form F_6 of F_5.

$F_6 = F_5$ because F_5 is a prefix from.

The proof of the correctness of Algorithm 1.7.20 is similar to that of Algorithm 1.7.12.

We can also find a disjunctive normal form of the formula F from the truth tables. For this we need the notion of *min-term*.

Definition 1.7.22 (min-term, the r min-term of $P_1, \ldots, P_n$) *1. A min-term of $P_1, \ldots, P_n$ is a conjunction $\bigwedge_{i=1}^{n} L_i$, where L_i is either P_i or $\neg P_i$.*

2. Let $P_1, \ldots, P_n$ be n atoms and r an integer that satisfies the inequalities $0 \leq r < 2^n - 1$. Let $r_1 \ldots r_n$ be the binary digits of r. The min-term of r has the support $L_1 \ldots L_n$, with the literals L_i defined below.

$$L_i = \begin{cases} P_i & \text{if } r_i = 1 \\ \neg P_i & \text{if } r_i = 0 \end{cases}$$

Examples 1.7.23 1. The min-terms of P_1, P_2, P_3 are $(P_1 \wedge P_2) \wedge P_3$, $(P_1 \wedge P_2) \wedge \neg P_3$, $(P_1 \wedge \neg P_2) \wedge P_3$, $(P_1 \wedge \neg P_2) \wedge \neg P_3$, $(\neg P_1 \wedge P_2) \wedge P_3$, $\neg(P_1 \wedge P_2) \wedge \neg P_3$, $(\neg P_1 \wedge \neg P_2) \wedge P_3$, $(\neg P_1 \wedge \neg P_2) \wedge \neg P_3$.

2. Let P_1, P_2, P_3, P_4 be a sequence of 4 atoms. Let us find the P_1, P_2, P_3, P_4 min-term of 13. First we write 13 in binary as 1101. So, $r_1 = 1, r_2 = 1, r_3 =$

$0, r_4 = 1$. Now we apply Definition 1.7.22. We get $L_1 = P_1$, $L_2 = P_2$, $L_3 = \neg P_3$, $L_4 = P_4$. The min-term of 13 is $((P_1 \wedge P_2) \wedge \neg P_3) \wedge P_4$.

Now we present the second algorithm for finding a DNF of a formula.

Algorithm 1.7.24 Step 1. Identify the atoms of F. Let's assume that the atoms are P_1, ..., P_n.
Step 2. Form a table with $n+1$ columns labeled P_1, ..., P_n, F. Make 2^n rows and label the rows with the numbers $0 - (2^n - 1)$.
Step 3. Fill in the first n columns of each row (under the headers P_1, ..., P_n) with the digits of the row number written in binary. The first binary digit is put under the header P_1, the second under P_2, and so on until the n-th digit is put under P_n.
Step 4. Evaluate F for each row.
Step 5. We form the min-terms of all rows that have 1 in the F column. These are the min-terms of F.
Step 6. Form a disjunction with the min-terms of F listed in the increasing order of the row. If there are no min-terms, the DNF is $\Box$, else put it in the prefix form.

Example 1.7.25 Let us use Algorithm 1.7.24 to find a disjunctive normal form for $F = (P_1 \wedge P_2) \longleftrightarrow (\neg P_3 \vee P_2)$.

Steps 1,2, 3. We form the table below.

row	P_1	P_2	P_3	$(P_1 \wedge P_2) \longleftrightarrow (\neg P_3 \vee P_2)$
0	0	0	0	
1	0	0	1	
2	0	1	0	
3	0	1	1	
4	1	0	0	
5	1	0	1	
6	1	1	0	
7	1	1	1	

Step 4. Now we evaluate F for each row.

row	P_1	P_2	P_3	$(P_1 \wedge P_2) \longleftrightarrow (\neg P_3 \vee P_2)$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

Step 5. We form the min-terms for the rows where F is 1.

For row 1 we get $\neg P_1 \wedge \neg P_2 \wedge P_3$, for row 5 we get $P_1 \wedge \neg P_2 \wedge P_3$, for row 6 we get $P_1 \wedge P_2 \wedge \neg P_3$, and for row 7 we get $P_1 \wedge P_2 \wedge P_3$.

Step 6. We form the disjunction of min-terms and we use the prefix form.

We get $((((\neg P_1 \wedge \neg P_2) \wedge P_3) \vee ((P_1 \wedge \neg P_2) \wedge P_3)) \vee ((P_1 \wedge P_2) \wedge \neg P_3)) \vee ((P_1 \wedge P_2) \wedge P_3)$.

We will prove that Algorithm 1.7.24 is correct, by showing that it terminates, produces a DNF, and the DNF is equivalent to F. For this we need Lemma 1.7.26. It states that there is one-to-one correspondence between the entries in the truth tables and the set of min-terms.

Lemma 1.7.26 *Let $\mathcal{A}$ be a truth assignment and m be a min-term of $P_1 \dots P_n$. Then, $\mathcal{A}$ is a model of m iff m has the row number $r = \mathcal{A}[P_1] \dots \mathcal{A}[P_n]$.*

Proof: Let $q = q_1 \dots q_n$ be the row number of m, written in binary. Then, Definition 1.7.22 gives us the literals L_i of m.

$$L_i = \begin{cases} P_i & \text{if } q_i = 1 \\ \neg P_i & \text{if } q_i = 0 \end{cases}$$

We use this definition to compute $\mathcal{A}[L_i]$.

$$\mathcal{A}[L_i] = \begin{cases} \mathcal{A}[P_i] & \text{if } q_i = 1 \\ \mathcal{A}[\neg P_i] & \text{if } q_i = 0 \end{cases}$$

From this definition we get

$\mathcal{A}[L_i] = 1$
iff $\mathcal{A}[P_i] = 1$ and $q_i = 1$, or $\mathcal{A}[P_i] = 0$ and $q_i = 0$
iff $\mathcal{A}[P_i] = q_i$.
Now $\mathcal{A}[m] = 1$
iff for all $1 \leq i \leq n$, $\mathcal{A}[P_i] = q_i$ by the preceding equality
iff $r = q$ because $r = \mathcal{A}[P_1] \dots \mathcal{A}[P_n]$ and $q = q_1 \dots q_n$. **Q.E.D.**

Theorem 1.7.27 *Algorithm 1.7.24 is correct.*

Proof: All steps of Algorithm 1.7.24 terminate, and the output is a DNF. We need to show that the input is equivalent to the output. Let G be the output of the algorithm and $\mathcal{A}$ a truth assignment. Let r be the row number $\mathcal{A}[P_1] \dots \mathcal{A}[P_n]$ of $\mathcal{A}$. We will show that $\mathcal{A}[F] = 1$ iff $\mathcal{A}[G] = 1$.

$\Longrightarrow$: Assume that $\mathcal{A}[F] = 1$. Then F has value 1 in row r. By the construction of the DNF, the min-term of r is a disjunct of G. By Lemma 1.7.26, $\mathcal{A}$ satisfies the r min-term of $P_1 \dots P_n$. So, $\mathcal{A}[G] = 1$ because it satisfies a disjunct of G.

$\Longleftarrow$: Assume that $\mathcal{A}[G] = 1$. Since G is a disjunction, $\mathcal{A}$ satisfies a disjunct of G. That disjunct is the min-term of some row q, $0 \leq 2^n - 1$. We know, from the construction of the DNF, that F is 1 for row q. We also know, from Lemma 1.7.26, that $\mathcal{A}$ satisfies the min-term iff $r = q$. So, F has value 1 for row r, i.e. $\mathcal{A}[F] = 1$. **Q.E.D.**

Observation 1.7.28 The DNF's computed by Algorithm 1.7.24 have property (1).

(1) Let F and G be two formulas, and $P_1, \ldots, P_n$ be a sequence that includes all atoms of F and of G. Let F_1 and G_1 be the DNF's for F, respectively G, computed by the algorithm. Then $F \equiv G$ if and only if $F_1 = G_1$.

This means that F and G are equivalent if and only if they have the *same* normal form. If F_1 and G_1 are computed using different sequences of atoms, the DNF's are not the same, as proved by this example.

We know, from Table 1.47, that $F = A \equiv G = A \wedge (A \vee B)$. Let us compute the DNF of F using only the atom A. We get $F_1 = A$. We need to compute the DNF of G using both A and B. We get $G_1 = (A \wedge \neg B) \vee (A \wedge B)$. We see that $F_1 \neq G_1$. However, if when we compute the DNF of F with the sequence A, B, we obtain $F_1 = (A \wedge \neg B) \vee (A \wedge B)$. So, the normal forms are identical.

Observation 1.7.29 Algorithm 1.7.24 provides us with a method for testing equivalence.
Step 1. Build a sequence of all atoms that occur in at least one of the formulas.
Step 2. Use Algorithm 1.7.24 to compute the DNFs F_1 and G_1 of F respectively G.
Step 3. Check if $F_1 = G_1$. If they are, then $F \equiv G$, otherwise $F \not\equiv G$.

Next, we will show that we can easily transform DNF's into CNF's and vice versa. So, if we know how to compute one form, we can find the other. The next lemma is a generalization of Demorgan's laws from Table 1.47.

Lemma 1.7.30 (Generalized DeMorgan's Laws) *1.* $\neg \bigvee_{i=1}^{n} F_i \equiv \bigwedge_{i=1}^{n} \neg F_i$.
2. $\neg \bigwedge_{i=1}^{n} F_i \equiv \bigvee_{i=1}^{n} \neg F_i$.

The proof is by induction on n and is left as exercise.

Now we give the theorem that allows us to go from DNF's to CNF's and back.

Theorem 1.7.31 *1. If* $\neg F \equiv \bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} L_{ij}$ *then* $F \equiv \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n_i} \overline{L}_{ij}$.
2. If $\neg F \equiv \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n_i} L_{ij}$ *then* $F \equiv \bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} \overline{L}_{ij}$.
Here $\overline{L}$ *is the complement of* L, *and* L_{ij}*'s are literals, for* $1 \leq i \leq n$, *and* $1 \leq j \leq n_i$.

Proof: We'll prove 1 and leave 2 as exercise.

$F \equiv \neg\neg F$ by the double negation introduction
$\equiv \neg \bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} L_{ij}$ by hypothesis
$\equiv \bigwedge_{i=1}^{n} \neg \bigwedge_{j=1}^{n_i} L_{ij}$ by part 1 of Lemma 1.7.30
$\equiv \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n_i} \neg L_{ij}$ by part 2 of Lemma 1.7.30
$\equiv \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n_i} \overline{L}_{ij}$ because $\neg L \equiv \overline{L}$. **Q.E.D.**

Example 1.7.32 Let us use Theorem 1.7.31 to find a CNF from the truth table below.

row	A	B	C	F
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

We first compute the DNF of $\neg F$. We get the table of $\neg F$ from the table of F, by replacing in the last column, 0's by 1's and 1's by 0's.

row	A	B	C	$\neg F$
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

We apply Algorithm 1.7.24. We form the min-terms for the lines 2, 3, 4, and 6 and we get $(\neg A \wedge B) \wedge \neg C$, $(\neg A \wedge B) \wedge C$, $(A \wedge \neg B) \wedge \neg C$, and $(A \wedge B) \wedge \neg C$.

The DNF of $\neg F$ is $((((\neg A \wedge B) \wedge \neg C) \vee ((\neg A \wedge B) \wedge C)) \vee ((A \wedge \neg B) \wedge \neg C)) \vee ((A \wedge B) \wedge \neg C)$.

Theorem 1.7.31 tells us that we get a CNF of F by replacing, at the same time, $\vee$'s by $\wedge$'s, $\wedge$'s by $\vee$'s and all literals by their complements. We remember that $\overline{P} = \neg P$ and $\overline{\neg P} = P$. So,

$F \equiv ((((A \vee \neg B) \vee C) \wedge ((A \vee \neg B) \vee \neg C)) \wedge ((\neg A \vee B) \vee C)) \wedge ((\neg A \vee \neg B) \vee C)$.

Now we will prove the termination of Steps 3 and 4 of Algorithms 1.7.12 and 1.7.20. The termination of Step 6 of these algorithms in done in the same way. The method requires a little explaining, so the uninterested reader can skip the rest of the section and go on to the resolution.

The complexity functions are used to prove the termination of a set of simplifications. A simplification is a pair of formulas $F \Longrightarrow G$. If a formula H contains an instance[9] of F then we can replace it by the corresponding instance of G and get the formula I. We say that H was reduced or simplified to I, and write $H \Longrightarrow I$.

The idea of the complexity functions is to attach to every formula a whole number in such a way that every time we do a simplication $F \Longrightarrow G$ the G number is less than the F-number.

Definition 1.7.33 (complexity function) *We say that $f : FORM \longrightarrow N$ is a complexity function if for all formulas F,G, H,*

[9] F and G are meta-formulas, so they may contain meta-variables. An instance is obtained by replacing the meta-variables by formulas.

1. $f[F] > f[G]$ *implies* $f[\neg F] > f[\neg G]$,
2a. $f[F] > f[G]$ *implies* $f[F \vee H] > f[G \vee H]$,
2b. $f[F] > f[G]$ *implies* $f[H \vee F] > f[H \vee G]$,
3a. $f[F] > f[G]$ *implies* $f[F \wedge H] > f[G \wedge H]$,
3b. $f[F] > f[G]$ *implies* $f[H \wedge F] > f[H \wedge G]$,
4a. $f[F] > f(G]$ *implies* $f[F \longrightarrow H] > f[G \longrightarrow H]$,
4b. $f[F] > f[G]$ *implies* $f[H \longrightarrow F] > f[H \longrightarrow G]$,
5a. $f[F] > f[G]$ *implies* $f[F \longleftrightarrow H] > f[G \longleftrightarrow H]$,
5b. $f[F] > f[G]$ *implies* $f[H \longleftrightarrow F] > f[H \longleftrightarrow G]$,

Examples 1.7.34 *1.* The function g below is a complexity function.
1. $g[P_i] = 1$
2. $g[\neg F] = 2g[F]$
3. $g[FCG] = g[F] + g[G] + 1$ when C is one of the binary connectives.

We can easily verify the 11 conditions above.
If $g[F] > g[G]$ then $2g[F] > 2g[G]$, i.e. $g[\neg F] > g[\neg G]$.
If $g[F] > g[G]$ then $g[F]+g[H]+1 > g[G]+g[H]+1$, so, $g[FCH] > g[GCH]$.
If $g[F] > g[G]$ then $g[H]+g[F]+1 > g[H]+g[F]+1$, so, $g[HCF] > g[HCG]$.
2. The function h defined beneath is also a complexity function.
1. $h[P_i] = 2$
2. $h[\neg F] = h[F] + 1$
3. $h[F \vee G] = h[F] * h[G]$
4. $h[FCG] = h[F] + h[G] + 1$ when C is one of the connectives $\wedge$, $\longrightarrow$, or $\longleftrightarrow$.

We leave the verification of the 11 conditions of the definition of the complexity function to the reader.

Proposition 1.7.35 *Let f be a complexity function and F,G,H be formulas such that G occurs in F and $f(G) > f(H)$. Let I is the formula obtained from F by replacing an occurrence of G by H. Then $f(F) > f(I)$.*

Proof: The proof is by structural induction on F. It is left as exercise.

Definition 1.7.36 *Let S be a set of simplifications $L \Longrightarrow R$, finite or infinite. We say that a complexity function satisfies S if for all simplifications $L \Longrightarrow R$, $f[L] > f[R]$.*

Lemma 1.7.37 *g satisfies the set of simplifications $S_1 = \{\neg\neg F \Longrightarrow F, \neg(F \vee G) \Longrightarrow (\neg F \wedge \neg G), \neg(F \wedge G) \Longrightarrow (\neg F \vee \neg G)\}$, and h satisfies the set $S_2 = \{F \vee (G \wedge H) \Longrightarrow ((F \vee G) \wedge (F \vee H)), (F \vee G) \wedge H \Longrightarrow ((F \vee H) \wedge (G \vee H))\}$.*

Proof: We will prove that
(1) $g[\neg\neg F] > g[F]$
and
(2) $g[\neg(FCG] > g[\neg FC_1\neg G]$
for any binary connectives C and C_1. The first inequality takes care of $\neg\neg F \Longrightarrow F$ and the second one handles the last two simplifications.

First we notice that for all formulas F, $g(F) \geq 1$ because the atoms receive 1 and g grows as the formula becomes more complex.

$g[\neg\neg F] = 2g[\neg F] = 4g[F] > g[F]$

$g[\neg(FCH)] = 2g[FCH] = 2[g[F] + g[G] + 1] = 2g[F] + 2g[G] + 2$

$> 2g[F] + 2g[G] + 1 = g[\neg F] + g[\neg G] + 1 = g[\neg F C_1 \neg G]$

So, g satisfies the set of simplifications $S_1 = \{\neg\neg F \Longrightarrow F, \neg(F \vee G) \Longrightarrow (\neg F \wedge \neg G), \neg(F \wedge G) \Longrightarrow (\neg F \vee \neg G)\}$.

Now we will show that h satisfies S_2. We will show that

$f[F \vee (G \wedge H)] > h[(F \vee G) \wedge (F \vee H)]$

and leave the other one to the reader. First we notice that for all formulas F, $h[F] \geq 2$. Then,

$h[F \vee (G \wedge H)] = h[F] * h[G \wedge H] = h[F] * [h[G] + h[H] + 1] = h[F] * h[G] + h[F] * h[H] + h[F]$

$\geq h[F] * h[G] + h[F] * h[H] + 1$ because $h[F] \geq 2$

$> h[F \vee G] + h[F \vee H] + 1 = h[(F \vee G) \wedge (F \vee H)]$.**Q.E.D.**

Lemma 1.7.38 *If a complexity function satisfies a set of simplifications S, then any sequence of simplifications $F_0 \Longrightarrow F_1 \Longrightarrow \ldots F_n \Longrightarrow \ldots$ terminates.*

Proof: For every simplification step $F_i \Longrightarrow F_{i+1}$, there is simplification $L \Longrightarrow R$ in S such that L occurs in F_i and F_{i+1} is obtained from F_i by replacing L by R. Since f satisfies the simplification, $f[L] > f[R]$. By Proposition 1.7.35 $f[F_i] > f[F_{i+1}]$. Then $f[F_0] > f[F_1] > \ldots f[F_n] > \ldots$ is a descending sequence of whole numbers. Since N does not have infinite descending sequences, the sequence is finite. Then, so is the sequence of simplifications. **Q.E.D.**

Corollary 1.7.39 *Steps 3 and 4 of the CNF algorithms terminate.*

Proof: This is a consequence of the 2 preceding lemmas.

Exercises

Exercise 1.7.1 *Identify n, the n_i's and the L_{ij}'s for the conjunctive normal form $((((P_3 \vee P_4) \wedge \neg P_4) \wedge (((P_1 \vee P_2) \vee \neg P_3) \vee P_4)) \wedge (\neg P_1 \vee \neg P_2))$.*

Exercise 1.7.2 *Which ones of these formulas are conjunctive normal forms?*

$(P_1 \vee \neg P_3)$

$((P_1 \wedge \neg P_3) \wedge P_4)$

$(P_1 \wedge (\neg P_2 \wedge P_3))$

$\neg(P_3 \wedge P_4)$

Exercise 1.7.3 *Let's look at the dominance relation on the set of operator occurrences of a formula. Check if that relation is reflexive, irreflexive, symmetric, antisymmetric, transitive.*

Exercise 1.7.4 *Find all dominance relations among the connective occurrences of the formula $((P_1 \vee (P_3 \vee \neg(P_4 \wedge \neg P_2))) \longrightarrow \neg(P_2 \wedge P_2))$.*

Exercise 1.7.5 *Which ones of these clauses is a prefix form?*

1. $(P_1 \vee P_2) \vee P_3$
2. $P_1 \vee (\neg P_3 \vee P_4)$

Exercise 1.7.6 *Let $G = \bigvee_{k=1}^{p} M_k$ and $H = \bigvee_{l=1}^{q} N_l$ be two clauses. Then $G \wedge H$ is a conjunctive normal form of the form $G \wedge H = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n_i} L_{ij}$. Identify n, the n_i's and the L_{ij}'s based on p, q, the M_k's and the N_l's.*

Exercise 1.7.7 *Use the equivalences from Figure 1.47 and The Substitution Theorem to show that $(G \wedge H) \vee I \equiv (G \vee I) \wedge (H \vee I)$.*

Exercise 1.7.8 *Compute a CNF form for each of the following formulas:*

1. $\neg(\neg A \vee B) \longleftrightarrow (C \wedge \neg D)$
2. $\neg((\neg(A \vee B) \vee C) \longleftrightarrow (\neg B \wedge D))$
3. $\neg(A \longrightarrow (\neg B \vee C)) \longleftrightarrow (C \vee \neg D)$
4. $((A \longrightarrow B) \longleftrightarrow (B \wedge C)$

Exercise 1.7.9 *Apply the construction from Observation 1.7.16 to get infinitely many CNF's for $F = ((P_1 \vee P_6) \wedge \neg P_6)$ and $G = ((P_1 \vee \neg P_7) \vee P_9)$.*

Exercise 1.7.10 *What happens if the CNF G from Observation 1.7.16 has no conjuncts? Is the statement still true?*

Exercise 1.7.11 *Find a complexity function that satisfies the simplification $\{F \wedge (G \wedge H) \Longrightarrow (F \wedge G) \wedge H\}$.*

Exercise 1.7.12 *Prove that the function m defined below is a bijection from the set of rows $\{0, \ldots, 2^n - 1\}$ to the set of min-terms formed with the atoms $P_1, \ldots, P_n$.*

$m[row] = s_1 P_1 \wedge s_2 P_2 \wedge \ldots \wedge s_n P_n$ where
$s_i = \lambda$ (the empty string) if $r_i = 0$, and
$s_i = \neg$ if $r_i = 1$.
Here, r_i is the i-th digit of row written in binary.

Exercise 1.7.13 *Prove Observation 1.7.28.*

Exercise 1.7.14 *Prove Lemma 1.7.30.*

Exercise 1.7.15 *Prove part 2 of Theorem 1.7.31.*

Exercise 1.7.16 *Compute disjunctive normal forms for the two formulas below, using the two methods given in this section.*

1. $(A \longrightarrow \neg B) \wedge ((C \vee B) \longleftrightarrow \neg A)$
2. $(A \vee \neg B) \longleftrightarrow \neg(B \wedge C)$

Exercise 1.7.17 *Give 3 different DNF's for $A \wedge (B \vee C)$.*

Exercise 1.7.18 *Use Theorem 1.7.31 to find a CNF for F when $\neg F$ takes the values listed below.*

1. $\neg F \equiv ((\neg A \wedge \neg B) \wedge C) \vee ((A \wedge B) \wedge \neg C)$

2. $\neg F \equiv ((\neg A \wedge \neg B) \vee ((\neg A \wedge B) \wedge \neg C)) \vee B$

Exercise 1.7.19 *Use Theorem 1.7.31 to find a DNF for F when $\neg F$ takes the values listed below.*

1. $\neg F \equiv ((A \vee \neg B) \vee C) \wedge ((\neg A \vee B) \vee \neg C)$

2. $\neg F \equiv (((A \vee B) \wedge \neg C) \wedge ((\neg A \vee \neg B) \vee D)) \wedge (\neg A \vee D)$

Exercise 1.7.20 *Prove the Generalized Distributive Equivalences.*

(1) $F \vee \bigwedge_{i=1}^{n} G_i \equiv \bigwedge_{i=1}^{n} (F \vee G_i)$

(2) $F \wedge \bigvee_{i=1}^{n} G_i \equiv \bigvee_{i=1}^{n} (F \wedge G_i)$

Exercise 1.7.21 *Prove, by structural induction, that every formula F has a DNF and a CNF. Use ony the Generalized DeMorgan's Laws, the Generalized Distributivity, the eliminations of $\longrightarrow$'s and $\longleftrightarrow$'s, and the properties of $\vee$'s and $\wedge$'s.*

Exercise 1.7.22 *Two sets of formulas are equivalent if they have the same set of models. Show that every set of formulas (finite or infinite) is equivalent to a set of clauses.*

1.8 Resolution

From the preceding section we know that every formula has a conjunctive normal form, so it is equivalent to a set of clauses. In this section we present the *resolution*, a method for proving that a set of clauses is unsatisfiable. We begin by defining resolvents, first for two clauses, and then for a set of clauses. Then we extend the set of resolvents of S, $Res[S]$, to the set $Res^*[S]$ that contains S, the resolvents of S, the resolvents of the resolvents of S, and so on. The high point of the section is The Resolution Theorem, which states that S is unsatisfiable iff $\Box \in Res^*[S]$. Along the way we introduce the notions of derivation sequence and derivation tree, that help us to understand the method and to implement it.

We know, from Theorem 1.6.18, that a non-tautology clause is identified with its set of literals. So, from now on, we will represent clauses as sets of literals.

Definition 1.8.1 (resolvent of two clauses, unifiable clauses) *1. Let C_1 and C_2 be two clauses and P an atom such that $P \in C_1$ and $\neg P \in C_2$. Then the clause $R = (C_1 - P) \cup (C_2 - \{\neg P\})$ is called the resolvent of C_1 and C_2 on P. The clauses C_1 and C_2 are called the parents of R, and P is the unifying atom.*

2. We say that R is a resolvent of C_1 and C_2 if for some atom P, R is the resolvent of C_1 and C_2 on P, or R is the resolvent of C_2 and C_1 on P.

3. We say that the clauses C_1 and C_2 are unifiable if there is some atom P such that P belongs to one of the two clauses and $\neg P$ to the other.

So, we compute the resolvent of C_1 and C_2 on A by deleting the atom A from C_1, the literal $\neg A$ from C_2 and then joining the reduced sets.

Examples 1.8.2 1. The resolvent of $C_1 = \{A, B, C\}$ and $C_2 = \{B, \neg D, \neg C\}$ on C is $R = \{A, B, \neg D\}$.

2. The resolvent of $C_1 = \{A\}$ and $C_2 = \{\neg A\}$ on A is $R = \{\}$, the empty clause. We will write $\square$ instead of $\{\}$. We recall that the empty clause is unsatisfiable.

3. The clauses $C_1 = \{A, B\}$ and $C_2 = \{\neg A, \neg B\}$ have two resolvents: $R_1 = \{B, \neg B\}$, and $R_2 = \{A, \neg A\}$. We get R_1 when we do the resolution on A and R_2 when we do the resolution on B. $R = \{\}$ is **not** a resolvent of C_1 and C_2 since we do the resolution on either A or B, but not on both.

Observations 1.8.3 1. When we say that *R is the resolvent of C_1 and C_2 on P*, we mean that the unifying atom is in the first clause and its negation is in the second. So, B is the resolvent of $C_1 = \{A, B\}$ and $C_2 = \{\neg A\}$ on A, because the unifying atom A is in the first clause and its negation in the second. We cannot say that B is the resolvent of C_2 and C_1 on A because A does not belong to C_2.

2. In the relation *R is a resolvent of C_1 and C_2* we can interchange the places of C_1 and C_2 because the unifying atom P can be in C_1 or in C_2. So, $R = \{A, B\}$ is both a resolvent of $C_1 = \{A, B, C\}$ and $C_2 = \{A, B, \neg C\}$ and of C_2 and C_1.

At this point we can ask ourselves two questions: *Is it possible for a clause to be unified with itself?*, and, *What happens when two clauses have more than one resolvent?*.

If C can be unified with itself, then it must contain an atom, say A, and its negation $\neg A$. So, C is a tautology. Let us compute the resolvent of C and C on A.

$R = (C - \{A\}) \cup (C - \{\neg A\})$ from the definition of the resolvent
$= ((C - \{A, \neg A\}) \cup \{\neg A\}) \cup ((C - \{A, \neg A\}) \cup \{A\})$ $A, \neg A \in C$
$= (C - \{A, \neg A\}) \cup \{\neg A\} \cup \{A\}$
$= C$ since $A, \neg A \in C$

So, whenever we unify a tautology with itself we do not get new clauses. This result is useful when we will compute the resolvents of a set of clauses.

Sometimes two clauses have more than one resolvent. For example, the resolvents of $C_1 = \{A, \neg B\}$ and $C_2 = \{\neg A, B\}$ are $R_1 = \{B, \neg B\}$ and $R_2 = \{A, \neg A\}$. We will show that whenever C_1 and C_2 have more than one resolvent, then the resolvents are tautologies.

Proposition 1.8.4 *If two clauses have more than one resolvent, the resolvents are tautologies.*

Proof: Let C_1 and C_2 be two clauses that have two distinct resolvents, R_1 and R_2. Let L and M be the literals of C_1 involved in the computation of R_1, rspectively R_2. Then $\overline{L}$ and $\overline{M}$ are in C_2.

Since $R_1 \neq R_2$, $L \neq M$. When we compute R_1 we delete L from C_1 but we do not delete M because $M \neq L$. We also delete $\overline{L}$ from C_2 but not $\overline{M}$, since $\overline{L} \neq \overline{M}$. Since R_1 contains all literals of C_1 and C_2 that are not deleted, both M and $\overline{M}$ are in R_1. So, R_1 is a tautology.

A similar argument shows that L, $\overline{L}$ are in R_2. **Q.E.D.**

Now we will show that the resolvents are consequences of the parents.

Proposition 1.8.5 (resolvents are consequences of the parents) *Let C_1 and C_2 be two clauses and R a resolvent of C_1 and C_2. Then $C_1 \wedge C_2 \equiv (C_1 \wedge C_2) \wedge R$.*

Proof: Let $\mathcal{A}$ be a truth assignment. We have to show that $\mathcal{A}[C_1 \wedge C_2] = 1$ if and only if $\mathcal{A}[(C_1 \wedge C_2) \wedge R] = 1$.
$\Longleftarrow$: Assume that $\mathcal{A}[(C_1 \wedge C_2) \wedge R] = 1$. Then, from the interpretation of $\wedge$, $\mathcal{A}[C_1 \wedge C_2] = 1$.
$\Longrightarrow$: Assume that $\mathcal{A}[C_1 \wedge C_2] = 1$. From the interpretation of $\wedge$, both $\mathcal{A}[C_1]$ and $\mathcal{A}[C_2] = 1$. We need to show that $\mathcal{A}[R] = 1$.

Since R is a resolvent of C_1 and C_2, then there is an atom P such that R is the resolvent of C_1 and C_2 on P, or the resolvent of C_2 and C_1 on P. In both cases, there is a literal $Q \in C_1$ such that its complement $\bar{Q}$ is in C_2.

From Definition 1.8.1, $R = (C_1 - \{Q\}) \cup (C_2 - \{\bar{Q}\})$. We have two cases, depending on whether $\mathcal{A}[Q] = 0$ or $\mathcal{A}[Q] = 1$.

Case 1: $\mathcal{A}[Q] = 0$.

Since $\mathcal{A}[C_1] = 1$, $\mathcal{A}[C_1 - \{Q\}] = 1$. But $(C_1 - \{Q\}) \subseteq R$, so $\mathcal{A}[R] = 1$.

Case 2: $\mathcal{A}[Q] = 1$.

Then $\mathcal{A}[\bar{Q}] = 0$. Since $\mathcal{A}[C_2] = 1$, $\mathcal{A}[C_2 - \{\bar{Q}\}] = 1$. But $(C_2 - \{\bar{Q}\}) \subseteq R$, so $\mathcal{A}[R] = 1$.

In either case $\mathcal{A}[R] = 1$. Since $\mathcal{A}[C_1]$ and $\mathcal{A}[C_2]$ are already 1, $\mathcal{A}[(C_1 \wedge C_2) \wedge R] = 1$. **Q.E.D.**

We can extend the notion of resolvent of two clauses to the **set of resolvents**.

Definition 1.8.6 ($Res[S]$) *Let S be a set of clauses. We define $Res[S]$, the set of resolvents of S, as*

$Res[S] = S \cup \{R | R$ is a resolvent of two clauses C_1 and C_2 from $S\}$.

Example 1.8.7 Let us find $Res[S]$ when $S = \{\{A, B\}, \{A, \neg B\}, \{\neg A, B\}\{\neg A, \neg B\}\}$. First of all we list the clauses in S.

1. $\{A, B\}$
2. $\{A, \neg B\}$
3. $\{\neg A, B\}$
4. $\{\neg A, \neg B\}$.

Next we take all possible pairs of clauses and find all their resolvents. So, we compute the resolvents of the pairs of clauses 1 and 2, 1 and 3, 1 and 4, 2 and

3, 2 and 4, 3 and 4. We don't worry about unifying a clause with itself since the resolvents, if they exist, are identical to the parent clause.

We get the 6 clauses below.

5. $\{A\}$ from 1 and 2 on B.
6. $\{B\}$ from 1 and 3 on A.
7. $\{B, \neg B\}$ from 1 and 4 on A.
8. $\{A, \neg A\}$ from 1 and 4 on B.
9. $\{\neg B\}$ from 2 and 4 on A.
10. $\{\neg A\}$ from 3 and 4 on B.

Clauses 7 and 8 are also the resolvents of clauses 2 and 3. So, $Res[S]$ is the set of clauses 1 to 10.

We notice that both 7 and 8 are tautologies.

Definition 1.8.8 (semantically equivalent sets) *We say that two sets of formulas S and T are semantically equivalent, written $S \equiv T$, if for every truth assignment $\mathcal{A}$, $\models_{\mathcal{A}} S$ if and only if $\models_{\mathcal{A}} T$.*

Proposition 1.8.9 *Let S be a set of clauses. Then $S \equiv Res[S]$.*

Proof: We need to show that for all truth assignments $\mathcal{A}$, $\models_{\mathcal{A}} S$ if and only if $\models_{\mathcal{A}} Res[S]$.
$\Longleftarrow$: This is easy. Since S is a subset of $Res[S]$ and $\mathcal{A}$ models every clause in $Res[S]$, then $\mathcal{A}$ is a model for every clause in S.
$\Longrightarrow$: Assume that $\models_{\mathcal{A}} S$. We will show that $\mathcal{A}$ models every clause in $Res[S]$. Let F be a clause in $Res[S]$. Then either F is in S or F is a resolvent of two clauses C_1 and C_2 of S. If F is a clause in S, then $\models_{\mathcal{A}} F$, because $\models_{\mathcal{A}} S$. If F is a resolvent of C_1 and C_2, then $\{F, C_1, C_2\} \equiv \{C_1, C_2\}$ by Proposition 1.8.5. Since $\mathcal{A}$ is a model of $\{C_1, C_2\}$,the semantic equivalence tells us that $\models_{\mathcal{A}} F$. **Q.E.D.**

We can extend the notion of resolvent to include resolvents of resolvents, resolvents of resolvents of resolvents, and so on.

Definition 1.8.10 ($Res^n[S]$,$Res^*[S]$) *Let S be a set of clauses.*

1. We define $Res^n[S]$, the set of all resolvents of height $\leq n$ of S, as

1.1. $Res^0[S] = S$

1.2. $Res^{n+1}[S] = Res[Res^n[S]]$.

2. We define the set of all resolvents of finite height of S as $Res^*[S] = \bigcup_{i=0}^{\infty} Res^i[S]$.

Let us find $Res^*[S]$ where S contains the 3 clauses below.

1. $\{A, B, C\}$
2. $\{\neg B, C\}$
3. $\{\neg C\}$

We first find $Res^1[S] = Res[S]$. We do this by computing all resolvents of all pairs of clauses in S. We get the following new clauses.

4. $\{A, C\}$ from 1 and 2 on B
5. $\{A, B\}$ from 1 and 3 on C
6. $\{\neg B\}$ from 2 and 3 on C

So, $Res^1[S]$ is the set of clauses 1-6.

Now let us find $Res^2[S]$, the set of resolvents of S of height ≤ 2.

We must compute $Res[Res^1[S]]$, i.e. add to $Res^1[S]$ all resolvents of all pairs of clauses numbered 1-6 that are not already in. We do not want to recompute the resolvents of $Res^1[S]$, so, we will work only with the pairs C_1, C_2, that have C_2 in $Res^1[S] - S$. Thus, we find all resolvents of the clauses 1 and 4, 1 and 5, 1 and 6, 2 and 4, 2 and 5, 2 and 6,... , 4 and 5, 4 and 6, 5 and 6. We get one new clause.

7. $\{A\}$ from 4 and 3 on C

Next we compute $Res^3[S]$ but we do not get any new clauses. So, $Res^3[S] = Res^2[S]$. We can apply the next proposition, and conclude that $Res^*[S] = Res^2[S]$.

Proposition 1.8.11 *If $Res^{n+1}[S] = Res^n[S]$ then $Res^*[S] = Res^n[S]$.*

Proof: We will show, by induction on m, that for all $m > n$, $Res^m[S] \subseteq Res^n[S]$.

Basis: $m = n + 1$. Since $Res^{n+1}[S] = Res^n[S]$, $Res^{n+1}[S] \subseteq Res^n[S]$

Inductive Step: Assume that $Res^m[S] \subseteq Res^n[S]$ and $m \geq n + 1$. Let F be a clause in $Res^{m+1}[S]$. Since $Res^{m+1}[S] = Res[Res^m[S]]$, F is a clause in $Res^m[S]$ or a resolvent of two clauses in $Res^m[S]$. If S is a clause in $Res^m[S]$ then $F \in Res^n[S]$ from the induction hypothesis.

Assume that F is a resolvent of two clauses in $Res^m[S]$. By induction hypothesis, the parents of F are in $Res^n[S]$. Then their resolvents are in $Res^{n+1}[S]$. So, $F \in Res^{n+1}[S]$. Since $Res^{n+1}[S] \subseteq Res^n[S]$, $F \in Res^n[S]$.

In either case $F \in Res^n[S]$, so $Res^{m+1}[S] \subseteq Res^n[S]$.

From Definition 1.8.10 we know that whenever $n \leq m$, $Res^n[S] \subseteq Res^m[S]$. Then

$Res^*[S] = \bigcup_{i=0}^{\infty} Res^i[S] \subseteq \bigcup_{i=0}^{n} Res^i[S] \cup \bigcup_{i=n+1}^{\infty} Res^i[S] \subseteq Res^n[S] \cup \bigcup_{i=n+1}^{\infty} Res^n[S] = Res^n[S]$.

So, $Res^*[S] \subseteq Res^n[S]$. Since $Res^n[S] \subseteq Res^*[S]$, $Res^*[S] = Res^n[S]$. **Q.E.D.**

Now we will show that for all clause sets S, $S \equiv Res^*[S]$.

Proposition 1.8.12 ($S \equiv Res^*[S]$) *$S \equiv Res^*[S]$, i.e. S is semantically equivalent to the set of its resolvents.*

Proof: We'll show, by induction on n, that $S \equiv Res^n[S]$.
Basis: $n = 0$. Then $Res^0[S] = S$ and $S \equiv S$.
Inductive Step: Assume that $Res^n[S] \equiv S$. Then, $Res^{n+1}[S] = Res[Res^n[S]]$ and $Res[Res^n[S]] \equiv Res^n[S]$ by Proposition 1.8.9. Since $\equiv$ is an equivalence relation, $Res^{n+1}[S] \equiv S$.

Now let us show that $Res^*[S] \equiv S$. We'll show that for every truth assignment $\mathcal{A}$, $\models_{\mathcal{A}} S$ iff $\models_{\mathcal{A}} Res^*[S]$.
$\Longleftarrow$: Assume that $\models_{\mathcal{A}} Res^*[S]$. Since S is a subset of $Res^*[S]$, $\models_{\mathcal{A}} S$.
$\Longrightarrow$: Assume that $\models_{\mathcal{A}} S$. Let C be a clause in $Res^*[S]$. Then, there is some n such that $C \in Res^n[S]$. Since $Res^n[S] \equiv S$, $\models_{\mathcal{A}} Res^n[S]$. So, $\models_{\mathcal{A}} C$. Since C is arbitrary in $Res^*[S]$ we conclude that $\models_{\mathcal{A}} Res^*[S]$. **Q.E.D.**

Before we go on to the next two theorems let us define the concept of *derivation sequence.*

Definition 1.8.13 (derivation sequence) *A derivation of C from the clause set S is a sequence $C_1, \ldots, C_m$ satisfying the following conditions:*

1. every C_i , $1 \leq i \leq m$, is either a clause in S , or a resolvent of two preceding clauses, and

2. $C_m = C$.

The number of clauses in the derivation is called the length of the derivation.

Examples 1.8.14 Let us look at three derivation sequences.

1. The sequence C_1, C_2, C_3, C_4, C_5 , C_6, C_7, C_8 shown below is a derivation of $\square$ from the clause set $S = \{\{A, B\}, \{A, \neg B\}, \{\neg A, B\}, \{\neg A, \neg B\}\}$.

   $C_1 = \{A, B\}$ clause in S

   $C_2 = \{A, \neg B\}$ clause in S

   $C_3 = \{A\}$ from C_1 and C_2 on B

   $C_4 = \{\neg A, B\}$ clause in S

   $C_5 = \{\neg A, \neg B\}$ clause in S

   $C_6 = \{\neg A\}$ from C_4 and C_5 on B

   $C_7 = \square$ from C_3 and C_6 on A.

   Let us check that the sequence satisfies the conditions listed in Definition 1.8.13.

   1a. The clauses C_1, C_2, C_4, C_5 belong to S.

   1b. The clause C_3 is a resolvent of C_1 and C_2, C_6 is a resolvent of C_4 and C_5, and C_7 is a resolvent of C_5 and C_6. In all 4 cases the parent clauses precede the resolvent.

   2. The last clause of the sequence is $\square$.

   So, the sequence is a derivation of $\square$ from S.

2. Let us verify that the sequence below is another derivation of $\square$ from S.

   $C_1 = \{A, B\}$ clause in S

   $C_2 = \{A, \neg B\}$ clause in S

   $C_3 = \{A\}$ from C_1 and C_2 on B

   $C_4 = \{\neg A, B\}$ clause in S

   $C_5 = \{B\}$ from C_3 and C_4 on A

   $C_6 = \{\neg A, \neg B\}$ clause in S

   $C_7 = \{\neg A\}$ from C_4 and C_6 on B

   $C_8 = \{\neg B\}$ from C_3 and C_6 on A

$C_9 = \square$ from C_3 and C_7 on A

1. The clauses C_1, C_2, C_4, C_6 are members of S, and the other ones (C_3, C_5, C_7, C_8, C_9) are obtained from preceding clauses by resolution, and

2. The last clause in the sequence is $\square$.

This derivation is almost the same as the preceding one, except that we inserted clauses C_5 and C_8.

3. We can verify that the sequence below is a derivation of $\{A\}$ from $S = \{\{B\}, \{A, \neg B, C\}, \{A, \neg B, \neg C\}\}$.

$C_1 = \{B\}$ clause in S

$C_2 = \{A, \neg B, C\}$ clause in S

$C_3 = \{A, C\}$ from C_1 and C_2 on B

$C_4 = \{A, \neg B, \neg C\}$ clause in S

$C_5 = \{A, \neg C\}$ from C_1 and C_4 on B

$C_6 = \{A\}$ from C_3 and C_5 on C.

In this derivation we use clause 1 for more than 1 resolution step.

Observation 1.8.15 1. A derivation sequence is just a list of clauses. For example,

$C_1 = \{\neg A\}$, $C_2 = \{A, B\}$, $C_3 = \{A, \neg B\}$, $C_4 = \{A\}$, $C_5 = \square$

is a derivation of $\square$ from $\{\{\neg A\}, \{A, B\}, \{A, \neg B\}\}$.

2. A non-empty prefix of a derivation sequence is also a derivation sequence. More precisely, if $C_1, \ldots, C_n$ is a derivation of C_n from S, and $m \leq n$, then the sequence $C_1, \ldots, C_m$ is a derivation of C_m from S.

3. In the preceding examples we did more than just list clauses. We gave derivation analyses. An analysis of a derivation $C_1, \ldots, C_n$ has the same number of steps as the derivation, but each step contains both the clause and the explanation of how it was derived. The step specifies if the clause is in S. If it not, the step lists the parent clauses and the resolution atom.

Since the analysis contains more information, it is preferable to work with them. We can transform a derivation sequence into an analysis, but we have to do some work. Figure 1.55 shows an algorithm that accomplishes this transformation. The input to the algorithm is a derivation sequence $C_1, \ldots, C_n$ from the clause set S.

Let's apply the algorithm from Figure 1.55 to find an analysis for the derivation sequence

$C_1 = \{A, B, C\}$, $C_2 = \{A, B, \neg C\}$, $C_3 = \{A, \neg B\}$, $C_4 = \{\neg A, C\}$, $C_5 = \{\neg A, \neg C\}$, $C_6 = \{\neg A\}$, $C_7 = \{A, B\}$, $C_8 = \{A\}$, $C_9 = \square$

that computes $\square$ from the set $S = \{\{A, B, C\}, \{A, B, \neg C\}, \{A, \neg B\}, \{\neg A, C\}, \{\neg A, \neg C\}\}$.

We see that C_1 is a clause in S, so we write

$C_1 = \{A, B, C\}$ is a clause in S.

```
for (i=1 to n) do
{
    if (C_i ∈ S) then
        write on a new line that C_i is a clause in S.
    else
    {
        for (j=1 to i-1) do
            for (k=1 to i-1) do
                if (C_i is a resolvent of C_j and C_k on P) then
                {
                    write C_i from C_j and C_k on P ;
                    exit the double loop;
                }
    }
}
```

Figure 1.55: Algorithm for generating analyses

We go to the next clause in the sequence. This one is also in S, so we write

$C_2 = \{A, B, \neg C\}$ is a clause in S.

The next 3 clauses, C_3, C_4 and C_5 are also in S, so we write

$C_3 = \{A, \neg B\}$ is a clause in S

$C_4 = \{\neg A, C\}$ is a clause in S

$C_5 = \{\neg A, \neg C\}$ is a clause in S.

The next clause, C_6, is not in S. So, we look for a pair of clauses from $\{C_1, C_2, C_3, C_4, C_5\}$ that produces it. We get that C_6 is the resolvent of C_4 and C_5 on C. So, we write

$C_6 = \{\neg A\}$ from C_4 and C_5 on C.

Clause C_7 is not in S either, so we have to find its parents in the set $\{C_1, C_2, C_3, C_4, C_5, C_6\}$. We find the resolvents of all pairs C_i, C_j in the set, including the ones with $i = j$. If one of the resolvents is C_7, then those clauses are its parents. By doing this, we discover that C_7 is the resolvent of C_1 and C_2 on C. We write

$C_7 = \{A, B\}$ from C_1 and C_2 on C.

Clause C_8 is not in S, so we have to search the set $\{C_1, C_2, C_3, C_4, C_5, C_6, C_7\}$ for its parents. When we unify C_7 and C_3 on B we get C_8. We append the new line

$C_8 = \{A\}$ from C_7 and C_3 on B.

The last clause, C_9 is not in S. So, we have to locate its parents in the set $\{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$. After some searching we discover that it is the resolvent of C_8 and C_6 on A. We have the last line of the analysis.

$C_9 = \square$ from C_8 and C_6 on A.

So, the analysis of

$C_1 = \{A, B, C\}$, $C_2 = \{A, B, \neg C\}$, $C_3 = \{A, \neg B\}$, $C_4 = \{\neg A, C\}$, $C_5 = \{\neg A, \neg C\}$, $C_6 = \{\neg A\}$, $C_7 = \{A, B\}$, $C_8 = \{A\}$, $C_9 = \square$
is the step sequence below.
$C_1 = \{A, B, C\}$ is a clause in S
$C_2 = \{A, B, \neg C\}$ is a clause in S
$C_3 = \{A, \neg B\}$ is a clause in S
$C_4 = \{\neg A, C\}$ is a clause in S
$C_5 = \{\neg A, \neg C\}$ is a clause in S
$C_6 = \{\neg A\}$ from C_4 and C_5 on C
$C_7 = \{A, B\}$ from C_1 and C_2 on C
$C_8 = \{A\}$ from C_7 and C_3 on B
$C_9 = \square$ from C_8 and C_6 on A

Remark 1.8.16 In most cases we will not identify the formulas of the sequence by the labels $C_1, \ldots, C_n$. So, we will write
$\{A, B, C\}$, $\{A, B, \neg C\}$, $\{A, \neg B\}$, $\{\neg A, C\}$, $\{\neg A, \neg C\}$, $\{\neg A\}$, $\{A, B\}$, $\{A\}$, $\square$
instead of
$C_1 = \{A, B, C\}$, $C_2 = \{A, B, \neg C\}$, $C_3 = \{A, \neg B\}$, $C_4 = \{\neg A, C\}$, $C_5 = \{\neg A, \neg C\}$, $C_6 = \{\neg A\}$, $C_7 = \{A, B\}$, $C_8 = \{A\}$, $C_9 = \square$.

Now we can state the fact that we can always construct an analysis from a derivation sequence.

Proposition 1.8.17 *Algorithm 1.55 is correct, i.e. it attaches to every derivation sequence Δ, an analysis Δ^*, that provides, for each clause in Δ, an explanation of how the clause was derived.*

Proof: Let $\Delta = C_1, \ldots, C_n$ be a derivation sequence. For each clause C_i, $1 \leq i \leq n$, the algorithm checks if C_i is in S. If so, it writes that the clause is in S. If not, it looks for a pair of clauses C_j and C_k, $j, l < i$, that has C_i as a resolvent. Such a pair exists by Definition 1.8.13. Since the algorithm searches all possible pairs, it is bound to find one. **Q.E.D.**

Remark 1.8.18 The explanations generated by Algorithm 1.55 may not coincide with those of the original sequence. For example, the sequence $\{A\}, \{\neg A, B\}, \{B\}$ is a derivation of B from $S = \{\{A\}, \{\neg A, B\}, \{B\}\}$. Here, the clause B is obtained from the preceding clauses by resolution. However, the algorithm will generate the analysis,
1. $\{A\}$ clause in S
2. $\{\neg A, B\}$ clause in S
3. $\{B\}$ clause in S.

This situation can occur when a clause is both in S and can be derived from 2 preceding clauses, or it can be derived in more than one way.

Many times it is convenient to represent derivations as trees.

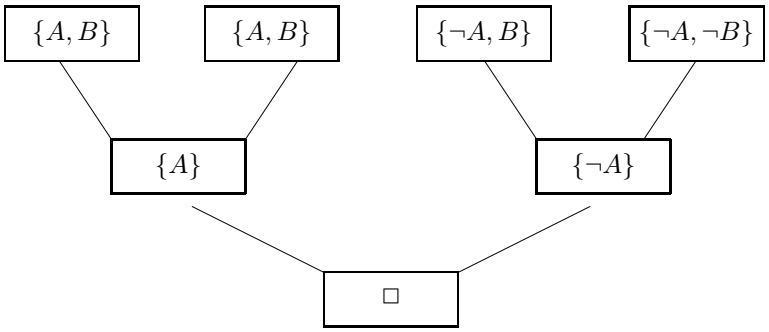


Figure 1.56: A tree for the first sequence of Example 1.8.14

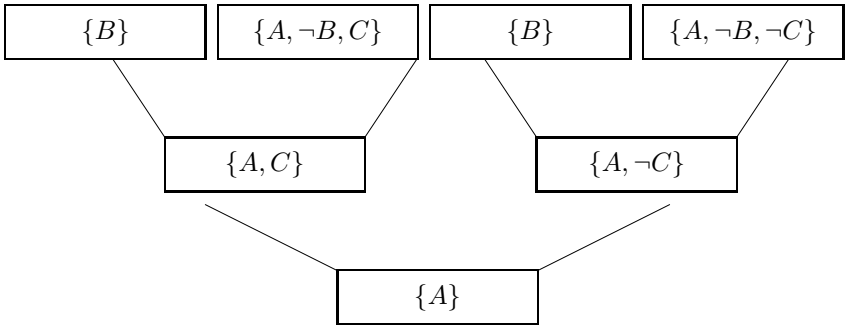


Figure 1.57: A tree for the third sequence of Example 1.8.14

Definition 1.8.19 (derivation tree) *A derivation tree*[10] *of a clause C from a clause set S is a labeled binary tree where*

1. *the leaves are labeled with clauses from S,*
2. *the label of the root is C,*
3. *the label of each branch is a resolvent of the labels of its children. We require that the left child contains the unification atom.*

Examples 1.8.20 Figures 1.56 and 1.57 show the derivation trees of the first and the third sequence of Examples 1.8.14.

We will show that we can transform derivation sequences into derivation trees and vice versa.

Proposition 1.8.21 *We can always construct a derivation sequence from a derivation tree.*

We will construct an algorithm, that performs exactly this. The algorithm is based on the observation that every derivation sequence must list the parents before the resolvent. In the resolution tree, the parents of the address u are

[10] Some resolution books call them refutation trees.

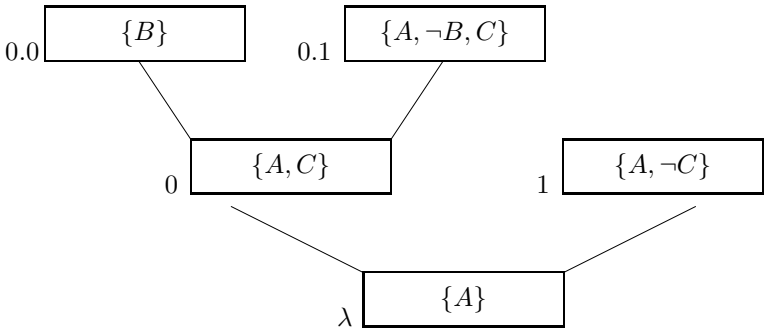


Figure 1.58: A derivation tree

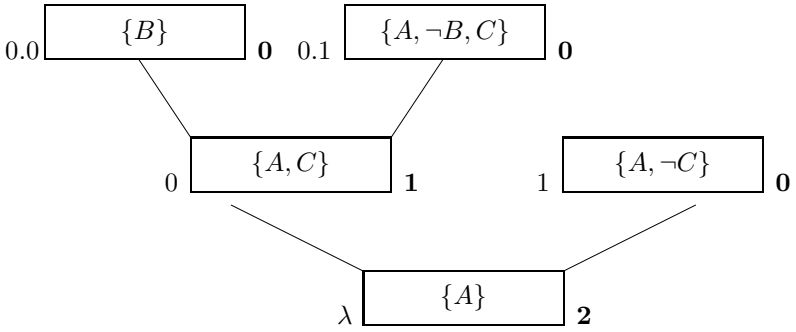


Figure 1.59: The ranked nodes of the tree from Figure 1.58

$u.0$ and $u.1$. The algorithm assigns to each node of the tree a natural number, called its **rank**. The rank is the *length* of the longest path from the node to a leaf.

$$rank(u) = \begin{cases} 0 & \text{if } u \text{ is a leaf} \\ max(rank(u.0), rank(u.1)) + 1 & \text{if } u \text{ is a branch} \end{cases}$$

Figure 1.59 attaches ranks to the nodes of the tree from Figure 1.58. The ranks are the bold numbers to the right of the nodes. Now, we list the nodes of the derivation tree in the increasing order of the rank. First we list all nodes of rank **0**, then the nodes of rank **1**, and so on. Within the same rank we can list the nodes in any order we like.

So, all six listing shown below are derivation sequences for Figure 1.58.

$\{B\}, \{A, \neg B, C\}, \{A, \neg C\}, \{A, C\}, \{A\}$
$\{B\}, \{A, \neg C\}, \{A, \neg B, C\}, \{A, C\}, \{A\}$
$\{A, \neg B, C\}, \{B\}, \{A, \neg C\}, \{A, C\}, \{A\}$
$\{A, \neg B, C\}, \{A, \neg C\}, \{B\}, \{A, C\}, \{A\}$
$\{A, \neg C\}, \{B\}, \{A, \neg B, C\}, \{A, C\}, \{A\}$
$\{A, \neg C\}, \{A, \neg B, C\}, \{B\}, \{A, C\}, \{A\}$

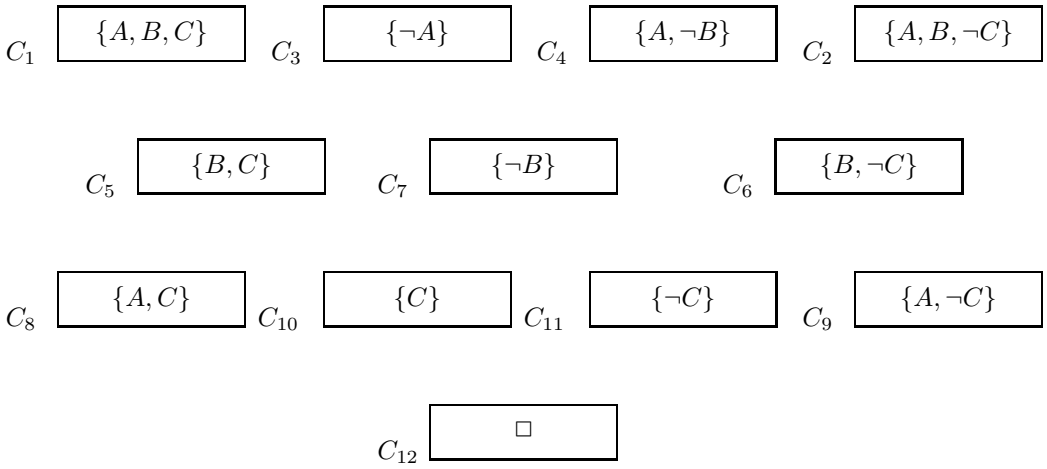


Figure 1.60: The node graph

The procedure works because each resolvent has higher rank than its parents, so it is listed after them. Also, the root, having the highest rank, is listed last.

Now we will give a procedure for generating trees from resolution sequences. Since Proposition 1.8.17 tells us that we can always transform a derivation into an analysis, the algorithm will take as input an analysis. We will illustrate the steps of the algorithm with an example, by constructing a derivation tree for the analysis below.

$C_1 = \{A, B, C\}$ clause in S
$C_2 = \{A, B, \neg C\}$ clause in S
$C_3 = \{\neg A\}$ clause in S
$C_4 = \{A, \neg B\}$ clause in S
$C_5 = \{B, C\}$ from C_1, C_3 on A
$C_6 = \{B, \neg C\}$ from C_2,C_3 on A
$C_7 = \{\neg B\}$ from C_4, C_3 on A
$C_8 = \{A, C\}$ from C_1, C_7 on B
$C_9 = \{A, \neg C\}$ from C_2, C_7 on B
$C_{10} = \{C\}$ from C_5, C_7 on B
$C_{11} = \{\neg C\}$ from C_6, C_7 on B
$C_{12} = \square$ from C_{10}, C_{11} on C.

Step 1: Represent all clauses as nodes labeled with that clause. We get the graph from Figure 1.60. The labels C_i are to the left of the nodes.
Step 2: Draw arrows from the parent clauses to the resolvents. We label with p the arrow from the positive parent (the clause that has the unification atom) and with n the negative parent (the one that has the complement of the unification atom). We get the graph from Figure 1.61.
Step 3: Remove all nodes that do not have a path to the root.

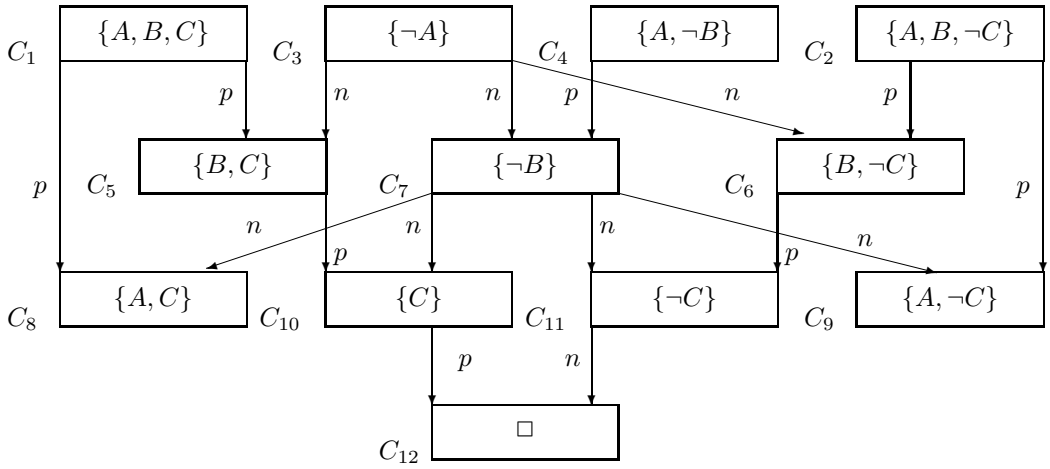


Figure 1.61: The graph with arrows

The nodes C_8 and C_9 have no path to C_{12}, so we remove them together with the connecting arrows. We get the reduced graph from Figure 1.62.
Step 4: The graph generated by Step 3 has no cycles because every arrow $C_i \longrightarrow C_j$ has $i < j$. At the same time, every node has a path to the root. But this graph may not be a tree because there may be more than one path from the node to the root. For example, C_3 from Figure 1.62 has 3 paths to the root, C_3, C_5, C_{10}, C_{12}, C_3, C_7, C_{10}, C_{12}, and C_3, C_6, C_{11}, C_{12}. We have multiple paths because some nodes have out-degree greater than 1. We call the arrows that have C_i as source *out-arrows* of C_i. For each extra arrow of C_i we will make a copy of the tree with root C_i and will make the new root point to the target of the extra arrow. But we have to be careful about the way we do it. So, let T be the set of clauses with more than one out-arrow. We sort T in the increasing order of the index and we process the nodes according to this order. Since the descendents of C_j have smaller indices than j, they are processed first. By the time we process C_j, they form a tree with root C_j.

Figure 1.62 has 2 nodes of out-degree greater than 1, C_3 and C_7. Since $3 < 7$, we process C_3 first. This node has 2 extra out-arrows, so we make two copies of the tree with root C_3. We make them point to C_6 and C_7. We get the graph from Figure 1.63. Now we process C_7. We make a copy of the tree with root C_7 that points to C_{11}, and we get the tree from Figure 1.64.

We label the newly created nodes with C_{13} through C_{17} and get the binary tree from Figure 1.65. The left child has label p and the right child has label n.

The algorithm for generating trees from sequences consists of the 4 steps described above. The algorithm does not implement a one-to-one mapping since the first two sequences of Examples 1.8.14, though distinct, generate the same tree, the graph shown in Figure 1.56.

However, the mapping is onto since every derivation tree is generated by at

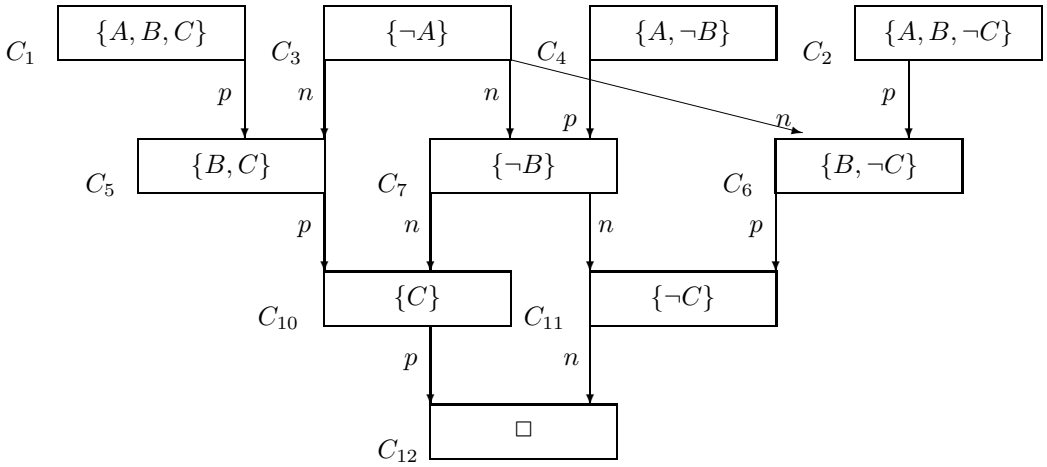


Figure 1.62: The reduced graph

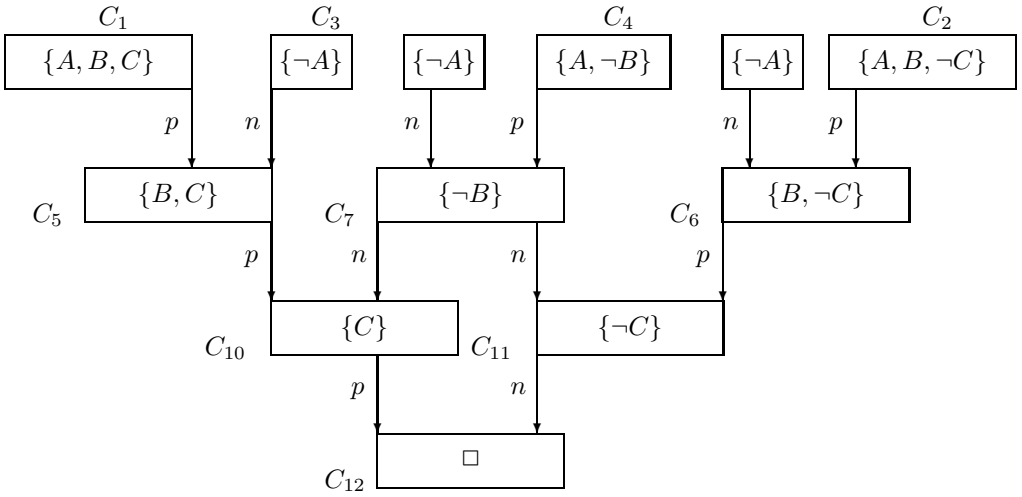


Figure 1.63: The graph after processing C_3

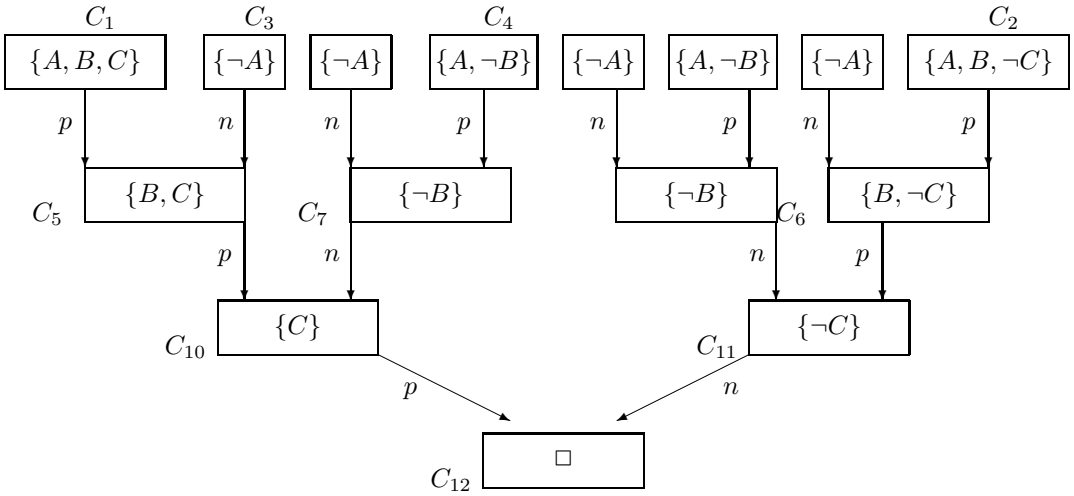


Figure 1.64: The resolution tree

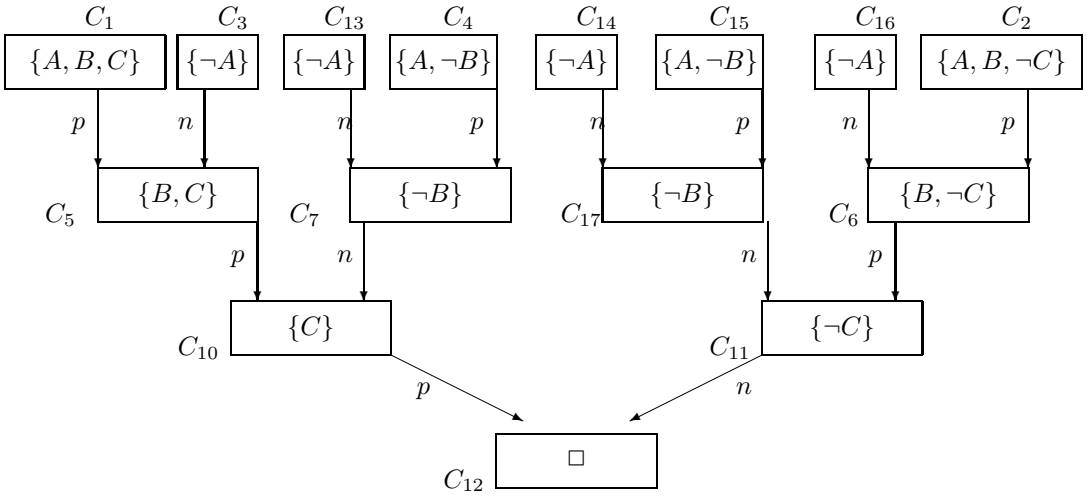


Figure 1.65: The labeled resolution tree

least one sequence.

Proposition 1.8.22 *The function computed by the sequence-to-tree algorithm is onto, i.e. every derivation tree is the image of some sequence.*

Proof: Let T be a tree and let $C_1,\ldots,C_n$ be the sequence generated by the tree-to-sequence algorithm. We transform this sequence into an analysis that preserves the structure of T. So, C_i is a resolvent of C_j and C_k on A in the sequence iff C_i is a resolvent of C_j and C_k on A in T.

Now let us apply the 4 steps of the algorithm. At Step 1 we draw the nodes for the n clauses. At Step 2 we draw arrows from the parents to the resolvents. Since the analysis mirrors T, we have an arrow from C_j to C_k iff we have an arrow from C_j to C_k in T. Moreover, the label of the arrow is the same.

So, the graph obtained at Step 2 is the same as the tree T. Then at Steps 3, and 4 we do not do anything because every node has a path to the root, and there are no extra out-arrows. **Q.E.D.**

Remark 1.8.23 Since the fourth step of the algorithm is straightforward, the book presents many resolution trees as acyclic graphs where every node has a path to the root. In many cases we will not even write p and n on the arrows since they can be determined by looking at the resolution step.

Examples 1.8.14 displayed two derivations of $\Box$ from $S = \{\{A, B\}, \{A, \neg B\}, \{\neg A, B\}, \{\neg A, \neg B\}\}$, the first being shorter than the second.

Let C be a clause and S be a set of clause. We write $Deriv(C, S)$ for the set of all derivations of C from S. If $Deriv(C, S)$ is not empty, then it contains derivations of minimal length. These are called *minimal derivations*. Since every derivation can be transformed into a tree, there are derivations trees of C from S. Among those trees are trees of minimal height. Those are called *minimal (derivation) trees*.

Proposition 1.8.24 *1. $Deriv(C, S)$ is either empty or countably infinite.*

2. If $C_1, \ldots, C_n = C$ is a minimal derivation, then no clause is repeated and every index $1 \leq i < n$ has a path to n, i.e. there are indices $i = j_0 < j_1 < \ldots < j_k = n$ such that for all l, $0 < l \leq k$, $C_{j_{l-1}}$ is a parent of C_{j_l}.

3. If t is a minimal tree of height $n+1$, one of its main subtrees is a minimal tree of height n.

Proof: 1. Since there are countably many formulas, $Derive(C, S)$ is at most countably infinite. So, all we have to show is that the set is either empty or infinite. Let us assume that $Derive(C, S)$ is not empty. Then there is a derivation $C_1, \ldots, C_n = C$ of C from S. At the same time, S cannot be empty. Let D be a clause in S. Then all sequences

$$\underbrace{D, \ldots, D}_{ntimes}, C_1 \ldots, C_n$$

are derivations of C from S. Since we have infinitely many such sequences, the set $Deriv(C, S)$ is infinite.

2. We will first show that the sequence $Seq = C_1, \ldots C_n$ has no duplications. Assume that for some $1 \leq i < j \leq n$, $C_i = C_j$.

If $j = n$ then $C_i = C$. By the second part of Observations 1.8.15, $C_1, \ldots, C_i = C$ is a derivation of C from S. The length of this sequence is i and i is less than n. This contradicts the assumption that $C_1 \ldots, C_n$ is a minimal sequence.

If $j < n$, we delete C_j from the sequence. We will show that $Seq^\dagger = C_1, \ldots, C_{j-1}, C_{j+1}, \ldots, C_n$ is a derivation sequence. We need to show that every clause in $Seq^\dagger$ is either in S or a resolvent of two clauses that precede it in $Seq^\dagger$.

Let C_k be a clause in $Seq^\dagger$. Since $C_k \in Seq$, C_k is either in S, or a resolvent of two clauses, C_p and C_q, that precede it in Seq. If C_k belongs to S we have no problem. If C_k is a resolvent of C_p and C_q, and both are different of C_j, then C_p and C_q are in $Seq^\dagger$, and precede C_k. Again there is no problem. If C_p, C_q, or both, are equal to C_j, then $k > j$, hence $k > i$. We define the clauses D_p and D_q as follows:

$$D_p = \begin{cases} C_p & \text{if } p \neq j \\ C_i & \text{if } p = j \end{cases}$$

$$D_q = \begin{cases} C_q & \text{if } q \neq j \\ C_i & \text{if } q = j \end{cases}$$

Now C_k is a resolvent of D_p and D_q and these clauses precede C_k in $Seq^\dagger$.

Now let us show that every clause of a minimal sequence has a path to the last clauses of the sequence. We prove it by contradiction. Assume that there is a minimal derivation $Seq = C_1, \ldots, C_n$ with clauses that have no paths to C_n. Since the sequence is finite, there is a largest m, such that m has no path to C_n. From the definition of a path to n, $m < n$. If C_m has a resolvent in Seq, then the resolvent has an index higher than m, and has no path to C_n. This contradicts the assumption that C_m is the last such clause with no path to C_n. So, C_m has no resolvents in Seq.

Then let $Seq^\dagger$ be the sequence obtained from $C_1, \ldots, C_m, \ldots, C_n$ by deleting C_m. This sequence is a derivation sequence, since no clause uses C_m. At the same time the end clause C_n is the same. This contradicts the fact that Seq is a minimal derivation.

3. Let t be a minimal derivation tree of C from S and $n+1$ be its height, as shown in Figure 1.66. Let C be the label of λ, C_0 the label of the address 0 and C_1 the label of the address 1. Let t_0 and t_1 be the main subtrees of t. Let us assume that all subtrees of t of height n are not minimal. Let us define the derivation trees t'_0 and t'_1 as follows:

$$t'_i = \begin{cases} t_i & \text{if the height of } t_i \text{ is less than } n \\ T_i & \text{if } T_i \text{ is a minimal derivation tree of } C_i \text{ from } S \end{cases}$$

.

Then the tree $t[0 \leftarrow t'_0][1 \leftarrow t'_1]$ is a derivation tree of C from S. Since both t'_1 and t'_2 have height less than n, $t[1 \leftarrow t'_1][2 \leftarrow t'_2]$ has height n or less. This contradicts the fact that t is a minimal tree. **Q.E.D.**

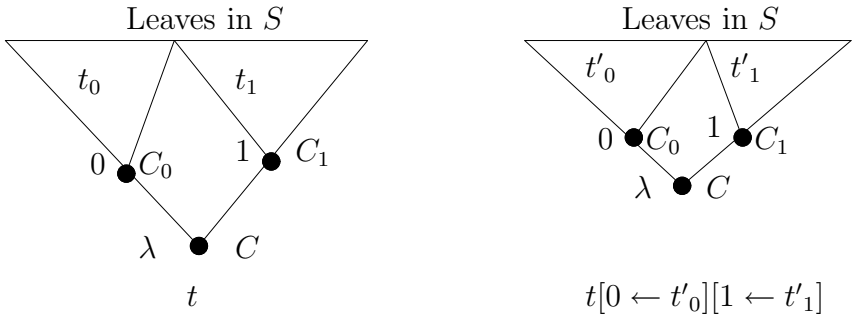


Figure 1.66: The trees from the proof of Proposition 1.8.24

The derivation trees give meaning to the sets $Res^n[S]$ from Definition 1.8.10.

1.$Res^n[S]$ is the set of all clauses that have an S derivation tree of height $\leq n$.

2. $Res^*[S]$ is the set of all clauses derivable from S.

3. $Res^{n+1}[S] - Res^n[S]$ is the set of clauses derivable from S that have minimal derivation trees of height $n+1$.

4. The condition $Res^{n+1}[S] = Res^n[S]$ of Proposition 1.8.11 tells us that there are no minimal trees of height $n+1$.

5. The conclusion $Res^*[S] = Res^n[S]$ of Proposition 1.8.11 says that all minimal trees have height less than or equal to n.

Now we will present a key result, **The Compactness Theorem** for the propositional calculus.

Theorem 1.8.25 (The Compactness Theorem) *Let S be set of formulas. Then S has a model if and only if every finite subset of S has a model.*

Proof: $\Longrightarrow$: Assume that $\models_{\mathcal{A}} S$. Then, for every subset T of S, $\models_{\mathcal{A}} T$. Since the finite subsets of S are subsets of S, $\mathcal{A}$ is a model for every finite subset.

$\Longleftarrow$: Assume that every finite subset of S has a model. If the whole set S is finite then S also has a model, since it is a finite subset of itself.

So, let us assume that S is infinite. We define the subset S_i of S to be the set of all formulas of S with atoms in the set $\{P_0, P_1, \ldots, P_i\}$. We notice that all formulas of S_i are also in S_{i+1}, S_{i+2}, etc. So, for all i, j, $i < j$ implies $S_i \subseteq S_j$.

The sets S_i can be infinite because we can form infinitely many formulas with only i atoms. However, S_i contains only a finite number of *non-equivalent* formulas (see Exercise 1.8.11 below). From each equivalence class of S_i we select a representative and form the representative set M_i. The sets M_i are finite and semantically equivalent to S_i. We require consistency in the selection of the representatives. This means that whenever an equivalence class of S_{i+1} has a member in M_i, that member is the representative of the class in M_{i+1}.

For example, let us assume that P_2 is in M_2 and C is the equivalence class of S_3 that contains $F = P_2 \wedge (P_2 \vee P_3)$. The formula F is not in S_2 because it

$\mathcal{A} = \phi$; $I = \{0, 1, 2, \ldots, n, \ldots\}$; $n = 0$;
loop
{
if ($\{i \in I | \mathcal{A}_i[P_n] = 1 \}$ is infinite) then
$\{\mathcal{A} = \mathcal{A} \cup \{< P_n, 1 >\}$; $I = I - \{i | \mathcal{A}_i[P_n] = 0\}$; $\}$
else
$\{\mathcal{A} = \mathcal{A} \cup \{< P_n, 0 >\}$; $I = I - \{i | \mathcal{A}_i[P_n] = 1\}$; $\}$
$n = n + 1$
}

Figure 1.67: The Compactness Theorem Algorithm

contains the atom P_3. However, $F \equiv P_2$, so C contains a member of M_2. Then, the representative of C in M_3 is P_2. This way,

(1) $M_0 \subseteq M_1 \subseteq M_2 \subseteq \ldots \subseteq M_n \subseteq \ldots$.

Since M_i is finite, it has a model, say $\mathcal{A}_i$. From (1) we get that for all $i \leq j$, $\mathcal{A}_j$ is a model of M_i.

We need to construct a model $\mathcal{A}$ of S from the truth assignments $\mathcal{A}_i$. We will construct $\mathcal{A}$ inductively, by first assigning a truth value to P_0, then to P_1, and so on. The problem is that the models $\mathcal{A}_i$ may assign different values to the same atom. For example, P_0 has value 1 in $\mathcal{A}_0$, 0 in $\mathcal{A}_1$, 1 in $\mathcal{A}_2$ and the value keeps flip-flopping between 1 and 0. So, what value should $\mathcal{A}$ assign to P_0?

We divide the models $\mathcal{A}_i$ into two categories: in the first category P_0 receives 1 and in the second it gets 0. If the first category has infinitely many indices, then $\mathcal{A}[P_0] = 1$ and we discard all indices that are in the second category, i.e. all i that have $\mathcal{A}_i[P_0] = 0$. If the first category is finite then $\mathcal{A}[P_0] = 0$ and we discard all indices i that have $\mathcal{A}_i[P_0] = 1$.

Let I be the reduced set of indices. For all models M_i, with $i \in I$, $\mathcal{A}_i[P_0] = \mathcal{A}[P_0]$. Now we worry about the value of P_1. We split I into two categories according to whether $\mathcal{A}_i[P_1] = 1$ or $\mathcal{A}_i[P_1] = 0$. If the first category is infinite then $\mathcal{A}[P_1] = 1$ and we remove from I all the indices that are in the second category. If the first category is finite then we set $\mathcal{A}_i[P_1] = 0$ and we remove from I the indices from the first category. This way, the indices left in I agree on *both* P_0 and P_1. We continue this way, according to the algorithm from Figure 1.67.

Now let us show that the above algorithm is correct, i.e. it produces a model of S. Let F be a formula in S. Since F is finite, it has a finite number of atoms. Then, there is a number n such that all atoms of F are in the set $\{P_0, P_1, \ldots, P_n\}$. So, $F \in S_n$. Since M_n contains a representative of each $\equiv$ class of S_n, there is a formula $G \in M_n$ such that $G \equiv F$.

Let us look what happens after we go $n + 1$ times through the loop of The Compactness Theorem Algorithm. The set I is infinite and for all indices i in I, $\mathcal{A}_i$ agrees with $\mathcal{A}$ on $\{P_0, P_1, \ldots, P_n\}$. Let us take the smallest index k in I that is greater than or equal to n. Then $\mathcal{A}_k$ is a model of M_n since $k \geq n$. So,

$n = 0$; $S = \{\phi(0)\}$;
loop
{
if (S is unsatisfiable) then
{ write("The set is unsatisfiable"); exit; }
else
$\{n = n + 1; S = S \cup \{\phi(n)\}$;
}

Figure 1.68: The Unsatisfiability Procedure

$\mathcal{A}_k$ is a model of G. Since $\mathcal{A}$ agrees with $\mathcal{A}_k$ on the atoms of G, $\mathcal{A}$ is a model of G. But $F \equiv G$, so $\mathcal{A}$ is a model **Q.E.D.**

The Compactness Theorem gives us a procedure for deciding the unsatisfiability of a set of formulas. Let $U = \{F_0, F_1, \ldots, F_n \ldots\}$ be a set of formulas that can be be *enumerated*, i.e. we have an algorithm ϕ such that $\phi(i) = F_i$.

Then, the procedure from Figure 1.68 will stop iff S is unsatisfiable. Let us see why. If U is unsatisfiable, then it has a finite subset of formulas, say $T = \{F_{i_1}, \ldots, F_{i_k}\}$ that is unsatisfiable. Let m be the largest of the indices $i_1, \ldots, i_k$. When $n = m$, the procedure will test the satisfiability of $S = \{F_0, \ldots, F_m\}$, a set that includes T. So, S is unsatisfiable and the procedure stops.

The procedure goes only half-way, because it stops only when U is unsatisfiable. When U is satisfiable, the procedure loops forever. For this reason we call it a *semi-decision procedure*.

Before we go on to prove The Resolution Theorem, we need to define the sets $S_{L=0}$ and $S_{L=1}$ that will occur in many proofs.

Definition 1.8.26 ($S_{L=1}$, $S_{L=0}$) *Let S be a set of clauses and L be a literal in S. The set $S_{L=1}$ is obtained from S by deleting the clauses that contain L, and removing $\bar{L}$ from the remaining clauses. Formally, $S_{L=1} = \{C - \{\bar{L}\} | C \in S$ and $L \notin C\}$.*

The set $S_{L=0}$ is S obtained by deleting the clauses that contain L, and then deleting L from the leftover clauses. Formally, $S_{L=0} = \{C - \{L\} | C \in S$ and $\bar{L} \notin C\}$.

Semantically, $S_{L=1}$ is the set of S clauses that are left unsatisfied by the truth assignment $\mathcal{A}[L] = 1$, and $S_{L=0}$ contains the clauses left unsatisfiable by the assignment $\mathcal{A}[L] = 0$. The two sets, $S_{L=1}$ and $S_{L=0}$, have fewer atoms than S since they do not contain neither L nor $\bar{L}$.

Example 1.8.27 Let $S = \{\{A, B, C\}, \{A, B, \neg C\}, \{A, \neg B\}, \{\neg A, B\}, \{\neg A, \neg B\}\}$. We get $S_{A=1}$ by deleting all clauses of S that contain A, i.e. the clauses $\{A, B, C\}$, $\{A, B, \neg C\}$, $\{A, \neg B\}$. We are left with $\{\neg A, B\}$ and $\{\neg A, \neg B\}$. From these clauses we delete $\neg A$. We obtain $S_{A=1} = \{\{B\}, \{\neg B\}\}$.

We get $S_{A=0}$ by deleting all clauses of S that contain $\neg A$, i.e. $\{\neg A, B\}$ and $\{\neg A, \neg B\}$. We are left with $\{A, B, C\}$, $\{A, B, \neg C\}$, and $\{A, \neg B\}$. From these clauses we delete A and obtain $S_{L=0} = \{\{B, C\}, \{B, \neg C\}, \{\neg B\}\}$.

Now, let us verify that the clauses of S left unsatisfied by the truth-assignment $\mathcal{A}[A] = 1$ is exactly $S_{A=1}$. The disjunctions that contain A are satisfied since $\mathcal{A}$ satisfies A. We left with $\{\neg A, B\}$ and $\{\neg A, \neg B\}$. Since A is interpreted as 1, $\mathcal{A}$ satisfies $\{\neg A, B\}$ iff is a model for $\{B\}$. The same goes for $\{\neg A, \neg B\}$. Then satisfies S iff $\mathcal{A}$ is a model for $\{\{B\}, \{\neg B\}\} = S_{A=1}$.

Lemma 1.8.28 *Let S be a set of clauses and L a literal in S. Then S is unsatisfiable iff both $S_{L=1}$ and $S_{L=0}$ are unsatisfiable.*

Proof: The above statement is equivalent to

S is satisfiable iff at least one of $S_{L=1}$ and $S_{L=0}$ is satisfiable.

$\Longleftarrow$: We will show that whenever $S_{L=1}$ is satisfiable, so is S. The other part, that S is satisfiable whenever $S_{L=0}$ has a model, is left as exercise.

Assume that $\mathcal{A}$ is a model of $S_{L=1}$. Let $\mathcal{B}$ be the truth assignment identical to $\mathcal{A}$ except that $\mathcal{B}[L] = 1$. Then $\mathcal{A}$ and $\mathcal{B}$ agree on $S_{L=1}$, because neither L nor $\bar{L}$ occur in $S_{L=1}$. Then $\mathcal{B}$ is a model for all clauses of $S_{L=1}$.

Now let C be a clause in S. We have the following 3 cases: C contains L, C contains $\bar{L}$, but not L, and neither L nor $\bar{L}$ are in C.

If C contains L then $\models_{\mathcal{B}} C$, because $\mathcal{B}[L] = 1$. If L is not in C but $\bar{L}$ is, $C - \{\bar{L}\}$ is in $S_{L=1}$. Since $\models_{\mathcal{B}} C - \{\bar{L}\}$, $\models_{\mathcal{B}} C$. If neither L nor $\bar{L}$ are in C, then C is in $S_{L=1}$, so $\models_{\mathcal{B}} C$. So, for all three cases, $\models_{\mathcal{B}} C$. We conclude that $\models_{\mathcal{B}} S$.

$\Longrightarrow$: Assume that $\models_{\mathcal{A}} S$. Then, either $\mathcal{A}[L] = 0$ or $\mathcal{A}[L] = 1$. Assume that $\mathcal{A}[L] = 0$. Now let C be a clause in $S_{L=0}$. Then either $C \in S$ or $C \cup \{L\}$ is in S. In the first case, $\models_{\mathcal{A}} C$. In the second case, $\models_{\mathcal{A}} C \cup \{L\}$. Since $\mathcal{A}[L] = 0$, the preceding relation implies that $\models_{\mathcal{A}} C$. In both cases $\models_{\mathcal{A}} C$, so $\mathcal{A}$ is a model for $S_{L=0}$. In a similar fashion we can show that $\mathcal{A}[L] = 1$ implies that $\models_{\mathcal{A}} S_{L=1}$. **Q.E.D.**

We will use The Compactness Theorem to prove *The Resolution Theorem.*

Theorem 1.8.29 (The Resolution Theorem) *A set of clauses S is unsatisfiable if and only if $\Box \in Res^*[S]$.*

Proof: $\Longleftarrow$ Assume that $\Box \in Res^*[S]$. By Proposition 1.8.12, $S \equiv Res^*[S]$. So, S is also unsatisfiable.

$\Longrightarrow$ Assume that S is unsatisfiable. By The Compactness Theorem, it has a finite subset T that is unsatisfiable. So, our proof reduces to proving that

whenever T is an unsatisfiable finite set of clauses, $\Box \in Res^*[T]$.

The proof is by induction on n, the number of atoms in T.

Basis ($n = 0$): Then $T = \{\Box\}$, and $\Box$ is unsatisfiable.

Inductive Step: Assume that for all unsatisfiable sets with n or fewer atoms, $\Box \epsilon Res^*[S]$. Let T be an unsatisfiable set with $n+1$ atoms and let P be an atom in T. We know, from Lemma 1.8.28, that $T_{P=1}$ and $T_{P=0}$ are both unsatisfiable. Neither one contains P, so they have at most n atoms. By induction hypothesis, both $Res^*[T_{P=1}]$ and $Res^*[T_{P=0}]$ have derivations trees of $\Box$, as shown in Figure 1.69.

The tree t_0 has leaves in $T_{P=0}$ and t_1 has leaves in $T_{P=1}$. If either one of these trees is in S, then we are done. If not, some leaves in t_0 are missing the P

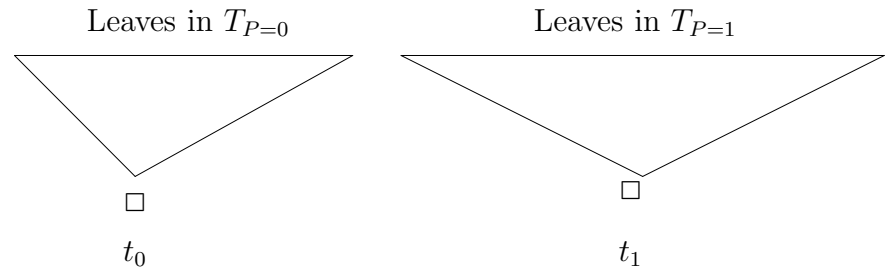


Figure 1.69: The resolution trees for $T_{P=0}$ and $T_{P=1}$

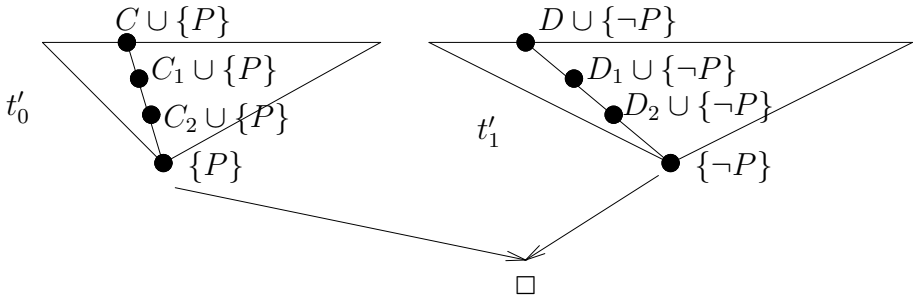


Figure 1.70: The resolution tree for T

literal and some leaves in t_1 are lacking $\neg P$. We add P to all leaves of t_0 that are not in T and to all clauses that lie on the path from these leaves to the root, including the root. This way we obtain a derivation tree t'_0 of P from T.

We do the same thing to t_1 and obtain a derivation tree t'_1 of $\neg P$ from T. Now we apply one more resolution step, to the roots of the two trees. We get the resolution tree from Figure 1.70. So, $\square \in Res^*[T]$. **Q.E.D.**

The next example illustrates the construction used in the inductive step of the proof.

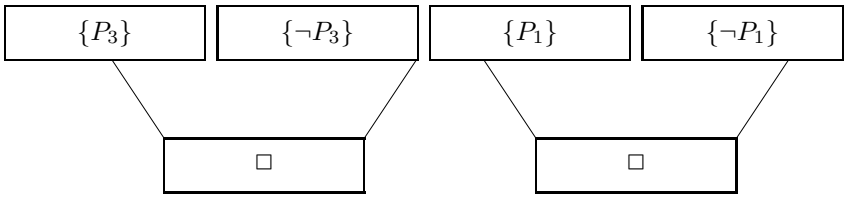
Example 1.8.30 Let us eliminate P_4 from $T = \{\{P_2, P_4, \neg P_4\}\{P_1, \neg P_4\}, \{\neg P_1\}, \{P_3, P_4\}, \{\neg P_3, P_4\}\}$.

We get $T_{P_4=0} = \{\{\neg P_1\}, \{P_3\}, \{\neg P_3\}\}$ and $T_{P_4=1} = \{\{P_1\}, \{\neg P_1\}\}$.

We apply the induction hypothesis to $T_{P_4=0}$ and to $T_{P_4=0}$ and get the trees t_0 and t_1 from Figure 1.71.

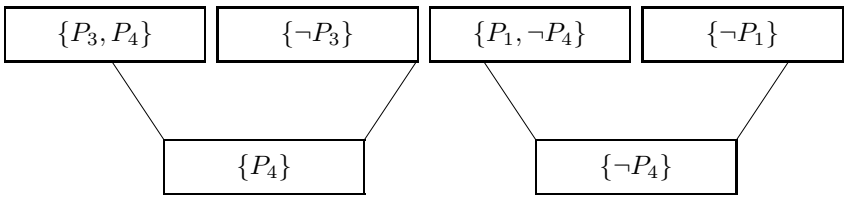
None of these trees is a derivation of $\square$ from T because the leaves $\{P_3\}$ and $\{P_1\}$ are not in T. So we add P_4 to $\{P_3\}$ and $\neg P_4$ to $\{P_1\}$. We also have to add P_4 to all ancestors of $\{P_3\}$ and $\neg P_4$ to all ancestors of P_1. We get the trees t'_1 and t'_2 from Figure 1.72. These trees have all leaves in T.

Now we derive $\square$ from $\{P_4\}$ and $\{\neg P_4\}$ and get a derivation tree of $\square$ from T, as shown in Figure 1.73.



t_0 t_1

Figure 1.71: The trees t_0 and t_1 from Example 1.8.30



t_0' t_1'

Figure 1.72: The trees t_0' and t_1' from Example 1.8.30

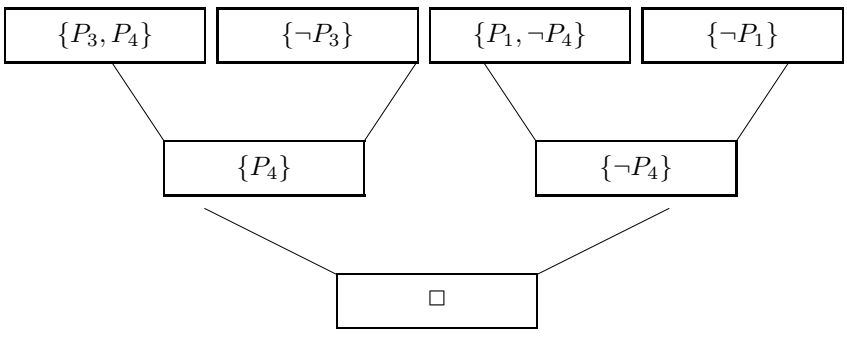


Figure 1.73: The derivation tree of □ for Example 1.8.30

$T = \phi$;
while ($\square \notin T$ and $T \neq S$) do
{
    $T = S; S = Res[S]$;
}
if ($\square \in S$) then success else failure;

Figure 1.74: A resolution algorithm

The Resolution Theorem gives us an algorithm for proving that a finite set of clauses is unsatisfiable (Figure 1.74). The algorithm keeps computing $Res^1[S], Res^2[S], \ldots$ until it encounters $\square$ or it finds some n such that $Res^n[S] = Res^{n+1}[S]$. Since S is finite, $Res^*[S]$ is also finite, and the algorithm terminates.

Exercises

Exercise 1.8.1 *Find all resolvents of the following pairs of clauses:*
1. $C_1 = \{A, \neg B, C\}$, $C_2 = \{\neg A\}$
2. $C_1 = \{A, \neg B\}$, $C_2 = \{\neg A, B\}$
3. $C_1 = \{A, B, \neg C\}$, $C_2 = \{\neg B\}$
4. $C_1 = \{\neg A\}$, $C_2 = \{A\}$

Remember that the positive literal can occur in either C_1 or C_2.

Exercise 1.8.2 *In Section 1.6 we defined the element-wise equivalence of two sets of formulas as*

S and T are element-wise equivalent iff every formula of S has an equivalent formula in T and every formula of T has an equivalent formula in S.

How does this equivalence relation compare with the semantic equivalence of sets?

Exercise 1.8.3 *Find $Res[S]$ for the following sets of clauses:*
1. $S = \{\{A, B, C\}, \{A, B, \neg C\}, \{A, \neg B, C\}, \{A, \neg B, \neg C\}, \{\neg A, B, C\}, \{\neg A, B, \neg C\}, \{\neg A, \neg B, C\}, \{\neg A, \neg B, \neg C\}\}$
2. $S = \{\{A\}, \{\neg A, B, C\}, \{\neg A, B, \neg C\}, \{\neg A, \neg B, C\}, \{\neg A, \neg B, \neg C\}\}$
3. $S = \{\{A, B\}, \{A, \neg B, C\}, \{A, \neg B, \neg C\}, \{\neg A, B\}, \{\neg A, \neg B, C\}, \{\neg A, \neg B, \neg C\}\}$

Exercise 1.8.4 *Let F,G, be two formulas in T such that $F \models G$. Show that $T \equiv T - \{G\}$.*

Exercise 1.8.5 *Prove that the relation semantic equivalence of sets is an equivalence relation. Do the set operations intersection, union, and difference preserve the equivalence?*

Exercise 1.8.6 *Show that every set of formulas, finite or infinite, is semantically equivalent to a countable set of clauses.*

Exercise 1.8.7 *Compute $Res^*[S]$ for the following sets of clauses:*
1. $S = \{\{\neg A, \neg B, C\}, \{A\}, \{\neg A, B\}, \{\neg A, \neg B, \neg C\}\}$
2. $S = \{\{A, B\}, \{A, \neg B\}, \{\neg A, B\}, \{\neg A, \neg B\}\}$.

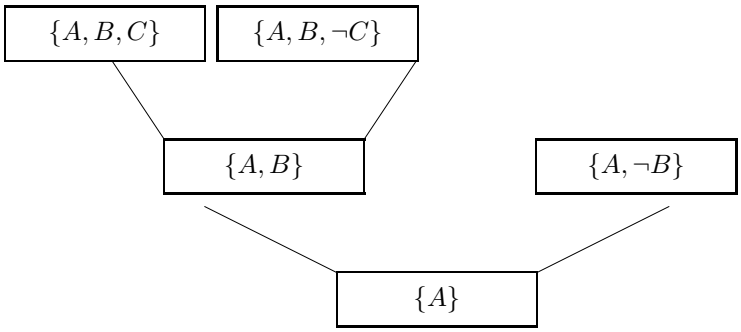


Figure 1.75: The first derivation tree

Exercise 1.8.8 *Specify $Res^n[S]$ and $Res^*[S]$ for the following sets:*
1. $S = \{P_0\} \cup \{\{\neg P_i, P_{i+1}\} | i \in \{0, 1, \ldots\}\}$
2. $S = \{P_0, P_1\} \cup \{\{\neg P_i, P_{i+2}\} | i \in \{0, 1, \ldots\}\}$.

Exercise 1.8.9 *Let U be the set of all clauses that can be formed with the atoms $P_0, \ldots, P_n$. Show that Res^* is a closure operation on S, i.e. the 4 relations below hold for all $S, T \subseteq U$.*
1. $S \subseteq Res^*[S]$,
2. $S \subseteq T$ *implies* $Res^*[S] \subseteq Res^*[T]$,
3. $Res^*[S] = Res^*[Res^*[S]]$,
4. $Res^*[U] = U$.

Exercise 1.8.10 *$Res^{n+1}[S] = Res^n[S]$ means that there are no S-resolution trees of minimal height $n + 1$. Use this fact to give another proof of Proposition 1.8.11.*

Exercise 1.8.11 *Show that we can form only $2^{2^{n+1}}$ nonequivalent formulas with the atoms $P_0, ..., P_n$.*

Exercise 1.8.12 *Construct a derivation of A from $S = \{\{\neg B\}, \{B, C\}, \{A, \neg C, \neg D\}, \{A, \neg C, D\}\}$*

Exercise 1.8.13 *Construct 2 different derivations of $\{B, C\}$ from $S = \{\{A, B, C, \neg D\}, \{A, B, C, D\}, \{\neg A, B\}, \{\neg A, \neg B\}\}$*

Exercise 1.8.14 *Construct an analysis for the derivation of $\neg A$ from $S = \{\{\neg A, B, C\}, \{\neg D, \neg C\}, \{D, \neg C\}, \{\neg A, \neg B, D\}, \{\neg A, \neg B, \neg D\}\}$ shown below.*
$C_1 = \{\neg A, B, C\}, C_2 = \{D, \neg C\}, C_3 = \{\neg D, \neg C\}, C_4 = \{\neg A, \neg B, D\}, C_5 = \{\neg C\}, C_6 = \{\neg A, B\}, C_7 = \{\neg A, \neg B, \neg D\}, C_8 = \{\neg A, \neg B\}, C_9 = \{\neg A\}$.

Exercise 1.8.15 *Apply the Tree-to-sequence Algorithm to generate all possible derivations of the trees from Figures 1.75 and 1.76.*

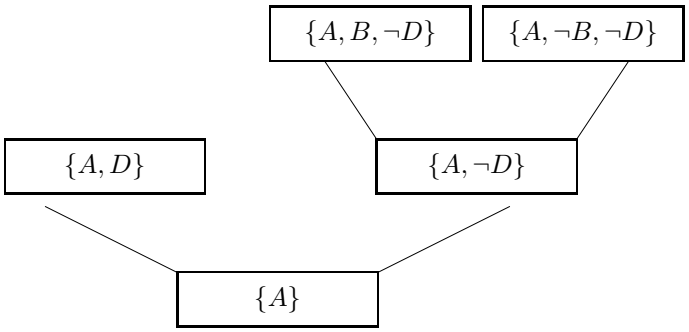


Figure 1.76: The second derivation tree

Exercise 1.8.16 *Apply the Sequence-to-tree Algorithm to get a derivation tree for the analysis below.*

$C_1 = \{\neg A\}$ *clause from* S
$C_2 = \{A, B, C\}$ *clause from* S
$C_3 = \{A, B, D\}$ *clause from* S
$C_4 = \{\neg B, D\}$ *clause from* S
$C_5 = \{A, \neg D\}$ *clause from* S
$C_6 = \{B, C\}$ *from* C_2, C_1 *on* A
$C_7 = \{B, D\}$ *from* C_3, C_1 *on* A
$C_8 = \{\neg D\}$ *from* C_5, C_1 *on* A
$C_9 = \{A, D\}$ *from* C_3, C_4 *on* B
$C_{10} = \{B\}$ *from* C_7, C_8 *on* D
$C_{11} = \{D\}$ *from* C_{10}, C_4 *on* B
$C_{12} = \Box$ *from* C_{11}, C_8 *on* D.

Exercise 1.8.17 *Give an example that shows that the trees-to-sequence algorithm does not always transforms a minimal derivation tree into a minimal derivation sequence.*

Exercise 1.8.18 *The trees-to-sequence algorithm does not always transform minimal derivation trees into minimal derivation sequences. Give a condition that guarantees that all minimal derivation trees satisfying it will generate minimal sequences.*

Exercise 1.8.19 *Let* $C_1, C_2, \ldots C_n = \Box$ *be a minimal derivation sequence. Show that*

1. *None of the clauses* $C_1 \ldots C_{n-1}$ *is a tautology.*
2. *There there are no indices* $0 \leq i < j < n$ *such that* $C_i \subseteq C_j$.

Exercise 1.8.20 *Prove that a set of formulas* S *is unsatisfiable iff there is a formula* F *such that* $S \models F$ *and* $S \models \neg F$.

Exercise 1.8.21 *Let* $\mathcal{A}$ *be a truth assignments and* K *be the set of all atoms modeled by* $\mathcal{A}$. *Characterize the set of all clauses modeled by* $\mathcal{A}$ *in terms of* K.

Exercise 1.8.22 *Find a model of S when $S_{P=0} = \phi$.*

Exercise 1.8.23 *Prove Koenig's Infinity Lemma: If the tree t has an infinite number of nodes, and each node has a finite number of children, then t has an infinite path.*

Exercise 1.8.24 *Let S be an infinite set of formulas. Show that F is a consequence of S iff F is a consequence of a finite subset of S.*

Exercise 1.8.25 *Let $T = \{\{P_1, P_3, \neg P_3\}, \{\neg P_1\}, \{P_1, P_3\}, \{P_1, P_2, \neg P_3\}, \{P_1, \neg P_2, \neg P_3\}\}$. Construct the sets $T_{P_3=1}$ and $T_{P_3=0}$.*

Exercise 1.8.26 *Let S and T be two unsatisfiable sets of clauses and Q be an atom such that $\{Q, \neg Q\} \cap S = \{Q, \neg Q\} \cap T = \phi$. We define the set $R = \{C \cup \{Q\} | C \in S\} \cup \{C \cup \neg\{Q\} | C \in T\}$, i.e. we add Q to all clauses of S and $\neg Q$ to all clauses of T. Show that R is unsatisfiable.*

Exercise 1.8.27 (Nerode and Shore) *Show that the set of unsatisfiable clauses is the set U inductively defined by the rules*

1. *if $\square \in S$ then $S \in U$, and*
2. *if $S_{P=0} \in U$ and $S_{P=1} \in U$, then $S \in U$.*

Do not use The Compactness Theorem.

Exercise 1.8.28 *Use Exercise 1.8.27 to prove The Compactness Theorem.*

Exercise 1.8.29 *Let S be a finitely satisfiable set, i.e. every finite subset of S is satisfiable. Let F be any formula. Show that at least one of the sets $S \cup \{F\}$, $S \cup \{\neg S\}$ is finitely satisfiable.*

Exercise 1.8.30 *Use the preceding exercise to give another proof of The Compactness Theorem.*

Exercise 1.8.31 *Show that the statement below is not always true.*

If S is minimally unsatisfiable and L is a literal in S, then $S_{L=1}$ and $S_{L=0}$ are minimally unsatisfiable.

1.9 Restrictions of Resolution

This section shows that Algorithm 1.74 from Section 1.8 is inefficient and offers several methods for speeding it up. These methods are strategies and restrictions of resolution. We discuss the P-resolution, the N-resolution, the subsumption, and the unit-resolution restrictions. We show that the first three are resolution complete, while the fourth is complete only for the Horn clauses.

Let us give an example that shows how inefficient Algorithm 1.74 is. Let $S_1 = \{\{P_0\}, \{\neg P_0, P_1\}, \{\neg P_1\}, \{P_2\}, \{\neg P_2, P_3\}, \{\neg P_3, P_4\}, \{\neg P_4, P_5\}, \{\neg P_5, P_6\}, \{\neg P_6, P_7\}, \{\neg P_7, P_8\}\}$.

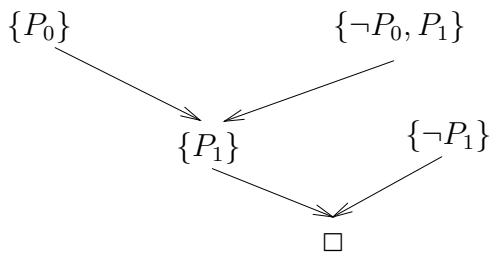


Figure 1.77: A derivation of $\square$ from S_1

First, the algorithm computes $Res^1[S_1] = S_1 \cup \{\{P_1\}, \{\neg P_0\}, \{P_3\}, \{\neg P_2, P_4\}, \{\neg P_3, P_5\}, \{\neg P_4, P_6\}, \{\neg P_5, P_7\}, \{\neg P_6, P_8\}\}$.

Since $Res^1[S_1] \neq S_1$ and $\square \notin S_1$, the algorithm computes $Res^2[S_1]$. We get the set

$Res^1[S_1] \cup \{\square, \{P_4\}, \{P_5\}, \{\neg P_2, P_5\}, \{\neg P_2, P_6\}, \{\neg P_3, P_6\}, \{\neg P_3, P_7\}, \{\neg P_4, P_7\}, \{\neg P_4, P_8\}, \{\neg P_5, P_8\}\}$.

This time the algorithm finds $\square$ and stops. Let us count the clauses in $Res^2[S_1] - S_1 = \{\{P_1\}, \{\neg P_0\}, \{P_3\}, \{\neg P_2, P_4\}, \{\neg P_3, P_5\}, \{\neg P_4, P_6\}, \{\neg P_5, P_7\}, \{\neg P_6, P_8\}, \square, \{P_4\}, \{P_5\}, \{\neg P_2, P_5\}, \{\neg P_2, P_6\}, \{\neg P_3, P_6\}, \{\neg P_3, P_7\}, \{\neg P_4, P_7\}, \{\neg P_4, P_8\}, \{\neg P_5, P_8\}\}$. We get 17 new clauses. However, Figure 1.77 shows that we needed only 2 extra clauses, $\{P_1\}$ and $\square$.

So, Algorithm 1.74 computed 15 extra clauses. Let us now add the clauses $\{\neg P_8, P_9\}, \{\neg P_9, P_{10}\}, \ldots, \{\neg P_{n-1}, P_n\}$, to S_1, where $n = 1000$. The minimal derivation tree is the same, but $Res^1[S]$ will contain all clauses $\{\neg P_i, P_{i+2}\}$ where i is any number between 2 and 998. The set $Res^2[S_1]$ would also contain all clauses $\{\neg P_j, P_{j+3}\}$ with $2 \leq j \leq 997$ and $\{\neg P_k, P_{k+4}\}$ with $2 \leq k \leq 996$. So, we will have to compute more than 3000 useless clauses.

There are two main reasons for the inefficiency of Algorithm 1.74.

1. First, the algorithm computes $Res^*[S_1]$, instead of concentrating on a small unsatisfiable subset of S_1. Ideally, we would like to find a subset T_1 that is *minimally unsatisfiable*, i.e. an unsatisfiable set whose proper subsets[11] are satisfiable. Then, the algorithm can look for the empty clause in $Res^*[T_1]$, instead of $Res^*[S_1]$. The set S_1 from the example above has such a subset, $T_1 = \{\{P_0\}, \{\neg P_0, P_1\}, \{\neg P_1\}\}$.

2. The second problem is that, even if T_1 is minimally unsatisfiable, $Res^*[T_1]$ is not a minimal derivation of $\square$ from T_1. $T = \{\{A, B\}, \{\neg A, B\}, \{A, \neg B\}, \{\neg A, \neg B\}\}$ is a minimally unsatisfiable set with resolvents $Res^*[T] = T \cup \{\square, \{A\}, \{B\}, \{\neg A\}, \{\neg B\}, \{A, \neg A\}, \{B, \neg B\}\}$. However,

$\{A, B\}$,$\{A, \neg B\}$,$\{\neg A, B\}$,$\{\neg A, \neg B\}$,$\{A\}$,$\{\neg A\}$, $\square$

is a derivation of $\square$ from T that uses only 7 of the 11 clauses of T. The difference may not seem much, but let us remember that T has only 2 atoms. The difference between the size of $Res^*[T]$ and the length of the minimal sequence increases exponentially in the number of atoms. (Exercise 1.9.4).

[11] A proper subset of A is a subset that is different from ϕ and the set itself.

So, our purpose is to improve Algorithm 1.74 to generate shorter derivations of $\Box$. We have two choices.

1. We can give criterias for selecting the two clauses that are going to be unified, called the *given clauses*.

2. We can restrict the set of given clauses. For example, we may require that one of the parents must contain only positive literals (i.e. atoms).

The first choice produces *strategies of resolution*. In general, they are heuristics that speed up the resolution. The term *heuristics* describes methods that serve to solve problems, but are not guaranteed to work in all cases. For example, a heuristic might require that one of the parent clauses be a literal. While intuitively this argument makes sense, there are cases where this strategy may cause more harm than good. (Exercise 1.9.5).

The second choice produces *restrictions of resolution* and this is the main thrust of this section. We are particularly interested in methods that cut down the number of generated clauses while allowing the program to find the box.

Definition 1.9.1 (completeness of a restriction of resolution) *A restriction of resolution is complete when every unsatisfiable set of clauses has a derivation of the box that obeys the restriction.*

We will show that the both the P and the N-resolution defined below are complete.

Definition 1.9.2 (positive and negative clauses) *A non-empty clause is positive if all its literals are positive, i.e. they are atoms.*

A non-empty clause is negative if all its literals are negative, i.e. they are complements of atoms.

The clauses $\{A\}$ and $\{B, C\}$ are positive, $\{\neg A\}$ and $\{\neg A, \neg B, \neg C\}$ are negative, $\Box$ and $\{A, \neg B\}$ are neither.

Definition 1.9.3 (P-resolution, N-resolution) *The P-resolution requires every resolvent to have as parent a positive clause.*

The N-resolution requires every resolvent to have as parent a negative clause.

Let us show that the clause set $S = \{\{A, B\}, \{A, \neg B\}, \{\neg A, B\}, \{\neg A, \neg B\}\}$ is unsatisfiable using only P-resolution. First let us look at a derivation of $\Box$ from S that uses unrestricted resolution.

1. $\{A, B\}$ clause in S
2. $\{A, \neg B\}$ clause in S
3. $\{A\}$ from 1, 2 on B
4. $\{\neg A, B\}$ clause in S
5. $\{\neg A, \neg B\}$ clause in S
6. $\{\neg A\}$ from 4,5 on B
7. $\Box$ from 3,6 on A

This sequence is not a P-derivation because clause 6 does not have a positive parent. Now let us construct a P-derivation of $\Box$ from S. Since S contains only

one positive clause, $\{A, B\}$, we unify it with the two clauses, $\{A, \neg B\}$ and $\{\neg A, B\}$. We get the derivation below.

*1. $\{A, B\}$ clause in S
2. $\{A, \neg B\}$ clause in S
*3. $\{A\}$ from 1, 2 on B
4. $\{\neg A, B\}$ clause in S
*5. $\{B\}$ from 1, 4 on A
6. $\{\neg A, \neg B\}$ clause in S
7. $\{\neg B\}$ from 3, 6 on A
8. $\square$ from 5, 7 on B

In this sequence we marked the positive clauses with *. Now, all resolvents have a positive parent: clauses 3 and 5 are obtained from clause 1, 3 is the parent of 7, and 5 is the parent of 8. So, we got a P-derivation of $\square$. At this point one might ask: How is this restriction an improvement, when the second derivation is longer than the first?

The answer is that we are comparing minimal derivations, and seldom do we get minimal derivations. Let us apply the unrestricted derivation to the set S_1, shown at the beginning of the section, by first listing the clauses of S_1, followed by the clauses of $Res^1[S_1] - S_1$, and, after that, by the clauses of $Res^2[S_1] - Res^1[S_1]$, stopping at the box. We get the sequence

$C_1 = \{P_0\}, C_2 = \{\neg P_0, P_1\}, C_3 = \{\neg P_1\}, C_4 = \{P_2\}, C_5 = \{\neg P_2, P_3\}, C_6 = \{\neg P_3, P_4\}, C_7 = \{\neg P_4, P_5\}, C_8 = \{\neg P_5, P_6\}, C_9 = \{\neg P_6, P_7\}, C_{10} = \{\neg P_7, P_8\}, \quad C_{11} = \{P_1\}, C_{12} = \{\neg P_0\}, C_{13} = \{P_3\}, C_{14} = \{\neg P_2, P_4\}, C_{15} = \{\neg P_3, P_5\}, C_{16} = \{\neg P_4, P_6\}, C_{17} = \{\neg P_5, P_7\}, C_{17} = \{\neg P_6, P_8\}, C_{18} = \square$.

Now, let us apply the same method, but using the P-resolution. First, we list the clauses of S_1. We apply the P-resolution to all clauses of S_1 and write their resolvents. Next, we apply the P-resolution to the sequence obtained so far, and we append the resolvents at the end of the sequence. We stop as soon as we get the box. We derive the sequence

$D_1 = \{P_0\}, D_2 = \{\neg P_0, P_1\}, D_3 = \{\neg P_1\}, D_4 = \{P_2\}, D_5 = \{\neg P_2, P_3\}, D_6 = \{\neg P_3, P_4\}, D_7 = \{\neg P_4, P_5\}, D_8 = \{\neg P_5, P_6\}, D_9 = \{\neg P_6, P_7\}, D_{10} = \{\neg P_7, P_8\}, \quad D_{11} = \{P_1\}, D_{12} = \{P_3\}, D_{13} = \square$.

This example shows that, in some cases, the P-restriction generates shorter sequences than the unrestricted resolution.

Theorem 1.9.4 (The Completeness of the P-resolution) *The P-resolution is complete.*

Proof:

The proof of this theorem is similar to the proof of The Resolution Theorem. We will prove that

(*) If S is an unsatisfiable set of clauses with n atoms, then there is a P-derivation tree of $\square$ from S.

The proof is by induction on n.

Basis: $n = 0$. Then S must contain $\square$.

Inductive Step: Assume that (*) is true for all S with n or fewer atoms. Let T be an unsatisfiable set of clauses that has $n + 1$ atoms. Let P be an atom

Figure 1.78: The P-trees t_0 and t_1

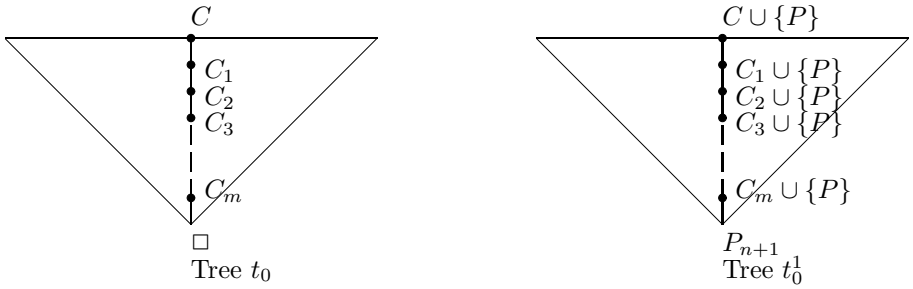


Figure 1.79: The trees t_0 and t_0^1

in T and $T_{P=0}$ and $T_{P=1}$ be the sets from Definition 1.8.26. We know, from Lemma 1.8.28, that both sets are unsatisfiable. Moreover, they have at most n atoms since P does not occur in either one. So, we can apply the induction hypothesis to $T_{P=0}$ and $T_{P=1}$. We get the P-resolution trees, t_0 and t_1, of Figure 1.78.

Any leaf of $T_{P=0}$ is either a T clause or is obtained from a T clause by deleting the atom P. Also, for any leaf C in $T_{P=1}$, either C or $C \cup \{\neg P\}$ is in T. If all leaves of t_0 or all leaves of t_1 are in T, then we have a P-derivation of $\square$ from T.

But what happens when both trees have labels that are not in T? Then, we take a leaf C of t_0 that is not in T and add P to the leaf and to all its ancestors including the root, as shown in Figure 1.79.

We notice that this addition of labels transforms the P-resolution tree t_0 into the P-resolution tree t_0^1. If some of t_0^1 leaves are not in T, we repeat the procedure, until all leaves are in T. We obtain the P-resolution tree t_0' from Figure 1.80 whose leaves are in T. Since t_1 is not a tree in T, some of its leaves are missing the literal $\neg P$. Figure 1.81 shows that we cannot add $\neg P$ to the leaves of t_1 without endangering the P-resolution. At the left we have a P-tree. The tree at the right is obtained by adding $\neg C$ to the leaf $\{A, B\}$ and to its parent. This resolution tree is no longer a P-tree because the root has no

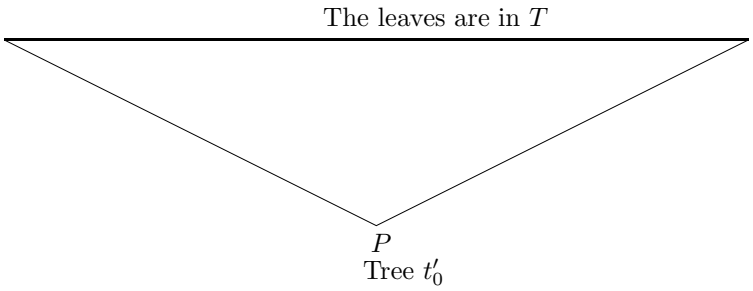


Figure 1.80: The P-tree $t_0{}'$

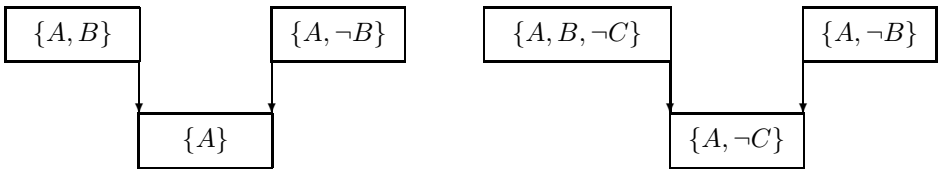


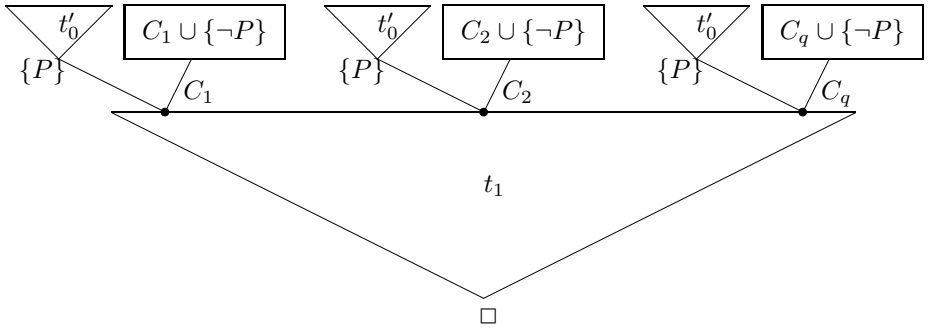
Figure 1.81: Adding $\neg C$ destroys the P-resolution

positive parent.

So, we need to find another way of converting t_1 into a P-tree with leaves in T. But we already have the P-tree t_0' with root P and leaves in T. So, we build a copy of t_0' for each leaf $C_1, \ldots, C_q$ of t_1 that is not in T. Then we unify the clauses $C_1 \cup \{\neg P\}$, $C_2 \cup \{\neg P\}$,...,$C_q \cup \{\neg P\}$ of T with the root of a copy. We get the P-resolution tree from Figure 1.82. This tree is a P-tree because t_0' and t_1 are P-trees and the intermediate steps of unifying $\{P\}$ with the clauses $C_1 \cup \{\neg P\}$,..., $C_q \cup \{\neg P\}$ are also P-resolutions. **Q.E.D.**

Example 1.9.5 Let us illustrate the inductive step of the proof of Theorem 1.9.4. Let T be the set $\{\{A, B, C\}, \{A, \neg B, C\}, \{\neg A, C\}, \{B, \neg C\}, \{\neg B, \neg C\}\}$ and let us eliminate the atom C. We get the sets $T_{C=0} = \{\{A, B\}, \{A, \neg B\}, \{\neg A\}\}$ and $T_{C=1} = \{\{B\}, \{\neg B\}\}$. By the induction hypothesis we get the P-trees from Figure 1.83. Neither t_0 nor t_1 are resolution trees in T because some of their leaves are not in T. So we add C to all leaves of t_0 that are not in T and to all their ancestors. We get the positive resolution tree t_0' from Figure 1.84. Now, we need to take care of the clauses of t_1 that are not in T, i.e. $\{B\}$ and $\{\neg B\}$. These clauses are missing the literal $\neg C$. For these clauses we need to unify $\{B, \neg C\}$ and $\{B, \neg C\}$ with the root $\{C\}$ of t_0'. We get the P-tree from Figure 1.85.

We marked the positive clauses with an asterisk, so we can check that the tree is indeed a P-tree.



The tree t

Figure 1.82: A P-derivation of $\square$ from T

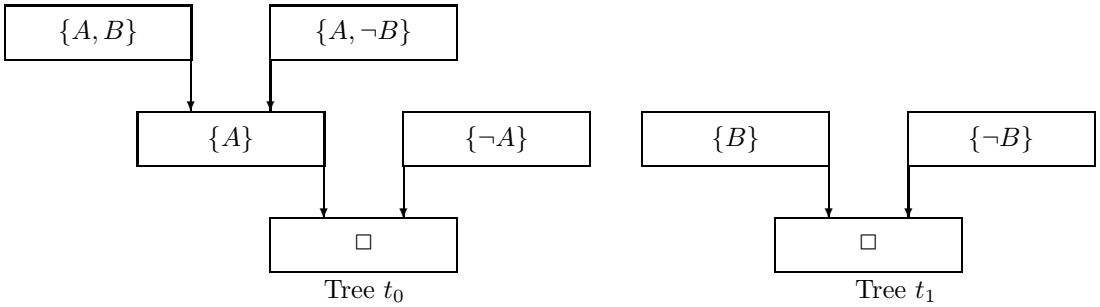


Figure 1.83: The trees t_0 and t_1 of Example 1.9.5

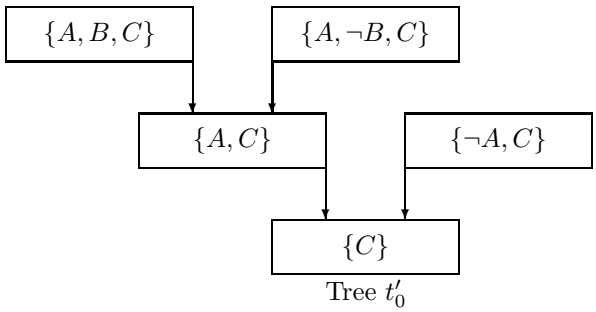


Figure 1.84: The tree t'_0 of Example 1.9.5

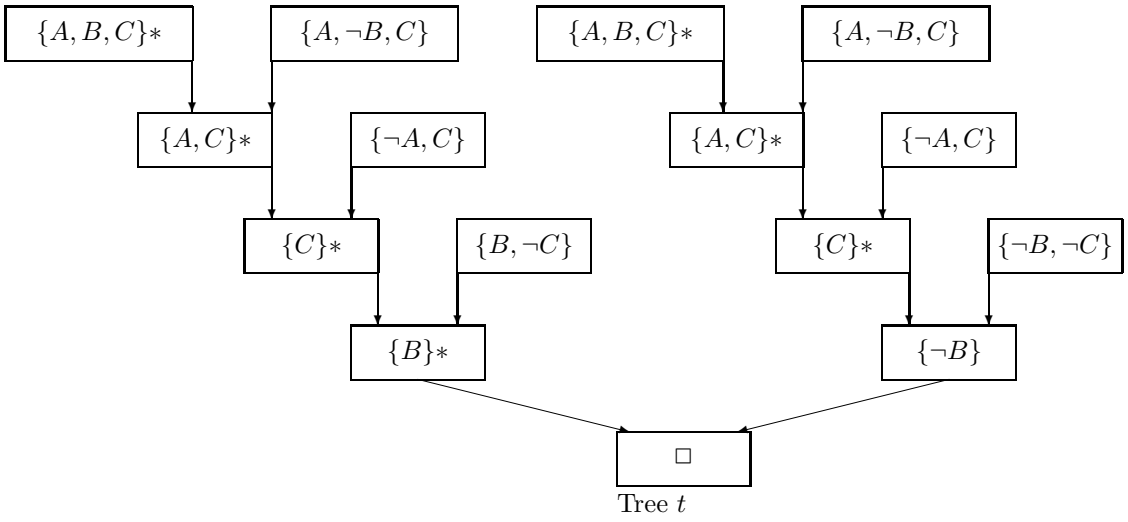


Figure 1.85: The P-tree from Example 1.9.5

We can also prove the completeness of *negative resolution*.

Theorem 1.9.6 (The Completeness of the N-Resolution) *The N-resolution is complete.*

Proof: It is similar to the proof of the completeness of the P-resolution, except for the construction of t from t'_0 and t_1. Instead of adding P to the leaves of t_0, we add $\neg P$ to the leaves of t_1 and get a N-resolution tree of $\{\neg P\}$ from T. Then we use $\{\neg P\}$ to generate the clauses of t_0 that are not T.**Q.E.D.**

Definition 1.9.7 (T-resolution) *In T-resolution no resolvent has a tautology as parent.*

Theorem 1.9.8 *The T-resolution is complete.*

Proof: The proof is almost identical to the proof of the completeness of the unrestricted resolution (Theorem 1.8.29). We will not go through the details of the proof; we will just pinpoint the differences.

$\Longleftarrow$: Assume that there is a T-resolution tree of $\Box$ from S. This tree is a resolution tree, so S is unsatisfiable by The Resolution Theorem.

$\Longrightarrow$: We need to show that for every unsatisfiable set S we can find a T-derivation tree of $\Box$ from S. Just like in the proof of The Resolution Theorem, we restrict ourselves to the finite sets and we do the proof by induction on the number of atoms in S.

Basis: $n = 0$. Then $= \{\Box\}$, and $\Box$ is T-tree.

Inductive Step: Assume that the theorem holds for n. Let S be an unsatisfiable set with $n + 1$ atoms. Just like in The Resolution Theorem, we form the

sets $S_{P=0}$ and $S_{P=1}$ and the induction hypothesis gives us the T-trees t_0 and t_1. If both trees have leaves that are not in S, we form the trees $t_0^{'}$ and $t_1^{'}$, just like we did in The Resolution Theorem. We will show that these trees are also T-trees. The clauses of $t_0^{'}$ are in t_0 or are obtained from a t_0-clause by adding P. Since t_0 has no tautologies and P is not a member of any t_0-clause, $t_0^{'}$ has no tautologies. A similar argument establishes that $t_1^{'}$ is a T-tree. Then the tree t obtained by unifying the roots of $t_0^{'}$ and $t_1^{'}$ is a T-tree. **Q.E.D**

The completeness of the T-resolution tells us that in any derivation of the box, we can discard the tautologies.

We can combine the T-restriction with other restrictions. In the P+T-resolution every resolvent has a positive parent and no parent is a tautology. In the N+T-resolution every resolvent has a negative parent, and none of the parents is a tautology.

Theorem 1.9.9 (The Completeness of the P+T-resolution) *The P+T-resolution is complete.*

Proof: The proof is similar to the completeness of the P-resolution. We just have to show that whenever the two trees from Figure 1.78 have no tautologies, the tree from Figure 1.82 is also tautology free. So, let us assume that the trees t_0 and t_1 have no tautologies. The tree $t_0^{'}$ from Figure 1.80 is obtained by adding P to some clauses of t_0. Since t_0 has no tautologies, and $\neg P$ does not occur in any of the t_0 clauses, $t_0^{'}$ is also tautology free. The clauses $C_1, \ldots, C_q$ of Figure 1.82 are in t_1, so they are not tautologies and do not contain P. So, the clauses $C_1 \cup \{\neg P\}, \ldots, C_q \cup \{\neg P\}$ are not tautologies. Then, t is tautology free. **Q.E.D.**

Another useful restriction is the *subsumption*.

Definition 1.9.10 (subsumption) *We say that a clause C_1 subsumes a clause C_2 if $C_1 \neq C_2$ and $C_1 \subseteq C_2$.*

For example, $C_1 = \{A, \neg C, E\}$ subsumes $C_2 = \{A, B, \neg C, \neg D, E\}$, because C_1 is a subset of C_2 and $C_1 \neq C_2$.

Definition 1.9.11 *We call the tree from Figure 1.86 a derivation tree of C_n with base clause C and side clauses $B_1, B_2, \ldots, B_n$.*

Lemma 1.9.12 (The Subsumption Lemma) *Let t be a derivation tree of $\Box$ with base clause C and side clauses $B_1, B_2 \ldots B_n$, and D be a clause that subsumes C. Then there is a derivation tree t' of $\Box$ with base clause D and side clauses $E_1, \ldots E_k$, $k \leq n$, such that $E_1, \ldots E_k$ is a subsequence of $B_1, B_2 \ldots B_n$. Moreover, if t is a P-tree, so is t'; if t is an N-tree, so is t'.*

Proof: Let us assume that the tree t from Figure 1.86 has $C_n = \Box$. Let $C_0 = C$, and L_i be the literal of C_i that is unified with B_{i+1}. Now we define a sequence of clauses $D_0, D_1 \ldots, D_n$.

$D_0 = D$

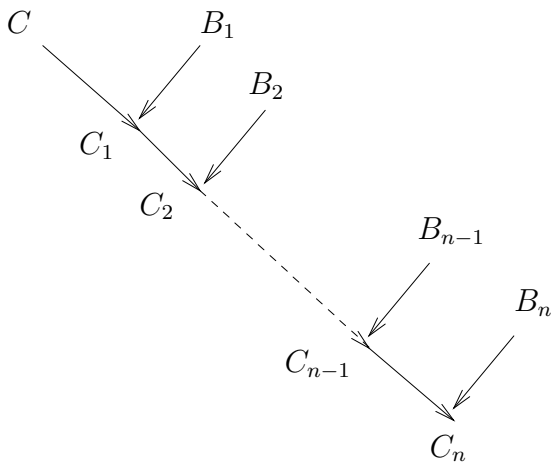


Figure 1.86: The side clauses of a derivation tree

$$D_{i+1} = \begin{cases} \textit{the resolvent of } D_i \textit{ and } B_{i+1} & \text{if } L_i \in D_i \\ D_i & \text{if } L_i \notin D_i \end{cases}$$

We will show, by induction on i, that for all $0 \leq i \leq n$, D_i subsumes C_i.
Basis: $n = 0$. $D_0 = D$, $C_0 = C$, and we assumed that D subsumes C.
Inductive Step: Assume that D_i subsumes C_i. If L_i is in D_i, then
$D_{i+1} = (D_i - \{L_i\}) \cup (B_{i+1} - \{\bar{L}_i\})$
$\subseteq (C_i - \{L_i\}) \cup (B_{i+1} - \{\bar{L}_i\})$ because $D_i \subseteq C_i$
$= C_{i+1}$ by the construction of C_{i+1}
If $L_i \notin D_i$ then
$C_{i+1} = (C_i - \{L_i\}) \cup (B_{i+1} - \{\bar{L}_i\})$
$\supseteq (D_i - \{L_i\}) \cup (B_{i+1} - \{\bar{L}_i\})$ because $D_i \subseteq C_i$
$= D_i \cup (B_i - \{\bar{L}_i\})$ because $L_i \notin D_i$, $D_i - \{L_i\} = D_i$
$\supseteq D_i$.
$= D_{i+1}$ from the construction of D_{i+1}
In both cases $D_{i+1} \subseteq C_{i+1}$, so we conclude the proof of $D_i \subseteq C_i$.

The inclusion $D_i \subseteq C_i$ tells us that $D_n = \square$. From the sequence $D_0 = D, D_1, D_2, \ldots, D_n = \square$ we delete all clauses that are equal to the preceding clause. We get the sequence $D_0 = D, D_{i_1}, D_{i_2}, \ldots D_{i_k}$, where $1 \leq i_1 < i_2 < \ldots i_k \leq n$. Each of these clause is unified with B_{i_k}, while the deleted clauses are not. Now let t' be the tree from Figure 1.87. This is a derivation tree of $\square$ from D with side clauses $E_1 = B_{i_1}, \ldots, E_k = B_{i_k}$.

Now, let us assume that the tree t from Figure 1.86 is a P-tree. Then, every C_i, $1 \leq i \leq n$, has a positive parent. The clause D_{i_j} of t' has parents B_{i_j} and $D_{i_{(j-1)}}$, as shown in Figure 1.88. Since we removed all duplicate D-clauses, $D_{i_{(j-1)}} = D_{(i_j - 1)}$. Now, let us look at the clause C_{i_j} of t. If the positive parent

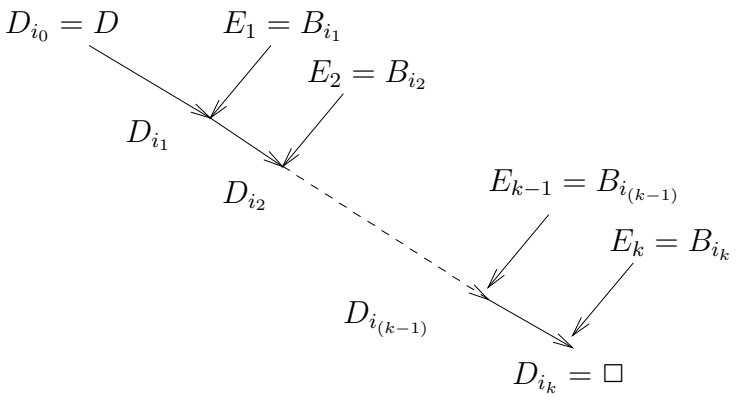


Figure 1.87: The tree t'

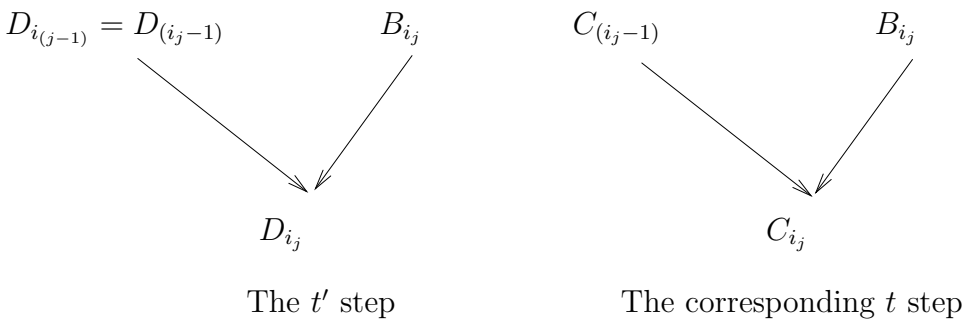


Figure 1.88: A step of t'

is B_{i_j}, then B_{i_j} is a positive parent of D_{i_j}. If the positive parent is $C_{(i_j-1)}$ then so is $D_{i_{(j-1)}} = D_{(i_j-1)}$, because $D_{(i_j-1)} \subseteq C_{(i_j-1)}$. So, t' is a P-tree. In a similar way we prove that t' is an N-tree whenever t is. We also note that if t has no tautologies, neither does t'. **Q.E.D.**

Let us illustrate the construction of t' from Figure 1.87 with an example.

Example 1.9.13 Let t be the tree from Figure 1.89 and $D = \{B, C\}$. The clause D subsumes $C = C_0$ because $\{B, C\} \subseteq \{A, B, C\}$. C_0 is unified with B_1 on $L_0 = A$, C_1 and B_2 on $L_1 = \neg D$, C_2 and B_3 on $L_2 = B$, C_3 and B_4 on $L_3 = C$, C_4 and B_5 on $L_4 = F$. Now let us form the D_i clauses. $D_0 = D = \{B, C\}$. Since $L_0 = A$ is not in D_0, $D_1 = D_0$. The literal L_1 is not in D_1, so $D_2 = D_1 = D_0 = \{B, C\}$. The next literal, $L_2 = B$, is in D_2, so D_3 is the resolvent of B_3 and D_2 on B. We get $D_3 = \{C, F\}$. The next literal, $L_3 = C$ is in D_3, so D_4 is the resolvent $\{F\}$ of D_3 and B_4. Finally, $L_4 = F$ is in D_4, so D_5 is the resolvent $\Box$ of D_4 and B_5. We get the sequence $D_0 = \{B, C\}, D_1 = \{B, C\}, D_2 = \{B, C\}, D_3 = \{C, F\}, D_4 = \{F\}, D_5 = \Box$.

We delete all clauses that equal to the preceding clause and get the sequence

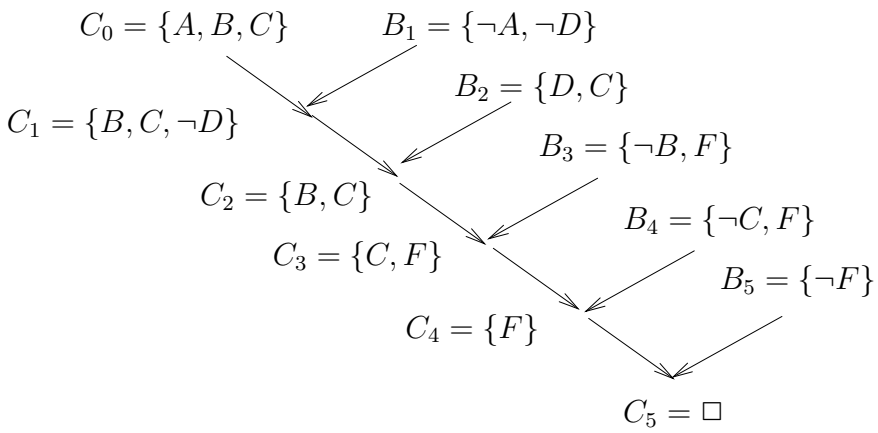


Figure 1.89: The tree t from Example 1.9.13

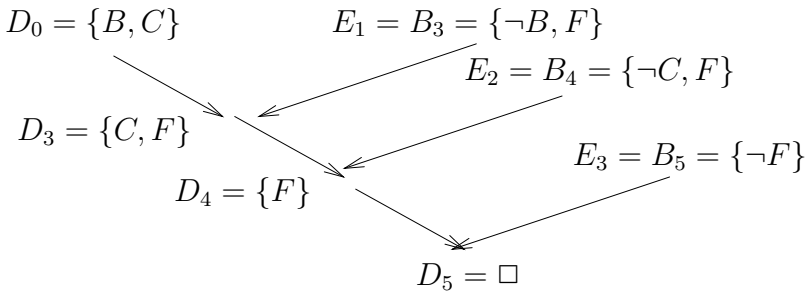


Figure 1.90: The tree t' from Example 1.9.13

$D_0 = \{B, C\}, D_3 = \{C, F\}, D_4 = \{F\}, D_5 = \square$. Now we take the parents $E_1 = B_3$, $E_2 = B_4$, $E_3 = B_5$ of D_3, D_4 and D_5 and get the tree t' from Figure 1.90. Since t is a P-tree, so is t'.

Lemma 1.9.14 *Let S be a set of unsatisfiable clauses that does not contain $\square$. Then $Res[S] - S$ contains a clause that is not subsumed by any clause in S.*

Proof: We assume that the assertion is false. So, let S be a set that does not include $\square$. Then, there are derivation trees of $\square$ from S. Let t be one of these trees that has *minimal height*. Since $\square$ is not in S, t has height $n > 0$. Now, let us assume that all clauses of $Res[S] - S$ are subsumed by clauses in S. The left tree of Figure 1.91 shows a path of length n in t. By our assumption, there is a clause D in S that subsumes or is equal to C_1. We apply The Subsumption Lemma and get the tree t' shown to the right. The paths from the root to C_0 and to C have been replaced by the shorter path from the root to D. Now let us see what happens to the paths of length n of t that connect the root with a leaf of B_i. The path disappears if B_i is not in the sequence $B_{i_1}, \ldots, B_{i_k}$. If

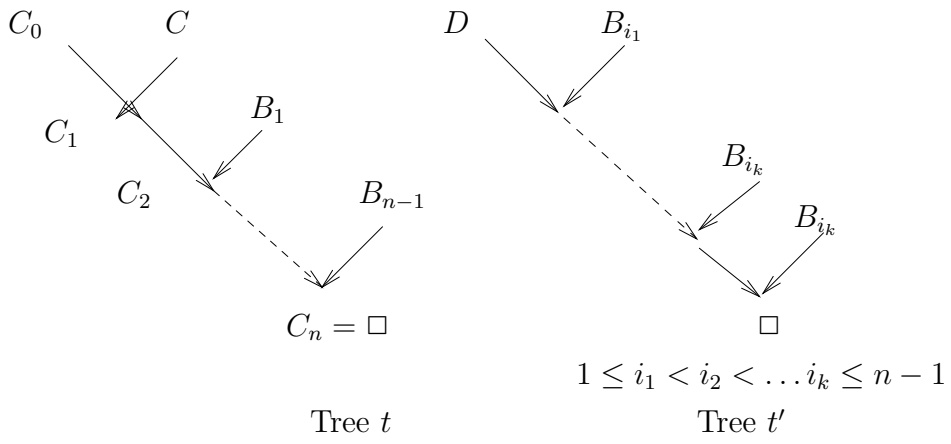


Figure 1.91: The trees from Lemma 1.9.14

B_i is in the sequence, the path has the same length in both tree t and t'. Now we can apply The Subsumption Lemma again to a path of length n of t', and continue this way until all paths of length n are reduced. We get a tree t^{red} of $\Box$ from S that has height less than n. But this contradicts the assumption that t has minimal length. **Q.E.D.**

The next lemma says that we can delete the subsumed clauses.

Lemma 1.9.15 *Let C_1 and C_2 be two clauses of S. If C_1 subsumes C_2, then $S \equiv S - \{C_2\}$.*

Proof: Let $\mathcal{A}$ be a truth assignment. We will show that $\models_{\mathcal{A}} S$ iff $\models_{\mathcal{A}} (S - \{C_2\})$. If $\models_{\mathcal{A}} S$ then $\models_{\mathcal{A}} (S - \{C_2\})$ because $S - \{C_2\}$ is a subset of S. If $\models_{\mathcal{A}} S - \{C_2\}$ then $\models_{\mathcal{A}} C_1$. Since $C_1 \subseteq C_2$, $\models_{\mathcal{A}} C_2$. Then $\models_{\mathcal{A}} ((S - \{C_2\}) \cup \{C_2\})$,i.e. $\models_{\mathcal{A}} S$. **Q.E.D.**

Now we can define the subsumption restriction.

Definition 1.9.16 (subsumption free derivation) *A derivation sequence C_1, $C_2, \ldots, C_n = \Box$ is subsumption free if no clause is equal to or subsumed by a preceding clause.*

Example 1.9.17 The sequence

$C_1 = \{A\}, C_2 = \{A, \neg B\}, C_3 = \{\neg A, \neg B\}, C_4 = \{\neg B\}, C_4 = \{B\}, C_5 = \Box$

is not subsumption free because C_2 is subsumed by C_1. The sequence

$C_1 = \{A, B\}, C_2 = \{A, \neg B\}, C_3 = \{A\}, C_4 = \{A\}, C_5 = \{\neg A\}, C_6 = \Box$

is not subsumption free because $C_4 = C_3$. The sequence

$C_1 = \{A, B\}, C_2 = \{A, \neg B\}, C_3 = \{A\}, C_4 = \{\neg A, B\}, C_5 = \{\neg A, \neg B\}, C_6 = \{\neg A\}, C_7 = \Box$

is subsumption free because no clause is equal to or subsumed by a preceding clause.

```
n = 0;
S_n = {C|C ∈ S and C is not subsumed by a clause in S};
list the clauses of S_n in any desired order;
loop
{
      if (□ ∈ S_n) then
            exit with success;
      else if (S_n has a resolvent C ∉ S_n that is not subsumed by a clause
        in S_n) then
      {
            write C;
            n = n + 1;
            S_n = S_{n-1} ∪ {C};
      }
      else
            exit with failure;
}
```

Figure 1.92: The Subsumption Algorithm

Theorem 1.9.18 (The Subsumption Theorem) *The Subsumption Algorithm is correct for finite sets, i.e. it terminates and produces the correct answer.*

Proof: First of all, we make the following observations about the algorithm.

1. The sequences S_n are subsumption free.
2. $S_n \equiv S$.
3. The algorithm keeps looping as long as S_n grows, i.e. a new clause is added to S_n.
4. The algorithm finishes with either success or failure. It terminates with success iff $\Box \in S_n$ and it ends with failure iff $Box \notin S_n$ and S_n stops growing.
5. $S_n \subseteq Res^*[S]$.

The get the first property from the construction of S_n, the 3rd and the 4th from the construction of the algorithm. We have the second property because $S_0 \equiv S$ by Lemma 1.9.15 and $S_0 \subseteq S_n$. The 5th property holds because S_n is a derivation sequence over S.

Now let us show that the algorithm terminates. The number of clauses that can be formed with the atoms of S is finite, so S_n cannot grow indefinitely. Since the loop continues only when S_n grows, the algorithm terminates.

Now let us show its correctness. We will show that the algorithm finishes with success iff S is unsatisfiable. Then, by the 4th observation, we conclude that it stops with failure iff S is satisfiable.

$\Longrightarrow$: Assume that the algorithm stops with success. Then there is some m such that $\Box \in S_m$. Since $S_m \subseteq Res^*[S]$, S is unsatisfiable by The Resolution Theorem.

$\Longleftarrow$: Assume that S is unsatisfiable. The algorithm terminates, so the loop exits with the sequence S_m. Let us assume that it terminates with failure. Then,

$n = 0$;
$S_n = \{C | C \in S_n$, C is not a tautology, and C is not subsumed by a clause in $S\}$;
list the clauses of S_n in any desired order;
loop
{
if ($\Box \in S_n$) then
exit with success;
else if (S_n has a resolvent $C \notin S_n$ that has a positive parent, is not a tautology, and is not subsumed by a clause in S_n) then
{
write C;
$n = n + 1$;
$S_n = \{D | D \in S_{n-1}$ and D is not subsumed by $C\} \cup \{C\}$;
}
else
exit with failure;
}

Figure 1.93: The P+T+Subsumption Algorithm

$\Box \notin S_m$ and all clauses in $Res[S_m] - S_m$ are subsumed by clauses in S_m. Since $S_m \equiv S_0$, it is unsatisfiable. So, S_m is unsatisfiable and does not contain the box. By Lemma 1.9.14, it contains a clause C that is not in S_m. But then, the algorithm does not stop with S_m; it adds C to S_m and continues. So, we got a contradiction. **Q.E.D.**

But how do we find the resolvent $C \in Res[S] - S$ that is not subsumed by any clause in S? The inefficient way is to generate $Res[S]$. A better way is to discard tautologies and all clauses that are subsumed by other clauses of S. Let S_0 be the reduced set. We compute all resolvents of S_0 that have a positive parent, and are not tautologies. We get the algorithm from Figure 1.93.

Theorem 1.9.19 *The P+T+Subsumption Algorithm is correct for finite inputs.*

Proof: First of all let us prove that the algorithm terminates. The sets S_n can both grow and shrink, so we cannot use the argument employed in the termination of The Subsumption Algorithm. Instead we will show that the algorithm generates a subsumption-free, tautology free derivation where every resolvent has a positive parent.

Lemma 1.9.20 *If $m < n$, every clause $D \in S_m$ belongs to S_n or is subsumed by a clause in S_n.*

Proof: We prove it by induction on n.

Basis: $n = 0$. Since there are no whole numbers less that 0, the lemma is true by default.

Inductive Step: Assume that the lemma is true for n and let D be a clause in the set S_m, $m \leq n$. We have 2 cases.

Case 1: $m = n$. Then, from the construction of the set S_{n+1}, either D is in the set or is subsumed by the memeber C of the set.

Case 2: $m < n$. We apply the induction hypothesis and get that D is either in S_n or is subsumed by a clause E in S_n. Case 1 deals with $D \in S_n$, so let assume that D is subsumed by E. From Case 1, E is in S_{n+1} or is is subsumed by C. If E is in S_{n+1} then D is subsumed by a clause in S_{n+1}. If E is subsumed by C, then D is subsumed by C since the subsumption is transitive. Again, D is subsumed by a clause in S_{n+1}.

Since the two cases cover all clauses $D \in S_m$, we conclude the inductive step. **Q.E.D. lemma**

Lemma 1.9.21 *If $C \in (S_{n+1} - S_n)$ and $m \leq n$, then $C \notin S_m$ and C is not subsumed by any clause in S_m.*

Proof: Assume that for some $m \leq n$, there is a clause $D \in S_m$ such that $D \subseteq C$. If D is not in S_n, we apply Lemma 1.9.20 to D and S_n and get a clause $E \in S_n$ that subsumes D. Again, we find a clause of S_n that is equal to or subsumes C. Then C does not satisfy the second if statement, so $C \notin S_{n+1}$. Contradiction! **Q.E.D. lemma**

Lemma 1.9.22 *The derivation produced by the algorithm from Figure 1.93 is subsumption and tautology free and every resolvent has a positive parent.*

Proof: First of all, the output list contains only clauses from the sets S_n. None of these sets contain tautologies, so neither does the list. The printed clauses belong to S_0 or are the C clauses of the S_n's, $n > 0$. In the first case, the clauses are members of S ; in the second they have a positive parent from the construction of S_n.

Now, let us show that the list $C_1, C_2, \ldots, C_n, \ldots$, produced by the algorithm, has no pair of indices $i < j$ such that $C_i \subseteq C_j$. Assume that there are. Then either they are printed within the same write statement, or not.

Let us assume that both are written in the same print statement. Then they are both in S_0, because the other print statement outputs only one clause. But this is impossible, since no clause in S_0 subsumes another clause of S_0, and all clauses of S_0 are listed once.

Now, let us assume that C_i and C_j are printed in different steps. Then, there are indices $m < n$ such that $C_i \in S_m$ and $C_j \in S_n$. But this means that C_j either belongs to S_m or is subsumed by a clause in S_m, contradicting Lemma 1.9.21. So, the sequence $C_1, C_2, \ldots, C_n, \ldots$ is subsumption free. **Q.E.D. lemma**

Corollary 1.9.23 *The algorithm from Figure 1.93 terminates.*

Proof: The sequence $C_1, C_2, \ldots, C_n, \ldots$ produced by the algorithm has no duplications. Since the number of clauses that can be formed with the atoms of S is finite, the sequence is finite. **Q.E.D. corollary**

Now we are left with the correctness proof. First we give another version of Lemma 1.9.14.

Lemma 1.9.24 *If S is unsatisfiable and $\Box \notin S$, then there is a clause $C \in Res[S] - S$, that is not a tautology, has a positive parent, and is not subsumed by any clause in S.*

Proof: The proof is just like the proof of Lemma 1.9.14. The only difference is that the tree t comes from the completeness of the P+T-resolution, and not from The Resolution Theorem. The Subsumption Lemma does not introduce any tautologies and carries P-trees into P-trees, so t^{red} is also a P-tree with no tautologies. **Q.E.D. lemma**

We can also prove a stronger version of Lemma 1.9.15.

Lemma 1.9.25 *$S \equiv S - \{C | C$ is a tautology or C is subsumed by another clause in $S\}$.*

The proof of this lemma is left as exercise.

Lemma 1.9.26 *For all n, $S \equiv S_n$.*

Proof: By induction on n.

Basis: $S_0 \equiv S$ by Lemma 1.9.25.

Inductive Step: Assume that $S_n \equiv S$. Since C is a resolvent of S_n, $S_n \cup \{C\} \equiv S_n$ by Proposition 1.8.5. By Lemma 1.9.15 $S_{n+1} \equiv S_n \cup \{C\}$. Since $\equiv$ is transitive, $S_{n+1} \equiv S$. **Q.E.D. lemma**

Now let us show that the algorithm produces the correct answer.

The algorithm terminates, so let S_n be the last set computed by it. Assume that it stopped because $\Box \in S_n$. Then S_n is unsatisfiable. Lemma 1.9.26 tells us that $S \equiv S_n$, so S is also unsatisfiable.

Now assume that the algorithm terminates because the box is not in S_n and we cannot find a C to compute S_{n+1}. By Lemma 1.9.24, S_n is satisfiable, or $\Box \in S_n$. Since $\Box \notin S_n$, S_n is satisfiable. By Lemma 1.9.26, $S \equiv S_n$, so S is also satisfiable. **Q.E.D. theorem**

Example 1.9.27 Let us show that the set $S = \{\{A, B, C\}, \{\neg A, \neg B, \neg C\}, \{A, B, \neg C\}, \{A, \neg B, \neg C\}, \{\neg A, B\}, \{\neg B, C\}\}$ is unsatisfiable using

The P+T+Subsumption Algorithm.

None of the clauses of S is subsumed by another and the set has no tautologies, so $S_0 = S$. We list the clauses of S_0.

$C_1 = \{A, B, C\}, C_2 = \{\neg A, \neg B, \neg C\}, C_3 = \{A, B, \neg C\}, C_4 = \{A, \neg B, \neg C\}$, $C_5 = \{\neg A, B\}, C_6 = \{\neg B, C\}$.

Since $\Box$ is not in S_0, we look for a resolvent that has a positive parent, is not a tautology, and is not subsumed by any clause in S_0. The set has only one positive clause, C_1, so we compute its resolvents. The unification with C_2 produces tautologies, but the resolvent $C_7 = \{A, B\}$ of C_1 and C_3 satisfies the requirements. So, we write C_7 at the end of the list.

$C_1 = \{A, B, C\}, C_2 = \{\neg A, \neg B, \neg C\}, C_3 = \{A, B, \neg C\}, C_4 = \{A, \neg B, \neg C\},$

$C_5 = \{\neg A, B\}, C_6 = \{\neg B, C\}, C_7 = \{A, B\}$

We compute S_1 by adding C_7 to S_0 and deleting all clauses subsumed by C_7. So, we delete C_1 and C_3 to get $S_1 = \{C_2, C_4, C_5, C_6, C_7\}$.

Since $\Box$ is not in S_1, we unify the positive clause C_7 with the other clauses of the set and look for a resolvent that is not a tautology and is not subsumed by any clause in S_1. The first such clause is $C_8 = \{A, \neg C\}$, the child of C_7 and C_4. We write it.

$C_1 = \{A, B, C\}, C_2 = \{\neg A, \neg B, \neg C\}, C_3 = \{A, B, \neg C\}, C_4 = \{A, \neg B, \neg C\},$
$C_5 = \{\neg A, B\}, C_6 = \{\neg B, C\}, C_7 = \{A, B\}, C_8 = \{A, \neg C\}$

The new clause subsumes C_4, so we delete it and get $S_2 = \{C_2, C_5, C_6, C_7, C_8\}$. The box is still not in S_2, so we find the next clause that has a positive parent, is not a tautology, and is not subsumed by any clause in S_2. The only positive clause in S_2 is C_7, and the first resolvent that satisfies these conditions is the child $C_9 = \{B\}$ of C_7 and C_5. We write it at the end of the list.

$C_1 = \{A, B, C\}, C_2 = \{\neg A, \neg B, \neg C\}, C_3 = \{A, B, \neg C\}, C_4 = \{A, \neg B, \neg C\},$
$C_5 = \{\neg A, B\}, C_6 = \{\neg B, C\}, C_7 = \{A, B\}, C_8 = \{A, \neg C\}, C_9 = \{B\}$

The new clause subsumes C_5 and C_7, so $S_3 = \{C_2, C_6, C_8, C_9\}$. The box is not in S_3, so we find a resolvent of S_3 that meets the requirements. The only positive clause in S_3 is C_9, so we unify it with the other clauses of the set. The first acceptable resolvent is $C_{10} = \{\neg A, \neg C\}$, the child of C_9 and C_2. We write it at the end of the list.

$C_1 = \{A, B, C\}, C_2 = \{\neg A, \neg B, \neg C\}, C_3 = \{A, B, \neg C\}, C_4 = \{A, \neg B, \neg C\},$
$C_5 = \{\neg A, B\}, C_6 = \{\neg B, C\}, C_7 = \{A, B\}, C_8 = \{A, \neg C\}, C_9 = \{B\}, C_{10} = \{\neg A, \neg C\}$

The new clause subsumes C_2, so $S_4 = \{C_6, C_8, C_9, C_{10}\}$. The box is not in S_4 and we compute an admissible resolvent of S_4. The first one is $C_{11} = \{C\}$, the child of C_6 and C_9. We write it.

$C_1 = \{A, B, C\}, C_2 = \{\neg A, \neg B, \neg C\}, C_3 = \{A, B, \neg C\}, C_4 = \{A, \neg B, \neg C\},$
$C_5 = \{\neg A, B\}, C_6 = \{\neg B, C\}, C_7 = \{A, B\}, C_8 = \{A, \neg C\}, C_9 = \{B\},$
$C_{10} = \{\neg A, \neg C\}, C_{11} = \{C\}.$

The new clause subsumes C_6, so $S_5 = \{C_8, C_9, C_{10}, C_{11}\}$. The box is not in S_6, so we search for an acceptable resolvent of S_5. The first one is $C_{12} = \{A\}$, the child of C_8 and C_{11}. We write C_{12}, and so far the output is the one below.

$C_1 = \{A, B, C\}, C_2 = \{\neg A, \neg B, \neg C\}, C_3 = \{A, B, \neg C\}, C_4 = \{A, \neg B, \neg C\},$
$C_5 = \{\neg A, B\}, C_6 = \{\neg B, C\}, C_7 = \{A, B\}, C_8 = \{A, \neg C\}, C_9 = \{B\},$
$C_{10} = \{\neg A, \neg C\}, C_{11} = \{C\}, C_{12} = \{A\}$

The new clause subsumes C_8, so $S_6 = \{C_9, C_{10}, C_{11}, C_{12}\}$. The box is not in S_6, so we look for an admissible resolvent. The first one is $C_{13} = \{\neg C\}$, the child of C_{12} and C_{10}. We write C_{13}.

$C_1 = \{A, B, C\}, C_2 = \{\neg A, \neg B, \neg C\}, C_3 = \{A, B, \neg C\}, C_4 = \{A, \neg B, \neg C\},$
$C_5 = \{\neg A, B\}, C_6 = \{\neg B, C\}, C_7 = \{A, B\}, C_8 = \{A, \neg C\}, C_9 = \{B\},$
$C_{10} = \{\neg A, \neg C\}, C_{11} = \{C\}, C_{12} = \{A\}, C_{13} = \{\neg C\}$

The clause C_{10} is subsumed by C_{13}, so $S_7 = \{C_9, C_{11}, C_{12}, C_{13}\}$. This set does not contain the empty clause, so we look for another admissible clause. That clause is $C_{14} = \Box$, the child of C_{11} and C_{13}. We write it.

$C_1 = \{A, B, C\}, C_2 = \{\neg A, \neg B, \neg C\}, C_3 = \{A, B, \neg C\}, C_4 = \{A, \neg B, \neg C\},$

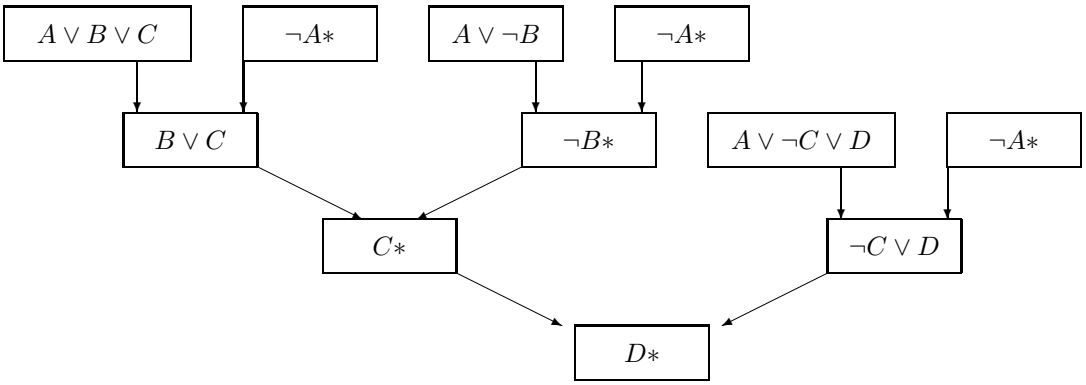


Figure 1.94: Unit resolution

$C_5 = \{\neg A, B\}, C_6 = \{\neg B, C\}, C_7 = \{A, B\}, C_8 = \{A, \neg C\}, C_9 = \{B\}$,
$C_{10} = \{\neg A, \neg C\}, C_{11} = \{C\}, C_{12} = \{A\}, C_{13} = \{\neg C\}, C_{14} = \square$

The empty clause subsumes all clauses, so $S_8 = \{\square\}$. Now the box is in S_8 and the algorithm finishes with success.

The N+T+Subsumption algorithm is almost identical to the P+T+Subsumption one, except that we change the word *positive* to *negative*. Its correctness proof is identical, except for changing *positive* to *negative*. Both algorithms are improvements over the resolution algorithm from the preceding section, but they are inefficient because seldom produce the shortest derivations. Their objective is to reduce the number of useless clauses produced in the search for the box, and not in finding a shortest derivation.

Now let us talk about another restriction of resolution, the *unit resolution*.

Definition 1.9.28 (unit clause, unit resolution) *A unit clause consists of single literal.*

A derivation tree is a unit resolution tree if each resolvent has as parent a unit clause.

Example 1.9.29 The tree from Figure 1.94 is a unit resolution tree of D from $T = \{\{A, B, C\}, \{\neg A\}, \{A, \neg B\}, \{A, \neg C, D\}\}$, because every resolvent has a unit clause as parent. For illustration purposes we marked the unit clauses with a $*$.

It is easy to show that the unit resolution is not complete. The sequence

$C_1 = \{A, B\}, C_2 = \{A, \neg B\}, C_3 = \{\neg A, B\}, C_4 = \{\neg A, \neg B\}, C_5 = \{A\}, C_6 = \{\neg A\}, C_7 = \square$

is a derivation of $\square$ from $T = \{\{A, B\}, \{A, \neg B\}, \{\neg A, B\}, \{\neg A, \neg B\}\}$. So, T is unsatisfiable. However, we cannot derive the box from T using only unit resolution because the set does not contain any unit clause.

We can show that the unit resolution is complete for *Horn clauses*.

Definition 1.9.30 (Horn clause, Horn set) *A clause C is Horn if it contains at most one positive literal.*

A set of clauses is Horn if every clause in the set is Horn.

Example 1.9.31 The clauses $\{A, \neg B, \neg C\}$, $\{\neg A, \neg B\}$, $\square$, $\{\neg A\}$, $\{B\}$ are Horn because they contain at most one positive literal. The clause $\{A, B\}$ is not Horn because it has 2 positive literals.

Observation 1.9.32 The resolvent of two Horn clauses is also Horn.

Theorem 1.9.33 *The unit resolution is complete for Horn-clauses.*

Proof: We know that the P-resolution is complete for all sets of clauses. In the case of the Horn clauses, the P-clauses are unit clauses, so the P-resolution is unit resolution. **Q.E.D.**

1.9.1 Exercices

Exercise 1.9.1 *Let T be a minimally unsatisfiable set. Prove that a derivation tree of $\square$ from T must contain all clauses in T.*

Exercise 1.9.2 *Let T be a minimally unsatisfiable set with n clauses, and t a minimal derivation tree of $\square$ from T. Give a lower bound for the number of nodes of t as a function of n.*

Exercise 1.9.3 *Let S be a minimally unsatisfiable set. Prove that T has no tautologies, and for every literal L in S, $\bar{L}$ is also in S.*

Exercise 1.9.4 *Let $S = \{P_1, P_2, \ldots, P_n\}$ be a set of atoms. A max-term of this set is a disjunction $D = \{s_1P_1, s_2P_2, \ldots, s_nP_n\}$ where the s_i's are either the empty string or the negation sign.*

a. Prove that there are 2^n non-equivalent max-terms.

b. Prove that every formula whose set of atoms is a subset of S is equivalent to a conjunction of max-terms.

c. Prove that the set of all max-terms, T, is minimally unsatisfiable.

d. Find the sizes of $Res^1[T]$, $Res^2[T]$, ...,$Res^n[T]$, $Res^*[T]$, where T is the set of all max-terms.

e. Find the length of a minimal derivation sequence of $\square$ from T.

The size is the number of non-equivalent clauses in the set.

Exercise 1.9.5 *Find a derivation of $\square$ from $S = \{\{A\}, \{\neg A, B\}, \{\neg B, C\}, \{\neg C, D\},$ $\{\neg D, E\}, \{\neg E, F\}, \{\neg F, G\}, \{H, I\}, \{\neg H, I\}, \{H, \neg I\}, \{\neg H, \neg I\}\}$ by computing first the resolvents of the unit clauses. Then find a minimal derivation sequence and compare the lengths.*

Exercise 1.9.6 *Show that the sets* $S = \{\{A, B, C\}, \{A, B, \neg C\}, \{A, \neg B, C\}, \{A, \neg B, \neg C\}, \{\neg A, B, C\}, \{\neg A, B, \neg C\}, \{\neg A, \neg B, C\}, \{\neg A, \neg B, \neg C\}\}$ *and* $T = \{\{A, B, C\}, \{A, B, \neg C\}, \{A, \neg B, C\}, \{A, \neg B, \neg C\}, \{\neg A, B, C\}, \{\neg A, B, \neg C\}, \{\neg A, \neg B\}\}$ *are unsatisfiable using unrestricted resolution, P-resolution and N-resolution.*

Exercise 1.9.7 *Show that the set* $S = \{\{A, B, C, D\}, \{A, B, C, \neg D\}, \{\neg A, \neg C\}, \{\neg A, C\}, \{\neg B, \neg D\}, \{\neg B, D\}, \{\neg C\}\}$ *is unsatisfiable using P-resolution and N-resolution.*

Exercise 1.9.8 (Nerode and Shore) *We say that a set of literals* $\mathcal{S}$ *is an assignment if it does not contain a pair,* L, $\bar{L}$, *where* $\bar{L}$ *is the complement of* L. *For example,* $\{P_0, \neg P_8\}$, $\{P_i | i \in N\}$, $\{P_i | i \geq 100\}$ *are assignments. The set of literals* $\{P_6, \neg P_7, P_7\}$ *is not an assignment because it contains a literal,* P_7, *and its negation* $,\neg P_7$.

Now, let $\mathcal{A}$ *be an assignment. The clause* C *is a* $\mathcal{A}$*-clause if* $C \cap \mathcal{A} = \phi$. *A resolution tree is a* $\mathcal{A}$*-tree if every resolvent has one or two* $\mathcal{A}$*-clauses as parents.*

Show that the unrestricted resolution, the P-resolution, and the N-resolution are special cases of $\mathcal{A}$ *resolution. Identify the assignments that characterize these resolutions.*

Exercise 1.9.9 *Show that the* $\mathcal{A}$*-resolution is complete, i.e. the clause set* S *is unsatisfiable iff there is a* $\mathcal{A}$*-tree tree of* $\square$ *from* S.

Hint: The proof is almost identical to the completeness of the P-resolution.

Exercise 1.9.10 *How many nonequivalent Horn clauses can be defined with* n *atoms? Prove your answer.*

Exercise 1.9.11 *Apply The P+T+Subsumption Algorithm to the sets*
$S = \{\{A, B, C\}, \{A, B, \neg C\}, \{A, \neg B, C\}, \{A, \neg B, \neg C\}, \{\neg A, B, C\}, \{\neg A, B, \neg C\}, \{\neg A, \neg B, C\}, \{\neg A, \neg B, \neg C\}\}$ *and*
$T = \{A, \neg B, C\}, \{A, \neg B, D\}, \{\neg D, E\}, \{A, B\}, \{B, \neg E\}\{\neg A, \neg E\}\}$.

Exercise 1.9.12 *Apply the N+T+Subsumption Algorithm to the sets from the preceding exercise.*

Exercise 1.9.13 *Prove Observation 1.9.32.*

Exercise 1.9.14 *Prove that every Horn set that has no positive clauses is satisfiable.*

Exercise 1.9.15 *Show that the following sets are unsatisfiable using unit resolution:*

a. $\{\{\neg A, \neg B, \neg C\}, \{\neg A, \neg B, C\}, \{B\}, \{\neg B, E\}, \{\neg E, A\}\}$
b. $\{\{\neg A, \neg B, C\}, \{\neg A, \neg B, E\}, \{A\}, \{B\}, \{\neg E, \neg F\}, \{\neg C, F\}\}$

Exercise 1.9.16 (Schöning) *The algorithm from Figure 1.95 tests the satisfiablility of the Horn clause set* S. *Trace it for the three sets below.*

a. $\{\{\neg A, B\}, \{\neg A, \neg B, \neg C\}, \{A\}, \{\neg D, \neg E, C\}, \{\neg D, E, D\}\}$
b. $\{\{\neg A, \neg B, C\}, \{\neg A, \neg B, E\}, \{A\}, \{\neg A, B\}, \{\neg C, \neg A\}\}$
c. $\{\{\neg A, B, C\}, \{\neg A, \neg E\}, \{A\}, \{D, \neg F\}, \{F\}\}$

$T = S$;
while (T has clauses that are atoms) do
{
    pick an atom clause, say A, from T;
    remove from T all clauses that contain A;
    remove from the remaining clauses the literal $\neg A$;
    if (the empty clause is in T) then
        { write('S is unsatisfiable'); halt; }
}
write('S is satisfiable');
The model is obtained by assigning 1 to all removed atoms and 0 to the the atoms that are still in T;

Figure 1.95: A satifiability algorithm for Horn sets

Exercise 1.9.17 *Prove that algorithm from Figure 1.95 is correct, i.e. it always finishes and produces the correct answer.*

Exercise 1.9.18 *A clause is* **Krom** *if it has at most 2 literals. Is the resolvent of two Krom clauses Krom?*

Exercise 1.9.19 *How many non-equivalent Krom clauses can de defined with n atoms? Prove your answer.*

1.10 The Linear Resolution

This section presents several key concepts that will be used in the chapter on prolog. It defines the linear resolution and proves that this restriction is complete. After that, it defines the SLD resolution and shows that it is complete for the Horn sets.

Definition 1.10.1 (linear resolution) *The clause C is linearly resolvable from the clause set S based on the clause $C_0 \in S$, if there is a sequence $C_0, C_1, \ldots, C_n = C$, such that for all $i = 1, \ldots, n$, C_i is a resolvent of the clauses C_{i-1} and B_{i-1} where B_{i-1} is either a clause in S, or $B_{i-1} = C_j$ for some $j < i$. We call C_0 the base and the disjunctions B_{i-1} the side clauses of the resolution. The clauses in S are called input clauses.*

Figure 1.96 shows the graph of a linear resolution sequence. The graph is not always a tree because some of the side clauses B_i may be in the set $\{C_0, \ldots, C_{i-1}\}$. So, some nodes can have an out-degree greater than 1.

Example 1.10.2 Figure 1.97 shows a linear derivation of $\square$ from $S = \{\{A, B\}, \{A, \neg B\}, \{\neg A, C\}, \{\neg A, \neg C\}\}$, based on $\{A, B\}$. Let us verify that the tree is linear. First, each of the clauses C_i, $1 \leq i \leq 4$, has C_{i-1} as a parent. The base clause, C_0, is in S, and the last clause, C_4, is the box. The side clauses of

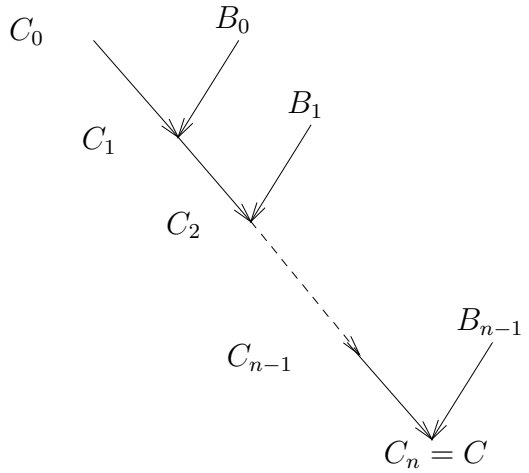


Figure 1.96: A linear resolution tree

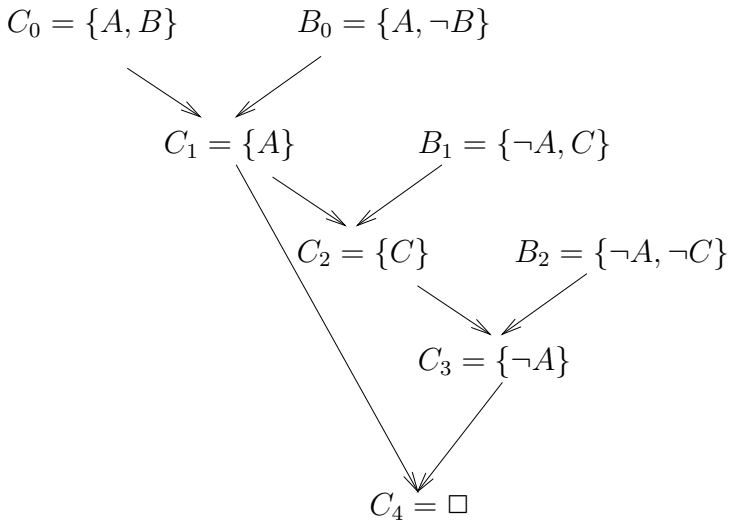


Figure 1.97: The linear tree from Example 1.10.2

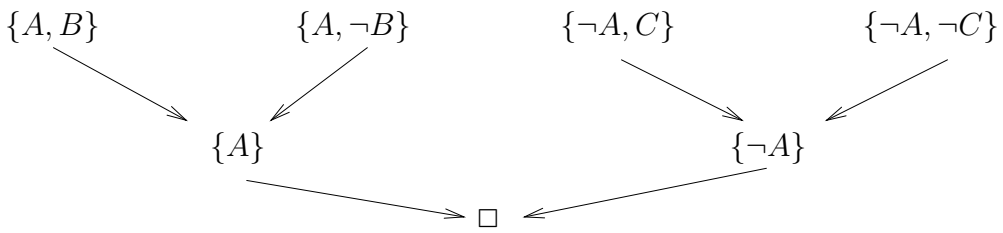


Figure 1.98: A non-linear derivation tree

C_1, C_2, C_3, C_4 are respectively, $B_0 = \{A, \neg B\}$, $B_1 = \{\neg A, C\}$, $B_2 = \{\neg A, \neg C\}$, and $B_3 = C_1$. They are either input clauses, or preceding C_j clauses. So, the tree from Figure 1.97 is a linear derivation.

Not all resolution trees are linear. One such tree is the derivation of $\Box$ from $S = \{\{A, B\}, \{A, \neg B\}, \{\neg A, B\}, \{\neg A, \neg B\}\}$ shown in Figure 1.98. Let us show that we cannot find a sequence $C_0, \ldots, C_n$ that satisfies Definition 1.10.1. First of all, C_0 must be an input clause. If it is $\{A, B\}$ or $\{A, \neg B\}$, $C_1 = \{A\}$ and $C_2 = \Box$. Then, the side clause of C_2, $B_1 = \{\neg A\}$, is neither an input clause, nor any of the clauses C_0, C_1, contradicting Definition 1.10.1.

If the base clause is either $\{\neg A, B\}$ or $\{\neg A, \neg B\}$, then $C_1 = \{\neg A\}$ and $C_2 = \Box$. Again, the side clause $B_1 = \{A\}$ is neither an input clause, nor one of the C_j clauses that precede C_2, contradicting Definition 1.10.1. So, the tree in Figure 1.98 is not linear.

Next, we will prove that the linear resolution is complete. But first we need three lemmas that characterize the minimally unsatisfiable sets.

Lemma 1.10.3 *Every minimally unsatisfiable set is finite.*

Proof: We prove it by contradiction. Let S be an infinite set that is minimally unsatisfiable. By The Compactness Theorem it has a finite unsatisfiable subset S'. Since S' is finite, $S' \neq S$. So, S has a proper subset that is unsatisfiable, contradicting the assumption that S is minimally unsatisfiable. **Q.E.D.**

Lemma 1.10.4 *Every unsatisfiable set contains a subset that is minimally unsatisfiable.*

Proof: Every unsatisfiable set contains a finite unsatisfiable subset, so it is sufficient to prove the lemma for finite, unsatisfiable sets. The proof is by induction on n, the number of clauses in the unsatisfiable set S.
Basis: $n = 0$. Then $S = \{\}$ is satisfiable, so the lemma is true by default.
Inductive Step: Assume that every set with less than or equal to n clauses contains a minimally unsatisfiable set. Let S be an unsatisfiable set with $n + 1$ clauses. We have two cases:

Case 1: S is minimally unsatisfiable. Then we are done.

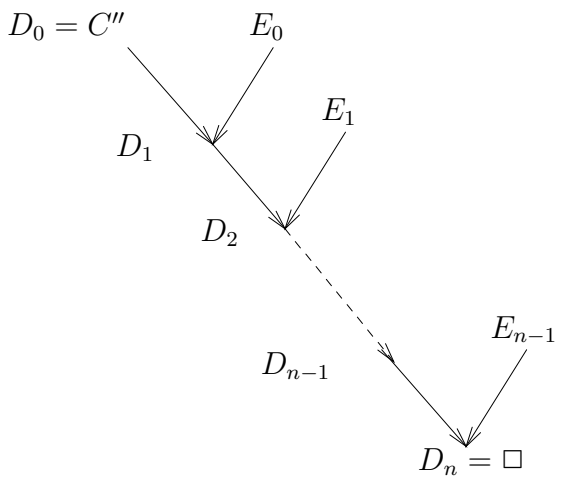


Figure 1.99: The IH tree from Case 1, Lemma 1.10.5

Case 2. S is not minimally unsatisfiable. Then, it has a proper subset $S' \neq S$ that is unsatisfiable. Since S' has n or fewer clauses, it contains a minimally unsatisfiable set S''. Because S'' is also a subset of S, the lemma is true for S. **Q.E.D.**

Lemma 1.10.5 *Let S' be a minimally unsatisfiable set and C be a clause in S'. Then there is a linear resolution of $\Box$ from S' with base C.*

Proof: By induction on the number of atoms of S'.
Basis: $n = 0$. Then $S' = \{\Box\}$, so $C = \Box$.
Inductive Step: Assume that for every minimally unsatisfiable set T with n or fewer atoms and for any $C \in T$, there is a linear derivation of $\Box$ with base C . Let S' be a minimally unsatisfiable set with $n + 1$ atoms and C be a clause in S.
Case 1: C is a unit clause, $\{L\}$. Then $S'_{L=1}$ is unsatisfiable, and contains only n atoms. Let S'' be a minimally unsatisfiable subset of $S'_{L=1}$. If S'' is included in S', then S'' is a proper subset of S' since $C \in (S' - S'')$. But this contradicts the fact that S' is minimally unsatisfiable. So, S'' is not a subset of S'. Then there is some $C'' \in S''$ such that $C'' \cup \{\bar{L}\}$ is a clause in S'. Since S'' has fewer than $n + 1$ atoms, we apply the induction hypothesis to S'' and get the derivation from Figure 1.99. There are two problems with this derivation. First, the base clause is not C, and second, some of the side clauses E_i may not be in S'. We construct a derivation tree t of $\Box$ from S' by simulating the resolutions from Figure 1.99. We start the tree with the resolution from Figure 1.100. Then, for every resolution $D_i \longrightarrow D_{i+1} \longleftarrow E_i$ of Figure 1.99, we append resolutions to the last D_i node of t. If E_i is S', we add the resolution $D_i \longrightarrow D_{i+1} \longleftarrow E_i$. If E_i is not in S', then it is obtained by deleting $\bar{L}$ from an S' clause. So, $E_i \cup \{\bar{L}\}$ is a clause in S'. In this case we append the tree τ from Figure 1.101. We claim

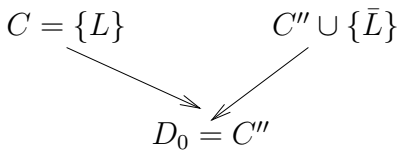


Figure 1.100: The base of the tree from Case 1, Lemma 1.10.5

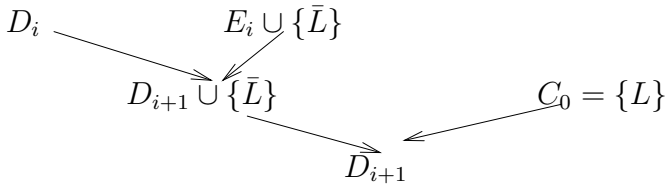


Figure 1.101: The tree τ from Lemma 1.10.5

that t is a linear derivation tree of $\Box$ from S' based on $C = \{L\}$. The sequence of $C_0, C_1, \ldots, C_m$ is the path from $C = \{L\}$ to $D_n = \Box$. The side clauses are $C'' \cup \{\bar{L}\}$, the E_i clauses that are in S', the $E_i \cup \{\bar{L}\}$ clauses with $E_i \notin S'$, and $C_0 = \{L\}$. All these clauses are in S', so t is linear derivation of $\Box$ from S'.
Case 2: C is not a literal. Then $C = C' \cup \{L\}$ where L is a literal and C' is a clause that does not contain L. Since S' is unsatisfiable, so is ${S'}_{L=0}$. By Lemma 1.10.4, ${S'}_{L=0}$ contains a minimally unsatisfiable subset S''. We leave the proof that C' is a clause in S'' to the reader. (Exercise 1.10.2) Now we apply the induction hypothesis to S'' and get the linear resolution tree displayed in Figure 1.102. Some of the clauses in this tree are not in S', so we add L to all disjunctions that are not in S' and get the derivation shown in Figure 1.103. This tree is a linear derivation of L from S' based on C. Now let us show that $(S' - \{C\}) \cup \{L\}$ is unsatisfiable. We do it by contradiction. Let us assume that $\models_{\mathcal{A}} (S' - \{C\}) \cup \{L\}$. Then $\mathcal{A}$ is a model of $S' - \{C\}$ and a model of L. Since L is a disjunct of C, $\models_{\mathcal{A}} C$. So, $\models_{\mathcal{A}} (S' - \{C\}) \cup \{C\} = S'$. But this is impossible since S' is unsatisfiable.

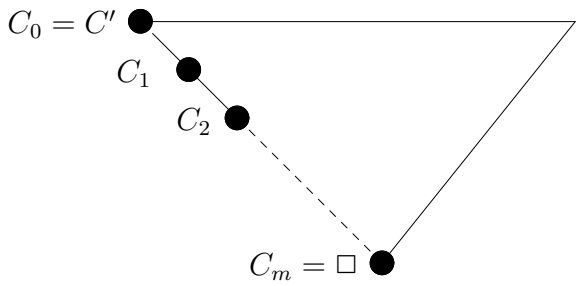


Figure 1.102: The IH tree for Case 2, Lemma 1.10.5

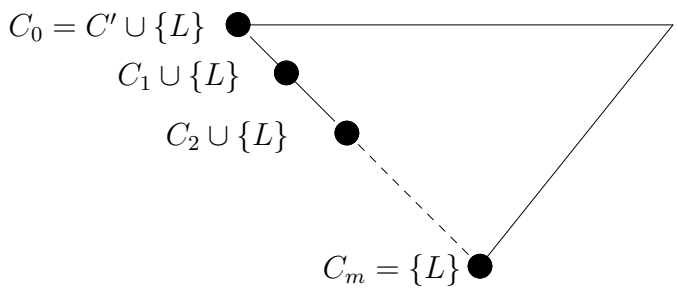


Figure 1.103: Linear derivation of $\{L\}$ from S'

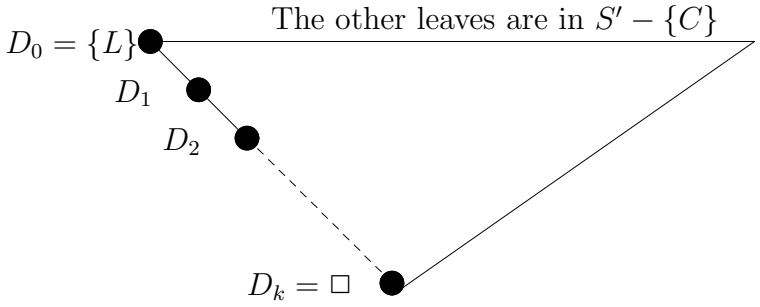


Figure 1.104: Linear derivation of $\square$ from $(S' - \{C\}) \cup \{L\}$

By Lemma 1.10.4, $(S' - \{C\}) \cup \{L\}$ has a minimally unsatisfiable subset T'. If $\{L\} \notin T'$, T' is a proper subset of S' because $C \notin T'$. Then S' has a proper unsatisfiable subset, contradicting the assumption that S' is minimally unsatisfiable. So, $\{L\} \in T'$.

Now we apply Case 1 to T' and we get a linear derivation of $\square$ from $T' \subseteq (S' - \{C\}) \cup \{L\}$ based on L. Finally we join the trees from Figures 1.103 and 1.104 and we get the linear derivation tree from Figure 1.105. **Q.E.D.**

Now, let us give 2 examples that illustrate the constructs from Lemma 1.10.5.

Example 1.10.6 Let us illustrate the construction from Case 1 of Lemma 1.10.5. So, let $S' = \{\{R\}, \{P, Q, \neg R\}, \{P, \neg Q, \neg R\}, \{\neg P\}\}$. The reader can check, by using a truth table, that S' is unsatisfiable and all the subsets with 3-elements are satisfiable. So, S' is minimally unsatisfiable. We want to build a linear resolution of $\square$ from S' based on $C = \{R\}$. First we form the set $S_{R=1} = \{\{P, Q\}, \{P, \neg Q, \}, \{\neg P\}\}$ by deleting the S' clauses that contain R and removing $\neg R$ from the remaining ones. This set is minimally unsatisfiable, so $S'' = S_{R=1}$. Now, we choose a clause $C'' \in S''$, such that $C'' \cup \{\neg R\}$ is in S'. Let us choose $C'' = \{P, Q\}$. Now, the induction hypothesis gives a linear derivation of $\square$ from S'' based on C''. Let us assume that we get the tree from Figure 1.106. We modify this tree to get a linear derivation of $\square$ from S' based on $C = \{R\}$. We start with the resolution from Figure 1.107, that computes the

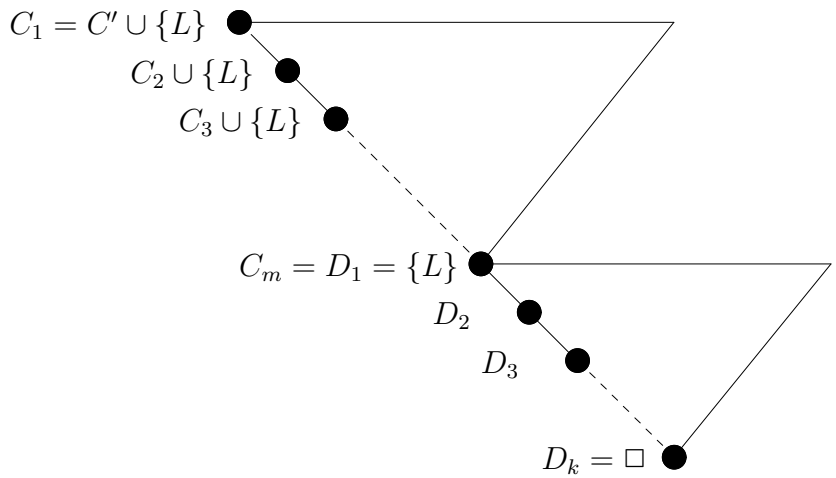


Figure 1.105: The linear derivation of Case 2, Lemma 1.10.5

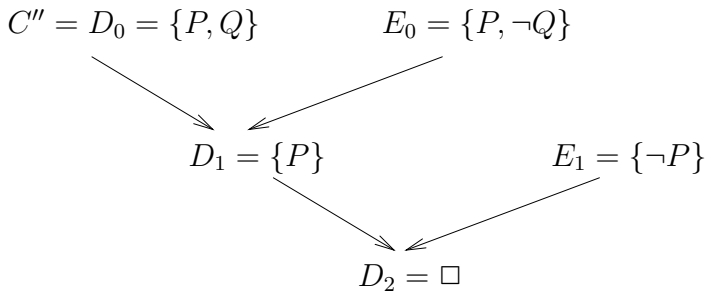


Figure 1.106: The IH tree from Example 1.10.6

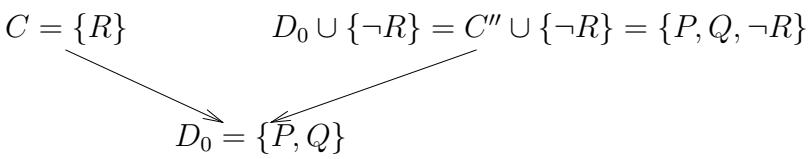


Figure 1.107: The top of the output tree of Example 1.10.6

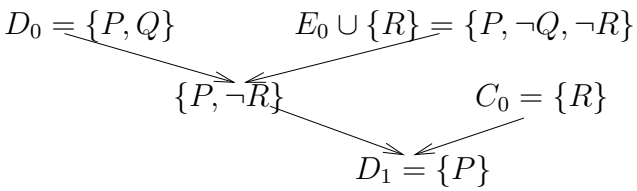


Figure 1.108: The middle of the output tree of Example 1.10.6

resolvent D_0 of $C = \{R\}$ and $C'' \cup \{\neg R\} = \{P, Q, \neg R\}$. Next, we look at the resolution step $D_0 \longrightarrow D_1 \longleftarrow E_0$ of Figure 1.106. The side clause $E_0 = \{P, \neg Q\}$ is not in S'', so we must add the derivations from Figure 1.108. Now, we process the step $D_1 = \{P\} \longrightarrow \square \longleftarrow E_1 = \{\neg P\}$ of Figure 1.106. The side clause E_1 is in S'', so we add the resolution $D_1 = \{P\} \longrightarrow \square \longleftarrow E_1 = \{\neg P\}$ at the bottom of the tree. We get the tree from Figure 1.109.

The reader is invited to verify that this is a linear derivation tree of $\square$ from S' based on $C = \{R\}$.

Example 1.10.7 Now let us apply the construction from Case 2 of Lemma 1.10.5 to the set $S' = \{\{R\}, \{P, Q, \neg R\}, \{P, \neg Q, \neg R\}, \{\neg P\}\}$ from Example 1.10.6. We select the clause $C = \{P, \neg Q, \neg R\}$ as the base clause. Next, we choose one literal, say $L = \neg Q$, of C and we set $C' = C - \{L\} = \{P, \neg R\}$. We compute $S'_{\neg Q=0} = S'_{Q=1}$. We delete all S' clauses that contain Q, we remove $\neg Q$ from the remaining clauses, and get $S'_{Q=1} = \{\{R\}, \{P, \neg R\}, \{\neg P\}\}$. This set is minimally unsatisfiable, so $S'' = S'_{Q=1}$. Since S'' has fewer atoms than S', we apply the lemma to S'' and C' and get the tree from Figure 1.110. We add $L = \neg Q$ to all leaves that are not in S' and to all their ancestors. We get the S'-tree from Figure 1.111. Now we find a minimally unsatisfiable subset T' of $(S' - \{C\}) \cup \{L\} = \{\{R\}, \{P, Q, \neg R\}, \{\neg P\}, \{\neg Q\}\}$. The last set is minimally unsatisfiable, so $T' = (S' - \{C\}) \cup \{L\}$. We apply Case 1 of the lemma to find a linear resolution of $\square$ from T' based on $\{L\} = \{\neg Q\}$. We get the tree from Figure 1.112. Finally, we join the trees from Figures 1.111 and 1.112 at $\{\neg Q\}$ and get the linear resolution from Figure 1.113.

Now we can state the completeness of the linear resolution.

Theorem 1.10.8 *The linear resolution restriction is complete.*

Proof: We need to show that every unsatisfiable set S has a linear derivation of $\square$. The Compactness Theorem tells us that S has a finite unsatisfiable

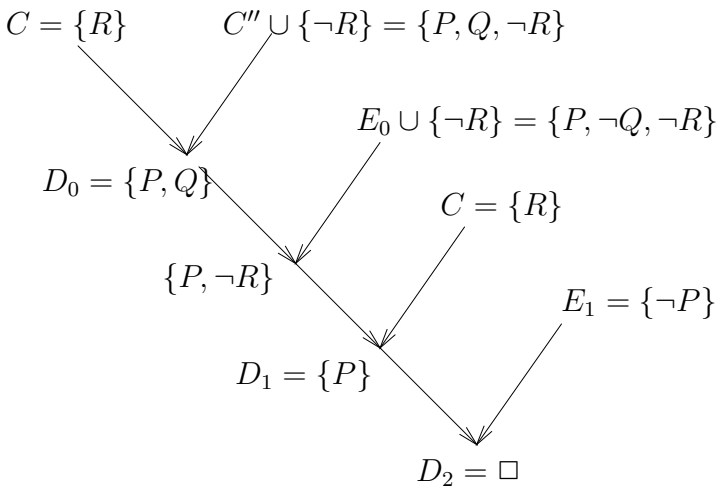


Figure 1.109: The linear derivation tree from Example 1.10.6

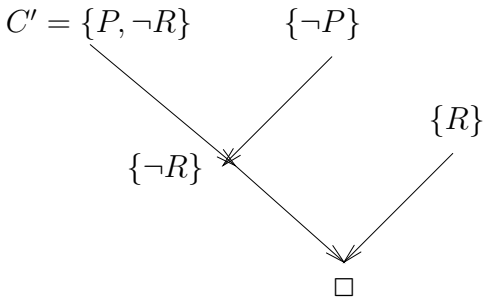


Figure 1.110: The IH tree for Example 1.10.7

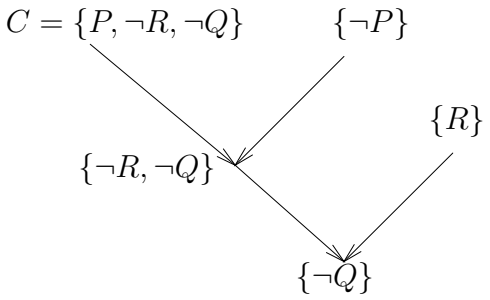


Figure 1.111: The derivation of $\neg Q$ from S'

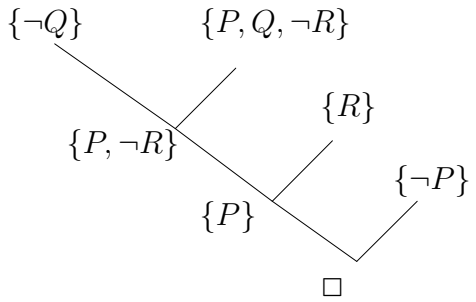


Figure 1.112: The derivation of $\square$ from T'

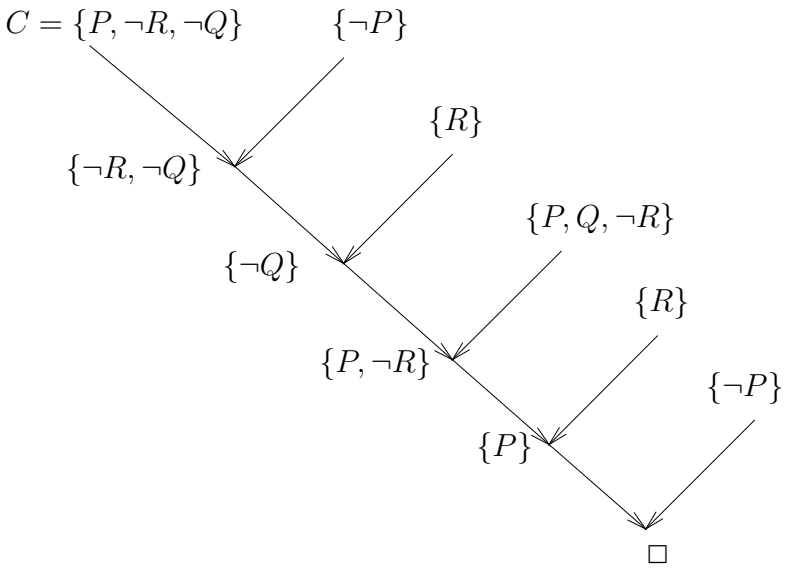


Figure 1.113: A linear derivation of $\square$ from S' based on $\{P, \neg Q, \neg R\}$

subset S'. Lemma 1.10.4 provides a minimally unsatisfiable subset, S'', of S'. Lemma 1.10.5 guarantees that there is a linear derivation of $\Box$ from S''. Since S'' is a subset of S, $\Box$ is linearly derivable from S.**Q.E.D.**

Now we will present another restriction of resolution, *the input resolution.*

Definition 1.10.9 (input resolution) *A derivation sequence $C_1, \ldots, C_m$ is an input resolution of C_m from S if every clause in the sequence is either an input clause, or has at least one parent that is an input clause.*

Example 1.10.10 The sequence $C_1 = \{A, B, C\}$, $C_2 = \{\neg A\}$, $C_3 = \{B, C\}$, $C_4 = \{\neg B\}$, $C_5 = \{\neg C, D\}$, $C_6 = \{C\}$, $C_7 = \{D\}$, $C_8 = \{\neg D\}$, $C_9 = \Box$ is an input resolution of $\Box$ from $S = \{\{A, B, C\}, \{\neg A\}, \{\neg B\}, \{\neg C, D\}, \{\neg D\}\}$ because the clauses C_1, C_2, C_4, C_5, C_8 are input clauses and C_3, C_6, C_7, C_8 have input clauses as parents (respectively C_1, C_4, C_5, C_8).

Proposition 1.10.11 *The input resolution is a special case of linear resolution.*

Proof: Let $s = C_1, C_2, \ldots, C_n$ be an input derivation of C from S. We construct a sequence $D_0, D_1, \ldots$ as follows:

We start with $D_0 = C$. If D_0 is an input clause, we stop. Otherwise we look at D_0's parents. Since s is an input derivation, at least one of them is an input clause. Let B_0 be a parent labeled with an input clause and D_1 be the other parent. Now we do the same thing for D_1. If D_1 is an input clause we stop; otherwise let B_1 be a parent that is an input clause and D_2 be the other parent. We continue this way and get two sequences $D_0, D_1, \ldots, D_i, \ldots$ and $B_0, B_1, \ldots, B_i, \ldots$.

Now let us look at the positions of the clauses D_0, D_1, $\ldots$,D_i in the s-sequence. The clause D_1 precedes D_0, D_2 precedes D_1 and so on. So, all clauses D_1,$\ldots$,D_i are distinct predecessors of D_0. Since D_0 has only $n-1$ predecessors, the sequence $D_0, D_1, \ldots, D_i, \ldots$ is finite. Hence, there is an m such that D_m is an input clause. Now let t' be the tree from Figure 1.114. This tree is an input derivation of $D_0 = C$ from S because all B_i's are input clauses. At the same time it is a linear derivation of C from S because D_m is an input clause and every D_{i-1}, $1 \leq i < m$, is a resolvent of D_i. **Q.E.D.**

Unlike the linear resolution, the input resolution is not complete. Let us try to derive $\Box$ from the set of input clauses $S = \{\{A, B\}, \{A, \neg B\}, \{\neg A, C\}, \{\neg A, \neg C\}\}$. We can start with any clause from the S and unify it with any input clause. The resolvent will have either one or two literals, depending on whether the two clauses have a common literal or not. We can unify the resolvent with an input clause and again we get a one or a two literal resolvent. This continues forever, since we cannot derive $\Box$ by unifying any resolvent with a two literal clause. This situation is illustrated in Figure 1.115.

For the rest of the section we will deal only with sets of Horn clauses. We remember that a Horn clause contains at most one atom. We classify the Horn clauses according to the number of atoms. The negative clauses are called *goals*, and the non-negative ones, *definite* or *program clauses*. The atom clauses are also called *facts*.

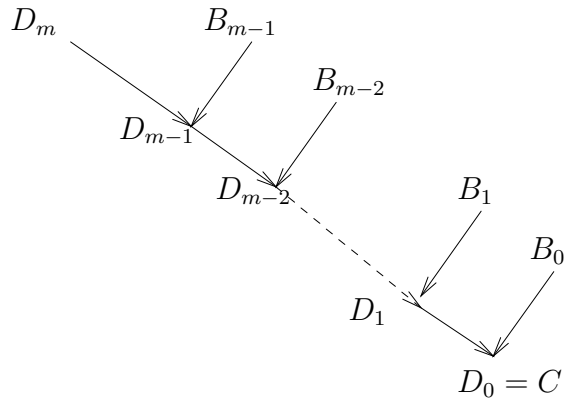


Figure 1.114: The linear derivation from Proposition 1.10.11

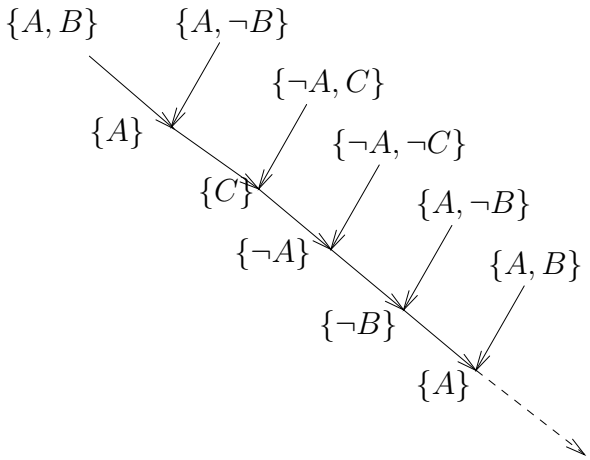


Figure 1.115: The input resolution is not complete

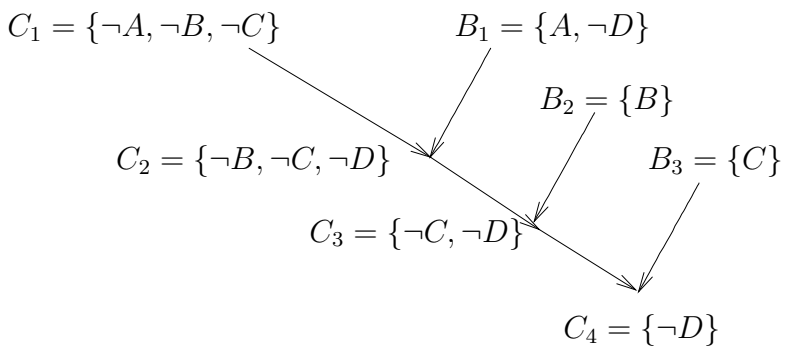


Figure 1.116: An *SLD* resolution

Definition 1.10.12 (*SLD* resolution) *A linear resolution $C_1, \ldots, C_n$ is an SLD resolution if the base is a negative clause and, at each step, the side clause is a non-negative input clause.*

The name *SLD* resolution stands for *linear resolution with selection function for definite clauses*. We notice that the SLD resolution is both linear and input resolution.

Example 1.10.13 Figure 1.116 is an SLD resolution of $\neg D$ from $S = \{\{\neg A, \neg B, \neg C\}, \{A, \neg D\}, \{B\}, \{C\}\}$.

Theorem 1.10.14 (The completeness of the SLD resolution for Horn sets) *The SLD resolution is complete for Horn sets.*

Proof: We use the completeness of the linear resolution. So, let S be an unsatisfiable set of Horn clauses. By Lemma 1.10.4, S contains a minimally unsatisfiable set S'. If S' contains no negative (goal) clause, the truth assignment that assigns 1 to all the atoms is a model of S', contradicting the fact that S' is unsatisfiable. So, S' has a goal clause. By Lemma 1.10.5 there is a linear derivation of $\square$ from S', based on the goal clause C_0, as shown in Figure 1.117. All we have to do is to show that the clauses B_0,...,B_{n-1} are program clauses. Since C_0 and B_0 are unifiable and C_0 is negative, B_0 must contain a positive literal. At the same time B_0 is a Horn clause, so it cannot have more than one positive literal. Then, the resolvent C_1 of C_0 and B_0 is a negative clause. We continue this reasoning and get that all clauses $C_0, C_1, \ldots, C_{n-1}$ are negative and the clauses $B_0, \ldots, B_{n-1}$ contain positive atoms.

Since Figure 1.117 is a linear tree, the clause $B_0, \ldots, B_{n-1}$ are either input clauses or preceding C_i clauses. But we cannot have $B_j = C_i$, $i \leq j$, because the C_i clauses are negative, and the B_j's contain positive literals. So, the side clauses are non-negative input clauses, i.e. program clauses. So, Figure 1.117 is an SLD tree. **Q.E.D.**

Corollary 1.10.15 *Every minimally unsatisfiable set of Horn clauses contains only one goal clause.*

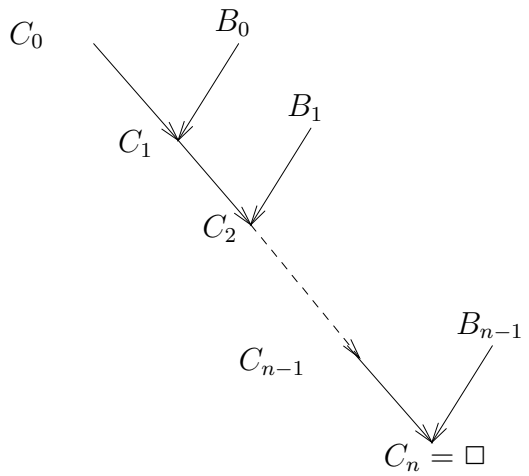


Figure 1.117: The linear tree from Theorem 1.10.14

Proof: Let S be a minimally unsatisfiable Horn set. By the completeness of the SLD resolution for Horn sets, S has a derivation tree like the one in Figure 1.117. The leaves of the tree are in S and they form an unsatisfiable set. Since S is minimally unsatisfiable, it is equal to the set of leaves of the tree. Since the tree has only one negative leaf, S has only one goal clause. **Q.E.D.**

Exercises

Exercise 1.10.1 *Construct a linear derivation of* $\square$ *based on* $\{\neg A, \neg B\}$ *from* $S = \{\{A, B\}, \{A, \neg B\}, \{\neg A, B\}, \{\neg A, \neg B\}\}$.

Exercise 1.10.2 *Prove the following assertion:*

Let S' *be a minimally unsatisfiable set of clauses and* C *a clause of* S' *and* L *a literal of* C. *Let* S'' *be a minimally unsatisfiable subset of* $S'_{L=0}$. *Then* $C' = C - \{L\}$ *is a clause of* S''.

Exercise 1.10.3 *True or false?*

If S *is a minimally unsatisfiable set of clauses and* L *is a literal that occurs in a clause of* S, *then* $S_{L=1}$ *is minimally unsatisfiable.*

Exercise 1.10.4 *Find a minimally unsatisfiable set* S' *that contains a clause* $C = C' \vee L$ *such that the set* $(S' - \{C\}) \cup \{L\}$ *is not minimally unsatisfiable.*

Exercise 1.10.5 *Use the SLD resolution to show that the set* $S = \{\{\neg A, \neg B, \neg C\}, \{A\}, \{\neg A, D\}, \{\neg A, E\}, \{\neg D, B\}, \{\neg E, C\}\}$ *is unsatisfiable.*

Exercise 1.10.6 *Assume that the algorithm from Figure 1.95 is correct for Horn clauses, i.e. it halts and produces the right answer. Prove that the correctness of this algorithm implies that every minimally unsatisfiable Horn set has only goal clause.*

Exercise 1.10.7 *Let S be a minimally unsatisfiable set of Horn clauses and B an atom of S. Show that B belongs to only one clause of S.*

Exercise 1.10.8 *Let S be a minimally unsatisfiable set of Horn clauses, and let $\{\neg A_1, \ldots, \neg A_k\}$ be the goal of S. Prove that there is some index i, $1 \leq i \leq k$, such that A_i belongs to a program clause of S, but $\neg A_i$ is not in any definite clause.*

Exercise 1.10.9 *Show that the correctness of the algorithm from Figure 1.95 implies that the SLD resolution is complete for Horn clauses.*

Chapter 2

First Order Logic

While the propositional logic is useful for many applications, it has drawbacks. One of them is that it cannot adequately represent the sentences of a natural language. For example, the sentences *Politicians love power*, *The governments love taxes*, *Mark loves money*, have something in common, namely the verb *love*. All three sentences describe a relation, denoted by the verb *love*, between the first and the second noun of the sentence. However, in propositional logic the three sentences are just symbols P_i, P_j, P_k in the alphabet of the language.

Also, the propositional language cannot capture the natural language inferences *if x is older than y and y is older than z then x is older than z* and *if x is married to y then y is married to x*.

The other problem with the propositional logic is that it is next to useless in expressing and handling mathematical statements. The propositional logic is not even capable of making the inference *f has a domain and a codomain* from the two statements *every function has a domain and a codomain* and *f is a function* because it does not contain the symbol *every* and does not have the necessary inference rule.

One of the differences between first order logic and propositional logic is that the atomic formulas have a well defined structure. Each atomic formula consists of a relation (predicate) symbol followed by a list of arguments. For example, the sentence *Michael loves Zelda* is represented as *loves(Michael, Zelda)*, where *loves* is the name of the relation, *Michael* is its first argument and *Zelda* is its second argument.

In first order logic we differentiate between constants and variables. In the statement *Mark is rich*, *Mark* is a constant while in *Some men are rich*, *men* is a variable. The first order logic has two quantifiers, *for all* and *there exists*. So, we can say *All men are mortal* and *There are dishonest politicians*.

The first order logic can describe a sizable part of English and it can be used for natural language understanding. Also, it is adequate for presenting many mathematical theories, like set theory. Moreover, first order logic has applications in artificial intelligence and in databases.

The first order logic is not a complement of the propositional logic, but an

extension. Many of the concepts and the results shown here are extensions of the ones presented in the preceding chapter. The chapter layout mirrors the organization of the propositional logic chapter.

2.1 FOL Formulas

Let us start by describing the alphabet of the first order logic , abbreviated FOL. The symbols of the alphabet are:

1. A set of symbols P_i^n called *predicate symbols*. Here i and n are numbers in the set $N = \{0, 1, \ldots, n, \ldots\}$. We call n *the arity* (the number of arguments) and i *the index* of P_i^n. If the arity of P_i^n is 0 then P_i^n is called a *predicate constant*.

2. A set of symbols f_i^n called *function symbols*. Just like in the previous set, i and n are natural numbers representing the *index*, respectively *the arity*, of the function symbol. If $n = 0$, f_i^n is called an *individual constant*.

3. A set of symbols x_i named *individual variables*. The natural number i is called the *index of the variable*.

4. The left parenthesis (, the right parenthesis), and the comma ,.

5. The connectives $\neg$, $\wedge$, $\vee$, $\longrightarrow$, $\longleftrightarrow$. We use the same terminology as in the propositional calculus: $\neg$ is called *the negation* symbol, $\wedge$ is the *and* symbol, $\longrightarrow$ is the *if* symbol, and $\longleftrightarrow$ is called the *if and only if* symbol.

6. The *equality predicate* E. This predicate has arity 2.

7. The *quantifiers* $\forall$ and $\exists$. The symbol $\forall$ is *the universal quantifier* and $\exists$ is *the existential quantifier*.

This is the alphabet of the *full predicate calculus*. If we take away both the equality predicate and the function symbols, we have the *pure predicate calculus*. In the pure predicate calculus we have only variables, predicate symbols, the connectives, the parentheses, the comma and the quantifiers.

If we remove only the equality predicate then we get the *predicate calculus with functions*. In the predicate calculus with functions, we have variables, function symbols, predicate symbols, the connectives, the parentheses, the comma and the quantifiers.

The *predicate calculus with equality* has all symbols of the full predicate calculus minus the function symbols. So, we have variables, predicate symbols, the equality predicate, the parentheses, the comma, the connectives and the quantifiers.

The relations among these calculi will be discussed in Section 2.7.

Now we are going to define *the terms*. Intuitively, terms are used to identify objects, just like the nouns and the pronouns do in the spoken languages. In English, the strings *John*, *a man*, *the man's father*, *my neighbor's maternal grandmother* represent objects[1]; the first is a constant, the second is a variable, and the last two are more complex nouns. In FOL, the terms can also be

[1] These strings are called noun phrases

constants, variables, or complex terms, formed by applying function symbols to constants and/or variables.

Definition 2.1.1 (terms) *1. The individual variables are terms.*

2. The individual constants are terms.

3. If t_1, t_2,..., t_n are terms and f_i^n is a function symbol of arity $n > 0$ then $f_i^n(t_1, t_2, \ldots, t_n)$ is a term.

Examples 2.1.2 1. x_4 is a term by rule 1 of Definition 2.1.1.

2. f_2^0 is a term by rule 2 of Definition 2.1.1.

3. $f_{11}^2(x_4, f_2^0)$ is a term by rule 3 because x_4 and f_2^0 are terms and f_{11}^2 is a function symbol of arity 2. So, $f_{11}^2(x_4, f_2^0)$ is a term by rule 3 of Definition 2.1.1.

4. x_0 is a term by rule 1 of Definition 2.1.1.

5. $f_0^3(f_2^0, f_{11}^2(x_4, f_2^0), x_0)$ is a term because f_0^3 is a function symbol of arity 3 and f_2^0, $f_{11}^2(x_4, f_2^0)$ and x_0 are terms.

Observation 2.1.3 The only terms that occur in the pure predicate calculus and in the predicate calculus with equality are the variables.

Definition 2.1.1 gives us an algorithm for recognizing terms. The algorithm from Figure 2.1 starts by underlining the constants and the variables of the input string S. Then it enters a loop that keeps sweeping S and underlines bigger and bigger substrings. It stops when S is fully underlined, the last sweep produced no underlines, or there is a collision. A *collision* occurs when one forgets to separate the terms $t_1, \ldots, t_n$ of $f(t_1, \ldots, t_n)$.

Example 2.1.4 Let us apply Algorithm 2.1 to the strings $S = f_0^3(f_1^2(x_2, f_1^0), x_2, f_2^0)$ and $T = f_5^2(f_6^2(x_1, x_2)f_1^1(x_3))$.

After Step 1 is finished, S has the underlines $f_0^3(f_1^2(\underline{x_2}, \underline{f_1^0}), \underline{x_2}, \underline{f_2^0})$

The algorithm produced new underlines, so we execute Step 2. We get the underlines $S = f_0^3(\underline{f_1^2(x_2, f_1^0)}, \underline{x_2}, \underline{f_2^0})$.

Step 2 produced new underlines and S is not fully underlined, so we execute Step 2. We get the underlines $S = \underline{f_0^3(f_1^2(x_2, f_1^0), x_2, f_2^0)}$. Now S is fully underlined and we stop with success.

Now let us apply the algorithm to T. After Step 1 is finished we have the underlines, $T = f_5^2(f_6^2(\underline{x_1}, \underline{x_2})f_1^1(\underline{x_3}))$.

The algorithm produced new underlines, and T is not fully marked, so we execute Step 2. We underline $f_6^2(x_1, x_2)$ but when we try to underline $f_1^1(x_3)$ we have a collision. So, we stop with failure; T is not a term.

Now let us define *atomic formulas*. Intuitively, atomic formulas correspond to natural language clauses because they describe a simple relationship and cannot be broken down into smaller sentences.

Definition 2.1.5 (atomic formulas) *1. The predicate constants P_i^0 are atomic formulas.*

2. If t_1 and t_2 are terms then $E(t_1, t_2)$ is an atomic formula. We read $E(t_1, t_2)$ as t_1 is identical to t_2.

Step 1: while [scanning S from left to right] do
    if [the scanned symbol is a variable or a constant] then
        if [the preceding symbol is underlined] then
            stop with failure;
        else
            underline the symbol;
Step 2: while [[S is not fully underlined] and [the last sweep produced new underlines]] do
    while [scanning S from left to right] do
        if [[the scanned symbol is the head of the substring $f(T_1, \ldots, T_n)$]
            and [the substrings T_1,...,T_n are underlined] and
            [the arity of f is $n > 1$] and
            [the other symbols of $f(T_1, \ldots, T_n)$ are not underlined]] then
            if (the symbol preceding f is underlined) then
                stop with failure;
            else
                underline $f(T_1, \ldots, T_n)$;
Step 3: if (S is fully underlined)
    stop with success;
    else
    stop with failure;

Figure 2.1: The Term Recognition Algorithm

3. If P_i^n is a predicate symbol of arity $n \geq 1$ and t_1, t_2,..., t_n are terms then $P_i^n(t_1, t_2, \ldots, t_n)$ is an atomic formula. We read it P_i^n of t_1, t_2, ..., t_n.

Examples 2.1.6 Let us show some atomic formulas.

1. P_3^0 is an atomic formula by rule 1 of Definition 2.1.5.
2. $E(f_3^1(x_2), f_5^0)$ is an atomic formula by rule 2 because $f_3^1(x_2)$ and f_5^0 are terms.
3. $P_4^5(x_3, f_3^0, f_3^1(f_0^0), f_2^2(x_5, f_4^0), f_{11}^0)$ is an atomic formula since P_4^5 has arity 5 and x_3, f_3^0, $f_3^1(f_0^0)$, $f_2^2(x_5, f_4^0)$ and f_{11}^0 are terms.

The formulas are formed from atomic formulas by repeatingly applying the connectives and the quantifier operators.

Definition 2.1.7 (Formulas) *1. The atomic formulas are formulas.*

2. If F is a formula then $\neg F$ is a formula, called the negation of F.

3. If F and G are formulas then $(F \vee G)$ is also a formula, called F or G.

4. If F and G are formulas then $(F \wedge G)$ is also a formula, called F and G.

5. If F and G are formulas then $(F \longrightarrow G)$ is also a formula, called if F then G.

6. If F and G are formulas then $(F \longleftrightarrow G)$ is also a formula, called F if and only if G.

7. If F is a formula and x is a variable then $\forall xF$ is also a formula, called for all x F. We say that x is the argument of the operator $\forall x$, and that x is universally quantified.

8. If F is a formula and x is a variable then $\exists xF$ is a formula, called for some x F or there exists x such that F. Again we say that x is the argument of operator $\exists x$ and that x is existentially quantified.

Remark 2.1.8 We will treat the strings $\forall x$ and $\exists x$ as unary **connectives**, just like $\neg$. These connectives are formed from a quantifier ($\forall$ or $\exists$) and an argument, which is a variable.

Examples 2.1.9 Let us show some formulas.

1. $P_1^2(x_3, f_2^0)$, $P_3^1(f_3^1(x_2))$ and $E(x_3, f_3^1(x_2))$ are formulas because they are atomic formulas.
2. Since $P_1^2(x_3, f_2^0)$ is a formula, $\neg P_1^2(x_3, f_2^0)$ is also a formula by rule 2.
3. Since $P_1^2(x_3, f_2^0)$ and $P_3^1(f_3^1(x_2))$ are formulas, $(P_1^2(x_3, f_2^0) \wedge P_3^1(f_3^1(x_2)))$ is a formula by rule 4.
4. We can apply rule 4 to the formulas $\neg P_1^2(x_3, f_2^0)$ and $E(x_3, f_3^1(x_2))$ get the formula $(\neg P_1^2(x_3, f_2^0) \vee E(x_3, f_3^1(x_2)))$.
5. Since $P_3^1(f_3^1(x_2))$ and $E(x_3, f_3^1(x_2))$ are formulas, $(P_3^1(f_3^1(x_2)) \longrightarrow E(x_3, f_3^1(x_2)))$ is a formula by rule 5.
6. We get the formula $(P_1^2(x_3, f_2^0) \longleftrightarrow P_3^1(f_3^1(x_2)))$ by applying rule 6 to the formulas $P_1^2(x_3, f_2^0)$ and $P_3^1(f_3^1(x_2))$.
7. The formula $\forall x_3 E(x_3, f_3^1(x_2))$ is created by applying rule 7 to $E(x_3, f_3^1(x_2))$.
8. We get the formula $\exists x_2(P_3^1(f_3^1(x_2)) \longrightarrow E(x_3, f_3^1(x_2)))$ by applying rule 8 to the formula $(P_3^1(f_3^1(x_2)) \longrightarrow E(x_3, f_3^1(x_2)))$

Now let us define *subterms* and *subformulas*.

Definition 2.1.10 (subterm) *Let t be a term. A subterm of t is a substring t_1 of t that is also a term.*

Example 2.1.11 Let us find the subterms of $f_2^3(f_2^2(x_3, f_1^1(f_3^0)), f_2^1(x_3), f_5^0)$. They are $f_2^3(f_2^2(x_3, f_1^1(f_3^0)), f_2^1(x_3), f_5^0)$, $f_2^2(x_3, f_1^1(f_3^0))$, x_3, $f_1^1(f_3^0)$, f_3^0, $f_2^1(x_3)$, and f_5^0. We also notice that the subterm x_3 has two occurrences in the term. We distinguish these occurrences by calling them *the first occurrence* and *the second occurrence*.

Definition 2.1.12 (subformula) *Let F be a formula. A subformula of F is a substring of F that is also a formula.*

Example 2.1.13 Let us find the subformulas of $F = (\forall x_3(P_2^1(x_3) \wedge P_4^0) \longrightarrow (P_4^2(x_2, f_8^0) \vee \neg\exists x_4 P_3^1(x_4)))$. They are F itself, $\forall x_3(P_2^1(x_3) \wedge P_4^0)$, $(P_2^1(x_3) \wedge P_4^0)$, $P_2^1(x_3)$, P_4^0, $(P_4^2(x_2, f_8^0) \vee \neg\exists x_4 P_3^1(x_4))$, $P_4^2(x_2, f_8^0)$, $\neg\exists x_4 P_3^1(x_4)$, $\exists x_4 P_3^1(x_4)$, $P_3^1(x_4)$.

In many applications it is convenient to represent terms and formulas as trees.

Sweep through t and underline all variables x_i and constants f_i^0;
For each underlined symbol create a node with that symbol as its label;
while [t is not fully underlined] do
{
Sweep through t and underline each substring $f_i^n(t_1, \ldots, t_n)$ that has $t_1, \ldots, t_n$ underlined and the rest of the symbols unmarked;
Create a new node labeled f_i^n and make t_1 its first child,
t_2 its second child,..., and t_n its nth child.
}

Figure 2.2: The Term to Tree Algorithm

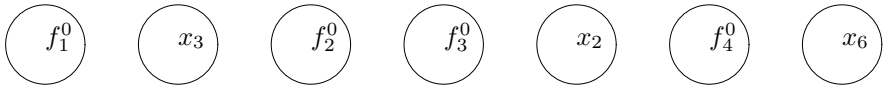


Figure 2.3: The trees from Example 2.1.15 after 1 sweep

Definition 2.1.14 (term tree) *A term tree is a labeled tree satisfying these conditions:*

1. The leaves are labeled with variables or constants.

2. The branches are labeled with functions of arity greater than 0; if the function has arity n, then it has n children.

Figure 2.2 shows an algorithm that converts a term t into a tree.

Example 2.1.15 Let us apply the algorithm from Figure 2.2 to $t = f_4^3(f_2^3(f_1^0, x_3, f_2^0), f_3^2(f_3^0, x_2), f_4^2(f_4^0, x_6))$.

After the first sweep through the string we get the underlines $f_4^3(f_2^3(\underline{f_1^0}, \underline{x_3}, \underline{f_2^0}), f_3^2(\underline{f_3^0}, \underline{x_2}), f_4^2(\underline{f_4^0}, \underline{x_6}))$ the trees from Figure 2.3.

After the second sweep through t we get the underlines $f_4^3(\underline{f_2^3(f_1^0, x_3, f_2^0)}, \underline{f_3^2(f_3^0, x_2)}, \underline{f_4^2(f_4^0, x_6)})$ and the trees from Figure 2.4.

After the third sweep the whole string t is underlined and we get the tree from Figure 2.5.

At times it is beneficial to represent formulas as trees.

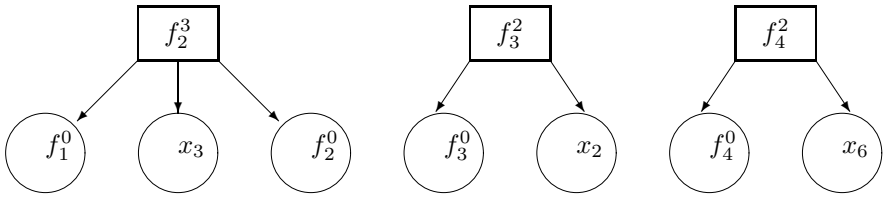


Figure 2.4: The trees from Example 2.1.15 after 2 sweeps

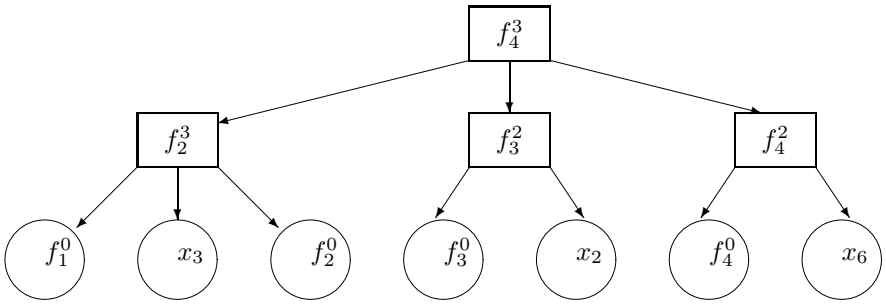


Figure 2.5: The Example 2.1.15 trees after 3 sweeps

Definition 2.1.16 (formula tree) *A formula tree is a labeled binary tree that satisfies the following conditions:*

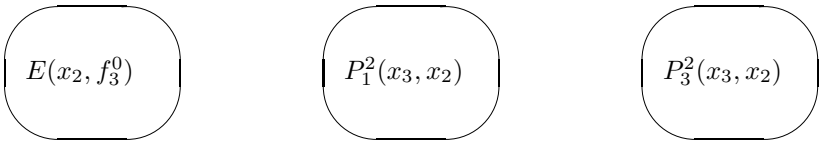
1. Every leaf is labeled with an atomic formula.

2. If a branch has only one child, then it is labeled with a unary connective $\neg$, $\forall x$, *or* $\exists x$; *if it has two children, its label is* $\vee$, $\wedge$, $\longrightarrow$, *or* $\longleftrightarrow$.

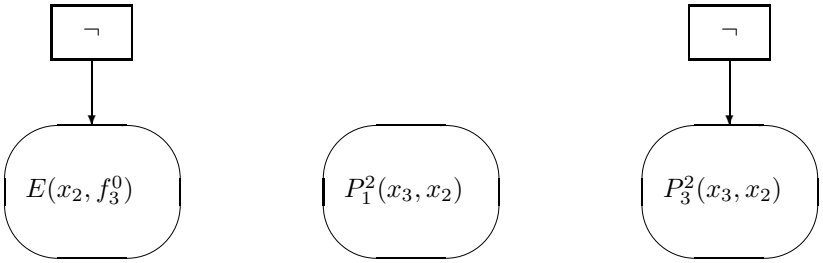
We can modify the algorithm from Figure 1.3 to generate formula trees from FOL formulas.

Example 2.1.17 Let us apply the algorithm from Figure 2.6 to find a tree representation of $F = \forall x_2(\neg E(x_2, f_3^0) \vee \exists x_3(P_1^2(x_3, x_2) \longrightarrow \neg P_3^2(x_3, x_2)))$.

After the first sweep we have the underlines $\forall x_2(\neg \underline{E(x_2, f_3^0)} \vee \exists x_3(\underline{P_1^2(x_3, x_2)} \longrightarrow \neg \underline{P_3^2(x_3, x_2)}))$ and the trees below.



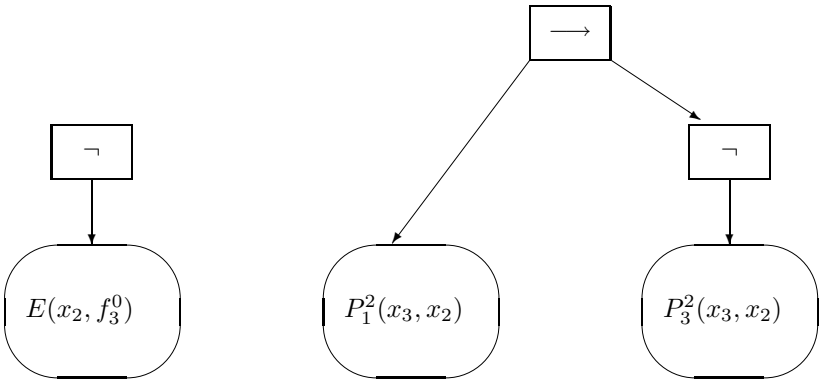
After the second sweep through F we get the underlines $\forall x_2(\underline{\neg E(x_2, f_3^0)} \vee \exists x_3(\underline{P_1^2(x_3, x_2)} \longrightarrow \underline{\neg P_3^2(x_3, x_2)}))$ and the 3 trees beneath.



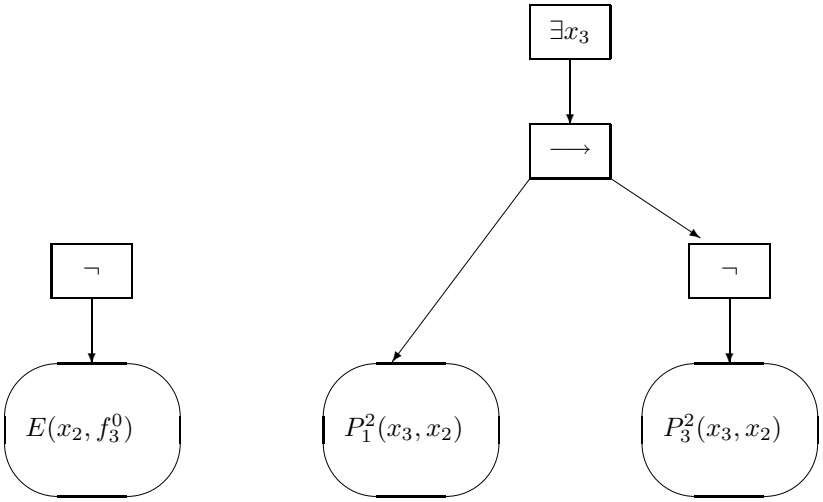
Sweep through the string F and underline each atomic formula. For each occurrence of a formula create a node with that formula as its label.
while (F is not fully underlined) do
{
Sweep through t and
1. underline each substring $\neg G$ that has G underlined and $\neg$ unmarked.
Create a new node labeled $\neg$ and make it point to the root of G.
2. underline each substring $(G \vee H)$ that has G and H underlined, the end parentheses and $\vee$ unmarked.
Create a new node labeled $\wedge$ and make G its first child and H its second child.
3. underline each substring $(G \wedge H)$ that has G and H underlined, the end parentheses and $\wedge$ unmarked.
Create a new node labeled $\wedge$ and make G its first child and H its second child.
4. underline each substring $(G \longrightarrow H)$ that has G and H underlined, the end parentheses and $\longrightarrow$ unmarked.
Create a new node labeled $\longrightarrow$ and make G its first child and H its second child.
5. underline each substring $(G \longleftrightarrow H)$ that has G and H underlined, the end parentheses and $\longleftrightarrow$ unmarked.
Create a new node labeled $\longleftrightarrow$ and make G its first child and H its second child.
6. underline each substring $\forall x G$ that has G underlined and $\forall x$ unmarked.
Create a new node labeled $\forall x$ and make it point to the root of G.
7. underline each substring $\exists x G$ that has G underlined and $\exists x$ unmarked.
Create a new node labeled $\exists x$ and make it point to the root of G.
}

Figure 2.6: An algorithm that converts FOL formulas to trees

After the third sweep we have the underlines $\forall x_2(\underline{\neg E(x_2, f_3^0)} \vee \exists x_3 \underline{(P_1^2(x_3, x_2) \longrightarrow \neg P_3^2(x_3, x_2))})$ and the 2 trees below.

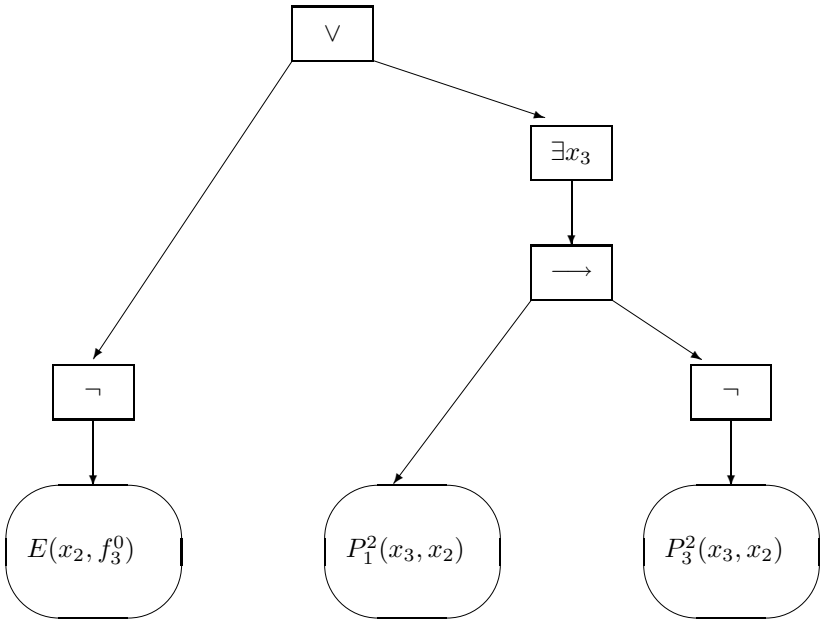


After the fourth sweep we get the underlines $\forall x_2(\underline{\neg E(x_2, f_3^0)} \vee \underline{\exists x_3(P_1^2(x_3, x_2) \longrightarrow \neg P_3^2(x_3, x_2))})$ and the trees beneath.

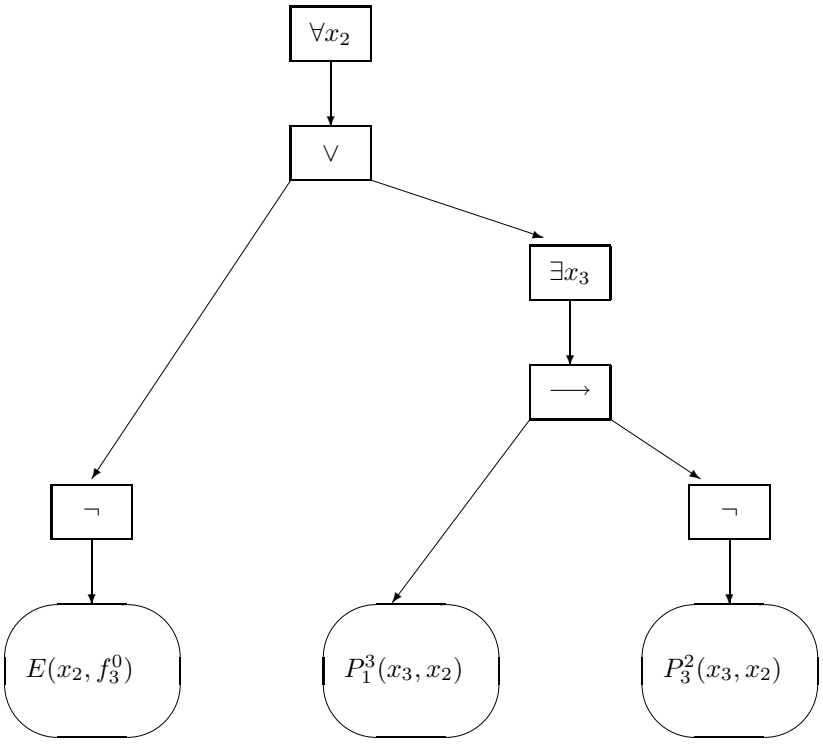


After the fifth sweep we get the underlines

$\forall x_2 \underline{(\neg E(x_2, f_3^0) \vee (\exists x_3(P_1^2(x_3, x_2) \longrightarrow \neg P_3^2(x_3, x_2)))}$ and only one tree.



After the sixth step the whole F is underlined and we obtain the formula tree.



We can modify the algorithm from Figure 1.8 to convert FOL formula trees to strings. All we have to do is to change the statement *write(¬);* of the *else* case to *write(label(Root));*.

Now let us introduce some notational conventions that will greatly simplify the notation.

Convention 2.1.18 1. We will use the lowercase letters x, y, z, u, v, w, z with or without subscript for individual variables.

2. We will use the lowercase letter a, b, c, d, e with or without subscripts to denote individual constants.

3. We will use the lowercase letters f, g, h with or without subscripts for function symbols. We will not write the arity when it can be deduced from the context.

4. We will use the uppercase letters F,G, H, I, J, with or without subscripts, for formulas.

5. We will use the uppercase letters P, Q, R S, with or without subscripts, for predicate symbols. We will skip the arity when it can be determined from the context.

6. We will use the conventions from Section 1.1 regarding the omission of the outer parentheses and the use of $\bigwedge_{i=m}^{n} F_i$ and $\bigvee_{i=1}^{n} F_i$.

Now we will present the notions of *the scope of a quantifier*, *free* and *bound* occurrences, *free* and *bound* variables. Intuitively, the scope of a quantifier $\forall x$

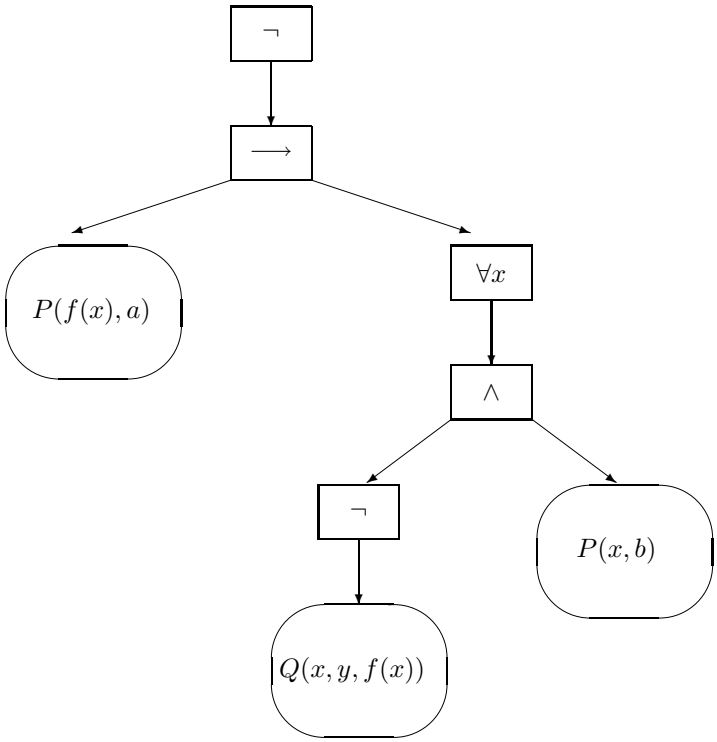


Figure 2.7: The scope of the quantifier ∀

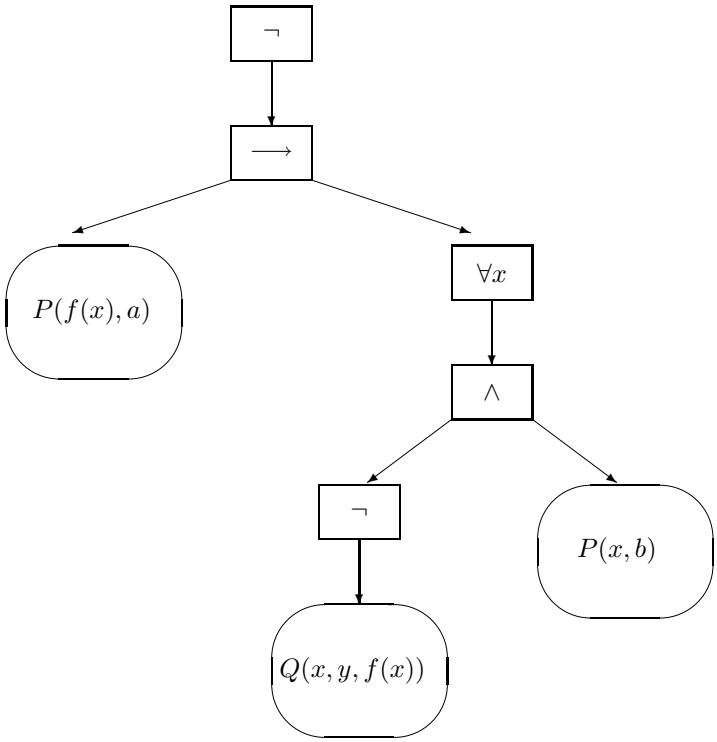
or $\exists x$ is the set of subformulas where the quantifier operates. In a tree formula, the scope of the quantifier at address A is the set of all subtrees whose roots are the children of A. For example, the scope of the quantifier $\forall x$ of Figure 2.7 is the subtree with the root labeled $\wedge$.

Definition 2.1.19 (bound and free occurrences) *We say that an occurrence of the variable x in a formula tree F is bound if it is under the scope of a quantifier $\forall x$ or $\exists x$.*

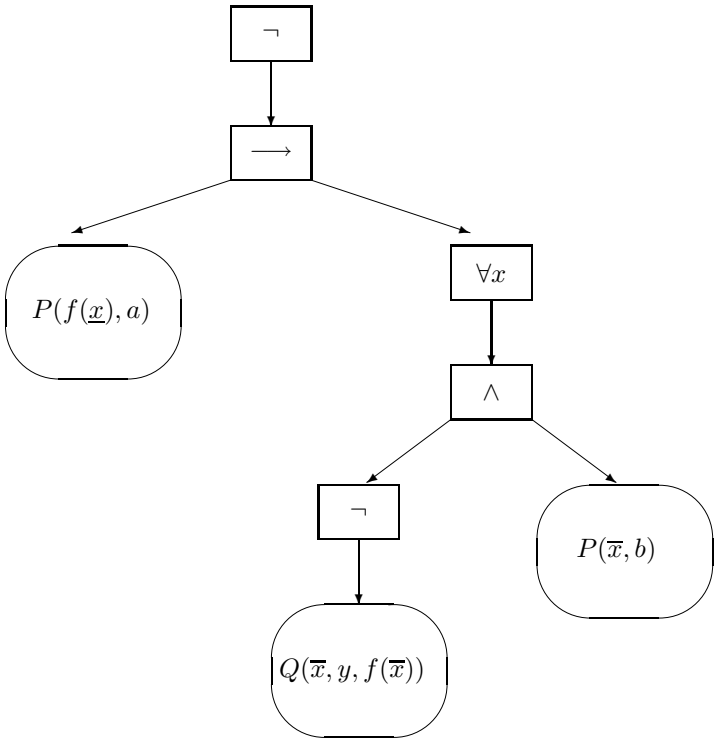
An occurrence is free when it is not bound.

Note 2.1.20 By variable occurrence we mean an individual variable that occurs in an atomic formula. The variable that follows the quantifier is *the argument* of the quantifier. So, the strings $\forall x$ and $\exists x$ are *operators* composed of a quantifier and an argument.

Example 2.1.21 Let us find whether the variable occurrences of the formula below are free or bound.



We recall that the scopes of the quantifiers $\forall x$ and $\exists x$ are the subtrees of their children and the variable occurrences are in the leaves of the tree. So, the variable occurrence x is free if that leaf has no ancestor node labeled $\exists x$ or $\forall x$. By inspecting the tree we see that the underlined occurrences are free while the overlined occurrences are bound.



Definition 2.1.22 (free variable, closed formula) *We say that a variable x is free in the formula tree F if it has free occurrences.*

A formula that has no free variables is called a closed formula.

Example 2.1.23 1. The variable x in Figure 2.8 is free because it has a free occurrence in $Q(x)$. The variable y is bound because all its occurrences are bound.

2. The formula tree from Figure 2.9 is closed because it has no free variables.

Exercises

Exercise 2.1.1 *Write an algorithm that takes as input a string and writes atomic formula if the string is an atomic formula and not an atomic formula if the string is not an atomic formula.*

Exercise 2.1.2 *List the subterms of $f_1^3(f_{11}^4(x_4, f_1^1(f_{23}^0), f_3^0, x_4), f_3^1(f_2^2(x_3, f_8^0)), x_5)$.*

Exercise 2.1.3 *List all subformulas of $(\neg(P_3^2(x_3, f_1^0) \wedge E(x_3, f_3^0)) \longrightarrow \forall x_2 \exists x_8 (E(x_8, x_2) \vee (P_3^1(x_2) \longleftrightarrow P_2^3(f_4^0, x_8, x_2))))$.*

Exercise 2.1.4 *Apply the algorithm from Figure 2.2 to generate the tree representation of $f_2^1(f_3^4(x_2, f_6^3(f_3^0, f_4^2(x_3, f_5^0), x_5), f_3^1(f_2^2(x_0, f_3^0)), x_4))$.*

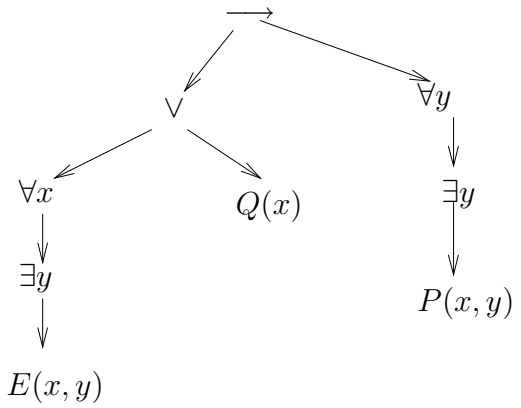


Figure 2.8: A formula tree with free variables

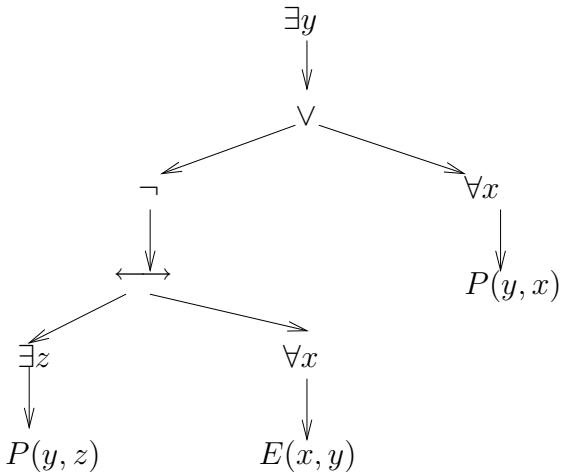
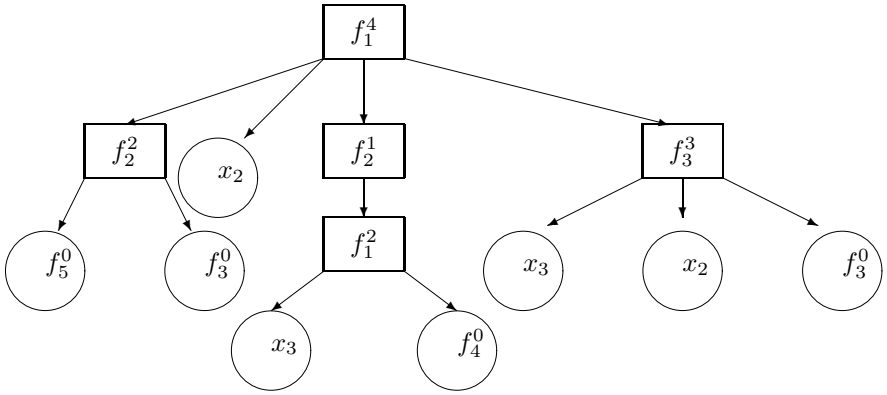


Figure 2.9: A closed formula tree

Exercise 2.1.5 *Write an algorithm that takes as input the tree representation of a term and outputs its string representation. Test it on the tree below.*

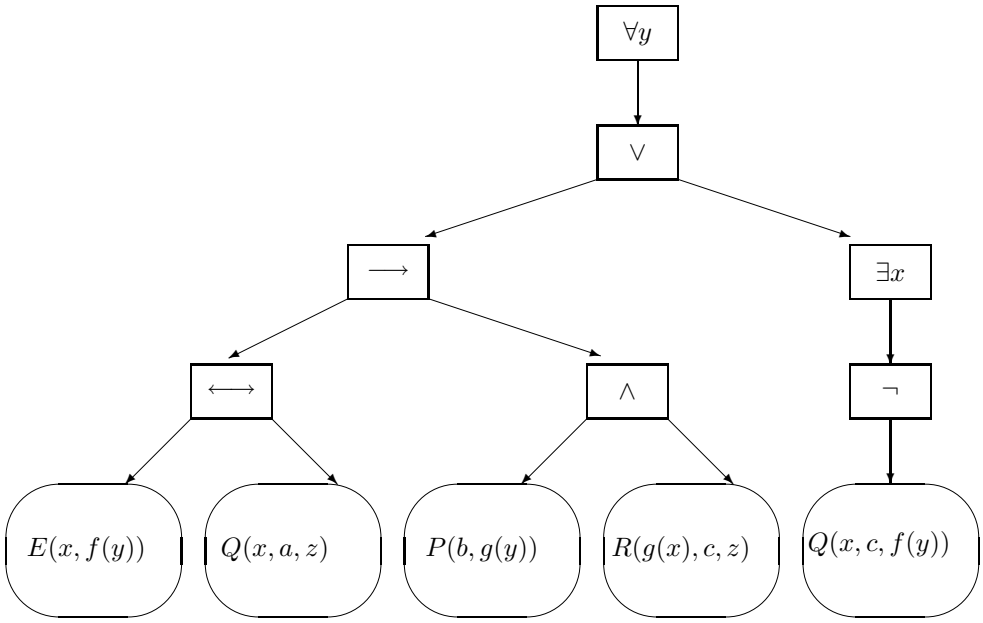


Exercise 2.1.6 *Define the scope of a quantifier for formulas. Then, specify if the variable occurrences of the formulas*

1. $F = \neg(\forall x(P(x,y) \longrightarrow \exists y \neg P(y,x)) \wedge \exists y P(y,x))$
2. $G = \forall y(\exists x P(x, f(y)) \longleftrightarrow (\forall y \neg P(x,y) \vee P(g(y,z),x)))$

are free or bound.

Exercise 2.1.7 *Specify if the variable occurrences in the tree below are free or bound.*



Exercise 2.1.8 *Translate the following English statements into FOL formulas:*

1. All students are smart.

2. Some students are smart.

3. The pairing axiom from set theory: For every two sets there is a set that contains exactly these sets as members.

4. The extensionality axiom from set theory: Two sets are equal iff they have the same members.

5. The following theorem from number theory: Every non-zero number is the successor of some number.

6. Every person has a mother.

7. Every woman is loved by some man.

Use the predicates student(x) for x is a student, smart(x) for x is smart, $\in$ for belongs to, E for the equality, person(x) for x is a person, mother(x,y) for x is the mother of y, woman(x) for x is a woman, man(x) for x is a man. Also use the function Sx for the successor of x.

2.2 Structural Induction in First Order Logic

In this section we will learn to do inductive proofs in first order logic. We suggest a review of Sections 1.2 and 2.1 for the description of the method and the definitions of terms and formulas. Just like we did in Section 1.2, we will state the structural induction theorems and then we will work our way towards The Unique Readability Theorem, needed to define the semantics of formulas.

In first order logic we have two kinds of strings, terms and formulas. So, we should be able to do induction on term structure as well as on formula structure.

Theorem 2.2.1 (The Structural Induction on Terms) *A proposition $P[t]$ is true for all terms t if and only if*

1. $P[t]$ is true when t is a variable.

2. $P[t]$ is true when t is an individual constant.

3. If f is a function of arity $n > 0$, $t_1,\ldots,$ t_n are terms, and $P[t_1],\ldots,$ $P[t_n]$ are true then $P[f(t_1,\ldots,t_n)]$ is true.

Example 2.2.2 Before we go on to prove the theorem let us see how it works. Assume that P is some property that satisfies the 3 conditions listed in Theorem 2.2.1. Let us see how these 3 rules enable P to satisfy $i(f(a,x,b),g(c,y),h(d,z))$, shown in Figure 2.10. The first rule tells us that P is true for all variables. So, we put a P to the right of every circle labeled with a variable, as shown in Figure 2.11.

Next we apply rule 2 that tells us to label with P all individual constants, i.e. all circles that contain a, b, c, and d. We get the tree from Figure 2.12.

Now we can apply rule 3 of Theorem 2.2.1, that tells us that whenever *all* children of a branch have the P label, we can label the branch. So, we can put P to the right of the boxes that contain the function symbols f, g, and h as shown in Figure 2.13. Figure 2.14 shows the term after the next application of rule 3. This example shows us that the rules 1 and 2 label the leaves, and that

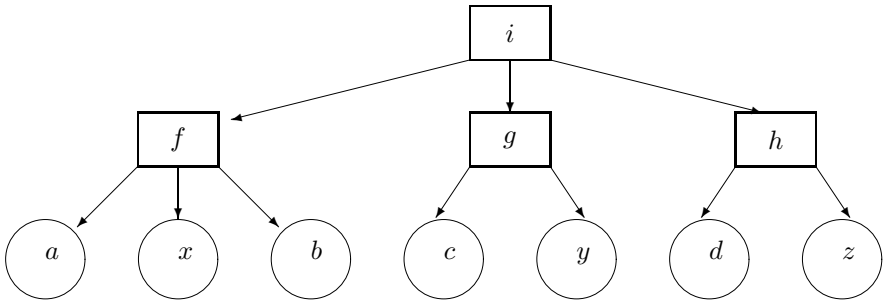


Figure 2.10: The term from Example 2.2.2

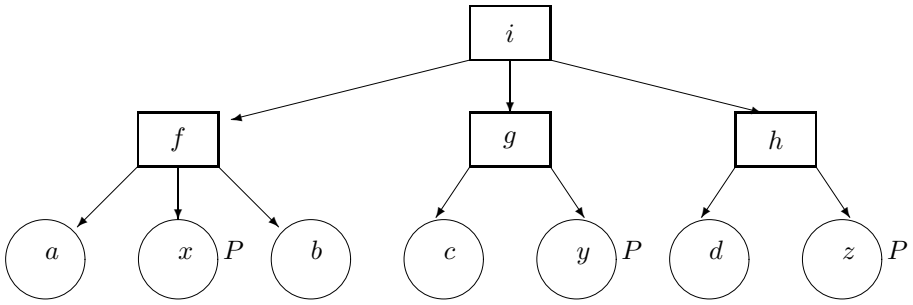


Figure 2.11: We label the variable of the term from Example 2.2.2

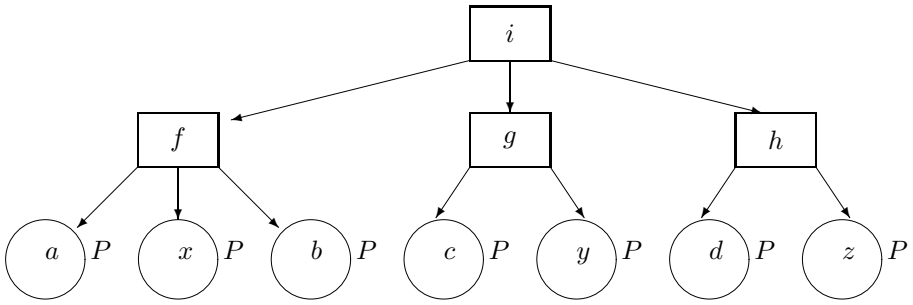


Figure 2.12: We label the constants of the term from Example 2.2.2

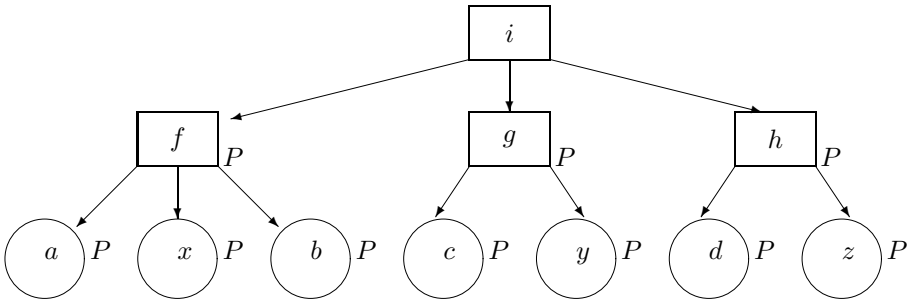


Figure 2.13: We label some branches of the term from Example 2.2.2

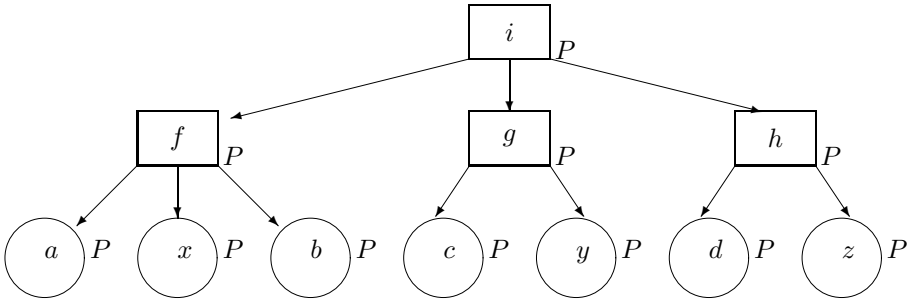


Figure 2.14: We label the root of the term from Example 2.2.2

rule 3 labels the branches.

Now let us return to the proof of Theorem 2.2.1.
Proof: We have to show that

(1) if $P[t]$ is true for all terms t then the conditions (rules) of Theorem 2.2.1 hold, and

(2) if the rules in Theorem 2.2.1 hold, then $P[t]$ is true for all t.

(1) is easy. Since $P[t]$ is true for all t then it is true for the cases when
-t is a variable, so we have condition 1.
-t is an individual constant, so we have condition 2.
-$t = f(t_1, \ldots, t_n)$, so we have condition 3.

We will prove (2) by contradiction. Assume that P satisfies the conditions 1, 2, and 3, but there are terms t for which $P[t]$ is false. Let t_0 be one of these terms that *has minimal length*, i.e. has the fewest number of symbols. Let's look at the structure of t_0.

t_0 cannot be variable since P satisfies condition 1.

t_0 cannot be an individual constant since P satisfies condition 2.

t_0 has the form $t_0 = f(t_1, \ldots, t_n)$. If P is true for all the terms $t_1, \ldots, t_n$, then P is true for t_0 by condition 3. So, P must be false for at least one of the terms $t_1, \ldots, t_n$. But then that term **has smaller length than** t_0, which contradicts our hypothesis. **Q.E.D.**

Now let us prove two lemmas that characterize the substrings of terms.

Lemma 2.2.3 *Let S be a prefix of a term τ and $n[fun, S]$ the number of function symbols of arity greater than 0 that occur in S. Then $n[fun, S] \geq n[(, S] \geq n[), S]$.*

Proof: We use structural induction on terms. If τ is a constant or a variable, then it has a single symbol and that character is neither a function of arity greater than 0, nor a parenthesis. So, $n[fun, S] = n[(, S] = n[), S] = 0$.

Now let $\tau = f(\tau_1, \ldots, \tau_n)$. We have 4 cases, $S = \lambda$, $S = f$, $S = f(\tau_1, \ldots, \tau_{i-1}, S_i$ with S_i a prefix of τ_i, and $S = \tau$. If $S = \lambda$, then $n[fun, S] = n[(, S] = n[), S] = 0$, so the double inequality holds.

If $S = f$ then $n[fun, S] = 1$ and $n[(, S] = n[), S] = 0$, so the inequality is true.

If $S = f(\tau_1, \ldots, \tau_{i-1}, S_i$, we have the derivations below.

$n[fun, S] = 1 + n[fun, \tau_1] + \ldots + n[fun, \tau_{i-1} + n[fun, S_i]$ $S = f(\tau_1, \ldots, \tau_{i-1}, S_i$

$\geq 1 + n[(, \tau_1] + \ldots + n[fun, \tau_{i-1} + n[fun, S_i]$ by IH applied to $\tau_1, \ldots, \tau_{i-1}$, S_i

$= n[(, S]$ $S = f(\tau_1, \ldots, \tau_{i-1}, S_i$

$n[(, S] = 1 + n[(, \tau_1] + \ldots + n[fun, \tau_{i-1} + n[fun, S_i]$ $S = f(\tau_1, \ldots, \tau_{i-1}, S_i$

$> n[(, \tau_1] + \ldots + n[(, \tau_{i-1} + n[(, S_i]$

$\geq n[), \tau_1] + \ldots + n[), \tau_{i-1} + n[), S_i]$ by IH applied to $\tau_1, \ldots, \tau_{i-1}$, S_i

$= n[), S]$ $S = f(\tau_1, \ldots, \tau_{i-1}, S_i$

From these two inequalities we get that $n[fun, S] \geq n[(, S] > n[), S]$.

The fourth case, $S = \tau$, is proved in the same fashion, by applying the induction hypothesis to the prefixes $\tau_1,\ldots,\tau_n$ of $\tau_1,\ldots,\tau_n$. It is left as exercise. **Q.E.D.**

We can state a similar result for suffixes.

Lemma 2.2.4 *If S is a suffix of a term, then $n[(, S] \geq n[), S] \geq n[fun, S]$.*

The proof of this lemma is left as exercise.

Corollary 2.2.5 *For every term τ, $n[fun, \tau] = n[(, \tau] = n[), \tau]$.*

Proof: The term τ is both a prefix and a suffix of τ. So,
(1) $n[fun, \tau] \geq n[(, \tau] \geq n[), \tau]$
by Lemma 2.2.3 and
(2) $n[), \tau] \geq n[(, \tau] \geq n[fun, \tau]$
by Lemma 2.2.4. From (1) and (2) we get the desired equality. **Q.E.D.**

Lemma 2.2.6 *Let S be a prefix of a term τ. If S is also a term, then $S = \tau$.*

Proof: We use induction on terms. If τ is a variable or an individual constant, then it has a single symbol. So, $S = \lambda$ or $S = \tau$. Since λ is not a term we are left with $S = \tau$.

Now let us assume that $\tau = f(\tau_1, \ldots, \tau_n)$ for some $n > 0$. Let S be a prefix of τ. We have cases, $S = \lambda$, $S = f$, $S = f(\tau_1, \ldots, \tau_{i-1}, S_i$ with S_i a prefix τ_i, and $S = \tau$. We reject $S = \lambda$ because every term has length greater than 0. We also turn down $S = f$, because f is not a 0-ary function, so S is not a term. We will also show that $S = f(\tau_1, \ldots, \tau_{i-1}, S_i$, with S_i a prefix of τ_i, is not a formula because it has more left parentheses than right parentheses, contradicting Corollary 2.2.5.

$n[(, S] = 1+n[(, \tau_1]+\ldots+n[(, \tau_{i-1}]+n[(, S_i]$ because $S = f(\tau_1, \ldots, \tau_{i-1}, S_i$
$> n[(, \tau_1] + \ldots + n[(, \tau_{i-1}] + n[(, S_i]$
$\geq n[), \tau_1] + \ldots + n[), \tau_{i-1}] + n[), S_i]$ by Lemma 2.2.3
$= n[(, S]$ because $S = f(\tau_1, \ldots, \tau_{i-1}, S_i$

So, we are left with $S = \tau$, i.e. S is the whole term. **Q.E.D.**

We can now prove the unique readability of terms.

Theorem 2.2.7 (The Unique Readability of Terms Theorem) *Every term is either a constant, a variable, or it has the form $f(t_1, \ldots, t_n)$ with $n > 0$ and the subterms $t_1 \ldots, t_n$ uniquely defined.*

Proof: First of all the 3 categories, constant, variable, and composed term are distinct since they start with different characters. Let us assume that $f(t_1, \ldots, t_n) = g(\tau_1, \ldots, \tau_m)$. The two strings are equal, so their first character is the same. This means that $f = g$, and $m = n$ because the symbol f has a unique arity. Now, let us assume that for some indices $1 \leq i \leq n$, $t_i \neq \tau_i$. Let j be the smallest such index. Then

$f(t_1, \ldots t_{j-1}, t_j, \ldots, t_n) = f(t_1, \ldots t_{j-1}, \tau_j, \ldots, \tau_n)$

The strings t_j and τ_j start in the same position, so either t_j is a prefix of τ_j or τ_j is a prefix of t_j. Since both are terms, Lemma 2.2.6 tells us that they are

1. $P[F]$ is true when P is an atomic formula.
2. If $P[F]$ is true then $P[\neg F]$ is true.
3. If $P[F]$ and $P[G]$ are true then $P[F \vee G]$ is true.
4. If $P[F]$ and $P[G]$ are true then $P[F \wedge G]$ is true.
5. If $P[F]$ and $P[G]$ are true then $P[F \longrightarrow G]$ is true.
6. If $P[F]$ and $P[G]$ are true then $P[F \longleftrightarrow G]$ is true.
7. If $P[F]$ is true then $P[\forall x F]$ is true.
8. If $P[F]$ is true then $P[\exists x F]$ is true.

Figure 2.15: The structural induction rules for formulas

equal. So, we got a contradiction, and we conclude that $t_i = \tau_i$ for all $1 \leq i \leq n$. **Q.E.D.**

The unique readability of atomic formulas is proved in the same fashion.

Lemma 2.2.8 (The Unique Readability of Atomic Formulas Lemma) *Every atomic formula is either a predicate constant, an equality $E(t_1, t_2)$ with t_1 and t_2 uniquely defined, or a formula $P(t_1, \ldots, t_n)$ with $n > 0$ and $t_1, \ldots, t_n$ uniquely defined.*

The proof is similar to the one for Theorem 2.2.7. It is left as exercise.

Now let us present the structural induction for formulas. Just like we did for the terms, we need a rule that takes care of each the 8 cases of Definition 2.1.7. We get the rules from Figure 2.15.

Rule 1 tells us that P must take care of the atomic formulas, i.e. the leaves of the formula tree. The next 7 rules tell us that whenever P is true for all children of a branch, it is also true for the branch. We have 7 cases because the label of the branch belongs to one of the 7 categories $\neg$, $\wedge$, $\vee$, $\longrightarrow$, $\longleftrightarrow$, $\forall x$, $\exists x$). The last seven rules are *propagation rules* since they propagate the property P towards the root of the tree.

Example 2.2.9 Let us assume that property P satisfies the 8 rules of Figure 2.15 and let us prove that P holds for the formula $\forall y(((E(x, f(x)) \longleftrightarrow Q(x, a, z)) \longrightarrow (P(b, g(y)) \wedge R(g(x), c, z))) \vee \exists x \neg Q(x, c, f(y)))$ shown in Figure 2.16. We apply rule 1 of Figure 2.15 that tells us that P is true for all atomic formulas and get Figure 2.17. We use rules 6, 4 and 2, to label the $\longleftrightarrow$, $\wedge$, and $\neg$ nodes. We can do this because *all* their children have the P label. We obtain the marked tree from Figure 2.18. Further on, we apply rule 5 to $\longrightarrow$ and rule 8 to $\exists x$, and we get Figure 2.19. Next, we mark the $\vee$ node and obtain Figure 2.20. Finally, we apply rule 7 to label the $\forall y$ node, as shown in Figure 2.21.

We notice that this labeling process starts from the leaves and moves up, level by level, towards the root. We recall that a P attached to a node signifies that P is true for the subformula rooted at that node. So, the labeling is a bottom-up procedure because it starts with the leaves and works its way up towards the root.

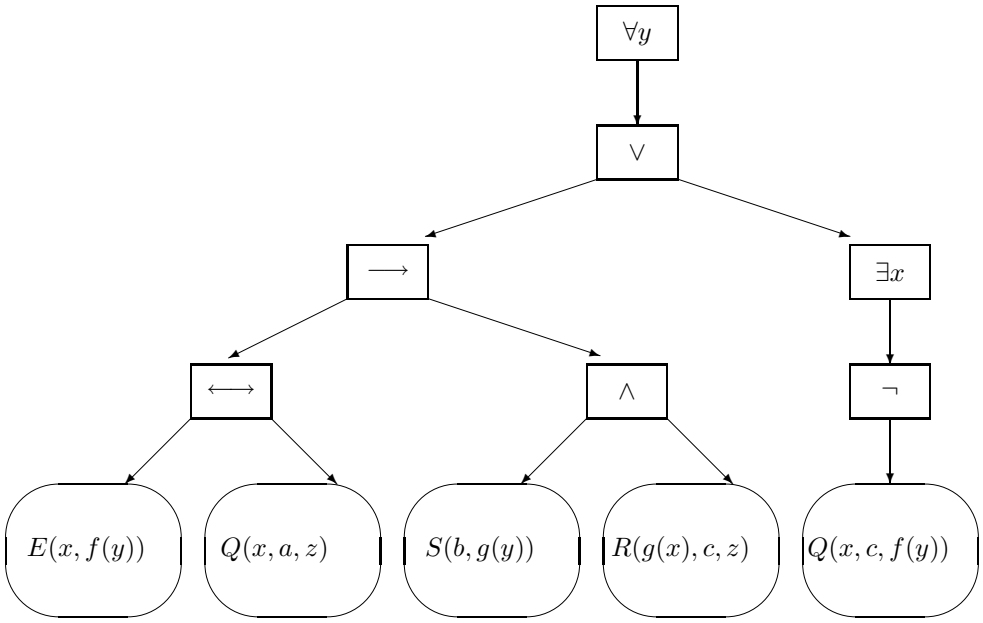


Figure 2.16: The formula from Example 2.2.9

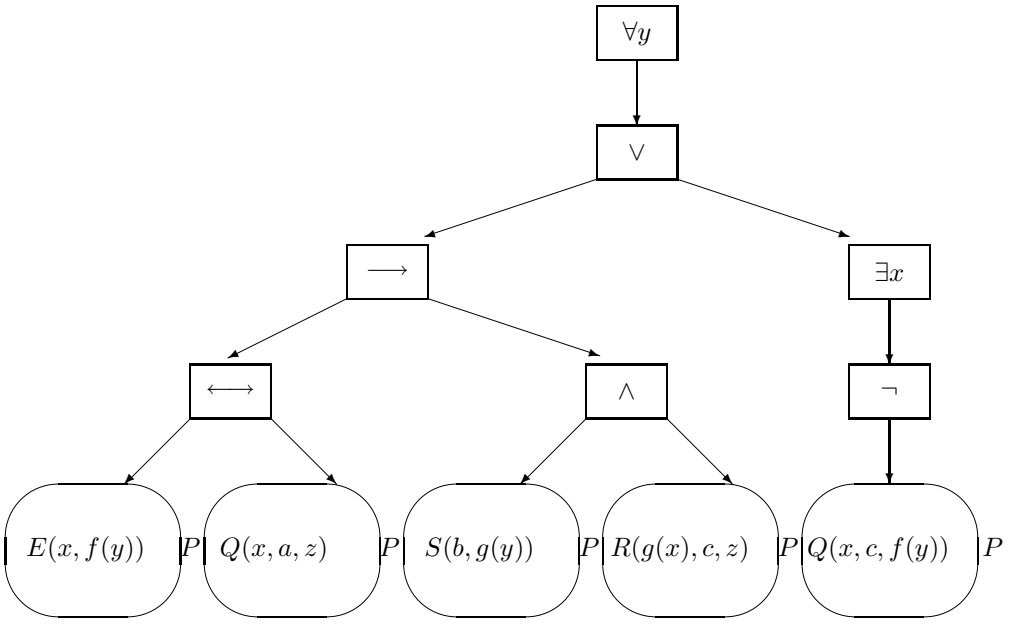


Figure 2.17: We label the leaves of the tree from Example 2.2.9

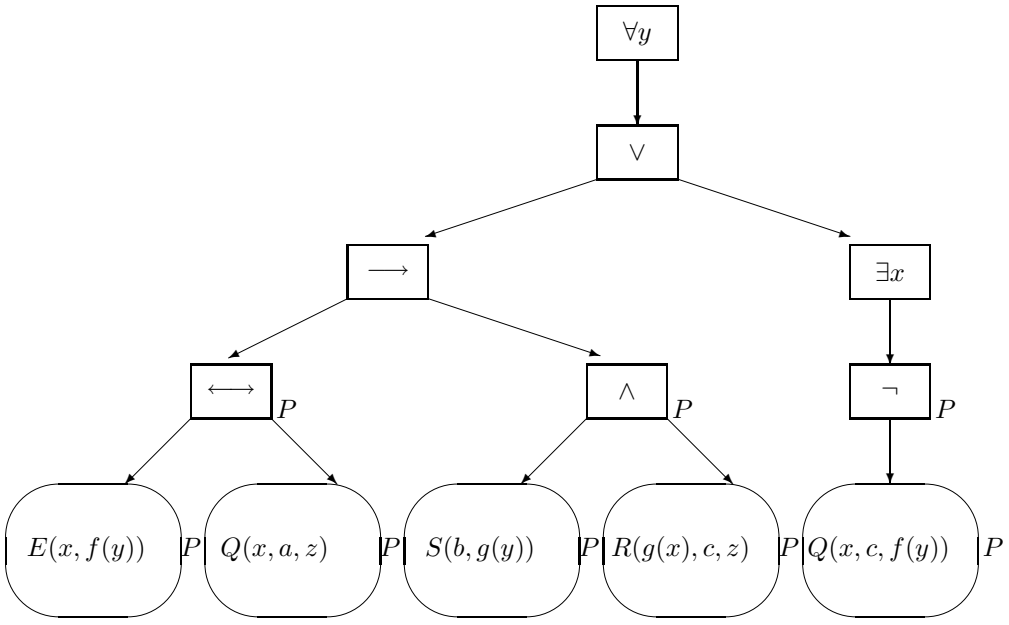


Figure 2.18: We label some branches of the tree from Example 2.2.9

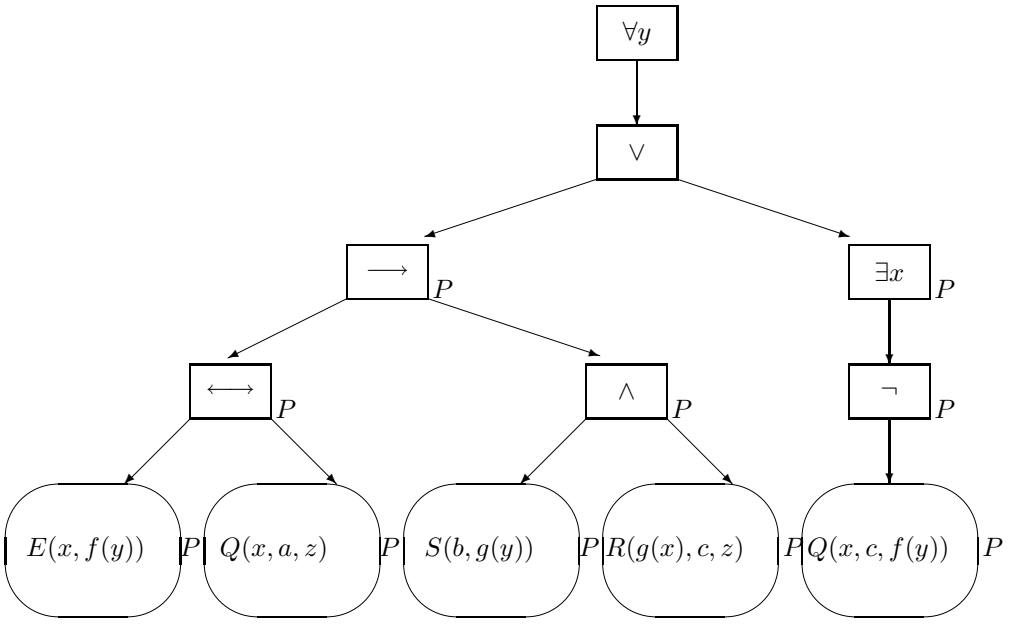


Figure 2.19: We label more branches of the tree from Example 2.2.9

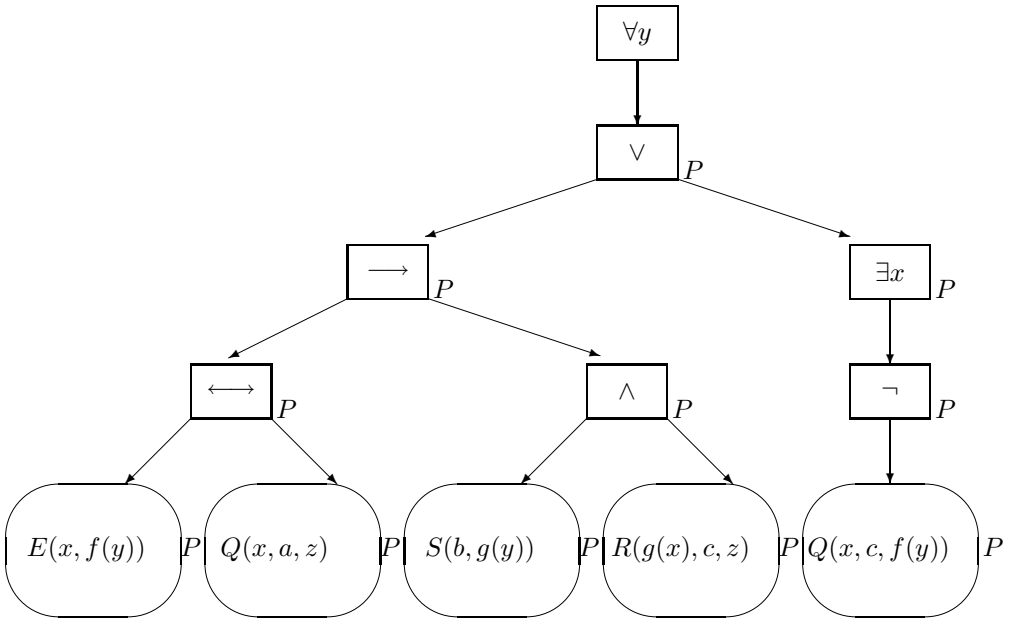


Figure 2.20: The tree from Example 2.2.9 after we label the $\vee$ node

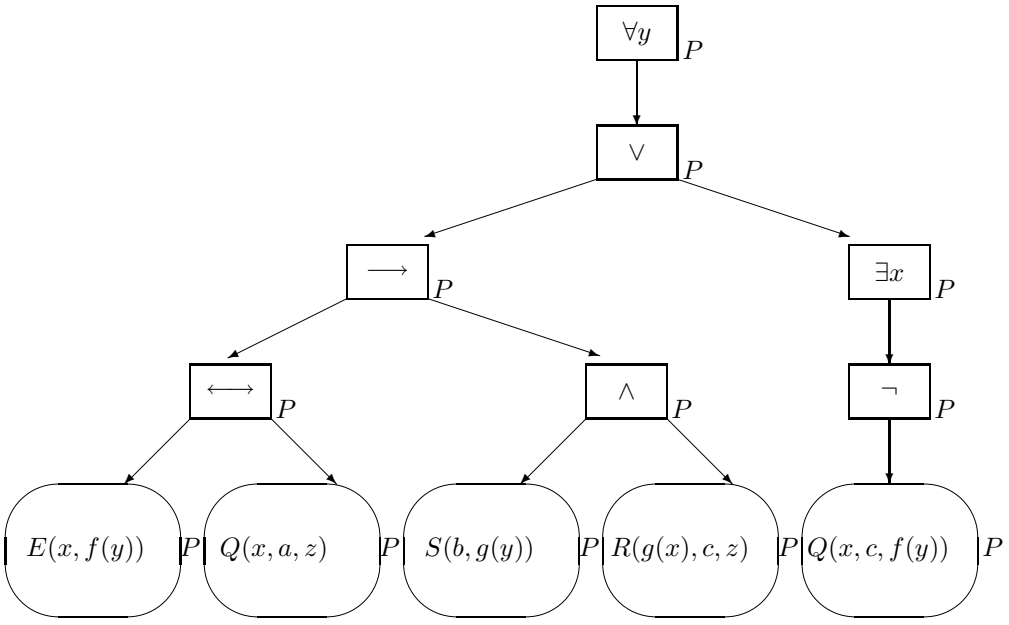


Figure 2.21: All nodes of the tree from Example 2.2.9 are labeled

Theorem 2.2.10 (The Structural Induction Theorem) *Let $P[F]$ be a property of the formulas of first order logic. Then $P[F]$ is true for all F if and only if it satisfies the 8 conditions from Figure 2.15.*

Proof: The theorem says that $P[F]$ is true for all F if and only if P satisfies the 8 conditions of Figure 2.15, we need to prove the two implications

(1) if $P[F]$ is true for all F then P satisfies the conditions of the table,

and

(2) if $P[F]$ satisfies the conditions of the table then $P[F]$ is true for all formulas F.

(1) is automatic. If $P[F]$ is true for all formulas F then it is true for all formulas of the type

atomic, $\neg G$, $G \vee H$, $G \wedge H$, $G \longrightarrow H$, $G \longleftrightarrow H$, $\forall xG$ and $\exists xG$. This tells us the 8 conditions of Figure 2.15 are fulfilled. For example, condition 4,

4. If $P[F]$ and $P[G]$ are true then $P[F \vee G]$ is true.

is satisfied because P is true for all formulas of the type $P[F \vee G]$, so the implication

Any condition then $P[F \vee G]$ *is true*

is true.

Now let us prove the second part. Assume that P satisfies the conditions listed in Figure 2.15. The proof is by contradiction.

Let us assume that $P[F]$ is false for some formulas F. Then among these formulas there is one that *has the smallest length*, i.e. the fewest number of symbols. Let F_0 be such a formula (it may not be unique since there are many formulas with the same length).

Then F_0, belongs to one of the 8 syntactic categories listed in Definition 2.1.7, i.e. F_0 is either atomic or has one of the forms $\neg G_0$, $G_0 \vee H_0$, $G_0 \wedge H_0$, $G_0 \longrightarrow H_0$, $G_0 \longleftrightarrow H_0$, $\forall xG_0$, $\exists xG_0$. We will show that the 8 rules from Figure 2.15 prevent F_0 from being in any of these categories.

For example, F_0 cannot be atomic since condition 1 tells us that P is true for atomic formulas.

If $F_0 = \neg G_0$ then condition 2 tells us that $P[G_0]$ must be false. But this contradicts the fact that F_0 is a shortest formula that falsifies P, since $length[G_0] < length(F_0)$.

The other cases ($F_0 = G_0 \vee H_0$, $F_0 = G_0 \wedge H_0$, $F_0 = G_0 \longrightarrow H_0$, $F_0 = G_0 \longleftrightarrow H_0$, $F_0 = \forall xG_0$, $F_0 = \exists xG_0$) are treated the same way. They are left as exercise.**Q.E.D.**

Now let us characterize the formulas in terms of their prefixes.

Lemma 2.2.11 *If S is a prefix of a formula F, then $n[(, S] \geq n[), S]$.*

Proof: We use structural induction.

Case 1: F is an atomic formula. Then we have 3 subcases.

Subcase 1.1: $F = P$, a predicate constant. Then its prefixes are $S = \lambda$ and $S = P$. For both categories $n[(, S] = n[), S] = 0$, so the lemma holds.

Subcase 1.2: $F = P(t_1, \ldots, t_n)$. Then the prefixes of F fall into one of the categories $S = \lambda$, $S = P$, $S = P(t_1, \ldots, t_{i-1}, \tau_i$, where τ_i is a prefix of t_i, and $S = F$. For the first 2 categories we have $n[(, S] = n[), S] = 0$, so the lemma holds. For the next two we apply Lemma 2.2.3 that tells us that every prefix of a term has at least as many left parentheses as right parentheses. For the first category we have the derivation below.

$n[(, S] = 1 + n[(, t_1] + \ldots + n[(, t_{i-1}] + n[(, \tau_i]$ since $S = P(t_1, \ldots, t_{i-1}, \tau_i$
$> n[(, t_1] + \ldots + n[(, t_{i-1}] + n[(, \tau_i]$
$\geq n[), t_1] + \ldots + n[), t_{i-1}] + n[), \tau_i]$ by Lemma 2.2.3
$= n[), S]$ since $S = P(t_1, \ldots, t_{i-1}, \tau_i$

The next derivation shows that the lemma holds for $S = F$.

$n[(, S] = 1 + n[(, t_1] + \ldots + n[(, t_n]$ since $S = P(t_1, \ldots, t_n)$
$\geq 1 + n[), t_1] + \ldots + n[), t_n]$ by Lemma 2.2.3
$= n[), S]$ since $S = P(t_1, \ldots, t_n)$

Subcase 1.3: $F = E(t_1, t_2)$. This subcase is similar to the preceding one. It is left as exercise.

Cases 2,3,4,5,6 are proved the same way as the corresponding cases of Lemma 1.2.7.

Cases 7, 8: $F = QxG$ where $Q \in \{\forall, \exists\}$ and x is a variable. The prefixes of F fall into one of the following categories: $S = \lambda$, $S = Q$, $S = QxI$ with I a prefix of G. For the first 2 categories, $n[(, G] = n[), G] = 0$. For the third one, the inequality is established by the derivation below.

$n[(, S] = n[(, I]$ since $S = QxI$
$\geq n[), I]$ by the IH applied to the prefix I of G
$= n[), S]$ since $S = QxI$ **Q.E.D.**

The next lemma characterizes the suffixes of a formula.

Lemma 2.2.12 *For all suffixes S of F, $n[), S] \geq n[(, S]$.*

Proof: We use structural induction on F.

Case 1: F is an atomic formula. We have 3 subcases.

Subcase 1.1:$F = P$. The suffixes of F are $S = \lambda$ and $S = P$. In both cases $n[(, S] = n[(, S] = 0$.

Subcase 1.2:$F = P(t_1, \ldots, t_n)$. Then, the suffixes of F fall into one of the categories $S = \lambda$, $S = S_i, \ldots, t_n)$ with S_i a suffix of t_i, $S = (t_1, \ldots, t_n)$, and $S = F$. For the empty string, $n[), \lambda] = n[(, \lambda] = 0$. The derivation below establishes the inequality for the second category.

$n[), S] = n[), S_i] + \ldots + n[), t_n] + 1$ because $S = S_i, \ldots, t_n)$
$> n[(, S_i] + \ldots + n[(, t_n]$
$\geq n[(, S_i] + \ldots + n[(, t_n]$ by Lemma 2.2.4
$= n[(, S]$ because $S = S_i, \ldots, t_n)$

The proof for the 3rd category, $S = (t_1, \ldots, t_n)$ is similar.

$n[), S] = n[), S_i] + \ldots + n[), t_n] + 1$ because $S = (t_1, \ldots, t_n)$
$\geq n[(, S_i] + \ldots + n[(, t_n] + 1$ by Lemma 2.2.4
$= n[(, S]$ because $S = (t_1, \ldots, t_n)$

The proof for $S = F$ is the same as the above.

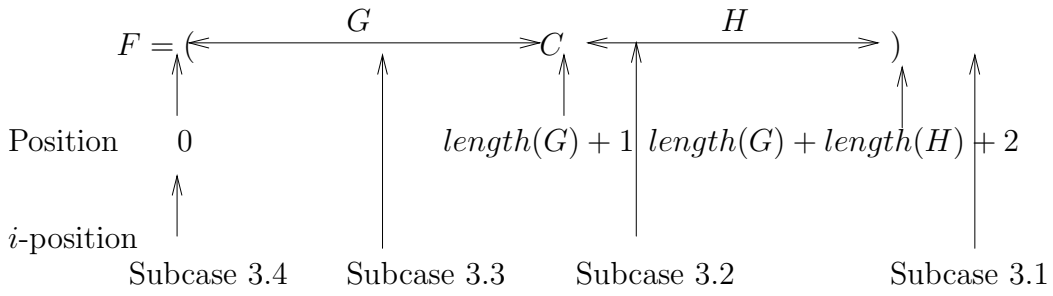


Figure 2.22: The subcases of Case 3 of Lemma 2.2.12

Subcase 1.3: $F = E(t_1, t_2)$. The suffixes of F fall into one of the categories $S = \lambda$, $S = S_2$) with S_2 a suffix of t_2, $S = S_1, t_2$) with S_1 a suffix of t_1, $S = (t_1, t_2)$ and $S = F$. We leave the proof to the reader.

Case 2: $F = \neg G$. Here we have two subcases.

Subcase 2.1: S is a suffix of G. Then we apply the induction hypothesis to the suffix S of G. So, $n[), S] \geq n[(, S]$.

Subcase 2.2: $S = \neg G$. Then

$n[), S] = n[), G]$ since $S = \neg G$

$\geq n[(, G]$ by the IH on the suffix G of G

$= n[(, G]$ since $S = \neg G$

Cases 3,4,5,6: $F = (GCH)$ where C is one of the connectives $\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$. Figure 2.22 shows the 4 subcases by displaying the first position of S. They are $S = \lambda$, $S = J$) with J a suffix of H, $S = ICH$) with I a suffix of G, and $S = F$. The subcase $S = \lambda$ was treated before, so we will deal with the other three. All three proofs are similar, so we will prove only the subcase $S = ICG$).

$n[), S] = n[), I] + n[), H] + 1$ since $S = ICH$)

$> n[), I] + n[), H]$

$\geq n[(, I] + n[(, H]$ by IH on G and H

$= n[(, S]$ since $S = ICH$)

Cases 7,8: $F = QxG$ with $Q \in \{\forall, \exists\}$. The suffixes of F are the suffixes of G, xG, and QxG. By applying the induction hypothesis, we get that $n[), S] \geq n[(, S]$. The other two cases are of the form $S = TG$ where $T = x$ or $T = Qx$. Let us show that the lemma is also true for them.

$n[), S] = n[), G]$ since $S = TG$ and there are no parentheses in T

$\geq n[(, G]$ by IH on the suffix G of G

$= n[(, S]$ since $S = TG$ and there are no parentheses in T **Q.E.D.**

Corollary 2.2.13 *For all formulas F, $n[(, F] = n[), F]$.*

Proof: F is both a prefix and a suffix of F. So,

(1) $n[(, F] \geq n[), F]$

by Lemma 2.2.11 and

(2) $n[), F] \geq n[(, F]$

by Lemma 2.2.12. From (1) and (2) we get
(3)$n[(, F] = n[), F]$.
Q.E.D.
Now we can prove that no proper prefix of a formula is a formula.

Lemma 2.2.14 *If S is a prefix of F and S is a formula, then $S = F$.*

Proof: The proof is by induction on F.
Case 1: F is an atomic formula.

Subcase 1.1: $F = P$. In this case the only prefixes of F are $S = \lambda$ and $S = F$. Since λ is not a formula, we are left with $S = F$.

Subcase 1.2: $F = P(t_1, \ldots, t_m)$ and P has arity $m > 0$. The prefixes of F are $S = \lambda$, $S = P$, $S = P(t_1, \ldots, S_i$ with S_i a prefix of t_i, and $S = F$. Both λ and P are not formulas because all formulas have length greater than 0 and the only formulas of length 1 are the predicate symbols of arity greater than 0. For $S = P(t_1, \ldots, S_i$ we will show that the string has more left parentheses than right parentheses.

$n[(, S] = 1 + n[(, t_1] + \ldots + n[(, t_{i-1}] + n[(, S_i]$ since $S = P(t_1, \ldots, S_i$
$> n[(, t_1] + \ldots + n[(, t_{i-1}] + n[(, S_i]$
$\geq n[), t_1] + \ldots + n[), t_{i-1}] + n[), S_i]$ by Lemma 2.2.3
$= n[), S]$ since $S = P(t_1, \ldots, S_i$

So, we discard these prefixes and we are left with $S = F$.

Subcase 1.3: $S = E(t_1, t_2)$. The prefixes are $S = \lambda$, $S = E$, $S = E(S_1$ with S_i a prefix of t_1, $S = E(t_1, S_2$ with S_2 a prefix of t_2, and $S = F$. The equality predicate has arity 2, so we reject $S = E$. The empty string is rejected for the reasons stated in Subcase 1.2. For $S = E(S_1$ and $S = E(t_1, S_2$ we count the left and the right parentheses and find that there are more left parentheses than right parentheses. By Corollary 2.2.13, they are not formulas. Again, we are left with $S = F$.

Cases 2,3,4,5,6 have the same proof as in Lemma 1.2.11.

Cases 7,8: $F = QxG$. The prefixes of $S = \lambda$, $S = Q$, and $S = QxI$ with I a prefix of G. We discard Q because the only formulas of length 1 are the predicate constants. $S = \lambda$ is not a formula, so we are left with $S = QxI$. If S is a formula, so is I. But I is a prefix of G, so we apply the induction hypothesis and get that $I = G$. Then $S = QxI = QxG = F$. **Q.E.D.**

Finally, we can prove the unique readability of FOL formulas.

Theorem 2.2.15 (The Unique Readability Theorem) *Every formula belongs to one and only one of the 8 categories. Moreover, the decomposition of the atomic formulas into terms and of the other formulas into the main subformulas is unique.*

Proof: The 8 categories are atomic formulas, $F = \neg G$, $F = (G \vee H)$, $F = (G \wedge H)$, $F = (G \longrightarrow H)$, $F = (G \longleftrightarrow H)$, $F = \forall xG$, and $F = \exists xG$.

The atomic formulas, $F = \neg G$, $F = \forall xG$, and $F = \exists xG$ can be recognized by looking at the first symbol of the string. So, the formulas that start with P,E, $\neg$, $\forall$, and $\exists$ belong to only one category. Let us prove the unique readability.

Lemma 2.2.8 establishes the unique decomposition of atomic formulas. For the rest, the unique decomposition follows from the equality of the strings. For example, if $F = \forall x G = \forall y H$, $x = y$ because both x and y denote the symbol at address 1 of F. Both G and H represent the suffix of F that starts at position 2, so $G = H$.

Now we need to distinguish the categories $F = (G \vee H)$, $F = (G \wedge H)$, $F = (G \longrightarrow H)$, $F = (G \longleftrightarrow H)$. Let us assume that $F = (GCH)$ and $F = (IC_1J)$ where C, C_1 are binary connectives. Both the formula I and the formula G start at position 1 of F. So, either they have equal length, or one is shorter than the other. Let us assume that I is shorter than G. Then I is a prefix of G that is a formula, so $I = G$ by Lemma 2.2.14. So, I is not shorter than G. By a similar reasoning we show that G is not shorter than I. Then the only possibility is that I and G have the same length. Then I and J are equal since they start at the same position of F and have equal length. Then $C = C_1$ since they both denote the symbol of F at address $1+length(G)$. Finally, $H = J$ because both are the suffix of F that starts at position $length(G) + 2$. **Q.E.D.**

Exercises

Exercise 2.2.1 *Prove the fourth case of Lemma 2.2.3.*

Exercise 2.2.2 *Prove Lemma 2.2.4.*

Hint: For a variable, or a constant, the suffixes are λ and the term itself. If $\tau = f(\tau_1, \ldots, \tau_n)$, $S = \lambda$, $S = S_i, \tau_{i+1}, \ldots, \tau_n)$ with S_i a suffix of τ_i, $S = (\tau_1, \ldots, \tau_n)$, and $S = \tau$.

Exercise 2.2.3 *We define the arity of the string S, written $n[arity, S]$ to be the sum of the arities of all function symbols in S. For example, the arity of $S = f_1^3(f_2^2(x_1$ is $3 + 2 = 5$.*

Show, by structural induction on terms, that for every prefix P of t,

$n[arity, P] \geq n[(, P] + n[commas, P]$

where $n[commas, P]$ is the number of commas in P.

Exercise 2.2.4 *Prove, by structural induction on t, that for every suffix S of t,*

$n[arity, S] \leq n[), S] + n[commas, S]$.

Exercise 2.2.5 *Let t_1 and t_2 be two different terms and let p be the leftmost position where they differ. Prove, by structural induction on t_1, that both t_1 and t_2 have subterms that start at position p.*

Exercise 2.2.6 *Prove that every non-empty prefix of the term t that satisfies the equality $n[fun, S] = n[(, S] = n[), S]$ is equal to t.*

Hint: By structural induction on t. If t is a variable or a constant, then it has only one non-empty prefix and that is t itself. If $t = f(t_1, \ldots, t_n)$, then the non-empty prefixes fall into one of the following categories: $S = f$, $S = f(t_1, \ldots, t_{i-1}, \tau_i$ with τ_i a (possibly empty) prefix of t_i, and $S = t$. In the first case, we have more function symbols than left parentheses, and in the second, more left parentheses than right parentheses.

Exercise 2.2.7 *Use the preceding exercise to show that the subterms of $f(t_1, \ldots, t_n)$ are t and the subterms of $t_1, \ldots, t_n$.*

Hint: A subterm of $t = f(t_1, \ldots, t_n)$ cannot start on (, a comma, or on). So, it must start either with on f, or in one of the t_i's. Lemma 2.2.6 takes care of the first case. For the second case, we have 2 subcases:

2.1. $S = \alpha_i, t_{i+1}, \ldots, \beta_j$ with α_i a suffix of t_i and β_j a prefix of t_j, $j > i$

2.2. $S = \alpha_i, \ldots, t_n)$, with α_i a suffix of t_i.

Apply the preceding exercise to Subcase 2.1 and show that S has more right parentheses than left parentheses in the second subcase.

Exercise 2.2.8 *Prove Lemma 2.2.8.*

Exercise 2.2.9 *Prove Subcase 1.3 of Lemma 2.2.11.*

Exercise 2.2.10 *Prove Subcase 1.3 of Lemma 2.2.12.*

Exercise 2.2.11 *Let F be a formula and F^* be the string obtained from F by deleting all occurrences of $\neg$, $\forall x$, and $\exists x$, where x is any variable. Prove, by structural induction on F, that F^* is a formula.*

Exercise 2.2.12 *Prove, by structural induction on F, that whenever a prefix of F ends with a binary connective, the prefix has more left parentheses than right parentheses.*

Exercise 2.2.13 *Use the preceding exercise to prove that the subformulas of $F = (GCH)$ are F, the subformulas of G and the subformulas of H. Also show that the subformulas of $F = \neg G$, $F = \forall x G$, $F = \exists G$ are F and the subformulas of G.*

Exercise 2.2.14 *Prove that the algorithm from Figure 2.1 is correct. You must show that it terminates, and accepts a string iff the string is a term.*

Hint: Imitate the correctness proof from Section 1.2. The termination is easy and then show that all max-occurrences are terms. Then use the fact that the subterms of $t = f(t_1, \ldots, t_m)$ are t and the subterms of $t_1, \ldots, t_m$, to show that every term is accepted.

Exercise 2.2.15 *Combine the algorithms from Figures 2.1 and Figure 1.1 to obtain a formula recognition algorithm. Then prove its correctness.*

Hint: Use Algorithm 2.1 as a first step, but stop with failure if the input is fully underlined. For the second step, write the instructions that underline the atomic formulas. As the third step, use Algorithm 1.1, modified to include the quantified formulas. The preceding exercise provides the correctness of the first step, and the correctness of the first step is easy.

Exercise 2.2.16 *Show that the algorithm from Figure 2.6 computes a one-to-one function, i.e. it produces different trees for different formulas.*

Hint: We define a top-down recursive function that converts a formula tree into a string. Then we show that convert[tree [F]]=F for all formulas F and tree[convert[τ]]=τ for all formula trees τ.

2.3 Semantics of Formulas

The Unique Readability Theorem tells us that every term and every formula can be decomposed into simpler substructures in a unique way. This allows us to define the semantics in a simple and precise way, starting with the symbols and ending with complex formulas. The meaning of the terms is determined by the interpretation of the function symbols and of the variables that occur in them. The interpretation of the atomic formulas depends on the interpretation of the predicate symbols and of the terms, and the meaning of the complex formulas depends on the meaning of atomic formulas.

So, let us first define a *structure*. The structure gives meaning to all symbols in the language, not only to the symbols that occur in a particular formula.

Definition 2.3.1 (structure) *A structure is a 4-tuple* $\mathcal{A} =< |\mathcal{A}|, \mathcal{A}^f, \mathcal{A}^p, \mathcal{A}^v >$ *that satisfies the following conditions:*
1. $|\mathcal{A}|$ *is a non-empty set called the universe of the structure,*
2. $\mathcal{A}^f$ *assigns to each function symbol* g *a function* $\mathcal{A}^f[g]$*;*
2.1. if g *has arity 0,* $\mathcal{A}^f[g] \in |\mathcal{A}|$*,*
2.2. if g *has arity* $n > 0$ *, then* $\mathcal{A}^f[g] : |\mathcal{A}|^n \longrightarrow |\mathcal{A}|$*,*
3. $\mathcal{A}^p$ *assigns to each predicate symbol* P *a function* $\mathcal{A}^p[P]$ *with codomain* $\{0,1\}$*;*
3.1 if P *has arity 0,* $\mathcal{A}^p[P] \in \{0,1\}$*,*
3.2. if P_i^n *has arity* n *greater than* 0*, then* $\mathcal{A}^p[P] : |\mathcal{A}|^n \longrightarrow \{0,1\}$*, and*
4. $\mathcal{A}^v$ *assigns to each variable* x *an element in* $|\mathcal{A}|$*.*

Examples 2.3.2 Let us define three structures, $\mathcal{B}, \mathcal{C}$, and $\mathcal{E}$.

The components of the structure $\mathcal{B} =< |\mathcal{B}|, \mathcal{B}^f, \mathcal{B}^p, \mathcal{B}^v >$ are defined below.

1. $|\mathcal{B}| = \{3,4,5\}$.

2.1. We assign meaning to the individual constants.

$$\mathcal{B}^f[f_i^0] = \begin{cases} 3 & \text{if } i \text{ divided by 3 gives remainder 0} \\ 4 & \text{if } i \text{ divided by 3 gives remainder 1} \\ 5 & \text{if } i \text{ divided by 3 gives remainder 2} \end{cases}$$

2.2. The next formula defines $\mathcal{B}^f[f_i^n] : \{3,4,5\}^n \longrightarrow \{3,4,5\}$.

$$\mathcal{B}^f[f_i^n][s_1,\ldots,s_n] = \begin{cases} 3 & \text{if the remainder of } s_1+s_2+\ldots+s_n \text{ divided by 3 is 0} \\ 4 & \text{if the remainder of } s_1+s_2+\ldots+s_n \text{ divided by 3 is 1} \\ 5 & \text{if the remainder of } s_1+s_2+\ldots+s_n \text{ divided by 3 is 2} \end{cases}$$

3.1. Now we define the meaning of the predicate constants.

$$\mathcal{B}^p[P_i^0] = \begin{cases} 0 & \text{if } i \text{ is divisible by 3} \\ 1 & \text{if } i \text{ is not divisible by 3} \end{cases}$$

3.2. The next formula characterizes the functions $\mathcal{B}^p[P_i^n] : \{3,4,5\}^n \longrightarrow \{0,1\}$.

$$\mathcal{B}^p[P_i^n][s_1,\ldots,s_n] = \begin{cases} 0 & \text{if the remainder of } s_1+s_2+\ldots+s_n \text{ divided by 3 is 1} \\ 1 & \text{if the remainder of } s_1+s_2+\ldots+s_n \text{ divided by 3 is 0 or 2} \end{cases}$$

4. Finally we define the meaning of the variables.

$$\mathcal{B}^v[x_i] = \begin{cases} 3 & \text{if } i \text{ divided by 3 gives remainder 2} \\ 4 & \text{if } i \text{ divided by 3 gives remainder 1} \\ 5 & \text{if } i \text{ divided by 3 gives remainder 0} \end{cases}$$

Next, we define the structure is $\mathcal{C} =< |\mathcal{C}|, \mathcal{C}^f, \mathcal{C}^p, \mathcal{C}^v >$.

1. The universe is $|\mathcal{C}| = \{3, 4\}$.

2.1. Now, we define the meaning of the function constants.

$$\mathcal{C}^f[f_i^0] = \begin{cases} 3 & \text{if } i \text{ is odd} \\ 4 & \text{if } i \text{ is even} \end{cases}$$

2.2. The next formula gives meaning to the function symbols of arity $n > 0$.
$\mathcal{C}^f[f_i^n][s_1, \ldots, s_n] = s_1$

3.1. Further on, we define the define $\mathcal{C}^p$ for the predicate constants.

$$\mathcal{C}^p[P_i^0] = \begin{cases} 0 & \text{if } i \text{ is odd} \\ 1 & \text{if } i \text{ is even} \end{cases}$$

3.2. Now, we give meaning to the predicate symbols of arity greater than 0.

$$\mathcal{C}^p[P_i^n][s_1, \ldots, s_n] = \begin{cases} 0 & \text{if } s_n \text{ is even} \\ 1 & \text{if } s_n \text{ is odd} \end{cases}$$

4. Finaly, we define $\mathcal{C}^v$.

$$\mathcal{C}^v[x_i] = \begin{cases} 3 & \text{if } i \text{ is prime (0 and 1 are not prime)} \\ 4 & \text{if } i \text{ is not prime} \end{cases}$$

The last structure, $\mathcal{E} =< |\mathcal{E}|, \mathcal{E}^f, \mathcal{E}^p, \mathcal{E}^v >$ has an infinite universe.

1. $|\mathcal{E}| = N$, where $N = \{0, 1, 2, \ldots, n, \ldots\}$ is the set of natural numbers.

2.1. $\mathcal{E}^f[f_i^0] = i$

2.2. For $n > 0$, $\mathcal{E}^f[f_i^n][s_1, \ldots, s_n] = max(s_1, \ldots, s_n)$, where $max(s_1, \ldots, s_n)$ is the largest of $s_1, \ldots, s_n$.

3.1. Next, we give meaning to the predicate constants.

$$\mathcal{E}^p[P_i^0] = \begin{cases} 0 & \text{if } i \text{ is even} \\ 1 & \text{if } i \text{ is odd} \end{cases}$$

3.2. Then, we define $\mathcal{E}^p$ for the predicate symbols of arity greater than 0.

$$\mathcal{E}^p[P_i^n][s_1, \ldots, s_n] = \begin{cases} 1 & \text{if } s_1 \geq s_2 \geq s_3 \ldots \geq s_n \\ 0 & \text{if the sequence } s_1, \ldots, s_n \text{ is not in descending order} \end{cases}$$

4. $\mathcal{E}^v[x_i] = i$.

Now we are ready to interpret the terms.

Definition 2.3.3 (Interpretation of terms) *Let $\mathcal{A} =< |\mathcal{A}|, \mathcal{A}^f, \mathcal{A}^p, \mathcal{A}^v >$ be a structure and t be a term. Then, the $\mathcal{A}$ interpretation of t, written $\mathcal{A}[t]$, is defined by the rules below.*

1. *If t is an individual variable then $\mathcal{A}[t] = \mathcal{A}^v[t]$.*
2. *If t is an individual constant then $\mathcal{A}[t] = \mathcal{A}^f[t]$.*
3. *If $t = f(t_1, \ldots, t_n)$ then $\mathcal{A}[t] = \mathcal{A}^f[f_i^n][\mathcal{A}[t_1], \ldots, \mathcal{A}[t_n]]$.*

Example 2.3.4 Let us use Definition 2.3.3 to compute $\mathcal{B}[x_3]$, $\mathcal{C}[f_4^3(x_4, f_2^0, f_3^0)]$, and $\mathcal{E}[f_2^4(f_2^1(x_1), f_3^0, x_5, f_5^0)]$ where $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{E}$ are the structures defined in Examples 2.3.2.

$\mathcal{B}[x_3] = \mathcal{B}^v[x_3]$ by rule 1 of Definition 2.3.3
$= 5$ by the definition of $\mathcal{B}^v$
$\mathcal{C}[f_4^3(x_4, f_2^0, f_3^0)] = \mathcal{C}^f[f_4^3][\mathcal{C}[x_4], \mathcal{C}[f_2^0], \mathcal{C}[f_3^0]]$ by rule 3
$= \mathcal{C}^f[f_4^3][\mathcal{C}^v[x_4], \mathcal{C}[f_2^0], \mathcal{C}[f_3^0]]$ by rule 1
$= \mathcal{C}^f[f_4^3][4, \mathcal{C}[f_2^0], \mathcal{C}[f_3^0]]$ from the definition of $\mathcal{C}^v$
$= 4$ from the definition of $\mathcal{C}^f$

$\mathcal{E}[f_2^4(f_2^1(x_1), f_3^0, x_5, f_5^0)] = \mathcal{E}^f[f_2^4][\mathcal{E}[f_2^1(x_1)], \mathcal{E}[f_3^0], \mathcal{E}[x_5], \mathcal{E}[f_5^0]]$ from rule 3
$= \mathcal{E}^f[f_2^4][\mathcal{E}[f_2^1][\mathcal{E}[x_1]], \mathcal{E}[f_3^0], \mathcal{E}[x_5], \mathcal{E}[f_5^0]]$ from rule 3
$= \mathcal{E}^f[f_2^4][\mathcal{E}[f_2^1][\mathcal{E}^v[x_1]], \mathcal{E}^f[f_3^0], \mathcal{E}^v[x_5], \mathcal{E}^f[f_5^0]]$ from rules 1 and 2
$= \mathcal{E}^f[f_2^4][\mathcal{E}[f_2^1][1], 3, 5, 5]$ from the definitions of $\mathcal{E}^f$, $\mathcal{E}^v$
$= \mathcal{E}^f[f_2^4][1, 3, 5, 5]$
$= 5$

Next, we give the interpretation of the atomic formulas.

Definition 2.3.5 (the interpretation of atomic formulas) *Let $\mathcal{A} =< |\mathcal{A}|, \mathcal{A}^f, \mathcal{A}^p, \mathcal{A}^v >$ be a structure. Then*

1. $\mathcal{A}[P_i^0] = \mathcal{A}^p[P_i^0]$
2. $\mathcal{A}[P_i^n(t_1, \ldots, t_n)] = \mathcal{A}^p[P_i^n][\mathcal{A}[t_1], \ldots, \mathcal{A}[t_n]]$
3. $$\mathcal{A}[E(t_1, t_2)] = \begin{cases} 1 & \text{if } \mathcal{A}[t_1] = \mathcal{A}[t_2] \\ 0 & \text{if } \mathcal{A}[t_1] \neq \mathcal{A}[t_2] \end{cases}$$

So, part 1 of Definition 2.3.5 gives meaning to the individual predicate symbols, part 2 interprets the atomic formulas with predicates of arity greater than 0, and part 3 defines the equality formulas.

Now let us compute the interpretation of several atomic formulas using the structures Examples 2.3.2. Let us find $\mathcal{B}[P_3^0]$, $\mathcal{C}[P_4^3(f_2^0, f_1^1(x_2), f_4^0)]$ and $\mathcal{E}[E(f_4^2(x_2, f_6^0), x_6)]$.

1. $\mathcal{B}[P_3^0] = \mathcal{B}^p[P_3^0]$ by rule 1 of Definition 2.3.5
$= 0$ by the definition of $\mathcal{B}^p[P_3^2]$

2. $\mathcal{C}[P_4^3(f_2^0, f_1^1(x_2), f_4^0)] = \mathcal{C}^p[P_4^3][\mathcal{C}[f_2^0], \mathcal{C}[f_1^1(x_2)], \mathcal{C}[f_4^0]]$ by rule 2 of Definition 2.3.5
$= \mathcal{C}^p[P_4^3][\mathcal{C}[f_2^0], \mathcal{C}[f_1^1(x_2)], \mathcal{C}^f[f_4^0]]$ by rule 3 of Definition 2.3.3
$= \mathcal{C}^p[P_4^3][\mathcal{C}[f_2^0], \mathcal{C}[f_1^1(x_2)], 4]$ by the definition of $\mathcal{C}^f$
$= 0$ by the definition of $\mathcal{C}[P_4^3]$

3. $\mathcal{E}[E(f_4^2(x_2, f_6^0), x_6)] = \mathcal{E}[E][\mathcal{E}[f_4^2(x_2, f_6^0)], \mathcal{E}[x_6]]$
$= \mathcal{E}[E][\mathcal{E}^f[f_4^2][\mathcal{E}[x_2], \mathcal{E}[f_6^0]], \mathcal{E}^v[x_6]]$ from the rules 3, 1 of Definition 2.3.3
$= \mathcal{E}[E][\mathcal{E}^f[f_4^2][\mathcal{E}^v[x_2], \mathcal{E}^f[f_6^0]], 6]$ from Definition 2.3.3 and $\mathcal{E}^v[x_6] = 6$
$= \mathcal{E}[E][\mathcal{E}^f[f_4^2][2, 6], 6]$ compute $\mathcal{E}^v[x_2]$ and $\mathcal{E}^f[f_6^0]$
$= \mathcal{E}[E][6, 6]$ compute $\mathcal{E}^f[f_4^2][2, 6]$
$= 1$ the two arguments of $[E]$ are equal.

Before we go on to discuss the semantics of formulas we need to define the structures $\mathcal{A}_{[x \leftarrow d]}$, that are identical to $\mathcal{A}$ except for the interpretation of x.

Definition 2.3.6 ($\mathcal{A}$ with x interpreted as d) *Let $\mathcal{A} =< |\mathcal{A}|, \mathcal{A}^f, \mathcal{A}^p, \mathcal{A}^v >$ be a structure, x be a variable, and d be an element in $|\mathcal{A}|$. $\mathcal{A}_{[x \leftarrow d]}$ is the structure $< |\mathcal{A}|, \mathcal{A}^f, \mathcal{A}^p, \mathcal{A}^v_{[x \leftarrow d]} >$ with*

$$\mathcal{A}^v_{[x \leftarrow d]}[y] = \begin{cases} \mathcal{A}^v[y] & \text{if } y \neq x \\ d & \text{if } y = x \end{cases}$$

We call $\mathcal{A}_{[x \leftarrow d]}$, $\mathcal{A}$ with x interpreted as d.

We define $\mathcal{A}_{[x \leftarrow d_1][y \leftarrow d_2]}$ as the structure $[\mathcal{A}_{[x \leftarrow d_1]}]_{[y \leftarrow d_2]}$ obtained from $\mathcal{A}$ by first assigning d_1 to x and then assigning d_2 to y.

Proposition 2.3.7 *If $x \neq y$, $\mathcal{A}_{[x \leftarrow d_1][y \leftarrow d_2]} = \mathcal{A}_{[y \leftarrow d_2][x \leftarrow d_1]}$.*

Proof: The two structures have the same domain, $|\mathcal{A}|$, the same interpretations for functions and predicates. Let us show that ${\mathcal{A}_{[x \leftarrow d_1][y \leftarrow d_2]}}^v = {\mathcal{A}_{[y \leftarrow d_2][x \leftarrow d_1]}}^v$. Let z be a variable. We have 3 cases.
Case 1: $z \neq x$ and $z \neq y$. Then
${\mathcal{A}_{[x \leftarrow d_1][y \leftarrow d_2]}}^v[z] = {\mathcal{A}_{[x \leftarrow d_1]}}^v[z]$ because $z \neq y$
$= \mathcal{A}^v[z]$ since $z \neq x$
${\mathcal{A}_{[y \leftarrow d_2][x \leftarrow d_1]}}^v[z] = {\mathcal{A}_{[y \leftarrow d_2]}}^v[z]$ since $z \neq x$
$= \mathcal{A}^v[z]$ because $z \neq x$
Case 2: $z = x$.
${\mathcal{A}_{[x \leftarrow d_1][y \leftarrow d_2]}}^v[x] = {\mathcal{A}_{[x \leftarrow d_1]}}^v[x]$ since $x \neq y$
$= d_1$
${\mathcal{A}_{[y \leftarrow d_2][x \leftarrow d_1]}}^v[y] = d_1$
Case 3: $z = y$.
${\mathcal{A}_{[x \leftarrow d_1][y \leftarrow d_2]}}^v[y] = d_2$
${\mathcal{A}_{[y \leftarrow d_2][x \leftarrow d_1]}}^v[y] = {\mathcal{A}_{[y \leftarrow d_2]}}^v[y]$ because $x \neq y$
$= d_2$
Q.E.D.

Proposition 2.3.7 tells us that when the variables are different, the order of the assignments is irrevelant. For this reason we write $\mathcal{A}_{[x_1 \leftarrow d_1, x_2 \leftarrow d_2, \ldots, x_n \leftarrow d_n]}$ to denote the structure obtained from $\mathcal{A}$ by assigning d_1 to x_1, d_2 to x_2,..., and d_n to x_n.

Now we are ready to define the interpretation of the formulas that contain the operators $\neg$, $\wedge$, $\vee$, $\longrightarrow$, $\longleftrightarrow$, $\forall x$ and $\exists x$.

Definition 2.3.8 (interpretation of formulas) *Let $\mathcal{A}$ be a structure with universe D, F and G be formulas and x be a variable. Then,*

1. $\mathcal{A}[\neg F] = \boxed{\neg}\mathcal{A}[F]$
2. $\mathcal{A}[F \wedge G] = \mathcal{A}[F]\boxed{\wedge}\mathcal{A}[G]$
3. $\mathcal{A}[F \vee G] = \mathcal{A}[F]\boxed{\vee}\mathcal{A}[G]$
4. $\mathcal{A}[F \longrightarrow G] = \mathcal{A}[F]\boxed{\longrightarrow}\mathcal{A}[G]$
5. $\mathcal{A}[F \longleftrightarrow G] = \mathcal{A}[F]\boxed{\longleftrightarrow}\mathcal{A}[G]$
6. $\mathcal{A}[\forall xF] = 1$ *iff for all* $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[F] = 1$
7. $\mathcal{A}[\exists xF] = 1$ *for some* $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[F] = 1$

Here, $\boxed{\neg}$, $\boxed{\wedge}$, $\boxed{\vee}$, $\boxed{\longrightarrow}$, $\boxed{\longleftrightarrow}$ *are the operations defined in Section 1.3.*

Example 2.3.9 Let us compute $\mathcal{B}[\forall x_3 P_1^3(x_3, x_3, x_3)]$. According to Definition 2.3.8,

$\mathcal{B}[\forall x_3 P_1^3(x_3, x_3, x_3)] = 1$ if and only if for all d in the universe of $\mathcal{B}$, $\mathcal{B}_{[x_3 \leftarrow d]}[P_1^3(x_3, x_3, x_3)] = 1$.

Since $|\mathcal{B}| = \{3, 4, 5\}$ we compute $\mathcal{B}_{[x_3 \leftarrow 3]}[P_1^3(x_3, x_3, x_3)]$, $\mathcal{B}_{[x_3 \leftarrow 4]}[P_1^3(x_3, x_3, x_3)]$, and $\mathcal{B}_{[x_3 \leftarrow 5]}[P_1^3(x_3, x_3, x_3)]$.

$\mathcal{B}_{[x_3 \leftarrow 3]}[P_1^3(x_3, x_3, x_3)]$
$= \mathcal{B}_{[x_3 \leftarrow 3]}[P_1^3][\mathcal{B}_{[x_3 \leftarrow 3]}[x_3], \mathcal{B}_{[x_3 \leftarrow 3]}[x_3], \mathcal{B}_{[x_3 \leftarrow 3]}[x_3]]$
$= \mathcal{B}[P_1^3][\mathcal{B}_{[x_3 \leftarrow 3]}[x_3], \mathcal{B}_{[x_3 \leftarrow 3]}[x_3], \mathcal{B}_{[x_3 \leftarrow 3]}[x_3]]$ because ${\mathcal{B}_{[x_3 \leftarrow 3]}}^p = \mathcal{B}^p$
$= \mathcal{B}[P_1^3][3, 3, 3]$ because ${\mathcal{B}_{[x_3 \leftarrow 3]}}^v[x_3] = 3$
$= 1$ because the sum 3+3+3 is divisible by 3.

Now, we compute $\mathcal{B}_{[x_3 \leftarrow 4]}[P_1^3(x_3, x_3, x_3)]$.

$\mathcal{B}_{[x_3 \leftarrow 4]}[P_1^3(x_3, x_3, x_3)]$
$= \mathcal{B}_{[x_3 \leftarrow 4]}[P_1^3][\mathcal{B}_{[x_3 \leftarrow 4]}[x_3], \mathcal{B}_{[x_3 \leftarrow 4]}[x_3], \mathcal{B}_{[x_3 \leftarrow 4]}[x_3]]$
$= \mathcal{B}[P_1^3][\mathcal{B}_{[x_3 \leftarrow 4]}[x_3], \mathcal{B}_{[x_3 \leftarrow 4][x_3]}, \mathcal{B}_{[x_3 \leftarrow 4]}[x_3]]$ because ${\mathcal{B}_{[x_3 \leftarrow 4]}}^p = \mathcal{B}^p$
$= \mathcal{B}[P_1^3][4, 4, 4]$ because ${\mathcal{B}_{[x_3 \leftarrow 4]}}^v[x_3] = 4$
$= 1$ because the sum 4+4+4 is divisible by 3.

Finally we compute $\mathcal{B}_{[x_3 \leftarrow 5]}[P_1^3(x_3, x_3, x_3)]$.

$\mathcal{B}_{[x_3 \leftarrow 5]}[P_1^3(x_3, x_3, x_3)]$
$= \mathcal{B}_{[x_3 \leftarrow 5]}[P_1^3][\mathcal{B}_{[x_3 \leftarrow 5]}[x_3], \mathcal{B}_{[x_3 \leftarrow 5]}[x_3], \mathcal{B}_{[x_3 \leftarrow 5]}[x_3]]$
$= \mathcal{B}[P_1^3][\mathcal{B}_{[x_3 \leftarrow 5]}[x_3], \mathcal{B}_{[x_3 \leftarrow 5]}[x_3], \mathcal{B}_{[x_3 \leftarrow 5]}[x_3]]$ because ${\mathcal{B}_{[x_3 \leftarrow 5]}}^p = \mathcal{B}^p$
$= \mathcal{B}[P_1^3][5, 5, 5]$ because ${\mathcal{B}_{[x_3 \leftarrow 5]}}^v[x_3] = 5$
$= 1$ because the sum 5+5+5 is divisible by 3.

Since $\mathcal{B}_{[x_3 \leftarrow 3]}[P_1^3(x_3, x_3, x_3)]$, $\mathcal{B}_{[x_3 \leftarrow 4]}[P_1^3(x_3, x_3, x_3)]$, and $\mathcal{B}_{[x_3 \leftarrow 5]}[P_1^3(x_3, x_3, x_3)]$ are all 1, $\mathcal{B}[\forall x_3 P_1^3(x_3, x_3, x_3)] = 1$.

Example 2.3.10 Let us evaluate $\mathcal{E}[\exists x_1 P_3^2(x_1, x_{20})]$.

Definition 2.3.8 tells us that $\mathcal{E}[\exists x_1 P_3^2(x_1, x_{20})] = 1$ if and only if for some d in the universe of $\mathcal{E}$, $\mathcal{E}_{[x_1 \leftarrow d]}[P_3^2(x_1, x_{20})] = 1$. Let us compute $\mathcal{E}_{[x_1 \leftarrow d]}[P_3^2(x_1, x_{20})]$.

$\mathcal{E}_{[x_1 \leftarrow d]}[P_3^2(x_1, x_{20})]$
$= \mathcal{E}_{[x_1 \leftarrow d]}[P_3^2][\mathcal{E}_{[x_1 \leftarrow d]}[x_1], \mathcal{E}_{[x_1 \leftarrow d]}[x_{20}]]$
$= \mathcal{E}[P_3^2][\mathcal{E}_{[x_1 \leftarrow d]}[x_1], \mathcal{E}_{[x_1 \leftarrow d]}[x_{20}]]$ since ${\mathcal{E}}^p = {\mathcal{E}_{[x_1 \leftarrow d]}}^p$
$= \mathcal{E}[P_3^2][d, \mathcal{E}_{[x_1 \leftarrow d]}[x_{20}]]$ because ${\mathcal{E}_{[x_1 \leftarrow d]}}^v[x_1] = d$
$= \mathcal{E}[P_3^2][d, \mathcal{E}[x_{20}]]$ since $\mathcal{E}_{[x_1 \leftarrow d]}$ changes only x_1
$= \mathcal{E}[P_3^2][d, 20]$

From Example 2.3.2 we know that $\mathcal{E}[P_3^2][d, 20] = 1$ only when $d \geq 20$. Since the universe of $\mathcal{E}$ is N, we pick $d = 20$ and get that $\mathcal{E}[P_3^2][20, 20] = 1$. So, there is some $d \in N$, such that $\mathcal{E}_{[x_1 \leftarrow d]}[P_3^2(x_1, x_{20})] = 1$. By Definition 2.3.8 $\mathcal{E}[\exists x_1 P_3^2(x_1, x_{20})] = 1$.

Example 2.3.11 Let us compute $\mathcal{C}[\forall x_1 \exists x_2 P_1^2(x_2, x_1)]$.

Here we have two quantifiers, $\forall x_1$ and $\exists x_2$.

The universe of $\mathcal{C}$ is $|\mathcal{C}| = \{3, 4\}$, so $\mathcal{C}[\forall x_1 \exists x_2 P_1^2(x_2, x_1)] = 1$ if and only if both $\mathcal{C}_{[x_1 \leftarrow 3]}[\exists x_2 P_1^2(x_2, x_1)]$ and $\mathcal{C}_{[x_1 \leftarrow 4]}[\exists x_2 P_1^2(x_2, x_1)]$ are 1.

We compute $\mathcal{C}_{[x_1 \leftarrow 3]}[\exists x_2 P_1^2(x_2, x_1)]$. This value is 1 iff at least one of the expressions $\mathcal{C}_{[x_1 \leftarrow 3][x_2 \leftarrow 3]}[P_1^2(x_2, x_1)]$ and $\mathcal{C}_{[x_1 \leftarrow 3][x_2 \leftarrow 4]}[P_1^2(x_2, x_1)]$ is 1.

$\mathcal{C}_{[x_1 \leftarrow 3][x_2 \leftarrow 3]}[P_1^2(x_2, x_1)]$
$= \mathcal{C}_{[x_1 \leftarrow 3][x_2 \leftarrow 3]}[P_1^2][\mathcal{C}_{[x_1 \leftarrow 3][x_2 \leftarrow 3]}[x_2], \mathcal{C}_{[x_1 \leftarrow 3][x_2 \leftarrow 3]}[x_1]]$
$= \mathcal{C}[P_1^2][\mathcal{C}_{[x_1 \leftarrow 3][x_2 \leftarrow 3]}[x_2], \mathcal{C}_{[x_1 \leftarrow 3][x_2 \leftarrow 3]}[x_1]]$ since $\mathcal{C}_{[x_1 \leftarrow 3][x_2 \leftarrow 3]}{}^p = \mathcal{C}^p$
$= \mathcal{C}[P_1^2][3, 3]$ since $\mathcal{C}_{[x_1 \leftarrow 3][x_2 \leftarrow 3]}$ assigns 3 to both x_1 and x_2
$= 1$ because the last argument of $\mathcal{C}[P_1^2]$, i.e. 3, is odd.

So, $\mathcal{C}_{[x_1 \leftarrow 3]}[\exists x_2 P_1^2(x_2, x_1)] = 1$. Now we compute $\mathcal{C}_{[x_1 \leftarrow 4]}[\exists x_2 P_1^2(x_2, x_1)]$.

The universe of $\mathcal{C}$ is $\{3, 4\}$, so, $\mathcal{C}_{[x_1 \leftarrow 4]}[\exists x_2 P_1^2(x_2, x_1)] = 1$ if and only if at least one of $\mathcal{C}_{[x_1 \leftarrow 4][x_2 \leftarrow 3]}[P_1^2(x_2, x_1)]$ and $\mathcal{C}_{[x_1 \leftarrow 4][x_2 \leftarrow 4]}[P_1^2(x_2, x_1)]$ is 1.

We evaluate these interpretations.

$\mathcal{C}_{[x_1 \leftarrow 4][x_2 \leftarrow 3]}[P_1^2(x_2, x_1)]$
$= \mathcal{C}[P_1^2][\mathcal{C}_{[x_1 \leftarrow 4][x_2 \leftarrow 3]}[x_2], \mathcal{C}_{[x_1 \leftarrow 4][x_2 \leftarrow 3]}[x_1]]$
$= \mathcal{C}[P_1^2][3, 4]$
$= 0$ because the last argument of $\mathcal{C}[P_1^2]$, 4, is even.

In the same way, $\mathcal{C}_{[x_1 \leftarrow 4][x_2 \leftarrow 4]}[P_1^2(x_2, x_1)] = 0$.

Since both $\mathcal{C}_{[x_1 \leftarrow 4][x_2 \leftarrow 3]}[P_1^2(x_2, x_1)]$ and $\mathcal{C}_{[x_1 \leftarrow 4][x_2 \leftarrow 4]}[P_1^2(x_2, x_1)]$ are 0, $\mathcal{C}_{[x_1 \leftarrow 4]}\exists x_2 P_1^2(x_2, x_1)] = 0$.

Then, $\mathcal{C}_{[x_1 \leftarrow 4]}[\exists x_2 P_1^2(x_2, x_1)] = 0$, makes $\mathcal{C}[\forall x_1 \exists x_2 P_1^2(x_2, x_1)]$ equal to 0.

Observation 2.3.12 1. The universal quantifier $\forall$ works like $\wedge$. If $\mathcal{A}$ is a structure with universe $D = \{d_1, \ldots, d_n\}$ and F is a formula then

$\mathcal{A}[\forall x F] = \mathcal{A}_{[x \leftarrow d_1]}[F] \boxed{\wedge} \ldots \boxed{\wedge} \mathcal{A}_{[x \leftarrow d_n]}[F]$.

We can do a lazy evaluation of $\mathcal{A}_{[x \leftarrow d_1]}[F]$,..., $\mathcal{A}_{[x \leftarrow d_1]}[F]$ and stop at the first assignment of x that evaluates F to 0.

2. The universal quantifier $\exists$ works like a $\vee$. So, for the structure $\mathcal{A}$ with universe $D = \{d_1, \ldots, d_n\}$ and the formula F,

$\mathcal{A}[\exists x F] = \mathcal{A}_{[x \leftarrow d_1]}[F] \boxed{\vee} \ldots \boxed{\vee} \mathcal{A}_{[x \leftarrow d_n]}[F]$

We can do a lazy evaluation of $\mathcal{A}_{[x \leftarrow d_1]}[F]$, ..., $\mathcal{A}_{[x \leftarrow d_1]}[F]$ by stopping at the first assignment of x that makes F equal to 1.

Example 2.3.13 Let us compute $\mathcal{B}[\exists x \forall y P_1^2(x, y)]$. Since the universe of $\mathcal{B}$ is the finite set $\{3, 4, 5\}$ we apply Observation 2.3.12. We do a lazy evaluation, meaning that we stop as soon as we have the answer for the formula, even if

some subformulas are not evaluated. We recall that $\mathcal{B}[P_1^2][m,n] = 0$ if and only if the sum $m+n$ divided by 3 gives remainder 1.

$\mathcal{B}[\exists x \forall y P_1^2(x,y)]$
$= \mathcal{B}_{[x\leftarrow 3]}[\forall y P_1^2(x,y)] \boxed{\vee} \mathcal{B}_{[x\leftarrow 4]}[\forall y P_1^2(x,y)] \boxed{\vee} \mathcal{B}_{[x\leftarrow 5]}[\forall y P_1^2(x,y)]$ by Observation 2.3.12
$= [\mathcal{B}_{[x\leftarrow 3][y\leftarrow 3]}[P_1^2(x,y)] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 3][y\leftarrow 4]}[P_1^2(x,y)] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 3][y\leftarrow 5]}[P_1^2(x,y)]] \boxed{\vee}$
$\mathcal{B}_{[x\leftarrow 4]}[\forall y P_1^2(x,y)] \boxed{\vee} \mathcal{B}_{[x\leftarrow 5]}[\forall y P_1^2(x,y)]$ by Observation 2.3.12
$= [\mathcal{B}[P_1^2][\mathcal{B}_{[x\leftarrow 3,y\leftarrow 3]}[x], \mathcal{B}_{[x\leftarrow 3,y\leftarrow 3]}[y]] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 3][y\leftarrow 4]}[P_1^2(x,y)] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 3][y\leftarrow 5]}[P_1^2(x,y)]] \boxed{\vee}$
$\mathcal{B}_{[x\leftarrow 4]}[\forall y P_1^2(x,y)] \boxed{\vee} \mathcal{B}_{[x\leftarrow 5]}[\forall y P_1^2(x,y)]$ we interpret the first disjunct
$= [\mathcal{B}[P_1^2][3,3] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 3,y\leftarrow 3]}[y]] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 3][y\leftarrow 4]}[P_1^2(x,y)] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 3][y\leftarrow 5]}[P_1^2(x,y)]] \boxed{\vee}$
$\mathcal{B}_{[x\leftarrow 4]}[\forall y P_1^2(x,y)] \boxed{\vee} \mathcal{B}_{[x\leftarrow 5]}[\forall y P_1^2(x,y)]$
$= [1 \boxed{\wedge} [\mathcal{B}[P_1^2][\mathcal{B}_{[x\leftarrow 3,y\leftarrow 4]}[x], \mathcal{B}_{[x\leftarrow 3,y\leftarrow 4]}[y]] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 3][y\leftarrow 5]}[P_1^2(x,y)]] \boxed{\vee} \mathcal{B}_{[x\leftarrow 4]}[\forall y P_1^2(x,y)] \boxed{\vee}$
$\mathcal{B}_{[x\leftarrow 5]}[\forall y P_1^2(x,y)]$ we interpret the second disjunct
$= [\mathcal{B}[P_1^2][3,4] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 3][y\leftarrow 5]}[P_1^2(x,y)]] \boxed{\vee} \mathcal{B}_{[x\leftarrow 4]}[\forall y P_1^2(x,y)] \boxed{\vee} \mathcal{B}_{[x\leftarrow 5]}[\forall y P_1^2(x,y)]$
$= 0 \boxed{\wedge} \mathcal{B}_{[x\leftarrow 3][y\leftarrow 5]}[P_1^2(x,y)]] \boxed{\vee} \mathcal{B}_{[x\leftarrow 4]}[\forall y P_1^2(x,y)] \boxed{\vee} \mathcal{B}_{[x\leftarrow 5]}[\forall y P_1^2(x,y)]$ because the sum 3+4 divided by 3 gives remainder 1
$= 0 \boxed{\vee} \mathcal{B}_{[x\leftarrow 4]}[\forall y P_1^2(x,y)] \boxed{\vee} \mathcal{B}_{[x\leftarrow 5]}[\forall y P_1^2(x,y)]$ by the lazy evaluation
$= [\mathcal{B}_{[x\leftarrow 4][y\leftarrow 3]}[P_1^2(x,y)] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 4][y\leftarrow 4]}[P_1^2(x,y)] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 4][y\leftarrow 5]}[P_1^2(x,y)]] \boxed{\vee}$
$\mathcal{B}_{[x\leftarrow 5]}[\forall y P_1^2(x,y)]$ evaluate the second conjunct
$= [\mathcal{B}[P_1^2][\mathcal{B}_{[x\leftarrow 4,y\leftarrow 3]}[x], \mathcal{B}_{[x\leftarrow 4,y\leftarrow 3]}[y]] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 4][y\leftarrow 4]}[P_1^2(x,y)] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 4][y\leftarrow 5]}[P_1^2(x,y)]] \boxed{\vee}$
$\mathcal{B}_{[x\leftarrow 5]}[\forall y P_1^2(x,y)]$
$= [\mathcal{B}[P_1^2[4,3]] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 4][y\leftarrow 4]}[P_1^2(x,y)] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 4][y\leftarrow 5]}[P_1^2(x,y)]] \boxed{\vee} \mathcal{B}_{[x\leftarrow 5]}[\forall y P_1^2(x,y)]$
$= [0 \boxed{\wedge} \mathcal{B}_{[x\leftarrow 4][y\leftarrow 4]}[P_1^2(x,y)] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 4][y\leftarrow 5]}[P_1^2(x,y)]] \boxed{\vee} \mathcal{B}_{[x\leftarrow 5]}[\forall y P_1^2(x,y)]$
because the sum 4 + 3 divided by 3 gives remainder 1
$= \mathcal{B}_{[x\leftarrow 5]}[\forall y P_1^2(x,y)]$ by the lazy evaluation
$= \mathcal{B}_{[x\leftarrow 5][y\leftarrow 3]}[P_1^2(x,y)] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 5][y\leftarrow 4]}[P_1^2(x,y)] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 5][y\leftarrow 5]}[P_1^2(x,y)]$
$= \mathcal{B}[P_1^2][5,3] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 5][y\leftarrow 4]}[P_1^2(x,y)] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 5][y\leftarrow 5]}[P_1^2(x,y)]$ compute the first conjunct
$= 1 \wedge \mathcal{B}_{[x\leftarrow 5][y\leftarrow 4]}[P_1^2(x,y)] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 5][y\leftarrow 5]}[P_1^2(x,y)]$ because the remainder of 5+3 divided by 3 is 2
$= \mathcal{B}[P_1^2][5,4] \boxed{\wedge} \mathcal{B}_{[x\leftarrow 5][y\leftarrow 5]}[P_1^2(x,y)]$ compute the second conjunct
$= 1 \boxed{\wedge} \mathcal{B}_{[x\leftarrow 5][y\leftarrow 5]}[P_1^2(x,y)]$ because the remainder of 5+4 divided by 3 is 0
$= \mathcal{B}_{[x\leftarrow 5][y\leftarrow 5]}[P_1^2(x,y)]$
$= \mathcal{B}[P_1^2][5,5]$ compute the third conjunct
$= 0$ because the remainder of 5+5 divided by 3 is 1

The preceding example shows that the interpretation of a formula in a finite structure reduces to a chain of equalities.

Convention 2.3.14 We will simplify the notation for structures, just like we did for formulas.

1. We will not specify the components $\mathcal{A}^f$, $\mathcal{A}^p$, $\mathcal{A}^v$ of the structure unless we really need to. So we will write $\mathcal{A}[P]$ instead of $\mathcal{A}^p[P]$, $\mathcal{A}[g]$ for $\mathcal{A}^f[g]$, and $\mathcal{A}[x]$ for $\mathcal{A}^v[x]$.

2. We will denote the interpretation of a symbol in a structure by puting the name of the structure as the superscript of the symbol. So, we will write $f^{\mathcal{A}}$ instead of $\mathcal{A}[f]$, $P^{\mathcal{A}}$ instead of $\mathcal{A}[P]$, and $x^{\mathcal{A}}$ instead of $\mathcal{A}[x]$. This way we avoid loading the formulas with parentheses.

The structures give meaning to all formulas of the language. However, the meaning of a term depends only on the symbols that occur in the term. The same thing goes for a formula; its meaning is determined by the functions, predicates, and the free variables that are part of the formula. In the next paragraphs we will prove these assertions.

Definition 2.3.15 (agreement on a term) *Let $\mathcal{A}$ and $\mathcal{B}$ be two structures with the same universe D. We say that $\mathcal{A}$ and $\mathcal{B}$ agree on a term t if the two structures give the same meaning to all individual constants, function symbols, and variables that occur in t, i.e.*

for every function symbol f of arity $n \geq 0$ that occurs in t, $f^{\mathcal{A}} = f^{\mathcal{B}}$, and for every variable x that occurs in t, $x^{\mathcal{A}} = x^{\mathcal{B}}$.

Example 2.3.16 Let $\mathcal{A}$ and $\mathcal{B}$ be two structures having the same universe $\{1, 2, 3\}$. Let $x^{\mathcal{A}} = x^{\mathcal{B}} = 2$, $a^{\mathcal{A}} = a^{\mathcal{B}} = 3$, $b^{\mathcal{A}} = 3$, $b^{\mathcal{B}} = 1$, and $f^{\mathcal{A}} = f^{\mathcal{B}}$, where $\mathcal{A}[f] : \{1, 2, 3\} \longrightarrow \{1, 2, 3\}$ is defined by:

$f^{\mathcal{A}}[1] = 2$, $f^{\mathcal{A}}[2] = 3$, $f^{\mathcal{A}}[3] = 1$.

Now, $\mathcal{A}$ and $\mathcal{B}$ agree on $f(x)$ and on $f(f(a))$ because they agree on f, a and x. However, they do not agree on $f(b)$ because they do not give the same value to b.

Theorem 2.3.17 (The Agreement Theorem for Terms) *Let $\mathcal{A}$ and $\mathcal{B}$ be two structures and t be a term. If $\mathcal{A}$ and $\mathcal{B}$ agree on t then $\mathcal{A}[t] = \mathcal{B}[t]$.*

Proof: We prove it by structural induction on t.

Case 1. t is a variable.

Then $t = x_i$ for some $i \in N$. So, $\mathcal{A}[x_i] = \mathcal{B}[x_i]$ because $\mathcal{A}$ and $\mathcal{B}$ agree on t.

Case 2. t is a constant.

Then $t = f_i^0$ for some $i \in N$. So, $\mathcal{A}[f_i^n] = \mathcal{B}[f_i^n]$ since $\mathcal{A}$ and $\mathcal{B}$ agree on t.

Case 3. $t = f_i^n(t_1, \ldots, t_n)$. Then

$\mathcal{A}[t]$

$= \mathcal{A}[f_i^n(t_1, \ldots, t_n)] =$

$= \mathcal{A}[f_i^n][\mathcal{A}[t_1], \ldots, \mathcal{A}[t_n]]$ from the $\mathcal{A}$ interpretation of $f_i^n(t_1, \ldots, t_n)$

$= \mathcal{A}[f_i^n][\mathcal{B}[t_1], \ldots, \mathcal{B}[t_n]]$ from the induction hypothesis applied to $t_1, \ldots, t_n$

$= \mathcal{B}[f_i^n][\mathcal{B}[t_1], \ldots, \mathcal{B}[t_n]]$ since $\mathcal{A}$ and $\mathcal{B}$ agree on f_i^n

$= \mathcal{B}[f_i^n(t_1, \ldots, t_n)]$ from the $\mathcal{B}$ interpretation of $f_i^n(t_1, \ldots, t_n)$

$= \mathcal{B}[t]$. **Q. E. D.**

Now let us define *agreement on a formula*.

Definition 2.3.18 (agreement on a formula) *Let $\mathcal{A}$ and $\mathcal{B}$ be two structures with the same universe D, and F be a formula. We say that $\mathcal{A}$ and $\mathcal{B}$ agree on F if they satisfy the conditions below.*

1. *For all free variables x that occur in F, $x^{\mathcal{A}} = x^{\mathcal{B}}$.*
2. *For all function symbols f that occur in F, $f^{\mathcal{A}} = f^{\mathcal{B}}$.*
3. *For all predicate symbols P in F, $P^{\mathcal{A}} = P^{\mathcal{B}}$.*

Observation 2.3.19 1. The second condition tells us that $\mathcal{A}$ and $\mathcal{B}$ must agree on the function symbols of arity 0 (the constants) *and* on the function symbols of arity greater than 0, i.e. the proper functions.

2. The third condition tells us that $\mathcal{A}$ and $\mathcal{B}$ must agree on the predicate symbols of arity 0 (the predicate constants) *and* on the predicate symbols of arity greater than 0, i.e. the proper predicates.

3. The structures $\mathcal{A}$ and $\mathcal{B}$ can differ on the variables that are not free in F. These are the variables that do not occur in F or they have only bound occurrences.

Now let us prove that whenever two structures agree on a formula, they interpret it the same way.

Theorem 2.3.20 (The Agreement Theorem) *Let $\mathcal{A}$ and $\mathcal{B}$ be two structures with the same universe D. If F and G agree on F then $\mathcal{A}[F] = \mathcal{B}[F]$.*

Proof: The proof is by structural induction on F.
Case 1: F is an atomic formula. According to Definition 2.1.5 we have 3 subcases.

Subcase 1.1: F is a predicate constant, i.e. a predicate symbol of arity 0. Let $F = P$. Then

$\mathcal{A}[P] = \mathcal{A}^p[P]$
$= \mathcal{B}^p[P]$ because $\mathcal{A}$ and $\mathcal{B}$ agree on $F = P$
$= \mathcal{B}[P]$.

Subcase 1.2: $F = P(t_1, \ldots, t_n)$. Then, all variables in the terms, if there are any, are free. Since $\mathcal{A}$ and $\mathcal{B}$ agree on F they agree on the terms of F. By Theorem 2.3.17 they give the same meaning to the terms, i.e. for all $1 \leq i \leq n$, $\mathcal{A}[t_i] = \mathcal{B}[t_i]$. Then

$\mathcal{A}[P(t_1, \ldots, t_n)$
$= \mathcal{A}[P][\mathcal{A}[t_1], \ldots, \mathcal{A}[t_n]]$ by the interpretation of $P(t_1, \ldots, t_n)$
$= \mathcal{B}[P][\mathcal{A}[t_1], \ldots, \mathcal{A}[t_n]]$ since $\mathcal{A}$ and $\mathcal{B}$ agree on P
$= \mathcal{B}[P][\mathcal{B}[t_1], \ldots, \mathcal{B}[t_n]]$ by Theorem 2.3.17
$= \mathcal{B}[P(t_1, \ldots, t_n)$ from the interpretation of $P(t_1, \ldots, t_n)$.

Subcase 1.3. $F = E(t_1, t_2)$

Since $\mathcal{A}$ and $\mathcal{B}$ agree on F, they agree on t_1 and t_2. So, by Theorem 2.3.17, they give the same value to t_1 and t_2. Then

$\mathcal{A}[E(t_1, t_2)] = 1$
iff $\mathcal{A}[t_1] = \mathcal{A}[t_2]$ from the $\mathcal{A}$ interpretation of $E(t_1, t_2)$
iff $\mathcal{B}[t_1] = \mathcal{B}[t_2]$ by Theorem 2.3.17 applied to t_1 and t_2
iff $\mathcal{B}[E(t_1, t_2)]$.

Case 2: $F = \neg G$. Since $\mathcal{A}$ and $\mathcal{B}$ agree on F then they agree on G. So, we can apply the induction hypothesis to G and get that $\mathcal{A}[G] = \mathcal{B}[G]$. Then

$\mathcal{A}[\neg G]$
$= \boxed{\neg}\mathcal{A}[G]$ from the interpretation of $\neg$
$= \boxed{\neg}\mathcal{B}[G]$ by the induction hypothesis
$= \mathcal{B}[\neg G]$ from the interpretation of $\neg$.

Case 3. $F = G \vee H$

Since $\mathcal{A}$ and $\mathcal{B}$ agree on F, they agree on both G and H. So we can apply the induction hypothesis to these two formulas and get that $\mathcal{A}[G] = \mathcal{B}[H]$ and $\mathcal{A}[H] = \mathcal{B}[H]$. Now,

$\mathcal{A}[G \vee H]$
$= \mathcal{A}[G] \boxed{\vee} \mathcal{A}[H]$ from the $\mathcal{A}$ interpretation of $G \vee H$
$= \mathcal{B}[G] \boxed{\vee} \mathcal{B}[H]$ from the induction hypothesis
$= \mathcal{B}[G \vee H]$ from the $\mathcal{B}$ interpretation of $G \vee H$.

Cases 4. $F = G \wedge H$, 5. $F = G \longrightarrow H$, and 6. $F = G \longleftrightarrow H$

are similar to Case 3. They are left as exercises.

Case 7. $F = \forall x G$

Here we have a little problem since $\mathcal{A}$ and $\mathcal{B}$ agreeing on F does not guarantee that they agree on G. The reason is that x is not free in F but it may be free after we remove the quantifier $\forall x$. So, while the agreement on F allows $x^{\mathcal{A}} \neq x^{\mathcal{B}}$, the agreement on G does not. However, we can say with certainty that for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}$ and $\mathcal{B}_{[x \leftarrow d]}$ agree on G. Then, by the induction hypothesis, $\mathcal{A}_{[x \leftarrow d]}[G] = \mathcal{B}_{[x \leftarrow d]}[G]$. Now, we can show that $\mathcal{A}[F] = 1$ iff $\mathcal{B}[F] = 1$.

$\mathcal{A}[\forall x G] = 1$
iff
for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[G] = 1$ from the $\mathcal{A}$ interpretation of $\forall x G$
iff
for all $d \in D$, $\mathcal{B}_{[x \leftarrow d]}[G] = 1$ from the induction hypothesis
iff
$\mathcal{B}[\forall x G] = 1$ from the $\mathcal{B}$ interpretation of $\forall x G$

Case 8. $F = \exists x G$

This case is similar to Case 7. It is left as exercise. **Q.E.D.**

Observation 2.3.21 The proof for Case 7, $F = \forall x G$, contained a sequence of the type:

Statement 1
iff *Statement 2*
iff ...
iff *Statement n*

where *Statement 1*,..., *Statement n* are some statements in the meta-language, and iff stands for if and only if. So, iff works like a two way street: if *Statement 1* is true then *Statement 2* is true (the if part) and if *Statement 2* is true then *Statement 1* is true (the only if part).

Then the sequence
Statement 1 iff *Statement 2* iff ... iff *Statement n*
is a proof of *Statement 1* iff *Statement 2*.

This is easy to show. If *Statement 1* is true then we can go in the if direction and prove successively that *Statement 2* is true from *Statement 1* iff *Statement 2*,

Statement 3 is true from *Statement 2* iff *Statement 3*,

and so on until

Statement n is true from *Statement (n-1)* iff *Statement n*

This is the proof of if *Statement 1* then *Statement n*. For the proof of if *Statement n* then *Statement 1* we use the other side of the street, i.e. the *only if* part. So, assume that *Statement n* is true then

Statement (n-1) is true from *Statement (n-1)* iff *Statement n*

Statement (n-2) is true from *Statement (n-2)* iff *Statement (n-1)*

and so on until

Statement 1 is true from *Statement 1* iff *Statement 2*

So, here is the proof of if *Statement n* then *Statement 1*.

We will see more of this type of proofs in the next sections. Most of them will have the property that establishes iff written next to the second statement. So, we will write

Statement 1

iff

Statement 2 *the reason for the iff*

Now, let us recall the semantic equivalence of Section 1.5. We said that two formulas F and G are equivalent if for all truth assignments $\mathcal{A}$, $\mathcal{A}[F] = \mathcal{A}[G]$. We can also define the semantic equivalence of the FOL formulas.

Definition 2.3.22 (semantic equivalence) *We say that two formulas F and G are semantically equivalent if for all structures $\mathcal{A}$, $\mathcal{A}[F] = \mathcal{A}[G]$. We write $F \equiv G$.*

The $\equiv$ relation is an equivalence on the set of FOL formulas. The next lemma shows that $\equiv$ is a congruence for the operations $\neg$, $\vee$, $\wedge$,$\longrightarrow$, $\longleftrightarrow$, $\forall x$, $\exists x$.

Lemma 2.3.23 ($\equiv$ is a congruence) *The implications below hold for all formulas F, G, H, and variable x.*

1. $F \equiv G$ implies $\neg F \equiv \neg G$.

2a. $F \equiv G$ implies $F \vee H \equiv G \vee H$.

2b. $F \equiv G$ implies $H \vee F \equiv H \vee G$.

3a. $F \equiv G$ implies $F \wedge H \equiv G \wedge H$.

3b. $F \equiv G$ implies $H \wedge F \equiv H \wedge G$.

4a. $F \equiv G$ implies $F \longrightarrow H \equiv G \longrightarrow H$.

4b. $F \equiv G$ implies $H \longrightarrow F \equiv H \longrightarrow G$.

5a. $F \equiv G$ implies $F \longleftrightarrow H \equiv G \longleftrightarrow H$.

5b. $F \equiv G$ implies $H \longleftrightarrow F \equiv H \longleftrightarrow G$.

6. $F \equiv G$ implies $\forall x F \equiv \forall x G$.

7. $F \equiv G$ implies $\exists x F \equiv \exists x G$.

Proof: We will prove only 1, 3b, and 7 and leave the rest as exercises. So, let us assume that $F \equiv G$ and let $\mathcal{A}$ be a structure. Then,

$\mathcal{A}[\neg F] = \boxed{\neg}\mathcal{A}[F]$ by the interpretation of $\neg$

$= \boxed{\neg}\mathcal{A}[G]$ since $\mathcal{A}[F] = \mathcal{A}[G]$

$= \mathcal{A}[\neg G]$ by the interpretation of $\neg$

Since $\mathcal{A}$ is any structure, we conclude that $\neg F \equiv \neg G$.

Now let us prove 3b.

$\mathcal{A}[H \wedge F] = \mathcal{A}[H]\boxed{\wedge}\mathcal{A}[F]$ by the interpretation of $\wedge$

$= \mathcal{A}[H]\boxed{\wedge}\mathcal{A}[G]$ since $\mathcal{A}[F] = \mathcal{A}[G]$

$= \mathcal{A}[H \wedge G]$ by the interpretation of $\wedge$

Again, since the structure $\mathcal{A}$ is arbitrary, $H \wedge F \equiv H \wedge G$.

Finally, let us show 7.

$\mathcal{A}[\exists x F] = 1$

iff there is some $d \in |\mathcal{A}|$ such that $\mathcal{A}_{[x \leftarrow d]}[F] = 1$ by the interpretation of $\exists x$

iff there is some $d \in |\mathcal{A}|$ such that $\mathcal{A}_{[x \leftarrow d]}[G] = 1$ since $F \equiv G$,$\mathcal{A}_{[x \leftarrow d]}[F] = \mathcal{A}_{[x \leftarrow d]}[G]$

iff $\mathcal{A}[\exists x G] = 1$ by the interpretation of $\exists x$

Since $\mathcal{A}$ is arbitrary, $\exists x F \equiv \exists x G$. **Q.E.D.**

Now we can prove the substitution theorem for first order logic.

Theorem 2.3.24 (The Substitution Theorem) *Let $F \equiv G$ and let H be a formula that has one or more occurrences of F. Let H_1 be the formula obtained by replacing an occurrence of F in H by G. Then $H \equiv H_1$.*

Before we go on to prove the theorem, let us look at an example. We will prove, in the next section, that $\forall x \forall y I \equiv \forall y \forall x I$. So, let F be $\forall x \forall y I$ and G be $\forall y \forall x I$. Let $H = \forall x \forall y I \longrightarrow \forall x \forall y I$. We can replace the second occurrence of $F = \forall x \forall y I$ in H by $G = \forall y \forall x I$, and we get $H_1 = \forall x \forall y I \longrightarrow \forall y \forall x I$. The theorem tells us that

$H = \forall x \forall y I \longrightarrow \forall x \forall y I \equiv \forall x \forall y I \longrightarrow \forall y \forall x I = H_1$.

Now let us prove The Substitution Theorem.

Proof: First of all, we notice that whenever $H = F$, $H_1 = G$, so $H \equiv H_1$. This tells us that we have to look at the case when the occurrence of F that is being replaced is inside H, i.e. a proper subformula of H. The proof is by structural induction on H.

Case 1: H is an atomic formula. Then H has a single subformula, namely itself. So $F = G$ and $H = H_1$ by the remark above.

Case 2: $H = \neg I$. If $F \neq H$ then F must be an occurrence in I. Let I_1 be the result of replacing the occurrence of F in I by G. Then $H_1 = \neg I_1$. By induction hypothesis, $I \equiv I_1$. Then $\neg I \equiv \neg I_1$, by part 1 of the Lemma 2.3.23.

Case 3: $H = I \vee J$. If $F \neq H$ then the occurrence of F that is being replaced is a subformula of I or of J. We will prove the subcase when the replacement takes place in J and leave the other subcase as exercise. Let J_1 be the result of replacing that occurrence of F by G. Then $H_1 = I \wedge J_1$. By the induction

hypothesis, $J \equiv J_1$. From Lemma 2.3.23, part 2b, we get that $I \vee J \equiv I \vee J_1$, i.e. $H \equiv H_1$.
Cases 4: $H = I \wedge J$, 5:$H = I \longrightarrow J$, and 6: $H = I \longleftrightarrow J$ are similar to Case 3. They are left as exercises.
Case 7: $H = \forall x I$. If the occurrence of F that is being replaced is not H itself then F is a subformula of I. Let I_1 be the result of replacing that occurrence of F in I. By the induction hypothesis, $I \equiv I_1$. From Lemma 2.3.23, part 6, we get that $\forall x I \equiv \forall x I_1$, i.e. $H \equiv H_1$.
Case 8. $H = \exists x I$. The proof for this case is similar to the one for case 7. It is left as exercise. **Q.E.D.**
Exercises

Exercise 2.3.1 *Find the following interpretations of terms:*
1. $\mathcal{B}^f[x_{21}]$
2. $\mathcal{C}^f[f_2^3(f_1^1(f_4^0), f_2^4(f_3^0, x_4, f_{23}^0, x_1))]$
3. $\mathcal{E}^f[f_{23}^3(f_{21}^1(x_6), f_2^2(x_9, x_4), f_3^2(f_{34}^0, x_7))]$

where $\mathcal{B}$, $\mathcal{C}$, $\mathcal{E}$ are the structures from Examples 2.3.2.

Exercise 2.3.2 *Find the following interpretations of atomic formulas:*
1. $\mathcal{B}[P_2^3(x_2, f_4^0, f_5^0)]$
2. $\mathcal{C}[E(f_3^1(x_7), f_2^0)]$
3. $\mathcal{E}[P_{23}^2(f_{22}^0, x_{21})]$
4. $\mathcal{B}[P_{101}^0]$

where $\mathcal{B}$, $\mathcal{C}$, $\mathcal{E}$ are the structures from Examples 2.3.2.

Exercise 2.3.3 *Find the following interpretations of formulas:*
1. $\mathcal{B}[P_2^0 \longrightarrow P_2^1(f_{23}^0)]$
2. $\mathcal{C}[P_2^1(f_{23}^0) \wedge P_3^2(x_5, x_2)]$
3. $\mathcal{E}[P_2^2(f_1^1(f_4^0), x_4) \vee P_2^0]$
4. $\mathcal{E}[P_6^3(f_3^0, x_7, f_8^0) \longleftrightarrow E(x_7, f_7^0)]$

where $\mathcal{B}$, $\mathcal{C}$, $\mathcal{E}$ are the structures from Examples 2.3.2.

Exercise 2.3.4 *Let $\mathcal{A}$ be a structure with universe D, d_1,d_2 be elements in D, x, y be variables and F be a formula. Prove that $\mathcal{A}_{[x \leftarrow d_1][x \leftarrow d_2]}[F] = \mathcal{A}_{[x \leftarrow d_2]}[F]$*

Exercise 2.3.5 *Find the following interpretations of formulas:*
1. $\mathcal{B}[\forall x_2 P_2^4(x_2, x_2, x_2, x_2)]$
2. $\mathcal{B}[\exists x_1 P_2^3(x_1, x_2, x_1)]$
3. $\mathcal{E}[\forall x_3(P_3^1(x_3) \wedge P_3^2(x_3, x_{10}))]$
4. $\mathcal{E}[\exists x_2 P_3^2(x_0, f_1^2(x_2, x_3))]$

where $\mathcal{B}$, $\mathcal{C}$, $\mathcal{E}$ are the structures from Examples 2.3.2.

Exercise 2.3.6 *Find the following interpretations of formulas:*
1. $\mathcal{E}[\forall x_1 \exists x_2 P_1^2(x_1, x_2)]$
2. $\mathcal{B}[\exists x_1 \forall x_2 P_2^3(x_1, x_2, x_2)]$
3. $\mathcal{E}[\exists x_1 \forall x_2 P_1^2(x_1, x_2)]$

where $\mathcal{B}$, $\mathcal{C}$, $\mathcal{E}$ are the structures from Examples 2.3.2.

u	$f^{\mathcal{A}}[u]$
3	3
4	5
5	4

u \ v	3	4	5
3	5	3	4
4	3	4	5
5	4	5	3

$g^{\mathcal{A}}[u, v]$

u \ v	3	4	5
3	1	0	0
4	0	1	1
5	0	1	0

$P^{\mathcal{A}}[u, v]$

Figure 2.23: Tables for Exercise 2.3.8

Exercise 2.3.7 *Use the method employed in Example 2.3.13 to compute the following interpretations:*

1. $\mathcal{B}[\exists x \forall y P_2^3(x, y, x)]$
2. $\mathcal{C}[\exists x \forall y (P_2^1(x) \vee \neg P_2^2(x, y))]$
3. $\mathcal{B}[\forall x \forall y \exists z P_1^3(x, y, z)]$.

The structures $\mathcal{B}$ and $\mathcal{C}$ are defined in Examples 2.3.2.

Exercise 2.3.8 *The structure $\mathcal{A}$ has universe $D = \{3, 4, 5\}$, $x^{\mathcal{A}} = 5$, $y^{\mathcal{A}} = 3$, and $a^{\mathcal{A}} = 4$. The interpretations of the unary function f, binary function g, and binary predicate P are shown in Figure 2.3.8.*

Find the interpretation of the terms and formulas below.

1. $\mathcal{A}[g(f(x), x)] =$
2. $\mathcal{A}[f(g(a, y))] =$
3. $\mathcal{A}[P(x, y)] =$
4. $\mathcal{A}[\forall x P(x, f(x))] =$
5. $\mathcal{A}[\exists x \forall y P(f(x), y)] =$

Exercise 2.3.9 *The universe of the structure $\mathcal{A}$ is $\{3, 4, 5\}$. The $\mathcal{A}$ interpretations of a, x, and y are $a^{\mathcal{A}} = 4$, $x^{\mathcal{A}} = 5$, $y^{\mathcal{A}} = 3$. The tables for the functions $f^{\mathcal{A}}$ and $g^{\mathcal{A}}$ and the predicates $P^{\mathcal{A}}$ and $Q^{\mathcal{A}}$ are shown in Figure 2.24. Evaluate the terms and formulas below.*

1. $\mathcal{A}[f(x)] =$
2. $\mathcal{A}[g(x, x)] =$
3. $\mathcal{A}[g(g(a, y), f(x))] =$
4. $\mathcal{A}[Q(g(y, x))] =$
5. $\mathcal{A}[P(x, f(a))] =$
6. $\mathcal{A}[E(f(x), g(y, a))] =$
7. $\mathcal{A}[\exists x Q(f(x))] =$
8. $\mathcal{A}[Q(x) \longrightarrow P(y, a)] =$
9. $\mathcal{A}[\forall x \exists y P(y, x)] =$
10. $\mathcal{A}[\exists y \forall x P(y, x)] =$

Exercise 2.3.10 *Show that the relation agreeing on F is reflexive, symmetric and transitive.*

x	$f^{\mathcal{A}}[x]$
3	4
4	3
5	3

$f^{\mathcal{A}}$

x \ y	3	4	5
3	3	4	5
4	4	5	3
5	5	3	4

$g^{\mathcal{A}}$

x \ y	3	4	5
3	1	0	0
4	0	1	1
5	0	1	0

$P^{\mathcal{A}}$

x	$Q^{\mathcal{A}}[x]$
3	0
4	0
5	1

$Q^{\mathcal{A}}$

Figure 2.24: Tables for Exercise 2.3.9

Exercise 2.3.11 *Prove/disprove the following statements:*

1. *If $\mathcal{A}$ and $\mathcal{B}$ agree on a term t they agree on all the subterms of t.*
2. *If $\mathcal{A}$ and $\mathcal{B}$ agree on a formula F they agree on all the subformulas of F.*

Exercise 2.3.12 *Let $\mathcal{A}$ and $\mathcal{B}$ be two structures with the same universe that agree on $\forall xF$. Let d be an element of their common universe. Prove that $\mathcal{A}_{[x \leftarrow d]}$ and $\mathcal{B}_{[x \leftarrow d]}$ agree on G.*

Exercise 2.3.13 *Do cases 4, 5, and 6 of the proof of The Agreement Theorem.*

Exercise 2.3.14 *Do case 8 of the proof of the Agreement Theorem.*

Exercise 2.3.15 *Prove that $\equiv$ is an equivalence relation on the set of first order formulas.*

Exercise 2.3.16 *On the set of structures we define the relation $\sim$ by $\mathcal{A} \sim \mathcal{B}$ iff for all predicate constants P, $P^{\mathcal{A}} = P^{\mathcal{B}}$.*

1. *Show that $\sim$ is reflexive, symmetric, and transitive.*
2. *Show that the equivalence classes of this relation correpond to the truth assignments from the propositional logic.*

Exercise 2.3.17 *Complete the proof of Lemma 2.3.23.*

Exercise 2.3.18 *Complete the proof of The Substitution Theorem.*

2.4 FOL Semantic Equivalences

The preceding section introduced the notion of semantic equivalence. Here, we present the equivalences needed for computing normal forms, the topic of the next section.

First of all, the semantic equivalences from Table 1.47 are also true for the first order predicate calculus. In Section 1.5 we used truth tables to prove them. Do the truth tables work here? We claim that they do. We will explain

F	G	F	$\wedge$	$(F$	$\vee$	$G)$	F
0	0	0	$\underline{0}$	0	0	0	$\underline{0}$
0	1	0	$\underline{0}$	0	1	1	$\underline{0}$
1	0	1	$\underline{1}$	1	1	0	$\underline{1}$
1	1	1	$\underline{1}$	1	1	1	$\underline{1}$

Figure 2.25: The truth table for $F \wedge (F \vee G) \equiv F$

1. $\forall x \forall y F \equiv \forall y \forall x F$
2. $\exists x \exists y F \equiv \exists y \exists x F$

Figure 2.26: Swapping quantifiers

the reasoning by following an example. Let us look at the truth table of the equivalence $F \wedge (F \vee G) \equiv F$, shown in Figure 2.25. In propositional logic we have an infinite number of truth assignments. However, we are only interested in their behavior with regard to F and to G, and all truth assignments are partitioned into 4 classes according to the values they give to F and to G. Each class corresponds to one of the lines shown in Figure 2.25. The equivalence holds if for all 4 lines the values of $F \wedge (F \vee G)$ and F are the same.

Now, let us assume that F and G are formulas in first order logic. Here we have structures, which are much more complicated than the truth assignments. However, as far as F and G are concerned, every structure assigns them a 0 or a 1. So, again, each structure falls into one of the 4 rows of Figure 2.25. The connectives $\wedge$ and $\vee$ are interpreted the same way, so Table 2.25 is also a valid proof in first order logic.

However, there are many useful equivalences that cannot be proven with truth tables, because the truth tables are unable to capture the meaning of the quantifiers. Let us look at the equivalences from Figure 2.26.

We give the proof of the first equivalence and leave the second proof as exercise.

Proof: If $x = y$, then the two formulas are identical[2], so they are equivalent.

So, let us assume that $x \neq y$. Let $\mathcal{A}$ be a structure with universe D. Then

$\mathcal{A}[\forall x \forall y F] = 1$

iff for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[\forall y F] = 1$ from the interpretation of $\forall x$

iff for all $d \in D$, for all $e \epsilon D$, $\mathcal{A}_{[x \leftarrow d][y \leftarrow e]}[F] = 1$ by the interpretation of $\forall y$

iff for all $d \in D$, for all $e \in D$, $\mathcal{A}_{[y \leftarrow e][x \leftarrow d]}[F] = 1$ $x \neq y$ implies $\mathcal{A}_{[x \leftarrow d][y \leftarrow e]}[F] = \mathcal{A}_{[y \leftarrow e][x \leftarrow d]}[F]$

iff for all $e \in D$, for all $d \in D$, $\mathcal{A}_{[y \leftarrow e][x \leftarrow d]}[F] = 1$

iff for all $e \in D$, $\mathcal{A}_{[y \leftarrow e]}[\forall x F] = 1$ by the interpretation of $\forall x$

iff $\mathcal{A}[\forall y \forall x F] = 1$ by the interpretation of $\forall y$.

[2] x and y are meta-variables that denote the variables of the language. As such they can be equal or different.

1. $\neg\forall x F \equiv \exists x \neg F$
2. $\neg\exists x F \equiv \forall x \neg F$

Figure 2.27: Pushing the negation inside

If x is not free in G then
1. $\forall x F \wedge G \equiv \forall x (F \wedge G)$
2. $\forall x F \vee G \equiv \forall x (F \vee G)$
3. $\exists x F \wedge G \equiv \exists x (F \wedge G)$
4. $\exists x F \vee G \equiv \exists x (F \vee G)$

Figure 2.28: Equivalences for advancing quantifiers

Now, let us provide a proof that the step
for all $d \in D$, for all $e \in D$, $\mathcal{A}_{[y \leftarrow e][x \leftarrow d]}[F] = 1$
iff for all $e \in D$, for all $d \in D$, $\mathcal{A}_{[y \leftarrow e][x \leftarrow d]}[F] = 1$.
is correct. We will show that the first statement is false iff the second one is.

The first statement is false iff
(1) for some pair $d, e \in D$, $\mathcal{A}_{[y \leftarrow e][x \leftarrow d]}[F] = 0$.
The second statement is false iff
(2) for some pair $e, f \in D$, $\mathcal{A}_{[y \leftarrow e][x \leftarrow d]}[F] = 0$.
Since (1) and (2) are the same condition, the step is correct.
Q.E.D.

The two equivalences from Figure 2.27 push the negation inside a quantified formula. Let us prove the second equivalence and leave the first one as exercise.
Proof: Let $\mathcal{A}$ be structure with universe D. Then
$\mathcal{A}[\neg\exists x F] = 1$
iff $\mathcal{A}[\exists x F] = 0$ from the interpretation of $\neg$
iff for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[F] = 0$ from the interpretation of $\exists x$
iff for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[\neg F] = 1$ from the interpretation of $\neg$
iff $\mathcal{A}[\forall x \neg F] = 1$ from the interpretation of $\forall x$.

Since $\mathcal{A}$ is an arbitrary structure we conclude that $\neg\exists x F \equiv \forall x \neg F$. **Q.E.D.**
The equivalences of Figure 2.28, advance the quantifiers past and's and or's.
Let us prove the first equivalence and let us leave the rest as exercises.
Proof: Let $\mathcal{A}$ be a structure with universe D. Then
$\mathcal{A}[\forall x F \wedge G] = 1$
iff $\mathcal{A}[\forall x F] = 1$ and $\mathcal{A}[G] = 1$ from the interpretation of $\wedge$
iff for all $d \in D$ $\mathcal{A}_{[x \leftarrow d]}[F] = 1$, and $\mathcal{A}[G] = 1$ by the interpretation of $\forall x$
iff for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[F] = 1$ and $\mathcal{A}[G] = 1$
iff for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[F] = 1$ and $\mathcal{A}_{[x \leftarrow d]}[G] = 1$ since $\mathcal{A}[G] = \mathcal{A}_{[x \leftarrow d]}[G]$
iff for all $d \in D$ $\mathcal{A}_{[x \leftarrow d]}[F \wedge G] = 1$ from the interpretation of $\wedge$
iff $\mathcal{A}[\forall x (F \wedge G)] = 1$ from the interpretation of $\forall x$.

If x is not free in F then
1. $F \wedge \forall x G \equiv \forall x(F \wedge G)$
2. $F \vee \forall x G \equiv \forall x(F \vee G)$
3. $F \wedge \exists x G \equiv \exists x(F \wedge G)$
4. $F \vee \exists x G \equiv \exists x(F \vee G)$

Figure 2.29: More equivalences for advancing quantifiers

$1. \forall x(F \wedge G) \equiv \forall x F \wedge \forall x G$
$2. \exists x(F \vee G) \equiv \exists x F \vee \exists x G$

Figure 2.30: Equivalences for distributing quantifiers

Here, we want to provide two explanations. The first is the proof that the step

for all $d \in D$ $\mathcal{A}_{[x \leftarrow d]}[F] = 1$, and $\mathcal{A}[G] = 1$ by the interpretation of $\forall x$

iff for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[F] = 1$ and $\mathcal{A}[G] = 1$

is correct. The difference between the two statements is that, in the first $\mathcal{A}[G] = 1$ is outside the loop

for all $d \in D$

while in the second it is inside.

We will show that the first statement is true iff the second is.

$\Longrightarrow$: Assume that the first statement is true and let $d \in D$. Then $\mathcal{A}_{[x \leftarrow d]}[F] = 1$, $\mathcal{A}[G] = 1$, so $\mathcal{A}_{[x \leftarrow d]}[F] = 1$ and $\mathcal{A}[G] = 1$ is also true. Since d is arbitrary, the second statement is true.

$\Longleftarrow$: Assume that the second statement is true. Then, for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[F] = 1$ and $\mathcal{A}[G] = 1$ is true. So,

for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[F] = 1$

and $\mathcal{A}[G] = 1$ are both true. So, the first statement is true.

The second explanation concerns $\mathcal{A}[G] = \mathcal{A}_{[x \leftarrow d]}[G]$. Since x is not free in G, $\mathcal{A}$ and $\mathcal{A}_{[x \leftarrow d]}$ agree on all function symbols, predicate symbols, and free variables of G. By The Agreement Theorem, $\mathcal{A}[G] = \mathcal{A}_{[x \leftarrow d]}[G]$. **Q.E.D.**

Let us switch the roles of the formulas F and G from Table 2.28, i.e. let G be the formula that is quantified by x. We get the (conditional) equivalences from Figure 2.29. These equivalences are easy to prove. Let's show 3.

Proof: Assume that x is not free in F.

$F \wedge \exists x G$
$\equiv \exists x G \wedge F$ from the commutativity of $\wedge$
$\equiv \exists x(G \wedge F)$ from part 3 of Table 2.28
$\equiv \exists x(F \wedge G)$ from the commutativity of $\wedge$

We did not use a semantic proof involving structures. Instead, we used the equivalences already established, and the properties of the equivalence relation. **Q.E.D.**

The equivalences from Figure 2.30 distribute the quantifiers over connectives. Let us prove the first equivalence and leave the second one as exercise.

Proof: Let $\mathcal{A}$ be a structure with universe D. Then

$$1.\exists x \forall y F \equiv \forall y \exists x F$$
$$2.\exists x (F \wedge G) \equiv \exists x F \wedge \exists x G$$
$$3.\forall x (F \vee G) \equiv \forall x F \vee \forall x G$$

Figure 2.31: Equivalences that are not always true

$\mathcal{A}[\forall x(F \wedge G)] = 1$
iff for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[F \wedge G] = 1$ from the interpretation of $\forall x$
iff for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[F] = 1$ and $\mathcal{A}_{[x \leftarrow d]}[G] = 1$ by the interpretation of $\wedge$
iff for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[F] = 1$ and for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[G] = 1$
iff $\mathcal{A}[\forall x F] = 1$ and for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[G] = 1$ by the interpretation of $\forall x$
iff $\mathcal{A}[\forall x F] = 1$ and $\mathcal{A}[\forall x G] = 1$ by the interpretation of $\forall x$
iff $\mathcal{A}[\forall x F \wedge \forall x G] = 1$ by the interpretation of $\wedge$.

We will now give an explanation for the step
for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[F] = 1$ and $\mathcal{A}_{[x \leftarrow d]}[G] = 1$ by the interpretation of $\wedge$
iff for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[F] = 1$ and for $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[G] = 1$.
The difference between them is that the first statement has only one for loop, while the second has two. Let us show that whenever one statement is false, so is the second.
The first statement is false
iff
there is some $d \in D$, such that at least one of $\mathcal{A}_{[x \leftarrow d]}[F]$, $\mathcal{A}_{[x \leftarrow d]}[G]$ is 0
iff
at least one of the 2 loops,
for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[F] = 1$,
for $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[G] = 1$
is false
iff the second statement is false. **Q.E.D.**

So, far we presented methods for showing equivalences. But how do we show that an equivalence does not hold?

For example, the equivalences[3] from Figure 2.31 are not always true. Before we disprove them, let us introduce the concepts of model and countermodel.

Definition 2.4.1 (model, countermodel, satisfiability, tautology, unsatisfiability) *Let $\mathcal{A}$ be a structure with universe D and F be a formula. We say that $\mathcal{A}$ is a model of F, or $\mathcal{A}$ satisfies F and write $\models_{\mathcal{A}} F$, when $\mathcal{A}[F] = 1$. If D is finite we say that $\mathcal{A}$ is a finite model of F.*

[3]Recall that both sides of the $\equiv$ sign are meta-formulas that contain meta-variables. The formulas are obtained by assigning values to the meta-variables. The resulting equivalence may hold or not, depending on the values of the meta-variables.

When $\mathcal{A}[F] = 0$, we say that $\mathcal{A}$ is a countermodel of F or $\mathcal{A}$ does not satisfy F, and we write $\not\models_{\mathcal{A}} F$. If D is finite we say that $\mathcal{A}$ is a finite countermodel of F.

A formula is satisfiable when it has a model; it is unsatisfiable when it does not. We write $\not\models F$ to show that F is unsatisfiable.

A formula F is a tautology when every structure is a model of F. We write $\models F$ to show that F is a tautology.

We are ready to disprove the equivalences from Figure 2.31. We will do the second one, and leave the other two as exercises.

Proof: We will find a structure that is a model for the right hand side and a countermodel for the left hand side of the equivalence.

Let $\mathcal{A}$ be a structure with universe $D = \{3, 4\}$ and P be a unary predicate defined by $P^{\mathcal{A}}[3] = 1$ and $P^{\mathcal{A}}[4] = 0$, $F = P(x)$ and $G = \neg P(x)$. Then

$\mathcal{A}[\exists x(F \wedge G)]$
$= \mathcal{A}[\exists x(P(x) \wedge \neg P(x))]$
$= \mathcal{A}[\exists x \square]$ because $P(x) \wedge \neg P(x)$ is unsatisfiable
$= \mathcal{A}[\square]$ contradiction law
$= 0$ since $\square$ is unsatisfiable.

At the same time,

$\mathcal{A}[\exists x F \wedge \exists x G]$
$= \mathcal{A}[\exists x P(x) \wedge \exists x \neg P(x)]$
$= \mathcal{A}[\exists x P(x)] \boxed{\wedge} \mathcal{A}[\exists x \neg P(x)]$ by the interpretation of $\wedge$
$= [P^{\mathcal{A}}[3] \vee P^{\mathcal{A}}[4]] \boxed{\wedge} \mathcal{A}[\exists x \neg P(x)]$ because $D = \{3, 4\}$
$= [1 \vee P^{\mathcal{A}}[4]] \boxed{\wedge} \mathcal{A}[\exists x \neg P(x)]$ from the definition of $P^{\mathcal{A}}$
$= 1 \boxed{\wedge} \mathcal{A}[\exists x \neg P(x)]$ by lazy evaluation
$= \mathcal{A}[\exists x \neg P(x)]$ a property of $\boxed{\wedge}$
$= \mathcal{A}_{[x \leftarrow 3]}[\neg P(x)] \boxed{\vee} \mathcal{A}_{[x \leftarrow 4]}[\neg P(x)]$ since $D = \{3, 4\}$
$= \boxed{\neg} P^{\mathcal{A}}[3] \boxed{\vee} \boxed{\neg} P^{\mathcal{A}}[4]$
$= \boxed{\neg} 1 \boxed{\vee} \boxed{\neg} 0$ from the definition of $P^{\mathcal{A}}$
$= 0 \boxed{\vee} 1$
$= 1$. **Q.E.D.**

Observation 2.4.2 At this point one might ask: How do I know to look for a structure that is a model for the right-hand-side and a countermodel for the left-hand-side and not the other way, i.e. $\mathcal{A}$ satisfies the left-side and does not satisfy the right one?

The intuitive answer is that when $\exists x(F \wedge G)$ is satisfied, *the same value of x* must make both F and G true, while $\exists x F \wedge \exists x G$ is true when F is satisfied by a value of x and G is satisfied by a value of x, but the two values *do not have to be the same*. So, the first formula has a stronger requirement than the second.

We have the same situation in English where the statements,

Susan has a boyfriend who is rich and handsome

does not have the same meaning as

Susan has a rich boyfriend and a handsome one,

because the rich one may not be the same person as the handsome one.

Definition 2.4.3 (satisfaction of a set of formulas, consequence) *1. Let S be a set of formulas. We say that a structure $\mathcal{A}$ satisfies S if $\mathcal{A}$ satisfies every formula in the set. We also say that $\mathcal{A}$ is a model of S, and write $\models_{\mathcal{A}} S$.*

2. We say that a formula G is a consequence of a set of formulas S if every structure that satisfies S is also a model for G. We write $S \models G$ to show that G is a consequence of S. When S is finite, say $S = \{F_1, \ldots, F_n\}$, we omit the braces and write $F_1, \ldots, F_n \models G$.

Example 2.4.4 Let us show that $\exists x(F \wedge G) \models (\exists x F \wedge \exists x G)$

Let $\mathcal{A}$ be a model for $\exists x(F \wedge G)$ and let D be the universe of $\mathcal{A}$. Then there is some $d \in D$ such that

$\mathcal{A}_{[x \leftarrow d]}[F \wedge G] = 1$.
This in turn implies that
$\mathcal{A}_{[x \leftarrow d]}[F] = 1$ and $\mathcal{A}_{[x \leftarrow d]}[G] = 1$.
From the last two equalities we infer that
$\mathcal{A}[\exists x F] = 1$ and $\mathcal{A}[\exists x G] = 1$,
which leads us to
$\mathcal{A}[\exists x G \wedge \exists x G] = 1$.

Example 2.4.5 Let us show that for $x \neq y$, $\exists x \forall y F \models \forall y \exists x F$.

Let $\mathcal{A}$ be a structure with universe D that models $\exists x \forall y F$.
Then, for some $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[\forall y F] = 1$.
From the interpretation of $\forall y F$, we get
for all $e \in D$, $\mathcal{A}_{[x \leftarrow d][y \leftarrow e]}[F] = 1$.
Since $x \neq y$,
$\mathcal{A}_{[x \leftarrow d][y \leftarrow e]}[F] = \mathcal{A}_{[y \leftarrow e][x \leftarrow d]}[F]$.
So, the statement
for all $e \in D$, $\mathcal{A}_{[y \leftarrow e][x \leftarrow d]}[F] = 1$
is also true. From the interpretation of $\exists x$, we get
for all $e \in D$, $\mathcal{A}_{[y \leftarrow e]}[\exists x F] = 1$.
Finally, we use the interpretation of $\forall y$ to get
$\mathcal{A}[\forall y \exists x F] = 1$.

Now a final observation.

Observation 2.4.6 There are many nonequivalences in which no side is a consequence of the other. For example, $P_1^0 \not\equiv P_2^0$, and neither $P_1^0 \models P_2^0$, nor $P_2^0 \models P_1^0$.

Exercises

Exercise 2.4.1 *Prove equivalence 2 of Figure 2.26.*

Exercise 2.4.2 *Prove equivalence 1 of Figure 2.27.*

Exercise 2.4.3 *Prove equivalences 2,3, 4 of Figure 2.28.*

Exercise 2.4.4 *Prove equivalence 2 of Figure 2.30.*

Exercise 2.4.5 *Prove the statements below.*
1. *If F is a tautology then $\forall xF$ is also a tautology.*
2. *If F is unsatisfiable then $\exists xF$ is also unsatisfiable.*

Exercise 2.4.6 *Disprove the first and the last equivalence of Figure 2.31.*

Exercise 2.4.7 *Display structures that disprove the following equivalences:*
1. $\exists xF \equiv \forall xF$
2. $\exists xF \equiv F$

Exercise 2.4.8 *Show that the set of connectives $S = \{F \longrightarrow G, \exists xF, \Box\}$ is adequate. The meta-variables F and G denote formulas and x variables.*

Exercise 2.4.9 *Show that the set of connectives $S = \{\neg F \wedge G, \forall xF\}$ is adequate.*

Exercise 2.4.10 *Show the following consequences:*
1. $\forall xF \models \exists xF$
2. $\exists xF, \forall xG \models \exists x(F \wedge G)$
3. $\forall x(F \vee G), \exists x\neg F \models \exists xG$.

2.5 Skolem Normal Forms

In this section we give an algorithm for computing *Skolem normal forms* for first order formulas. Along the way we present the definitions and the propositions that assure the correctness of our algorithm. We start by showing how to *eliminate redundant quantifiers* and how to *rectify* formulas. Then we compute a *prenex form* and later on we *close* it. After that we *Skolemize* the formula and we compute the *clause form* of the *matrix*. At the end we put together all this steps to form the algorithm and we apply it to compute 2 Skolem normal forms.

Definition 2.5.1 (redundant quantifier) *We say that Qx is redundant in $F = QxG$ if x is not free in G.*

Examples 2.5.2 1. The first $\forall x$ quantifier is redundant in $F = \forall x \forall x P(x, y, x)$ because x is not free in $G = \forall x P(x, y, x)$.

2. The quantifier $\exists x$ is redundant in $F = \exists x P(a, y, f(z))$ because x is not free (actually it does not occur) in $G = P(a, y, f(z))$.

The next proposition tells us that we can remove the redundant quantifiers without changing the meaning of the formula.

Proposition 2.5.3 (removal of the redundant quantifiers) *If Qx is redundant in QxF, then $QxF \equiv F$.*

Proof: We will prove the equivalence for $Q = \forall$ and leave the case $Q = \exists$ as exercise. Assume that $F = \forall xG$ and x is not free in G. Let $\mathcal{A}$ be a structure with universe D and d be an element of D. Since x is not free in G, $\mathcal{A}_{[x \leftarrow d]}$ and $\mathcal{A}$ agree on G. By The Agreement Theorem, $\mathcal{A}_{[x \leftarrow d]}[G] = \mathcal{A}[G]$. Now,

$\mathcal{A}[F] = 1$
iff for all $d \in D$, $\mathcal{A}_{[x \leftarrow d]}[G] = 1$ from the interpretation of $\forall$
iff for all $d \in D$, $\mathcal{A}[G] = 1$ from $\mathcal{A}_{[x \leftarrow d]}[G] = \mathcal{A}[G]$
iff $\mathcal{A}[G] = 1$ from the interpretation of $\forall x$.
Q.E.D.

Next, we present a key concept in the semantics of FOL logic.

Definition 2.5.4 (rectified formula) *The formula F is rectified if it has no redundant quantifiers, no variable occurs both free and bound and no variable is quantified more than once.*

Examples 2.5.5 1. The formula $G = \exists xP(x) \vee Q(x, y)$ is not rectified because x occurs both free and bound; the first occurrence of x is bound by $\exists x$ while the second one is free.

2. The formula $H = \exists xP(x) \vee \forall xQ(y, x)$ is not rectified because x is quantified twice, once by $\exists x$ and a second time by $\forall x$.

3. The formula $I = \forall xQ(y, z) \vee \exists uP(u, v)$ is not rectified because it has the redundant quantifier $\forall x$.

4. The formula $J = \forall x \exists y(P(x) \wedge (Q(x, y) \vee \neg P(z))$ is rectified since no variable occurs both bound and free. The variables x and y have only bound occurrences, z has only one (free) occurrence, and different quantifier occurrences have different variables.

We notice that rectified formulas can have free variables.

We will show that every formula can be rectified. In order to prove it, we introduce the notations $\tau[x/t]$ and $F[x/t]$ and the notion of *a term is free for a variable in a formula*.

Note 2.5.6 ($\tau[x/t]$, $F[x/t]$) 1. Let τ and t be terms and x be a variable. Then $\tau[x/t]$ is the term obtained from τ by replacing every occurrence of x by t.

2. Let F be a formula, x a variable and t a term. $F[x/t]$ is the formula obtained from F be replacing every free occurrence of x by t.

Examples 2.5.7 1. Let $\tau = f(x, g(z, x))$ and $t = h(y)$. We replace both x's of τ by $h(y)$ and get $\tau[x/t] = f(h(y), g(z, h(y)))$.

2. Let $F = (\forall xP(x, y) \longrightarrow Q(x, y)) \longleftrightarrow \neg Q(x, z)$ and $t = f(a)$. The x's that occur in the Q predicates are free, so we replace them by t. We get $F[x/t] = (\forall xP(x, y) \longrightarrow Q(f(a), y)) \longleftrightarrow \neg Q(f(a), z)$.

We did not change the first occurrence of x, in $\forall xP(x, y)$, because it is not free.

Definition 2.5.8 (a term is free for a variable) *Let x be a variable, t a term, and F a formula. Then t free for x in F if, $F[x/t]$ contains no extra bindings, i.e. no variable from t is bound by a quantifier from F.*

Examples 2.5.9 1. Let $F = \forall x P(x, y)$. Then, $f(x)$ is *not* free for y because $F[y/f(x)] = \forall x P(x, f(x))$ introduces a new binding; the second x in $F[y/f(x)]$ is bound by $\forall x$.

2. Let $F = \forall x P(x, y) \vee Q(z)$. Here, x is free for z since $F[x/z] = \forall x P(x, y) \vee Q(x)$ has no extra bindings.

Observation 2.5.10 The property *a term is free for a variable in a formula* is hereditary. This means that whenever t is free for x in F, it is free for x in *any* subformula of F.

The next lemma establishes a link between semantic substitutions, more precisely the assignment of values to the variables, and the syntactic (string) substitutions. It tells us that we can find the meaning of the term $\tau[x/t]$ in two ways. We can either substitute every x in τ by t and then interpret $\tau[x/t]$, or we can first interpret t, assign the value of t to x, and finally interpret τ in the new structure.

Lemma 2.5.11 (The Translation Lemma for Terms) *Let τ and t be terms, and $\mathcal{A}$ be a structure. Then $\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[\tau] = \mathcal{A}[\tau[x/t]]$.*

Proof: We prove it by structural induction on τ.
Case 1: τ is a variable. Here we have two subcases, depending on whether τ is x or not.

Subcase 1.1: $\tau = x$. Then $\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[x] = \mathcal{A}[t]$.

The term $x[x/t]$ is obtained from x by replacing x by t, so we get t. Then $\mathcal{A}[x[x/t]] = \mathcal{A}[t]$.

So, both $\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[\tau]$ and $\mathcal{A}[\tau[x/t]]$ are equal to $\mathcal{A}[t]$.

Subcase 1.2: $\tau \neq x$. Let $\tau = y$. Then, $\mathcal{A}[y[x/t] = \mathcal{A}[y]$.

At the same time,

$\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[y] = \mathcal{A}[y]$

because $\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}$ and $\mathcal{A}$ agree on y.

So, both $\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[\tau]$ and $\mathcal{A}[\tau[x/t]]$ are equal to $\mathcal{A}[y]$.

Case 2: τ is an individual constant. Let $\tau = a$. Then

$\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[a]$
$= \mathcal{A}[a]$ since $\mathcal{A}$ and $\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}$ agree on constants
$= \mathcal{A}[a[x/t]$ since $a[x/t] = a$.

So, the equality holds.

Case 3:$\tau = f(t_1, \ldots, t_n)$. In this case

$\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[f(t_1, \ldots, t_n]$
$= f^{\mathcal{A}}[\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[t_1], \ldots, \mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[t_n]$ by the interpretation of τ
$= f^{\mathcal{A}}[\mathcal{A}[t_1[x/t]], \ldots, \mathcal{A}[t_n[x/t]]$ by the IH applied to $t_1, \ldots, t_n$
$= \mathcal{A}[f(t_1[x/t], \ldots, t_n[x/t])]$ by the interpretation of τ
$= \mathcal{A}[f(t_1, \ldots, t_n)[x/t]]$ since x occurs only in the terms.

So, again, the equality holds. **Q.E.D.**

Now we can extend the lemma to formulas.

Lemma 2.5.12 (The Translation Lemma) *Let x be a variable, t a term, and F a formula. If t is free for x in F then*
$\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[F] = \mathcal{A}[F[x/t]$.

Before we go on to prove the lemma, let us see what it says. The lemma gives us two different paths for computing the truth value of $F[x/t]$. One path changes all free occurrences of x to t, and then interprets $F[x/t]$. The other path interprets t, assign that value to x, and then interprets F in the new structure. The substitution in the first path is syntactic because it is just string replacement. In the second path, the substitution is semantic because it modifies the structure. The lemma "translates" term substitutions to structure substitutions.

Now, let us prove Lemma 2.5.12.

Proof: The proof is by structural induction on F.

Case 1: F is an atomic formula. Here we have 3 subcases.

Subcase 1.1: $F = P_i^0$. Then
$\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[P_i^0]$
$= \mathcal{A}[P_i^0]$ since $\mathcal{A}$ and $\mathcal{A}_{[x \leftarrow \mathcal{A}[t]}$ agree on predicate symbols
$= \mathcal{A}[P_i^0[x/t]]$ since $P_i^0[x/t] = P_i^0$ because there are no x's in P_i^0.

Subcase 1.2 $F = P(t_1, \ldots, t_n)$
Then
$\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[F]$
$= \mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[P(t_1, \ldots, t_n)]$
$= P^{\mathcal{A}}[\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[t_1], \ldots, \mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[t_n]]$ by the interpretation of F
$= P^{\mathcal{A}}[\mathcal{A}[t_1[x/t]], \ldots, \mathcal{A}[t_n[x/t]]]$ by The Translation Lemma for Terms
$= \mathcal{A}[P(t_1[x/t], \ldots, t_n[x/t])]$
$= \mathcal{A}[P(t_1, \ldots, t_n)[x/t]]$
$= \mathcal{A}[F[x/t]$.

Subcase 1.3 $F = E(t_1, t_2)$
Then
$\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[E(t_1, t_2)] = 1$
iff $\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[t_1] = \mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[t_2]$ by the interpretation of E
iff $\mathcal{A}[t_1[x/t]] = \mathcal{A}[t_2[x/t]]$ by The Translation Lemma for Terms
iff $\mathcal{A}[E(t_1[x/t], t_2[x/t])] = 1$
iff $\mathcal{A}[E(t_1, t_2)[x/t]] = 1$

This concludes the proof of Case 1.

Case 2. $F = \neg G$
Then
$\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[F]$
$= \mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[\neg G]$
$= \boxed{\neg}\ \mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[G]$ from the interpretation of $\neg$
$= \boxed{\neg}\ \mathcal{A}[G[x/t]]$ by the induction hypothesis on G
$= \mathcal{A}[\neg G[x/t]]$ from the interpretation of $\neg$

$= \mathcal{A}[F[x/t]]$.

Case 3. $F = G \vee H$

Then

$\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[F]$

$= \mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[G \vee H]$

$= \mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[G] \boxed{\vee} \mathcal{A}_{[x \leftarrow \mathcal{A}[t]}[H]$

$= \mathcal{A}[G[x/t] \boxed{\vee} \mathcal{A}[H[x/t]]$ by the IH applied to G and H

$= \mathcal{A}[G[x/t] \vee H[x/t]]$ from the interpretation of $\vee$

$= \mathcal{A}[(G \vee H)[x/t]]$

$= \mathcal{A}[F]$.

Cases 4 ($F = G \wedge H$), 5 ($F = G \longrightarrow H$) and 6 ($F = G \longleftrightarrow H$) are left as exercises.

Case 7. $F = \forall y G$

If $y = x$ then we have no problem since

$\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[F]$

$= \mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[\forall x G]$

$= \mathcal{A}[\forall x G]$ since x is not free in $\forall x G$

$= \mathcal{A}[[\forall x G][x/t]]$ $[\forall x G][x/t] = \forall x G$, since x is not free in $\forall x G$

$= \mathcal{A}[F[x/t]]$

So let us assume that $y \neq x$. Since t is free for x in F, y **is not** a variable in t. Let D be the universe of $\mathcal{A}$. Then

$\mathcal{A}_{[x \leftarrow \mathcal{A}[t]]}[F] = 1$

iff for all $d \in D$ $\mathcal{A}_{[x \leftarrow \mathcal{A}[t]][y \leftarrow d]}[G] = 1$

iff for all $d \in D$ $\mathcal{A}_{[y \leftarrow d][x \leftarrow \mathcal{A}[t]]}[G] = 1$ because y is not in t

iff for all $d \in D$ $\mathcal{A}_{[y \leftarrow d]}[G[x/t]] = 1$ by IH on G

iff $\mathcal{A}[\forall y G[x/t] = 1$ from the interpretation of $\forall x$

iff $\mathcal{A}[F[x/t]] = 1$

Case 8. $F = \exists y G$

This case is similar to case 7 and is left as exercise. **Q.E.D.**

The Relabeling Lemma tells us that we can change the name of a quantified variable without changing the meaning of the formula.

Lemma 2.5.13 (The Relabeling Lemma) *Let $y \neq x$ be a variable that does not occur in F neither free nor as an argument to a quantifier. Then $QxF \equiv QyF[x/y]$, where Q is one of the quantifiers $\forall$ or $\exists$.*

Proof: First of all, y is free for x in F because there is no Qy to bind the occurrences of y. Let $\mathcal{A}$ be a structure with universe D. Then

$\mathcal{A}[QyF[x/y]] = 1$

iff for all/some $d \in D$ $\mathcal{A}_{[y \leftarrow d]}[F[x/y]] = 1$ write all for $\forall$, some for $\exists$

iff for all/some $d \in D$ $\mathcal{A}_{[y \leftarrow d][x \leftarrow \mathcal{A}_{[y \leftarrow d]}[y]]}[F]$ by The Translation Lemma applied to $\mathcal{A}_{[y \leftarrow d]}[F[x/y]]$

iff for all/some $d \in D$ $\mathcal{A}_{[y \leftarrow d][x \leftarrow d]}[F] = 1$ since $\mathcal{A}_{[y \leftarrow d]}[y] = d$

iff for all/some $d \in D$ $\mathcal{A}_{[x \leftarrow d][y \leftarrow d]}[F] = 1$ because $x \neq y$
iff for all/some $d \in D$ $\mathcal{A}_{[x \leftarrow d]}[F] = 1$ since y is not free in F
iff $\mathcal{A}[QxF] = 1$ by the interpretation of Q. **Q.E.D.**
Let us look at some applications of The Relabeling Lemma.

Examples 2.5.14 1. The variable x does not occur in $P(x, z)$, so $\forall x P(x, z) \equiv \forall y P(y, z)$.

2. The variable z does not occur in $F = \forall y P(x, y) \vee \neg Q(y)$ so, we can apply The Relabeling Lemma and get the equivalence $\exists y(\forall y P(x, y) \vee \neg Q(y)) \equiv \exists z(\forall y P(x, y) \vee \neg Q(z))$.

We replaced only the second occurrence of y by z because the first occurrence, in $P(x, y)$, is bound by $\forall y$.

3. The variable y occurs in $F = E(x, y)$, and we cannot apply The Relabeling Lemma to $\forall x F$. In fact the equivalence $\forall x F \equiv \forall y F[x/y]$ is false. Let us see why. The formula $\forall y F[x/y] = \forall y E(y, y)$ is a tautology, while $\forall x E(x, y)$ is false for all structures with more than 1 element. We notice that y is free for x in F, so this is not a sufficient condition for the lemma.

The Relabeling Lemma allows us to rectify formulas.

Lemma 2.5.15 (The Rectification Lemma) *Every formula F has an equivalent formula G that has no occurrences of $\longrightarrow$ and $\longleftrightarrow$ and is rectified.*

Proof: Let F be a FOL formula. We perform the 4 steps below.
Step 1: Eliminate the redundant quantifiers.

We keep applying the reduction
if x is not free in I then $QxI \Longrightarrow I$
to every occurrence QxI of F.

This simplification preserves the equivalence, since whenever x is not free in I, $QxI \equiv I$.

So, we get a formula F_1, equivalent to F, that has no redundant quantifiers.
Step 2: Eliminate $\longleftrightarrow$.

We repeatedly apply the rewrite
$I \longleftrightarrow J \Longrightarrow (I \longrightarrow J) \wedge (J \longrightarrow I)$
to the subformulas that have only one $\longleftrightarrow$.

This way, each rewrite decreases the number of $\longleftrightarrow$'s, so the algorithm terminates. Since $I \longleftrightarrow J \equiv (I \longrightarrow J) \wedge (J \longrightarrow I)$, the rewrites preserve equivalences and we obtain a formula F_2, equivalent to F_1, that has no $\longleftrightarrow$ connectives. We will show that F_2 has no redundant quantifiers by proving that every application of the rewrite rule does not introduce redundant quantifiers. So, let $H_1 \Longrightarrow H_2$ be the rewrite shown in Figure 2.32 and let us assume that H_1 has no redundant quantifier. Let v be the address of a quantifier, say Qx, of H_2. We have 2 cases, depending on the position of v relative to u.

Case 1: u is a prefix of v. Then v is inside one of the occurrences of I or J. Assume that it is located in an I-occurrence. If Qx is redundant in H_2, then it is redundant in I. Since I is a subformula of H_1 it is redundant in H_1, contradicting our assumption.

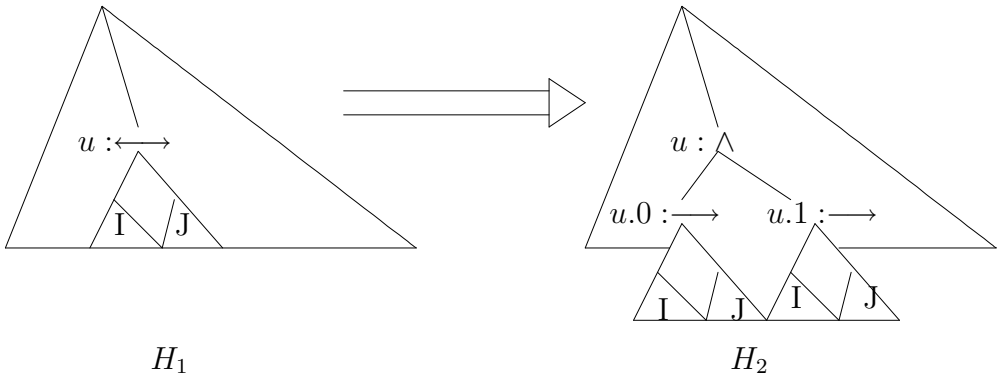


Figure 2.32: The reduction $I \longleftrightarrow J \equiv (I \longrightarrow J) \wedge (J \longrightarrow I)$

Case 2: u is not a prefix of v. Then, the label of v in H_1 is also Qx. Since H_1 has no redundant quantifiers, there is a path $P = w_0, \ldots, w_m$ from $v.0$ to a leaf of H_1 such that the label of w_m contains the variable x and there is no quantifier $Q'x$ on the path. We have 2 subcases, depending on whether P passes through u or not.

Subcase 2.1: the path does not pass through u. Then we have the situation shown in Figure 2.33. Here P, represented by the thick line, is also a path in H_2, so Qx is not redundant.

Subcase 2.2: the path passes through u. Then the path ends up in either I or in J. We construct a path P' in H_2 that starts from $v.0$ passes through $u.0$ and then follows P, i.e. if P goes left so does P'. If P goes right, so does P'. Formally, P has the property that $w_k = u$ for some k, $0 \leq k < m$, and for all $l \geq k$, $w_k = u.s_l$. Let us look at the labels of $P' = w_0, \ldots, w_k, u.0, u.0.s_{k+1}, \ldots, u.0.s_m$.

The labels of $w_0, \ldots w_{k-1}$ are the same as in H_1, the label of w_k is $\wedge$, the label of $u.0$ is $\longrightarrow$, and the labels of $u.0.s_{k+1}, \ldots, u.0.s_m$ are respectively the labels of $u.s_{k+1} = w_{k+1}, \ldots, u.s_m = w_m$. So, the labels of P' are the labels of P minus one occurrence of $\longleftrightarrow$ and plus one $\longrightarrow$ and one $\wedge$. Then P' has no quantifier $Q'x$ and the leaf contains x. Then, the quantifier at v is not redundant. This situation is illustrated in Figure 2.34.

Step 3: Eliminate $\longrightarrow$.

We keep replacing the occurrences $G \longrightarrow H$ of F_1 by $\neg G \vee H$ until we have no more $\longrightarrow$'s. Each rewrite reduces the number of $\longrightarrow$'s, so the step terminates. Since $G \longrightarrow H \equiv \neg G \vee H$, these rewrites preserve equivalences. So, we get a formula F_3, equivalent to F_2 that does not contain $\longrightarrow$'s. Since the simplification does not contain $\longleftrightarrow$'s, neither does F_3. The proof that F_3 does not have redundant quantifiers is similar to the one done in Step 2.

Step 4: Rectify the formula.

We keep applying the following two rules until the formula is rectified.

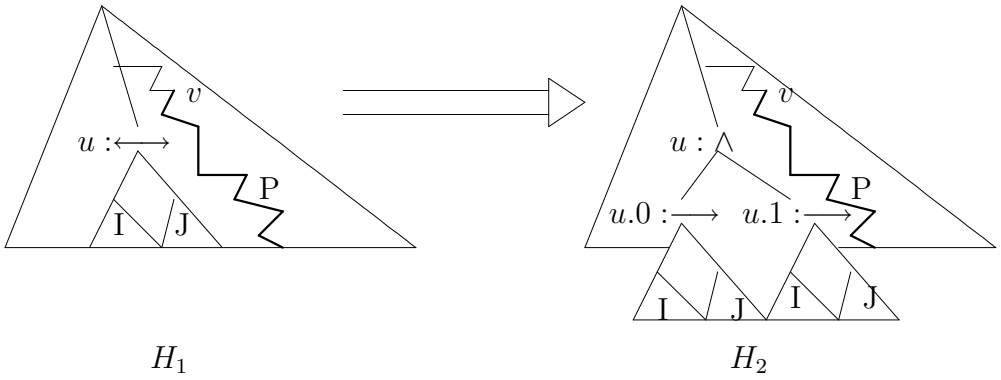


Figure 2.33: The path P does not pass through u

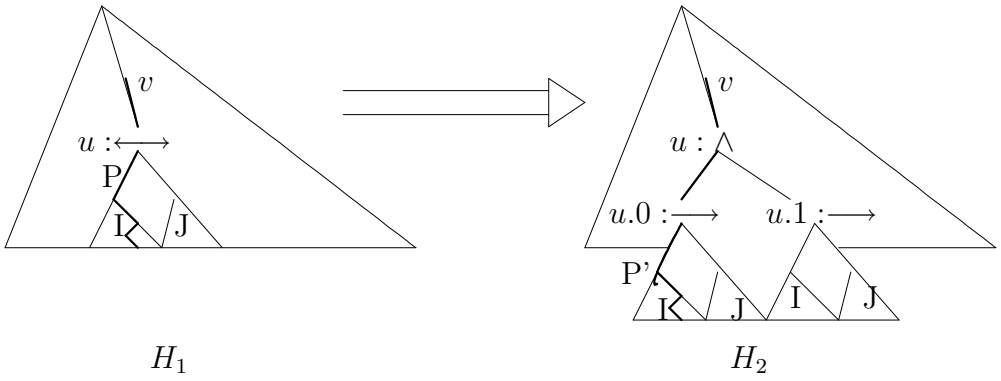


Figure 2.34: The path P does passes through u

Rule 1: if Q_1xG_1 and Q_2xG_2 are two distinct occurrences, then choose a variable y that is not in the formula and change Q_1xG_1 to $Q_1yG_1[x/y]$.

Rule 2: if there is an occurrence QxG and x is also occurs free, then replace QxG by $QyG[x/y]$, where y is a variable that is not in the formula.

In these rules, Q, Q_1, Q_2 are any of the quantifiers $\forall$ and $\exists$.

We can easily show that this step also terminates. For this, let $m[I]$ be the sum, for all variables x, of how many times x is quantified in the formula I. Each application of the rule decreases m. Since $m[F_3]$ is finite, the step terminates.

The relabelings preserve equivalences, so the output of the step is a formula $F_4 \equiv F_3$, that is rectified and contains no $\longleftrightarrow$'s or $\longrightarrow$'s. A proof similar to the one in Step 2, shows that F_4 has no redundant quantifiers.

Then $G = F_4 \equiv F$ from the transitivity of $\equiv$. **Q.E.D.**

Example 2.5.16 Let us rectify $F = \exists x \forall z(\forall y P(x,y) \longleftrightarrow \forall x \neg P(y,x))$.

We apply the algorithm given in the proof of The Rectification Lemma.

Step 1: Eliminate the redundant quantifiers.

The quantifier $\forall z$ is redundant, so we eliminate it. We get

$F_1 = \exists x(\forall y P(x,y) \longleftrightarrow \forall x \neg P(y,x))$.

Step 2: Eliminate $\longleftrightarrow$'s.

We replace $(\forall y P(x,y) \longleftrightarrow \forall x \neg P(y,x))$ by

$(\forall y P(x,y) \longrightarrow \forall x \neg P(y,x)) \wedge (\forall x \neg P(y,x) \longrightarrow \forall y P(x,y))$.

We get the formula

$F_2 = \exists x((\forall y P(x,y) \longrightarrow \forall x \neg P(y,x)) \wedge (\forall x \neg P(y,x) \longrightarrow \forall y P(x,y)))$,

equivalent to F_1, that contains neither redundant quantifiers, nor $\longleftrightarrow$'s.

Step 3: Eliminate $\longrightarrow$.

In F_2 we replace $\forall y P(x,y) \longrightarrow \forall x \neg P(y,x)$ by $\neg \forall y P(x,y) \vee \forall x \neg P(y,x)$ and $\forall x \neg P(y,x) \longrightarrow \forall y P(x,y)$ by $\neg \forall x \neg P(y,x) \vee \forall y P(x,y)$.

We get the formula

$F_3 = \exists x((\neg \forall y P(x,y) \vee \forall x \neg P(y,x)) \wedge (\neg \forall x \neg P(y,x) \vee \forall y P(x,y)))$

, equivalent to F_2, that has no redundant quantifiers, no $\longleftrightarrow$'s and no $\longrightarrow$'s.

Step 3. Rectify the formula.

The formula F_3 is not rectified because both variables are quantified more than once, and y has both free and bound occurrences. We will first take care of x, and then of y. The variable x is quantified three times, once by $\exists$ and twice by $\forall$'s. We apply rule 1 twice to relabel the first two x quantifiers.

We choose the variable z, that does not occur in F_3, replace the first quantifier $\exists x$ by $\exists z$ and substitute z for all free occurrences of x in $(\neg \forall y P(x,y) \vee \forall x \neg P(y,x)) \wedge (\neg \forall x \neg P(y,x) \vee \forall y P(x,y))$.

We get the formula

$\exists z((\neg \forall y P(z,y) \vee \forall x \neg P(y,x)) \wedge (\neg \forall x \neg P(y,x) \vee \forall y P(z,y)))$

that is equivalent to H_2.

Next, we relabel the subformula $\forall x \neg P(y,x)$. We replace x by v and we get the formula

$\exists z((\neg \forall y P(z,y) \vee \forall v \neg P(y,v)) \wedge (\neg \forall x \neg P(y,x) \vee \forall y P(z,y)))$.

The above formula is not rectified since y occurs both free and bound. So we must apply rule 2 to relabel the two subformulas $\forall y P(z,y)$. In the first subformula we relabel y by w and we get the formula

$\exists z((\neg\forall w P(z,w) \vee \forall v \neg P(y,v)) \wedge (\neg\forall x \neg P(y,x) \vee \forall y P(z,y)))$.

Now we replace the y in the subformula $\forall y P(z,y)$ by y_1. We get the formula

$G = F_4 = \exists z((\neg\forall w P(z,w) \vee \forall v \neg P(y,v)) \wedge (\neg\forall x \neg P(y,x) \vee \forall y_1 P(z,y_1)))$.

The formula G is equivalent to F, is rectified, and does not have $\longleftrightarrow$'s and $\longrightarrow$'s.

After we rectify the formula we can find a *prenex form*.

Definition 2.5.17 (prenex form) *We say that F is a prenex form if $F = Q_1x_1Q_2x_2\ldots Q_nx_nG$ where $Q_1,\ldots Q_n$ are quantifiers and G contains no quantifiers. The formula G is called the matrix of F, and written F^M.*

Examples 2.5.18 1. $F = \forall x \exists y \forall z(P(x,y) \wedge \neg Q(z))$ is a prenex form because all quantifiers are in front.

2. $F = \forall x(\exists y P(x,y) \longrightarrow Q(z))$ is not a prenex form because $\exists y$ is dominated by $\longrightarrow$. Notice that the (in front of $\exists y$ tells us that the formula is not a prenex form.

3. $F = \exists x \forall y P(x,y,z)$ is a prenex form.

The last example shows that a prenex form can have free variables.

Now we will show that every formula has a prenex form.

Lemma 2.5.19 (The Prenex Form Lemma) *Every formula H has a prenex form that does not contain redundant quantifiers, $\longleftrightarrow$'s and $\longrightarrow$'s.*

Proof: From The Rectification Lemma we know that H has an equivalent formula I that is rectified and does not contain neither $\longleftrightarrow$ nor $\longrightarrow$. The only connectives of I are $\neg$, $\wedge$, $\vee$ and the quantifiers.

We have to find a prenex form of I. For this we transform the semantic equivalences from Tables 2.27, 2.28, 2.29 into rewrites by orienting them towards the formula that has the quantifier in front. We get the simplifications shown in Figure 2.35. The first two come from Table 2.27 and the rest from Tables 2.28 and 2.29. We will show that whenever we apply a rewrite from Figure 2.35 to a rectified formula J , the result is a rectified formula K equivalent to J.

First let us show that $K \equiv J$. If we apply the first two rewrites, then $I \equiv J$ from Table 2.27. Let us assume that we apply one of the other 8. We will show that x is not free in G. Since J has no redundant quantifiers, there is a free occurrence of x in F. Now assume that x occurs in G. Then, that occurrence of x is either free or is bound somewhere else in J. In the first case, x has both free (in G) and bound (in F) occurrences. In the second case we have two quantifiers with the same argument x. In either case, J is not rectified.

So, x does not occur in G fulfilling the condition of Tables 2.28 and 2.29. So, $K \equiv J$.

It remains to show that K is rectified.

First we notice that J is rectified iff it satisfies the following properties:

1. $\neg\forall x F \Longrightarrow \exists x \neg F$
2. $\neg\exists x F \Longrightarrow \forall x \neg F$
3. $\forall x F \wedge G \Longrightarrow \forall x (F \wedge G)$
4. $\forall x F \vee G \Longrightarrow \forall x (F \vee G)$
5. $\exists x F \wedge G \Longrightarrow \exists x (F \wedge G)$
6. $\exists x F \vee G \Longrightarrow \exists x (F \vee G)$
7. $G \wedge \forall x F \Longrightarrow \forall x (G \wedge F)$
8. $G \vee \forall x F \Longrightarrow \forall x (G \vee F)$
9. $G \wedge \exists x F \Longrightarrow \exists x (G \wedge F)$
10. $G \vee \exists x F \Longrightarrow \exists x (G \vee F)$

Figure 2.35: Rewrites for computing a prenex form

(1) if x occurs free in J, there is no quantifier Qx

(2) for every x there is at most one quantifier Qx.

(3) if QxG is a subformula of J, then G has occurrences of x.

The rewrites change the position and/or the type of one quantifier while preserving its argument. For each each variable x and each rewrite in Figure 2.35, the number of quantifiers Qx on the left is equal to the number of quantifiers $Q'x$ on the right. So, the rules preserve properties (1) and (2). Moreover, the scope of Qx on the right hand side includes the scope of $Q'x$ on the left hand side. So, the rules preserve property (3).

Since J satisfies conditions (1)-(3), and the rewrites preserve them, K is rectified.

Now we apply the rewrites from Table 2.35 until all quantifiers are in front of the formula. We need to show that

(4) if the formula is not in prenex form we can apply one of the 10 rewrites, and

(5) we can find a prenex form after a finite number of applications of the rewrites.

The proof of (4) is easy. If J is not in prenex form then some occurrence of a quantifier Qx is dominated by an occurrence of one of the connectors $\neg$, $\wedge$, $\vee$ (Exercise 2.5.9).

Let t be the tree representation of J, A the address of Qx and B the address of the connective. For example, in Figure 2.36, the quantifier at address 0.1.0 is dominated by the $\neg$ at address λ.

Let us look at the ancestors of A that are labeled with one of the connectives $\neg$, $\vee$ and $\wedge$. The set is not empty because it contains B. Let C be the address of the ancestor labeled with one of these 3 connectives that is closest to A. In Figure 2.36 that ancestor is $C = 0$. Now one, or both, of the children of C is labeled with a quantifier. The subtree at address C has one of the 10 forms shown on the left-hand sides of the rewrites from Figure 2.35.

Now let us show that the rewrite rules terminate. For each rewrite $I \Longrightarrow J$, I and J have the same number of quantifiers. Moreover, the sum of the distances of the quantifiers from the root decreases with each application of $\Longrightarrow$. So, the

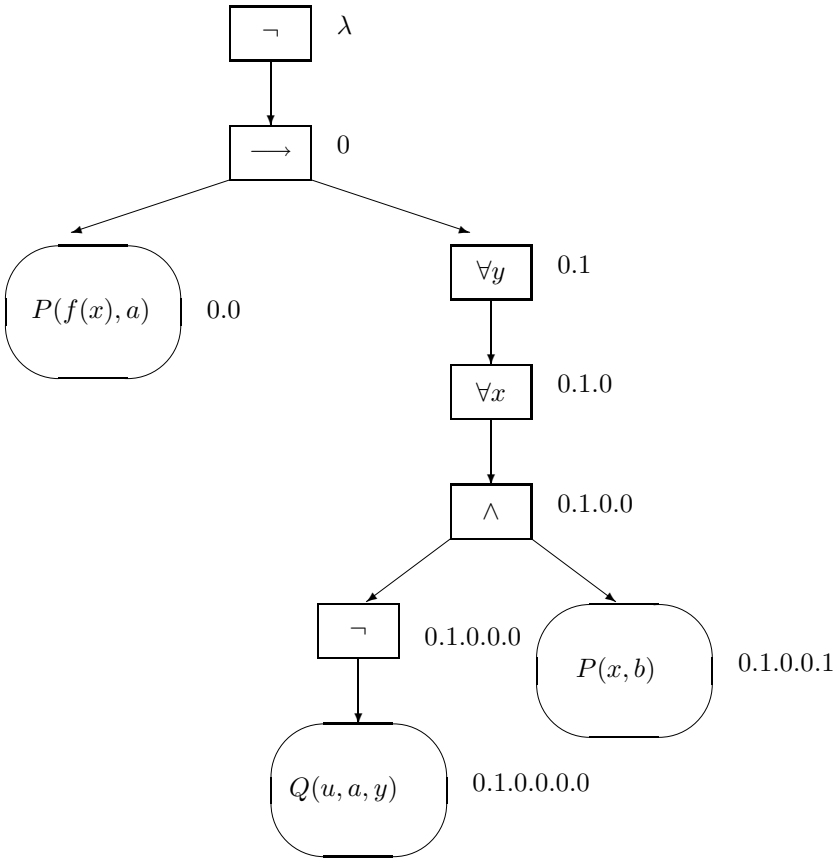


Figure 2.36: The negation dominates the quantifiers

sequence of rewrites
$I_1 \Longrightarrow I_2 \Longrightarrow I_3 \Longrightarrow \ldots \Longrightarrow I_n$
must terminate. **Q.E.D.**

The next transformations do not preserve semantic equivalences, but this is no problem since we are interested only in the preservation of the satisfiability. So, we define the *s-equivalence*, $\equiv_s$.

Definition 2.5.20 (satisfiably equivalent) *Let F and G be formulas. We say that F is s-equivalent to G, and write $F \equiv_s G$, if both are satisfiable or both are unsatisfiable.*

Proposition 2.5.21 *The relation $\equiv_s$ is an equivalence relation on the set of FOL formulas. It is weaker than the semantic equivalence $\equiv$, because $F \equiv G$ implies $F \equiv_s G$ but $F \equiv_s G$ does not always imply $F \equiv G$.*

Proof: The proof of the first part is left as exercise (Exercise 2.5.12).

Let us show the second part, that two formulas can be satisfiably equivalent without being equivalent. Let $F = P_1^0$ and $G = P_2^0$. Then $P_1^0 \equiv_s P_2^0$ because both are satisfiable. However, $P_1^0 \not\equiv P_2^0$ because any structure $\mathcal{A}$ with $\mathcal{A}^p[P_1^0] = 1$ and $\mathcal{A}^p[P_2^0] = 0$ is a model for F and a countermodel for G.

Since $F \equiv G$ implies $F \equiv_s G$ but not vice versa we conclude that $\equiv$ is finer than $\equiv_s$. **Q.E.D.**

The next step in the computation of a formula is to *close* the formula, i.e. to bind each free variable with an existential quantifier inserted in front of the formula. We call the result the *closure* of the formula.

For example, the closure of $G = \forall x P(x, f(y), z)$ is $\exists y \exists z \forall x P(x, f(y), z)$. The formula $\exists x \exists z \forall y P(x, f(y), z)$ is also a closure of G. We refer to these formulas as *the closure* because the closures are semantically equivalent (the relation $\exists x \exists y F \equiv \exists y \exists x F$ of Figure 2.26 allows us to change the order within a sequence of existential quantifiers).

Lemma 2.5.22 (The Closure Lemma) *For every formula F there a closed formula G that is satisfiably equivalent to it.*

Proof: We will prove that the s-equivalence $H \equiv_s \exists x H$ holds for all x and H.

Then the lemma follows because every time we add an existential quantifier we preserve the s-equivalence.

If x is not free in H then $H \equiv \exists x H$, because $\exists x$ is redundant. So, let us assume that H has free occurrences of x.

We need to show that
(1) if H has a model, so does $\exists x H$, and
(2) if $\exists x H$ has a model, so does H.

We prove (1) first. Assume that $\mathcal{A}[H] = 1$. Since $\mathcal{A}_{[x \leftarrow x^{\mathcal{A}}]} = \mathcal{A}$, $\mathcal{A}_{[x \leftarrow x^{\mathcal{A}}]}[H] = 1$. Then $\mathcal{A}[\exists x H] = 1$ from the interpretation of $\exists x$.

(2) Assume that $\mathcal{A}$ is a model of $\exists x H$. Then, from the interpretation of $\exists x$, there are some d in the universe of $\mathcal{A}$ such that $\mathcal{A}_{[x\leftarrow d]}[H] = 1$. So, $\mathcal{A}_{[x\leftarrow d]}$ is a model of H. **Q.E.D.**

Observation 2.5.23 Every model of G is a model of $\exists x G$, but not vice versa. For example, let $G = P(x)$ and $\mathcal{A}$ be a structure with universe $D = \{3, 4\}$. Let $P^{\mathcal{A}}[3] = 0$, $P^{\mathcal{A}}[4] = 1$, and $x^{\mathcal{A}} = 3$. Then

$\mathcal{A}[P(x)] = P^{\mathcal{A}}[x^{\mathcal{A}}] = P^{\mathcal{A}}[3] = 0$.

At the same time,

$\mathcal{A}[\exists x P(x)] = \mathcal{A}_{[x\leftarrow 3]}[P(x)] \boxed{\vee} \mathcal{A}_{[x\leftarrow 4]}[P(x)]$

$= P^{\mathcal{A}}[3] \boxed{\vee} P^{\mathcal{A}}[4]$

$= 0 \boxed{\vee} 1 = 1$.

The difference between the interpretations of G and $\exists x G$ is that G works with a single value of x while $\exists x G$ searches the universe for a value of x that would satisfy G.

So, $G \equiv_s \exists x G$ is always true, but $G \equiv \exists x G$ is sometimes false.

For the next step we need the notion of the *matrix* of a formula.

Definition 2.5.24 (matrix of a formula) *Let F be a formula. The matrix of F is the formula obtained by removing all quantifiers (and their arguments) from F. We denote the matrix of F by F^M.*

Examples 2.5.25 1. The matrix of $F = \forall x(\exists y P(x, y) \wedge Q(x))$ is $F^M = (P(x, y) \wedge Q(x))$.

2. The matrix of $F = \forall x \exists y(P(x, y) \longrightarrow Q(x))$ is $F^M = (P(x, y) \longrightarrow Q(x))$.

The next step in the computation of the normal form is to *Skolemize* the formula. Skolemization means to replace the existentially quantified variables by functions having as arguments the universally quantified variables that are in front of the existential variables. The input to this algorithm is a closed prenex formula, that is, a formula $F = Q_1 x_1 \ldots Q_n x_n F^M$ with no free variables. We start with the leftmost existential quantifier. Assume that $Q_i x_i = \exists x_i$ is the leftmost existential quantifier. This means that $Q_1 = Q_2 = \ldots = Q_{i-1} = \forall$. We create a new function symbol, f, that does not occur in F^M, and replace every occurrence of x_i in F^M by $f(x_1, \ldots, x_{i-1})$. If $i = 1$, i.e. the leftmost quantifier of F is existential, we replace x by a constant.

We get a formula $G = Q_1 x_1 \ldots Q_{i-1} x_{i-1} Q_{i+1} x_{i+1} \ldots Q_n x_n F^M[x_i / f(x_1, \ldots, x_{i-1})]$. We repeat the elimination until the formula has no more existential quantifiers.

Examples 2.5.26 1. Let us Skolemize the formula $H = \exists x \forall y \exists z P(a, x, f(y), z)$.

The leftmost existential quantifier is $\exists x$, so we eliminate it first. There are no universal quantifiers in front of it, so we replace x by a constant that is not in the formula. Since a occurs in H, we replace x by b. We get the formula

$H_1 = \forall y \exists z P(a, x, f(y), z)[x/b] = \forall y \exists z P(a, b, f(y), z)$.

Next we remove the leftmost existential quantifier from H_1. The quantifier $\exists z$ is preceded by $\forall y$, so z will be replace by a function of argument y. Since f already occurs in the formula, we replace z by $g(y)$. We get the formula

$H_2 = \forall y P(a, b, f(y), z)[z/g(y)] = \forall y P(a, b, f(y), g(y))$.

The formula H_2 does not have any existential quantifiers, so the algorithm stops.

2. Let us Skolemize the formula $I = \forall x \exists y \forall z \exists u (P(x, f(y), u) \wedge Q(u, z, a))$.

We eliminate the leftmost existential quantifier first. $\exists y$ is preceded by $\forall x$, so we replace y by $g(x)$. We cannot use f because f is already in I^M. We get

$I_1 = \forall x \forall z \exists u (P(x, f(y), u) \wedge Q(u, z, a))[y/g(x)]$

$= \forall x \forall z \exists u (P(x, f(g(x)), u) \wedge Q(u, z, a))$.

Next we eliminate the leftmost existential quantifier of I_1. We replace u by the function $h(x, z)$ because the quantifiers $\forall x$ and $\forall z$ are in front of $\exists u$. We get

$I_2 = \forall x \forall z (P(x, f(g(x)), u) \wedge Q(u, z, a))[u \leftarrow h(x, z)]$

$= \forall x \forall z (P(x, f(g(x)), h(x, z)) \wedge Q(h(x, z), z, a)$.

The last formula has no more existential quantifiers, so the algorithm stops.

Lemma 2.5.27 (The Skolemization Lemma) *Let $F = \forall y_1 \forall y_2 \ldots \forall y_n \exists x G$ be a rectified formula and let f be a function symbol that is not in F. Then $F \equiv_s H = \forall y_1 \forall y_2 \ldots \forall y_n G[x/f(y_1, \ldots, y_n)]$.*

Proof: We need to show that

(1) if F has a model, so does H, and

(2) if H has a model, so does F.

We prove (1) first. Let $\mathcal{A}$ be a structure with universe D. Since $\mathcal{A}$ is a model of $F = \forall y_1 \forall y_2 \ldots \forall y_n \exists x G$,

for all $d_1, \ldots, d_n \in D$ there are some $d \in D$ such that

(3) $\mathcal{A}_{[y_1 \leftarrow d_1] \ldots [y_n \leftarrow d_n][x \leftarrow d]}[G] = 1$.

Now let $\mathcal{B}$ be the structure that is identical to $\mathcal{A}$ except for the interpretation of f.

We define $f^{\mathcal{B}}[d_1, \ldots, d_n]$ to be one of the d's that satisfy relation (3). So,

for all $d_1, \ldots, d_n \in D$,

(4) $\mathcal{A}_{[y_1 \leftarrow d_1] \ldots [y_n \leftarrow d_n][x \leftarrow f^{\mathcal{B}}[d_1, \ldots, d_n]]}[G] = 1$.

Since f does not occur in G, $\mathcal{A}_{[y_1 \leftarrow d_1] \ldots [y_n \leftarrow d_n][x \leftarrow f^{\mathcal{B}}[d_1, \ldots, d_n]]}$ and $\mathcal{B}_{[y_1 \leftarrow d_1] \ldots [y_n \leftarrow d_n][x \leftarrow f^{\mathcal{B}}[d_1, \ldots, d_n]]}$ agree on G, so

(5) $\mathcal{A}_{[y_1 \leftarrow d_1] \ldots [y_n \leftarrow d_n][x \leftarrow f^{\mathcal{B}}[d_1, \ldots, d_n]]}[G] = \mathcal{B}_{[y_1 \leftarrow d_1] \ldots [y_n \leftarrow d_n][x \leftarrow f^{\mathcal{B}}[d_1, \ldots, d_n]]}[G]$.

Now,

$\mathcal{B}[\forall y_1 \forall y_2 \ldots \forall y_n G[x/f(y_1, \ldots, y_n)]] = 1$

iff for all $d_1, \ldots, d_n \in D$, $\mathcal{B}_{[\forall y_1 \leftarrow d_1] \ldots [y_n \leftarrow d_n][x \leftarrow f^{\mathcal{B}}[d_1, \ldots, d_n]]}[G[x/f(y_1, \ldots, y_n)]] =$ 1 from the interpretation of $\forall$

iff for all $d_1, \ldots, d_n \in D$, $\mathcal{B}_{[\forall y_1 \leftarrow d_1] \ldots [y_n \leftarrow d_n][x \leftarrow \mathcal{B}_{[\forall y_1 \leftarrow d_1] \ldots [y_n \leftarrow d_n]}[f(y_1, \ldots, f_n)]]}[G] =$ 1 from The Translation Lemma

iff for all $d_1, \ldots, d_n \in D$, $\mathcal{B}_{[\forall y_1 \leftarrow d_1] \ldots [y_n \leftarrow d_n][x \leftarrow f^{\mathcal{B}}[d_1, \ldots, d_n]]}[G] = 1$ from the evaluation of $\mathcal{B}_{[\forall y_1 \leftarrow d_1] \ldots [y_n \leftarrow d_n]}[f(y_1, \ldots, f_n)]$

iff $d_1, \ldots, d_n \in D$,

$\mathcal{B}_{[\forall y_1 \leftarrow d_1]\ldots[y_n \leftarrow d_n][x \leftarrow f^{\mathcal{B}}[d_1,\ldots,d_n]]}[G] = 1$ from (5)
The last statement is exactly (4) which is true.

Now we prove (2). We will show that every model of H is also a model of F.

Let $\mathcal{A}$ be a model of H. Then
$\mathcal{A}[H] = 1$
iff for all $d_1, \ldots, d_n \in D$, $\mathcal{A}_{[y_1 \leftarrow d_1]\ldots[y_n \leftarrow d_n]}[G[x/f(y_1, \ldots, y_n)]] = 1$
by the interpretation of the universal quantifiers
iff for all $d_1, \ldots, d_n \in D$, $\mathcal{A}_{[y_1 \leftarrow d_1]\ldots[y_n \leftarrow d_n][x \leftarrow \mathcal{A}[f(y_1,\ldots y_n)]]}[G] = 1$ by The Translation Lemma
The last relation yields
for all $d_1, \ldots, d_n \in D$ there are some $d \in D$ such that $\mathcal{A}_{[y_1 \leftarrow d_1]\ldots[y_n \leftarrow d_n][x \leftarrow d]}[G] = 1$
iff for all $d_1, \ldots, d_n \in D$ $\mathcal{A}_{[y_1 \leftarrow d_1]\ldots[y_n \leftarrow d_n]}[\exists x G] = 1$ interpretation of $\exists x$
iff $\mathcal{A}[\forall y_1 \ldots \forall y_n \exists x G] = 1$ interpretation of the $\forall$'s
iff $\mathcal{A}[F] = 1$.

We can apply The Translation Lemma to G because $\forall y_1 \ldots \forall y_n \exists x G$ is rectified. So, G has no occurrences $Q_1 y_1, \ldots, Q_n y_n$. Then $f(y_1, \ldots, y_n)$ is free for x in G. **Q.E.D.**

Observation 2.5.28 The proof of The Skolemization Lemma tels us that every model of H is a model of F (part 2 of the proof), but not every model of F is a model of H.

The notions of *literal* and *clause* are the same as in propositional logic.

Definition 2.5.29 (literal, positive literal, negative literal, clause) *1. A literal is either an atomic formula or the negation of an atomic formula. If the literal contains the negation it is called a negative literal; otherwise it is called a positive literal.*

2. A clause is a (possibly empty) disjunction of literals.

The Skolemization Lemma allows us to get rid of the existential quantifiers. The next step is to put the matrix of the formula in clause form. The matrix does not contain any quantifiers, so we can apply Algorithm 1.7.12 from Section 1.7. The result is a *Skolem normal form.*

Definition 2.5.30 (Skolem normal form) *1. Skolem normal forms or Skolem forms are closed formulas of the type $\forall y_1 \forall y_2 \ldots \forall y_n \bigwedge_{i=1}^{m} C_i$, where C_i, $1 \leq i \leq m$, are clauses.*

2. A Skolem normal form of F is a Skolem form that is s-equivalent to F.

Now let us combine all the little transformations presented so far into an algorithm that computes Skolem forms.

Algorithm 2.5.31 (The Skolem Form Algorithm) Step 1. Eliminate redundant quantifiers.

We apply the reduction

if x is not free in G, replace QxG by G

until there are no more redundant quantifiers. We get a formula F_1 equivalent to $F_0 = F$.

Step 2. Eliminate the $\longleftrightarrow$'s from F_1.

We replace every occurrence of $G \longleftrightarrow H$ by $(G \longrightarrow H) \wedge (H \longrightarrow G)$. We can minimize the number of reductions by applying this simplification to the formulas $G \longleftrightarrow H$ that do not have $\longleftrightarrow$'s in neither G nor H. This way we eliminate all $\longrightarrow$'s in $n[\longleftrightarrow, F_1]$ steps. We get a formula F_2 equivalent to F_1.

Step 3. Eliminate the $\longrightarrow$'s from F_2.

We replace every subformula $G \longrightarrow H$ in F_2 by $\neg G \vee H$. We get a formula F_3 equivalent to F_2 whose only connectives are $\neg$, $\wedge$, $\vee$ and the quantifiers.

Step 4. Rectify the formula F_3.

We use The Relabeling lemma to rename the variables that are quantified more than once and the ones that occur both free and bound. We get a rectified formula F_4 equivalent to F_3 whose only connectives are $\neg$, $\wedge$, $\vee$ and the quantifiers.

Step 5. Find the prenex form of F_4.

We pull out all quantifiers of F_4 using the rewrites from Figure 2.35. We obtain a formula F_5 equivalent to F_4, that is in prenex form and contains only the connectives $\neg$, $\wedge$, $\vee$ and the quantifiers.

Step 6. Close F_5.

We bound all free variables of F_5 with existential quantifiers inserted in front of the formula. We get a formula F_6, that is s-equivalent to F_5. Then F_6 is a closed prenex form whose only connectives are $\neg$, $\wedge$, $\vee$ and the quantifiers.

Step 7. Skolemize F_6.

We replace every existential quantifier by a new function having as arguments the universal quantifiers that precede it. We obtain a prenex form F_7 whose only connectives are $\neg$, $\wedge$, $\vee$ and the universal quantifiers.

Step 8. Put the matrix of F_7 in clause form.

Here we treat the matrix of F_7 as a formula in propositional logic and apply Algorithm 1.7.12 from Section 1.7. Since the matrix does not contain neither $\longleftrightarrow$'s nor $\longrightarrow$'s, we can skip the first two steps of the algoritm. So, we start by pushing the negation inward.

Step 8.1. Push the negation inward until it disappears or it rests in front of an atomic formula.

We apply the rewrites

1. $\neg\neg G \Longrightarrow G$
2. $\neg(G \wedge H) \Longrightarrow (\neg G \vee \neg H)$
3. $\neg(G \vee H) \Longrightarrow (\neg G \wedge \neg H)$

until all remaining negations rest on atomic formulas.

Step 8.2. We distribute $\vee$ over the $\wedge$.

We apply the rewrites

1. $(G \wedge H) \vee I \Longrightarrow (G \vee I) \wedge (H \vee I)$

2. $I \vee (G \wedge H) \longrightarrow (I \vee G) \wedge (I \vee H)$

until the formula is a conjunction of clauses.

Step 8.3. We simplify the matrix by applying the tautology, unsatisfiablity and absorption laws from Table 1.47.

The clause form is equivalent to the original matrix, so the output of step 8, F_8, is equivalent to F_7.

Theorem 2.5.32 (The Skolem Form Theorem) *Every formula is satisfiably equivalent to a set universally quantified clauses.*

Proof: We will show that Algorithm 2.5.31 correct. The Rectification Lemma proves that the first 4 steps terminates and F_4 is equivalent to F, is rectified and $n[\longleftrightarrow, F_4] = n[\longrightarrow, F_4] = 0$. The Prenex Form Lemma proves the termination and the correctness of the fifth step. The termination and the correctness of Steps 6, and 7 are established by The Closure Lemma and The Skolemization Lemma, in this order. The correctness of the last step, that the matrix is equivalent to a set of clauses, is assured by The CNF Theorem from Section 1.7. **Q.E.D.**

Now let us apply the algorithm to compute CNF's for two formulas. For clarity, we will replace the outermost parentheses of the formulas by braces and brackets.

Example 2.5.33 Let us compute a Skolem form for $F = \neg\{[\forall x(P(x) \longrightarrow Q(x)) \wedge \forall z \forall x(P(x) \longrightarrow (Q(x) \longrightarrow R(x)))] \longrightarrow \forall x(P(x) \longrightarrow R(x))\}$.

Step 1. Eliminate redundant quantifiers.

The quantifier $\forall z$ is redundant in $\forall z \forall x(P(x) \longrightarrow (Q(x) \longrightarrow R(x)))$ because z is not free in $\forall x(P(x) \longrightarrow (Q(x) \longrightarrow R(x)))$. So we remove it and get

$F_1 = \neg\{[\forall x(P(x) \longrightarrow Q(x)) \wedge \forall x(P(x) \longrightarrow (Q(x) \longrightarrow R(x)))] \longrightarrow \forall x(P(x) \longrightarrow R(x))\}$.

Step 2. Eliminate $\longleftrightarrow$'s from F_1.

We do not have $\longleftrightarrow$'s in F_1, so $F_2 = F_1$.

Step 3. Eliminate $\longrightarrow$'s from F_2.

We replace the $\longrightarrow$'s from left to right by using the rewrites $G \longrightarrow H \Longrightarrow \neg G \vee H$.

$F_2 = \neg\{[\forall x(P(x) \longrightarrow Q(x)) \wedge \forall x(P(x) \longrightarrow (Q(x) \longrightarrow R(x)))] \longrightarrow \forall x(P(x) \longrightarrow R(x))\}$

$\equiv \neg\{[\forall x(\neg P(x) \vee Q(x)) \wedge \forall x(P(x) \longrightarrow (Q(x) \longrightarrow R(x)))] \longrightarrow \forall x(P(x) \longrightarrow R(x))\}$

$\equiv \neg\{[\forall x(\neg P(x) \vee Q(x)) \wedge \forall x(\neg P(x) \vee (Q(x) \longrightarrow R(x)))] \longrightarrow \forall x(P(x) \longrightarrow R(x))\}$

$\equiv \neg\{[\forall x(\neg P(x) \vee Q(x)) \wedge \forall x(\neg P(x) \vee (\neg Q(x) \vee R(x)))] \longrightarrow \forall x(P(x) \longrightarrow R(x))\}$

$\equiv \neg\{\neg[\forall x(\neg P(x) \vee Q(x)) \wedge \forall x(\neg P(x) \vee (\neg Q(x) \vee R(x)))] \vee \forall x(P(x) \longrightarrow R(x))\}$

$\equiv \neg\{\neg[\forall x(\neg P(x) \vee Q(x)) \wedge \forall x(\neg P(x) \vee (\neg Q(x) \vee R(x)))] \vee \forall x(\neg P(x) \vee R(x))\} = F_3$

Step 4. Rectify F_3.

The variable x is quantified 3 times, so we replace the x's in $\forall x(\neg P(x)\vee Q(x))$ by y and the x's in $\forall x(\neg P(x) \vee (\neg Q(x) \vee R(x)))$ by z. We get the formula

$F_4 = \neg\{\neg[\forall y(\neg P(y) \vee Q(y)) \wedge \forall z(\neg P(z) \vee (\neg Q(z) \vee R(z)))] \vee \forall x(\neg P(x) \vee R(x))\}$.

Step 5. We find the prenex form of F_4.

We underline the quantifiers that we pull out.

$F_4 = \neg\{\neg[\forall y(\neg P(y)\vee Q(y))\wedge\forall z(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\vee\underline{\forall x}(\neg P(x)\vee R(x))\}$

$\equiv \neg\underline{\forall x}\{\neg[\forall y(\neg P(y)\vee Q(y))\wedge\forall z(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\vee(\neg P(x)\vee R(x))\}$

$\equiv \exists x\neg\{\neg[\underline{\forall y}(\neg P(y)\vee Q(y))\wedge\forall z(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\vee(\neg P(x)\vee R(x))\}$

$\equiv \exists x\neg\{\neg\underline{\forall y}[(\neg P(y)\vee Q(y))\wedge\forall z(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\vee(\neg P(x)\vee R(x))\}$

$\equiv \exists x\neg\{\underline{\exists y}\neg[(\neg P(y)\vee Q(y))\wedge\forall z(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\vee(\neg P(x)\vee R(x))\}$

$\equiv \exists x\underline{\exists y}\neg\{\neg[(\neg P(y)\vee Q(y))\wedge\forall z(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\vee(\neg P(x)\vee R(x))\}$

$\equiv \exists x\forall y\neg\{\neg[(\neg P(y)\vee Q(y))\wedge\underline{\forall z}(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\vee(\neg P(x)\vee R(x))\}$

$\equiv \exists x\forall y\neg\{\neg\underline{\forall z}[(\neg P(y)\vee Q(y))\wedge(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\vee(\neg P(x)\vee R(x))\}$

$\equiv \exists x\forall y\neg\{\underline{\exists z}\neg[(\neg P(y)\vee Q(y))\wedge(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\vee(\neg P(x)\vee R(x))\}$

$\equiv \exists x\forall y\neg\underline{\exists z}\{\neg[(\neg P(y)\vee Q(y))\wedge(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\vee(\neg P(x)\vee R(x))\}$

$\equiv \exists x\forall y\forall z\neg\{\neg[(\neg P(y)\vee Q(y))\wedge(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\vee(\neg P(x)\vee R(x))\} = F_5$

Step 6. Close F_5.

F_5 has no free variables, so, the closure of F_5 is $F_6 = F_5$.

Step 7. Skolemize F_6.

We replace x by a.

$F_7 = \forall y\forall z\neg\{\neg[(\neg P(y)\vee Q(y))\wedge(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\vee(\neg P(a)\vee R(a))\}$

Step 8. Put the matrix of F_7 in conjunctive normal form.

We first push the negation inward. At each step we underline the negations that are rewritten.

$F_7 = \forall y\forall z\underline{\neg}\{\neg[(\neg P(y)\vee Q(y))\wedge(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\vee(\neg P(a)\vee R(a))\}$

$\equiv \forall y\forall z\{\underline{\neg\neg}[(\neg P(y)\vee Q(y))\wedge(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\wedge\underline{\neg}(\neg P(a)\vee R(a))\}$

$\equiv \forall y\forall z\{[(\neg P(y)\vee Q(y))\wedge(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\wedge(\underline{\neg\neg}P(a)\wedge\neg R(a))\}$

$\equiv \forall y\forall z\{[(\neg P(y)\vee Q(y))\wedge(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\wedge(P(a)\wedge\neg R(a))\}$.

At this point we are done since $F_8 = \forall y\forall z\{[(\neg P(y)\vee Q(y))\wedge(\neg P(z)\vee(\neg Q(z)\vee R(z)))]\wedge(P(a)\wedge\neg R(a))\}$ is a CNF. The set of clauses of F_8 is $\{\{\neg P(y), Q(y)\}, \{\neg P(z), \neg Q(z), R(z)\}, \{P(a)\}, \{\neg R(a)\}\}$.

Example 2.5.34 Let us find a Skolem form for $F = \neg\{[\forall x\forall y(P(x,y) \longrightarrow P(y,x))\wedge\forall x\forall y\forall z((P(x,y)\wedge P(y,z)) \longrightarrow P(x,z))] \longrightarrow (\forall x\exists y P(x,y) \longleftrightarrow \forall x P(x,x))\}$.

Step 1. Eliminate redundant quantifiers.

We do not have redundant quantifiers here, so $F_1 = F$.

Step 2. Eliminate $\longleftrightarrow$'s.

We get

$F_2 = \neg\{[\forall x\forall y(P(x,y) \longrightarrow P(y,x))\wedge\forall x\forall y\forall z((P(x,y)\wedge P(y,z)) \longrightarrow P(x,z))] \longrightarrow [(\forall x\exists y P(x,y) \longrightarrow \forall x P(x,x)) \wedge (\forall x P(x,x) \longrightarrow \forall x\exists y P(x,y))]\}$.

Step 3. Eliminate $\longrightarrow$'s.

We eliminate them from left to right

$F_2 \equiv \neg\{[\forall x\forall y(\neg P(x,y)\vee P(y,x))\wedge\forall x\forall y\forall z((P(x,y)\wedge P(y,z)) \longrightarrow P(x,z))] \longrightarrow [(\forall x\exists y P(x,y) \longrightarrow \forall x P(x,x)) \wedge (\forall x P(x,x) \longrightarrow \forall x\exists y P(x,y))]\}$

$\equiv \neg\{[\forall x\forall y(\neg P(x,y)\vee P(y,x))\wedge\forall x\forall y\forall z(\neg(P(x,y)\wedge P(y,z))\vee P(x,z))]\longrightarrow [(\forall x\exists yP(x,y)\longrightarrow\forall xP(x,x))\wedge(\forall xP(x,x)\longrightarrow\forall x\exists yP(x,y))]\}$

$\equiv \neg\{\neg[\forall x\forall y(\neg P(x,y)\vee P(y,x))\wedge\forall x\forall y\forall z(\neg(P(x,y)\wedge P(y,z))\vee P(x,z))]\vee [(\forall x\exists yP(x,y)\longrightarrow\forall xP(x,x))\wedge(\forall xP(x,x)\longrightarrow\forall x\exists yP(x,y))]\}$

$\equiv \neg\{\neg[\forall x\forall y(\neg P(x,y)\vee P(y,x))\wedge\forall x\forall y\forall z(\neg(P(x,y)\wedge P(y,z))\vee P(x,z))]\vee [(\neg\forall x\exists yP(x,y)\vee\forall xP(x,x))\wedge(\forall xP(x,x)\longrightarrow\forall x\exists yP(x,y))]\}$

$\equiv \neg\{\neg[\forall x\forall y(\neg P(x,y)\vee P(y,x))\wedge\forall x\forall y\forall z(\neg(P(x,y)\wedge P(y,z))\vee P(x,z))]\vee [(\neg\forall x\exists yP(x,y)\vee\forall xP(x,x))\wedge(\neg\forall xP(x,x)\vee\forall x\exists yP(x,y))]\} = F_3$

Step 4. Rectify F_3.

We relabel x and y in $\forall x\forall y(\neg P(x,y)\vee P(y,x))$ by u respectively v. We get

$F_3\equiv\neg\{\neg[\forall u\forall v(\neg P(u,v)\vee P(v,u))\wedge\forall x\forall y\forall z(\neg(P(x,y)\wedge P(y,z))\vee P(x,z))]\vee [(\neg\forall x\exists yP(x,y)\vee\forall xP(x,x))\wedge(\neg\forall xP(x,x)\vee\forall x\exists yP(x,y))]\}$.

Next, we relabel the x and the y in $\forall x\forall y\forall z(\neg(P(x,y)\wedge P(y,z))\vee P(x,z))$ by w, respectively y_1. We get

$F_3\equiv\neg\{\neg[\forall u\forall v(\neg P(u,v)\vee P(v,u))\wedge\forall w\forall y_1\forall z(\neg(P(w,y_1)\wedge P(y_1,z))\vee P(w,z))]\vee[(\neg\forall x\exists yP(x,y)\vee\forall xP(x,x))\wedge(\neg\forall xP(x,x)\vee\forall x\exists yP(x,y))]\}$.

Futher on, we relabel x and y in the first $\forall x\exists yP(x,y)$ by x_1, respectively y_2.

$F_3\equiv\neg\{\neg[\forall u\forall v(\neg P(u,v)\vee P(v,u))\wedge\forall w\forall y_1\forall z(\neg(P(w,y_1)\wedge P(y_1,z))\vee P(w,z))]\vee[(\neg\forall x_1\exists y_2P(x_1,y_2)\vee\forall xP(x,x))\wedge(\neg\forall xP(x,x)\vee\forall x\exists yP(x,y))]\}$

We relabel with x_2 the x's in the first occurrence of the subformula $\forall xP(x,x)$ and get

$F_3\equiv\neg\{\neg[\forall u\forall v(\neg P(u,v)\vee P(v,u))\wedge\forall w\forall y_1\forall z(\neg(P(w,y_1)\wedge P(y_1,z))\vee P(w,z))]\vee[(\neg\forall x_1\exists y_2P(x_1,y_2)\vee\forall x_2P(x_2,x_2))\wedge(\neg\forall xP(x,x)\vee\forall x\exists yP(x,y))]\}$.

Finally we change the x in the subformula $\forall xP(x,x)$ to x_3 and get

$F_4=\neg\{\neg[\forall u\forall v(\neg P(u,v)\vee P(v,u))\wedge\forall w\forall y_1\forall z(\neg(P(w,y_1)\wedge P(y_1,z))\vee P(w,z))]\vee[(\neg\forall x_1\exists y_2P(x_1,y_2)\vee\forall x_2P(x_2,x_2))\wedge(\neg\forall x_3P(x_3,x_3)\vee\forall x\exists yP(x,y))]\}$.

Step 5. Find a prenex formula for F_4.

We advance the quantifiers u, v, w, y_1 and z passed the $\wedge$.

$F_4\equiv\neg\{\neg\forall u\forall v\forall w\forall y_1\forall z[(\neg P(u,v)\vee P(v,u))\wedge(\neg(P(w,y_1)\wedge P(y_1,z))\vee P(w,z))]\vee[(\neg\forall x_1\exists y_2P(x_1,y_2)\vee\forall x_2P(x_2,x_2))\wedge(\neg\forall x_3P(x_3,x_3)\vee\forall x\exists yP(x,y))]\}$

Next we advance the quantifiers x_1, y_2, x_3 passed the negations in front of them.

$F_4\equiv\neg\{\neg\forall u\forall v\forall w\forall y_1\forall z[(\neg P(u,v)\vee P(v,u))\wedge(\neg(P(w,y_1)\wedge P(y_1,z))\vee P(w,z))]\vee[(\exists x_1\forall y_2\neg P(x_1,y_2)\vee\forall x_2P(x_2,x_2))\wedge(\exists x_3\neg P(x_3,x_3)\vee\forall x\exists yP(x,y))]\}$

We pass the quantifiers x_2, x_1, y_2 and x, y, x_3 over the $\vee$'s that dominate them.

$F_4\equiv\neg\{\neg\forall u\forall v\forall w\forall y_1\forall z[(\neg P(u,v)\vee P(v,u))\wedge(\neg(P(w,y_1)\wedge P(y_1,z))\vee P(w,z))]\vee[\forall x_2\exists x_1\forall y_2(\neg P(x_1,y_2)\vee P(x_2,x_2))\wedge\forall x\exists y\exists x_3(\neg P(x_3,x_3)\vee P(x,y))]\}$

Now we pass the quantifiers u, v, w, y_1 z over the first negation.

$F_4\equiv\neg\{\exists u\exists v\exists w\exists y_1\exists z\neg[(\neg P(u,v)\vee P(v,u))\wedge(\neg(P(w,y_1)\wedge P(y_1,z))\vee P(w,z))]\vee[\forall x_2\exists x_1\forall y_2(\neg P(x_1,y_2)\vee P(x_2,x_2))\wedge\forall x\exists y\exists x_3(\neg P(x_3,x_3)\vee P(x,y))]\}$

We advance x, x_2, x_1, y_2, y, x_3 passed the $\wedge$.

$F_4\equiv\neg\{\exists u\exists v\exists w\exists y_1\exists z\neg[(\neg P(u,v)\vee P(v,u))\wedge(\neg(P(w,y_1)\wedge P(y_1,z))\vee P(w,z))]\vee\forall x\forall x_2\exists x_1\forall y_2\exists y\exists x_3[(\neg P(x_1,y_2)\vee P(x_2,x_2))\wedge(\neg P(x_3,x_3)\vee P(x,y))]\}$

We advance x, x_2, x_1, y_2, y, x_3, u, v, w, y_1, z, in this order, passed the $\vee$.

$F_4 \equiv \neg\forall x\forall x_2\exists x_1\forall y_2\exists y\exists x_3\exists u\exists v\exists w\exists y_1\exists z\{\neg[(\neg P(u,v)\vee P(v,u))\wedge(\neg(P(w,y_1)\wedge P(y_1,z))\vee P(w,z))]\vee[(\neg P(x_1,y_2)\vee P(x_2,x_2))\wedge(\neg P(x_3,x_3)\vee P(x,y))]\}$

Finally, we move all quantifiers ahead of $\neg$ and get the prenex form

$F_5 = \exists x\exists x_2\forall x_1\exists y_2\forall y\forall x_3\forall u\forall v\forall w\forall y_1\forall z\neg\{\neg[(\neg P(u,v)\vee P(v,u))\wedge(\neg(P(w,y_1)\wedge P(y_1,z))\vee P(w,z))]\vee[(\neg P(x_1,y_2)\vee P(x_2,x_2))\wedge(\neg P(x_3,x_3)\vee P(x,y))]\}$.

Step 6. Close the formula.

F_5 is closed, so $F_6 = F_5$.

Step 7. Skolemize F_6.

We replace x by a, x_2 by b, and y_2 by $f(x_1)$.

$F_7 = \forall x_1\forall y\forall x_3\forall u\forall v\forall w\forall y_1\forall z\neg\{\neg[(\neg P(u,v)\vee P(v,u))\wedge(\neg(P(w,y_1)\wedge P(y_1,z))\vee P(w,z))]\vee[(\neg P(x_1,f(x_1))\vee P(b,b))\wedge(\neg P(x_3,x_3)\vee P(a,y))]\}$

Step 8. Put the matrix of F_7 in conjunctive normal form.

First we push the negation inward. We will do a leftmost computation, i.e. we will push the leftmost negation inside until it rests on an atomic formula or it is eliminated.

$F_7 \equiv \forall x_1\forall y\forall x_3\forall u\forall v\forall w\forall y_1\forall z\{\neg\neg[(\neg P(u,v)\vee P(v,u))\wedge(\neg(P(w,y_1)\wedge P(y_1,z))\vee P(w,z))]\wedge\neg[(\neg P(x_1,f(x_1))\vee P(b,b))\wedge(\neg P(x_3,x_3)\vee P(a,y))]\}$

$\equiv \forall x_1\forall y\forall x_3\forall u\forall v\forall w\forall y_1\forall z\{[(\neg P(u,v)\vee P(v,u))\wedge((\neg P(w,y_1)\vee\neg P(y_1,z))\vee P(w,z))]\wedge[\neg(\neg P(x_1,f(x_1))\vee P(b,b))\vee\neg(\neg P(x_3,x_3)\vee P(a,y))]\}$

$\equiv \forall x_1\forall y\forall x_3\forall u\forall v\forall w\forall y_1\forall z\{[(\neg P(u,v)\vee P(v,u))\wedge((\neg P(w,y_1)\vee\neg P(y_1,z))\vee P(w,z))]\wedge[(\neg\neg P(x_1,f(x_1))\wedge\neg P(b,b))\vee(\neg\neg P(x_3,x_3)\wedge\neg P(a,y))]\}$

$\equiv \forall x_1\forall y\forall x_3\forall u\forall v\forall w\forall y_1\forall z\{[(\neg P(u,v)\vee P(v,u))\wedge((\neg P(w,y_1)\vee\neg P(y_1,z))\vee P(w,z))]\wedge[(P(x_1,f(x_1))\wedge\neg P(b,b))\vee(P(x_3,x_3)\wedge\neg P(a,y))]\}$

Now we distribute $\vee$ over $\wedge$. We underline the $\vee$ and the $\wedge$ that are involved in the rewrite.

$\equiv \forall x_1\forall y\forall x_3\forall u\forall v\forall w\forall y_1\forall z\{[(\neg P(u,v)\vee P(v,u))\wedge((\neg P(w,y_1)\vee\neg P(y_1,z))\vee P(w,z))]\wedge[(P(x_1,f(x_1))\underline{\wedge}\neg P(b,b))\underline{\vee}(P(x_3,x_3)\wedge\neg P(a,y))]\}$

$\equiv \forall x_1\forall y\forall x_3\forall u\forall v\forall w\forall y_1\forall z\{[(\neg P(u,v)\vee P(v,u))\wedge((\neg P(w,y_1)\vee\neg P(y_1,z))\vee P(w,z))]\wedge[(P(x_1,f(x_1))\underline{\vee}(P(x_3,x_3)\underline{\wedge}\neg P(a,y)))\wedge(\neg P(b,b)\underline{\vee}(P(x_3,x_3)\underline{\wedge}\neg P(a,y)))]\}$

$\equiv \forall x_1\forall y\forall x_3\forall u\forall v\forall w\forall y_1\forall z\{[(\neg P(u,v)\vee P(v,u))\wedge((\neg P(w,y_1)\vee\neg P(y_1,z))\vee P(w,z))]\wedge[((P(x_1,f(x_1))\vee P(x_3,x_3))\wedge(P(x_1,f(x_1))\vee\neg P(a,y)))\wedge((\neg P(b,b)\vee P(x_3,x_3))\wedge(\neg P(b,b)\vee\neg P(a,y)))]\} = F_8$.

The last formula is a Skolem normal form. Its clause set is $\{\{\neg P(u,v),P(v,u)\},\{\neg P(w,y_1),\neg P(y_1,z),P(w,z)\},\{P(x_1,f(x_1)),P(x_3,x_3)\},\{P(x_1,f(x_1)),\neg P(a,y)\},\{\neg P(b,b),P(x_3,x_3)\},\{\neg P(b,b),\neg P(a,y)\}\}$.

Exercises

Exercise 2.5.1 *Prove cases 4, 5, 6, and 8 of The Translation Lemma.*

Exercise 2.5.2 *Prove the second part of the redundant quantifier elimination proposition:*

if x is not free in F then $\exists xF \equiv F$.

Exercise 2.5.3 *Show that if t is free for x in F then $\forall xF \models F[x/t]$.*

Exercise 2.5.4 *Give a example that shows the above statement is not true when t is not free for x in F.*

Exercise 2.5.5 *Prove the consequence* $\forall x(F \vee G), \neg F[x/a] \models \exists x G$.
Hint: Use the Translation Lemma.

Exercise 2.5.6 *Prove the consequence* $\forall x(F \longrightarrow G), \exists x(F \vee G) \models \exists x G$.

Exercise 2.5.7 *Let* u *and* v *two variables that do not occur in* $\forall x \forall y F$. *Prove that* $\forall x \forall y F \models \forall u \forall v F[x/f(u,v)][y/g(u,v)]$

Exercise 2.5.8 *In the proof of The Rectification Lemma we eliminated the connective* $\longleftrightarrow$ *before we performed the rectification. Why?*

Exercise 2.5.9 *Prove that* F *is a prenex form iff no instance of an operator* $\neg$, $\wedge$, $\vee$, *dominates an instance of a quantifier.*

Exercise 2.5.10 *Prove or disprove:*
F *is a prenex form iff its matrix is a subformula of* F.

Exercise 2.5.11 *Prove that the property of being a prenex form is hereditary, i.e. if* F *is a prenex form, every subformula of* F *is a prenex form.*

Exercise 2.5.12 *Prove that* $\equiv_s$ *is an equivalence relation on the set of first order formulas.*

Exercise 2.5.13 *Show that* $F \equiv G$ *implies* $F \equiv_s G$.

Exercise 2.5.14 *Which one of this consequences holds,* $G \models \exists x G$, *or* $\exists x G \models G$? *Prove or disprove.*

Exercise 2.5.15 *Prove by structural induction that the matrix of formula is a formula.*

Exercise 2.5.16 *Prove that every set of FOL sentences is satisfiably equivalent to a set of clauses. Hint: Add an infinite set of function formulas* g_i^n *that allow us to find the Skolem form of each formula.*

Exercise 2.5.17 *Apply the Skolem normal form algorithm to the formula* $F = \neg(\exists x \forall y P(x,y) \longrightarrow \forall y \exists z \exists x P(x,y))$.

Exercise 2.5.18 *Apply the Skolem normal form algorithm to the formula* $F = \neg((\forall x(P(x) \longrightarrow Q(x)) \wedge \exists x(P(x) \longrightarrow \neg Q(x))) \longrightarrow \exists x \neg P(x))$.

Exercise 2.5.19 *The following algorithm generates better Skolem forms.*
Step 1: Eliminate redundant quantifiers.
Step 2: Eliminate $\longleftrightarrow$*'s.*
Step 3: Eliminate $\longrightarrow$*'s.*
Step 4: Rectify the formula.
Step 5: Close the formula.
Step 6: Push the negation inside until it disappears or rests on atomic formulas.

Step 7: Push the quantifiers inside as far as possible using the rewrite below:
If x is not free in G then
1. $Qx(F \wedge G) \Longrightarrow QxF \wedge G$
2. $Qx(G \wedge F) \Longrightarrow G \wedge QxF$
3. $Qx(F \vee G) \Longrightarrow QxF \vee G$
4. $Qx(G \vee F) \Longrightarrow G \vee QxF$
If need be we can also use the rewrites 5 and 6 to pass a quantifier over a $\vee$ or a $\wedge$.
5. $\forall x \forall y G \Longrightarrow \forall y \forall x G$
6. $\exists x \exists y G \Longrightarrow \exists y \exists x G$
Step 8: Skolemize the formula by replacing every existentially quantified variable by a new function of the universally quantified variables that dominate it.
Step 9: Compute the prenex form.
Step 10. Find the CNF of the matrix by applying the distributivity.
In many cases this algorithm generates Skolem functions of smaller arity than our algorithm, as shown below.
The prenex forms of $F = \forall x \exists y P(x, y) \wedge \forall u \exists v \neg P(u, v)$ *are equivalent to one of the following 3 formulas:*
$\forall x \exists y \forall u \exists v (P(x, y) \wedge \neg P(u, v))$,
$\forall u \exists v \forall x \exists y (P(x, y) \wedge \neg P(u, v))$, *and*
$\forall x \forall u \exists u \exists v (P(x, y) \wedge \neg P(u, v))$.
These prenex forms generate the Skolem forms
$F_1 = \forall x \forall u (P(x, f(x)) \wedge \neg P(u, g(x, u))$,
$F_2 = \forall u \forall x (P(x, f(u, x)) \wedge \neg P(u, g(u))$, *and*
$F_3 = \forall x \forall u (P(x, f(x, u)) \wedge \neg P(u, g(x, u))$.
The above algorithm generates $G = \forall x \forall u (P(x, f(x)) \wedge \neg P(u, g(u)))$, *so the arity of at least one of the functions is reduced.*
Prove that the new procedure is correct.

2.6 First Order Theories

In this section we look at the relation between formulas and models. For example, are there formulas that have only finite models? Are there formulas that have only infinite models? What properties encountered in mathematics can be specified in first order logic? Also, what relation is there between structures?

This section will provide examples of formulas that have only finite or only infinite models, and will present several algebraic structures as formulas in first order logic.

Definition 2.6.1 (finite and infinite models) *Let $\mathcal{A}$ be a structure and let D be its universe. We say that $\mathcal{A}$ is finite if $\mathcal{D}$ is finite and we say that $\mathcal{A}$ is infinite when D is infinite.*

We say that a formula F has a finite model if there is a finite structure $\mathcal{A}$ such that $\mathcal{A}[F] = 1$. We say that F has an infinite model if there is an infinite structure $\mathcal{A}$ such that $\mathcal{A}[F] = 1$.

Example 2.6.2 Every model $\mathcal{A}$ of $F = \forall x(E(x,a) \vee E(x,b) \vee E(x,c))$ must contain interpretations for the constants a, b, and c. At the same time F requires that every element of the universe must be equal to $a^{\mathcal{A}}$, $b^{\mathcal{A}}$, or $c^{\mathcal{A}}$. So, the universe of $\mathcal{A}$ is $\{a^{\mathcal{A}}, b^{\mathcal{A}}, c^{\mathcal{C}}\}$, a set with at most 3 elements.

The universe has 3 elements if $a^{\mathcal{A}}$, $b^{\mathcal{A}}$, $c^{\mathcal{A}}$ are distinct, 2 elements when 2 of them are equal, and 1 element when $a^{\mathcal{A}} = b^{\mathcal{A}} = c^{\mathcal{A}}$.

At the same time, every set with 1, 2, or 3 elements is the universe of a model of F. The set $\{1\}$ is the universe of a model $\mathcal{B}$ with $a^{\mathcal{B}} = b^{\mathcal{B}} = c^{\mathcal{C}} = 1$. The set $\{1,2\}$ is the universe of a model $\mathcal{C}$ with $a^{\mathcal{C}} = b^{\mathcal{C}} = 1$ and $c^{\mathcal{C}} = 2$. The set $\{1,2,3\}$ is the universe of a model $\mathcal{E}$ with $a^{\mathcal{E}} = 1$, $b^{\mathcal{E}} = 2$, $c^{\mathcal{E}} = 3$.

Example 2.6.3 The models of $G = \forall x(E(x,a) \vee E(x,b) \vee E(x,c)) \wedge \neg E(a,b) \wedge \neg E(a,c) \wedge \neg E(b,c)$ have exactly 3 elements. Let's see why. The formula G is a conjunction of F, $\neg E(a,b)$, $\neg E(a,c)$, and $\neg E(a,c)$. Every model of G must be a model of F, so the models of G cannot have more than 3 elements. However, the conditions $\neg E(a,b)$, $\neg E(a,c)$, $\neg E(b,c)$ tell us that for every model $\mathcal{A}$ of G, $a^{\mathcal{A}} \neq b^{\mathcal{A}}$, $a^{\mathcal{A}} \neq c^{\mathcal{A}}$, and $b^{\mathcal{A}} \neq c^{\mathcal{A}}$. So, the elements $a^{\mathcal{A}}$, $b^{\mathcal{A}}$, $c^{\mathcal{A}}$ of the universe of A must be distinct, i.e. $\mathcal{A}$ has exactly 3 elements.

At the same time all sets with 3 elements are the universe of some model of G. For example $\{1,2,3\}$ is the universe of the model of G that interprets a as 1, b as 2 and c as 3.

Example 2.6.4 The formula $H = \forall x \neg P(x,x) \wedge \forall x P(x, f(x)) \wedge \forall x \forall y \forall z((P(x,y) \wedge P(y,z)) \longrightarrow P(x,z))$ has only infinite models. First we will prove that H has an infinite model and then we will show that it has no finite model.

Let $\mathcal{A}$ be a structure with universe $N = \{0,1,2,...\}$, $P^{\mathcal{A}}$ the ordering $<$, and $f^{\mathcal{A}}$ the successor function $f^{\mathcal{A}}[n] = n+1$. The structure $\mathcal{A}$ satisfies the first conjunct of H, $\forall x \neg P(x,x)$, because for all natural numbers n, $n \not< n$. It also models the second conjunct of H, because for all natural numbers n, $n < f^{\mathcal{A}}[n] = n+1$. Finally, $\mathcal{A}$ satisfies the third conjunct of H because for all natural numbers m, n, and p, $n < m$ and $m < p$ imply $n < p$. So, $\mathcal{A}$ models H since it satisfies all 3 conjuncts.

Now, let us assume that H has a finite model $\mathcal{B}$. Let d_0 be an element of the universe of $\mathcal{B}$. Let's look at the set

$d_1 = f^{\mathcal{B}}[d_0], d_2 = f^{\mathcal{B}}[d_1], d_3 = f^{\mathcal{B}}[d_2], \ldots$.

This set is finite, since it is a subset of the universe of $\mathcal{B}$. So, there must be some repetitions in the above sequence. Let d_j be the first element that is repeated. Then, there is some $0 \leq i < j$, such that

(1) $d_i = d_j$.

From the second conjunct of H, $\forall x P(x, f(x))$, we have

(2) $P^{\mathcal{B}}[d_i, d_{i+1}] = P^{\mathcal{B}}[d_{i+1}, d_{i+2}] = \ldots = P^{\mathcal{B}}[d_{j-1}, d_j] = 1$

Now we apply the 3rd conjunct of H, $\forall x \forall y \forall z((P(x,y) \wedge P(y,z)) \longrightarrow P(x,z))$ to this sequence.

If we assign d_i to x, d_{i+1} to y, and d_{i+2} to z, the 3-rd conjunct tells us that $P^{\mathcal{B}}[d_i, d_{i+1}] \boxed{\wedge} P^{\mathcal{B}}[d_{i+1}, d_{i+2}] \boxed{\longrightarrow} P^{\mathcal{B}}[d_i, d_{i+2}]$ is true. But $P^{\mathcal{B}}[d_i, d_{i+1}]$, and $P^{\mathcal{B}}[d_{i+1}, d_{i+2}]$, listed in the sequence (2) above, are true. So,

(3) $P^{\mathcal{B}}[d_i, d_{i+2}] = 1$

Now we apply the third conjunct again, this time assigning d_i to x, d_{i+2} to y and d_{i+3} to z. We get that $[P^{\mathcal{B}}[d_i, d_{i+2}] \boxed{\wedge} P^{\mathcal{B}}[d_{i+2}, d_{i+3}]] \boxed{\longrightarrow} P^{\mathcal{B}}[d_i, d_{i+3}] = 1$. Since $P^{\mathcal{B}}[d_i, d_{i+2}] = 1$ from (3) and $P^{\mathcal{B}}[d_{i+2}, d_{i+3}] = 1$ from (1) we get

(4) $P^{\mathcal{B}}[d_i, d_{i+3}] = 1$.

Next we apply the third conjunct assigning d_i to x, d_{i+3} to y and d_{i+4} to z and get

(5) $P^{\mathcal{B}}[d_i, d_{i+4}]$.

By repeatingly applying the 3rd conjunct we get that

(6) $P^{\mathcal{B}}[d_i, d_j] = 1$.

Now we apply the first conjunct, $\forall x \neg P(x, x)$. If we assign d_i to x we get

(7) $\boxed{\neg} P^{\mathcal{B}}[d_i, d_i] = 1$

or

(8) $P^{\mathcal{B}}[d_i, d_i] = 0$.

Since $d_j = d_i$ by (1), (8) contradicts (6). So, H has no infinite models.

The first two examples displayed formulas that have only finite models, and both formulas contained the equality predicate E. Is this by chance, or there is something more to it? We will show that our intuition is correct, because every satisfiable formula that does not contain E has infinite models. Moreover, we will develop tools for proving many other properties of the models.

Definition 2.6.5 (structure homomorphism) *Let $\mathcal{A}$ and $\mathcal{B}$ be two structures with universes $D_{\mathcal{A}}$, respectively $D_{\mathcal{B}}$. A function $\phi : D_{\mathcal{A}} \longrightarrow D_{\mathcal{B}}$ is a homomorphism, if it satisfies the 5 conditions below.*

1. For all constants a, $a^{\mathcal{B}} = \phi[a^{\mathcal{A}}]$.

2. For all variables x, $x^{\mathcal{B}} = \phi[x^{\mathcal{A}}]$.

3. For all functions symbols f of arity $q > 0$ and all $q-$tuples $< e_1, e_2, ..., e_q >$ of $D_{\mathcal{A}}$,

$f^{\mathcal{B}}[\phi[e_1], \ldots, \phi[e_q]] = \phi[f^{\mathcal{A}}[e_1, \ldots, e_q]]$.

4. For all predicate symbols P of arity 0, $P^{\mathcal{B}} = P^{\mathcal{A}}$.

5. For all predicate symbols P of arity $q > 0$, and all $q-$tuples $< e_1, e_2, ..., e_q >$ of $D_{\mathcal{A}}$,

$P^{\mathcal{B}}[\phi[e_1], \ldots, \phi[e_q]] = P^{\mathcal{A}}[e_1, \ldots, e_q]$.

The homomorphism is called monomorphism when ϕ is one-to-one, epimorphism when ϕ is onto and isomorphism when ϕ is a bijection.

Examples 2.6.6 1. Let $\mathcal{A}$ be any structure with domain $D_{\mathcal{A}} = \{a_0, \ldots, a_{n-1}\}$ and a_n be an element that is not in $D_{\mathcal{A}}$. Let $D_{\mathcal{B}} = \{a_0, \ldots, a_{n-1}, a_n\}$. We will extend $\mathcal{A}$ to a structure $\mathcal{B}$ with domain $D_{\mathcal{B}}$ by making a_n mimic a_{n-1}.

So let us define $\phi : D_{\mathcal{B}} \longrightarrow D_{\mathcal{A}}$ with the formula below.

$$\phi[a_i] = \begin{cases} a_i & \text{if } i \neq n \\ a_{n-1} & \text{if } i = n \end{cases}$$

The structure $\mathcal{B}$ is specified by the following 5 rules:

1. For all constants a, $a^{\mathcal{B}} = a^{\mathcal{A}}$.

2. For all variables x, $x^{\mathcal{B}} = x^{\mathcal{A}}$.

3. For all functions symbols f of arity $q > 0$ and all $q-tuples < d_1, d_2, \ldots, d_q >$ of $D_{\mathcal{B}}$,

$f^{\mathcal{B}}[d_1, \ldots, d_q] = f^{\mathcal{A}}[\phi[d_1], \ldots, \phi[d_q]]$.

4. For all predicate symbols P of arity 0, $P^{\mathcal{B}} = P^{\mathcal{A}}$.

5. For all predicate symbols P of arity $q > 0$, and all $q-tuples < d_1, d_2, \ldots, d_q >$ of $D_{\mathcal{B}}$,

$P^{\mathcal{B}}[d_1, \ldots, d_q] = P^{\mathcal{A}}[\phi[d_1], \ldots, \phi[d_q]]$.

We notice that for all $d \in D_{\mathcal{A}}$,

(6) $\phi[d] = d$.

We claim that ϕ is an epimorphism from $\mathcal{B}$ to $\mathcal{A}$. Let us show that ϕ satisfies the 5 conditions of the homomorphism.

Condition 1: for all constants a,

$\phi[a^{\mathcal{B}}] = \phi[a^{\mathcal{A}}]$ by 1, $a^{\mathcal{B}} = a^{\mathcal{A}}$

$= a^{\mathcal{A}}$ by (6)

Condition 2: for all variables x,

$\phi[x^{\mathcal{B}}] = \phi[x^{\mathcal{A}}]$ by 2, $a^{\mathcal{B}} = a^{\mathcal{A}}$

$= x^{\mathcal{A}}$ by (6)

Condition 3: let f be a function of arity $q > 0$ and $e_1, \ldots, e_q \in D\mathcal{B}$.

$\phi[f^{\mathcal{B}}[e_1, \ldots, e_q]]$

$= \phi[f^{\mathcal{A}}[\phi[d_1], \ldots, \phi[d_q]]]$ by 3

$= f^{\mathcal{A}}[\phi[d_1], \ldots, \phi[d_q]]$ by (6), since $f^{\mathcal{A}}[\phi[d_1], \ldots, \phi[d_q]]] \in D_{\mathcal{A}}$.

Condition 4: let P be a predicate constant. Then $P_{\mathcal{B}} = P^{\mathcal{A}}$ by rule 4.

Condition 5: let P be a predicate symbol of arity $q > 0$ and $d_1, \ldots, d_q$ be a q-tuple from $D_{\mathcal{B}}$. Then

$P^{\mathcal{A}}[\phi[d_1], \ldots, \phi[d_q]]$

$= P^{\mathcal{B}}[d_1, \ldots, d_q]$ by rule 5.

So, the last homomorphism condition is also satisfied, and ϕ is a homomorphism. Since ϕ is onto, it is an epimorphism.

2. Let $\mathcal{A}$ be a structure with universe $D_{\mathcal{A}} = \{a_0, \ldots, a_{n-1}\}$. Let $S = \{a_n, a_{n+1}, \ldots, a_{n+m}, \ldots\}$ be a countable set disjoint from $D_{\mathcal{A}}$. Let $D_{\mathcal{C}} = D_{\mathcal{A}} \cup S = \{a_0, \ldots, a_{n-1}, a_n, \ldots\}$. The relation below defines a function $\psi : D_{\mathcal{B}} \longrightarrow D_{\mathcal{A}}$.

$$\psi[a_i] = \begin{cases} a_i & \text{if } i < n \\ a_{n-1} & \text{if } i \geq n \end{cases}$$

The rules for the structure $\mathcal{C}$ are same as for $\mathcal{B}$, except that ϕ is replaced by ψ. Then we show, just like we did for ϕ, that ψ is an epimorphism from $\mathcal{C}$ to $\mathcal{A}$.

The next two lemmas state two properties of the homomorphism that will be used in the proof of The Homomorphism Theorem.

Lemma 2.6.7 *If ϕ is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$, and $d \in universe(\mathcal{A})$, then ϕ is also a homomorphism from $\mathcal{A}_{[y \leftarrow d]}$ to $\mathcal{B}_{[y \leftarrow \phi[d]]}$.*

Proof: Let ϕ is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$ and d an element of the domain of $\mathcal{A}$. Then ϕ satisfies the 5 conditions of Definition 2.6.5. The structures $\mathcal{A}$ and $\mathcal{A}_{[y \leftarrow d]}$ differ only on the interpretation of y and the same is true for the pair $\mathcal{B}$, $\mathcal{A}_{[y \leftarrow \phi[d]]}$. So, conditions 1, 3, 4, 5 are also true when we replace $\mathcal{A}$ by $\mathcal{A}_{[y \leftarrow d]}$ and $\mathcal{B}$ by $\mathcal{B}_{[y \leftarrow \phi[d]]}$. It remains to verify 2. If $x \neq y$, then

$\mathcal{B}_{[y \leftarrow \phi[d]]}[x] = x^{\mathcal{B}}$
$= \phi[x^{\mathcal{A}}]$ by 2
$= \phi[\mathcal{A}_{[y \leftarrow d]}[x]]$ $\mathcal{A}$ and $\mathcal{A}_{[y \leftarrow d]}$ agree on x
If $x = y$ then
$\mathcal{B}_{[y \leftarrow \phi[d]]}[x] = \phi[d]$
$= \phi[\mathcal{A}_{[y \leftarrow d]}[x]]$.
So, condition 2 also holds. **Q.E.D.**

The next lemma tells us that the homomorphism $\phi : \mathcal{A} \longrightarrow \mathcal{B}$ maps the $\mathcal{A}$ interpretation of a term into the $\mathcal{B}$ interpretation of the term.

Lemma 2.6.8 (The Homomorphism Lemma for Terms) *Let ϕ be a homomorphism from $\mathcal{A}$ to $\mathcal{B}$ and t a term. Then $\phi[\mathcal{A}[t]] = \mathcal{B}[t]$.*

Proof: The proof is by structural induction on t. If t is a constant or a variable, the equality holds by the homomorphism conditions 1 and 2.

If $t = f(t_1, \ldots, t_n)$ then
$\mathcal{B}[t] = \mathcal{B}[f(t_1, \ldots, t_n)]$
$= f^{\mathcal{B}}[\mathcal{B}[t_1], \ldots, \mathcal{B}[t_n]]$ by the interpretation of $f(t_1, \ldots, t_n)$
$= f^{\mathcal{B}}[\phi[\mathcal{A}[t_1]], \ldots, \phi[\mathcal{A}[t_n]]]$ by the induction hypothesis
$= \phi[f^{\mathcal{A}}[\mathcal{A}[t_1], \ldots, \mathcal{A}[t_n]]]$ by condition (3) of Definition 2.6.5
$= \phi[\mathcal{A}[f(t_1, \ldots, t_n)]]$ by the interpretation of $f(t_1, \ldots, t_n)$
$= \phi[\mathcal{A}[t]]$.
Q.E.D.

Theorem 2.6.9 (The Homomorphism Theorem) *Let ϕ be a homomorphism from $\mathcal{A}$ to $\mathcal{B}$. 1. If F is a formula without E and without quantifiers, then $\mathcal{A}$ is a model of F iff $\mathcal{B}$ is a model of F.*

2. If ϕ is a monomorphism and F is a formula without quantifiers, then $\mathcal{A}$ is a model of F iff $\mathcal{B}$ is a model of F.

3. If ϕ is a epimorphism and F is a formula without equality, then $\mathcal{A}$ is a model of F iff $\mathcal{B}$ is a model of F.

Proof: Part 1. The proof is by structural induction on F.

Case 1: F is an atomic formula. Then $F = P$ or $F = P(t_1, \ldots, t_n)$. If $F = P$, then $P^{\mathcal{B}} = P^{\mathcal{A}}$ tells us that $\mathcal{A}$ is a model for P iff $\mathcal{B}$ is a model for P.

Let us now assume that $F = P(t_1, \ldots, t_n)$. Then
$\mathcal{B}[P(t_1, \ldots, t_n)]$
$= P^{\mathcal{B}}[\mathcal{B}[t_1], \ldots, \mathcal{B}[t_n]]$ by the interpretation of $P(t_1, \ldots, t_n)$
$= P^{\mathcal{B}}[\phi[\mathcal{A}[t_1]], \ldots, \mathcal{A}[t_1]]]$ by The Homomorphism Lemma for Terms
$= P^{\mathcal{A}}[\mathcal{A}[t_1], \ldots, \mathcal{A}[t_n]]$ by the homomorphism condition (3)
$= \mathcal{A}[P(t_1, \ldots, t_n)]$ by the interpretation of $P(t_1, \ldots, t_n)$.
Again, $\mathcal{A}$ is a model for P iff $\mathcal{B}$ is a model for P.

Case 2: $F = \neg G$. Then
$\mathcal{B}[\neg G] = \boxed{\neg}[\mathcal{B}[G]]$ by the interpretation of $\neg$
$= \boxed{\neg}[\mathcal{A}[G]]$ by the induction hypothesis
$= \mathcal{B}[\neg G]$ by the interpretation of $\neg$
Cases 3, 4, 5, 6: $F = G \vee H, F = G \wedge H, G \longrightarrow H, F = G \longleftrightarrow H$ are similar to Case 2.
Part 2: ϕ is a monomorphism. The proof is the same except that we need to show that $\models_{\mathcal{A}} E(t_1, t_2)$ iff $\models_{\mathcal{B}} E(t_1, t_2)$.
$\mathcal{A}[E(t_1, t_2)] = 1$
iff $\mathcal{A}[t_1] = \mathcal{A}[t_2]$ by the interpretation of E
iff $\phi[\mathcal{A}[t_1]] = \phi[\mathcal{A}[t_2]]$ because ϕ is one-to-one
iff $\mathcal{B}[t_1] = \mathcal{B}[t_2]$ by The Homomorphism Lemma for Terms
iff $\mathcal{B}[E(t_1, t_2)] = 1$ by the interpretation of E
Part 3: ϕ is an epimorphism. The proof is identical to Part 1, except that we need to add proofs for Cases 7 and 8.
Case 7: $F = \forall x G$.
$\mathcal{B}[\forall x G] = 0$
iff for some $e \in universe(\mathcal{B})$, $\mathcal{B}_{[x \leftarrow e]}[G] = 0$ by the interpretation of $\forall x$
iff for some $d \in universe(\mathcal{A})$,$\mathcal{B}_{[x \leftarrow \phi[d]]}[G] = 0$ since ϕ is onto, $e = \phi[d]$ for some $d \in universe(\mathcal{A})$
iff for some $d \in universe(\mathcal{A})$, $\mathcal{A}_{[x \leftarrow d]}[G] = 0$ by the IH applied the homomorphism $\phi : \mathcal{A}_{[x \leftarrow d]} \longrightarrow \mathcal{B}_{[x \leftarrow \phi[d]]}$
iff $\mathcal{A}[\forall x G] = 0$ by the interpretation of $\forall x$
Case 8: $F = \exists x G$ is similar to Case 7. It is left as exercise. **Q.E.D.**

Corollary 2.6.10 *Let F be a formula without the equality predicate.*
1. If F has a model with n elements, then it has a model with $n+1$ elements.
2. If F has a finite model, it has an infinite model.

Proof: 1. Let $\mathcal{A}$ be a model of F with n-elements, $a_0, \ldots, a_{n-1}$. Let a_n be an element distinct from $a_0, \ldots, a_{n-1}$. We recall that the function ϕ from Examples 2.6.6 is an epimorphism. By The Homomorphism Theorem, Part 3, $\mathcal{A}$ satisfies F iff $\mathcal{B}$ satisfies F. So, $\mathcal{B}$ is a model of F. Since the universe of $\mathcal{C}$ is has $n + 1$, F has a model with $n + 1$ elements.
2. We use the function ψ from Examples 2.6.6. Using the same reasoning from Part 1 we get that $\mathcal{C}$ is a model of F. **Q.E.D.**

In many cases we do not need all functions and predicates of the full predicate calculus. The pure predicate calculus does uses neither functions nor equality and group theory, presented below, employs only the binary function $+$, the unary function $-$, the constant 0 and the equality predicate. Set theory also uses only two predicates, $\in$ and E, and no function symbols of arity greater than 0. However, all of them use a countable subset of variables. We can extend the concepts and the theorems of the full predicate calculus to the languages that use a restricted alphabet. So let L be the alphabet that contains all variables of the full calculus, a possibly empty subset of function symbols, and a non-empty subset of predicate letters.

We can modify the definitions from Section 2.1 to define L-terms, L-atomic formulas, L-formulas. We just make sure that all symbols of these strings are in L.

Definition 2.6.11 (L-terms) *1. The individual variables are L-terms.*

2. If a is a constant of L, then a is an L-term.

3. If t_1, t_2,..., t_n are L-terms and $f \in L$ is function symbol of arity $n > 0$, then $f(t_1, t_2, \ldots, t_n)$ is a L-term.

Definition 2.6.12 (L-atomic formulas) *1. The predicate constants P are atomic formulas.*

2. If t_1 and t_2 are terms and $E \in L$, then $E(t_1, t_2)$ is an atomic formula. The formula $E(t_1, t_2)$ is read t_1 is identical to t_2.

3. If P is a predicate symbol of arity $n \geq 1$ and t_1, t_2,..., t_n are terms, then $P(t_1, t_2, \ldots, t_n)$ is an atomic formula. We read it P of t_1, t_2,..., t_n.

Definition 2.6.13 (L-Formulas) *1. The L atomic formulas are formulas.*

2. If F is an L-formula then $\neg F$ is an L-formula, called the negation of F.

3. If F and G are L-formulas then $(F \vee G)$ is also an L-formula, called F or G.

4. If F and G are L-formulas then $(F \wedge G)$ is also an L-formula, called F and G.

5. If F and G are L-formulas then $(F \longrightarrow G)$ is also an L-formula, called if F then G.

6. If F and G are L-formulas then $(F \longleftrightarrow G)$ is also an L-formula, called F if and only if G.

7. If F is an L-formula and x is a variable, then $\forall xF$ is also an L-formula, called for all xF. We say that x is the argument of the operator $\forall x$, and that x is universally quantified.

8. If F is an L-formula and x is a variable then $\exists xF$ is an L-formula, called for some xF or there exists x such that F. Again we say that x is the argument of operator $\exists x$ and that x is existentially quantified.

Example 2.6.14 We recall that the language of group theory contains only the binary function $+$, the unary function $-$, the constant 0, and the equality E. For readability purposes we will use the infix notation and eliminate the parentheses whenever possible. We will also use the more familiar symbol $\doteq$ instead of the cumbersome E. We will not use $=$ because this symbol belongs to the meta-language. So, we will write $x + y$ instead of $+(x, y)$ and $u \doteq v$ instead of $E(u, v)$.

Then 0, x, $-y$, $x + y$, $-(x + y)$ are terms of the theory. The strings $x \doteq 0$, $-(x+y) \doteq -y+-x$ are atomic formulas, and $\forall x \exists y(x+y) \doteq (y+x)$ is a formula.

The definitions of subterm and subformula are the same as for full predicate calculus. A subterm of the L-term t is a substring of t that is an L-term. A subformula of the L-formula F is a substring of F that is an L-formula.

We can also prove the structural induction theorem for the L-terms and for L-formulas.

Theorem 2.6.15 (The L-Structural Induction Theorem for Terms) *A proposition $P[t]$ is true for all L-terms t if and only if*

1. $P[t]$ is true when t is a variable.

2. $P[t]$ is true when t is an L-constant.

3. If f is an L-function of arity $n > 0$, $t_1, \ldots, t_n$ are L-terms, and $P[t_1], \ldots, P[t_n]$ are true then $P[f(t_1, \ldots, t_n)]$ is true.

Theorem 2.6.16 (The L-Structural Induction Theorem) *A proposition $P[t]$ is true for all L-formulas F if and only if the following 8 conditions are satisfied.*

1. $P[F]$ is true when F is an atomic formula.

2. If $P[F]$ is true then $P[\neg F]$ is true.

3. If $P[F]$ and $P[G]$ are true then $P[F \vee G]$ is true.

4. If $P[F]$ and $P[G]$ are true then $P[F \wedge G]$ is true.

5. If $P[F]$ and $P[G]$ are true then $P[F \longrightarrow G]$ is true.

6. If $P[F]$ and $P[G]$ are true then $P[F \longleftrightarrow G]$ is true.

7. If $P[F]$ is true then $P[\forall x F]$ is true.

8. If $P[F]$ is true then $P[\exists x F]$ is true.

The proofs of these theorems are identical to the ones from Section 2.2. The L-structures assign meaning to the symbols of L. They are defined the same way as the structures from Section 2.3.

Definition 2.6.17 (L-structure) *An L- structure is a 4-tuple $\mathcal{A} = < |\mathcal{A}|, \mathcal{A}^f, \mathcal{A}^p, \mathcal{A}^v >$ that satisfies the following conditions:*

1. $|\mathcal{A}|$ is a non-empty set called the universe of the structure,

2. $\mathcal{A}^f$ assigns to each function symbol $g \in L$ a function $\mathcal{A}^f[g]$;

2.1. if g has arity 0, $\mathcal{A}^f[g] \in |\mathcal{A}|$,

2.2. if g has arity $n > 0$, then $\mathcal{A}^f[g] : |\mathcal{A}|^n \longrightarrow |\mathcal{A}|$,

3. $\mathcal{A}^p$ assigns to each predicate symbol $P \in L$ a function $\mathcal{A}^p[P]$ with codomain $\{0, 1\}$;

3.1 if P has arity 0, $\mathcal{A}^p[P] \in \{0, 1\}$,

3.2. if P has arity $n > 0$, then $\mathcal{A}^p[P] : |\mathcal{A}|^n \longrightarrow \{0, 1\}$, and

4. $\mathcal{A}^v$ assigns to each variable x an element in $|\mathcal{A}|$.

Observations 2.6.18 1. If L does not have any function symbols, then $\mathcal{A}^f$ is undefined; if it does not have any predicate symbols, $\mathcal{A}^p$ is undefined. For example, the group theory presented in Example 2.6.14 has no predicate symbol besides E.

2. If we are only interested in the truth value of the closed formulas, we can ignore $\mathcal{A}^v$.

The Unique Readability Theorem holds for the terms and the formulas of L, so we can define the semantics of the L-terms and L-formulas according to Definitions 2.3.3, 2.3.5, and 2.3.8. Since L-formulas are formulas in the full predicate calculus, The Agreement Theorem is also valid for L.

We define L-homomorphisms by modifying Definition 2.6.5.

(1) $\forall x \forall y \forall z (x + y) + z \doteq (x + y)z$
(2) $\forall x (0 + x) \doteq x$
(3) $\forall x (x + 0) \doteq x$
(4) $\forall x (-x + x) \doteq 0$
(5) $\forall x (x + -x) \doteq 0$

Figure 2.37: The group axioms

Definition 2.6.19 (L-structure homomorphism) *Let $\mathcal{A}$ and $\mathcal{B}$ be two structures with universes $D_{\mathcal{A}}$, respectively $D_{\mathcal{A}}$. A function $\phi : D_{\mathcal{A}} \longrightarrow D_{\mathcal{B}}$ is a homomorphism, if it satisfies the 5 conditions below.*

1. For all constants $a \in L$, $a^{\mathcal{B}} = \phi[a^{\mathcal{A}}]$.

2. For all variables x, $x^{\mathcal{B}} = \phi[x^{\mathcal{A}}]$.

3. For all functions symbols $f \in L$ of arity $q > 0$ and all $q-$tuples $< e_1, e_2, ..., e_q >$ of $D_{\mathcal{A}}$,

$f^{\mathcal{B}}[\phi[e_1], \ldots, \phi[e_q]] = \phi[f^{\mathcal{A}}[e_1, \ldots, e_q]]$.

4. For all predicate symbols $P \in L$ of arity 0, $P^{\mathcal{B}} = P^{\mathcal{A}}$.

5. For all predicate symbols $P \in L$ of arity $q > 0$, and all $q-$tuples $< e_1, e_2, \ldots, e_q >$ of $D_{\mathcal{A}}$,

$P^{\mathcal{B}}[\phi[e_1], \ldots, \phi[e_q]] = P^{\mathcal{A}}[e_1, \ldots, e_q]$.

The Homomorphism Theorem is also valid for the L-formulas. The proof is identical to the preceding one, except that we prefix the words function, predicate, constant, term and formula by the letter L.

Now, let us talk about first order theories. A first order theory, or theory for short, consists of an alphabet L as defined above together with a set of formulas called *axioms*. A *model of a theory* is an L-structure that is a model for all the axioms.

Example 2.6.20 Let us define the theory of groups. We already know the alphabet L; it consist of variables, the function symbols $+$, $-$, and 0 and the equality symbol E. The axioms of the theory, are displayed in Figure 2.37. The first formula states that the operation $+$ is associative, the second that 0 is a left zero, the third that 0 is right zero, the fourth that $-$ is a left inverse for $+$, and the fifth that $-$ is right inverse for $+$. We use the notation introduced in Example 2.6.14.

Let us look at some models of this theory. The universe of the first model is the set of real numbers R. The symbols $+$, $-$, 0 are interpreted as the $+$, the opposite, and the 0 of the real numbers.

The second model has as universe the set of real numbers minus zero. The symbols $+$, $-$, 0 of L are the multiplication, the inverse, and the 1 of the real numbers.

A third model has as universe the permutations of the set $\{0, 1, 2\}$. The symbols $+$, $-$, 0 are the composition, the inverse permutation and the identity permutation.

(1) $\forall x R(x,x)$ (reflexivity)
(2) $\forall x \neg R(x,x)$ (irreflexivity)
(3) $\forall x \forall y (R(x,y) \longrightarrow R(y,x))$ (symmetry)
(4) $\forall x \forall y ((R(x,y) \wedge R(y,x)) \longrightarrow x \doteq y)$ (antisymmetry)
(5) $\forall x \exists y R(y,x)$ (left connectiveness)
(6) $\forall x \exists y R(x,y)$ (right connectiveness)
(7) $\forall x \forall y \forall z ((R(x,y) \wedge R(y,z)) \longrightarrow R(x,z))$ (transitivity)
(8) $\forall x \forall y (R(x,y) \vee x \doteq y \vee R(y,x))$ (linearity)

Figure 2.38: Relation properties

In all three models we did not specify the interpretations of the variables because the axioms are closed formulas, so the truth value of the axiom does not depend on the particular interpretations of the variables.

Now let us use first order theories to describe relations on a set. The alphabet consists of a binary predicate R and the identity $\doteq$. There is no need to specify the domain of the relation because every model $\mathcal{A}$ of R has a universe and $R^{\mathcal{A}}$ is a relation on the universe.

Figure 2.38 shows the formulas that correspond to reflexivity, irreflexivity, symmetry, antisymmetry, left and right connectiveness, transitivity, and linearity. Properties like R is a *well ordering* and R is a *well founded ordering* cannot be formalized in this theory. Let us see why. We recall that a relation R is a well ordering on the set A if all non-empty subsets of A have a least element. Similarly, R is a well-founded ordering on the set A if all nonempty subsets of A have a minimal element. But how do we describe the fact that S is a subset of A? This is not hard since every unary predicate P is interpreted as a subset of the universe. That subset is the set of all elements d of the universe that satisfy $P^{\mathcal{A}}[d] = 1$.

So, the statement that a non-empty subset has a least element is written as (†).

(†) $\exists x P(x) \longrightarrow \exists y (P(y) \wedge \forall z ((P(z) \wedge \neg y \doteq z) \longrightarrow R(z,y)))$.

The formula $\exists x P(x)$ tells us that the subset defined by P is not empty, and $\exists y (P(y) \wedge \forall z ((P(z) \wedge \neg y \doteq z) \longrightarrow R(z,y)))$ tells us that the subset has a minimal element (namely y).

However, if we want to say that *every* non-empty subset has a least element we must write

(‡) $\forall P (\exists x P(x) \longrightarrow \exists y (P(y) \wedge \forall z ((P(z) \wedge \neg y \doteq z) \longrightarrow R(z,y)))$.

But what have we done? We quantified a *predicate symbol*! This is allowed in *second order logic*, but not in first order logic. So, formula (‡) is not in the first order logic[4].

The *theorems* of a theory are the formulas that can be obtained from its axioms by some well defined *inference rules*, like the resolution, that will be

[4]The formula R *is well-founded* is a FOL formula in set theory. There *well-founded(R)* is $irreflexive(R) \wedge transitive(R) \wedge \forall x ((x \neq \phi \wedge x \subseteq domain(R)) \longrightarrow \exists y (y \in x \wedge \forall z ((z \in x \wedge y \neq z) \longrightarrow R(y,z))))$.

discussed in Section 2.11. The inference rules are said to be *sound* when the theorems are valid in every model of the theory. If every consequence of the axioms is a theorem, we say that the set of the inference rules is *complete*.

Exercises

Exercise 2.6.1 *Find a formula whose models have at most 4 elements.*

Exercise 2.6.2 *Find a formula whose models have at least 4 elements.*

Exercise 2.6.3 *Find a formula whose models have exactly 4 elements.*

Exercise 2.6.4 *Write a formula F that is satisfied by all structures of size n.*

Exercise 2.6.5 *Prove Case 8 of Part 3 of The Homomorphism Teorem.*

Exercise 2.6.6 *Let ϕ be an isomomorphism from $\mathcal{A}$ to $\mathcal{B}$ and F be a formula. Prove that $\models_{\mathcal{A}} F$ iff $\models_{\mathcal{B}} F$*

Exercise 2.6.7 *We say that two structures $\mathcal{A}$ and $\mathcal{B}$ are elementarily equivalent, written $\mathcal{A} \equiv \mathcal{B}$, if they satisfy the same set of formulas. Prove that whenever $\mathcal{B}$ is elementarily equivalent to a structure with n elements, $\mathcal{B}$ must also have n elements.*

Exercise 2.6.8 *Find two countably infinite structure $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \equiv \mathcal{B}$, but the universe of $\mathcal{A}$ is a proper subset of the universe of $\mathcal{A}$.*

Exercise 2.6.9 *Define a group homomorphism.*

Exercise 2.6.10 *The theory of semigroups has an operation $+$ that is associative. Show that the structure with universe $\{\mathbf{a}, \mathbf{b}\}^*$ and the concatenation operation is a semigroup.*

Exercise 2.6.11 *Write the formulas that define the algebraic structure named ring. A ring has two operations, * and +. The operation + is associative, commutative, has an identity written 0, and every element has an inverse (with regard to +). The operation * is associative. Moreover the operation * distributes over +, i.e. the equalities*

$x * (y + z) \doteq (x * y) + (x * z)$ *and* $(y + z) * x \doteq (y * z) + (x * z)$

are valid for all x, y, z.

Exercise 2.6.12 *Write a formula that states that a nonempty subset has a minimal element.*

Exercise 2.6.13 *Write the axioms of the set theory listed in the appendix as first order formulas.*

2.7 The equivalence of the 4 types of predicate calculus

In Section 2.1 we defined 4 types of predicate calculi: the *pure predicate calculus* that has no function symbols and no equality predicate, the *predicate calculus with equality* that has no function symbols but contains the equality , the *predicate calculus with functions* that has function symbols but does not contain the equality predicate, and the *full predicate calculus* that contains function symbols as well as the equality. In this section we will show that these 4 calculi (the plural of calculus) are equivalent in the following sense: there are procedures that convert formulas from any of the 4 calculi into s-equivalent formulas in any of the other calculi.

From the definition of the 4 calculi we know that

(1) the formulas of the pure predicate calculus are included in all the 4 calculi, and

(2) the formulas of all 4 predicate calculi are included in the full predicate calculus.

So, all what we need is a procedure that takes as input a formula F in the full calculus and computes a formula G s-equivalent to F in the pure predicate calculus.

This procedure will establish not only the equivalence of the full calculus with the pure calculus, but the equivalence of all 4 calculi. For example, let us show that the calculus with equality is equivalent to the predicate calculus with functions.

We need to show that

(3) every formula in the predicate calculus with equality has an s-equivalent formula in the predicate calculus with functions, and

(4) every formula in the predicate calculus with functions has an s-equivalent formula in the predicate calculus with equality.

Let us prove (3). Let F be a formula in the predicate calculus with equality. Then F belongs to the full calculus. We apply the procedure to F and get a formula G, s-equivalent to F, in the pure calculus. Since the pure calculus is included in the calculus with functions, G belongs to the predicate calculus with functions.

The proof of 4 is identical to the proof of 3, except that we replace *predicate calculus with functions* by *predicate calculus with equality* and we replace *predicate calculus with equality* by *predicate calculus with functions.*

Definition 2.7.1 (pure form) *A pure form of the formula F is a formula G, s-equivalent to F, that does not contain neither functions, nor equality.*

Figure 2.39 sketches an algorithm that computes pure forms. We will describe each step and accompanied by an example and a correctness proof.

Step 1 finds a prenex form of F.

Input: Formula F of the full predicate calculus.
Output: A pure form of F.
Step 1: Compute a prenex form of F.
Step 2: Eliminate the constants from the prenex form.
Step 3: Eliminate the function of arity greater than 0, from the output of Step 2.
Step 4: Eliminate E from the ouput of Step 3.

Figure 2.39: Algorithm for computing a pure form

Input: A prenex form $H = Q_1x_1 \dots Q_mx_mH^M$.
Output: A prenex form I, s-equivalent to H, that has no constants.
$n = 0$; $I_n = H$;
while [I_n contains constants] do
{
choose a constant a from I_n;
find a variable z that is not in I_n;
$n = n + 1$;
$I_n = \exists z I_{n-1}[a/z]$;
}
$I = I_n$;

Figure 2.40: Algorithm for removing constants

We already know how to do this from Section 2.5. The correctness of this step is provided by The Prenex Form Lemma. We get the formula $H = Q_1x_1 \dots Q_mx_mH^M$, where H^M is the matrix of H.

Step 2 replaces the constants of H by existentially quantified variables. We do a reverse Skolemization by replacing constants with existential variables, as shown in Figure 2.40.

Let us trace this algorithm for $H = \forall x \exists y((P(x, a) \vee Q(f(x))) \wedge (\neg P(y, b) \vee Q(x)))$.

Since $I_0 = H$ has constants, we execute the loop body. We replace a by a new variable, say z. We obtain $I_1 = \exists z \forall x \exists y((P(x, z) \vee Q(f(x))) \wedge (\neg P(y, b) \vee Q(x)))$.

The formula I_1 has constants, so we replace b by the new variable u. We get $I_2 = \exists u \exists z \forall x \exists y((P(x, z) \vee Q(f(x))) \wedge (\neg P(y, u) \vee Q(x)))$.

Since I_2 has no more constants, $I = I_2$.

Lemma 2.7.2 *The algorithm for removing constants is correct.*

Proof: The formula H has finitely many constants and each step eliminates one constant. So, the algorithm terminates. At the same time, $I_j = \exists z I_{j-1}[a/z]$ means that I_{j-1} is a Skolemization of I_j. So, they are s-equivalent by The Skolemization Lemma. **Q. E. D.**

Step 3 eliminate the function symbols of arity greater than 0 from I.

Input: A function symbol f of arity $n > 0$ and a formula $K = Q_1x_1Q_2x_2\dots Q_lx_lK^M$, where K^M is the matrix of K.
Output: A formula L, s-equivalent to F, that has no occurrences of f.
pick a predicate letter P_f of arity $n+1$ that does not occur in K;
$m = 0$; $K_m = K$;
while [K_m has occurrences of f] do
{
pick a term $f(t_1,\dots,t_n)$ of $K_m = Q_1x_1Q_2x_2\dots Q_{l_m}x_{l_m}F_m^M$ that
has only one occurrence of f;
choose a variable z that does not occur in F_m ; $m = m+1$;
$F_m = Q_1x_1Q_2x_2\dots Q_{l_m}x_{l_m}\exists z(P_f(t_1,\dots,t_n,z) \wedge F_{m-1}^M[f(t_1,\dots,t_n)/z])$,
where $F_{m-1}^M[f(t_1,\dots,t_n)/z]$ is the result of replacing
every occurrence of $f(t_1,\dots,t_n)$ in F_{m-1}^M by z;
}
$L = U \wedge V \wedge K_m$, where U and V are the formulas (7),respectively (8);

Figure 2.41: The function removal algorithm

The idea is to capture the meaning of the function f of arity $n > 0$ by a relation (predicate) P_f of arity $n+1$. We remember that a function is a relation that is

(5) defined for all elements of its domain, and

(6) for any given element of the domain the function has at most one value.

In first order logic these properties are written as

(7) $U = \forall y_1\forall y_2\dots\forall y_n\exists x P_f(y_1,y_2,\dots,y_n,x)$, and

(8) $V = \forall y_1\forall y_2\dots\forall y_n\forall u\forall v(\neg P_f(y_1,y_2,\dots,y_n,u) \vee \neg P_f(y_1,y_2,\dots,y_n,v) \vee E(u,v))$.

The formula (7) tells us that $f(y_1,\dots,y_n)$ has values, and (8) states that $f(y_1,\dots,y_n)$ has at most one value. Together they say that $f(y_1,\dots,y_n)$ has a unique value. Then $z = f(y_1,\dots,y_n)$ is represented by the formula $\exists z P_f(y_1,\dots,y_n,z)$.

With this definition in mind we replace $f(t_1,\dots,t_n)$ by a new variable z that satisfies the formula $\exists z P_f(t_1,\dots,t_n,z)$.

The formula may have several f-terms occurrences, so we replace them one at a time, starting with the terms that do not have f-occurrences in their subterms. This is done by The Function Removal Algorithm from Figure 2.41.

Example 2.7.3 Let us eliminate the function symbol f from the formula $K = \forall x\exists y(P(x,f(y)) \wedge \neg P(g(a,f(y)),f(g(f(x),y))))$.

First we choose a binary predicate P_f that is not in K. Then we make the assignments $m = 0$ and $K_0 = K$. We search K_0 for f-terms and encounter $f(y)$, $f(x)$, and $f(g(f(x),y))$. The last term, $f(g(f(x),y))$, has 2 occurrences of f, so we first eliminate $f(y)$. We choose the variable z_1 and substitute it for every occurrence of $f(y)$. From $K_0 = \forall x\exists y(P(x,f(y)) \wedge \neg P(g(a,f(y)),f(g(f(x),y))))$ we get $K_1 = \forall x\exists y\exists z_1(P_f(y,z_1) \wedge (P(x,z_1) \wedge \neg P(g(a,z_1),f(g(f(x),y)))))$.

Now we go back to the loop repetition test and find out that K_1 contains occurrences of f. So, we repeat the loop. We choose the new variable z_2 to replace the term $f(x)$. In K_1, we insert the quantifier $\exists z_2$ after $\exists z_1$, and we replace every $f(x)$ in the matrix of K_1 by z_2. We obtain

$K_2 = \forall x \exists y \exists z_1 \exists z_2 (P_f(x, z_1) \wedge F_1^M[f(x)/z])$

$= \forall x \exists y \exists z_1 \exists z_2 (P_f(x, z_2) \wedge (P_f(y, z_1) \wedge (P(x, z_1) \wedge \neg P(g(a, z_1), f(g(z_2, y))))))$.

We go back to the loop repetition test. The formula K_2 has occurrences of f so we execute the loop. We choose the variable z_3 to replace $f(g(z_2, y))$. In K_2 we insert $\exists z_3$ after $\exists z_2$ and we replace the matrix of K_2 by $(P_f(g(z_2, y), z_3) \wedge K_2^M[f(g(z_2, y))/z_3])$, where $K_2^M[f(g(z_2, y))/z_3]$ is the result of substituting z_3 for every $f(g(z_2, y))$ in K_2^M.

We get the formula

$F_3 = \forall x \exists y \exists z_1 \exists z_2 \exists z_3 (P_f(g(z_2, y), z_3) \wedge (P_f(x, z_2) \wedge (P_f(y, z_1) \wedge (P(x, z_1) \wedge \neg P(g(a, z_1), z_3)))))$.

We go to the repetition test and discover that K_3 has no occurrences of f. So, we end the loop and have we the have

$L = \forall y \exists x P_f(y, x) \wedge \forall y_1 \forall u \forall v (\neg P_f(y_1, u) \wedge \neg P_f(y_1, v) \vee E(u, v)) \wedge K_3$

$= \forall y \exists x P_f(y, x) \wedge \forall y \forall u \forall v (\neg P_f(y, u) \wedge \neg P_f(y, v)) \vee E(u, v)) \wedge$

$\forall x \exists y \exists z_1 \exists z_2 \exists z_3 (P_f(g(z_2, y), z_3) \wedge (P_f(x, z_2) \wedge (P_f(y, z_1) \wedge (P(x, z_1) \wedge \neg P(g(a, z_1), z_3)))))$.

Observation 2.7.4 1. In general, the formula L is significantly longer than the formula K.

2. All the formulas K_m are in prenex form.

Now let us prove that $G \equiv_s F$.

Lemma 2.7.5 *The Function Removal Algorithm is correct.*

Proof: We need to show that the procedure terminates and $K \equiv_s L$. The termination is easy; at each step we eliminate an f-term. Since K is finite, the algorithm terminates.

Now we show the s-equivalence. We define the formulas $N_i = U \wedge V \wedge K_i$. Since the algorithm terminates, we have a sequence of formulas $N_0, \ldots, N_r = L$ such that N_0 has no P_f predicates, and N_r has no f functions. We prove (1) and (2).

(1) If K is satisfiable, then so are $N_0, \ldots, N_r$.

(2) If L is satisfiable, so are $N_r, \ldots, N_0$.

The proof of (1).

Let $\mathcal{A}$ be a model of K and let D be the universe of $\mathcal{A}$. Let us define the model $\mathcal{B}$ that differ from $\mathcal{A}$ only on the interpretation of P_f. The predicate P_f is interpreted as

for all $d_1, d_2, \ldots, d_n, d \in D$,

(3) $P_f^{\mathcal{B}}[d_1, \ldots, d_n, d] = 1$ iff $f^{\mathcal{A}}[d_1, \ldots, d_n] = d$.

First of all, we prove that $\mathcal{B}[U] = 1$ and $\mathcal{B}[V] = 1$.

$\mathcal{B}[U] = 1$ iff for all $d_1, d_2, \ldots, d_n \in D$, there is a $d \in D$ such that

(4) $P_f^{\mathcal{B}}[d_1, \ldots, d_n, d] = 1$

From (3) we get that that $f^{\mathcal{A}}[d_1, \ldots, d_n]$ satisfies (4).

So, $\mathcal{B}[U] = 1$. Now, $\mathcal{B}[V] = 1$ iff for all $d_1, d_2, \ldots, d_n, d, e \in D$,
$P_f^{\mathcal{B}}[d_1, \ldots, d_n, d] = 1$ and $P_f^{\mathcal{B}}[d_1, \ldots, d_n, e] = 1$ imply $d = e$.
Again we use (3).
From (3) and $P_f^{\mathcal{B}}[d_1, \ldots, d_n, d] = 1$ we get
(5) $f^{\mathcal{A}}[d_1, \ldots, d_n] = d$.
From (3) and $P_f^{\mathcal{B}}[d_1, \ldots, d_n, e] = 1$ we get
(6) $f^{\mathcal{A}}[d_1, \ldots, d_n] = e$
Since $f^{\mathcal{A}}$ is a function, $d = e$. So, $\mathcal{B}[V] = 1$.
We will show that $\mathcal{B}$ is a model of N_i by finite induction on i.
Basis: $\models_{\mathcal{B}} N_0$.
The structures $\mathcal{A}$ and $\mathcal{B}$ agree on $K_0 = K$ because P_f is not in K. So, $\mathcal{B}[K] = \mathcal{A}[K] = 1$. It remains to show that $\models_{\mathcal{B}} U$ and $\models_{\mathcal{B}} V$.
$\mathcal{B}[U] = 1$
iff $\mathcal{B}[\forall y_1 \forall y_2 \ldots \forall y_n \exists x P_f(y_1, y_2, \ldots, y_n, x)] = 1$
iff for all $d_1, \ldots, d_n \in D$, $\mathcal{B}_{[y_1 \leftarrow d_1, \ldots, y_n \leftarrow d_n]}[\exists x P_f(y_1, y_2, \ldots, y_n, x)] = 1$
iff for all $d_1, \ldots, d_n \in D$, there is some $d \in D$ such that
$\mathcal{B}_{[y_1 \leftarrow d_1, \ldots, y_n \leftarrow d_n, x \leftarrow d]}[P_f(y_1, y_2, \ldots, y_n, x)] = 1$
iff for all $d_1, \ldots, d_n \in D$, there is some $d \in D$ such that $P_f^{\mathcal{B}}[d_1, \ldots, d_n, d] = 1$.
Now let $d_1, \ldots, d_n \in D$ and let $d = f^{\mathcal{A}}[d_1, \ldots, d_n]$. Then, by (3), $P_f^{\mathcal{B}}[d_1, \ldots, d_n, d] = 1$. Since $d_1, \ldots, d_n \in D$ are arbitrary we conclude that
for all $d_1, \ldots, d_n \in D$, there is some $d \in D$ such that $P_f^{\mathcal{B}}[d_1, \ldots, d_n, d] = 1$,
i.e. $\mathcal{B}[U] = 1$.
Next we show that $\mathcal{B}$ satisfies $V = \forall y_1 \forall y_2 \ldots \forall y_n \forall u \forall v (\neg P_f(y_1, y_2, \ldots, y_n, u) \vee \neg P_f(y_1, y_2, \ldots, y_n, v) \vee E(u, v))$.
All we need to show is that (7) is true.
(7) for all $d_1, \ldots, d_n, d, e \in D$,
$\mathcal{B}[y_1 \leftarrow d_1, \ldots, y_n \leftarrow d_n, u \leftarrow d, v \leftarrow e][\neg P_f(y_1, y_2, \ldots, y_n, u) \vee \neg P_f(y_1, y_2, \ldots, y_n, v) \vee E(u, v))$.
The statement (7) reduces to (8).
(8) for all $d_1, \ldots, d_n, d, e \in D$,
if $\mathcal{B}[y_1 \leftarrow d_1, \ldots, y_n \leftarrow d_n, u \leftarrow d, v \leftarrow e][P_f(y_1, y_2, \ldots, y_n, u)] = 1$
and $\mathcal{B}[y_1 \leftarrow d_1, \ldots, y_n \leftarrow d_n, u \leftarrow d, v \leftarrow e][P_f(y_1, y_2, \ldots, y_n, u)] = 1$, then
$\mathcal{B}[y_1 \leftarrow d_1, \ldots, y_n \leftarrow d_n, u \leftarrow d, v \leftarrow e][u] = \mathcal{B}[y_1 \leftarrow d_1, \ldots, y_n \leftarrow d_n, u \leftarrow d, v \leftarrow e][v]$.
So, let us assume that $\mathcal{B}[y_1 \leftarrow d_1, \ldots, y_n \leftarrow d_n, u \leftarrow d, v \leftarrow e][P_f(y_1, y_2, \ldots, y_n, u)] = 1$ and $\mathcal{B}[y_1 \leftarrow d_1, \ldots, y_n \leftarrow d_n, u \leftarrow d, v \leftarrow e][P_f(y_1, y_2, \ldots, y_n, u)] = 1$.
We know that $\mathcal{B}[y_1 \leftarrow d_1, \ldots, y_n \leftarrow d_n, u \leftarrow d, v \leftarrow e][u] = d$ and
$\mathcal{B}[y_1 \leftarrow d_1, \ldots, y_n \leftarrow d_n, u \leftarrow d, v \leftarrow e][v] = e$.
Since $\mathcal{B}[y_1 \leftarrow d_1, \ldots, y_n \leftarrow d_n, u \leftarrow d, v \leftarrow e][P_f(y_1, y_2, \ldots, y_n, u)] = 1$ reduces to $P_f^{\mathcal{B}}[d_1, \ldots, d_n, d] = 1$, the relation (3) implies that $d = f_{\mathcal{A}}[d_1, \ldots, d_n]$.
By the same reason, $\mathcal{B}[y_1 \leftarrow d_1, \ldots, y_n \leftarrow d_n, u \leftarrow d, v \leftarrow e][P_f(y_1, y_2, \ldots, y_n, v)] = 1$ implies that $e = f^{\mathcal{A}}[d_1, \ldots, d_n]$. Since both d and e are values of the function $f^{\mathcal{A}}$ at $< d_1, \ldots, d_n >$, $d = e$.
So, $\models_{\mathcal{B}} V$.
Inductive Step: $\models_{\mathcal{B}} N_i$ implies $\models_{\mathcal{B}} N_{i+1}$.

Let us assume that $\mathcal{B}$ is a model of $N_i = U \wedge V \wedge K_i$ and let K_i be the prenex form $Q_1x_1 \ldots Q_lx_l{K_i}^M$. Then $N_{i+1} = U \wedge V \wedge K_{i+1}$, where $K_{i+1} = Q_1x_1 \ldots Q_lx_l\exists z(P_f(t_1, \ldots, t_n, z) \wedge {K_i}^M[f(t_1, \ldots, t_n)/z])$.

We know, from the basis, that $\mathcal{B}$ satisfies U and V, so we need only to show that $\mathcal{B}$ is a model of N_{i+1}.

Let $d_1, \ldots, d_l$ be l elements of D, the universe of $\mathcal{B}$. In order to simplify the notation we will write

(9) $\mathcal{B}^*$ instead of $\mathcal{B}_{[x_1 \leftarrow d_1, \ldots, x_l \leftarrow d_l]}$.

Let

(10) $\mathcal{B}^*[t_1] = e_1$, ..., $\mathcal{B}^*[t_n] = e_n$, and

(11) $f^{\mathcal{A}}[e_1, \ldots, e_n] = e$.

Let us show that

(12) $\mathcal{B}^*_{[z \leftarrow e]}[P_f(t_1, \ldots, t_n, z)] = 1$.

$\mathcal{B}^*_{[z \leftarrow e]}[P_f(t_1, \ldots, t_n, z)] = 1$

iff $P^{\mathcal{B}}[\mathcal{B}^*_{[z \leftarrow e]}[t_1], \ldots, \mathcal{B}^*_{[z \leftarrow e]}[t_n], \mathcal{B}^*_{[z \leftarrow e]}[z]] = 1$ by the interpretation of $P_f(t_1, \ldots, t_n, z)$

iff $P^{\mathcal{B}}[\mathcal{B}^*[t_1], \ldots, \mathcal{B}^*[t_n], e] = 1$ because z is not in $t_1, \ldots, t_n$

iff $P^{\mathcal{B}}[e_1, \ldots, e_n, e] = 1$ by (10)

iff $f^{\mathcal{A}}[e_1, \ldots, e_n] = e$ by (3)

The last assertion is exactly (11), so it is true.

Now let us show that

(13)$\mathcal{B}^*_{[z \leftarrow e]}[{K_i}^M[f(t_1, \ldots, t_n)/z]] = \mathcal{B}^*[{K_i}^M]$.

$\mathcal{B}^*_{[z \leftarrow e]}[{K_i}^M[f(t_1, \ldots, t_n)/z]]$

$= \mathcal{B}^*_{[z \leftarrow \mathcal{B}^*[f(t_1, \ldots, t_n)]]}[{K_i}^M[f(t_1, \ldots, t_n)/z]]$ by (10) and (11)

$= \mathcal{B}^*[{K_i}^M[f(t_1, \ldots, t_n)/z][z/f(t_1, \ldots, t_n)]]$ by The Translation Lemma

$= \mathcal{B}^*[{K_i}^M]$ because the occurrences of $f(t_1, \ldots, t_n)$ we changed to z and then back to $f(t_1, \ldots t_n)$.

(12) and (13) tells us that

(14) $\mathcal{B}^*_{[z \leftarrow e]}[P_f(t_1, \ldots, t_n, z) \wedge {K_i}^M[f(t_1, \ldots, t_n)/z]] = \mathcal{B}^*[{K_i}^M]$.

From (14) we get that

(15) $\mathcal{B}[K_i] = \mathcal{B}[K_{i+1}]$.

Now let us prove (2), that whenever $L = N_r$ is satisfiable, so is K.

Let $\mathcal{C}$ be a model of L and let D be its domain. Since L has no occurrence of f we define a structure $\mathcal{E}$ that is exactly like $\mathcal{C}$ except for the interpretation of f. We define $f^{\mathcal{E}}$ as

(16) for all $d_1, \ldots, d_n, e \in D$, $f^{\mathcal{E}}[d_1, \ldots, d_n] = e$ iff $P_f^{\mathcal{C}}[d_1, \ldots, d_n, e] = 1$.

Now let us show that $f^{\mathcal{E}}$ is indeed a function, i.e. that for all $d_1, \ldots, d_n, e \in D$, $f^{\mathcal{E}}[d_1, \ldots, d_n]$ has one and only one value. Since $\mathcal{C}$ models $L = U \wedge V \wedge K_r$, it models U and V. From U we get that there is some $e \in D$ such that

(17) ${P_f}^{\mathcal{C}}[d_1, \ldots, d_n, e] = 1$.

So, $f^{\mathcal{E}}[d_1, \ldots, d_n]$ has a value. If it has two values, e_1 and e_2, then we have

(18) ${P_f}^{\mathcal{C}}[d_1, \ldots, d_n, e_1] = 1$

and

(19) ${P_f}^{\mathcal{C}}[d_1, \ldots, d_n, e_2] = 1$.

Since $\mathcal{C}$ satisfies V, we get

(20) $\mathcal{C}_{[y_1\leftarrow d_1,\ldots,y_n\leftarrow d_n,u\leftarrow e_1,v\leftarrow e_2]}[\neg P_f(y_1,y_2,\ldots,y_n,u)\vee\neg P_f(y_1,y_2,\ldots,y_n,v))\vee E(u,v)]=1$.

From (18), (19), and (20) we get

(21) $E^{\mathcal{C}}[e_1,e_2]=1$,

or $e_1=e_2$. So, the function $f^{\mathcal{E}}$ is well defined.

Since f is not in N_r, $\mathcal{E}[N_r]=\mathcal{C}[N_r]=1$.

Now we will show that

(22) $\models_{\mathcal{E}}N_{i+1}$ implies $\models_{\mathcal{E}}N_i$.

Let us assume that $K_i=Q_1x_1\ldots Q_lx_lK_i{}^M$, where $K_i{}^M$ is the matrix of K_i. Since $N_{i+1}=U\wedge V\wedge Q_1x_1\ldots Q_lx_l\exists z(P_f(t_1,\ldots,t_n,z)\wedge K_i{}^M[f(t_1,\ldots,t_n)/z])$,

(23) $\mathcal{E}[U]=\mathcal{E}[V]=1$

and

(24) $\mathcal{E}[Q_1x_1\ldots Q_l\exists z(P_f(t_1,\ldots,t_n,z)\wedge K_i{}^M[f(t_1,\ldots,t_n)/z])]=1$.

Let $d_1,\ldots,d_l\in D$, and $\mathcal{E}^*=\mathcal{E}_{[x_1\leftarrow d_1,\ldots,x_l\leftarrow d_l]}$.

We will show that

(25) $\mathcal{E}^*[\exists z(P_f(t_1,\ldots,t_n,z)\wedge K_i{}^M[f(t_1,\ldots,t_n)/z])]=\mathcal{E}^*[K_i{}^M]$.

$\mathcal{E}^*[[Q_1x_1\ldots Q_lx_l\exists z(P_f(t_1,\ldots,t_n,z)\wedge K_i{}^M[f(t_1,\ldots,t_n)/z])]=1$

iff for some $e\in D$, $\mathcal{E}^*{}_{[z\leftarrow e]}[P_f(t_1,\ldots,t_n,z)]=1$ and $\mathcal{E}^*{}_{[z\leftarrow e]}[K_i{}^M[f(t_1,\ldots,t_n)/z]]=1$ by the interpretation of $\exists z$

iff for some $e\in D$, $P_f{}^{\mathcal{E}}[\mathcal{E}^*{}_{[z\leftarrow e]}[t_1],\ldots,\mathcal{E}^*{}_{[z\leftarrow e]}[t_n],e]=1$ and $\mathcal{E}^*{}_{[z\leftarrow e]}[K_i{}^M[f(t_1,\ldots,t_n)/z]]=1$ by the interpretation of $P_f(t_1,\ldots,t_n,z)$

iff for some $e\in D$, $P_f{}^{\mathcal{E}}[\mathcal{E}^*[t_1],\ldots,\mathcal{E}^*[t_n],e]=1$ and $\mathcal{E}^*{}_{[z\leftarrow e]}[K_i{}^M[f(t_1,\ldots,t_n)/z]]=1$ because z does not occur in $t_1,\ldots,t_n$

iff $\mathcal{E}^*{}_{[z\leftarrow f^{\mathcal{E}}[\mathcal{E}^*[t_1],\ldots,\mathcal{E}^*[t_n]]]}[K_i{}^M[f(t_1,\ldots,t_n)/z]]=1$ because $P_f{}^{\mathcal{E}}[\mathcal{E}^*[t_1],\ldots,\mathcal{E}^*[t_n],e]=1$ is the same as $e=f^{\mathcal{E}}[\mathcal{E}^*[t_1],\ldots,\mathcal{E}^*[t_n]]$

iff $\mathcal{E}^*[K_i{}^M[f(t_1,\ldots,t_n)/z][z/f(t_1,\ldots,t_n)]]=1$ by The Translation Lemma

iff $\mathcal{E}^*[K_i{}^M]$ because $K_i{}^M[f(t_1,\ldots,t_n)/z][z/f(t_1,\ldots,t_n)]=K_i{}^M$

We leave the task of showing that (25) implies $\models_{\mathcal{E}}N_i$ (Exercise 2.7.1).

So, by finite induction, $\mathcal{E}$ satisfies every N_i. Since $N_0=U\wedge V\wedge K_0$, and $K_0=K$, $\mathcal{E}$ satisfies K. This concludes the proof of (2). **Q.E.D.**

We remove the function symbols of arity greater than 0 from I by applying the algorithm from Figure 2.41 for each of such symbols. We get a formula J s-equivalent to I that has no function symbols.

Step 4: Eliminate the equality from J.

At this point, J does not have any function symbols. However, we give an algorithm that eliminates the equality predicate from *any* formula, not just those of the predicate calculus with equality. This generalized algorithm will be used in the next section, when we prove the Skolem-Lőwenheim Theorem.

The idea is to replace E by a binary predicate Q that has the essential properties of E. Namely, Q must be reflexive, symmetric, transitive and must be congruent for all functions and predicates of I.

Input: A formula J.
Output: A formula G s-equivalent to J that does not contain E.
The formula G is the conjunction of the following predicates:
1. the formula $J[E/Q]$ obtained by replacing every occurrence of E in J by Q,
2. the predicates U, V and W above,
3. a predicate Q_g for each function symbol g of arity greater than 0 that occurs in J.
3. a predicate Q_P for each predicate symbol P of arity $n > 0$ that occur in J.

Figure 2.42: Algorithm for eliminating the equality

We know, from Section 2.6, how to express the properties of reflexivity, symmetry and transitivity.

(Reflexivity) $U = \forall x Q(x, x)$

(Symmetry) $V = \forall x \forall y (Q(x, y) \longrightarrow Q(y, x))$

(Transitivity) $W = \forall x \forall y \forall z ((Q(x, y) \wedge Q(y, z)) \longrightarrow Q(x, z))$

We recall that the congruence properties tell us that in any function and predicate we can replace a term by an equivalent term. So, Q is congruent for the function g of arity $n > 0$ when the predicate Q_g holds.

$Q_g = \forall y_1 \forall y_2 \ldots \forall y_{n-1} \forall u \forall v (Q(u, v) \longrightarrow$
$Q(g(u, y_1, \ldots, y_{n-1}), g(v, y_1, \ldots, y_{n-1})) \wedge$
$Q(g(y_1, u, y_2, \ldots, y_{n-1}), g(y_1, u, y_2, \ldots, y_{n-1})) \wedge \ldots$
$\wedge Q(g(y_1, y_2, \ldots, y_{n-1}, u), g(y_1, y_2, \ldots, y_{n-1}, v)))$

Basically, Q_g, tells us that when $Q(u, v)$ holds we can replace u by v at any position i, $1 \leq i \leq n$, of g. So, $Q(g(u, y_1, \ldots, y_{n-1}), g(v, y_1, \ldots, y_{n-1}))$ says that we can do the replacement at position 1, $Q(g(y_1, u, y_2, \ldots, y_{n-1}), g(y_1, v, y_2, \ldots, y_{n-1}))$ states the same for position 2, ..., and $Q(g(y_1, y_2, \ldots, y_{n-1}, u), g(y_1, y_2, \ldots, y_{n-1}, v))$ for position n. We notice that in all the Q formula we replace only one term and keep the rest unchanged.

We define the congruence property for the predicates in a similar fashion. Q is congruent for the predicate symbol P of arity $n > 0$ when the formula Q_P holds.

$Q_P = \forall y_1 \forall y_2 \ldots \forall y_{n-1} \forall u \forall v (Q(u, v) \longrightarrow$
$((P(u, y_1, \ldots, y_{n-1}) \longleftrightarrow P(v, y_1, \ldots, y_{n-1})) \wedge$
$(P(y_1, u, y_2, \ldots, y_{n-1}) \longleftrightarrow P(y_1, u, y_2, \ldots, y_{n-1})) \wedge \ldots$
$\wedge (P(y_1, y_2, \ldots, y_{n-1}, u) \longleftrightarrow P(y_1, y_2, \ldots, y_{n-1}, v))))$

The formula Q_P tells us that whenever $Q(u, v)$ holds, we can replace u by v at any position in the predicate without changing the truth value of P. We notice that for the equality of predicates we used the connective $\longleftrightarrow$ while for the equality of terms we used Q. The reason is that the predicates can have only two values, 0 or 1, while the terms can have any value in the universe of the structure.

Figure 2.42 shows an algorithm that eliminates E from J. Now let us give an example.

Example 2.7.6 Let us construct G for the formula $J = \exists x \forall u(E(f(x), g(u, v)) \wedge (P(u, f(v)) \wedge \neg R(x)))$.

$G = J[E/Q] \wedge U \wedge V \wedge W \wedge Q_f \wedge Q_g \wedge Q_P \wedge Q_R$
$= \exists x \forall u(Q(f(x), g(u, v)) \wedge (P(u, f(v)) \wedge \neg R(x)))$
$\wedge \forall x Q(x, x)$
$\wedge \forall x \forall y(Q(x, y) \longrightarrow Q(y, x))$
$\wedge \forall x \forall y \forall z((Q(x, y) \wedge Q(y, z)) \longrightarrow Q(x, z))$
$\wedge \forall u \forall v(Q(u, v) \longrightarrow Q(f(u), f(v)))$
$\wedge \forall x \forall u \forall v(Q(u, v) \longrightarrow (Q(g(u, x), g(v, x)) \wedge Q(g(x, u), g(x, v))))$
$\wedge \forall x \forall y \forall z(Q(x, y) \longrightarrow ((P(x, z) \longleftrightarrow P(y, z)) \wedge (P(z, x) \longleftrightarrow P(z, y)))$
$\wedge \forall u \forall v(Q(u, v) \longrightarrow (R(u) \longleftrightarrow R(v)))$.

Before we go on to prove that $G \equiv_s F$, we show that whenever $\mathcal{B}$ interprets Q as the equality predicate, we can substitute Q for E.

Lemma 2.7.7 *If* $\models_{\mathcal{B}} \forall x \forall y(Q(x, y) \longleftrightarrow E(x, y))$ *then*
$\models_{\mathcal{B}} [F[Q/E] \longleftrightarrow F]$.

Proof: By structural induction on J.

Assume that
(1) $\models_{\mathcal{B}} \forall x \forall y(Q(x, y) \longleftrightarrow E(x, y)$.
Formula (1) is true iff
(2) for all $d_1, d_2 \in universe(\mathcal{B}), Q^{\mathcal{B}}[d_1, d_2] = 1$ iff $d_1 = d_2$.

Case 1: F is an atomic formula.

Subcase 1.1: The predicate symbol of F is not E.
Then $F[Q/E] = F$, so $\models_{\mathcal{B}}(F[Q/E] \longleftrightarrow F)$ reduces to
$\models_{\mathcal{B}}(F \longleftrightarrow F)$, which is true because $(F \longleftrightarrow F)$ is a tautology.
Subcase 1.2: $F = E(t_1, t_2)$.
Then $\mathcal{B}[Q(t_1, t_2)] = 1$
iff $Q^{\mathcal{B}}[\mathcal{B}[t_1], \mathcal{B}[t_2]] = 1$ by the interpretation of Q
iff $\mathcal{B}[t_1] = \mathcal{B}[t_2]$ by (2)
iff $\mathcal{B}[E(t_1, t_2)] = 1$ by the interpretation of E.

Case 2: $F = \neg G$.
By induction hypothesis $\models_{\mathcal{B}}(G[Q/E] \longleftrightarrow G)$.
We can show, by truth tables, that
$(G[Q/E] \longleftrightarrow G) \models \neg G[Q/E] \longleftrightarrow \neg G$.
So, $\models_{\mathcal{B}}(F[Q/E] \longleftrightarrow F)$.

Cases 3,4,5,6.
In all these cases $F = (GCH)$, C being one of the connectives $\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$. By induction hypotheses
(3) $\models_{\mathcal{B}}(G[Q/E] \longleftrightarrow G)$
and
(4) $\models_{\mathcal{B}}(H[Q/E] \longleftrightarrow H)$.
We can show, by truth tables, that
(5) $(G[Q/E] \longleftrightarrow G), (H[Q/E] \longleftrightarrow H) \models (G[Q/E]CH[Q/E]) \longleftrightarrow (GCH))$.
From (3), (4), and (5) we get that
$\models_{\mathcal{B}}(F[Q/E] \longleftrightarrow F)$.

Case 7: $F = \forall x G$.

Let $d \in universe(\mathcal{B})$. Since $\mathcal{B}$ satisfies (1), so does $\mathcal{B}_{[x \leftarrow d]}$ because they give the same interpretation to Q. So, we get

(6) $\models_{\mathcal{B}_{[x \leftarrow d]}} (G[Q/E] \longleftrightarrow G)$.

$\models_{\mathcal{B}} (F[Q/E] \longleftrightarrow F)$

iff $\mathcal{B}[F[Q/E]] = \mathcal{B}[F]$

Now we show that $\mathcal{B}[F[Q/E]] = \mathcal{B}[F]$.

$\mathcal{B}[F[Q/E] = 1$

iff $\mathcal{B}[\forall x G[Q/E]] = 1$

iff for all $d \in universe(\mathcal{B}), \mathcal{B}_{[x \leftarrow d]}[G[Q/E]] = 1$

iff for all $d \in universe(\mathcal{B}), \mathcal{B}_{[x \leftarrow d]}[G] = 1$ by (6)

iff $\mathcal{B}[\forall x G] = 1$

iff $\mathcal{B}[F] = 1$.

Case 8: $F = \exists x G$.

This case is similar to Case 7. It is left as exercise. **Q.E.D.**

Theorem 2.7.8 *The algorithm for removing the equality is correct.*

Proof: We need to show that G has a model if and only if J has a model.

$\Longleftarrow$ Let us assume that J has a model $\mathcal{A}$ with universe D. Let $\mathcal{B}$ be structure identical to $\mathcal{A}$ except for the interpretation of Q which is interpreted as the equality predicate. Then

$$Q^{\mathcal{B}}[d_1, d_2] = \begin{cases} 1 & \text{if } d_1 = d_2 \\ 0 & \text{if } d_1 \neq d_2 \end{cases}$$

We claim that $\mathcal{B}$ is a model of G. Since $\mathcal{B}$ interprets Q as the equality predicate it satisfies the properties of reflexivity, symmetry, transitivity, and congruence. At the same time $\mathcal{B}[F[Q/E]] = \mathcal{B}[F]$ by Lemma 2.7.7. So, $\mathcal{B}$ is a model of G.

$\Longrightarrow$

Let us assume that G has a model $\mathcal{A}$ with universe D. We will construct a model $\mathcal{B}$ of J. Since the relation $Q^{\mathcal{A}}$ is reflexive, symmetric and transitive, $Q^{\mathcal{A}}$ is an equivalence on D.

So, let the domain of $\mathcal{B}$ be the equivalence classes of $Q^{\mathcal{A}}$, i.e. $D_{\mathcal{B}} = \{d^* | d \in D\}$, where d^* is the $Q^{\mathcal{A}}$ equivalence class of d. Let us define the $\mathcal{B}$ interpretations of the function, predicate and variable symbols of G.

If a is a constant then

(1) $a^{\mathcal{B}} = [a^{\mathcal{A}}]^*$.

If f is a function of arity $n > 0$, then

(2) $f^{\mathcal{B}}[{d_1}^*, \ldots, {d_n}^*] = [f^{\mathcal{A}}[d_1, \ldots, d_n]]^*$.

If P is a predicate constant then

(3) $P^{\mathcal{B}} = P^{\mathcal{A}}$.

If P is predicate symbol of arity $n > 0$, then

(4) $P^{\mathcal{B}}[{d_1}^*, \ldots, {d_n}^*] = P^{\mathcal{A}}[d_1, \ldots, d_n]$.

We need to show that the values of $f^{\mathcal{B}}[d_1{}^*, \ldots, d_n{}^*]$ and $P^{\mathcal{B}}[d_1{}^*, \ldots, d_n{}^*]$ do not depend on the choice of the representatives $d_1, \ldots, d_n$. The proof that $f^{\mathcal{B}}[d_1{}^*, \ldots, d_n{}^*]$ is independent of the representatives is found in Section **??**.

We have to show that $P^{\mathcal{B}}[d_1{}^*, \ldots, d_n{}^*]$ does not depend on the choice of the representatives d_1,...,d_n.

We present the proof for the case $n = 3$ and leave the proof for the general case as exercise.

We need to show that for all values $d_1, d_2, d_3, e_1, e_2, e_3 \in D$,

(10) if $Q^{\mathcal{A}}[d_1, e_1] = 1$, $Q^{\mathcal{A}}[d_2, e_2] = 1$, and $Q^{\mathcal{A}}[d_3, e_3] = 1$ then

$P^{\mathcal{A}}[d_1, d_2, d_3] \boxed{\longleftrightarrow} P^{\mathcal{A}}[e_1, e_2, e_3] = 1$.

So let us assume that

(11) $Q^{\mathcal{A}}[d_1, e_1] = 1$, $Q^{\mathcal{A}}[d_2, e_2] = 1$, and $Q^{\mathcal{A}}[d_3, e_3] = 1$.

We need to show that

(12) $P^{\mathcal{A}}[d_1, d_2, d_3] \boxed{\longleftrightarrow} P^{\mathcal{A}}[e_1, e_2, e_3] = 1$.

The conjunct Q_P of G is

(13) $\forall y_1 \forall y_2 \forall u \forall v (Q(u, v) \longrightarrow ((P(u, y_1, y_2) \longleftrightarrow P(v, y_1, y_2)) \wedge$

$(P(y_1, u, y_2) \longleftrightarrow P(y_1, v, y_2)) \wedge (P(y_1, y_2, u) \longleftrightarrow P(y_1, y_2, v)))$.

This conjunct is interpreted by $\mathcal{A}$ as

for all $\alpha, \beta, \gamma, \delta$ in D

(14) $Q^{\mathcal{A}}[\alpha, \beta] \boxed{\longrightarrow} [[P^{\mathcal{A}}[\alpha, \gamma, \delta] \boxed{\longleftrightarrow} P^{\mathcal{A}}[\beta, \gamma, \delta]]$

$\boxed{\wedge} [P^{\mathcal{A}}[\gamma, \alpha, \delta] \boxed{\longleftrightarrow} P^{\mathcal{A}}[\gamma, \beta, \delta]] \boxed{\wedge} [P^{\mathcal{A}}[\gamma, \delta, \alpha] \boxed{\longleftrightarrow} P^{\mathcal{A}}[\gamma, \delta, \beta]]] = 1$.

(14) is equivalent to the conjunction of (15), (16), and (17) (see Exercise 2.7.8 below).

For all $\alpha, \beta, \gamma\delta$ in D,

(15) $Q^{\mathcal{A}}[\alpha, \beta] = 1$ implies $P^{\mathcal{A}}[\alpha, \gamma, \delta] \boxed{\longleftrightarrow} P^{\mathcal{A}}[\beta, \gamma, \delta] = 1$,

(16) $Q^{\mathcal{A}}[\alpha, \beta] = 1$ implies $P^{\mathcal{A}}[\gamma, \alpha, \delta] \boxed{\longleftrightarrow} P^{\mathcal{A}}[\gamma, \beta, \delta] = 1$,

(17) $Q^{\mathcal{A}}[\alpha, \beta] = 1$ implies $P^{\mathcal{A}}[\gamma, \delta, \beta] \boxed{\longleftrightarrow} P^{\mathcal{A}}[\gamma, \delta, \beta] = 1$.

Now we go on to prove (12).

In (15) we replace α by d_1, β by e_1, γ by d_2, and δ by d_3 and get

(18) $Q^{\mathcal{A}}[d_1, e_1] = 1$ implies $P^{\mathcal{A}}[d_1, d_2, d_3] \boxed{\longleftrightarrow} P^{\mathcal{A}}[e_1, d_2, d_3] = 1$.

From the first equality of (11) and (18) we get

(19)$P^{\mathcal{A}}[d_1, d_2, d_3] \boxed{\longleftrightarrow} P^{\mathcal{A}}[e_1, d_2, d_3] = 1$.

In (16) we replace α by d_2, β by e_2, γ by e_1, and δ by d_3 and get

(20) $Q^{\mathcal{A}}[d_2, e_2] = 1$ implies $P^{\mathcal{A}}[e_1, d_2, d_3] \boxed{\longleftrightarrow} P^{\mathcal{A}}[e_1, e_2, d_3] = 1$.

From the second equality of (11) and (20) we get

(21) $P^{\mathcal{A}}[e_1, d_2, d_3] \boxed{\longleftrightarrow} P^{\mathcal{A}}[e_1, e_2, d_3] = 1$.

In (17) we replace α by d_3, β by e_3, γ by e_1, and δ by e_2 in and get

(22) $Q^{\mathcal{A}}[d_3, e_3] = 1$ implies $P^{\mathcal{A}}[e_1, e_2, d_3] \boxed{\longleftrightarrow} P^{\mathcal{A}}[e_1, e_2, e_3] = 1$.

From the third equality of (11) and (22) we get

(23) $P^{\mathcal{A}}[e_1, e_2, d_3] \boxed{\longleftrightarrow} P^{\mathcal{A}}[e_1, e_2, e_3] = 1$.

Now we apply the transitivity of $Q^{\mathcal{A}}$ to the formulas (19), (21), and (23) and get

(24)$P^{\mathcal{A}}[d_1, d_2, d_3] \boxed{\longleftrightarrow} P^{\mathcal{A}}[e_1, e_2, e_3] = 1$.

which is exactly the formula (12) that we wanted to prove.

So, the relations $P^{\mathcal{B}} : D_{\mathcal{B}}{}^n \longrightarrow \{0, 1\}$ defined by (2) are functions.

Now we need to give the $\mathcal{B}$ interpretation for the free variables. The structure $\mathcal{B}$ interprets x as the equivalence class of x.

(5) $x^{\mathcal{B}} = [x^{\mathcal{A}}]^*$

Now let L be the set of all function and predicate symbols that occur in G to which we add E. Then G is an L formula and ϕ and L-homomorphism. At the same time, equations (1)-(5) tell us that $\phi : D \longrightarrow D_{\mathcal{B}}$ is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$. Moreover, ϕ is an epimorphism because every equivalence class in $D_{\mathcal{B}}$ is the image of some $d \in D$. By The L-homomorphism Theorem from the preceding section, $\models_{\mathcal{B}} G$. Since $J[Q/E]$ is a conjunct of G, $\models_{\mathcal{B}} J[Q/E]$.

Now let us show that for all

(6) $Q^{\mathcal{B}}[{d_1}^*, {d_2}^*] = 1$ iff ${d_1}^* = {d_2}^*$.

We have

$Q^{\mathcal{B}}[{d_1}^*, {d_2}^*] = 1$

iff $Q^{\mathcal{A}}[d_1, d_2] = 1$ $\quad\quad$ $\phi[d] = d^*$ is a homomorphism

iff ${d_1}^* = {d_2}^*$ $\quad\quad$ from Section 2.6

So, (6) holds and we can apply Lemma 2.7.7 and get that $\mathcal{B}$ is a model of J. **Q.E.D.**

Corollary 2.7.9 *If either one of G and J has a countable model, so does the other.*

Proof: From the preceding proof we know that for every model of J there is a model of G with the same universe. So, whenever J has a countable model, so does G.

Now assume that $\mathcal{A}$ is a countable model of G. Then J has a model having as elements the $Q^{\mathcal{A}}$ equivalence classes of the universe of $\mathcal{A}$. Since the equivalence classes are not empty we can, by The Axiom of Choice, assign to each equivalence class a representative in the universe of $\mathcal{A}$. Since distinct equivalence classes do not intersect, the representative function is one-to-one. By the Cantor-Bernstein Theorem, the set of equivalence classes is countable. **Q.E.D.**

Exercises

Exercise 2.7.1 *Let $\mathcal{E}$ be a structure with universe D. Show that (1) and (2) imply (3).*

(1) $\mathcal{E}[QxF] = 1$

(2) for all $d \in D$, $\mathcal{E}_{[x \leftarrow d]}[F] = \mathcal{E}_{[x \leftarrow d]}[G]$

(3) $\mathcal{E}[QxG] = 1$

Exercise 2.7.2 *Apply the algorithm for removing constants to eliminate a and b from the formula $H = \forall x \exists y(Q(x, a) \wedge (\neg Q(b, y) \vee \neg Q(f(x), x))$.*

Exercise 2.7.3 *Use the algorithm for removing a function of arity greater than 0 to eliminate g from $K = \forall x \forall y(Q(x, g(x, y)) \wedge (\neg Q(g(g(x, x), y), y) \vee Q(g(x, y), x)))$.*

Exercise 2.7.4 *Eliminate the equality from the formulas*

$F = (\neg E(x, y) \vee E(u, v))$

$G = \forall x \forall y \forall z(\neg P(x) \vee R(y, z) \vee \neg E(x, y) \vee E(x, z))$.

Exercise 2.7.5 *Provide truth table proofs for the consequences shown in the proof of Lemma 2.7.7.*

Exercise 2.7.6 *Prove Case 8 of Lemma 2.7.7.*

Exercise 2.7.7 *Let $Q^{\mathcal{A}}$ be a congruence on the universe of $\mathcal{A}$ for the predicate P of arity greater than 0. Let $[d]$ be the $Q^{\mathcal{A}}$ equivalence class of $d \in universe(\mathcal{A})$. Then*

$P^{\mathcal{B}}([d_1], \ldots, [d_n]) = P^{\mathcal{A}}[d_1, \ldots, d_n]$

is well defined, i.e. it does not depend on the choice of d_1,...,d_n.

Hint: All you have to do is to show that for all d_1,...,d_n,e_1,...,e_n in D,

$Q^{\mathcal{A}}[d_1, e_1] = 1$,...,$Q^{\mathcal{A}}[d_n, e_n] = 1$ imply $Q^{\mathcal{A}}[d_1, \ldots, d_n] \boxed{\longleftrightarrow} Q^{\mathcal{A}}[e_1, \ldots, e_n]$. Look at the proof for $n = 3$ as a guide.

Exercise 2.7.8 *Show that $F \longrightarrow (G \wedge H \wedge I) \equiv (F \longrightarrow G) \wedge (F \longrightarrow H) \wedge (F \longrightarrow I)$.*

2.8 Herbrand Structures

In this section we present a method for determining the unsatisfiability of the Skolem forms that do not contain E. The procedure starts by constructing the *Herbrand universe* of the formula, built from the function symbols that occur in it. Then, the procedure develops *the Herbrand expansion* by interpreting the variables of the matrix in that universe. Both the universe and the expansion have no variables, and the expansion becomes a set of formulas in the propositional logic. So, we can apply the techniques developed in the first chapter to determine the unsatisfiability of the set. The bad news is that in all but the most trivial cases the Herbrand expansion is infinite, so the procedure may not terminate.

The correctness of the procedure is assured by The Herbrand Theorem that states that the formula is unsatisfiable iff the expansion is unsatisfiable.

We recall, from Section 2.7, that every formula F is satisfiably equivalent to a formula G that does not contain the equality predicate E. From Section 2.5 we know that G has a Skolem normal form H. Moreover, since G has no occurrences of E, neither does H. So, every formula is satisfiably equivalent to a formula $\forall x_1 \ldots \forall x_n (C_1 \wedge \ldots C_m)$ where $C_1, \ldots, C_m$ are clauses with variables in the set $\{x_1, \ldots, x_n\}$.

Definition 2.8.1 (Herbrand universe) *Let F be a Skolem normal form. The Herbrand universe of F, written $D(F)$, is the set of terms defined below.*

1. All constants of F are in $D(F)$. If F has no constants then we put a constant, say a, into $D(F)$.

2. For every function symbol f of arity $n > 0$ that occurs in F, and for all n-tuples t_1, ...,t_n in $D(F)$, $f(t_1, \ldots, t_n)$ is also in $D(F)$.

Example 2.8.2 Let us find the Herbrand Universe of the formula $F = \forall x \forall y (P(x, f(y)) \wedge (\neg P(f(x), y) \vee \neg P(y, y)))$.

F has no constants, so we put the constant a into $D(F)$. Now we apply the second rule of the defintion of the Herbrand universe.

Since $a \in D$ and f is a unary function of F, $f(a) \in D$. We apply the same rule to $f(a)$ and f, and get $f(f(a)) \in D(F)$. By repeatingly applying the function f to the last entry in $D(F)$ we obtain the set $D(F) = \{a, f(a), f^2(a), f^3(a), \ldots, f^n(a), \ldots\}$, where $f^n(a)$ is the result of applying f to a n-times.

Example 2.8.3 Let us find the Herbrand universe of the formula $G = \forall x \forall y \forall z ((\neg P(x, g(a, x)) \vee \neg R(f(z))) \wedge (P(f(b), g(x, z)) \vee R(z)))$.

The formula F has two constants, a and b. So, we put them into $D(G)$.

Now we apply rule 2. The formula G has two functions of arity greater than 0, f and g. We apply them to the terms in $D(G)$ and get $f(a)$, $f(b)$, $g(a, a)$, $g(a, b)$, $g(b, a)$, $g(b, b)$.

Now we apply f and g to the terms of the terms a, b, $f(a)$, $f(b)$, $g(a, a)$, $g(a, b)$, $g(b, a)$, $g(b, b)$ and get the new terms,

$f(f(a)), f(f(b)), f(g(a, a)), f(g(a, b)), f(g(b, a)), f(g(b, b)), g(a, f(a)), g(a, f(b))$,
$g(a, g(a, a)), g(a, g(a, b)), g(a, g(b, a)), g(a, g(b, b)), g(b, f(a)), g(b, f(b)), g(b, g(a, a))$,
$g(b, g(a, b)), g(b, g(b, a)), g(b, g(b, b)), g(f(a), a), g(f(a), b), g(f(a), f(a)), g(f(a), f(b))$,
$g(f(a), g(a, a)),\ g(f(a), g(a, b)),\ g(f(a), g(b, a)),\ g(f(a), g(b, b)),\ g(f(b), a)$, $g(f(b), b)$,
$g(f(b), f(a)),\ g(f(b), f(b)),\ g(f(b), g(a, a)),\ g(f(b), g(a, b)),\ g(f(b), g(b, a))$, $g(f(b), g(b, b))$,
$g(g(a, a), a),\ g(g(a, a), b),\ g(g(a, a), f(a)),\ g(g(a, a), f(b)),\ g(g(a, a), g(a, a))$, $g(g(a, a), g(a, b))$,
$g(g(a, a), g(b, a)), g(g(a, a), g(b, b)), g(g(a, b), a), g(g(a, b), b), g(g(a, b), f(a))$, $g(g(a, b), f(b))$,
$g(g(a, b), g(a, a)), g(g(a, b), g(a, b)), g(g(a, b), g(b, a)), g(g(a, b), g(b, b)), g(g(b, a), a)$,
$g(g(b, a), b), g(g(b, a), f(a)), g(g(b, a), f(b)), g(g(b, a), g(a, a)), g(g(b, a), g(a, b))$,
$g(g(b, a), g(b, a)),\ g(g(b, a), g(b, b)),\ g(g(b, b), a),\ g(g(b, b), b),\ g(g(b, b), f(a))$, $g(g(b, b), f(b))$,
$g(g(b, b), g(a, a)), g(g(b, b), g(a, b)), g(g(b, b), g(b, a)), g(g(b, b), g(b, b))$.

We can keep going and apply the functions f and g to the existing set.

Observations 2.8.4 1. The Herbrand universe contains at least one constant.

2. If F contains function symbols of arity greater than 0, then $D(F)$ is countably infinite; otherwise $D(F)$ is finite.

Definition 2.8.5 ($D(F, n)$) *We write $D(F, n)$ for the set of Herbrand terms of height $\leq n$.*

Example 2.8.6 Let us find $D(F, n)$ for the formula F from Example 2.8.2.

$D(F, 0)$ is the set of constants of F, so $D(F, 0) = \{a\}$.

$D(F, 1)$ is the set of terms of height 0 or 1, so $D(F, 1) = \{a, f(a)\}$.

In general, $D(F, n)$, the set of Herbrand terms of height less than or equal to n, is $D(F, n) = \{a, f(a), \ldots, f^n(a)\}$.

Definition 2.8.7 (Herbrand structure) $\mathcal{H} =< D(F), \mathcal{H}^f, \mathcal{H}^p, \mathcal{H}^v >$ *is a Herbrand structure of* F *if it satisfies the following conditions:*

1. for every constant a *in* $D(F)$, $f^{\mathcal{H}}[a] = a$, *and*

2. for every function symbol g *in* F *of arity* $n > 0$ *and for all terms* $t_1, \ldots, t_n$ *in* $D(F)$, $g^{\mathcal{H}}[t_1, \ldots, t_n] = g(t_1, \ldots, t_n)$.

Observations 2.8.8 1. All Herbrand structures of F have the same universe, $D(F)$, and they agree on the function symbols of F.

2. The interpretation of variables, $\mathcal{H}^v$, is irrelevant since F has no free variables.

3. The Herbrand interpretations may differ on the interpretation of the predicate symbols of F.

4. Our definition of structure requires that all symbols in the language be interpreted, not just the symbols of the formula. However, we do not care about the interpretation of the symbols that are not in F. Hence, we will deal with the *restriction* of the structure to the symbols of F.

We can prove a stronger version of the first part of Observations 2.8.8.

Lemma 2.8.9 *Let* F *be a Skolem normal form without equality and* $D(F)$ *be the Herbrand universe of* F. *Let* $\mathcal{H}$ *be a Herbrand structure and* t *an element of* $D(F)$. *Then* $\mathcal{H}[t] = t$.

Proof: The proof is done by structural induction on t. It is left as exercise. **Q.E.D.**

Now let us show that every Skolem form without equality has a model iff it has a Herbrand model.

Theorem 2.8.10 (The Herbrand Theorem) *Let* F *be a Skolem normal form that does not contain the equality symbol. Then* F *has a model iff it has a Herbrand model.*

Proof: $\Longleftarrow$: If F has a Herbrand model, then it has a model.

$\Longrightarrow$: We will prove that whenever F has a model, it also has a Herbrand model. Let $\mathcal{A} =< D, \mathcal{A}^f, \mathcal{A}^P, \mathcal{A}^v >$ be a model of F. We will construct a Herbrand model $\mathcal{H}$ of F, by imitating the behavior of $\mathcal{A}$.

We define the mapping $\alpha : D(F) \longrightarrow D$ as follows:

1. if a is a constant in F, then $\alpha[a] = a^{\mathcal{A}}$;

2. if a is in $D(F)$ but not in F, we choose any element of D as $a^{\mathcal{A}}$ and set $\alpha[a] = a^{\mathcal{A}}$;

3. if g has arity $n > 0$ and $t_1, \ldots, t_n$ are terms in $D(F)$, then
$\alpha[g(t_1, \ldots, t_n)] = g^{\mathcal{A}}[\alpha[t_1], \ldots, \alpha[t_n]]$.

Now we define the $\mathcal{H}$ interpretation of the predicates P of F.

4. If P has arity 0, then $P^{\mathcal{H}} = P^{\mathcal{A}}$.

5. If P has arity $n > 0$ and $t_1, \ldots, t_n$ are terms in $D(F)$, then $P^{\mathcal{H}}[t_1, \ldots, t_n] = P^{\mathcal{A}}[\alpha[t_1], \ldots, \alpha[t_n]]$.

Now we prove that α maps every term t into the $\mathcal{A}$-interpretation of t.

Lemma 2.8.11 *For all terms $t \in D(F)$, $\alpha[t] = \mathcal{A}[t]$.*

Proof: Case 1: t is a constant, say a. Then $\alpha[a] = a^{\mathcal{A}}$ from Parts 1 and 2 of the definition of α.
Case 2: $t = g(t_1, \ldots, t_n)$. Then
$\alpha[g(t_1, \ldots, t_n)]$
$= g^{\mathcal{A}}[\alpha[t_1], \ldots, \alpha[t_n]]$ from part 3 of the definition of α
$= g^{\mathcal{A}}[\mathcal{A}[t_1], \ldots, \mathcal{A}[t_n]]$ from the induction hypotheses
$= \mathcal{A}[g(t_1, \ldots, t_n)]$ from the interpretation of $g(t_1, \ldots, t_n)$.
Q.E.D. lemma

Now let $F = \forall y_1 \forall y_2 \ldots \forall y_m F^M$, where F^M is the matrix of F. Let $\tau_1, \ldots, \tau_m$ be m terms in $D(F)$. We will show, by structural induction on the subformulas I of F^M, that
$\mathcal{H}_{[y_1 \leftarrow \tau_1, \ldots, y_m \leftarrow \tau_m]}[I] = \mathcal{A}_{[y_1 \leftarrow \alpha[\tau_1], \ldots, y_m \leftarrow \alpha[\tau_m]]}[I]$.
For simplicity, we abbreviate $\mathcal{H}_{[y_1 \leftarrow \tau_1, \ldots, y_m \leftarrow \tau_m]}$ by $\mathcal{H}^*$ and $\mathcal{A}_{[y_1 \leftarrow \alpha[\tau_1], \ldots, y_m \leftarrow \alpha[\tau_m]]}$ by $\mathcal{A}^*$.
Case 1: $I = P(t_1, \ldots, t_n)$ or $I = P$. The subcase $F = E(t_1, t_2)$ is missing because F does not contain the equality. The subcase $I = P$ is easy because
$\mathcal{H}^*[P]$
$= P^{\mathcal{H}}$ because $\mathcal{H}^*$ and $\mathcal{H}$ agree on P
$= P^{\mathcal{A}}$ from the construction of $\mathcal{H}$
$= \mathcal{A}^*[P]$ because $\mathcal{A}$ and $\mathcal{A}^*$ agree on P.
Now let us treat the subcase $I = P(t_1, \ldots, t_n)$.
$\mathcal{H}^*[P(t_1, \ldots, t_m)]$
$= \mathcal{H}[P(t_1, \ldots, t_n)[y_1/\tau_1, \ldots, y_m/\tau_m]]$ by The Translation Lemma
$= P^{\mathcal{H}}[t_1[y_1/\tau_1, \ldots, y_m/\tau_m], \ldots, t_n[y_1/\tau_1, \ldots, y_m/\tau_m]]$ from the interpretation of $P(t_1, \ldots, t_n)[y_1/\tau_1, \ldots, y_m/\tau_m]]$
$= P^{\mathcal{A}}[\alpha[t_1[y_1/\tau_1, \ldots, y_m/\tau_m]], \ldots, \alpha[t_n[y_1/\tau_1, \ldots, y_m/\tau_m]]]$ from the construction of $P^{\mathcal{H}}$
$= P^{\mathcal{A}}[\mathcal{A}[t_1[y_1/\tau_1, \ldots, y_m/\tau_m]], \ldots, \mathcal{A}[t_n[y_1/\tau_1, \ldots, y_m/\tau_m]]]$ from Lemma 2.8.11
$= P^{\mathcal{A}}[\mathcal{A}_{[y_1 \leftarrow \mathcal{A}[\tau_1], \ldots, y_m \leftarrow \mathcal{A}[\tau_m]]}[t_1], \ldots, \mathcal{A}_{[y_1 \leftarrow \mathcal{A}[\tau_1], \ldots, y_m \leftarrow \mathcal{A}[\tau_m]]}[t_n]]$ by The Translation Lemma for Terms
$= \mathcal{A}^*[P(t_1, \ldots, t_n)]$.
Case 2: $I = \neg J$.
$\mathcal{H}^*[\neg J]$
$= \boxed{\neg} \mathcal{H}^*[J]$ by the interpretation of $\neg$
$= \boxed{\neg} \mathcal{A}^*[J]$ by the induction hypothesis
$= \mathcal{A}^*[\neg J]$ by the interpretation of $\neg J$.
Cases 3,4,5,6: $I = JCK$, where C is one of the connectives $\vee$, $\wedge$, $\longrightarrow$, $\longleftrightarrow$.
$\mathcal{H}^*[JCK]$
$= \mathcal{H}^*[J] \boxed{C} \mathcal{H}^*[K]$ by the interpretation of C
$= \mathcal{A}^*[J] \boxed{C} \mathcal{A}^*[K]$ by the induction hypotheses
$= \mathcal{A}^*[JCK]$ by the interpretation of C.
Cases 7 and 8 do not apply since F^M has no quantifiers.

So, $\mathcal{H}^*[F^M] = \mathcal{A}^*[F^M]$. Since, $\mathcal{A}[F] = 1$, we get that $\mathcal{A}^*[F^M] = 1$ for all assignments $\mathcal{A}^*$. Then $\mathcal{H}^*[F^M] = 1$ for all assignments $[y_1 \leftarrow \tau_1, \ldots, y_m \leftarrow \tau_m]$. So, $\mathcal{A}[\forall x_1 \ldots \forall x_m F^M] = 1$. **Q.E.D.**

Since the Herbrand models are countable, The Herbrand Theorem tells us that for every formula F we can find an s-equivalent formula G such that G is satisfiable iff it has a countable model. Now we can ask the question *If G has a countable model, does F have a countable model?*

The answer to this question is yes.

Theorem 2.8.12 (Skolem-Lőwenheim Theorem) *Every FOL F is satisfiable iff it has a countable model.*

Proof: We need only to show that every satisfiable F has a countable model. Assume that F is satisfiable. First we apply the E-elimination algorithm from Section 2.7 and get a formula G, satisfiably equivalent to F, that has no E-predicates. We apply the Skolem Form Algorithm from Section 2.5 to G and obtain a Skolem normal form H that has no equality and is satisfiably equivalent to G. By Herbrand's Theorem, H has a Herbrand model, which is countable.

Now let us look at the steps that transform G into H. The 8 steps the Skolem form algorithm are

1. eliminate redundant quantifiers,
2. eliminate the $\longleftrightarrow$'s,
3. eliminate the $\longrightarrow$'s,
4. rectify the formula,
5. find the prenex form,
6. close the formula,
7. Skolemize the formula, and
8. compute the CNF of the matrix.

The input of each step is the output of the previous step. So,
$G = G_0 \overset{\text{Step1}}{\Longrightarrow} G_1 \overset{\text{Step2}}{\Longrightarrow} G_2 \overset{\text{Step3}}{\Longrightarrow} G_3 \overset{\text{Step4}}{\Longrightarrow} G_4 \overset{\text{Step5}}{\Longrightarrow} G_5 \overset{\text{Step6}}{\Longrightarrow} G_6 \overset{\text{Step7}}{\Longrightarrow} G_7 \overset{\text{Step8}}{\Longrightarrow} G_8 = H$

We will show that for every step i, $G_{i-1} \overset{\text{Stepi}}{\Longrightarrow} G_i$, G_{i-1} has a countable model whenever G_i does.

The steps 1, 2, 3, 4, 5, and 8 are equivalences, so every model of G_i is a model of G_{i-1}. Now let us look at the steps 6 and 7 that are only s-equivalences.

The proof of the Closure Lemma tells us for every model of G_5 is a model of G_6. Moreover, for every model of G_6, we can find a model of G_5, with the same universe, that differ only on the interpretation of a finite number of free variables.

From the proof of the Skolemization Lemma we know that every model of G_7 is a model of G_6. Moreover, for every model of G_6 we can find a model of G_7, with the same universe, that differ from G_7 on the interpretation of a finite number of function symbols.

We put all these facts together and get that $H = G_8$ has a model with universe D iff G has a model with universe D. It follows that G has a countable model. Then, by Corollary 2.7.9, F has a countable model. **Q.E.D.**

Corollary 2.8.13 *Let* **T** *be a FOL theory with a finite number of axioms. Then* **T** *has a model iff it has a countable model.*

Proof: Let $A_1,\ldots,A_n$ be the axioms of **T**. The theory has a model iff the conjunction $A_1 \wedge A_2 \wedge \ldots \wedge A_n$ has a model. By the Skolem-Lőwenheim Theorem, $A_1 \wedge A_2 \wedge \ldots \wedge A_n$ has a model iff it has a countable model. **Q.E.D.**

Now let us use the Herbrand Theorem to find out if the formula is unsatisfiable. For this we need the notion of *Herbrand Expansion.*

Definition 2.8.14 (The Herbrand Expansion) *Let* $F = \forall x_1 \ldots \forall x_n F^M$ *be a Skolem normal form without equality and* $D(F)$ *be the Herbrand universe of* F*. Then* $E(F)$*, the Herbrand expansion of* F*, is the set* $\{F^M[x_1/t_1,\ldots,x_n/t_n] | t_1,\ldots,t_n \in D(F)\}$*.*

So, the Herbrand expansion is obtained by replacing all variables in the matrix of F by terms in the Herbrand universe.

Example 2.8.15 Let $F = \forall x((P(a) \vee P(x)) \wedge \neg P(f(x)))$. The Herbrand universe of F is $D(F) = \{a, f(a), f^2(a), \ldots, f^n(a), \ldots\}$, where $f^n(a) = f(f(\ldots f(a)\ldots))$ has n occurrences of f. The Herbrand expansion of F is
$E(F) = \{F^M[x/a], F^M[x/f(a)], \ldots, F^M[x/f^n(a)], \ldots\}$
$= \{(P(a) \vee P(a)) \wedge \neg P(f(a)), (P(a) \vee P(f(a))) \wedge \neg P(f^2(a))), \ldots,$
$(P(a) \vee P(f^n(a))) \wedge \neg P(f^{n+1}(a)), \ldots\}$.

Since the formulas of $E(F)$ contain no variables, they are treated as formulas in propositional logic. The atoms of $E(F)$ are the predicate constants Q and the formulas $P(t_1,\ldots,t_m)$, where P is a predicate symbol of F of arity $m > 0$ and $t_1,\ldots,t_m$ are terms in $D(F)$.

For example, the atomic formulas of the expansion $E(F)$ from the above example are $P(a)$ and all formulas $P(f^n(a))$, where n is any integer greater than 0.

Definition 2.8.16 ($E(F,n)$) *Let* $F = \forall x_1 \ldots \forall x_n F^M$ *be a Skolem normal form without equality and* $D(F)$ *be the Herbrand universe of* F*. Then* $E(F,n)$*, the Herbrand expansion of* F *of height* n*, is the set* $\{F^M[x_1/t_1,\ldots,x_n/t_n] | t_1,\ldots,t_n \in D(F,n)\}$*.*

The set $E(F,n)$ is obtained by replacing the variables of F^M by terms of height less than or equal to n, that is, by terms in $D(F,n)$.

Example 2.8.17 Let us find $E(F,2)$ for the function $F = \forall x \forall y(P(x,f(y)) \wedge (\neg P(f(x),y) \vee \neg P(y,y)))$ from Example 2.8.2. The Herbrand universe of height 2 is $D(F,2) = \{a, f(a), f^2(a)\}$. The Herbrand expansion of height 2 is obtained by replacing the x and the y variables in $F^M = P(x,f(y)) \wedge (\neg P(f(x),y) \vee \neg P(y,y))$ by terms in $D(F,2)$. Since $D(F,2)$ has 3 terms and F has 2 variables, $E(F,2)$ has 3^2 formulas. We can list them by row, by column, or by diagonal.

If we list them by row we get
$E(F,2) = \{F^M[x/a,y/a], F^M[x/a,y/f(a)], F^M[x/a,y/f^2(a)], F^M[x/f(a),y/a],$

$F^M[x/f(a), y/f(a)], F^M[x/f(a), y/f^2(a)], F^M[x/f^2(a), y/a],$
$F^M[x/f^2(a), y/f(a)], F^M[x/f^2(a), y/f^2(a)]\}$
$= \{P(a, f(a)) \wedge (\neg P(f(a), a) \vee \neg P(a, a)),$
$P(a, f^2(a)) \wedge (\neg P(f(a), f(a)) \vee \neg P(f(a), f(a))),$
$P(a, f^3(a)) \wedge (\neg P(f(a), f^2(a)) \vee \neg P(f^2(a), f^2(a))),$
$P(f(a), f(a)) \wedge (\neg P(f^2(a), a) \vee \neg P(a, a)),$
$P(f(a), f^2(a)) \wedge (\neg P(f^2(a), f(a)) \vee \neg P(f(a), f(a))),$
$P(f(a), f^3(a)) \wedge (\neg P(f^2(a), f^2(a)) \vee \neg P(f^2(a), f^2(a))),$
$P(f^2(a), f(a)) \wedge (\neg P(f^3(a), a) \vee \neg P(a, a)),$
$P(f^2(a), f^2(a)) \wedge (\neg P(f^3(a), f(a)) \vee \neg P(f(a), f(a))),$
$P(f^2(a), f^3(a)) \wedge (\neg P(f^3(a), f^2(a)) \vee \neg P(f^2(a), f^2(a)))\}.$

If we list E(F,2) by column or by diagonal, we get the same set, but the order of the elements in the set is different.

Now let us show that F has a model iff the Herbrand expansion of F is satisfiable.

Proposition 2.8.18 *Let $F = \forall x_1 \ldots \forall x_m F^M$ be a Skolem normal form without equality. Then F is satisfiable iff $E(F)$ has a model.*

Proof: Let $F = \forall x_1 \ldots \forall x_m F^M$ be a Skolem normal form without equality. By The Herbrand Theorem, F has a model iff it has a Herbrand model. So, it is sufficient to show that F has a Herbrand model iff $E(F)$ is satisfiable.

$\Longrightarrow$: Assume that $\mathcal{H}$ is a Herbrand model of F. Then for all $t_1,\ldots,t_m$ in $D(F)$,

$1 = \mathcal{H}_{[x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_m]}[F^M]$ by the interpretation of $\forall$

$= \mathcal{H}_{[x_1 \leftarrow \mathcal{H}[t_1], \ldots, x_m \leftarrow \mathcal{H}[t_m]]}[F^M]$ by Lemma 2.8.9

$= \mathcal{H}[F^M[x_1/t_1, \ldots, x_n/t_m]]$ by The Translation Lemma.

So, $\mathcal{H}$ models every formula of $E(F)$. It follows that $\models_{\mathcal{H}} E(F)$.

$\Longleftarrow$: Now let us assume that there is a truth assignment $\mathcal{T}$ that satisfies $E(F)$. We construct a structure $\mathcal{H} =< D(F), \mathcal{H}^f, \mathcal{H}^P, \mathcal{H}^v >$ with the rules below.

(1) If a is a constant in $D(F)$, $a^{\mathcal{H}} = a$.

(2) If g is a function symbol of arity $n > 0$ that occurs in F, $g^{\mathcal{H}}[t_1, \ldots, t_n] = g[\mathcal{H}[t_1], \ldots, \mathcal{H}[t_n]]$.

(3) If P is a predicate constant, $P^{\mathcal{H}} = \mathcal{T}[P]$.

(4) If P is a predicate symbol of arity $n > 0$ that occurs in F, and $t_1,\ldots,t_n$ are terms in $D(F)$, then $P^{\mathcal{H}}[t_1, \ldots, t_n] = 1$ iff $\mathcal{T}[P(t_1, \ldots, t_n)] = 1$.

The first 2 conditions tell us that $\mathcal{H}$ is a Herbrand structure.

The next task is to show that $\mathcal{H}$ is a model of F.

Let $t_1,\ldots,t_m$ be m terms in $D(F)$. We will show, by structural induction on F^M, that for all subformulas I of F^M,

$\mathcal{H}[I[x_1/t_1, \ldots, x_m/t_m]] = \mathcal{T}[I[x_1/t_1, \ldots, x_m/t_m]]$.

We leave the details of the proof as exercise. For Case 1 we use the relations (3) and (4), and for the remaining 3 cases (Cases 3,4,5), we use the induction hypotheses and the truth tables of the connectives $\neg$, $\vee$ and $\wedge$. The cases 5,6,7,8 do not arise since F^M does not contain $\longrightarrow$'s, $\longleftrightarrow$'s and quantifiers. **Q.E.D.**

Now we can apply The Compactness Theorem from Section 1.8 to $E(F)$.

$n = 0$;
while $[E(F, n)$ is satisfiable] do
    $n = n + 1$;

write ("F is unsatisfiable");

Figure 2.43: An algorithm for testing the unsatisfiability of a Skolem form

Proposition 2.8.19 *$E(F)$ is unsatisfiable iff for some n, $E(F, n)$ is unsatisfiable.*

The proof of this proposition is left as exercise.

The next result is a consequence of Propositions 2.8.18 and 2.8.19.

Corollary 2.8.20 *Let F be a Skolem normal from without equality. Then F is unsatisfiable iff there is some n such that $E(F, n)$ is unsatisfiable.*

We use this corollary to generate the algorithm from Figure 2.43. The theorem prover built by Gilmore [] in 1959, is based on this algorithm.

Its correctness is assured by Corollary 2.8.20. However, the algorithm halts only when F is unsatisfiable. If F is satisfiable, then the procedure loops forever.

Exercises

Exercise 2.8.1 *Let S be a infinite set of universally quantified clauses $\forall x_1 \ldots \forall x_n C$ where C does not contain the equality symbol and all its variables are in the set $\{x_1, \ldots, x_n\}$. Show that*

1. *the Herbrand universe of S is countable, and*
2. *S is satisfiable iff it has a Herbrand model.*

Exercise 2.8.2 *Generalize Corollary 2.8.13 to formulas that have an infinite set of axioms.*

Hint: Exercise 2.5.16 tells us that every set of formulas is satisfiably equivalent to a set of S of formulas $\forall x_1 \ldots \forall x_n C$ where all variables of the clause C are in the set $\{x_1, \ldots, x_n\}$. Then eliminate the equality from S, and apply the preceding exercise.

Exercise 2.8.3 *Find the Herbrand universe for the formula $F = \forall x \forall y (R(x, f(y)) \wedge (S(g(x)) \vee \neg R(y, x)) \wedge \neg S(y))$.*

Exercise 2.8.4 *Find $D(F, 2)$ for the formula $F = \forall x \forall y ((P(x, g(y, x)) \vee \neg P(g(y, y), x)) \wedge P(g(x, y), y))$.*

Exercise 2.8.5 *Prove Lemma 2.8.9.*

Exercise 2.8.6 *Complete the proof of Proposition 2.8.18.*

Exercise 2.8.7 *Prove Proposition 2.8.19.*

Exercise 2.8.8 *Modify the algorithm from Figure 2.43 to halt with failure when* $E(F, n+1) = E(F, n)$.

Exercise 2.8.9 *Let* $F = \forall x_1 \forall x_2 \ldots \forall x_n F^M$ *be a Skolem form without* E *and let* $D(F)$ *and* $E(F)$ *be the Herbrand universe and the Herbrand expansion of* F. *Give an algorithm for obtaining* $L(F)$, *the set of atoms of* $E(F)$.

Exercise 2.8.10 *Let* S *be a set of universally closed, equality free clauses. This means that the elements of* S *are formulas* $\forall x_1 \ldots \forall x_m C$ *that have no free variables, no equality predicate, and the matrix* C *is a clause. Show that* S *is unsatisfiable iff for some finite subset* T *of* S *and for some finite integer* n, $E(T, n)$ *is unsatisfiable.*

Exercise 2.8.11 *The algorithm from Figure 2.43 does not always work when the input is an infinite set of Skolem forms* $S = \{G_0, G_1, \ldots, G_n, \}$. *For these sets,* $E(S, n)$ *may be infinite. Modify the algorithm to handle sets of Skolem forms. Assume that there is an algorithm* g *that takes as input a number* $m \in \{0, 1, 2 \ldots\}$ *and outputs the Skolem form* G_m.

Exercise 2.8.12 *Prove the correctness of the algorithm from the preceding exercise. You must show that it stops with success iff* S *is unsatisfiable.*

2.9 FOL Unification

The following two sections present the first order *resolution*. We recall, from Section 1.8, that the resolution is a method for proving the unsatisfibility of a set of clauses. In propositional logic we defined the resolvent of two clauses $C_1 = A \vee C$ and $C_2 = \neg A \vee D$ to be the clause $R = C \vee D$, obtained by removing the atom A from the first clause and its complement $\neg A$ from the second. We said that two clauses C_1 and C_2 are *unifiable* if C_1 contains an atom A and C_2 contains its negation $\neg A$. In first order logic the things get a little more complicated since the atomic formula A can have variables. So, two atomic formulas can be unified without being identical. This section deals with the unification problem for first order logic. It presents an unification algorithm and proves its correctness. The theoretically minded reader will find many interesting concepts throughout this section. The more practically oriented reader should master the unification algorithm and go on to the next section.

Let V be the set of variables $\{x_0, \ldots, x_n, \ldots\}$ and T be the set of terms defined in Section 2.1.

Definition 2.9.1 (substitution) *A substitution is a finite set of pairs* $s = \{y_1/t_1, \ldots, y_n/t_n\}$, *where*
(1) $y_1, \ldots, y_n$ *are variables from* V,
(2) $t_1, \ldots, t_n$ *are terms in* T,

(3) for all pairs $y_i/t_i, y_j/t_j \in s$, $y_i = y_j$ *implies* $t_i = t_j$, *and*
(4) for all pairs y_i/t_i *in* s, $y_i \neq t_i$.

Since s is a set of ordered pairs, it is a relation. Its domain is the set of variables $\{y_1, \ldots, y_n\}$, and its range is the set of terms $\{t_1, \ldots, t_n\}$. The third condition tells us that s does not contain 2 pairs, x/τ_1 and x/τ_2 with $\tau_1 \neq \tau_2$. This means that s is a function. Then, the terms $t_1, \ldots, t_n$ are, respectively, the values of s at $y_1, \ldots, t_n$.

So, a substitution is a function s from a finite subset of V into T that does not contain any identity pairs x/x.

Examples 2.9.2 1. The set $s_1 = \{x/f(x,y), y/a, z/u\}$ is a substitution because it is a function from $\{x, y, z\}$ into T and does not contain any identity pairs.

2. The relation $s_2 = \{x/g(x), y/y, z/g(a), x/g(x)\}$ is not a substitution because it contains the identity pair y/y.

3. The relation $s_3 = \{x/y, x/z, y/u\}$ is not a substitution because it is not a function; the variable x has two values, y and z.

4. The relation $s_4 = \{x_0/x_1, x_1/x_2, \ldots, x_n/x_{n+1}, \ldots\}$ is not a substitution because its domain $\{x_0, x_1, \ldots, x_n, \ldots,\}$ is infinite.

5. The function $s_5 = \{a/x, y/f(x)\}$ is not a substitution because its domain contains an element, a, that is not a variable.

Definition 2.9.3 (empty, relabeling, ground, away substitutions) *The substitution* $\{\}$ *is called the empty substitution, or the identity.*

A substitution $s = \{y_1/t_1, \ldots, y_n/t_n\}$ *is a relabeling (rename) when its range contains only variables and* s *is a one-to-one function.*

A substitution $s = \{y_1/t_1, \ldots, y_n/t_n\}$ *is a ground substitution when the terms* $t_1, \ldots, t_n$ *do not contain variables.*

A substitution $s = \{y_1/t_1, \ldots, y_n/t_n\}$ *is away from a set of variables* U, *when none of the terms* $t_1, \ldots, t_n$ *have variables in* U.

Example 2.9.4 1. The substitution $s_1 = \{x/y, y/z, u/v\}$ is a relabeling because the range contains only variables and the values at x, y, z are distinct.

2. The set $s_2 = \{x/f(a,b), y/a, z/g(b)\}$ is a ground substitution because the terms $f(a,b), a, g(b)$ have no variables.

3. The substitution $s_3 = \{x/g(u), y/f(u,v), z/f(v,w)\}$ is away from $U = \{x, y, z\}$ because the terms $g(u), f(u,v), f(v,w)$ do not have variables in U.

We recall that the substitution $\{y_1/t_1, \ldots, y_n/t_n\}$ is a function from $\{y_1, \ldots, y_n\}$ to the set of terms. For our purposes, it is useful to extend the function s to the set V. So, we define the function below.

$$s^*[x] = \begin{cases} t_i & \text{if for some } i,\ 1 \leq i \leq n,\ x = y_i \\ x & \text{otherwise} \end{cases}$$

Now we can characterize the extensions.

Proposition 2.9.5 *A function $\sigma : V \longrightarrow T$ is a substitution extension iff $\sigma[x] \neq x$ for a finite number of variables.*

Proof: $\Longrightarrow$: If $s = \{y_1/t_1, \ldots, y_n/t_n\}$, then $s^*[x] \neq x$ for $x = y_1$, $x = y_2$, $\ldots x = y_n$. So, $s[x] \neq x$ for a finite number of variables.

$\Longleftarrow$: Now let $\sigma : V \longrightarrow T$ be such that $\sigma[x] \neq x$ for a finite number of variables. Let $y_1, \ldots, y_n$ be those variables and define $s = \{y_1/\sigma[y_1], \ldots, y_n/\sigma[y_n]\}$. Since s is a subset of σ, it is a function. Moreover, we selected only the pairs $x/\sigma[x]$ with $x \neq \sigma[x]$, so s does not contain any identity pairs. So, s is a substitution. We compute the function s^*.

$$s^*[x] = \begin{cases} \sigma[y_i] & \text{if for some } i,\ 1 \leq i \leq n,\ x = y_i \\ x & \text{otherwise} \end{cases}$$

Since $\sigma[x] = x$ for $x \notin \{y_1, \ldots, y_n\}$,

$$s^*[x] = \begin{cases} \sigma[y_i] & \text{if for some } i,\ 1 \leq i \leq n,\ x = y_i \\ \sigma[x] & \text{otherwise} \end{cases}$$

The last formula states that the function s^* is equal to σ. **Q.E.D.**

It is easy to prove that different substitutions have different extensions.

Proposition 2.9.6 *Let s_1, s_2 be two substitutions. Then, $s_1 \neq s_2$ implies ${s_1}^* \neq {s_2}^*$.*

Proof: Let us assume that $s_1 \neq s_2$. Then there is a pair x/t that belongs to one substitution but not to the other. Let us assume, without loss of generality (wlog), that x/t is in s_1, but not in s_2. Then ${s_1}^*[x] = t$. We have 2 subcases.

Subcase 1: x is in the domain of s_2. Then, there is a pair $x/\tau \in s_2$ and ${s_2}^*[x] = \tau$. Since $x/t \notin s_2$, $\tau \neq t$, so ${s_1}^*[x] \neq {s_2}^*[x]$.

Subcase 2. x is not in the domain of s_2. Then ${s_2}^*[x] = x$. We know that $t \neq x$ because no substitution contains a pair of identical variables, so ${s_1}^*[x] \neq {s_2}^*[x]$.

In both subcases, ${s_1}^*[x] \neq {s_2}^*[x]$, so ${s_1}^* \neq {s_2}^*$. **Q.E.D.**

[the support of an extension]Let $\sigma : V \longrightarrow T$ be a substitution extension. The substitution s that satisfies the equality $s^* = \sigma$ is called the support of σ.

We can go a step further and extend $s = y_1/t_1, \ldots, y_n/t_n$ to the set of terms T. The extension $\bar{s}$ is defined as

$\bar{s}[t] = t[y_1/t_1, \ldots, y_n/t_n]$,

where the term $\bar{s}[t]$ is computed by simultaneously replacing all occurrences of y_1 by t_1, of y_2 by t_2,..., of y_n by t_n.

Example 2.9.7 Let $s = \{x/f(x,y), y/g(z)\}$ and $t = f(x, h(y, z, f(u, a)))$. Then

$\bar{s}[t] = \bar{s}[f(x, h(y, z, f(u, a)))]$

$= f(x, h(y, z, f(u, a)))[x/f(x, y), y/g(z)]$

$= f(f(x, y), h(g(z), z, f(u, a)))$.

Notation 2.9.8 Since we have already used the notation $[y_1/t_1, \ldots, y_n/t_n]$ for substitutions, we will also write $s = [y_1/t_1, \ldots, y_n/t_n]$ instead of the set theoretical notation $s = \{y_1/t_1, \ldots, y_n/t_n\}$.

Observation 2.9.9 It is easy to check that $\bar{s}$ is an extension of s^*. Let $s = [y_1/t_1, \ldots, y_n/t_n]$.

Then $\bar{s}[y_i] = y_i[y_1/t_1, \ldots, y_n/t_n] = t_i = s^*[y_i]$.

If $x \notin \{y_1, \ldots, y_n\}$, then $\bar{s}[x] = x[y_1/t_1, \ldots, y_n/t_n] = x = s^*[x]$.

So, the restriction of $\bar{s}$ to V is s^*.

Next, we define the composition of substitutions.

Let s_1, s_2 be two substitutions and $\sigma : V \longrightarrow T$, be the function defined by $\sigma[y] = \overline{s_2}[{s_1}^*[y]]$.

Proposition 2.9.10 *The function σ defined above is a substitution extension.*

Proof: Let $s_1 = \{y_1/t_1, \ldots, y_n/t_n\}$ and $s_1 = \{z_1/\tau_1, \ldots, z_m/\tau_m\}$ be the two substitutions. Let $x \notin \{y_1, \ldots, y_n, z_1, \ldots, z_m\}$. Then

$\sigma[x] = \overline{s_2}[{s_1}^*[x]]$

$= \overline{s_2}[x]$ because $x \notin \{y_1, \ldots, y_n\}$

$= s_2[x]$ since x is a variable, $\overline{s_2}[x] = s_2[x]$

$= x$ because $x \notin \{z_1, \ldots, z_m\}$.

So, $\sigma[x] = x$ for all but a finite number of variables, so it is a substitution extension, by Proposition 2.9.5. **Q.E.D.**

Since σ is a substitution extension, it has a unique support, by Proposition 2.9.6. We call that support *the composition of s_1 and s_2* and write it $s_2 \diamond s_1$.

Example 2.9.11 Let us find the composition $s_2 \diamond s_1$ for $s_1 = [x/y, y/f(z), u/a]$ and $s_2 = [x/h(u, x), y/x, v/w]$. The proof of Proposition 2.9.10 tells us that $(s_2 \diamond s_1)^*[x] = x$ for all x that are not in $domain(s_1) \cup domain(s_2)$. So, we need to compute $(s_2 \diamond s_1)^*$ for $domain(s_1) \cup domain(s_2) = \{x, y, u\} \cup \{x, y, v\} = \{x, y, u, v\}$. We use the formula from the definition of composition.

$(s_2 \diamond s_1)^*[x] = \overline{s_2}[{s_1}^*[x]]$ by the definition of composition

$= \overline{s_2}[y]$ because $x/y \in s_1$

$= y[x/h(u, x), y/x, v/w]$ by the definition of $\overline{s_2}$

$= x$ because $y/x \in s_2$

$(s_2 \diamond s_1)^*[y] = \overline{s_2}[{s_1}^*[y]]$ by the definition of composition

$= \overline{s_2}[f(z)]$ because $x/f(z) \in s_1$

$= f(z)[x/h(u, x), y/x, v/w]$ by the definition of $\overline{s_2}$

$= f(z)$ since z is not in the domain of s_2

$(s_2 \diamond s_1)^*[u] = \overline{s_2}[{s_1}^*[u]]$ by the definition of composition

$= \overline{s_2}[a]$ because $x/a \in s_1$

$= a[x/h(u, x), y/x, v/w]$ by the definition of $\overline{s_2}$

$= a$ the constants remain unchanged

$(s_2 \diamond s_1)^*[v] = \overline{s_2}[{s_1}^*[v]]$

$= \overline{s_2}[v]$ because v is not in the domain of s_1

$= v[x/h(u,x), y/x, v/w]$ by the definition of $\overline{s_2}$
$= w$ since $v/w \in s_2$

Now, $s_2 \diamond s_1$ contains all non-identical pairs of $(s_2 \diamond s_1)^*$. We eliminate x/x and are left with $[y/f(z), u/a, v/w]$.

Observation 2.9.12 We must be careful to distinguish the compositions $s_2 \circ s_2$ and $s_2 \diamond s_1$. The first is the composition of s_1 and s_2 as functions and the second is obtained by applying s_2 to the range of s_1. In many cases the two compositions yield different results, as shown below.

Let $s_1 = [x/y, z/x]$ and $s_2 = [x/u, v/w]$. Then $s_2 \diamond s_1 = [z/u]$ while $s_2 \diamond s_1 = [x/y, z/u, v/w]$.

Proposition 2.9.13 relates the composition of substitutions to the composition of functions.

Proposition 2.9.13 $\overline{s_2 \diamond s_1} = \overline{s_2} \circ \overline{s_1}$.

Proof: By structural induction on terms.
Case 1. t is a variable, say y.
Then
$\overline{s_2 \diamond s_1}[y] = (s_2 \diamond s_1)^*[y]$ because y is variable
$= \overline{s_2}[{s_1}^*[y]]$ by the definition of substitution composition
$= \overline{s_2}[\overline{s_1}[y]]$ because $\overline{s_1}$ is an extension of ${s_1}^*$
$= (\overline{s_2} \circ \overline{s_1})[y]$ definition of function composition
Case 2. t is a constant, say a.
Then
$\overline{s_2 \diamond s_1}[a] = a$ and
$(\overline{s_2} \circ \overline{s_1}] = \overline{s_2}[\overline{s_1}[a]]$
$= \overline{s_2}[a] = a$
because the substitutions do not modify constants.
Case 3. $t = f(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms.
Then
$\overline{s_2 \diamond s_1}[f(t_1, \ldots, t_n)] = f(\overline{s_2 \diamond s_1}[t_1], \ldots, \overline{s_2 \diamond s_1}[t_n])$ definition of the $^-$ extension
$= f((\overline{s_2} \circ \overline{s_1})[t_1], \ldots, (\overline{s_2} \circ \overline{s_1})[t_n])$ by induction hypothesis
$= f((\overline{s_2}[\overline{s_1}[t_1]], \ldots, \overline{s_2}[\overline{s_1}[t_n]])$ definition of composition
$= \overline{s_2}[f(\overline{s_1}[t_1], \ldots, \overline{s_1}[t_n])]$ by the definition of substitution
$= \overline{s_2}[\overline{s_1}[f(t_1, \ldots, t_n)]]$ by the definition of substitution
$= (\overline{s_2} \circ \overline{s_1})[f(t_1, \ldots, t_n)]$ by the definition of the composition
Q.E.D.

We will use Proposition 2.9.13 to show that the composition of substitutions is associative.

Proposition 2.9.14 *The composition of substitutions is associative, i.e.*
$(s_3 \diamond s_2) \diamond s_1 = s_3 \diamond (s_2 \diamond s_1)$.

Proof: Let y be a variable.

$((s_3 \diamond s_2) \diamond s_1)^*[y] = (\overline{(s_3 \diamond s_2)}[s_1^*[y]]$ by the definition of substitution composition

$= (\overline{s_3} \circ \overline{s_2})[s_1^*[y]]$ by Proposition 2.9.13

$= \overline{s_3}[\overline{s_2}[s_1^*[y]]]$. by the definition of function composition

At the same time,

$(s_3 \diamond (s_2 \diamond s_1))^*[y] = \overline{s_3}[(s_2 \diamond s_1)^*[y]]$ from the definition of substitution composition

$= \overline{s_3}[\overline{s_2}[s_1^*[y]]]$ by the definition of substitution composition

So, $((s_3 \diamond s_2) \diamond s_1)^* = (s_3 \diamond (s_2 \diamond s_1))^*$.

By Proposition 2.9.6, the above equality yields $(s_3 \diamond s_2) \diamond s_1 = s_3 \diamond (s_2 \diamond s_1)$.
Q.E.D.

The associativity allows us to skip the parentheses around the operands of the expressions that contain the composition operator $\diamond$.

Notation 2.9.15 In order to simplify the notation, we will always omit the star on top of the substitution. Many times we will also omit the bar on top of the substitutions. We will use them only when we want to emphasize the application of a proposition or to add clarity to the presentation. So, we will write $s[t]$ instead of $\overline{s}[t]$, $s[y]$ instead of $s^*[y]$.

Definition 2.9.16 (unifier) *Let* $S = \{P(t_1^1, \ldots, t_n^1), P(t_1^2, \ldots, t_n^2), \ldots, P(t_1^m, \ldots, t_n^m)\}$ *be a set of atomic formulas. A unifier of* S *is a substitution* s *such that*

$s[P(t_1^1, \ldots, t_n^1)] = s[P(t_1^2, \ldots, t_n^2)] = \ldots = s[P(t_1^m, \ldots, t_n^m)]$.

This definition tells us that s is a unifier of $S = \{P(t_1^1, \ldots, t_n^1), P(t_1^2, \ldots, t_n^2), \ldots, P(t_1^m, \ldots, t_n^m)\}$ if and only if

$s[t_1^1] = s[t_1^2] = \ldots = s[t_1^m]$,

$s[t_2^1] = s[t_2^2] = \ldots = s[t_2^m]$,

$\ldots$,

$s[t_n^1] = s[t_n^2] = \ldots = s[t_n^m]$.

So, $s[S]$ contains only one formula, $P(s[t_1^1], \ldots, s[t_n^1])$. This formula is also called a unifier of S. When the meaning of the word *unifier* is not clear from the context, we use *unifying substitution* or *unifying term (formula)*. If S has a unifying substitution we say that the set S is *unifiable*.

Examples 2.9.17 1. The substitution $s = [x/g(u), y/g(u), z/f(g(u)), v/g(u)]$ is a unifier for $S = \{P(x, f(y)), P(y, z), P(g(u), f(v))\}$ because it makes all 3 formulas identical.

$s[P(x, f(y))] = P(s[x], f(s[y])) = P(g(u), f(g(u)))$

$s[P(y, z)] = P(s[y], s[z]) = P(g(u), f(g(u)))$,

$s[P(g(u), f(v))] = P(g(s[u]), f(s[v])) = P(g(u), f(g(u)))$

2. The substitution $\sigma = [x/g(a), y/g(a), z/f(g(a)), v/g(a)]$ is also a unifier for the set S, since

$s[P(x, f(y)] = s[P(y, z)] = s[P(g(u), f(v))] = P(g(a), f(g(a)))$.

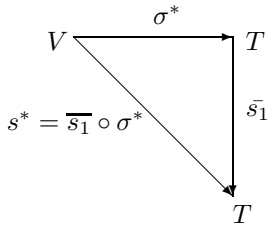


Figure 2.44: The factoring property

Definition 2.9.18 (most general unifier, factoring property) *A most general unifier, abbreviated mgu, of*
$S = \{P(t_1^1, \ldots, t_n^1), P(t_1^2, \ldots, t_n^2), \ldots, P(t_1^m, \ldots, t_n^m)\}$
is a unifier σ of S such that all unifiers of S factor through σ.
We say that s factors through σ when there is a substitution s_1 that satisfies the equality $s = s_1 \diamond \sigma$.

The factoring property is shown in Figure 2.44.

Definition 2.9.19 ($Var[S]$) *1. If t is a term, then $Var[t]$ is the set of variables that occur in t.*
2. If S is a set of terms, $Var[S]$ is the set of variables that occur in at least one term of S.
3. If F is a formula, $Var[F]$ is the set of variables that occur in the terms of the formula.
4. If S is a set of formulas, $Var[S]$ is the set of variables that occur in the elements of S.

Examples 2.9.20 1. $Var[f(x,y)] = \{x,y\}$.
2. $Var[\{g(u,a), x, h(b,z)\}] = \{u,x,z\}$.
3. $Var[P(f(x,y), b, g(z,c))] = \{x,y,z\}$.
4. $Var[\{Q(f(x,y),a), \neg R(z), E(x,f(u))\}] = \{x,y,z,u\}$.

Let us show that whenever σ_1 and σ_2 are two mgu's of the same set, they are relabelings of each other. This means that there are renames s_1 and s_2 such that $\sigma_1 = s_1 \diamond \sigma_2$ and $\sigma_2 = s_2 \diamond \sigma_1$.

Lemma 2.9.21 *Let t and τ be two terms and s, σ be two substitutions such that $t = s[\tau]$ and $\tau = \sigma[t]$. Then, there is a relabeling $r = [x_1/\alpha_1, \ldots, x_n/\alpha_n]$ that satisfies 1-4.*
1. $r \subseteq \sigma$
2. $r^{-1} \subseteq s$, where $r^{-1} = [\alpha_1/x_1, \ldots \alpha_n/x_n]$,
3. $t = r[\tau]$
4. $\tau = r^{-1}[t]$.

Proof: First of all, let us represent t as a tree, and let $x_1, \ldots, x_n$ be the variables of t that are also in the domain of σ. Let $u_1, \ldots, u_m$, $m \geq n$, be the

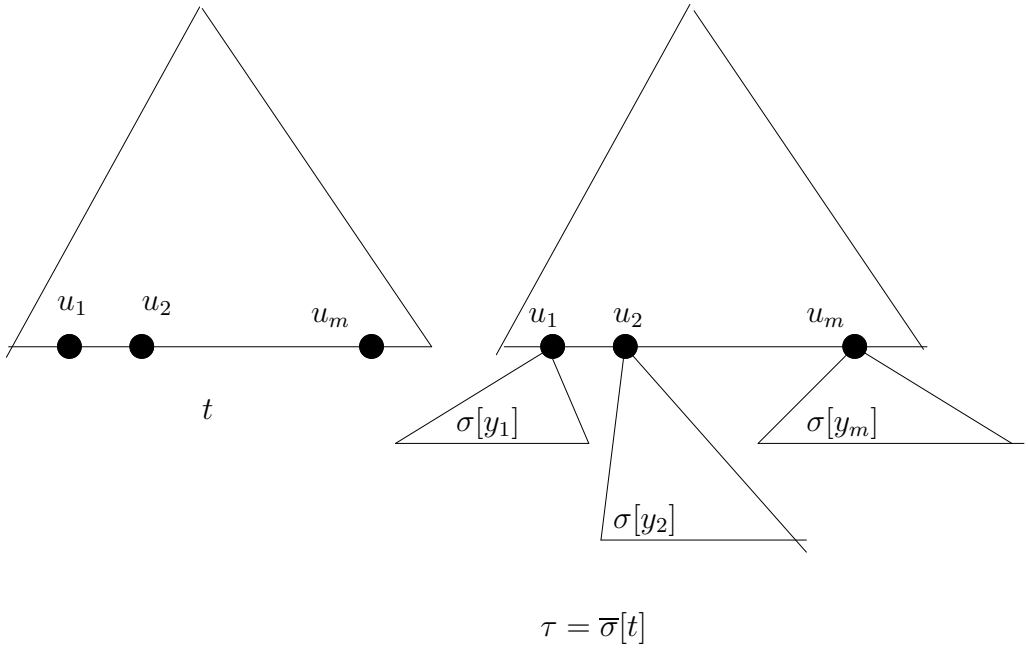


Figure 2.45: The tree $\sigma[t]$ from Lemma 2.9.21

Dewey addresses where these variables are found. Since $u_1, \ldots, u_m$ are labeled with variables, they are leaves of t. Let $y_1, \ldots, y_m$ be the labels of $u_1, \ldots, u_m$. Then $\tau = \sigma[t]$ is the tree from Figure 2.45. From the picture we see that

(1) the address set of t is a subset of the address set of τ, and

(2) the labels of all addresses of t, save $u_1, \ldots, u_m$, are the same in t as in τ.

We do the same thing for τ. Let $z_1, \ldots, z_p$ be the variables of τ that occur in the domain of s and $v_1, \ldots, v_q$ the addresses where they are found. When we do the substitution, each v_i is replaced by the tree $s[w_i]$, w_i being the label of v_i.

The tree $t = \overline{s}[\tau]$ is shown in Figure 2.46. From the equality $t = \overline{s}[\tau]$ we deduce (3) and (4).

(3) the address set of τ is a subset of the address set of t, and

(4) the labels of all addresses of τ, save the leaves $v_1, \ldots, v_q$, are the same in τ and t.

From (1) and (3) we get that t and τ have the same set of addresses. If $[x_1/\alpha_1, \ldots, x_n/\alpha_n]$ is the restriction of σ to the variables of t, then the terms $\alpha_1, \ldots, \alpha_n$, must be either variables or constants. Otherwise, some of the α's have height greater than 0 and the tree τ has more addresses than t.

Let us assume that α_i is a constant, say a, and that x_i occurs at address u_j of t. Then the labels at address u_j are x_i in t and a in τ. But then the equality $t = \overline{s}[\tau]$ cannot hold since no substitution can change a constant into a variable.

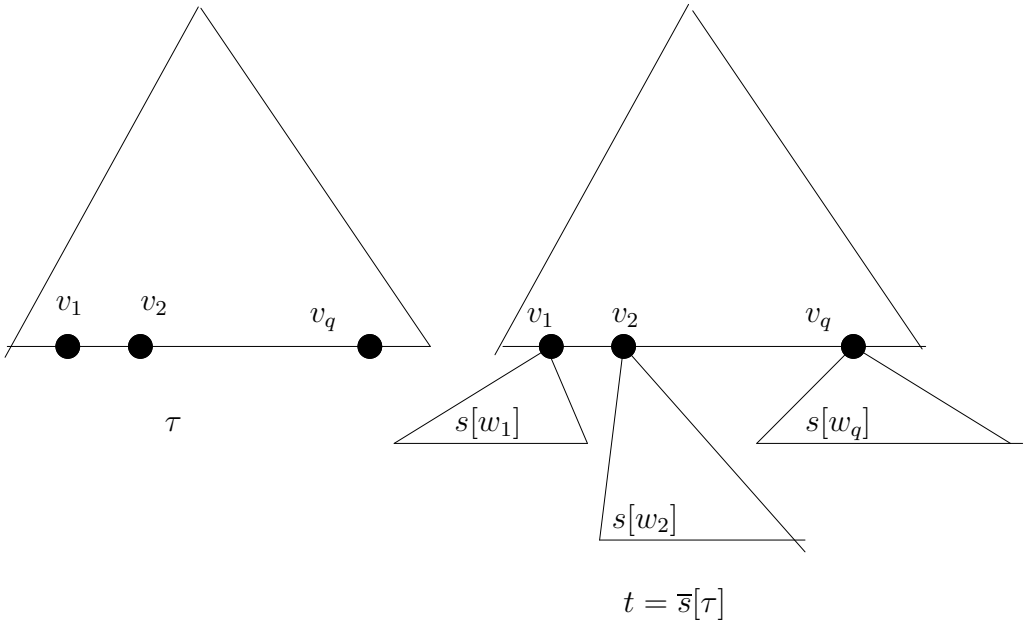


Figure 2.46: The tree $\sigma[t]$ from Lemma 2.9.21

So, all the α_i's are variables. It remains to show that $x_i \neq x_j$ requires that $\alpha_i \neq \alpha_j$. Again we assume that for some i and j, $x_i \neq x_j$ and $\alpha_i = \alpha_j$. Let u, v be the addresses of x_i, respectively x_j in t. Since τ is the result of the substitution, τ has the same label at u and v while t has different labels. But then no substitution s can transform τ into t, since all will give the same label to both u and v. Contradiction!

So, σ restricted to t is a relabeling $r = [x_1/\alpha_1, \ldots, x_n/\alpha_n]$. Now let A be an address where the labels of t and τ differ. Since $\overline{r}[t] = \tau$, t must be labeled with one of the variables $x_1, \ldots, x_n$, say x_i. Then τ has label α_i at A. When we apply the relabeling $r^{-1} = [\alpha_1/x_1, \ldots, \alpha_n/x_n]$ to τ, the label at A becomes x_i. So, $\overline{r^{-1}}[\tau] = t$. **Q.E.D.**

Lemma 2.9.22 *Let $\sigma_1, \sigma_2, s_1, s_2$ be 4 substitutions such that*
$dom(s_1) \subseteq dom(\sigma_1) \cup dom(\sigma_2)$ and
$dom(s_2) \subseteq dom(\sigma_1) \cup dom(\sigma_2)$.
Let $\{x_1, \ldots, x_n\} = dom(\sigma_1) \cup dom(\sigma_2)$ and let f be a function of arity n. Let $t = f(x_1, \ldots, x_n)$, $t_1 = \overline{\sigma_1}[t]$ and $t_2 = \overline{\sigma_2}[t]$.
Then
$\sigma_1 = s_1 \diamond \sigma_2$ and $\sigma_2 = s_2 \diamond \sigma_1$ iff
$t_1 = \overline{s_1}[t_2]$ and $t_2 = \overline{s_2}[t_1]$

Proof: Let $\{x_1, \ldots, x_n\} = dom(\sigma_1) \cup dom(\sigma_2)$ and let f be a function of arity n. Let $t = f(x_1, \ldots, x_n)$, $t_1 = \overline{\sigma_1}[t]$ and $t_2 = \overline{\sigma_2}[t]$.

$\Longrightarrow$: Assume that $\sigma_1 = s_1 \diamond \sigma_2$ and $\sigma_2 = s_2 \diamond \sigma_1$. Then

$\overline{s_1}[t_2]$

$= \overline{s_1}[\overline{\sigma_2}[t]]$ because $t_2 = \overline{\sigma_2}[t]$

$= (\overline{s_1} \circ \overline{\sigma_2})[t]$ by the composition of substitutions

$= \overline{s_1 \diamond \sigma_2}[t]$ the composition of the extensions is the extension of the composition

$= \overline{\sigma_1}[t]$ since $\sigma_1 = s_1 \diamond \sigma_2$

$= t_1$ by the definition of t_1.

In a similar fashion we show that $t_2 = \overline{s_2}[t_1]$.

$\Longrightarrow$: Assume that $t_1 = \overline{s_1}[t_2]$ and $t_2 = \overline{s_2}[t_1]$. We will show that $\sigma_1 = s_1 \diamond \sigma_2$. If $y \notin dom(\sigma_1) \cup dom(\sigma_2)$,

$\sigma_1[y] = y$

and

$(s_1 \diamond \sigma_2)[y] = \overline{s_1}[\sigma_2[y]]$ definition of composition

$= \overline{s_1}[y]$ because y is not in the domain of σ_2

$= s_1[y]$ since y is a variable

$= y$ y is not in the domain of s_1.

So, when $y \notin dom(\sigma_1) \cup dom(\sigma_2)$, $\sigma_1[y] = (s_1 \diamond \sigma_2)[y]$.

Now let us assume that $y \in dom(\sigma_1) \cup dom(\sigma_2)$. Then y is a leaf of the tree $t = f(x_1, \ldots, x_n)$. Let us assume that y occurs at address $i-1$, i.e. $y = x_i$. Since $t_1 = \overline{s_1}[t_2]$ they have the same subtree at address $i-1$. But $t_1[i-1] = \sigma_1[x_i]$ while

$(\overline{s_1}[t_2])[i-1] = \overline{s_1}[[t_2][i-1]]$

$= \overline{s_1}[\sigma_2[f(x_1, \ldots, x_n)][i-1]]$

$= \overline{s_1}[[\sigma_2[x_i]]$ t_2 at address $i-1$ is $\sigma_2[x_i]$

$= (s_1 \diamond \sigma_2)[x_i]$ by the definition of compositions of substitutions

So, $\sigma_1[y] = (s_1 \diamond \sigma_2)[y]$.

In both cases, $\sigma_1[y] = (s_1 \diamond \sigma_2)[y]$. So,

$\sigma_1 = s_1 \diamond \sigma_2$. **Q.E.D.**

Theorem 2.9.23 *If $\sigma_1 = s_1 \diamond \sigma_2$ and $\sigma_1 = s_1 \diamond \sigma_2$, there is a relabeling r such that $\sigma_1 = r \diamond \sigma_2$ and $\sigma_2 = r^{-1} \diamond \sigma_1$.*

Proof: Since $\sigma_1 = s_1 \diamond \sigma_2$, $dom(s_1) \subseteq dom(\sigma_1) \cup dom(\sigma_2)$. At the same time, $\sigma_2 = s_2 \diamond \sigma_1$, implies $dom(s_2) \subseteq dom(\sigma_1) \cup dom(\sigma_2)$. Let $\{x_1, \ldots, x_n\}$ be the variables of $dom(\sigma_1) \cup dom(\sigma_2)$. We choose f to be a function of arity n and set $t = f(x_1, \ldots, x_n)$, $t_1 = \overline{\sigma_1}[t]$ and $t_2 = \overline{\sigma_2}[t]$.

By Lemma 2.9.22, $t_1 = \overline{s_1}[t_2]$ and $t_2 = \overline{s_2}[t_1]$.

By Lemma 2.9.21, there is a relabeling $r \subseteq s_1$ such that $r^{-1} \subseteq s_2$, $t_1 = \overline{r}[t_2]$, and $t_1 = \overline{r^{-1}}[t_2]$ Since r is subset of s_1 and r^{-1} a subset of s_2, $dom(r) \subseteq dom(\sigma_1) \cup dom(\sigma_2)$ and $dom(r^{-1}) \subseteq dom(\sigma_1) \cup dom(\sigma_2)$. This means that the substitutions $\sigma_1, \sigma_2, r, r^{-1}$ satisfy the conditions of Lemma 2.9.22. Then,

$\sigma_1 = r \diamond \sigma_2$ and $\sigma_1 = r^{-1} \diamond \sigma_2$.**Q.E.D.**

Corollary 2.9.24 *The mgu's are unique up to a relabeling.*

Proof: If both σ_1 and σ_2 are mgu's they factor though each other, i.e. $\sigma_1 = s_1 \diamond \sigma_2$ and $\sigma_2 = s_2 \diamond \sigma_1$. By the preceding theorem there are relabelings r_1 and r_2 such that $\sigma_1 = r_1 \diamond \sigma_2$ and $\sigma_2 = 2_2 \diamond \sigma_1$. **Q.E.D.**

We will frequently talk about *the mgu* instead of *an mgu* since the differences between the mgu are just relabelings.

Before we continue our discussion about unifiers let us introduce the concept of *commutative diagrams*, figures that will occur in the following sections. Commutative diagrams are labeled directed graphs. The vertices are sets and the arrows are functions. If two paths start at the same vertex and end in the same vertex, then the compositions of the functions along the two paths are equal. In Figure 2.44 we have two paths from V to T, $V \xrightarrow{s} T$ and $V \xrightarrow{\sigma} T \xrightarrow{\overline{s_1}} T$. The compositions of the functions along the two paths are equal, i.e. $s = \overline{s_1} \circ \sigma$. We must remember that the order of the functions in the composition is the reverse of their occurrence in the path. So, if the path is

$A_0 \xrightarrow{f_1} A_1 \xrightarrow{f_2} A_2 \ldots A_{n-2} \xrightarrow{f_{n-1}} A_{n-1} \xrightarrow{f_n} A_n$

we must write $f_n \circ f_{n-1} \circ \ldots \circ f_2 \circ f_1$.

Now let us return to our discusion of mgu's. We will give an algorithm for finding the mgu of a set of atomic formulas. For this we need the notion of *disagreement set*. Before we do that, let us recall, from Exercise 2.2.5, that the leftmost position where two terms differ is the beginning of two diferent subterms.

Definition 2.9.25 (disagreement set) *Let $S = \{A_1, \ldots, A_m\}$, be the set of atoms $A_1 = P(t_1^1, \ldots, t_n^1)$, $A_2 = P(t_1^2, \ldots, t_n^2)$,..., $A_m = P(t_1^m, \ldots, t_n^m)$. The disagreement set of S is the set of the terms of A_1,...,A_m, that begin at the leftmost position at which A_1,...,A_n have different symbols.*

Example 2.9.26 Let $S = \{P(x, f(x), y), P(x, u, y), P(x, f(g(v)), z)\}$. The leftmost position where the strings $P(x, f(x), y)$, $P(x, u, y)$, $P(x, f(g(v)), z)$ differ is the 5-th. So, we take the terms of $P(x, f(x), y)$, $P(x, u, y)$, $P(x, f(g(v)), z)$ that start at position 5. We get the set $D = \{f(x), u, f(g(v))\}$.

The *unification* algorithm shown in Figure 2.47 computes the mgu of a set S, of atomic formulas that have the same predicate symbol. Before we go on to prove the correctness of The Unification Algorithm, let us see how it works.

Example 2.9.27 Let us find the mgu for $S = \{P(f(x, y), y, g(z)), P(u, h(v), g(v)),$
$P(f(x, y), h(v), w)\}$. First we set $\sigma_0 = []$ and find the disagreement set D_0 of $\sigma_0[S]$. The empty substitution [] leaves S unchanged , so $\sigma_0[S] = \{P(f(x, y), y, g(z)), P(u, h(v), g(v)), P(f(x, y), h(v), w)\}$.

The smallest position where the 3 formulas of $\sigma_0[S]$ differ is 3. So, the disagreement set D_0 is the set of terms that start at this position. We get $D_0 = \{f(x, y), u, f(x, y)\}$.

The set D_0 is not empty, so we execute the loop. D_0 contains a variable and a term that does not have that variable as a subterm, so we define the

$n = 0$; $\sigma_n = []$; compute D_n, the disagreement set of $\sigma_n[S]$;
while $[D_n \neq \phi]$
{
    if [D_n contains a variable x and a term t that does not have x as a proper subterm] then
    {
        $n = n + 1$; $\sigma_n = [x/t] \diamond \sigma_{n-1}$;
        find D_n, the disagreement set of $\sigma_n[S]$;
    }
    else
        exit with failure;
}
mgu $= \sigma_n$;

Figure 2.47: The Unification Algorithm

substitution $\sigma_1 = [u/f(x,y)] \diamond \sigma_0 = [u/f(x,y)] \diamond [] = [u/f(x,y)]$. Then we compute the set $\sigma_1[S]$ as

$[u/f(x,y)][\sigma_0[S]]$
$= [u/f(x,y)][\{P(f(x,y),y,g(z)), P(u,h(v),g(v)), P(f(x,y),h(v),w)\}]$
$= \{P(f(x,y),y,g(z))[u/f(x,y)], P(u,h(v),g(v))[u/f(x,y)],$
$P(f(x,y),h(v),w)[u/f(x,y)]\}$
$= \{P(f(x,y),y,g(z)), P(f(x,y),h(v),g(v)), P(f(x,y),h(v),w)\}$.

The leftmost position where the formulas of $\sigma_1[S]$ differ is 10. We get the disagreement set $D_1 = \{y, h(v)\}$.

We go to the loop repetition test. D_1 is not empty, so we execute the loop body. D_1 contains both the variable y and the term $g(v)$ that does not have y as a subterm. So, we set $\sigma_2 = [y/h(v)] \diamond \sigma_1$, and we compute $\sigma_2[S]$.

$\sigma_2[S] = [y/h(v)][\sigma_1[S]]$
$= [y/h(v)][\{P(f(x,y),y,g(z)), P(f(x,y),h(v),g(v)), P(f(x,y),h(v),w)\}]$
$= \{P(f(x,y),y,g(z))[y/h(v)], P(f(x,y),h(v),g(v))[y/h(v)],$
$P(f(x,y),h(v),w)[y/h(v)]\}$
$= \{P(f(x,h(v)),h(v),g(z)), P(f(x,g(v)),h(v),g(v)), P(f(x,h(v)),h(v),w)\}$

The formulas of $\sigma_2[S]$ differ at position 18, and we get the disagreement set $D_2 = \{g(z), g(v), w\}$.

We go back to the loop test. Since D_2 is not empty, we repeat the loop. In D_2 we encounter a variable, w, and two terms that do not contain the variable. So, we choose one of the terms $g(z)$ and $g(v)$ to be part of the substitution σ_3. The next theorem will show that it does not matter which one we choose, so let $\sigma_3 = [w/g(z)] \diamond \sigma_2$. We compute the set $\sigma_3[S] = [w/g(z)][\sigma_2[S]]$.

$[w/g(z)][\sigma_2[S]]$
$= \{P(f(x,h(v)),h(v),g(z))[w/g(z)], P(f(x,h(v)),h(v),g(v))[w/g(z)],$
$P(f(x,h(v)),h(v),w)[w/g(z)]\}$
$= \{P(f(x,h(v)),h(v),g(z)), P(f(x,h(v)),h(v),g(v)), P(f(x,h(v)),h(v),g(z))\}$

The disagreement set of $\sigma_3[S]$ is the set of terms that start at position 20, so $D_3 = \{v, z\}$. We go back to the loop test. D_3 is not empty, so we repeat the loop. Since both elements of D_3 are variables, we can choose between the substitutions $[v/z]$ or $[z/v]$. The Unification Theorem will tell us that both choices are valid, i.e. if the set is unifiable they will lead to an mgu. So, let us choose $\sigma_4 = [z/v] \diamond \sigma_3$. We compute $\sigma_4[S] = [z/v][\sigma_3[S]]$ and get

$[z/v]\sigma_3[S]]$
$= \{P(f(x, h(v)), h(v), g(z))[z/v], P(f(x, h(v)), h(v), g(v))[z/v],$
$P(f(x, h(v)), h(v), g(z))[z/v]\}$
$= \{P(f(x, h(v)), h(v), g(v)), P(f(x, h(v)), h(v), g(v)), P(f(x, h(v)), h(v), g(v))\}$
$= \{P(f(x, h(v)), h(v), g(v))\}$.

The disagreement set of $\sigma_4[S]$ is $D_4 = \phi$.

We go back to the loop test. Since D_4 is empty, we exit. There we compute σ_4,

$mgu = \sigma_4 = [z/v] \diamond [w/g(z)] \diamond [y/h(v)] \diamond [u/f(x, y)] = [z/v, w/g(v), y/h(v), u/f(x, h(v))]$.

Now let us give an example of a set of formulas that is not unifiable.

Example 2.9.28 Let us find an mgu for the set $S = \{P(f(x), y), P(z, z), P(u, g(v))\}$. First we initialize σ_0 to [] and find the disagreement set of $\sigma_0[S] = S$. The leftmost where the formulas differ is 3, so D_0 is the set of terms that start at that position. We get $D_0 = \{f(x), z, u\}$. We go to the loop repetition test.

The disagreement set is not empty, so we execute the loop. The set D_0 contains a variable and a term that does not have that variable as a subterm. So, we pick one of the substitutions $[z/u]$, $[u/z]$, $[z/f(x)]$, $[u/f(x)]$. If S is unifiable, all four substitutions will lead to an mgu. So, let us choose $\sigma_1 = [z/f(x)] \diamond \sigma_0$. We compute the set of formulas $\sigma_1[S] = [z/f(x)][\sigma_1[S]$.

$\sigma_1[S] = [z/f(x)]\{P(f(x), y), P(z, z), P(u, g(v))\}$
$= \{P(f(x), y)[z/f(x)], P(z, z)[z/f(x)], P(u, g(v))[z/f(x)]\}$
$= \{P(f(x), y), P(f(x), f(x)), P(u, g(v))\}$

The leftmost position where the formulas of $\sigma_1[S]$ differ is 3. We get the disagreement set $D_1 = \{u, f(x)\}$and we go to the loop repetition test.

Since D_1 is not empty we execute the loop. D_1 contains a variable and a term that does not have that variable as a subterm. So, we define the substitution $\sigma_2 = [u/f(x)] \diamond \sigma_1$ and we compute the set $\sigma_2[S] = [u/f(x)][\sigma_1[S]$.

$\sigma_2[S] = [u/f(x)]\{P(f(x), y), P(f(x), f(x)), P(u, g(v))\}$
$= \{P(f(x), y)[u/f(x)], P(f(x), f(x))[u/f(x)], P(u, g(v))[u/f(x)]\}$
$= \{P(f(x), y), P(f(x), f(x)), P(f(x), g(v))\}$

The formulas of $\sigma_3[S]$ differ at position 8. The set of terms that begin at that position is $D_3 = \{y, f(x), g(v)\}$. We go back to the loop repetition test.

Since D_3 is not empty, we execute the loop. Here we test if D_3 has a variable and a term that does not contain that variable as a subterm. It does, so we define the substitution $\sigma_4 = [y/f(x)] \diamond \sigma_3$. We compute the set $\sigma_4[S] = [y/f(x)][\sigma_3[S]]$.

$\sigma_4[S] = [y/f(x)]\{P(f(x), y), P(f(x), f(x)), P(f(x), g(v))\}$
$= \{P(f(x), y)[y/f(x)], P(f(x), f(x))[y/f(x)], P(f(x), g(v))[y/f(x)]\}$
$= \{P(f(x), f(x)), P(f(x), f(x)), P(f(x), g(v))\}$

$= \{P(f(x), f(x)), P(f(x), g(v))\}$

The formulas of $\sigma_4[S]$ differ at position 8, so $D_4 = \{f(x), g(v)\}$. We go to the loop repetition test.

The set D_4 is not empty, so we repeat the loop. We first ask if D_4 has a variable and a term that does not contain that variable. Since D_4 has no variables, the algorithm exits with failure. The set S is not unifiable.

Before we go on to prove the correctness of the unification algorithm, let us make some observations that will help us in the proofs.

Observations 2.9.29 1. If the terms t_1 and t_2 start with distinct function symbols then they cannot be unified.

This is easy to see. Let s be a substitution.

If $t_1 = f(u_1, \ldots, u_n)$ and $t_2 = g(v_1, \ldots, v_m)$ then $s[t_1] = f(s[t_1], \ldots, s[t_n])$ and $s[t_2] = g(s[v_1], \ldots, s[v_m])$. So, $s[t_1]$ and $s[t_2]$ cannot be equal since they start with different letters.

The same thing occurs when one or both terms are constants.

So, if t_1 and t_2 start with distinct symbols and are unifiable, at least one of them must be a variable.

2. A variable cannot be unified with a term that contains it as a proper subterm.

Let us show that. Let x be a variable, and $t = f(t_1, \ldots, t_n)$ be a term that contains one or more occurrences of x. Let s be a substitution. Then $s[x]$ is a substring of $s[t] = f(s[t_1], \ldots, s[t_n])$. The string $s[t]$ is longer than the string $s[t]$ since it contains $s[t]$ plus the symbol f and two parentheses. Then, the strings $s[t]$ and $s[t]$ cannot be equal no matter what s is.

Lemma 2.9.30 *The substitutions σ_n have the form*

$\sigma_n = [y_n/t_n] \diamond [y_{n-1}/t_{n-1}] \diamond \ldots \diamond [y_1/t_1]$,

where $y_1, \ldots, y_n$ are distinct variables, and $t_1, \ldots, t_n$ are terms that satisfy the condition

for all i, $1 \leq i \leq n$, t_i does not contain any of the variables $y_1, \ldots y_i$.

At the same time, $\sigma_n[S]$ does not contain any of the variables $y_1, \ldots, y_n$.

Proof: We prove the lemma by induction on n.

Basis: $n = 1$.

σ_1 is defined as a pair y_1/t_1 where t_1 is a term that does not contain y_1. At the same time, $\sigma_1[S]$ is obtained by replacing every y_1 in S by t_1. Since t_1 does not contain occurrences of y_1, neither does $\sigma_1[S] = S[y_1/t_1]$.

Inductive step: Assume that

$\sigma_n = [y_n/t_n] \diamond [y_{n-1}/t_{n-1}] \diamond \ldots \diamond [y_1/t_1]$,

where $t_1, \ldots, y_n$ are distinct variables,

for all i, $1 \leq i \leq n$, the terms t_i do not contain any of the variables $y_1, \ldots, y_i$,

and $\sigma_n[S]$ does not contain any of the variables $y_1, \ldots, y_n$.

The substitution σ_{n+1} is defined as $\sigma_{n+1} = [y_{n+1}/t_{n+1}] \diamond \sigma_n$. We apply the induction hypothesis and get that

$\sigma_{n+1} = [y_{n+1}/t_{n+1}] \diamond [y_n/t_n] \diamond [y_{n-1}/t_{n-1}] \diamond \ldots \diamond [y_1/t_1]$.

This substitution is defined only when D_n is not empty and contains a pair y_{n+1}, t_{n+1}, such that t_{n+1} does not have y_{n+1} as a subterm. Since $\sigma_n[S]$ does not contain any of the variables $y_1, \ldots, y_n$, neither does its disagreement set D_n. So, the variable y_{n+1} is distinct from the variables $y_1, \ldots, y_n$. Since $y_1, \ldots, y_n$ are distinct by hypothesis,

$y_1, \ldots, y_{n+1}$ are distinct variables.

Now, t_{n+1} does not contain any the variables $y_1, \ldots y_n$ since t_{n+1} is a term of $\sigma_n[S]$ and $\sigma_n[S]$ has no occurrences of these variables. Moreover, by the choice of t_{n+1}, t_{n+1} does not contain y_{n+1}. So, the term t_{n+1} satisfies the condition that it is free of occurrences of y_1,..., y_{n+1}. By adding this fact to the induction hypothesis we get that

for all i, $1 \leq i \leq n+1$, the terms t_i do not contain any of the variables y_1,...,y_i.

Finally, $\sigma_{n+1}[S] = [y_{n+1}/t_{n+1}][\sigma_n[S]] = \sigma_n[S][y_{n+1}/t_{n+1}]$, i.e. $\sigma_{n+1}[S]$ is obtained by replacing every occurrence of y_{n+1} by a term that does not contain any of the variables y_1,...,y_n,y_{n+1}. Since $\sigma_n[S]$ does not have occurrences of y_1,...,y_n,

$\sigma_{n+1}[S]$ has no occurrences of y_1,...,y_n, y_{n+1}.

Q.E.D.

Corollary 2.9.31 *The Unification Algorithm terminates.*

Proof: If the algorithm does not terminate then it keeps computing the substitutions σ_n. By Lemma 2.9.30, the domain of σ_n has n distinct variables from S. Since S has a finite number of variables, n cannot be greater than the number of variables in S. **Q.E.D.**

The next theorem is due to Robinson ([]).

Theorem 2.9.32 (The Unification Theorem) *If S is a unifiable set, then The Unification Algorithm produces an mgu.*

Proof: Let $S = \{A_1, A_2, \ldots, A_m\}$ and s be a unifier for S. Since the algorithm terminates, it is sufficient to show that s factors through all σ_n, i.e. for every n there is a substitution τ_n such that $s = \tau_n \diamond \sigma_n$.

The proof is by induction on n.

Basis: $n = 0$. Then $\sigma_0 = []$, and $s = s \diamond []$. So, s factors through σ_0.

Inductive Step: Assume that $s = \tau_n \diamond \sigma_n$. Let D_n be the disagreement set of $\sigma_n[S]$. If D_n is empty we are done, since σ_n is a unifier of S and $s = \tau_n \diamond \sigma_n$.

Assume that D_n is not empty. Then $D_n = \{t_1, t_2, \ldots, t_k\}$, $2 \leq k \leq n$. Since s unifies S,

(1) $s[t_1] = s[t_2] = \ldots = s[t_k]$.

By Observations 2.9.29.1, D_n must contain a variable, say t_1. By Observations 2.9.29.2, S must have an element, say t_2, that does not contain t_1 as subterm.

Case 1: t_1 is in the domain of s.

Then there is some t_3 such that $t_1/t_3 \in s$, i.e.

(2) $s[t_1] = t_3$

From (1) and (2) we get that

(3) $s[t_2] = t_3$

By Lemma 2.9.30, neither t_1 nor the variables of t_2 are in the domain of σ_n. We apply the equality $s = \tau_n \diamond \sigma_n$ to t_2 and get

$s[t_2] = \tau_n[\sigma_n[t_2]] = \tau_n[t_2]$, i.e.

(4) $s[t_2] = \tau_n[t_2]$.

From (3) and (4) we have

(5) $\tau_n[t_2] = t_3$

Let

(6) $\tau_{n+1} = \tau_n - \{t_1/t_3\}$.

Now,

$\tau_{n+1} \diamond [t_1/t_2] = \tau_{n+1} \cup \{t_1/\tau_{n+1}[t_2]\}$ by the definition of $\diamond$

$= (\tau_n - \{t_1/t_3\}) \cup \{t_1/\tau_{n+1}[t_2]\}$ by (6)

$= (\tau_n - \{t_1/t_3\}) \cup \{t_1/\tau_n[t_2]\}$ because t_1 in not in t_2

$= (\tau_n - \{t_1/t_3\}) \cup \{t_1/t_3\}$ by (5)

$= \tau_n$.

So,

(7) $\tau_{n+1} \diamond [t_1/t_2] = \tau_n$.

From (7) we get that

$\tau_{n+1} \diamond \sigma_{n+1}$

$= \tau_{n+1} \diamond [t_1/t_2] \diamond \sigma_n$ by the construction of σ_{n+1}

$= \tau_n \diamond \sigma_n$ by (7)

$= s$ by induction hypothesis

So, σ_{n+1} is defined and $\sigma_{n+1} = [t_1/t_2] \diamond \sigma_n$.

Case 2: t_1 is not in the domain of s.

Then $s[t_1] = t_1$. Since $s[t_1] = s[t_2]$, t_2 must be a variable and s must contain the pair $t_2/s[t_1] = t_2/t_1$.

Since t_2 is not in the domain of σ_n, t_2/t_1 must be in τ_n.

Now let

(8) $\tau_{n+1} = \tau_n$.

Then

$\tau_{n+1} \diamond [t_1/t_2]$

$= \tau_n \diamond [t_1/t_2]$ by (8)

$= \tau_n \cup \{t_1/\tau_n[t_2]\}$ by the definition of $\diamond$

$= \tau_n \cup \{t_1/t_1\}$ because $\tau_n[t_2] = t_1$

$= \tau_n$.

Then the proof that $\tau_{n+1} \diamond \sigma_{n+1} = s$ is identical to the one provided in Case 1.

Q.E.D.

Let us illustrate the proof of The Unification Theorem with an example.

Example 2.9.33 Let $S = \{P(x, f(x)), P(y, z)\}$. The substitution $s = [y/x, z/f(x)]$ is a unifier of S because

$s[S] = s[\{P(x, f(x)), P(y, z)\}]$

$= \{P(x, f(x)[y/x, z/f(x)], P(y, z)[y/x, z/f(x)]\}$
$= \{P(x, f(x)), P(x, f(x))\} = \{P(x, f(x))\}$.

Let us follow the proof of the Unification Theorem to show that s factors through the mgu found by the unification algorithm.

At the beginning $\sigma_0 = []$, and $s = s \diamond []$, so $\tau_0 = s = [y/x, z/f(x)]$.

We compute $\sigma_0[S]$ and get $\sigma_0[S] = [][S] = S$. The disagreement set of $\sigma_0[S]$ is $D_0 = \{x, y\}$. We can choose either one of the substitutions $[x/y]$ or $[y/x]$ as σ_1. Let us choose the first one. We apply the second case of the proof to find τ_1. So, $\tau_1 = \tau_0 = [y/x, z/f(x)]$.

Now let us find $\sigma_1[S]$. We get $\sigma_1[S] = \{P(x, f(x))[x/y], P(y, z)[x/y]\} = \{P(y, f(y)), P(y, z)\}$. The disagreement set of $\sigma_1[S]$ is $D_1 = \{z, f(y)\}$. We must take $\sigma_2 = [z/f(y)] \diamond \sigma_1 = [z/f(y)] \diamond [x/y]$. We apply the first case of the proof to find τ_2. We compute it by removing the pair $z/\tau_1[z]$ from τ_1 to get $\tau_2 = [y/x]$.

We go on with the computation of the mgu. We compute $\sigma_2[S]$.

$\sigma_2[S] = [z/f(y)][\sigma_1[S]$
$= [z/f(y)][\{P(y, f(y)), P(y, z)\}$
$= \{P(y, f(y))[z/f(y)], P(y, z)[z/f(y)]\}$
$= \{P(y, f(y)), P(y, f(y))\} = \{P(y, f(y))\}$.

So, the disagreement set of $\sigma_2[S]$ is $D_2 = \phi$. The unification algorithm tells us that $\sigma_2 = [z/f(y)] \diamond [x/y] = [x/y, z/f(y)]$ is an mgu. From the proof of the theorem,

$s = \tau_2 \diamond \sigma_2$.

So, $\tau_2 = [y/x]$ is the factoring substitution. As an exercise in substitution composition, let us verify this equality.

$\tau_2 \diamond \sigma_2 = [y/x] \diamond [x/y, z/f(y)]$
$= [y/x, x/y[y/x], z/f(y)[y/x]]$ definition of composition
$= [y/x, x/x, z/f(x)]$
$= [y/x, z/f(x)]$ remove the pairs of the type v/v
$= s$.

Exercises

Exercise 2.9.1 *Prove that* $dom(s_2) - dom(s_1) \subseteq dom(s_2 \diamond s_1) \subseteq dom(s_1) \cup dom(s_2)$.

Exercise 2.9.2 *A substitution s has a left inverse, when there is a substitution σ such that $\sigma \diamond s = []$, where $[]$ is the empty substitution. What property do the members of s satisfy?*

If $\sigma \diamond s = []$, what is the value of $s \diamond \sigma$?

Exercise 2.9.3 *Let $s = [x_1/y_1, \ldots x_n/y_n]$ be a relabeling away from $\{x_1, \ldots, x_n\}$. Show that there is a substitution σ such that $s \diamond \sigma = s$ and $\sigma \diamond s = \sigma$. Moreover, σ is unique, i.e. $s \diamond \rho = s$ and $\rho \diamond s = \rho$ implies $\rho = \sigma$.*

Exercise 2.9.4 *Let s be a substitution and $Var[s]$ be the set of variables that occur in at least one of the pairs of s, i.e. $Var[s] = dom(s) \cup Var[ran(s)]$. Show that $Var[\pi] \cap Var[\sigma] = \phi$ implies $\pi \diamond \sigma = \sigma \diamond \pi = \pi \cup \sigma$.*

Exercise 2.9.5 *Let π and ρ be two substitutions such that $dom[\pi] \cap dom[\rho] = \phi$.*

1. Show that $\sigma \diamond (\pi \cup \rho) \subseteq (\sigma \diamond \pi) \cup (\sigma \diamond \rho)$.

2. Show that $\sigma \diamond (\pi \cup \rho) = (\sigma \diamond \pi) \cup (\sigma \diamond \rho)$ is not always true.

Exercise 2.9.6 *On the set of substitutions we define the relation $\geq$ by*

$s_1 \geq s_2$ iff there is a substitution s such that $s_1 = s \diamond s_2$.

Show that the relation $\geq$ is reflexive and transitive. Give an example that proves that $\geq$ is not antisymmetric.

Exercise 2.9.7 *The pair $x/t \in s_2$ is useless in the composition $s_2 \diamond s_1$ if $s_2 \diamond s_1 = (s_2 - \{x/t\}) \diamond s_1$.*

1. Show that the pair $x/t \in s_2$ is useless when $x \in Var[s_1] - Var[ran[s_1]]$.

2. The composition $s_2 \diamond s_1$ is reduced when s_2 has no useless pairs. Show that for every composition $s_2 \diamond s_1$ we can find a substitution σ such that $s_2 \diamond s_1 = \sigma \diamond s_1$ and $\sigma \diamond s_1$ is reduced.

Exercise 2.9.8 *On the set of substitutions we define the relation $\approx$ as $s_1 \approx s_2$ if $s_1 \geq s_2$ and $s_2 \geq s_1$.*

a. Show that $\approx$ is an equivalence relation.

b. If $s_1 \approx s_2$ then s_1 and s_2 differ by a relabeling, i.e. $s_1 = s \diamond s_2$ and s is a relabeling.

Exercise 2.9.9 *Find $s_1 \diamond s_2$ and $s_2 \diamond s_1$ for the following pairs of substitutions:*

a. $s_1 = [x/f(x,y), y/g(x), z/f(u,v)]$, $s_2 = [x/h(y), y/g(z), v/g(w)]$

b. $s_1 = [y/f(x)]$, $s_2 = [z/g(y), u/h(y,v)]$

Exercise 2.9.10 *Let $[y_1/t_1]$ and $[y_2/t_2]$ be two substitutions such that y_2 does not occur neither in t_1 nor in t_2, and y_1 does not occur t_1.*

a. Find two terms t_1 and t_2 that satisfy the above conditions and the inequality $[y_1/t_1] \diamond [y_2/t_2] \neq [y_2/t_2] \diamond [y_1/t_1]$.

b. What condition must t_2 satisfy for the equation $[y_1/t_1] \diamond [y_2/t_2] = [y_2/t_2] \diamond [y_1/t_1]$ to be true?

Exercise 2.9.11 *The unification relation is the set of all pairs of terms that are unifiable. Show that the unification relation is reflexive and symmetric, but not transitive.*

Exercise 2.9.12 *Let σ be an mgu of $S = \{A_1, \ldots, A_m\}$ and v a variable that does not occur in S. Show that $v \notin \sigma$.*

Exercise 2.9.13 *What is the mgu of $S = \{P(t_1, \ldots, t_n)\}$?*

Exercise 2.9.14 *Use the unification algorithm to find the mgu of the following sets:*

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

b. $S = \{P(f(g(x)), h(y), z), P(f(u), v, y), P(f(y), h(g(w)), g(z_1))\}$

c. $S = \{P(f(x), g(x,y)), P(y, g(z,u)), P(f(v), g(f(v), w))\}$.

Exercise 2.9.15 *Assume that 2 unifiable atoms have an mgu. Use this fact to prove that every unifiable set of formulas has an mgu.*

Exercise 2.9.16 *Using Example 2.9.33 as a guide find the mgu and the factoring of the following unifiers:*

a. The set is $S = \{P(x,y), P(y,x)\}$ *and the unifier is* $s = [x/f(x,y), y/f(x,y)]$.

b. The set is $T = \{P(x,f(x)), P(x,z), P(g(u),f(v))\}$ *and the unifier is* $s = [x/g(a), z/f(g(a)), u/a, v/g(a)]$.

2.10 Resolvents

The main objective of this section is to define FOL *resolvents* and to study their basic properties. The development of the theory of resolvents follows the steps of Section 1.8. It also uses the mgu algorithm from the preceding section. So, we recommend a quick examination of these sections before advancing into the current one.

We start with the definition of the 1 step resolvents and then show that every model of the parent clauses is also a model of the resolvent. Then we define the set $Res^*[S]$, that contains all resolvents of the set of clauses S. The resolvents in $R^*[S]$ are obtained in 0, 1, 2, or any finite number of steps from the clauses of S. The section concludes with the proposition that states that $Res^*[S]$ is semantically equivalent to S.

In propositional logic we identified clauses with sets of literals and this formalism facilitated the presentation of recursion. We will use the same representation for FOL clauses. Now let us start by defining the (1-step) *resolvents*.

Definition 2.10.1 (FOL resolvents) *Let* C_1, C_2 *be two FOL clauses,* s_1 *a rename away from* $Var[C_1]$, *and* s_2 *be a relabeling away from* $Var[C_2]$ *such that* $s_1[C_1]$ *and* $s_2[C_2]$ *have no variables in common. Let* $L_1,\ldots,L_n$ *be atoms of* $s_1[C_1]$ *and* $M_1,\ldots,M_m$ *be literals of* $s_2[C_2]$ *such that* $S = \{L_1,\ldots,L_n,\overline{M_1},\ldots,\overline{M_m}\}$ *is unifiable. Let* σ *be an mgu of* S. *Then*

$R = \sigma[(s_1[C_1] - \{L_1,\ldots,L_n\}) \cup (s_2[C_2] - \{M_1,\ldots,M_m\})]$

is a resolvent of C_1 *and* C_2.

We represent this derivation in Figure 2.48.

Example 2.10.2 Figure 2.49 shows the 3 resolvents of $C_1 = \{P(x,y)\}$ and $C_2 = \{\neg P(x,y), \neg P(y,z), Q(x,z)\}$. The variables x and y occur in both clauses, so we relabeled the variables of C_2. We could have changed the x and the y of C_1, or change them in both clauses. The rename s_2 changes x to u, and y to v. Now $s_1[C_1] = \{P(x,y)\}$ and $s_2[C_2] = \{\neg P(u,v), \neg P(v,z), Q(u,z)\}$ have no variables in common. We can unify the sets $S_1 = \{P(x,y), P(u,v), P(v,z)\}$, $S_2 = \{P(x,y), P(u,v)\}$, and $S_3 = \{P(x,y), P(v,z)\}$. The first atom is from $s_1[C_1]$ and the negations of the remaining ones are literals in $s_2[C_2]$.

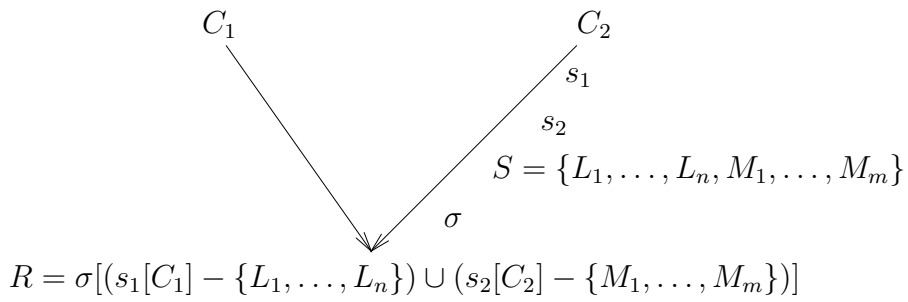


Figure 2.48: R is a resolvent of C_1 and C_2

We use The Unification Algorithm and get the mgu's $\sigma_1 = [u/x, v/x, y/x, z/x]$, $\sigma_2 = [u/x, v/y]$ and $\sigma_3 = [v/x, z/y]$. The resolvent R_1 is obtained by removing $P(x, y)$ from $s_1[C_1]$ and $\neg P(u, v), \neg P(v, z)$ from $s_2[C_2]$. We are left with the literal $Q(u, z)$ from $s_2[C_2]$. We apply σ_1 to the remaining literal and get $R_1 = \{Q(x, x)\}$.

Our goal is to show that every model of the parent clauses is also a model of the resolvents.

Lemma 2.10.3 *Let C be a clause with variables $y_1, \ldots, y_n$, and let s be a substitution. Let $z_1, \ldots, z_m$ be the variables of the clause $s[C]$. Then every model of $\forall y_1 \forall y_n C$ is a model of $\forall z_1 \ldots \forall z_m s[C]$.*

Proof: We assume

(1) $\models_{\mathcal{A}} \forall y_1 \forall y_2 \ldots \forall y_n C$,

and need to show

(2) $\models_{\mathcal{A}} \forall z_1 \forall z_2 \ldots \forall z_m s[C]$.

We rename all variables in the sequence $z_1, \ldots, z_m$ that occur in C. Let σ be the renaming substitution, and $u_1, \ldots, u_m$ be the new set of variables.

By The Relabeling Lemma,

(3) $\forall z_1 \forall z_2 \ldots \forall z_m s[C] \equiv \forall u_1 \forall u_2 \ldots \forall u_m (\sigma \diamond s)[C]$.

So, we need to show

(4) $\mathcal{A} \models \forall u_1 \forall u_2 \ldots \forall u_m (\sigma \diamond s)[C]$.

The substitution $\sigma \diamond s$ relabels all variables of C, so

(5) $\sigma \diamond s = [y_1/t_1, \ldots, y_n/t_n]$,

and $t_1, \ldots, t_n$ do not contain any of the variables $y_1, \ldots, y_n$.

Now let D be the universe of $\mathcal{A}$. We can get (4) if we show that for any m values in D, $d_1, \ldots, d_m$,

(6) $\mathcal{A}_{[u_1 \leftarrow d_1, \ldots, u_m \leftarrow d_m]}[\sigma[s[C]]] = 1$

We will simplify the notation and write $\mathcal{A}^*$ instead of $\mathcal{A}_{[u_1 \leftarrow d_1, \ldots, u_m \leftarrow d_m]}$.

$\mathcal{A}^*[\sigma[s[C]]]$

$= \mathcal{A}^*[C[y_1/t_1, \ldots, y_n/t_n]]$ by (5)

$= \mathcal{A}^*_{[y_1 \leftarrow \mathcal{A}^*[t_1], \ldots, y_n \leftarrow \mathcal{A}^*[t_n]]}[C]$ by The Translation Lemma

$= \mathcal{A}_{[y_1 \leftarrow \mathcal{A}^*[t_1], \ldots, y_n \leftarrow \mathcal{A}^*[t_n]]}[C]$ since $u_1, \ldots, u_m$ are not in C

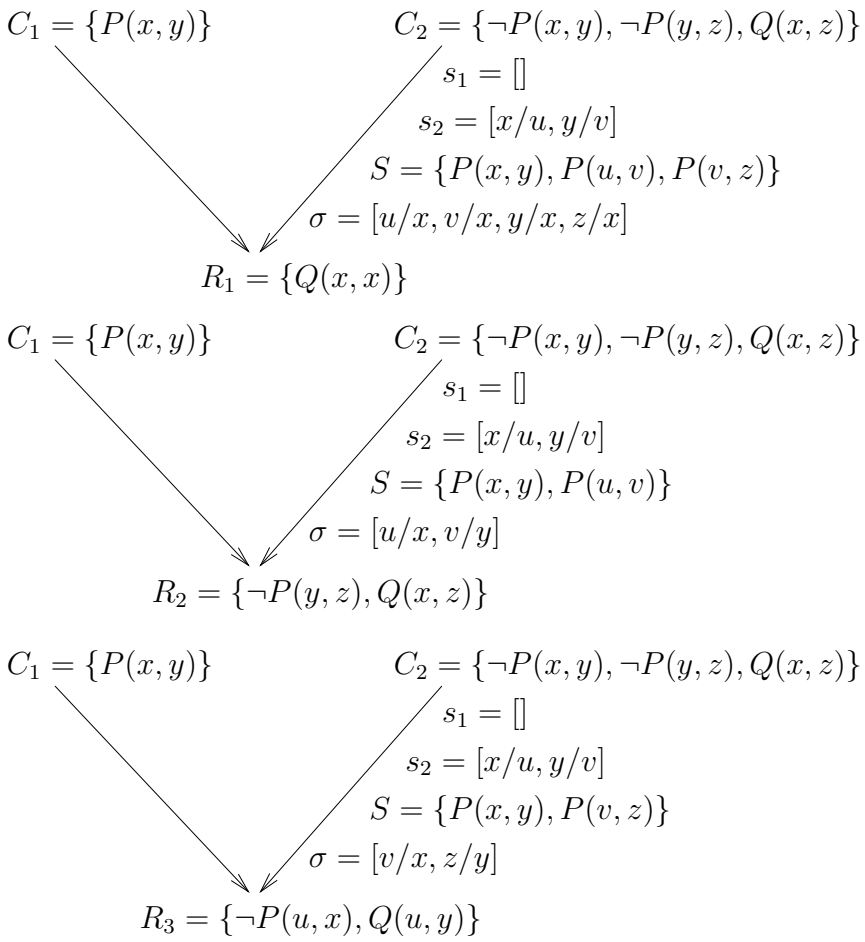


Figure 2.49: The 3 resolvents of $\{P(x,y)\}$ and $\{\neg P(x,y), \neg P(y,z), Q(x,z)\}$

$= 1$ by (1).

The last equality is true since

(7) $\mathcal{A}_{[y_1 \leftarrow e_1, \ldots, y_n \leftarrow e_n]}[C] = 1$

for all $e_1, \ldots, e_n$ in D. So, if we take $e_1 = \mathcal{A}^*[t_1]$, $e_2 = \mathcal{A}^*[t_2], \ldots, e_n = \mathcal{A}^*[t_n]$ we get the desired equality. **Q.E.D.**

Lemma 2.10.4 *Let C_1 and C_1 be two clauses, $x_1, \ldots, x_k$ be the variables of C_1, and $y_1, \ldots, y_l$ the variables of C_2. Let R be a resolvent of C_1 and C_2 and $z_1, \ldots, z_m$ be the variables of R. Then every model of $\forall x_1 \ldots \forall x_k C_1 \wedge \forall y_1 \ldots y_m C_2$ is a model of $\forall z_1 \ldots \forall z_n R$.*

Proof: Let s_1 and s_2 be the relabeling substitutions of C_1 and C_2. Let $u_1, \ldots, u_k$ be the s_1 relabelings of $x_1, \ldots, x_k$, and $v_1, \ldots, v_l$ the s_2 renames of $y_1, \ldots, y_l$. By The Relabeling Lemma,

(1) $\forall u_1 \ldots \forall u_k s_1[C_1] \equiv \forall x_1 \ldots \forall x_k C_1$

and

(2)$\forall v_1 \ldots \forall v_l s_2[C_2] \equiv \forall v_1 \ldots \forall v_l C_2$

Let S be the unifying set of R, σ its unifier, and $w_1, \ldots, w_o$ the variables of $\sigma[s_1[C_1] \wedge s_2[C_2]]$. Now let $\mathcal{A}$ be a model of $\forall x_1 \ldots \forall x_k C_1 \wedge \forall y_1 \ldots y_m C_2$. Then

(3) $\models_{\mathcal{A}} \forall x_1 \ldots \forall x_k C_1$

and

(4)$\models_{\mathcal{A}} \forall y_1 \ldots \forall y_l C_2$.

We apply (1) and (2) to (3) and (4) and get

(5) $\models_{\mathcal{A}} \forall u_1 \ldots \forall u_k s_1[C_1]$

and

(6) $\models_{\mathcal{A}} \forall v_1 \ldots \forall v_l s_2[C_2]$.

We apply Lemma 2.10.3 to (5) and (6) and get (7) and (8).

(7) $\models_{\mathcal{A}} \forall w_1 \ldots \forall w_o (\sigma \diamond s_1)[C_1]$

(8) $\models_{\mathcal{A}} \forall w_1 \ldots \forall w_o (\sigma \diamond s_2)[C_2]$

Now let $d_1, \ldots, d_o$ be o-elements of the universe of $\mathcal{A}$. We abbreviate $\mathcal{A}_{[w_1 \leftarrow d_1, \ldots, w_o \leftarrow d_o]}$ by $\mathcal{A}^*$.

We will show that $\mathcal{A}^*[R] = 1$.

Let S be the unifying set of R and L be the unique atom of $\sigma[S]$.

Case 1: $\mathcal{A}^*[L] = 1$.

By (8), $\mathcal{A}^*$ satisfies the clause $\sigma[s_2[C_2]]$, so $\mathcal{A}^*$ must satisfy some literal $Q \in \sigma[s_2[C_2]]$. Since $\mathcal{A}^*[\neg L] = 0$, $Q \neq \neg L$. Then $Q \in R$, and $\mathcal{A}^*[R] = 1$.

Case 2: $\mathcal{A}^*[L] = 0$. This time we use (7), and get that $\mathcal{A}^*$ models some literal Q of $\sigma[S_1[C_1]]$. Since $\mathcal{A}^*[L] = 0$, $Q \neq L$. Then $Q \in R$, and $\mathcal{A}^*[r] = 1$.

In both cases, $\mathcal{A}^*$ is a model of R. Since the o-tuple $d_1, \ldots, d_o$ is arbitrary,

(9) $\models_{\mathcal{A}} \forall w_1 \ldots \forall w_o R$

We remove the redundant quantifiers from (9) and get

(10) $\models_{\mathcal{A}} \forall z_1 \ldots \forall z_n R$.

Q.E.D.

The one step resolvents, introduced in Definition 2.10.1, are represented as trees of height 1, as shown in Figure 2.48. Now, we follow the pattern set in Section 1.8, and extend the concept to cover not only the one step resolvents, but

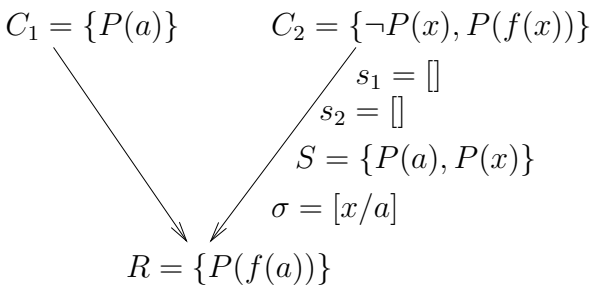


Figure 2.50: $Res^1[S]$ resolvents for Example 2.10.6

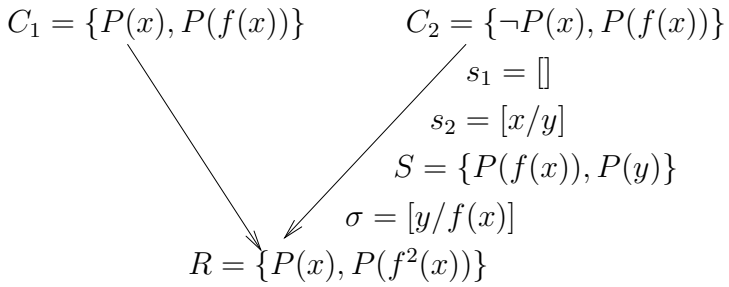


Figure 2.51: More $Res^1[S]$ resolvents for Example 2.10.6

also resolvents of the resolvents, resolvents of the resolvents of the resolvents, and so on.

Definition 2.10.5 (**$Res^n[S]$, $Res^*[S]$**) *Let S be a set of clauses in first order logic.*

$Res^0[S] = S$

$Res^{n+1}[S] = Res^n[S] \cup \{R | R$ *is a resolvent of two clauses of* $Res^n[S]\}$

$Res^*[S] = \bigcup_{i=0}^{\infty} Res^n[S]$.

$Res^n[S]$ is the set of resolvents of height $\leq n$, of S, and $Res^*[S]$ is the set of all resolvents of S.

Example 2.10.6 Let us compute $Res^*[S]$ when $S = \{P(a), \neg P(x) \vee P(f(x))\}$.

We know that $Res^0[S] = S$. We find $Res^1[S]$ by computing all resolvents of the two clauses of $Res^0[S] = S$. We get the trees from Figures 2.50 and 2.51. So, $Res^1[S] = \{P(a), P(f(a)), \neg P(x) \vee P(f(x)), \neg P(x) \vee P(f(f(x)))\}$.

We make the following claim.

Claim 2.10.7 *If $n > 0$, then* $Res^n[S] = \{\{P(a)\}, \ldots, \{P(f^{2^n-1}(a))\}\} \cup \{\{\neg P(x), P(f^{2^i}(x))\} | 1 \leq i \leq 2^n\}$.

Here $f^i(y)$ is the term $f(f(\ldots f(y)\ldots))$, with i occurrences of f. If $i = 0$, then $f^0(x) = x$.

Proof: We do it by induction on n.

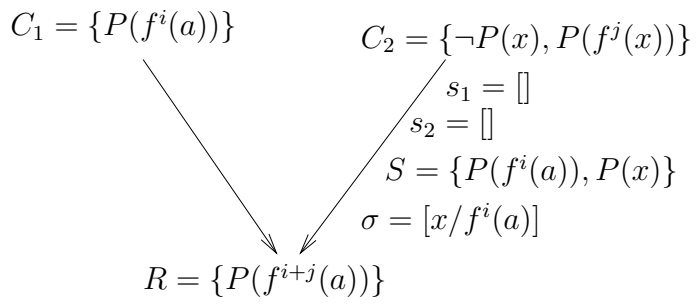


Figure 2.52: $Res^{n+1}[S]$ resolvents for Example 2.10.6

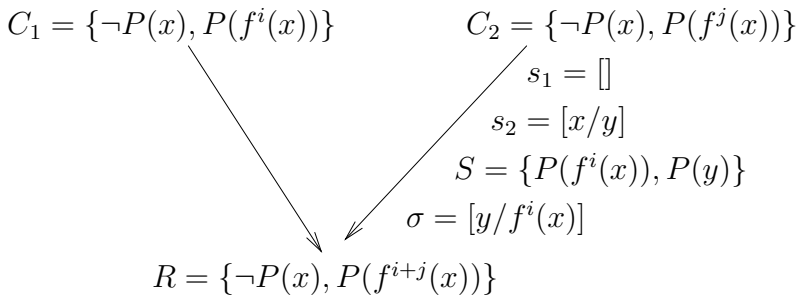


Figure 2.53: More $Res^{n+1}[S]$ resolvents for Example 2.10.6

Basis: $n = 1$. We already computed $Res^1[S] = \{\{P(a)\}, \{P(f(a)\}, \{\neg P(x), P(f(x))\}, \{\neg P(x), P(f^2(x))\}\}$. Since $2^1 - 1 = 1$, and $2^1 = 2$, $Res^1[S] = \{\{P(f^i(a))\}|0 \leq i \leq 2^1 - 1\} \cup \{\{\neg P(x), P(f^i(x))\}|1 \leq i \leq 2^n\}$.

Inductive Step: Assume that $Res^n[S] = \{\{P(f^i(a))\}|0 \leq i \leq 2^n - 1\} \cup \{\{\neg P(x), P(f^i(x))\}|1 \leq i \leq 2^n\}$.

$Res^{n+1}[S]$ contains $Res^n[S]$ as well as the resolvents of the clauses in $Res^n[S]$. The resolvents are obtained by matching a clause in the first set of union with a clause in the second set and by matching two clauses in the second set. These unifications are shown in Figures 2.52 and 2.53. In Figure 2.52, $0 \leq i \leq 2^n - 1$ and $1 \leq j \leq 2^n$. So, the resolvents are all clauses $P(f^{i+j}(a))$ with $1 \leq i + j \leq 2^{n+1} - 1$. Since $\{P(a)\}$ is already in S, all clauses $\{P(f^i(a))\}$ with i between 0 and $2^{n+1} - 1$ are in $Res^{n+1}[S]$.

In Figure 2.53, $1 \leq i \leq 2^n$ and $1 \leq j \leq 2^n$. So, the resolvents are exactly the clauses $\{\neg P(x), P(f^{i+j}(x))\}$ with $2 \leq i + j \leq 2^{n+1}$. Since $\{\neg P(x), P(f(x))\}$ is already in S, the clauses $\{\neg P(x), P(f^k(x))\}$ with $1 \leq k \leq 2^{n+1}$ are in $Res^{n+1}[S]$.

Since $Res^{n+1}[S]$ has no other resolvents except those shown in Figures 2.52 and 2.53, $Res^{n+1}[S] = \{\{P(f^k(a))\}|0 \leq k \leq 2^{n+1}-1\} \cup \{\{\neg P(x), P(f^k(x))\}|1 \leq k \leq 2^{n+1}\}$. **Q.E.D. claim**

Now $Res^*[S] = \{P(f^k(a))|k \geq 0\} \cup \{\neg P(x) \vee P(f^k(x))|k \geq 1\}$.

Observation 2.10.8 Example 2.10.6 shows 2 fundamental differences between the FOL resolvents and the propositional logic resolvents.

1. In FOL it is possible to have an infinite set of resolvents for a finite set of clauses. The set S from Example 2.10.6 is finite, but $Res^*[S]$ is not. This situation does not occur in the propositional calculus where $Res^*[S]$ is finite whenever S is finite.

2. In FOL we must compute the resolvents of two identical clauses. In propositional calculus, these resolvents, when they exist, are tautologies, so they are discarded.

We define the *height* of a resolvent R to be the smallest n such that $R \in Res^n[S]$.

Definition 2.10.9 (universal closure of a set of clauses) *Let S be a set of clauses. The universal closure of S, written $\overline{S}$, is the set $\{\forall y_1 \ldots \forall y_n C | C \in S$ and $\{y_1, \ldots, y_n\} = Var[C]\}$.*

Lemma 2.10.10 *Let S be a set of clauses. Then $\overline{S} \equiv \overline{Res^*[S]}$.*

Proof: Assume that

(1) $\models_{\mathcal{A}} \overline{S}$.

We will prove that for all $\forall z_1 \ldots \forall z_n R \in \overline{Res^*[S]}$, $\models_{\mathcal{A}} \forall z_1 \ldots \forall z_n R$.

The proof is by induction on n, the height of R.

Basis: $n = 0$. Then $\forall z_1 \ldots \forall z_n R$ is a clause in $\overline{S}$, so $\models_{\mathcal{A}} \forall z_1 \ldots \forall z_n R$.

Inductive Step: Let us assume that $\models_{\mathcal{A}} \overline{Res^m[S]}$ and let R be a resolvent of height $m+1$. Then R is a resolvent of two clauses C_1 and C_2 of $Res^m[S]$. Since $\mathcal{A}$ is a model of $\overline{Res^n[S]}$,

(2) $\models_{\mathcal{A}} \forall x_1 \ldots \forall x_p C_1$,

and

(3) $\models_{\mathcal{A}} \forall y_1 \ldots \forall y_q C_2$.

where $\forall x_1 \ldots \forall x_n C_1$ and $\forall y_1 \ldots \forall y_p C_2$ are the universal closures of C_1 and C_2. By Lemma 2.10.4,

(4) $\models_{\mathcal{A}} \forall z_1 \ldots \forall z_n R$.

Q.E.D.

Corollary 2.10.11 *Let $F = \forall x_1 \ldots \forall x_n (C_1 \wedge C_2 \ldots \wedge C_m)$ be a Skolem normal form without equality. If $\square \in Res^*[\{C_1, C_2, \ldots, C_m\}]$ then F is unsatisfiable.*

Proof: From the construction of F we know that

(1) $F \equiv \overline{\{C_1, C_2, \ldots, C_m\}}$.

From Lemma 2.10.10, we get

(2) $\overline{\{C_1, C_2, \ldots, C_m\}} \equiv \overline{Res^*[\{C_1, C_2, \ldots, C_m\}]}$.

So,

(3) $F \equiv \overline{\{C_1, C_2, \ldots, C_m\}} \equiv \overline{Res^*[\{C_1, C_2, \ldots, C_m\}]}$.

Since $\overline{Res^*[\{C_1, C_2, \ldots, C_m\}]}$ is unsatisfiable, so is F.

Q.E.D.

Exercises

Exercise 2.10.1 *Let C be a clause and s be a rename away from $Var[C]$.*

1. Show that the function $s : C \longrightarrow s[C]$ defined by $L \mapsto \overline{S}[L]$ is one-to-one and onto.

2. Let $L_1, \ldots, L_n \in C$. Show that $s[C - \{L_1, \ldots, L_n\}] = s[C] - s[\{L_1, \ldots, L_n\}]$.

Exercise 2.10.2 *Find all resolvents of the pairs of clauses shown below.*

a. $C_1 = \{P(x, y), P(u, f(z))\}$ and $C_2 = \{\neg P(g(x), y)\}$

b. $C_1 = \{\neg P(x, y), \neg P(y, z), P(x, z)\}$ and $C_2 = \{P(a, b)\}$

c. $C_1 = \{P(x, f(x)), P(y, y), Q(x, y)\}$ and $C_2 = \{\neg P(x, y), \neg Q(x, x)\}$

d. $C_1 = \{\neg P(x, y), \neg P(y, z), P(x, z)\}$ and $C_2 = \{\neg P(a, x), \neg P(x, a)\}$.

Exercise 2.10.3 *Find all resolvents of the following pairs of clauses:*

a. $C_1 = \{P(x, y), P(x, f(x)), Q(x, y)\}$, $C_2 = \{\neg P(g(x), y), \neg Q(f(z), g(w)), \neg Q(u, v)\}$.

b. $C_1 = \{P(x, y), P(g(x), y), \neg Q(x)\}$ and $C_2 = \{\neg P(x, x), Q(x), Q(b)\}$.

Exercise 2.10.4 *We say that a clause C_1 subsumes a clause C_2 if there is a substitution s such that $s[C_1]$ is a subset of C_2. For example, the clause $C_1 = \{P(x, y), Q(x)\}$ subsumes $C_2 = \{P(f(x), b), P(g(x), a), Q(b)\}$ because for the substitution $s = [x/f(x), y/b]$, $s[C_1] = \{P(f(x), b), Q(b)\}$ is a subset of C_2.*

The set $s[C_1]$ does not have to be a proper subset of C_2. For example, $C_1 = \{P(x, y), Q(y)\}$ subsumes $C_2 = \{P(u, v), Q(v)\}$ since the substitution $s = [x/u, y/v]$ has the property that $s[C_1] = C_2$.

Write an algorithm that takes as input two clauses C_1 and C_2 and determines if C_1 subsumes C_2.

Exercise 2.10.5 *Prove that if C_1 subsumes C_2 then $\{C_1, C_2\} \equiv \{C_1\}$.*

This result allows us to eliminate the subsumed clauses.

Exercise 2.10.6 *The binary resolvents of two clauses C_1 and C_2 are obtained by unifying one literal from C_1 with one literal from C_2. This restriction of resolution is called binary resolution. Show that we cannot derive the empty clause from $C_1 = \{P(x), P(y)\}$ and $C_2 = \{\neg P(u), \neg P(v)\}$ using binary resolution.*

Exercise 2.10.7 *Let S be a unifiable subset of the clause C and σ an mgu of S. Then we say that $\sigma[S]$ is a factoring of C. For example $s = [y/x]$ is a factoring of $C = \{P(x), P(y)\}$ because $s[P(x)] = s[P(y)] = P(x)$.*

Give a definition of resolvents using binary resolution and factoring and show that your statement is equivalent to our definition of resolvent.

Exercise 2.10.8 *Let $C_1 = \{A_1, A_2, A_3, \ldots, A_n\}$ and $C_2 = \{B_1, B_2, \ldots, B_m\}$ be two clauses that that have no common variables. Let us assume that $S = \{A_1, A_2, \overline{B_1}$ is unifiable.*

Let σ be the mgu of $\{A_1, B_1\}$ and R_1 be the resolvent and let us assume that $\sigma[A_2] \neq \sigma[A_1]$.

1. Show that there is a relabeling ρ of C_2 such that R_1 and $\rho[C_2]$ are unifiable on $\{\sigma[A_2], \overline{\rho[B_1]}\}$.

2. Let R be the resolvent of C_1 and C_2 on $\{A_1, A_2, \overline{B_1}\}$ and R_2 be the resolvent of R_1 and $\rho[C_2]$ on $\{\sigma[A_2], \overline{\rho[B_1]}\}$, as shown in Figure 2.54. How does R compare to R_2? Does either one factor through the other?

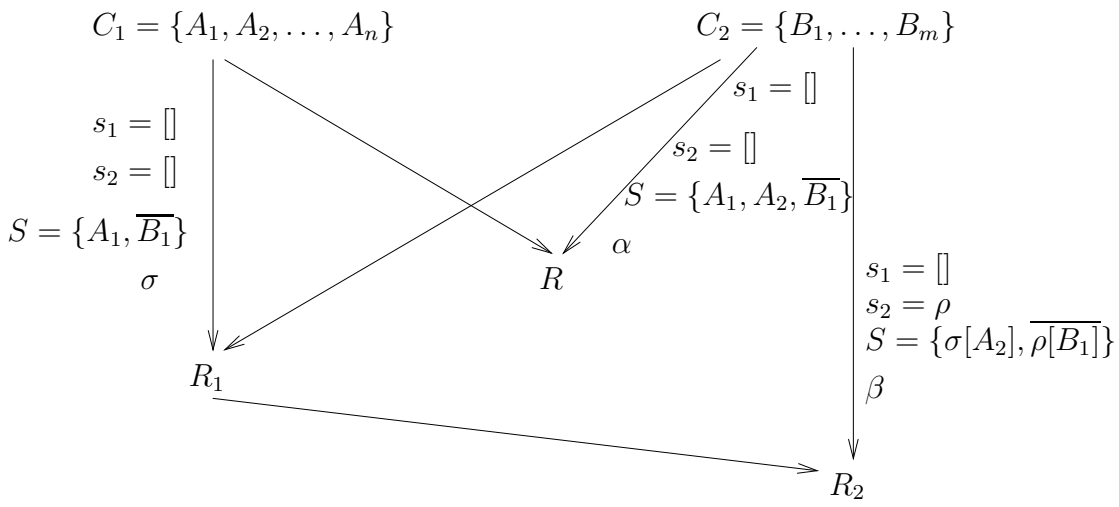


Figure 2.54: The 3 resolvents from Exercise 2.10.8

Exercise 2.10.9 *Describe the sets Res^n and Res^* for the following sets:*
a. $S = \{\{P(a)\}, \{P(b)\}, \{\neg P(x), P(f(x))\}\}$,
b. $T = \{\{P(0,1)\}, \{\neg P(x,y), P(s(x), s(y))\}\}$.

Exercise 2.10.10 *Find $Res^1[S]$ for $S = \{\{P(a,b)\}, \{P(b,a)\}, \{P(b,c)\}, \{\neg P(x,y), \neg P(y,z), P(x,z)\}\}$.*

Exercise 2.10.11 *Let C be a clause and $x_1, \ldots, x_n$ be a list that contains, without repetitions, all variables of C. Let $\{y_1, \ldots, y_m\}$ be a set of variables that includes all variables of C. As in any set, a variable may be repeated several times. Show that $\forall x_1 \ldots \forall x_n C \equiv \forall y_1 \ldots \forall y_m C$.*

2.11 Resolution in First Order Logic

In this section we use the concepts and results from Sections 2.10 and 1.8 to define resolution proofs in first order logic. We start by defining *derivation sequences* and *derivation trees* and then we prove *The Lifting Lemma*, that allows us to *lift* the Herbrand Expansion derivations to resolutions in first order logic.

The definitions of derivation sequence and derivation tree are identical to the ones defined for propositional logic. We list them here for completeness's sake.

Definition 2.11.1 (derivation of a clause) *A derivation of the clause C from the set of clauses S is $C_1, C_2, \ldots, C_n$ that satisfies the following restrictions:*

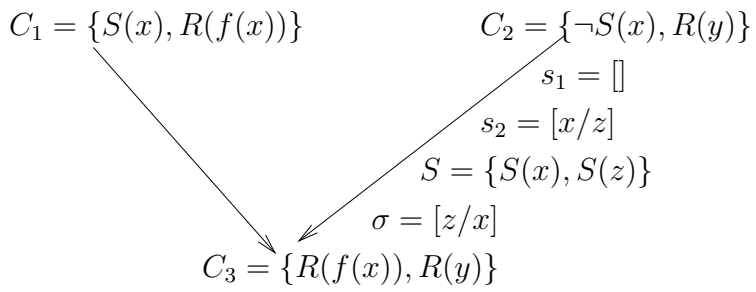


Figure 2.55: C_3 is a resolvent of C_1 and C_2

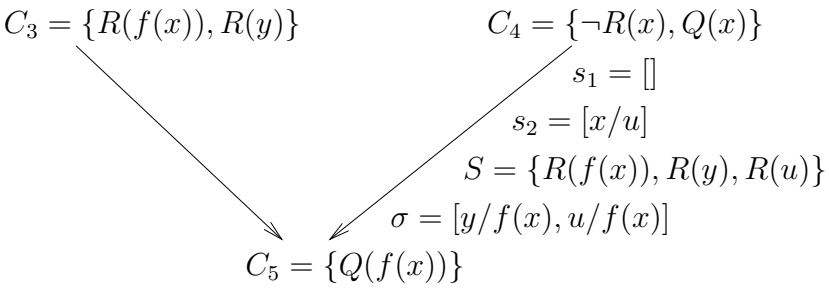


Figure 2.56: C_5 is a resolvent of C_3 and C_4

1. for all indices i, $\leq i \leq n$, C_i is either a clause in S or is derived from two preceding clauses C_j and C_j by FOL-resolution. The clauses C_i and C_j do not have to be distinct.

2. $C_n = C$.

We say that n is the length of the derivation.

Let us give a couple of examples.

Example 2.11.2 The sequence

$C_1 = \{S(x), R(f(x))\}$, $C_2 = \{\neg S(x), R(y)\}$, $C_3 = \{R(f(x)), R(y)\}$, $C_4 = \{\neg R(x), Q(x)\}$, $C_5 = \{Q(f(x))\}$

is a derivation of $\{Q(f(x))\}$ from the set of clauses $S = \{\{S(x), R(f(x))\}$, $\{\neg S(x), R(y)\}, \{\neg R(x), Q(x)\}\}$. Let us see why.

First of all, let us check that every clause in the sequence is either a member of S or a resolvent of two preceding clauses.

C_1, C_2, and C_4 are members of S and Figures 2.55, 2.56 show that C_3 and C_5 are resolvents of preceding clauses.

Example 2.11.3 The sequence

$C_1 = \{P(x, y), Q(x)\}, C_2 = \{\neg P(f(x), z)\}, C_3 = \{Q(f(x))\}, C_4 = \{R(x, y), \neg Q(y)\}$, $C_5 = \{\neg R(x, x), \neg Q(f(a))\}, C_6 = \{\neg Q(x), \neg Q(f(a))\}, C_7 = \square$ is a derivation of $C = \square$ from $S = \{\{P(x, y), Q(x)\}, \{\neg P(f(x), z)\}, \{R(x, y), \neg Q(y)\}, \{\neg R(x, x), \neg Q(f(a))\}\}$.

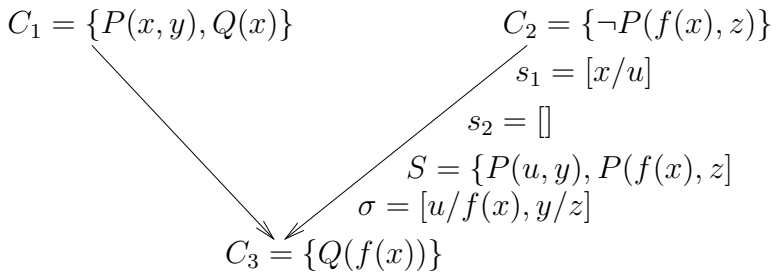


Figure 2.57: C_3 is a resolvent of C_1 and C_2

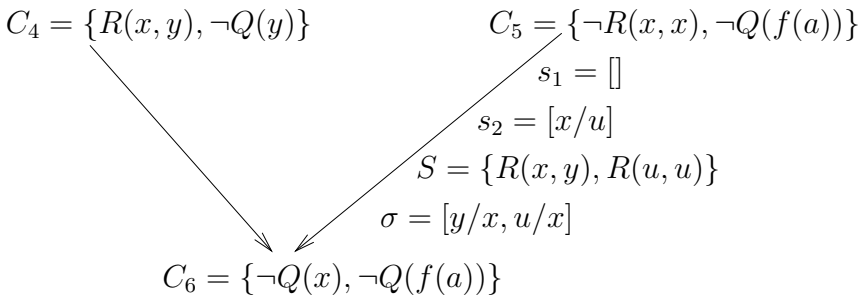


Figure 2.58: C_6 is a resolvent of C_4 and C_5

Let us check that it obeys the 2 conditions of the definition of a derivation sequence. The clauses C_1, C_2, C_4, C_5 are members of S and Figures 2.57, 2.58, 2.59 show the derivations of C_3, C_6, and C_7 from preceding clauses.

The second part of the definition of the derivation sequence is satisfied since $\square$ is the last formula in the sequence.

Observation 2.11.4 Each of the clauses $C_1, \ldots, C_n$ of the derivation sequence stands for the sequence $\overline{C_1}, \ldots, \overline{C_n}$, where C_i is the universal closure of C_i. By The Relabeling Lemma, we can change the variables of $\overline{C_i}$. So, each clause can be *standardized* by replacing the first variable by x_1, the second by x_2, and so

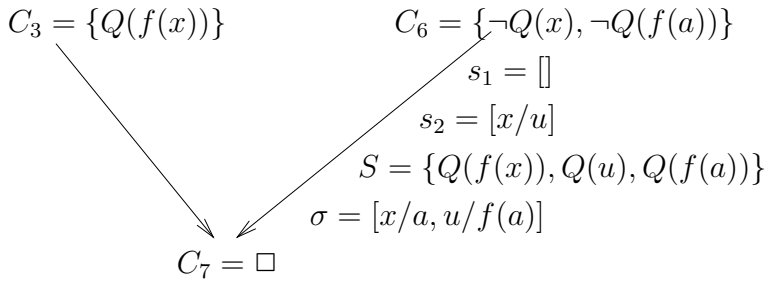


Figure 2.59: C_7 is a resolvent of C_3 and C_6

$C_1 = \{P(x,y), Q(x)\}$ clause in S
$C_2 = \{\neg P(f(x), z)\}$ clause in S
$C_3 = \{Q(f(x))\}$ resolvent of C_1 and C_2 on $P(x,y), \neg P(f(x), y)$
$C_4 = \{R(x,y), \neg Q(y)\}$ clause in S
$C_5 = \{\neg R(x,x), \neg Q(f(a))\}$ clause in S
$C_6 = \{\neg Q(x), \neg Q(f(a))\}$ resolvent of C_4 and C_5 on $R(x,y), \neg R(x,x)$
$C_7 = \square$ resolvent of C_1 and C_2 on $Q(f(x)), \neg Q(x), \neg Q(f(a))$.

Figure 2.60: An analysis for Example 2.11.3

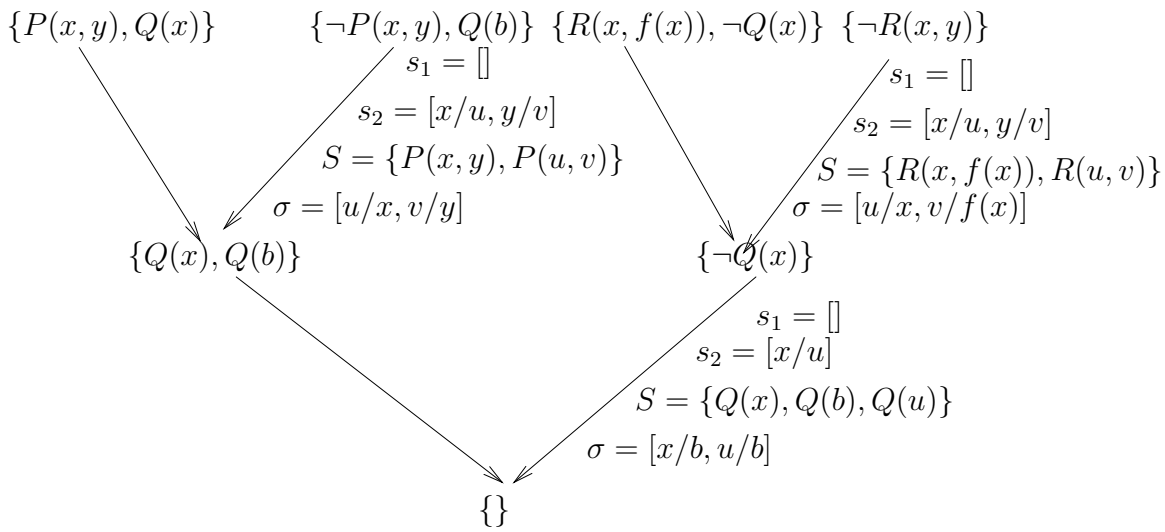


Figure 2.61: An FOL derivation tree

on. We will not do it, but this technique is useful in implementing automatic theorem provers.

The definition of *analysis* is the same as in Section 1.8; it not only lists the clauses, but also describes how they were derived. For each clause the analysis specifies if it belongs to S, or is a resolvent of two preceding clauses. In the later case, it lists its parents. In general, we list the clause that contains the positive literal first. Figure 2.60 shows the analysis of the sequence from Example 2.11.3. The derivation tree definition, introduced in Section 1.8, is valid for FOL. Figure 2.61 shows a derivation tree of $\square$ from the set $S = \{\{\{P(x,y), Q(x)\}, \{\neg P(x,y), Q(b)\}, \{R(x, f(x)), \neg Q(x)\}, \{\neg R(x,y)\}\}$.

Just like we showed in Section 1.8, a clause may have more than one derivation tree and more than one derivation sequence. Figures 2.62 and 2.63 show two derivations of $\{P(f^4(a))\}$ from $S = \{\{P(a)\}, \{\neg P(x), P(f(x))\}\}$. Figures 2.64 and 2.65 show the trees that correspond to these derivations. We simplified the figures by showing only the parents and the resolvents. We say that a derivation of C from S has *minimal length*, or is a *minimal length derivation*, if there

1. $\{P(a)\}$ clause in S
2. $\{\neg P(x), P(f(x))\}$ clause in S
3. $\{P(f(a))\}$ resolvent of 1,2
4. $\{P(f^2(a))\}$ resolvent of 3,2
5. $\{P(f^3(a))\}$ resolvent of 4,2
6. $\{P(f^4(a))\}$ resolvent of 5,2

Figure 2.62: A derivation of $\{P(f^4(a))\}$

1. $\{\neg P(x), P(f(x))\}$ clause in S
2. $\{\neg P(x), P(f^2(x))\}$ resolvent of 1,1
3. $\{\neg P(x), P(f^4(x))\}$ resolvent of 2,2
4. $\{P(a)\}$ clause in S
5. $\{P(f^4(a))\}$ resolvent of 4,3

Figure 2.63: Another derivation of $\{P(f^4(a))\}$

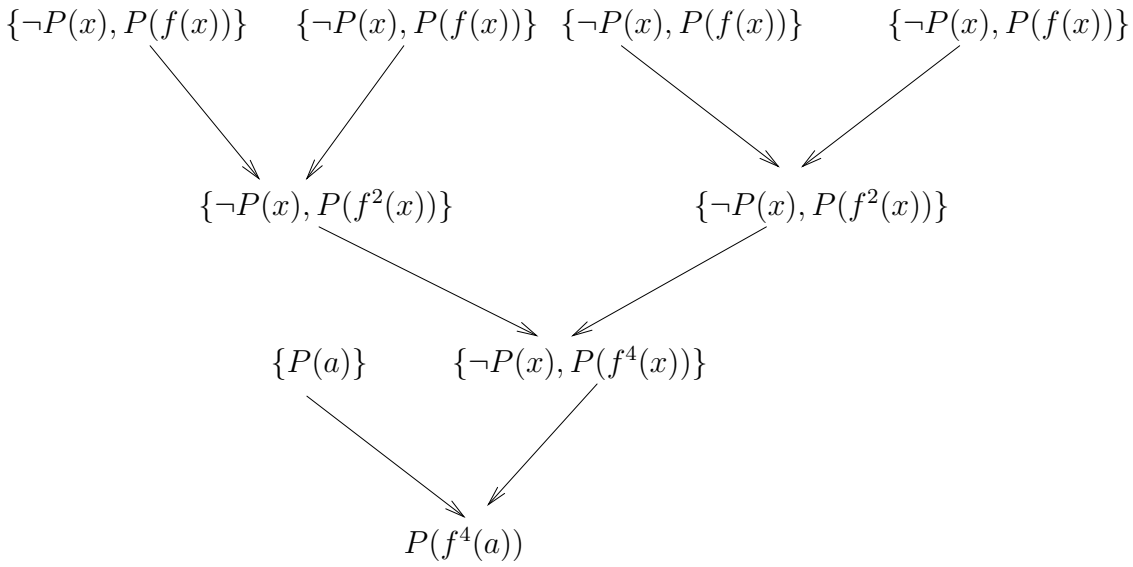


Figure 2.64: A derivation tree of $\{P(f^4(a))\}$

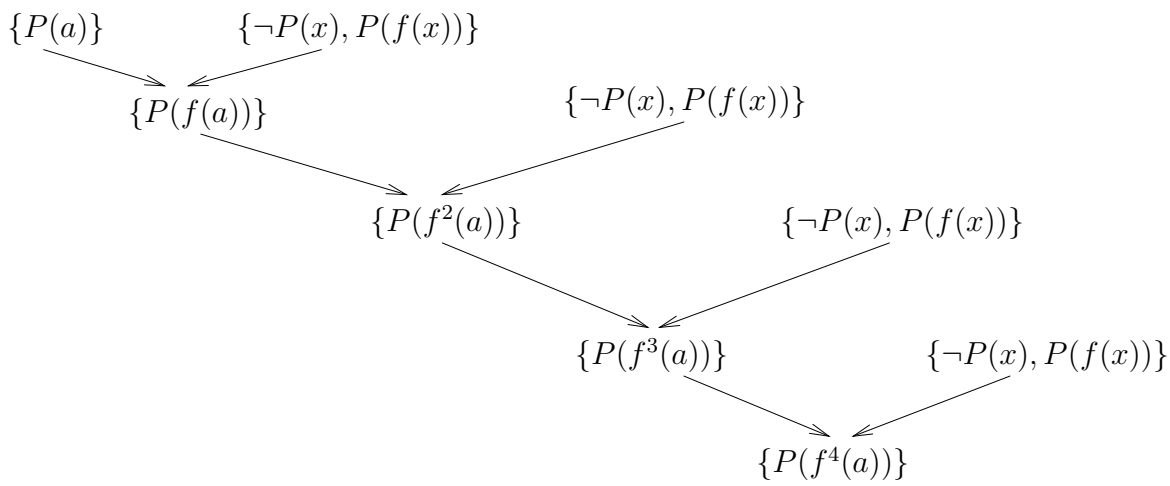


Figure 2.65: Another derivation tree of $\{P(f^4(a))\}$

is no shorter derivation of C from S. In a similar way, a *minimal derivation tree of C from S* has the shortest height of all derivation trees of C from S.

The tree from Figure 2.64 is minimal, and so is the sequence from Figure 2.63, but we have to prove it. We recall, from Example 2.10.6, that $Res^n[\{\{P(a)\}, \{\neg P(x), P(f(x))\}\}] = \{\{P(a)\}, \ldots, \{P(f^{2^n-1}(a))\}\} \cup \{\{\neg P(x),$

$P(f^{2^i}(x))\}|1 \leq i \leq 2^n\}$. So, $\{P(f^4(a))\} \in Res^3[S]$, but $\{P(f^4(a))\} \notin Res^2[S]$. This tells us that the minimal derivation trees of $\{P(f^4(a))\}$ have height 3. So, Figure 2.64 shows a minimal tree. Now let us see how many different clauses are in such a tree. The label of the root is in $Res^3[S] - Res^2[S]$. At least one of its children is in $Res^2[S] - Res^1[S]$, and at least one grandchild is in $Res^2[S] - Res^0[S]$. So, these 3 nodes have different labels. Now we are left with the leaves. The tree must contain both $\{P(a)\}$ and $\{\neg P(x), P(f(x))\}$. If $\{P(a)\}$ is missing, then the root is not a ground clause, and when $\{\neg P(x), P(f(x))\}$ is not present, we cannot get any resolvent because $\{P(a)\}$ cannot be unified with itself. So, a minimal derivation sequence of $\{P(f^4(a))\}$ must have at least 5 clauses. It follows that Figure 2.63 is a minimal derivation.

Now let us return to our main objective, of proving that the clause set S is unsatisfiable iff $\Box \in Res^*[S]$.

The proof of this theorem uses **The Lifting Lemma**.

Lemma 2.11.5 (The Lifting Lemma) *Let C_1 and C_2 be two clauses and s_1 and s_2 two ground substitutions. Let $C_1' = s_1[C_1]$, $C_2' = s_2[C_2]$, and R' be a resolvent of C_1' and C_2'. Then there is a resolvent R of C_1 and C_2 and a ground substitution s such that $R' = s[R]$.*

Before we go on with the proof , let us see what the lemma says by looking at

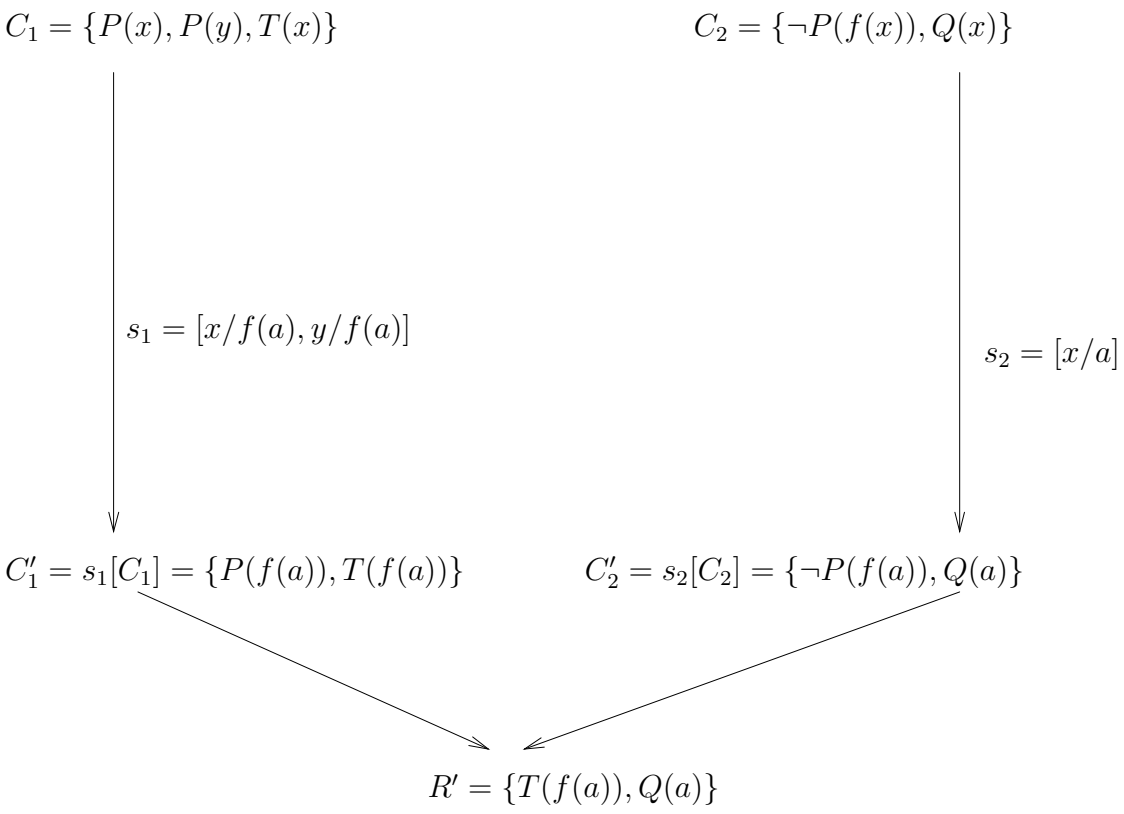


Figure 2.66: The input to The Lifting Lemma

an example.

Example 2.11.6 The input of the lemma consists of the following clauses: C_1, C_2, their ground instances $C_1' = s_1[C_1]$ and $C_2' = s_2[C_2]$, and the resolvent R' of C_1' and C_2'.(Figure 2.66) The instances C_1' and C_2' contain no variables, so they are clauses in the *propositional logic*. The resolvent R' is computed as a resolvent of two propositional clauses.

The Lifting Lemma tells us that we can find a resolvent R of C_1 and C_2, computed according to the rules given in Section 2.10, such that R' is an instance of R. The dash arrows of Figure 2.67 show the output of the lemma. The proof of the lemma tells us how to get the resolvent R and the substitution s. Here, we will show their computation for the clauses from Figure 2.66. From the diagram we see that

$s_1[P(x)] = s_1[P(x)] = s_2[P(f(x))] = P(f(a))$.

This tells us that the literals $P(x)$ and $P(y)$ of C_1 can be unified with the complement of $\neg P(f(x))$. So we compute R. We first relabel the variables of C_1 and C_2, in such a way that they do not have any common variables. We get

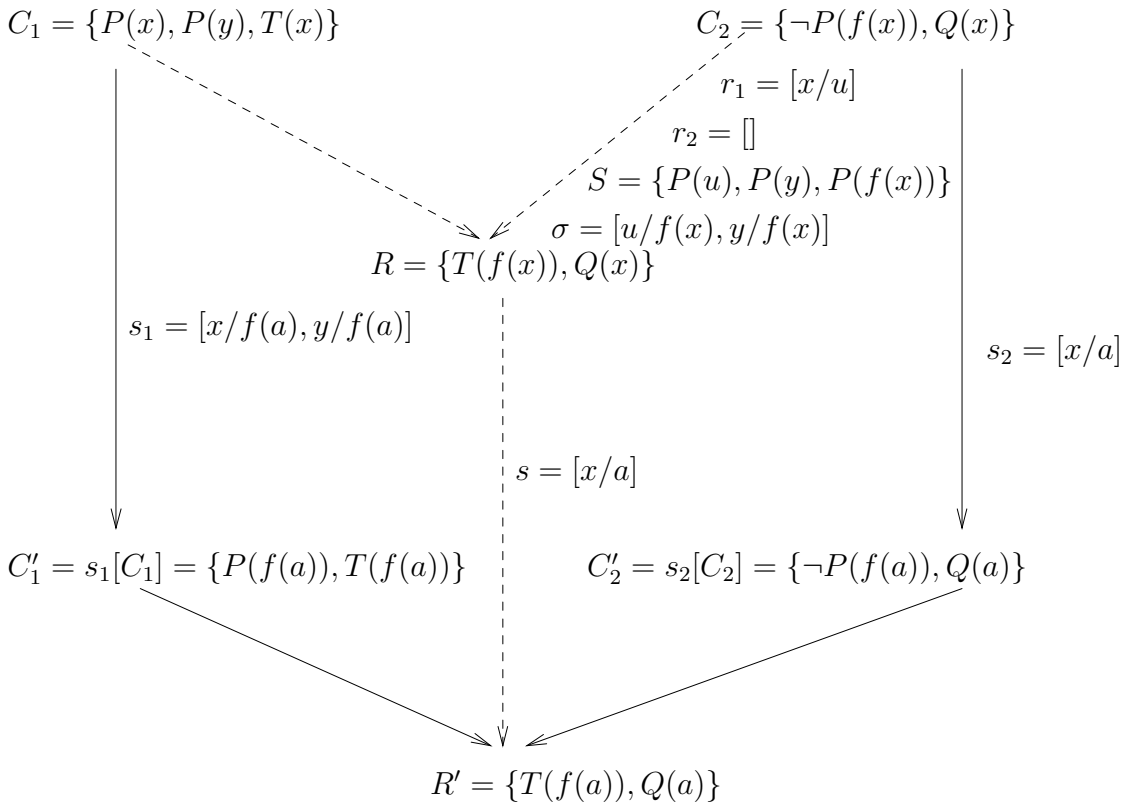


Figure 2.67: The output of The Lifting Lemma

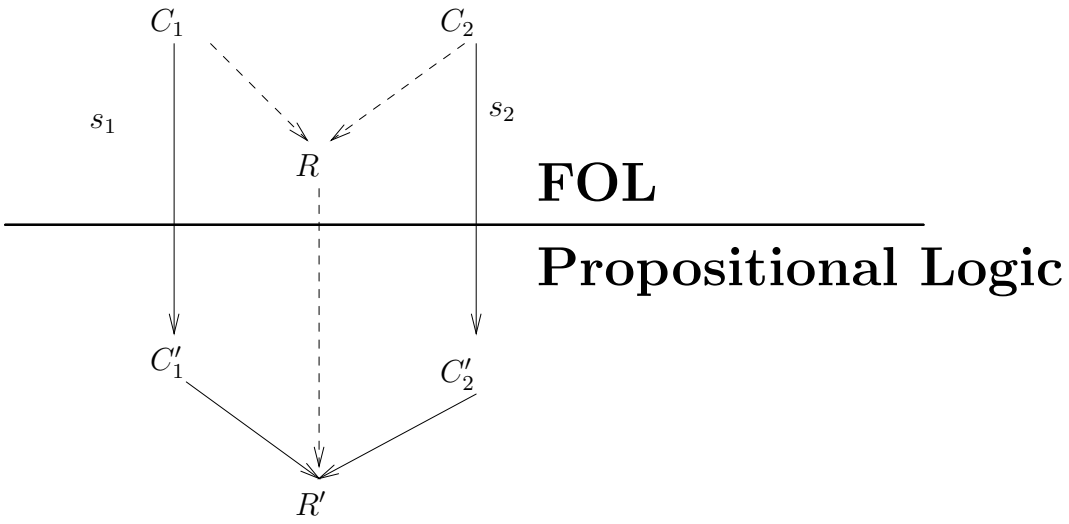


Figure 2.68: The statement of The Lifting Lemma

the relabelings $r_1 = [x/u]$ and $r_2 = []$.

Then we find an mgu of $\{r_1[P(x)], r_1[P(y)], r_2[P(f(x))]\} = \{P(u), P(y), P(f(x))\}$.

We get $\sigma = [u/f(x), y/f(x)]$ and the resolvent

$R = \sigma[(r_1[C_1] - r_1[\{P(x), P(y)\}]) \cup (r_2[C_2] - r_2[\{\neg P(f(x))\}])]$

$= \sigma[\{T(u), Q(x)\}] = \{T(f(x)), Q(x)\}$.

The condition $s[R] = R'$ becomes the equation

$\{T(f(a)), Q(a)\} = s[\{T(f(x)), Q(x)\}$

that is satisfied by $s = [x/a]$.

Now let us return to the proof of The Lifting Lemma.

Proof: The statement of the lemma is presented in Figure 2.68. The solid lines describe the hypotheses of the lemma and the dotted line the conclusion. R' is a resolvent of C'_1 and C'_2, so there are ground literals $B_1, \ldots, B_l, D_1, \ldots, D_m$ and a ground atom A such that

(1) $C'_1 = \{B_1, \ldots, B_l, A\}$,

(2) $C'_2 = \{D_1, \ldots, D_m, \neg A\}$,

and

(3) $R' = \{B_1, \ldots, B_l, D_1, \ldots, D_m\}$.

Since $C'_1 = s_1[C_1]$ and $C'_2 = s_2[C_2]$, there are natural numbers n, o, p and q, and literals $L_1, \ldots, L_n$, $M_1, \ldots, M_o, N_1, \ldots, N_p, O_1, \ldots, O_q$, such that

(4) $C_1 = \{L_1, \ldots, L_n, M_1, \ldots, M_o\}$,

(5) $C_2 = \{N_1, \ldots, N_p, O_1, \ldots, O_q\}$,

(6) $s_1[\{L_1, \ldots, L_n\}] = \{B_1, \ldots, B_l\}$,

(7) $s_2[\{N_1, \ldots, N_p\}] = \{D_1, \ldots, D_m\}$,

(8) $s_1[\{M_1, \ldots, M_o\}] = A$,

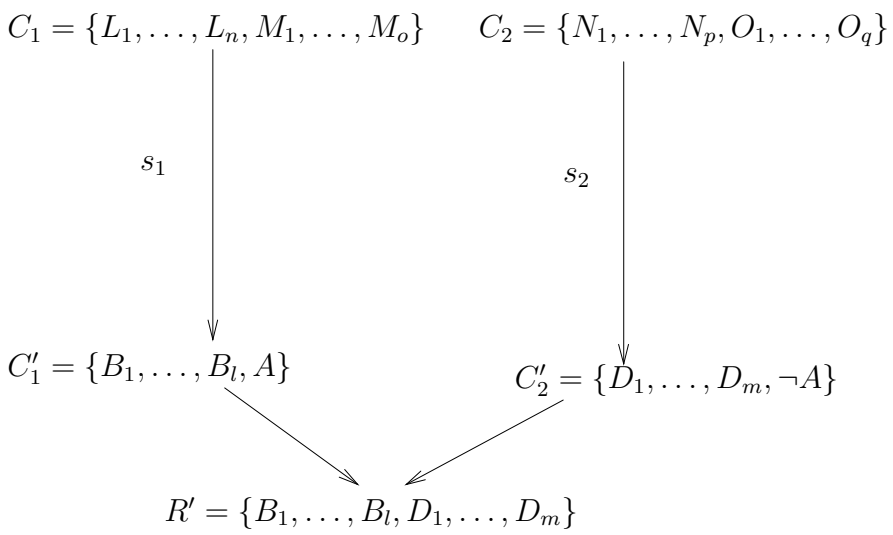


Figure 2.69: The hypotheses of The Lifting Lemma

and

(9) $s_2[\{O_1, \ldots, O_q\}] = \neg A$.

So, we get the diagram from Figure 2.69. Now we find two relabelings, $r_1 = [x_1/y_1, \ldots, x_a/y_a]$ and $r_2 = [u_1/v_1, \ldots, u_b/v_b]$, of the variables of C_1 respectively C_2, such that

(10) $y_1, \ldots, y_a$ are not in C_1,

(11) $v_1, \ldots, v_b$ are not in C_2, and

(12) $r_1[C_1]$ and $r_2[C_2]$ have no variables in common.

The substitutions $r_1^{-1} = [y_1/x_n, \ldots, y_a/x_a]$ and $r_2^{-1} = [v_1/u_1, \ldots, v_b/u_b]$ are called the *inverses* of r_1, respectively r_2. We can show that

(13) $s_1[C_1] = (s_1 \diamond r_1^{-1} \diamond r_1)[C_1]$,

and

(14) $s_2[C_2] = (s_2 \diamond r_2^{-1} \diamond r_2)[C_2]$.

The proof of these equalities is left as exercise.

Then, the equalities (8) and (9) tell us that the set

$S = \{r_1[M_1], \ldots, r_1[M_o], \overline{r_2[O_1]}, \ldots, \overline{r_2[O_q]}\}$

is unifiable since

$((s_1 \diamond r_1^{-1}) \cup (s_2 \diamond r_2^{-1}))[S]$

$= \{(s_1 \diamond r_1^{-1} \diamond r_1)[M_1], \ldots, (s_1 \diamond r_1^{-1} \diamond r_1)[M_o], (s_2 \diamond r_2^{-1} \diamond r_2)[\overline{O_1}], \ldots, (s_2 \diamond r_2^{-1} \diamond r_2)[\overline{O_q}]\}$

$= \{s_1[M_1], \ldots, s_1[M_o], s_2[\overline{O_1}], \ldots, s_2[\overline{O_q}]\}$ by (13), (14)

$= \{A\}$ by (8), (9).

Then S has an mgu σ. Let $R = \{(\sigma \diamond r_1)[L_1], \ldots, (\sigma \diamond r_1)[L_n], (\sigma \diamond r_2)[N_1], \ldots, (\sigma \diamond r_2)[N_p]\}$ be the resolvent of $r_1[C_1]$ and $r_2[C_2]$ generated by S and σ. Figure 2.70 shows how we got the resolvent R. Since σ is an mgu of S, and $((s_1 \diamond r_1^{-1}) \cup (s_2 \diamond r_2^{-1}))$ is a unifier of S, the last substitution factors through σ.

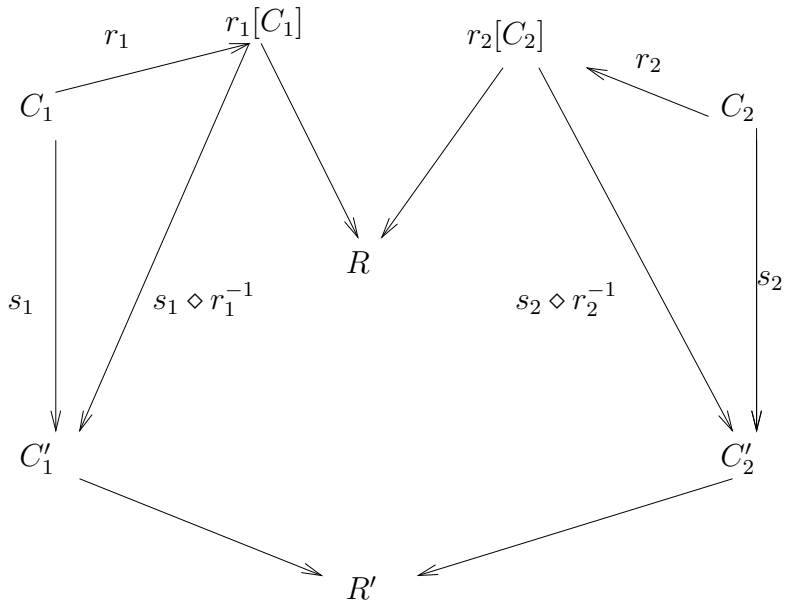


Figure 2.70: The resolvent of The Lifting Lemma

(15) $(s_1 \diamond r_1^{-1}) \cup (s_2 \diamond r_2^{-1}) = \rho \diamond \sigma$.

This means that

$\rho[R] = \rho[\{(\sigma \diamond r_1)[L_1], \ldots, (\sigma \diamond r_1)[L_n], (\sigma \diamond r_2)[N_1], \ldots, (\sigma \diamond r_2)[N_p]\}]$

$= \{(\rho \diamond \sigma \diamond r_1)[L_1], \ldots, (\rho \diamond \sigma \diamond r_1)[L_n], (\rho \diamond \sigma \diamond r_2)[N_1], \ldots, (\rho \diamond \sigma \diamond r_2)[N_p]\}$

$= \{(s_1 \diamond r_1^{-1} \diamond r_1)[L_1], \ldots, (s_1 \diamond r_1^{-1} \diamond r_1)[L_n], (s_2 \diamond r_2^{-1} \diamond r_2)[N_1], \ldots, (s_2 \diamond r_2^{-1} \diamond r_2)[N_p]\}$ by (15)

$= \{s_1[L_1], \ldots, s_1[L_n], s_2[N_1], \ldots, s_2[N_p]\}$ by (13), (14)

$= \{B_1, \ldots, B_l, D_1, \ldots, D_m\}$ by (6), (7)

$= R'$ by (3).

This situation is illustrated in Figure 2.71.

Q.E.D.

Example 2.11.7 Let us apply The Lifting Lemma to the resolution tree from Figure 2.72. The tree of $\Box$ has 3 resolution steps. We can lift them up in any order. Since we do not want to clutter the figures with too much text, we do not list the identity relabelings in the derivation steps. We first lift the resolvent of $\{P(a)\}$ and $\{\neg P(a), P(f(a))\}$ (Figure 2.73). Then we lift the resolvent of $\{\neg P(f(a)), P(f^2(a))\}$ and $\{\neg P(f^2(x))\}$ (Figure 2.74). Finally, we lift the derivation of the box and get Figure 2.75.

The above example explains the name *The Lifting Lemma*; it lifts the derivation trees from the set of ground instances into the set of first order clauses.

Now we can state the Resolution Theorem for First Order Logic.

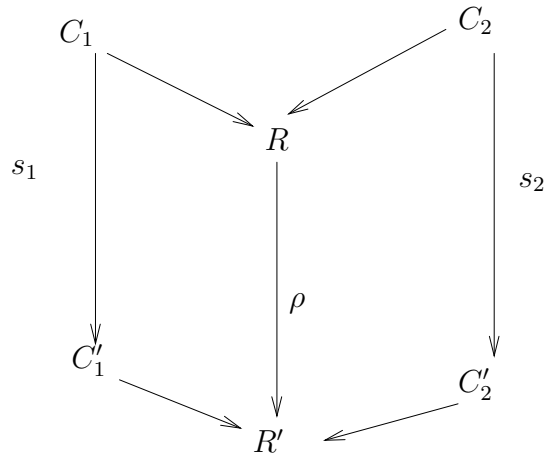


Figure 2.71: The output of The Lifting Lemma

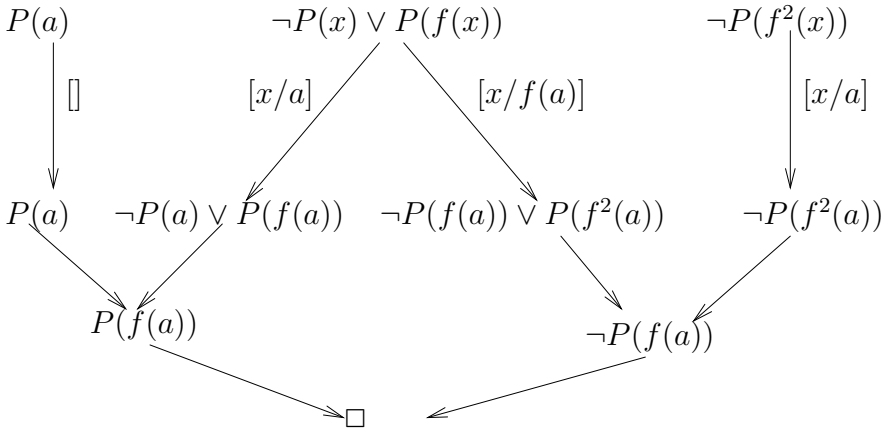


Figure 2.72: The input to Example 2.11.7

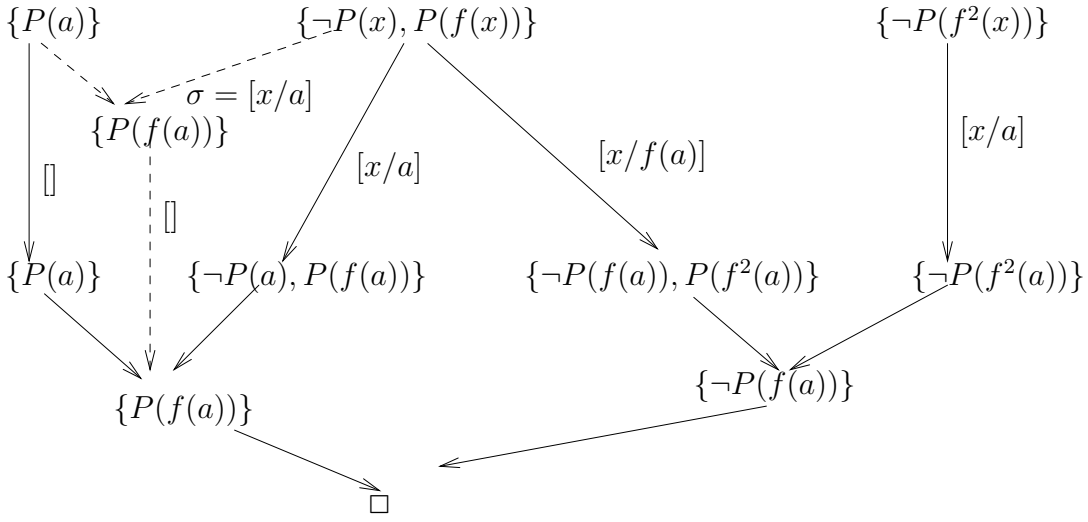


Figure 2.73: Example 2.11.7 after one lift

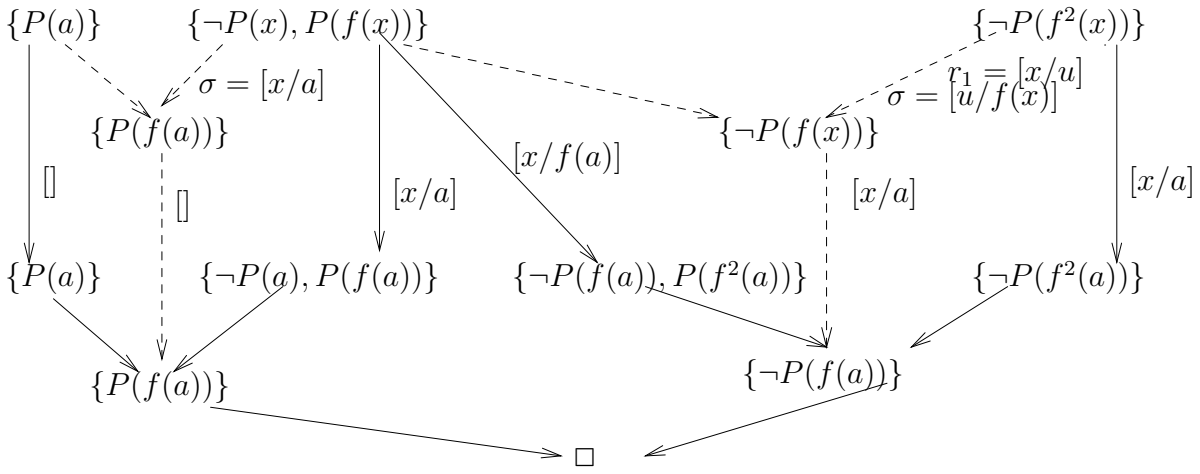


Figure 2.74: Example 2.11.7 after two lifts

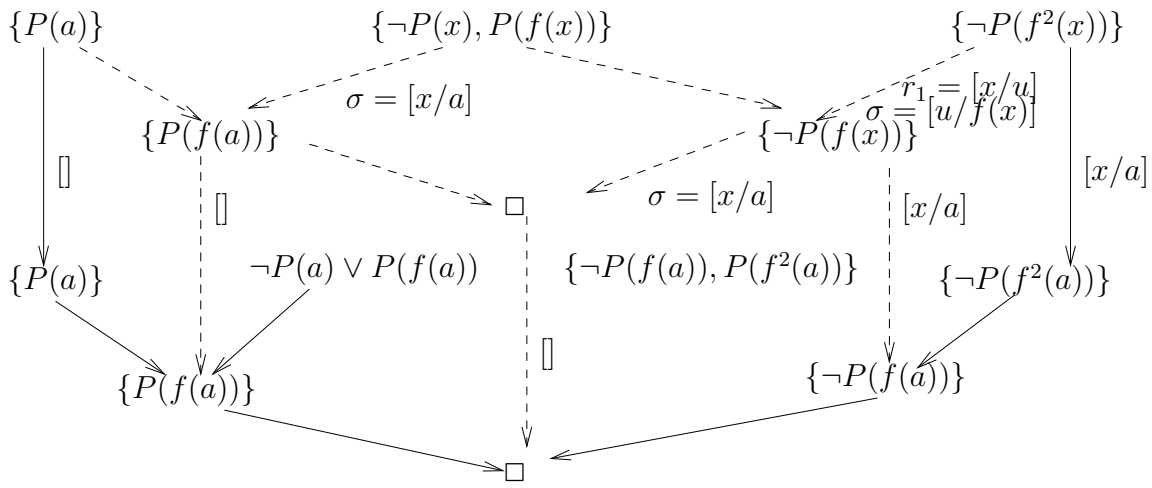


Figure 2.75: Example 2.11.7 after three lifts

Theorem 2.11.8 (The Resolution Theorem for FOL) *Let* $F = \forall x_1 \ldots \forall x_n(C_1 \wedge C_2 \wedge \ldots \wedge C_m)$ *be a Skolem form without* E. *Then* F *is unsatisfiable iff* $\Box \in Res^*[\{C_1, \ldots, C_m\}]$.

Proof: Let $F = \forall x_1 \ldots \forall x_n(C_1 \wedge C_2 \ldots \wedge C_m)$ and $S = \{C_1, \ldots, C_m\}$.
$\Longleftarrow$: Let us assume that $\Box \in Res^*[S]$. Then F is unsatisfiable by Corollary 2.10.11.
$\Longrightarrow$: Assume that F is unsatisfiable. By Proposition 2.8.19, there is some natural number p such that $E(F,p)$, the set of Herbrand expansions of F of height $\leq p$, is unsatisfiable. Since each formula of $E(F,p)$ is a conjunction of ground instances of the clauses $C_1, \ldots, C_m$, $E(F,p)$ is equivalent to the set $S = \{C[x_1/t_1, \ldots, x_n/t_n] | C \in \{C_1, \ldots, C_m\}$ and $t_1, \ldots, t_n \in D(F,p)\}$. The Resolution Theorem for the propositional logic tells us that there is a derivation tree of $\Box$ from the clauses of S. The Lifting Lemma says that we can lift this derivation tree to a derivation tree of $\Box$ from the clauses $C_1, \ldots, C_m$. **Q.E.D.**

We define the P, N, linear, and SDL resolution just like we did for the propositional calculus. We also say that a restriction of resolution is complete iff for every unsatisfiable set of clauses we can derive the empty clause using that restriction.

The Lifting Lemma carries a P-tree onto a P-tree, an N-tree onto an N-tree, and a linear tree onto a linear tree. These restrictions are complete for the propositional calculus, so they are complete for the first order calculus.

Exercises

Exercise 2.11.1 *Let* $S = \{\{P(a)\}, \{\neg P(x), P(f(x))\}\}$.
a. Write a minimal derivation of $\{P(f^{15}(a))\}$.

b. Express the length of a minimal derivation of $\{P(f^n(a))\}$ *from* S *as a function of* n.

Exercise 2.11.2 *Rewrite the trees from Figures 2.64 and 2.65 according to the format used in Figure 2.61. For each resolution step specify the relabeling* s_1, s_2, *the unifying set* S *and the unifier* σ.

Exercise 2.11.3 *Let* s *be a substitution,* C *a clause and* $r = [x_1/y_1, \ldots, x_n/y_n]$ *be a relabeling away from* $Var[C]$. *We define* r^{-1} *to be the substitution* $[y_1/x_1, \ldots, y_n/x_n]$. *Show that* $s[C] = (s \circ r^{-1} \circ r)[C]$.

Exercise 2.11.4 *Let* $F = \forall x_1 \ldots \forall x_n (C_1 \wedge C_2 \ldots \wedge C_m)$ *be a Skolem form without equality. Show that* $E(F, n) \equiv \{C_i[x_1/t_1, \ldots, x_n/t_n] | 1 \leq i \leq n, t_1, \ldots, t_n \in D(F, n)\}$.

Exercise 2.11.5 *Prove The Compactness Theorem for FOL: A set of formulas* S *is unsatisfiable iff it has a finite unsatisfiable subset.*

Hint: Show that S *is satisfiably equivalent to a set* T *of Skolem Forms. Then prove that* T *is unsatisfiable iff its Herbrand expansion* $E(T)$ *is unsatisfiable. Apply The Resolution Theorem for propositional calculus and get a derivation tree of* $\square$ *from* $E(T)$. *Then use The Lifting Lemma to get derivation of* $\square$ *from* T. *So, the set of clauses of* S *that produced the clauses of the tree is unsatisfiable.*

Exercise 2.11.6 *Prove that The Lifting Lemma diagram from Figure 2.68 satisfies the following statements:*

1. C_1 *is a* P*-clause iff* C_1' *is a* P*-clause.*
2. C_2 *is an* N*-clause iff* C_2' *is an* N*-clause.*
3. *if* C_1 *and* C_2 *are Horn clauses then* C_1', C_2', R', R *are Horn clauses.*

Exercise 2.11.7 *Find a derivation tree of* $\square$ *from* $S = \{\{P(x,a), P(y,x), Q(x,f(x))\}, \{\neg P(x,y), Q(x,z)\}, \{\neg Q(x,f(y)), R(y,x)\}, \{\neg R(x,a), \neg R(a,y)\}\}$.

Exercise 2.11.8 *Use resolution to show that the set* $S = \{\neg P(x,y), \neg P(y,z), P(x,z)\}, \{\neg P(x,y), P(y,x)\}, \{P(f(x,e),e)\}, \{P(f(d,x),d)\}, \{\neg P(d,e)\}\}$ *is unsatisfiable.*

Exercise 2.11.9 *Derive* $\square$ *from the set of clauses* $S = \{\{P(x,f(y)), P(x,z), Q(x,z)\}, \{\neg P(g(x),y), Q(z,y)\}, \{\neg Q(x,y), \neg Q(g(z),y), R(x,y)\}, \{\neg R(x,f(y)), \neg R(g(u),f(v))\}\}$.

Exercise 2.11.10 *Use P-resolution to show that the set of clauses* $S = \{\{P(f(x),y), P(y,f(z)), Q(g(y),y)\}, \{\neg P(f(x),y), Q(g(y),z)\}, \{\neg Q(x,f(y)), \neg Q(g(u),v), \neg R(x,v)\}, \{R(x,f(a)), R(g(y),y)\}\}$ *is unsatisfiable.*

Exercise 2.11.11 *Use N-resolution to show that the set of clauses* $S = \{\{P(x,f(x)), P(x,y), Q(y), R(x,y)\}, \{\neg P(x,y), S(x)\}, \{\neg Q(f(x))\}, \{P(x,y), Q(f(x)), \neg R(x,f(x))\}, \{\neg P(x,y), \neg S(g(z))\}\}$ *is unsatisfiable.*