Logic and automated proof:
An introduction to propositional logic and first-order logic

Stéphane Devismes Pascal Lafourcade Michel Lévy
Translation by Benjamin Wack assisted by Libretranslate

2024/2025

Contents

Introduction

MATHEMATICIANS have reasoned for centuries *without knowing* formal logic. Logic was born of the ambition to formalize (automatize) mathematical mathematical reasoning: this project received a decisive kickstart when it became clear that the absence of formalization leads to contradictions. The current development of logic continues with

— the need to prove the correctness of programs (especially when these programs are used in areas where security is at stake);

— the ambition to represent all mathematical knowledge into computers.

However, the terminology used by mathematicians and logicians differs from that of logicians. To express that p implies q, the mathematician would say, for example:

p entails q

p is a sufficient condition for q

for q to be true, it is sufficient for p to be true

q is a necessary condition for p

for p to be true, it is necessary for q to be true

We will restrict our study to *classical* logic (as opposed to intuitionistic logic). Classical logic is the logic with two truth values. In addition, it is the logic of combinatorial circuits, which explains its great practical importance.

Plan : In the first part, we introduce propositional logic. More precisely, in the first chapter we give the basic definitions and results of statement logic. In the second chapter, we talk about propositional binary resolution and the `DPLL` algorithm. Finally, we illustrate a method of logical reasoning by presenting natural deduction. In the second part of the course, we revisit all the concepts, results and techniques presented in the first part for first-order logic.

Exercises: At the end of each chapter, we offer a series of exercises on the chapter's content. The difficulty of the proposed exercises is denoted with stars: the more stars, the more difficult the exercise. We indicate by ⟳ the exercises that complete the proofs seen during the course. The exercises are divided into three categories. We have:

— Basic exercises, which help students familiarize themselves with the vocabulary and manipulate concepts introduced in class.

— Technical exercises, which are direct or immediate applications of results seen in class.

— Reflexive exercises, which enable the learner to reason, demonstrate and prove results based on the techniques and notions learned through the other two types of exercise.

Objectives: The skills and knowledge we want to impart are the following :

— *Understand a reasoning* : be able to determine whether a logical reasoning is correct or not.

— *Reasoning*, i.e., constructing a correct reasoning using the tools of propositional and first-order logic.

— *Model and formalize a problem*.

— *Write a rigorous proof*.

Part I

Propositional logic

Chapter 1

Propositional logic

Contents

ARISTOTELES was one of the first to try to formalize reasoning using the logic of syllogisms. The logic is used to specify what is a correct reasoning, regardless of the field of application. Reasoning is a means of reaching a conclusion from given hypotheses. *Correct* reasoning doesn't tell anything about the truth of the assumptions, it only states that *assuming the truth of the hypotheses, we can deduce the truth of the conclusion*. We begin with the laws of propositional logic. Propositional logic is the logic *without quantifiers*, which is concerned solely with the laws governing the the following logical operations: negation ($\neg$), conjunction, in other words "and" ($\wedge$), disjunction, otherwise known as "or" ($\vee$), implication ($\Rightarrow$) and equivalence ($\Leftrightarrow$). These operations are also called *connectives*. Propositional logic is used to construct reasonings using these connectives. Let us consider the following example, which involves three hypotheses:

1. if Peter is tall, then John is not Peter's son;
2. if Peter is not tall, then John is Peter's son;
3. if John is Peter's son then Mary is John's sister.

We conclude that Mary is John's sister or Peter is tall.

In order to reason, we extract the logical structure of the assumptions. We represent the sentences "Peter is tall", "John is Peter's son" and "Mary is John's sister" respectively by the letters p, j, m. The hypotheses can therefore be written:

1. $p \Rightarrow \neg j$,
2. $\neg p \Rightarrow j$,
3. $j \Rightarrow m$,

and the conclusion is formalized as $m \vee p$. We then show that the hypotheses imply the conclusion independently of the nature of the statements p, j, m. To do this, we prove that the following formula is true whatever the truth value of the propositions p, j, m.

$$((p \Rightarrow \neg j) \wedge (\neg p \Rightarrow j) \wedge (j \Rightarrow m)) \Rightarrow (m \vee p).$$

Outline: We begin this chapter with the *syntax of logical formulas*, i.e., the rules for writing formulas. A formula can be true or false, so we need to be able to evaluate the *meaning of a formula*. To do this, we introduce the meaning of each connective and how to compute the value of a formula derived from it. Next, we present some *important equivalences* useful for simplifying logical logical reasonings. Other methods may simplify logical reasonings, such as formula *replacement* and *substitution*. We then show how to construct conjunctive or disjunctive normal forms of a formula using remarkable equivalences. These normal forms make it easy to show the models or counter-models of a formula. We then show that propositional logic is an instance of a Boolean algebra. We introduce the notion of *Boolean functions*. Finally, we briefly present the tool `BDDC`[1] developed by Pascal Raymond. This tool automatizes some manipulations of propositional formulas.

1.1 Syntax

Before we start reasoning, let us define the language we're using. This language is that of formulas constructed from the *vocabulary*:

— The constants: $\top$ and $\bot$ representing respectively *true* and the *false*.
— The variables: a variable is an identifier, with or without a subscript, for example x, y_1.
— The parentheses: opening (and closing).
— The connectives: $\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow$ respectively called negation, disjunction (or), conjunction (and), implication and equivalence.

Syntax defines the rules for constructing a propositional logic formula. We introduce two ways of writing a formula, one strict, the other more flexible in the sense that it allows several expressions of the same formula. This flexibility is achieved by introducing priorities between logical connectives.

1.1.1 Strict formulas

We give the rules for constructing a strict formula from the vocabulary given above.

Definition 1.1.1 (Strict formula) *A* strict formula *is defined inductively as follows:*
— $\top$ *and* $\bot$ *are strict formulas.*
— *A variable is a strict formula.*
— *If A is a strict formula, then* $\neg A$ *is a strict formula.*
— *If A and B are strict formulae and if circ is one of the of the operations* $\vee, \wedge, \Rightarrow, \Leftrightarrow$ *then* $(A \circ B)$ *is a strict formula.*

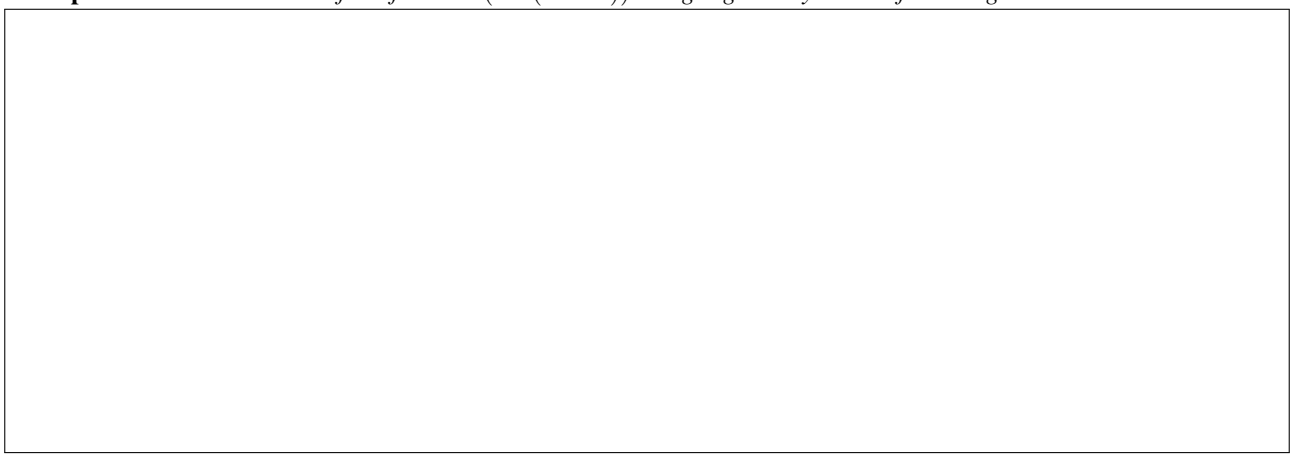
In the following, we'll refer to any binary connective as *circ*, and simply we simply call "formula" a strict formula. Formulas other than $\top$, $\bot$ and variables are decomposable formulas.

Example 1.1.2 *The expression* $(a \vee (\neg b \wedge c))$ *is a strict formula constructed according to the previous rules. On the other hand,* $a \vee (\neg b \wedge c)$ *and* $(a \vee (\neg(b) \wedge c))$ *are not formulas in the sense of definition 1.1.1.*

The advantage of defining strict formulas is that parentheses make it possible to find the structure of the formulas unambiguously. formulas. We represent the structure of formulas with a tree, where the leaves contain the constants or variables and the nodes contain the logical connectives. The root node is the connective to be applied last.

[1] `http://www-verimag.imag.fr/~raymond/home/tools/bddc/`

Example 1.1.3 *The structure of the formula* $(a \vee (\neg b \wedge c))$ *is highlighted by the the following tree:*



A formula can be seen as a sequence of symbols (connectives, parentheses, variables and constants). A factor of such a list is a sequence of consecutive symbols in the list.

Definition 1.1.4 (Subformula) *We call* subformula *of a (strict) formula A any factor of A which is a (strict) formula.*

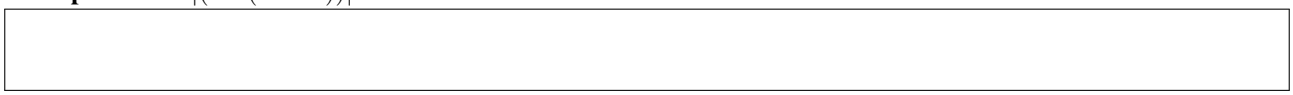
Example 1.1.5 $(\neg b \wedge c)$ *is a subformula of* $(a \vee (\neg b \wedge c))$.

We show that formulas are uniquely decomposable into their subformulas. This result, clarified by theorem 1.1.13, is obvious on the examples. The uniqueness of the decomposition implies that we can identify a formula and its decomposition tree. Thus, a subformula of formula A can be identified as a subtree of the tree representing formula A.

In order to reason inductively about the structure of a formula, we define the size of a formula. define the size of a formula. We note that the size of a formula corresponds to the number of connectives it contains.

Definition 1.1.10 (Formula size) *The* size of a formula *A, noted* $|A|$*, is defined inductively by :*
- $|\top| = 0$ *and* $|\bot| = 0$.
- *If A is a variable, then* $|A| = 0$.
- $|\neg A| = 1 + |A|$.
- $|(A \circ B)| = |A| + |B| + 1$.

Example 1.1.11 $|(a \vee (\neg b \wedge c))| =$



The *size* of formulas defined in definition vrefdef:size is a useful measure for proving properties of formulas by induction.

Theorem 1.1.13 *For any formula A, one and only one of these cases applies:*
- *A is a variable,*
- *A is a constant,*
- *A can uniquely be written as* $\neg B$ *where B is a formula,*
- *A can uniquely be written as* $(B \circ C)$ *where B and C are formulas.*

With the definition of (strict) formulas, we write a lot of unnecessary brackets, such as the parentheses that surround each formula. We now introduce more flexibility into our syntax by defining priorities.

1.1.2 Prioritized formulas

To avoid an overabundance of parentheses, we define *prioritized formulas*.

Definition 1.1.14 (Prioritized formula) *A prioritized formula is defined inductively by :*
- $\top$ *and* $\bot$ *are prioritized formulas,*
- *a variable is a prioritized formula,*

— *if A is a prioritized formula then $\neg A$ is a prioritized formula,*
— *if A and B are prioritized formulas, then $A \circ B$ is a prioritized formula.*
— *if A is a prioritized formula then (A) is a prioritized formula.*

Example 1.1.15 *Let's consider the formula $a \vee \neg b \wedge c$ which is a prioritized formula but not a formula.*

In general, a prioritized formula is not a (strict) formula. We show in exercise 2 that any formula is a prioritized formula. In order to be able to remove parentheses without ambiguity we define an order of precedence between the different connectives.

Definition 1.1.16 (Order of precedence of connectives) *Negation has highest precedence, then in descending order of precedence we find the conjunction ($\wedge$), the disjunction ($\vee$) disjunction ($\vee$), implication ($\Rightarrow$) and equivalence ($\Leftrightarrow$).*

When precedence is equal, the left connective takes precedence, **except for implication** *which is right-associative.* [2]

We consider a prioritized formula to be *the abbreviation* of the formula that can be reconstituted using priorities. With a few exceptions we identify a formula and its abbreviation. In other words, what interests us in a formula is not how it's written, it's its *structure*, which is highlighted by the "strict" syntax. So the size of a prioritized formula will be equal to the size of the strict formula it abbreviates. Similarly, for sub-formulas, we'll always consider the strict formula for which the prioritized formula is an abbreviation.

Example 1.1.17 *We give several examples of formula abbreviation by a prioritized formula: with priority:*
— *$a \wedge b \wedge c$ is an abbreviation of*

— *$a \wedge b \vee c$ is an abbreviation of*

— *$a \vee b \wedge c$ is an abbreviation of*

Now that we've defined the syntax, let's define the meaning of formulas.

1.2 Meaning of formulas

We seek to determine whether a formula is true or false regardless of the values assigned to its variables. We first define the meaning of the logical connectives. We then explain how to compute the value of a formula. We end this section with the definitions of basic logical concepts that constitute the common language of logicians.

1.2.1 Meaning of connectives

We denote the truth values by 0 for false and 1 for true. The constant $\top$ evaluates to 1 and the constant $\bot$ evaluates to 0, so we often assimilate the constants and their values, and use $\top$, 1 and true interchangeably, respectively $\bot$, 0 and false. The meaning of logical connectives is given by the table 1.1 on the next page, which indicates the values of the formulas in the first line according to the values *assigned* to variables x and y.

1.2.2 Value of a formula

Everyone knows how to evaluate a formula: to each variable in the formula we assign a value in the set $\mathbb{B} = \{0,1\}$. The value of the formula is obtained by replacing the variables by their values and performing the operations according to table 1.1 on the facing page. Nevertheless, to reason about formulas, we formally define the value of a formula.

[2] That is, $a \Rightarrow b \Rightarrow c$ is the abbreviation of $(a \Rightarrow (b \Rightarrow c))$.

x	y	$\neg x$	$x \vee y$	$x \wedge y$	$x \Rightarrow y$	$x \Leftrightarrow y$
0	0					
0	1					
1	0					
1	1					

Table 1.1: Truth table of connectives.

Definition 1.2.1 (Assignment) *An* assignment *is an application from the set of all variables in a formula to the set* $\mathbb{B}$.

Definition 1.2.2 (Value of a formula) *Let A be a formula and v an assignment,* $[A]_v$. *denotes the value of formula A in assignment v. The* value *of* $[A]_v$ *is defined by induction on the set of formulas. Let A, B be formulas, x a variable and v an assignment.*

- $[x]_v = v(x)$.
- $[\top]_v = 1$, $[\bot]_v = 0$.
- $[\neg A]_v = 1 - [A]_v$ *i.e. to calculate the value of* $\neg A$*, we subtract the value of A from* 1.
- $[(A \vee B)]_v = \max\{[A]_v, [B]_v\}$.
- $[(A \wedge B)]_v = \min\{[A]_v, [B]_v\}$.
- $[(A \Rightarrow B)]_v =$ *if* $[A]_v = 0$ *then* 1, *otherwise* $[B]_v$.
- $[(A \Leftrightarrow B)]_v =$ *if* $[A]_v = [B]_v$ *then* 1, *oterwise* 0.

According to theorem 1.1.13 on page 13, any (strict) formula can be uniquely decomposed into one of the above cases. So the extension of v to formulas is an application from formulas to $\mathbb{B}$. Indeed, let 4 formulas A, A', B, B' and two operations operations $\circ$ and $\circ'$ such that $(A \circ B) = (A' \circ' B')$. By uniqueness of decomposition, $A = A', B = B', \circ = \circ'$, so the value of the formula $(A \circ B)$ is defined by one and only one of the lines of the definition of value. Clearly, the value of a formula depends only on its variables and its structure, so we can present the evaluation of a formula as a *truth table*.

Definition 1.2.3 (Truth table of a formula) *A* truth table of a formula *A is an array of values that represents the value of A for all possible values of the variables in A.*

Each row of the truth table defines an assignment for the variables of the formulas present in the table's columns, and each column gives the value of a formula.

Example 1.2.4 *We give the truth table for the following formulas:* $x \Rightarrow y$, $\neg x$, $\neg x \vee y$, $(x \Rightarrow y) \Leftrightarrow (\neg x \vee y)$ *and* $x \vee \neg y$.

x	y	$x \Rightarrow y$	$\neg x$	$\neg x \vee y$	$(x \Rightarrow y) \Leftrightarrow (\neg x \vee y)$	$x \vee \neg y$
0	0					
0	1					
1	0					
1	1					

1.2.3 Definitions and elementary notions of logic

We list the elementary notions of logic. We illustrate with examples and counter-examples each of the definitions to illustrate its subtleties.

Definition 1.2.5 (Equivalent formulas) *Two formulas A and B are* equivalent *if they have the same value for every assignment.*

Example 1.2.6 *The columns of the two formulas* $x \Rightarrow y$ *and* $\neg x \vee y$ *are identical in the example 1.2.4. These two formulas are therefore equivalent. On the other hand, the formulas* $x \Rightarrow y$ *and* $x \vee \neg y$ *are not equivalent because they don't have the same truth tables.*

Remark 1.2.7 *We don't use the symbol of the logical connective $\Leftrightarrow$ to express that A and B are equivalent. We denote that the formulas A and B are equivalent by $A \equiv B$ or simply $A = B$ if we understand from the context that the equal sign indicates equivalence. Thus, $x \Rightarrow y = \neg x \vee y$ means that the formula $x \Rightarrow y$ is equivalent to $\neg x \vee y$.*

Definition 1.2.8 (Valid, tautology) *A formula is* valid *if it has the value* 1 *for every assignment. A valid formula is also called a* tautology.

Example 1.2.9 *Looking at the truth table of the example 1.2.4 on the previous page we obtain that:*
- *the formula $(x \Rightarrow y) \Leftrightarrow (\neg x \vee y)$ is valid;*
- *the formula $x \Rightarrow y$ is invalid because*

Notation: $\models A$ denotes the fact that formula A is valid. We can write $models x \vee \neg x$, because $x \vee \neg x$ is a tautology.

Property 1.2.10 *The formulas A and B are equivalent if and only if the formula $A \Leftrightarrow B$ is valid.*

Proof : This property is a consequence of the table 1.1 on the preceding page and the previous definitions.

$\Rightarrow$ If formulas A and B are equivalent, this means that they have the same truth table, so according to the the definition of the $\Leftrightarrow$ connective given in the table 1.1 on the previous page the truth table of $A \Leftrightarrow B$ contains only 1s, so $A \Leftrightarrow B$ is valid.

$\Leftarrow$ If the formula $A \Leftrightarrow B$ is valid, we deduce that the truth table of $A \Leftrightarrow B$ contains only 1s, so according to the definition of the connective $\Leftrightarrow$ given in the table 1.1 on the preceding page the truth tables for A and B are identical. Therefore the formulas A and B are equivalent.

□

Definition 1.2.11 (Model of a formula) *An assignment v that yields the value* 1 *for a formula is a* model *of the formula. We'll also say that v* satisfies *the formula or that v makes the formula* true.

Example 1.2.12 *$x \Rightarrow y$ has a model*

This notion of model extends to formula sets as follows.

Definition 1.2.13 (Model of a set of formulas) *An assignment is a* model of a set of formulas *if and only if it is a model of every formula in the set.*

Example 1.2.14

Property 1.2.15 *An assignment is a model of a set of formulas if and only if it is a model of the conjunction of the formulas in the set.*

Proof : The proof is requested in Exercise 11 on page 32. □

Example 1.2.16 *The set of formulas $\{a \Rightarrow b, b \Rightarrow c\}$ and the formula $(a \Rightarrow b) \wedge (b \Rightarrow c)$ have the same models.*

Definition 1.2.17 (Counter-model) *An assignment v that gives the value* 0 *to a formula is a* counter-model *of the formula. We'll say that v doesn't satisfy the formula or v makes the formula* false.

Example 1.2.18 *$x \Rightarrow y$ has a counter-model :*

Remark 1.2.19 (Counter-model of a set of formulas) *The notion of counter-model extends to sets of formulas in the same way as the notion of model.*

Definition 1.2.20 (Satisfiable formula) *A formula (respectively a set of formulas) is* satisfiable *if there exists an assignment that is a model for it.*

Definition 1.2.21 () *A formula (respectively a set of formulas) is* unsatisfiable *if it is not satisfiable.*

An unsatisfiable formula (respectively set of formulas) has no model. Its validity table contains only 0s. The negation of a tautology is therefore an unsatisfiable formula.

Example 1.2.22

.

Remark 1.2.23 *Logicians use the word* consistent *as a synonym for satisfiable and* contradictory *as a synonym for unsatisfiable.*

Definition 1.2.24 (Consequence) *Let Γ be a set of formulas and A a formula: A is a* consequence *of the set of hypotheses Γ if every model of Γ is a model of A. The fact that A is a consequence of Γ is denoted by $\Gamma \models A$.*

Example 1.2.25 *According to the following truth table, the formula $a \Rightarrow c$ is a consequence of the assumptions $a \Rightarrow b$ and $b \Rightarrow c$.*

a	b	c	$a \Rightarrow b$	$b \Rightarrow c$	$a \Rightarrow c$
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Remark 1.2.26 (Validity and consequence) *We denote A is valid by $\models A$, because A is valid if and only if A is a consequence of the empty set.*

Now let's establish the equivalence of the validity of a formula composed of hypotheses and a conclusion with the consequence of the conclusion from the assumptions, but also with the insatisfiability of the assumptions and the negation of the conclusion. These relationships are used extensively in exercises and demonstrations.

Property 1.2.27 *Let $A_1, \ldots, A_n$, B be $n+1$ formulas. Let H_n be the* conjunction *of formulas $A_1, \ldots, A_n$. These three formulations are equivalent:*

1. *$A_1, \ldots, A_n \models B$, i.e. B is a consequence of assumptions $A_1, \ldots, A_n$.*
2. *The formula $H_n \Rightarrow B$ is valid.*
3. *$H_n \wedge \neg B$ is unsatisfiable.*

Proof : The property is a consequence of table 1.1 on page 15 and previous definitions. Let $A_1, \ldots A_n, B$ be $n+1$ formulas and $H_n = A_1 \wedge \ldots \wedge A_n$.

1 ⇒ 2 : Assume $A_1,\ldots,A_n \models B$, i.e. any model of $A_1,\ldots,A_n$ is also a model of B.

So we get in all cases that $H_n \Rightarrow B$ is valid.

2 ⇒ 3 : Suppose that $H_n \Rightarrow B$ is valid, this means that for any assignment v, $[H_n \Rightarrow B]_v = 1$. Based on table 1.1 on page 15 we have two possibilities: either $[H_n]_v = 0$, or $[H_n]_v = 1$ and $[B]_v = 1$. But $[H_n \wedge \neg B]_v = \min([H_n]_v, [\neg B]_v) = \min([H_n]_v, 1-[B]_v)$. In both cases, we get $[H_n \wedge \neg B]_v = 0$. We conclude that $H_n \wedge \neg B$ is unsatisfiable.

3 ⇒ 1 : Suppose $H_n \wedge \neg B$ is insatisfiable, i.e. for any assignment formula $H_n \wedge \neg B$ is contradictory. Let us show that the models of $A_1,\ldots,A_n$ are also models of B.

Let v be an assignment model of $A_1,\ldots,A_n$, it follows that $[A_i]_v = 1$ for $i = 1,\ldots,n$ so $[H_n]_v = [A_1 \wedge \ldots \wedge A_n]_v = 1$. According to our hypothesis, we deduce that $[\neg B]_v = 0$. Hence, $1-[B]_v = 0$. So we have $[B]_v = 1$, i.e. v is also a model of B.

Using the result of exercise **??** on page ?? we conclude. □

Example 1.2.28 *We consider both formulas $a \Rightarrow b$ and $b \Rightarrow c$, we illustrate the previous theorem by proving that $a \Rightarrow b, b \Rightarrow c \models a \Rightarrow c$, showing either that $(a \Rightarrow b) \wedge (b \Rightarrow c) \Rightarrow (a \Rightarrow c)$ is a tautology, or that $(a \Rightarrow b) \wedge (b \Rightarrow c) \wedge \neg(a \Rightarrow c)$ is unsatisfiable. To that effect we give the truth table associated with these formulas.*

a	b	c	$a \Rightarrow b$	$b \Rightarrow c$	$a \Rightarrow c$	$(a \Rightarrow b) \wedge (b \Rightarrow c) \Rightarrow (a \Rightarrow c)$	$(a \Rightarrow b) \wedge (b \Rightarrow c) \wedge \neg(a \Rightarrow c)$
0	0	0	1	1	1		
0	0	1	1	1	1		
0	1	0	1	0	1		
0	1	1	1	1	1		
1	0	0	0	1	0		
1	0	1	0	1	1		
1	1	0	1	0	0		
1	1	1	1	1	1		

1.2.5 Important equivalences

Reasoning by equivalence is to use the properties of equivalence (reflexivity, symmetry, transitivity), and the property of replacement of a formula with another equivalent formula to obtain new equivalences from equivalences already proven or admitted. Below we list some important equivalences of logic.

1. The disjunction is:
   — Associative, i.e. $x \vee (y \vee z) \equiv (x \vee y) \vee z$.
   — Commutative, i.e., $x \vee y \equiv y \vee x$.
   — 0 is the neutral element of the disjunction, i.e., $0 \vee x \equiv x$.
2. The conjunction is:
   — Associative, i.e. $x \wedge (y \wedge z) \equiv (x \wedge y) \wedge z$.
   — Commutative, i.e. $x \wedge y \equiv y \wedge x$.
   — 1 is the neutral element of the conjunction, i.e. $1 \wedge x \equiv x$.
3. The conjunction is distributive over the disjunction: $x \wedge (y \vee z) \equiv (x \wedge y) \vee (x \wedge z)$.

4. The disjunction is distributive over the conjunction: $x \vee (y \wedge z) \equiv (x \vee y) \wedge (x \vee z)$.
5. The laws of negation:
   — $x \wedge \neg x \equiv 0$.
   — $x \vee \neg x \equiv 1$ (*excluded-middle*).
6. $\neg\neg x \equiv x$.
7. $\neg 0 \equiv 1$.
8. $\neg 1 \equiv 0$.
9. De Morgan's laws:
   — $\neg(x \wedge y) \equiv \neg x \vee \neg y$.
   — $\neg(x \vee y) \equiv \neg x \wedge \neg y$.
10. 0 is the absorbing element of the conjunction: $0 \wedge x \equiv 0$.
11. 1 is the absorbing element of the disjunction: $1 \vee x \equiv 1$.
12. idempotence for the disjunction: $x \vee x \equiv x$.
13. idempotence for the conjunction: $x \wedge x \equiv x$.

The equivalences 1 on the preceding page to 5 are demonstrated using the truth tables[3]. The equivalences 6 to 13 are deduced from the equivalences 1 on the preceding page to 5, as shown in section 1.5 on page 24. Below we give some simplification laws which allow us to accelerate logical reasoning.

Property 1.2.31 (Simplification laws) *For any x, y we have:*
— $x \vee (x \wedge y) \equiv x$.
— $x \wedge (x \vee y) \equiv x$.
— $x \vee (\neg x \wedge y) \equiv x \vee y$.
— $x \wedge (\neg x \vee y) \equiv x \wedge y$.

Proof : By using the important equivalences we prove the four simplification laws. The proof is required in exercise 12 on page 32. □

1.3 Substitution and replacement

In this paragraph, we extend the set of valid formulas by substitution and replacement.

1.3.1 Substitution

Definition 1.3.1 (Substitution) *A* substitution *is an application from the set of variables to the set of formulas. The application of a substitution σ to a formula consists in replacing in the formula every variable x by the formula $\sigma(x)$.*

Notation: Let A be a formula, we note $A\sigma$ or $\sigma(A)$ the application of substitution σ to the formula A.

Definition 1.3.2 (Domain of a substitution, finite domain substitution) *The* domain *of a substitution σ is the set of variables x such that $x\sigma \neq x$. A* finite domain *substitution σ is written $< x_1 := A_1, \ldots, x_n := A_n >$, where $A_1, \ldots, A_n$ are formulas, $x_1, \ldots, x_n$ are* distinct *variables and the substitution is such that:*
— *for i from 1 to n, $x_i\sigma = A_i$*
— *for any y variable such that $y \notin \{x_1, \ldots, x_n\}$, we have: $y\sigma = y$*

Example 1.3.3 *Let the formula $A = x \vee x \wedge y \Rightarrow z \wedge y$ and the substitution $\sigma =< x := a \vee b, z := b \wedge c >$, σ applied to A gives:*

The domain of σ is finite and contains variables x and z.

[3] We leave it to the reader to convince themselves of the veracity of these equivalences by building the corresponding tables.

Property 1.3.4 *Let A be a formula, v an assignment and σ a substitution, we have $[A\sigma]_v = [A]_w$ where for any variable x, $w(x) = [\sigma(x)]_v$.*

Proof : Let A be a formula, v an assignment and σ a substitution, we want to prove that $[A\sigma]_v = [A]_w$ where for any variable x, $w(x) = [\sigma(x)]_v$. By induction on the formula size, we get:

Base case: $|A| = 0$.

Induction:

Suppose the property holds for any formula with size lower or equal to n, let's show that it is true for a formula A with size $n+1$:

□

Example 1.3.5 *Let $A = x \vee y \vee d$ be a formula. Let $\sigma = < x := a \vee b, y := b \wedge c >$ be a substitution. Let v be an assignment such that $v(a) = 1$, $v(b) = 0$, $v(c) = 0$, $v(d) = 0$. We have:*

Theorem 1.3.6 *The application of a substitution to a valid formula gives a valid formula.*

Proof : Let A be a valid formula, σ a substitution and v an assignment. Based on property 1.3.4: $[A\sigma]_v = [A]_w$ where for any variable x, $w(x) = [\sigma(x)]_v$. Since A is valid, $[A]_w = 1$. As a result $A\sigma$ evaluates to 1 in any assignment, so it is a valid formula. □

Example 1.3.7 *Let A be the formula $\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$. This formula is valid, it is an important equivalence. Let σ be the following substitution: $< p := (a \vee b), q := (c \wedge d) >$. The formula*

Substitution is defined over formulas; to apply it without error to prioritized formulas, it is sufficient that the image, by the substitution, of every variable is a variable, a constant or a formula into parentheses.

1.3.2 Replacement

The concept of substitution does not allow us to replace a formula by a formula, we introduce the concept of replacement to this end.

Definition 1.3.8 (Replacement) *Let A,B,C,D be formulas. The formula D is obtained by replacing in C some occurrences of A with B, if there exists a formula E and a variable x such that $C = E < x := A >$ and $D = E < x := B >$.*

Example 1.3.9 *Consider the formula $C = ((a \Rightarrow b) \vee \neg(a \Rightarrow b))$.*

— *The formula obtained by replacing all occurrences of $(a \Rightarrow b)$ with $(a \wedge b)$ in C is*

it is obtained considering the formula $E = (x \vee \neg x)$ and substitutions $< x := (a \wedge b) >$ and $< x := (a \Rightarrow b) >$.

— *The formula obtained by replacing the first occurrence of $(a \Rightarrow b)$ with $(a \wedge b)$ in C is*

it is obtained by considering the formula $E = (x \vee \neg(a \Rightarrow b))$ and substitutions $< x := (a \wedge b) >$ and $< x := (a \Rightarrow (b) >$.

The difference between a substitution and a replacement is that substitution replaces a set of variables with formulas whereas a replacement replaces the occurrences of certain formulas with another formula using substitutions.

Theorem 1.3.10 *Let C be a formula and D the formula obtained by replacing, in C, some* occurrences *of formula A with formula B, then $(A \Leftrightarrow B) \models (C \Leftrightarrow D)$.*

Proof : By definition of replacement, there is a formula E and a variable x such that $C = E < x := A >$ and $D = E < x := B >$. Suppose that v is an assignment model of $(A \Leftrightarrow B)$. Thus we have $[A]_v = [B]_v$. Based on property 1.3.4 on the preceding page:

— $[C]_v = [E]_w$ where w is identical to v except for $w(x) = [A]_v$

— $[D]_v = [E]_{w'}$ where w' is identical to v except for $w'(x) = [B]_v$

Since $[A]_v = [B]_v$, the assignments w and w' are identical, so $[C]_v = [D]_v$. As a result v is a model of $(C \Leftrightarrow D)$. □

Corollary 1.3.11 *Let C be a formula and D the formula obtained by replacing, in C, an* occurrence *of the formula A with the formula B, then $A \equiv B$ implies $C \equiv D$.*

Proof : If $A \equiv B$, then the formula $(A \Leftrightarrow B)$ is valid (property 1.2.10), so the formula $(C \Leftrightarrow D)$ also is since it is, according to the above theorem, a consequence of $(A \Leftrightarrow B)$, as a result $C \equiv D$. □

Example 1.3.12 *Replacement of an occurrence of a formula A with an occurrence of B is highlighted by boxes marking these occurrences.*

— *Based on theorem 1.3.10 :* $p \Leftrightarrow q \models (p \vee (\boxed{p} \Rightarrow r)) \Leftrightarrow (p \vee (\boxed{q} \Rightarrow r))$.

— *Based on corollary 1.3.11:* $(\neg(p \vee q) \Rightarrow (\boxed{\neg(p \vee q)} \vee r)) \equiv (\neg(p \vee q) \Rightarrow (\boxed{(\neg p \wedge \neg q)} \vee r))$, *since* $\neg(p \vee q) \equiv (\neg p \wedge \neg q)$.

Remark 1.3.13 *The previous theorem and its corollary apply to formulas. When we make a replacement directly on a prioritized formula, we must make sure that this replacement remains correct on (strict) formulas, or we may commit errors. For example, consider both equivalencies $a \wedge \boxed{b} \equiv a \wedge \boxed{b}$ and $\neg c \Rightarrow d \equiv c \vee d$. Let us replace b on the left with $\neg c \Rightarrow d$ and on the right with $c \vee d$. We observe that although $\neg c \Rightarrow d \equiv c \vee d$, $a \wedge \neg c \Rightarrow d \not\equiv a \wedge c \vee d$, because for $a = c = d = 0$, the left formula evaluates to 1 and the right one to 0. Here the corollary must not be applied to the framed occurrence because $a \wedge \neg c \Rightarrow d$ is an abbreviation of $((a \wedge \neg c) \Rightarrow d)$, so $\neg c \Rightarrow d$ does not appear as a possible occurrence of $a \wedge \neg c \Rightarrow d$.*

We have defined replacements from substitutions, let us now apply the replacements to a formula to transform it into an equivalent normal form.

1.4 Normal Forms

To put a formula into normal form is to transform it into a equivalent formula having structural properties. We introduce two notions of normal forms: disjunctive normal form which allows to highlight the models, and conjunctive normal form which emphasizes the counter-models. The definition of normal form requires the introduction of a few concepts: literal, monomial and clause.

Definition 1.4.1 (Literal, monomial, clause)
— *A* literal *is a variable or the negation of a variable.*
— *A* monomial *is a conjunction of literals.*
— *A* clause *is a disjunction of literals.*

Example 1.4.2 *We illustrate these new concepts with a few simple examples:*
— $x, y, \neg z$ *are literals.*
— $x \wedge \neg y \wedge z$ *is a monomial whose only model is* $x = 1, y = 0, z = 1$.
— *The monomial* $x \wedge \neg y \wedge z \wedge \neg x$ *contains a variable and its negation: its value is* 0.
— $x \vee \neg y \vee z$ *is a clause whose only counter-model is* $x = 0, y = 1, z = 0$.
— *The clause* $x \vee \neg y \vee z \vee \neg x$ *contains a variable and its negation: its value is* 1.

1.4.1 Transformation into normal form

We introduce the notion of normal form and show that it is always possible to transform a formula into an equivalent normal form.

Definition 1.4.3 (normal form) *A formula is in* normal form *if it uses only the operators* $\wedge, \vee, \neg$ *and its negations are only applied to variables.*

Example 1.4.4 *The formula* $\neg a \vee b$ *is in normal form, while the formula* $a \Rightarrow b$ *is not in normal form although it is equivalent to the first.*

We now explain how to turn any formula into an equivalent formula in normal form thanks to replacements. To apply the $A \equiv B$ equivalence to the C formula, means to replace in C an occurrence of A with an occurrence of B: we have proved in theorem 1.3.10 on the preceding page that such a replacement changes C into an equivalent formula. Thus, to transform a formula into an equivalent formula in normal form, we apply the following transformations:

1. **Elimination of equivalences**: replace an occurrence $A \Leftrightarrow B$ with one of the subformulas:
   (a) $(\neg A \vee B) \wedge (\neg B \vee A)$.
   (b) $(A \wedge B) \vee (\neg A \wedge \neg B)$.
2. **Elimination of implications**: replace an occurrence of $A \Rightarrow B$ with $\neg A \vee B$.
3. **Shifting negations towards variables**: replace an occurrence of
   (a) $\neg\neg A$ with A.
   (b) $\neg(A \vee B)$ with $\neg A \wedge \neg B$.
   (c) $\neg(A \wedge B)$ with $\neg A \vee \neg B$.
   so that negations apply only to variables.

By applying these three transformations in the indicated order, it is clear that the initial formula has been transformed into an equivalent normal form. In exercise **??** on page ??, we prove that the order of the transformations is not important: by performing the above transformations in any order, we end up with an equivalent formula in normal form.

Remark 1.4.5 *For example, it is recommended to replace a subformula of the form* $\neg(A \Rightarrow B)$ *with* $A \wedge \neg B$*, which actually combines an elimination of implication and a shift of a negation. In practice, it is more efficient to simplify as early as possible as follows:*
— *Replace with* 0 *any conjunction that includes either a formula and its negation, or a* 0.
— *Replace with* 1 *any disjunction that includes either a formula and his negation, or a* 1.
— *Replace* $\neg 1$ *with* 0 *and* $\neg 0$ *with* 1.
— *Remove* 0 *from disjunctions and* 1 *from conjunctions.*
— *Apply the simplifications* $x \vee (x \wedge y) \equiv x$, $x \wedge (x \vee y) \equiv x$, $x \vee (\neg x \wedge y) \equiv x \vee y$.
— *Apply the idempotence of disjunction and conjunction.*

1.4.2 Transformation into disjunctive normal form (sum of monomials)

The disjunctive normal form makes it easy to find models.

Definition 1.4.6 (Disjunctive normal form) *A formula is a* disjunctive normal form *(in short dnf) if and only if it is a disjunction (sum) of monomials.*

The point of disjunctive normal forms is to highlight models.

Example 1.4.7 $(x \wedge y) \vee (\neg x \wedge \neg y \wedge z)$ *is a dnf with two monomials, each of them gives us one of the two possible models:*

Starting from a normal form and distributing all the conjunctions over the disjunctions, we obtain a disjunction of monomials. To do this, you also need to know how to group several applications of distributivity.

Example 1.4.8 *When transforming into a disjunction of monomials, we can apply from left to right the equivalence* $(a \vee b) \wedge (c \vee d \vee e) \equiv$

The transformation by equivalence of a formula into a disjunction of monomials can be used to determine whether a formula is valid or not. Let A be a formula whose validity we want to determine: we transform $\neg A$ into a disjunction of monomials *equivalent* to $\neg A$. If $\neg A = 0$ then $A = 1$ so A is valid, otherwise after all simplifications are done, $\neg A$ is equal to a disjunction of non-zero monomials, which give us models of $\neg A$, so counter-models of A.

Example 1.4.9 *Let* $A = (p \Rightarrow (q \Rightarrow r)) \Rightarrow (p \wedge q \Rightarrow r)$*. Let us determine whether A is valid by transformation of* $\neg A$ *into a disjunction of monomials.*

Example 1.4.10 *Let* $A = (a \Rightarrow b) \wedge c) \vee (a \wedge d)$*. Let us determine whether A is valid by transforming* $\neg A$ *into a disjunction of monomials.*

$\neg A$

$\equiv \neg((a \Rightarrow b) \wedge c) \wedge \neg(a \wedge d)$	*by shifting the negations*
$\equiv (\neg(a \Rightarrow b) \vee \neg c) \wedge (\neg a \vee \neg d)$	*by shifting the negations*
$\equiv (a \wedge \neg b \vee \neg c) \wedge (\neg a \vee \neg d)$	*by a combination of shifting a negation and eliminating an implication*
$\equiv (a \wedge \neg b \wedge \neg a) \vee (a \wedge \neg b \wedge \neg d) \vee (\neg c \wedge \neg a) \vee (\neg c \wedge \neg d)$	*by distributivity of disjunction over conjunction*
$\equiv (a \wedge \neg b \wedge \neg d) \vee (\neg c \wedge \neg a) \vee (\neg c \wedge \neg d)$	*by simplification*

We get 3 models of $\neg A$ *(*$a = 1, b = 0, d = 0$*;* $a = 0, c = 0$*;* $c = 0, d = 0$*), i.e., counter-models of A. Thus A is not valid.*

1.4.3 Transformation into conjunctive normal form (product of clauses)

The conjunctive normal form makes it easy to exhibit counter-models.

Definition 1.4.11 (Conjunctive normal form) *A formula is a* conjunctive normal form *(in short cnf) if and only if it is a conjunction (product) of clauses.*

The point of conjunctive normal forms is to highlight counter-models.

Example 1.4.12 *$(x \vee y) \wedge (\neg x \vee \neg y \vee z)$ is a cnf, which has two counter-models*

By convention, we consider 0 and 1 to be disjunctions of monomials and conjunctions of clauses. We apply the (unusual) distributivity of disjunction over conjunction, in other words we replace any subformula $A \vee (B \wedge C)$ with $(A \vee B) \wedge (A \vee C)$, and any subformula $(B \wedge C) \vee A$ with $(B \vee A) \wedge (C \vee A)$.

Example 1.4.13 *We take again the example 1.4.8 on the preceding page and get $(a \wedge b) \vee (c \wedge d \wedge e) \equiv$*

The transformation by equivalence of a formula into a conjunction of literal clauses may also be used to determine whether a formula is valid or not. Let A be a formula whose validity we want to determine: we transform A into a conjunction of clauses *equivalent* to A. If $A = 1$ then A is valid, otherwise after all simplifications have been made, A is equal to a conjunction of clauses different from 1, and each of these disjunctions gives us a counter-model of A.

1.5 Boolean algebra

This concept was introduced by British mathematician George Boole in the middle of the 19th century. In particular, it allows the statements to be translated into equations (usually this expression is more concise). Let us first recall the definition of a Boolean algebra. Then we deduce that propositional logic is a Boolean algebra. We also prove some usual properties of Boolean algebras.

1.5.1 Definition and notations

The definition of a Boolean algebra given below is using a minimal number of axioms.

Definition 1.5.1 *A* Boolean algebra *is a set with at least two items, 0, 1, and three operations, complement (the complement of x is noted $\bar{x}$), sum ($+$) and product ($.$), which comply with the following axioms:*

1. *The sum is:*
   - *associative: $x+(y+z)=(x+y)+z$,*
   - *commutative: $x+y=y+x$,*
   - *0 is the neutral element of the sum: $0+x=x$.*
2. *The product is :*
   - *associative: $x.(y.z)=(x.y).z$,*
   - *commutative: $x.y=y.x$,*
   - *1 is the neutral element of the product: $1.x=x$.*
3. *The product is distributive over the sum: $x.(y+z)=(x.y)+(x.z)$.*
4. *The sum is distributive over the product: $x+(y.z)=(x+y).(x+z)$.*
5. *The laws of negation:*
   - *$x+\bar{x}=1$,*
   - *$x.\bar{x}=0$.*

Warning: the fact that the distributivity of the sum over the product is unusual makes its application prone to errors.

As mentioned above in subsection 1.2.5 on page 18, we can prove using truth tables that propositional logic complies with all the axioms of a Boolean algebra. Thus, we can consider propositional logic as the smallest Boolean algebra, since it contains only two elements. As a result, we can use the (more condensed) boolean notations instead of the propositional notations, as indicated in the correspondence table in figure 1.1. We recommend to use them to perform large calculations. Furthermore, note that these notations are frequently used in the hardware field.

logical notations	boolean notations
$\neg a$	$\overline{a}$
$a \wedge b$	$a.b$
$a \vee b$	$a+b$
$a \Rightarrow b$	$\overline{a}+b$
$a \Leftrightarrow b$	$a.b+(\overline{a}.\overline{b})$ or $(a+\overline{b}).(\overline{a}+b)$

Figure 1.1: Correspondence between Boolean algebra and propositional logic.

Remark 1.5.2 (Writing conventions)

— *Sometimes the (arithmetic) negation is used instead of the the complement. For example, $-a$ denotes $\overline{a}$ and $-(a+b)$ represents $\overline{a+b}$.*

— *In order to be able to omit the dot, in the boolean notations, the variables are often denoted by a single letter (with or without index). So when a and b are variables, formulas inside parentheses or under a negation overline, we abbreviate a.b into ab following a usual practice in mathematics.*

— *Finally, the equivalence of two formulas is systematically noted with the $=$ symbol instead of the $\equiv$ symbol.*

It is important to notice that propositional logic is not the only Boolean algebra. For example, the set $\mathcal{P}(X)$ of the subsets of a set X is a Boolean algebra, as indicated by the matching of set operators with their designation in Boolean algebra in figure 1.2.

Boolean Algebra	$\mathcal{P}(X)$
1	X
0	$\emptyset$
$\overline{p}$	$X-p$
$p+q$	$p \cup q$
$p.q$	$p \cap q$

Figure 1.2: Correspondence between Boolean algebra and $\mathcal{P}(X)$.

Since simplification laws are deduced from the laws of Boolean algebra, as we will prove next, they apply to any Boolean algebra. In particular in the Boolean algebra $\mathcal{P}(X)$ we have $A \cup (A \cap B) = A$. A *proof in a Boolean algebra* is a sequence of equalities, allowing to transform a formula into another equivalent formula using axioms or simplifications of Boolean algebra.

1.5.2 Properties

We present some properties derived from the definition of Boolean algebra which are commonly used in reasonings. To shorten the proofs of these properties, we implicitly use associativity and commutativity of the sum and product.

Property 1.5.3 *In a Boolean algebra, for every x, there is one and only one y such that $x+y=1$ and $xy=0$, i.e. the negation is unique.*

Proof : There is an element y such that $x+y=1$ and $xy=0$, it is $\overline{x}$ according to the axioms of Boolean algebra. Let's show it is unique. Let us reason by absurd, assume it is not unique, so there are y and z two distinct elements such that $x+y=1$, $xy=0$, $x+z=1$, and $xz=0$. We consider $u=xy+z$. Since $xy=0$, we have: $u=z$. By distributivity of the

sum over the product, we have: $u = (x+z)(y+z)$. Since $x+z=1$ and 1 is the neutral element of the product, we have: $u = y+z$. Therefore we conclude that $z = y+z$. Considering $v = xz+y$ and switching the roles of y and z, we get similarly $y = y+z$. Thus $y = z$, which proves the uniqueness of the negation. □

Property 1.5.4 *In a Boolean algebra, the following equivalences are true for every x and every y:*

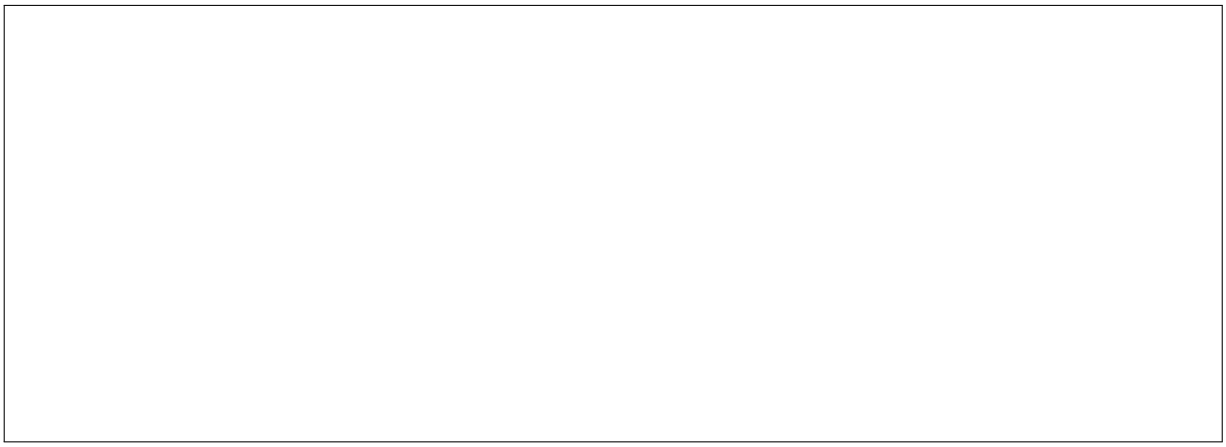
1. $\overline{1} = 0$.
2. $\overline{0} = 1$.
3. $\overline{\overline{x}} = x$.
4. *Idempotence of the product:* $x.x = x$.
5. *Idempotence of the sum:* $x+x = x$.
6. 1 *is the absorbing element of the sum:* $1+x = 1$.
7. 0 *is the absorbing element of the product:* $0.x = 0$.
8. *De Morgan's laws:*
   — $\overline{xy} = \overline{x}+\overline{y}$.
   — $\overline{x+y} = \overline{x}.\overline{y}$.

Proof :

1. $\overline{1} = 0$.
   By definition of negation, $x.\overline{x} = 0$. Thus, $1.\overline{1} = 0$. Since 1 is the neutral element of the product, we have $\overline{1} = 0$.
2. $\overline{0} = 1$.
3. $\overline{\overline{x}} = x$.
4. Idempotence of the product: $x.x = x$.
   In this proof and in the following, we omitted the justification of the equalities. We suggest that the reader check these proofs and adds the omitted justifications.

$$\begin{aligned} x &= x.1 \\ &= x.(x+\overline{x}) \\ &= x.x+x.\overline{x} \\ &= x.x+0 \\ &= x.x \end{aligned}$$

5. Idempotence of the sum: $x+x = x$.



6. 1 is the absorbing element of the sum: $1+x=1$.
   We use the idempotence of the sum.

$$\begin{aligned} 1+x &= (x+\overline{x})+x \\ &= x+\overline{x} \\ &= 1 \end{aligned}$$

7. 0 is the absorbing element of the product: $0.x=0$.

   De Morgan's laws:
   — $\overline{x.y}=\overline{x}+\overline{y}$: We use the uniqueness of the negation (property 1.5.3 on page 25). We first show that $x.y+(\overline{x}+\overline{y})=1$

$$\begin{aligned} x.y+(\overline{x}+\overline{y}) &= (x+\overline{x}+\overline{y}).(y+\overline{x}+\overline{y}) \\ &= (1+\overline{y}).(1+\overline{x}) \\ &= 1.1 \\ &= 1 \end{aligned}$$

   We also prove that $x.y.(\overline{x}+\overline{y})=0$.

$$\begin{aligned} x.y.(\overline{x}+\overline{y}) &= x.y.\overline{x}+x.y.\overline{y} \\ &= 0.y+x.0 \\ &= 0+0 \\ &= 0 \end{aligned}$$

   Since the negation is unique, $\overline{x}+\overline{y}$ is indeed the negation of xy.
   — $\overline{x+y}=\overline{x}.\overline{y}$.
   We use the uniqueness of the negation. We first show that $(x+y)+\overline{x}.\overline{y}=1$

$$\begin{aligned} (x+y)+\overline{x}.\overline{y} &= (x+y+\overline{x}).(x+y+\overline{y}) \\ &= (1+y).(x+1) \\ &= 1.1 \\ &= 1 \end{aligned}$$

We also prove that $(x+y).\overline{x}.\overline{y} = 0$.

$$\begin{aligned}(x+y).\overline{x}.\overline{y} &= (x.\overline{x}.\overline{y}) + (y.\overline{x}.\overline{y}) \\ &= (0.\overline{y}) + (0.\overline{x}) \\ &= 0+0 \\ &= 0\end{aligned}$$

From the uniqueness of the negation, we conclude that $\overline{x}.\overline{y}$ is indeed the negation of $(x+y)$.

□

1.6 Boolean functions

We now look at our formulas as mathematical functions over domain $\mathbb{B} = \{0,1\}$. We formally define what a Boolean function is, then we revisit the conjunctive and disjunctive normal forms with this formalism. Note that Boolean functions are notions used in cryptology for example when building symmetric encryptions (One-Time-Pad encryption, one of the safest encryptions, is based on the "exclusive or" Boolean function).

Definition 1.6.1 (Boolean function) *A* Boolean function f *is a function whose arguments and result are in domain* $\mathbb{B} = \{0,1\}$.

$$f : \mathbb{B}^n \to \mathbb{B}$$

Example 1.6.2 *We give examples and counterexamples of Boolean functions:*

- *The function* $f : \mathbb{B} \to \mathbb{B} : f(x) = \neg x$ *is a Boolean function.*
- *The function* $f : \mathbb{N} \to \mathbb{B} : f(x) = x \bmod 2$ *is not a Boolean function.*
- *The function* $f : \mathbb{B} \to \mathbb{N} : f(x) = x+1$ *is not a Boolean function.*
- *The function* $f : \mathbb{B} \times \mathbb{B} \to \mathbb{B} : f(x,y) = \neg(x \wedge y)$ *is a Boolean function.*

1.6.1 Boolean functions and sum of monomials

Thanks to Boolean functions, we are able, starting from the truth values of a formula, to reconstruct the equivalent sum of monomials. For any variable x, let us note $x^0 = \overline{x}$ and $x^1 = x$.

Theorem 1.6.3 *Let* f *be a Boolean function with* n *arguments. This function is represented using* n *variables* $x_1, \ldots, x_n$. *Let* A *be the following formula:*

$$A = \sum_{f(a_1,\ldots,a_n)=1} x_1^{a_1} \ldots x_n^{a_n}.$$

The a_i *are Boolean values and* A *is the sum of the monomials* $x_1^{a_1} \ldots x_n^{a_n}$ *such that* $f(a_1,\ldots,a_n) = 1$. *By convention, if the function* f *always evaluates to* 0 *then* $A = 0$.

For any assignment v *such that* $v(x_1) = a_1, \ldots, v(x_n) = a_n$, *we have* $f(a_1,\ldots,a_n) = [A]_v$. *Omitting the distinction between a variable and its value, this result is written more briefly:* $f(x_1,\ldots x_n) = A$.

Example 1.6.4 *The maj function with 3 arguments evaluates to* 1 *when at least 2 of its arguments are equal to* 1. *We give the table representing this function:*

x_1	x_2	x_3	$maj(x_1,x_2,x_3)$					
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
0	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*
0	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
0	*1*	*1*	**1**	**1**	*0*	*0*	*0*	**1**
1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
1	*0*	*1*	**1**	*0*	**1**	*0*	*0*	**1**
1	*1*	*0*	**1**	*0*	*0*	**1**	*0*	**1**
1	*1*	*1*	**1**	*0*	*0*	*0*	**1**	**1**

We notice on the truth table, that any monomial evaluates to 1 *only on one line of the table and that the sum of the monomials is equivalent to* $maj(x_1,x_2,x_3)$*. The* maj *function can therefore be expressed by*

1.6.2 Boolean functions and product of clauses

We conduct the same reasoning for the dual form.

Theorem 1.6.5 *Let* f *be a Boolean function with n arguments. This function is represented using n variables* $x_1,\ldots,x_n$*. Let A be the following formula:*

$$A = \prod_{f(a_1,\ldots,a_n)=0} x_1^{\overline{a_1}} + \ldots + x_n^{\overline{a_n}}.$$

. The a_i *are Boolean values and A is the product of the clauses* $x_1^{\overline{a_1}} + \ldots + x_n^{\overline{a_n}}$ *such that* $f(a_1,\ldots,a_n) = 0$*. By convention if the function* f *always evaluates to* 1 *then* $A = 1$*.*

For any assignment v such that $v(x_1) = a_1,\ldots,v(x_n) = a_n$*, we have* $f(a_1,\ldots,a_n) = [A]_v$*. Omitting the distinction between a variable and its value, this result is written more briefly:* $f(x_1,\ldots,x_n) = A$*.*

Example 1.6.6 *We represent the* maj *function defined in example 1.6.4 on the facing page by a product of clauses:*

x_1	x_2	x_3	$maj(x_1,x_2,x_3)$					
0	*0*	*0*	**0**	**0**	*1*	*1*	*1*	**0**
0	*0*	*1*	**0**	*1*	**0**	*1*	*1*	**0**
0	*1*	*0*	**0**	*1*	*1*	**0**	*1*	**0**
0	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
1	*0*	*0*	**0**	*1*	*1*	*1*	**0**	**0**
1	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
1	*1*	*0*	*1*	*1*	*1*	*1*	*1*	*1*
1	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*

The maj *function can therefore be expressed as a product of clauses as follows:*

1.7 The BDDC Tool

`BDDC` (*Binary Decision Diagram based Calculator*) is a tool for handling propositional formulas developed by Pascal Raymond and available at the following URL:

`http://www-verimag.imag.fr/~raymond/home/tools/bddc/`.

This tool is a "propositional calculator" based on *binary decision diagrams*. These diagrams allow an internal representation of logical formulas as an acyclic oriented graph which is canonical. Among other possibilities, this calculator offers to:

— assess whether a formula is a tautology,
— assess whether a formula has a model,
— transform a formula into a conjunctive normal form,
— transform a formula into a disjunctive normal form,
— ...

Thus, it is possible to use `BDDC` to solve several of the exercises proposed thereafter, among which 8 on page 31 to **??** on page ?? and 23 on page 33 to 24 on page 33.

1.8 Exercises

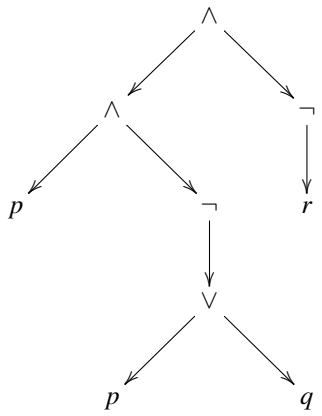
Exercise 1 (Strict Formulas, Prioritized Formulas) *Which of the following expressions are strict formulas?*

1. x
2. $(x$
3. (x)
4. $(x \Rightarrow (y \wedge z)$
5. $(x \Rightarrow (y \wedge z))$
6. $\neg(x \vee y)$
7. $(\neg(x \vee y))$
8. $\neg(x \Rightarrow y) \vee \neg(y \wedge z)$
9. $\neg(x)$
10. $(\neg((x \vee y) \wedge z) \Leftrightarrow (u \Rightarrow v)))$

□

Exercise 2 (Formulas and Priority) *Prove that every (strict) formula is a prioritized formula.* □

Exercise 3 (Formulas and Priority) *Consider the following tree representation of the formula* $((p \wedge \neg(p \vee q)) \wedge \neg r)$.



Using priority, this formula can be written with fewer parentheses: $p \wedge \neg(p \vee q) \wedge \neg r$ *using logical notation,* $p\ \overline{p+q}\ \overline{r}$ *using boolean notation. Similarly, give the tree representation, logical notation and boolean notation, both using as few parentheses as possible of the following formulas:*

1. $(\neg(a \wedge b) \Leftrightarrow (\neg a \vee \neg b))$.
2. $((\neg a \vee b) \wedge (\neg b \vee a))$.
3. $(((a \wedge b) \wedge c) \vee ((\neg a \wedge \neg b) \wedge \neg c))$.
4. $(p \Rightarrow (q \Rightarrow r))$.
5. $((p \Rightarrow q) \Rightarrow r)$.

□

Exercise 4 (Formulas and Priority) *Give the tree representation of the following prioritized formulas:*

1. $p \Leftrightarrow \neg q \vee r$.
2. $p \vee q \Rightarrow r \wedge s$.
3. $p \vee q \Rightarrow r \Leftrightarrow s$.
4. $p \vee q \wedge r \Rightarrow \neg s$.
5. $p \Rightarrow r \wedge s \Rightarrow t$.
6. $p \vee q \wedge s \vee t$.
7. $p \wedge q \Leftrightarrow \neg r \vee s$.

8. $\neg p \wedge q \vee r \Rightarrow s \Leftrightarrow t$.

□

Exercise 5 (Height of a Formula,*) *We define the height h of a prioritized formula as:*

— $h(\top) = 0$ *and* $h(\bot) = 0$.
— *If A is a variable* $h(A) = 0$.
— $h(\neg A) = 1 + h(A)$.
— $h((A)) = h(A)$.
— $h(A \circ B) = max(h(A), h(B)) + 1$, *if* $\circ$ *is one of the following operations:* $\vee, \wedge, \Rightarrow, \Leftrightarrow$.

1. *Show that this definition is ambiguous, that is, show that there exists at least one formula (by presenting one) that can have two different heights under this definition.*
2. *Give a new recuseive definition of height of a formula that does not have this problem.*

□

Exercise 6 (Validity) *Give the truth table of the following formulas:*

1. $p \Rightarrow (q \Rightarrow p)$.
2. $p \Rightarrow (q \Rightarrow r)$.
3. $(p \Rightarrow q) \Rightarrow (p \Rightarrow r)$.
4. $(p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r))$.

Indicate which formulas are valid, and which are equivalent. □

Exercise 7 (Circular Reasoning) *Give the truth table of the following formulas:* $a \Rightarrow b$, $b \Rightarrow c$, $c \Rightarrow a$, $a \Leftrightarrow b$, $b \Leftrightarrow c$, *and* $c \Leftrightarrow a$. *Conclude that* $(a \Rightarrow b) \wedge (b \Rightarrow c) \wedge (c \Rightarrow a) \equiv (a \Leftrightarrow b) \wedge (b \Leftrightarrow c) \wedge (c \Leftrightarrow a)$. □

Exercise 8 (Equivalence) *Give the truth table of the following formulas:*

1. $p \wedge (p \vee q)$.
2. $\neg p \wedge \neg q$.
3. $\neg(p \wedge q)$.
4. $\neg(p \vee q)$.
5. $p \vee (p \wedge q)$.
6. $\neg p \vee \neg q$.
7. p.

Indicate which formulas are equivalent. □

Exercise 9 (Equivalence) *Give the truth table of the following formulas:*

1. $(p \Rightarrow q) \wedge (q \Rightarrow p)$.
2. $p \Rightarrow q$.
3. $p \Leftrightarrow q$.
4. $\neg q \Rightarrow \neg p$.
5. $(p \wedge q) \vee (\neg p \wedge \neg q)$.

Indicate which formulas are equivalent. □

Exercise 10 (Equivalence) *Which of the following formulas are equivalent to* $p \Rightarrow q \vee r$*?*

1. $q \wedge \neg r \Rightarrow p$.
2. $p \wedge \neg r \Rightarrow q$.
3. $\neg q \wedge \neg r \Rightarrow \neg p$.
4. $q \vee \neg p \vee r$.

□

Exercise 11 (Model for a set of formulas,*) *Show that an assignment is a model for a set of formulas if and only if it is a model for the conjunction of all the formulas in the set.* □

Exercise 12 (Simplification Laws) *Prove that for any x, y*

— $x \vee (x \wedge y) \equiv x$.
— $x \wedge (x \vee y) \equiv x$.
— $x \vee (\neg x \wedge y) \equiv x \vee y$.

□

Exercise 13 (Simplification of Formula,*) *Show by simplification that the following formula is a tautology.*

$$(a+b) \cdot (\overline{b}+c) \Rightarrow (a+c)$$

□

Exercise 14 (Models and Normal Forms) *Let A be the following formula:*

$$(((a \Rightarrow \neg b) \Leftrightarrow \neg c) \wedge (c \vee d)) \wedge (a \Leftrightarrow d).$$

1. *Is A a tautology? (justify)*
2. *Is A a contradiction ? (justify)*
3. *Give the conjunctive normal form for A.*
4. *Give the disjunctive normal form for A.*

□

Exercise 15 (Natural induction,*) *Show that if a formula contains only one variable, the constants $\top$ and $\bot$ and the operations $\vee$ and $\wedge$ (no negation), then it is equivalent to a formula of size* 0. □

Exercise 16 (Boolean Algebra) *Using truth tables, determine if the operations $\Rightarrow, \Leftrightarrow$ are commutative, associative, idempotent, transitive.* □

Exercise 17 (Boolean Algebra) *Show that a formula with only one variable (say, that variable is p) is equivalent to either* 0, 1, *p, or $\neg p$.* □

Exercise 18 (Boolean Algebra) *Write* 16 *formulas such that any formula with two variables, say p and q, is equivalent to one of those* 16 *formulas.* □

Exercise 19 (Boolean Function) *Determine the number of distinct (non-equivalent) boolean functions with k arguments (express your answer as a function of k).* □

Exercise 20 (Consequence) *During an inquiry, adjutant Tinet makes the following reasoning:*

— *If the murder occurred during the day, then the murderer is a friend of the victim*
— *However, the murder occurred at night*

Therefore, the murderer is not a friend of the victim. Is adjutant Tinet correct? To determine this, proceed in three steps:

1. *Formalize the facts*
2. *Formalize the reasoning that allows to arrive to the conclusion from the hypotheses*
3. *Determine if the reasoning is correct.*

□

Exercise 21 (Consequence) *Pinnochio, Quasimodo and Romeo are singing in a choir. They decide that:*

1. *If Pinocchio does not sing, then Quasimodo will.*
2. *If Quasimodo sings, then so will Pinocchio and Romeo.*
3. *If Romeo sings, then at least one of Quasimodo or Pinocchio will not.*

Determine whether Pinocchio will sing. Justify your answer by formalizing the reasoning. □

Exercise 22 (Consequence) *Formalize the following statements using logic connectors.*

(a) If Peter went home, then John went to the cinema.

(b) Mary went to the library or Peter went home.

(c) If John went to the cinema, then Mary went to the library or Peter went home.

(d) Mary is not at the library and John went to the cinema.

(e) Peter went home.

Is the last statement a consequence of the preceding ones? □

Exercise 23 (Normal Forms) *For each of the following formulas, write the equivalent disjunctive normal form, and prove whether or not it is satisfiable (by giving a model for the formula if possible).*

— $\neg(a \Leftrightarrow b) \vee (b \wedge c) \Rightarrow c$.

— $(a \Rightarrow b) \wedge (b \Rightarrow \neg a) \wedge (\neg a \Rightarrow b) \wedge (b \Rightarrow a)$.

□

Exercise 24 (Normal Forms) *Let A be the formula $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$. Find the formula in disjunctive normal form equivalent to $\neg A$. Is A valid?* □

Exercise 25 (↷Normal Forms,)** *We come back to the proofs regarding the transformation of formulas into normal forms.*

1. *Show that the repeated application of the elimination of equivalence, elimination of implications and displacement of negations on a formula yields a formula in normal form.*
2. *Show also regardless of the order in which the elinimation of equivalence, elimination of implications and displacement of negations are used, the algorithm for transforming a formula in normal form will terminate.*

□

Exercise 26 (Consequence and normal forms) *Adjutant Tinet is on a new inquiry. His hypotheses are as follows:*

— *If John did not meet Peter the other night, then Peter is the murderer or John is a liar.*

— *If Peter is not the murderer, then John did not meet Peter the other night and the crime occurred after midnight.*

— *If the crime occurred after midnight, then Peter is the murderer or John is a liar.*

He concludes that Peter is the murderer. Is his reasoning correct? Give your answer by constructing the conjunction of the hypotheses and the negation of the conclusion*, and by putting this disjunction in conjunctive normal form. Recall that a reasoning is incorrect if and only if one of the monomials does not have any complementary litterals: this monomial gives a model of the hypotheses that is a counter-model of the conclusion.* □

Exercise 27 (Formalization, sum of monomials*) *On an isolated island, the natives are split into two tribes: the Tame, and the Lame. The Tame always tell the truth, the Lame always lie. We meet two natives: Aha and Beeby*[4]

(a) Aha says: " At least one of us is a Lame ". Can we deduce Aha and Beeby's tribe?

(b) Aha says: " At most one of us is a Lame ". Can we deduce Aha and Beeby's tribe?

(c) Aha says: " Both of us are in the same tribe ". Can we deduce Aha and Beeby's tribe?

□

Exercise 28 (Complete and incomplete set of connectors,)** *A set of constants and connectors is called* complete *if any boolean function can be expressed with these connectors. Theorem 6.4 page 25 shows thaat the set $\{0,1,+,-,\cdot\}$ is complete.*

1. *Show that the set $\{+,-\}$ is complete.*
   Hint: it suffices to show that $\{0,1,\cdot\}$ can be defined only with $+$ and $-$.
2. *Show that the set $\{0,\Rightarrow\}$ is complete.*
3. *Let $\mid$ be the following operation: $x \mid y$ is true if and only if neither x nor y is true, that is $x \mid y$ is true only if $x = 0$ and $y = 0$. This operation is also called* NOR *(for "not OR") because it is the negation of the "OR" ($\vee$) connector. Show that $\{\mid\}$ is complete.*

[4]This problem comes from a book by Raymond M. Smullyan : " What is the Name of this Book ", which describes many other amusing problems with these natives.

4. *(**) Show that the set $\{0,1,+\cdot\}$ is not complete.*
   Hint: boolean function that are defined using only these operations have a property that is not true of all boolean functions.
   We say that a boolean function is monotone *(or* monotonic*) if whenever $a_1 \leq b_1, \ldots, a_n \leq b_n$, then $f(a_1,\ldots,a_n) \leq f(b_1,\leq,b_n)$.*
   Show that every boolean function defined only with $\{0,1,+,\cdot\}$ is monotone.
   Give a boolean function that is not monotone. Conclude that $\{0,1,+,\cdot\}$ is not complete.
5. *(*) Is the set $\{1,\Rightarrow\}$ complete?*
6. *(**) Is the set $\{0,\Leftrightarrow\}$ complete?*

□

Exercise 29 (↷Duality, recurrence,)** *Consider formulas that only contains the operations $0,1,\neg,\wedge,\vee$. The dual of one of these formulas is obtained by turning conjunctions into disjunctions, 0 into 1 and vice versa. We write A^* to denote the dual of formula A.*

1. *Define A^* y recurrence on these formulas.*
2. *Show that if 2 formulas are equivalent, then so are their duals.*
   Hint: let v be an assignment, and let $\overline{v}$ be the assignment complementary to v (0 becomes 1, 1 becomes 0).
   (a) *Show that for every formula A above, $[A^*]_v = \overline{[A]_{\overline{v}}}$.*
   (b) *Deduce that if 2 such formulas are equivalent, then so are their duals.*
3. *Deduce from the preceding question that if a formula is valid, its dual is contradictory.*

□

Exercise 30 (Canonical Form,*)** *Let $x_{i(0\leq i\leq n)}$ be a list of n distinct variables. In this exercise, all the variables used in formulas are in this list. To any formula A, we associate a boolean function $f(x_1,\ldots,x_n) = A$. This means that for $a_1,\ldots a_n \in \mathbb{B}$, the value of $f(a_1,\ldots,a_n)$ is the value taken by A when $x_1 = a_1,\ldots,x_n = a_n$. We say that two formulas A and B are in the same class if A and B are equivalent.*

1. *How many classes of formulas (with our n variables) are there?*
2. *We denote by $\oplus$ le exclusive-or operation. We have $x \oplus y = 1$ if and only if $x = 1$ or $y = 1$, but not both (hence the name exclusive-or). We can also define this operation as $x \oplus y = \neg(x \Leftrightarrow y)$. It should be clear that this operation is commutative, associative, that $\wedge$ distributes on $\oplus$ and that 0 is the neutral element for $\oplus$. Express negation, disjunction, implication and equivalence as an exclusive-or sum of 1's and conjunctions or variables (i.e. like a formula in disjunctive normal form, except that the $\oplus$ is used instead of $\vee$).*
3. *Show that every boolean formula is equivalent to an exclusive-or sum of 1's and conjunctions or variables.*
4. *As a convention, the product of zero variables is considered to be 1 and the exclusive-or sum of zero variables is 0. Show that every boolean formula is equivalent to an exclusive-or sum of products such that not two monomials have the same set of variables.*
5. *Fix an arbitrary ordering of the n variables used in the formulas (when variables are identifiers, we often use the lexicographic order). An exclusive-or sum of products of variables is in Reed-Muller Canonical Form if and only if:*
   — *the products do not contain any repeated variables, and inside each products, the variables are ordered from left to right in increasing order,*
   — *the products are orderes from left to right in inverse lexicographic order deduced from the ordering of the variables.*

   Show that every boolean formula is equivalent to a formula in Reed-Muller Cacnonical Form. Deduce that the expression of a formula in Reed-Muller Canonical form is unique.

□

Chapter 2

Propositional resolution

Contents

TRUTH tables and transformations into sums of monomials or products of clauses allow to answer various questions: is a formula valid? and is a reasoning correct? However, the computational cost of these methods increases exponentially with the number of variables. To illustrate this, let us consider the following consequence:

$$a \Rightarrow b, b \Rightarrow c, c \Rightarrow d, d \Rightarrow e, e \Rightarrow f, f \Rightarrow g, g \Rightarrow h, h \Rightarrow i, i \Rightarrow j \models a \Rightarrow j$$

Using a truth table you have to test $2^{10} = 1024$ lines. By *deduction*, using the transitivity of implication, we know immediately that the above reasoning is correct. Thus, in propositional logic, deductions can turn out really useful, or even mandatory when the size of the problem to be treated is important. We study now a very simple deductive system. In this system, assumptions and deduced formulas are not any formulas but only *clauses*, that is to say, disjunctions of literals. In addition, the system features a single rule called *resolution*.

Outline: We first introduce the *resolution* method invented in 1965 by J. Alan Robinson [23] in the more general framework of search for first-order proofs. Then we study the `DPLL` algorithm, proposed in the 1960s by Martin Davis, Hilary Putnam, George W. Logemann and Donald W. Loveland [10, 9]. Using the concept of resolution, simplification methods and branching/backtracking, this algorithm efficiently proves whether a formula has a model or not. Although invented more than fifty years ago, it is still the core of most current complete SAT solvers. Notice that in this chapter we adopt the more concise Boolean notation.

2.1 Resolution

We want to prove that $a+c$ is a consequence of $a+b$ and $\overline{b}+c$, i.e. $a+b,\ \overline{b}+c \models a+c$. Using property 1.2.27 on page 17, it is sufficient to prove that the formula $(a+b).(\overline{b}+c) \Rightarrow (a+c)$ is valid. To that effect, we can use a truth table (8 lines) or a simplification of formulas as in exercise 13 on page 32. These two methods are tedious even for such simple

formulas (3 variables only). Using the resolution rule presented in this chapter, we directly deduce the clause $a+c$ from both clauses $a+b$ and $\overline{b}+c$.

We now give the definition of resolution and some useful properties derived from it. We then prove *consistency* and *completeness for refutation* of our system based on deduction. Consistency allows us to state that if a clause C is obtained by resolution starting from a set Γ of clauses then it is a consequence of it, that is to say every model of Γ is a model of C. Completeness for refutation allows us to say that if $\bot$ is a consequence of Γ, in other words if Γ is unsatisfiable, then $\bot$ can be obtained by resolution starting from Γ.

2.1.1 Definitions

At first we introduce the concepts necessary for the definition of resolution. We recall that the notions of clauses and literals are defined in section 1.4 of the previous chapter (page 22). As we shall see later, two clauses sharing the same set of literals play the the same role in resolution. Thus, we define two equal clauses as two clauses with the same set of literals.

Definition 2.1.1 (Members of a clause, equal clauses and sub-clauses)
- *A literal is a* member of a clause*, if it is a member of the set of literals appearing in the clause.*
- *A clause A is* included in a clause *B, if all the literals of clause A are members of clause B. In this case, A is a* sub-clause *of B.*
- *Two clauses are* equal *if they have the same set of literals.*

Example 2.1.2 *We illustrate these concepts with simple examples:*
- *The literal p is a member of the clause $\overline{q}+p+r+p$.*
- *Clause $p+\overline{q}$ is included in clause $\overline{q}+p+r+p$.*
- *Removing the literal p from clause $\overline{q}+p+r+p$ gives the clause $\overline{q}+r$, removing the literal p from clause $p+p+p$ gives the clause $\bot$.*
- *Adding the literal r to clause p gives the clause $p+r$, adding the literal p to the clause $\bot$ gives the p clause.*
- *The clauses $p+\overline{q}$, $\overline{q}+p$, and $p+\overline{q}+p$ are equal.*

Notation: We note (A) the set of literals appearing in clause A. By convention $\bot$ is the empty clause and $s(\bot)=\emptyset$.

Example 2.1.3 $s(\overline{q}+p+r+p+\overline{p})=\{\overline{q},p,r,\overline{p}\}$.

In exercise 37 on page 47 we ask to formalize what a clause is and its set of literals. We introduce the notion of *complementary* literal, noted L^c for a literal L, to manipulate only literals in our reasonings. Indeed, negation is not acceptable because $\overline{\overline{L}}$ is equivalent to L but is not a literal: a literal is either a variable or the negation of a variable but the double negation of a variable is not a literal.

Definition 2.1.4 (Complementary literal) *We denote by L^c the* complementary literal *of a literal L, defined by:*
- *If L is a variable, L^c is the negation of L.*
- *If L is the negation of a variable, L^c is obtained by removing the negation in L.*

Example 2.1.5 $x^c=\overline{x}$ *and* $\overline{x}^c=x$.

We present how to compute a resolvent of two clauses, which is the only rule of the resolution system.

Definition 2.1.6 (Resolvent) *Let A and B be two clauses. The clause C is a resolvent of A and B if and only if there is a literal L such that $L\in s(A)$, $L^c\in s(B)$ and $s(C)=(s(A)-\{L\})\cup(s(B)-\{L^c\})$. We use the following representation when C is a resolvent of A and B :*

$$\frac{A \qquad B}{C}$$

If the clause C is a resolvent of A and B, then we say that C is generated *by A and B, and that A and B are the* parents *of clause C.*

Example 2.1.7 *We give some examples of resolution:*

Property 2.1.8 *If one of the parents of a resolvent is valid, the resolvent is either valid or contains the other parent.*

Proof : This proof is requested in exercise 39 on page 47 . □

Given two clauses A and B, the formula $A+B$ is not a clause if one of the two operands of the disjunction is the empty clause. For example $\bot + p$ is not a clause, although the two operands of the disjunction are clauses. So we introduce a new operand, $\tilde{+}$, to combine disjunction and elimination of $\bot$. This operand will allow us to simplify the definition of resolvent.

Definition 2.1.9 ($C\tilde{+}D$) *Let C and D be two clauses. We write $C\tilde{+}D$ the following clause:*
- *If $C = \bot$ then $C\tilde{+}D = D$,*
- *otherwise if $D = \bot$ then $C\tilde{+}D = C$ otherwise $C\tilde{+}D = C + D$.*

To add the literal L to the clause C is to build $C\tilde{+}L$. With this definition, we can reformulate the resolution rule in a more simple way.

Definition 2.1.10 (Resolvent) *Let A and B be two clauses. The clause C is a* resolvent *of A and B if and only if there is a literal L such that:*
- *L is a member of clause A, L^c is a member of clause B*
- *C is equal to the clause $A'\tilde{+}B'$ where $A' = A - \{L\}$ is obtained by removing L from A and $B' = B - \{L^c\}$ is obtained by removing L^c from B.*

The combination of successive resolutions generates new clauses. A clause resulting from these resolvents will be a proof by resolution of this clause starting from a set of clauses.

Definition 2.1.11 (Proof) *Let Γ be a set of clauses and C a clause. A* proof *of C starting from Γ is a list of clauses that ends with C. Every clause in the proof is equal to an element of Γ or is a resolvent of two clauses preceding it in the proof. The clause C is deduced from Γ, written $\Gamma \vdash C$, if there is proof of C starting from Γ.*

Example 2.1.12 *Let Γ be the set of clauses $\overline{p}+q, p+\overline{q}, \overline{p}+\overline{q}, p+q$. We show that $\Gamma \vdash \bot$ with the following proof:*

We can also see a proof as a *tree* whose leaves are the assumptions and whose internal nodes are obtained by construction of resolvents.

Definition 2.1.13 (Size of a proof) *A proof P of C from a set of clauses Γ has size n if it contains n lines.*

The size of the proof given in example 2.1.12 on the previous page is 8, which is not really obvious on the tree-like representation. We will often use implicitly the two properties stated below in the proofs.

Property 2.1.14 (Monotonicity and composition) *Let Γ, Δ be two sets of clauses and A, B two clauses.*

1. *Monotonicity of deduction: If $\Gamma \vdash A$ and if Γ is included in Δ then $\Delta \vdash A$.*
2. *Composition of deductions: If $\Gamma \vdash A$, $\Gamma \vdash B$ and if C is a resolvent of A and B then $\Gamma \vdash C$.*

Proof : Requested in exercise 38 on page 47. □

2.1.2 Consistency

The consistency of resolution means that if a clause C is obtained after one or several applications of the resolution rule to a set of clauses Γ then any model of Γ is also a model of C (i.e. C is a consequence of Γ). The first part of the proof consists in procing the consistency of a single application of the resolution rule (theorem 2.1.15). We then generalize by induction in theorem 2.1.16.

Theorem 2.1.15 (Consistency of the resolution rule) *If C is a resolvent of A and B then $A, B \models C$.*

Proof : Suppose C is a resolvent of A and B. By definition there is a literal L such that $L \in s(A), L^c \in s(B), s(C) = (s(A) - \{L\}) \cup (s(B) - \{L^c\})$. Let v be an assignment which models both A and B, i.e., $[A]_v = 1$ and $[B]_v = 1$. Let us prove that v is a model of C. We distinguish two possible cases:

□

Theorem 2.1.16 (Coherence of deduction) *Let Γ be a set of clauses and C a clause. If $\Gamma \vdash C$ then $\Gamma \models C$.*

Proof : Assume $\Gamma \vdash C$. By definition, there is a proof P of C from Γ. Assume that for any proof of D from Γ, *shorter* than P, we have $\Gamma \vdash D$. Let's show that $\Gamma \models C$. We have two possible cases:

□

Corollary 2.1.17 *According to theorem 2.1.16 on the facing page, if* $\Gamma \vdash \bot$ *then* $\Gamma \models \bot$*, i.e.* Γ *is unsatisfiable.*

2.1.3 Completeness for refutation

Completeness for refutation is the following property: if Γ is an unsatisfiable set of clauses, there is a proof of the empty clause starting from Γ. We show next in this section the completeness for refutation when Γ is a *finite* set of clauses.

We prove the refutational completeness by induction on the number of variables. For this, we build from Γ two sets of clauses both containing one variable less. More specifically, we build the sets $\Gamma[L := 1]$ and $\Gamma[L := 0]$ by assigning a literal L in Γ to true and false, respectively. We then study the properties of these two sets with respect to Γ (lemmas 2.1.22 and 2.1.23 on the following page). These properties will be used during the induction step.

Definition 2.1.18 ($\Gamma[L := 1]$) *Let* Γ *be a set of clauses and L a literal.* $\Gamma[L := 1]$ *is the set of clauses obtained by removing the clauses of which L is a member and removing* L^c *from the other clauses. Similarly* $\Gamma[L := 0] = \Gamma[L^c := 1]$.

$\Gamma[L := x]$ is equal to the set of clauses obtained by giving L the value x ($x = 0$ or $x = 1$) and simplifying the result.

Example 2.1.19 *Let* Γ *be the set of clauses* $\overline{p}+q, \overline{q}+r, p+q, p+r$*. We have:*

— $\Gamma[p := 1] =$

— $\Gamma[p := 0] =$

To give an intuition of how to compute $\Gamma[p := 1]$ *and* $\Gamma[p := 0]$*, we consider the product of clauses* $(\overline{p}+q)(\overline{q}+r)(p+q)(p+r)$ *and we observe:*

— $(\overline{1}+q)(\overline{q}+r)(1+q)(1+r) = q(\overline{q}+r) = \Gamma[p := 1]$.

— $(\overline{0}+q)(\overline{q}+r)(0+q)(0+r) = (\overline{q}+r)qr = \Gamma[p := 0]$.

We write $v[L := 1]$ the assignment that gives value 1 to L, value 0 to L^c and does not change the value of other literals.

Definition 2.1.20 ($v[L := 1]$) *Let v be an assignment, the assignment* $v[L := 1]$ *is the assignment identical to v except possibly for x the variable of L. If* $L = x$ *then* $v[L := 1](x) = 1$*, if* $L = \overline{x}$ *then* $v[L := 1](x) = 0$.

Similarly $v[L := 0] = v[L^c := 1]$. The following property is derived from the fact that if Γ has a model v, then v gives either value 1 or value 0 to the literal L.

Property 2.1.21 *Let* Γ *be a set of clauses and L a literal.* Γ *has a model if and only if* $\Gamma[L := 1]$ *or* $\Gamma[L := 0]$ *has a model.*

Lemma 2.1.22 *Let* Γ *be a set of clauses, C a clause and L a literal. If* $\Gamma[L := 1] \vdash C$ *then* $\Gamma \vdash C$ *or* $\Gamma \vdash C \tilde{+} L^c$.

Reminder: The $\tilde{+}$ operation is introduced in definition 2.1.9 on page 37 .

Lemma 2.1.23 *Let Γ be a set of clauses, C a clause and L a literal. If $\Gamma[L := 0] \vdash C$ then $\Gamma \vdash C$ or $\Gamma \vdash C \tilde{+} L$.*

Proof : Suppose $\Gamma[L := 0] \vdash C$. Since $\Gamma[L := 0] = \Gamma[L^c := 1]$ and $L^{cc} = L$, according to lemma 2.1.23 we have $\Gamma \vdash C$ or $\Gamma \vdash C \tilde{+} L$. □

Theorem 2.1.24 *Let Γ be a finite set of clauses. If Γ is unsatisfiable then $\Gamma \vdash \bot$.*

Proof : Suppose Γ is unsatisfiable. Let n be the number of variables in Γ. We show that $\Gamma \vdash \bot$ by induction on n. Suppose that for any set Δ of unsatisfiable clauses with less than n variables, we have $\Delta \vdash \bot$.

Base case: Suppose $n = 0$. So $\Gamma = \emptyset$ or $\Gamma = \{\bot\}$. The first case is impossible, because the empty set is valid (any assignment is a model). So $\Gamma = \{\bot\}$ and hence $\Gamma \vdash \bot$.

Induction: Suppose $n > 0$. Let x be a variable in Γ. Based on property 2.1.21 on the previous page, $\Gamma[x := 0]$ and $\Gamma[x := 1]$ are unsatisfiable. Since the variable x does not appear in these two sets of clauses, the induction hypothesis applies, so: $\Gamma[x := 0] \vdash \bot$ and $\Gamma[x := 1] \vdash \bot$. From lemmas 2.1.22 on the preceding page and 2.1.23, we deduce either $\Gamma \vdash \bot$, or $\Gamma \vdash \bar{x}$ and $\Gamma \vdash x$. In the first case, we have completed the proof. In the second case, since $\bot$ is a resolvent of $\bar{x}$ and x, we also have $\Gamma \vdash \bot$.

□

Corollary 2.1.25 *Let Γ be a finite set of clauses. Γ is unsatisfiable if and only if $\Gamma \vdash \bot$.*

Proof : This corollary is an immediate consequence of the above lemma and the remark made at the beginning of paragraph 2.1.3 on the preceding page. □

2.1.4 Reduction of a set of clauses

To reduce a set of clauses is to remove the valid clauses and clauses containing *another* clause of the set. A set of clauses is reduced if it is no longer reducible.

Definition 2.1.26 (Reduced set of clauses) *A set of clauses is* reduced *if it does not contain any valid clause and no clause is included in another clause.*

Example 2.1.27 *Reduction of the set of clauses $\{p+q+\bar{p}, p+r, p+r+\bar{s}, r+q\}$ gives the reduced set :*

Property 2.1.28 *A set of clauses E is equivalent to the reduced set of clauses obtained from E.*

Proof :

□

2.2 Davis-Putnam-Logemann-Loveland algorithm

The Davis-Putnam-Logemann-Loveland method (`DPLL`) was created in 1960 by Martin Davis and Hillary Putnam, then improved in 1962 by Martin Davis, George Logemann and Donald Loveland. The `DPLL` algorithm allows decide whether a set of clauses is satisfactory. Although this algorithm dates back more than 60 years, many variants are still studied, because it is much better than those previously studied (truth table, transformation into sum of monomials or product of clauses, complete strategy). First we notice that there are two types of formula transformations:

1. Those that preserve the meaning, i.e. transform a formula into another equivalent formula.
2. Those that preserve only the satisfiability, i.e. transform a satisfiable formula into another satisfiable formula.

The efficiency of the DPLL algorithm comes from the use of the transformations preserving only satisfiability, as these transformations are less costly than those preserving meaning. We first introduce pure literal elimination, then unit resolution and finally the simplification of valid clauses. These transformations are the basic bricks of the DPLL algorithm. The main idea of this algorithm, starting from a set of "irreducible" clauses, is to choose a variable and consider both possible situations: the case where it is true and the case where it is false. This way, both sets of clauses may be simplified again, otherwise a new variable has to be chosen. In the worst case it boils down to building the truth table corresponding to the clauses, but in practice many simplifications appear and allow this algorithm to conclude efficiently.

2.2.1 Deletion of clauses containing pure literals

As we do not seek to preserve the equivalence of formula but only satisfiability, we have the freedom to choose the value of certain variables. In particular a variable which appears only in the form of a positive or negative literal will be said to be *pure*. We can then assign the appropriate value to this variable to make true all the occurrences of the associated literal. This does not change the satisfiability of the considered set of clauses.

Definition 2.2.1 (Pure literal) *Let Γ be a set of clauses and L a literal member of a clause in Γ, L is a* pure literal *(relative to Γ), if no clause in Γ includes the complementary literal of L.*

Lemma 2.2.2 *The deletion of clauses which contain a pure literal, preserves satisfiability.*

Proof is requested in exercise **??** on page ?? .

Example 2.2.3 *Let Γ be the set of clauses $\{p+q+r, \overline{q}+\overline{r}, q+s, \overline{s}+t\}$.*

2.2.2 Unit resolution

Definition 2.2.4 *A* unit clause *is a clause that contains only one literal.*

Unit clauses are also specific clauses which are:

— either pure and will therefore be treated by the previous simplification,
— or potential parents of resolvents, in this case the so-called "unit resolution" of these clauses has to be performed.

Lemma 2.2.5 *Let Γ be a set of clauses. Let L be the set of literals of the unit clauses in Γ. Let Θ be the set of clauses thus obtained from Γ:*

— *If L contains two complementary literals, then $\Theta = \{\bot\}$.*
— *Otherwise Θ is obtained as follows:*
  - *We remove clauses that contain an element of L.*
  - *Inside the remaining clauses we remove the complementary literals of the elements of L.*

Γ has a model if and only if Θ has one.

Proof is requested in exercise 50 on page 48.

Example 2.2.6

— *Let* Γ *be the set of clauses:* $p+q, \overline{p}, \overline{q}$.

— *Let* Γ *be the set of clauses:* $a+b+\overline{d}, \overline{a}+c+\overline{d}, \overline{b}, d, \overline{c}$.

— *Let* Γ *be the set of clauses:* $p, q, p+r, \overline{p}+r, q+\overline{r}, \overline{q}+s$.

2.2.3 Removal of valid clauses

Finally the last transformation used in the DPLL algorithm is to simply remove the valid clauses from a set of clauses. This may seem obvious but this transformation is one of the central ideas of this algorithm, once a value is chosen for a variable, it is implicitly used in the removal of clauses containing pure literals.

Lemma 2.2.7 *Let* Γ *be a set of clauses. Let* Θ *be the set of clauses obtained by removing valid clauses from* Γ. Γ *has a model if and only if* Θ *has one.*

Proof :

— Suppose Γ has a model v, as Θ is a subset of Γ, v is also a model of Θ. So Θ has a model.

— Suppose Θ has a model v. Let v' be an assignment of Γ such that $v'(x) = v(x)$ for any variable x present in both Γ and Θ. Let C be a clause in Γ. If C is also a clause in Θ, then v' is a model of C because v' and v' give the same value to C. If C is not a clause in Θ, then C is valid, therefore any assignment, v' in particular, is a model of C. Hence, Γ has a model: v'.

□

2.2.4 The algorithm

We give a version of the DPLL algorithm in Figure 2.1 on the facing page in the Algo_DPLL function. We can see in the algorithm that we remove only once the valid clauses before actually starting simplifications. This is justified by the fact that the algorithm will not be able to produce new valid clauses. Indeed, it only removes literals from the original clauses. Therefore, it is unnecessary to remove valid clauses during reduction, as there will no longer be such clauses. The function DPLL first tests whether the empty clause has been generated, then performs all possible simplifications before choosing a variable to apply the same function recursively in the case where that variable is assigned to true and in case it is assigned to false.

To get a trace of the algorithm, we delete the valid clauses, abbreviated VAL, then we draw the call tree with the argument of the function and the sets obtained in steps 2 (reduction, abbreviated RED), 3 (pure literal eliminiation, abbreviated PLE) and 4 (unit resolution, abbreviated UR). Because of the "or else", it is useless to continue the construction of this tree as soon as the true answer (attached to an empty set) is obtained. In this case, a model can be exhibited by tracing back recursion, taking into account the simplifications: model must make true each pure literal or unit clause we found.

Example 2.2.8 *We illustrate the application of this algorithm on two sets of clauses.*

— *Let* Γ *be the set of clauses:* $\overline{a}+\overline{b}, a+b, \overline{a}+\overline{c}, a+c, \overline{b}+\overline{c}, b+c$. *We give the trace of the algorithm.*

bool function Algo_DPLL(Γ : set of clauses)

0 Remove valid clauses from Γ.
If $\Gamma = \emptyset$, return (`true`).
Otherwise return (`DPLL`(Γ)

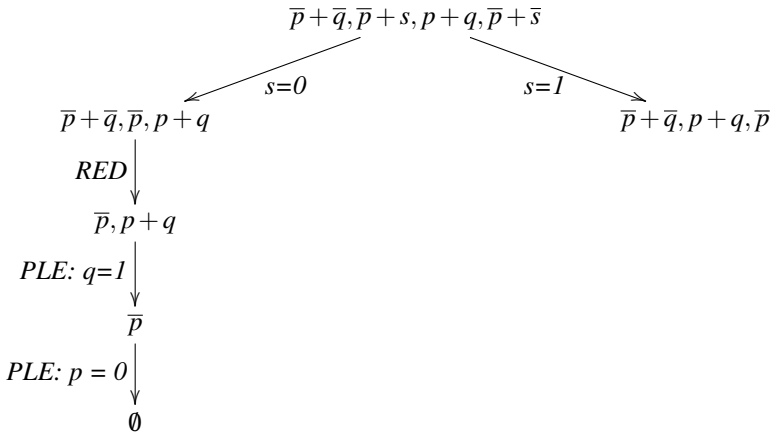
bool function `DPLL`(Γ: set of invalid clauses)

The function returns true if and only if Γ is satisfiable

1 **If** $\bot \in \Gamma$, return(`false`).
If $\Gamma = \emptyset$, return (`true`).
2 Reduce Γ: just remove any clause containing *another* clause.
3 Remove clauses with pure literals from Γ (see paragraph 2.2.1 on page 41).
If the set Γ has been modified, go to 1.
4 Apply unit resolution to Γ (see paragraph 2.2.2 on page 41).
If the set Γ has been modified, go to 1.
5 Choose any variable x appearing in Γ
return (`DPLL`($\Gamma[x := 0]$) or else `DPLL`($\Gamma[x := 1]$)

Figure 2.1: Davis-Putnam-Logemann-Loveland algorithm.

— *Let* Γ *be the set of clauses:* $\overline{p}+\overline{q}, \overline{p}+s, p+q, \overline{p}+\overline{s}$. *We give the trace of the algorithm.*



Since a leave bears the empty set, the set Γ *is satisfiable. There is no point in building the right branch any further.*

Theorem 2.2.9 *The Algo_DPLL algorithm is correct.*

Proof : Reduction and transformations 2.2.1 on page 41, 2.2.2 on page 41 and 2.2.3 on page 42 preserve the existence of a model, so in steps 0 and 1, the algorithm complies with the following invariant: the current value of the set of clauses Γ has a model if and only if Γ has a model. We immediately deduce from it that the algorithm's answers, provided in steps 0 or 1, are correct. At step 5, for the same reasons, the current value of the set of clauses Γ, when calling `DPLL`, has a model if and only if Γ has a model. Assume the recursive calls are correct:

— If `DPLL`$(\Gamma[x := 0])$ is true, by induction $\Gamma[x := 0]$ is satisfiable so Γ is satisfiable (we use result 2.1.21 on page 39: Γ is satisfiable iff $\Gamma[x := 0]$ is satisfiable or $\Gamma[x := 1]$ is satisfiable), which corresponds to the "true" value for `DPLL`(Γ).

— If `DPLL`$(\Gamma[x := 0])$ is false, by induction $\Gamma[x := 0]$ is unsatisfiable. In this case, `DPLL`(Γ) evaluates to `DPLL`$(\Gamma[x := 1])$:

  — Suppose that `DPLL`$(\Gamma[x := 1])$ is true then by induction $\Gamma[x := 1]$ is satisfiable so Γ is satisfiable, which corresponds to the "true" value for `DPLL`(Γ).

  — Suppose that `DPLL`$(\Gamma[x := 1])$ is false, then by induction $\Gamma[x := 1]$ is unsatisfiable. So Γ is unsatisfiable, which corresponds to the "false" value for `DPLL`(Γ).

□

Theorem 2.2.10 *Algorithm Algo_DPLL halts.*

Proof : On step 0, the Algo_DPLL function consists of a test, then either returning the value true or the function call `DPLL`(Γ). Step 0 consists in removing clauses from the starting set, this set being finite, step 0 terminates. So the Algo_DPLL function halts if the `function DPLL`(Γ) halts.

Consider a call of function `DPLL`(Γ). Step 1 consists of a test and elementary instructions, so it terminates. Step 2 is to reduce Γ which is a finite set. Since reduction consists in removing clauses, it terminates. Step 3 is to delete clauses, then possibly return to step 1. The number of clauses being finite, either the algorithm ends in 1, or step 4 begins. The same way, step 4 is to delete clauses and variables, which are in finite numbers, then eventually return to step 1. The number of clauses and variables being finite, either the algorithm ends in 1, or step 5 begins. Step 5 consists in one or two recursive calls on a set of clauses where one variable has disappeared (the value of a variable is fixed). Thus, the recurrence in step 5 ends because at each recursive call the number of variables decreases strictly. Thus, the Algo_DPLL algorithm halts. □

Remark 2.2.11 (Forgetting simplifications) *We have given algorithm Algo_DPLL with "simplification" steps: removal of valid clauses (0), reduction (2), pure literal elimination (3) and unit resolution (4). The algorithm remains correct if we omit these steps, or if we only do part of the simplifications, which is often the case when we build a trace of the execution without a computer. Because without a machine it is common to forget some simplifications, which does not harm because the algorithm remains correct.*

Remark 2.2.12 (Variable selection) *A good choice for the variable x in step (5), is to choose the variable that appears most often. An even better choice consists in choosing the variable that will eventually lead to the most simplifications: trade-offs must be made over time spent choosing the "good" variable and the magnitude of simplifications induced by this choice. Several classic heuristics for variable choice are presented in paragraph 2.2.5.1 on the next page.*

2.2.5 SAT solvers

The *SAT* problem is a decision problem consisting in determining whether a propositional formula in conjunctive normal form admits a model or not. This problem is the reference NP-complete problem [5]. Usually, only one version of this problem, called 3-SAT, where all the clauses of the formula are made of exactly three literals, is considered, because *linear reductions* exist from SAT to 3-SAT. An example of reduction from SAT to 3-SAT is studied in exercise 51 on page 48.

Efficient programs are dedicated to solving the SAT problem [1]. These programs are called *SAT solvers* and are usually classified into two categories:

— "Complete" SAT solvers are generally improved versions of the `DPLL` algorithm. Among the most used complete SAT solvers, we can cite for example *zchaff*, *satz*, *march* or *GRASP*.

— "Incomplete" SAT solvers are semi-decision algorithms that end up finding a model to the formula, if it exists and do not end in the opposite case. These algorithms are based on random walks in the state space. In this category, we can cite algorithms such as *WalkSAT* and *GSAT*.

We now describe, in a non-exhaustive way, the main enhancements to the `DPLL` algorithm used in complete SAT solvers.

[1] In these programs, the 3-SAT formalism is usually preferred to the SAT formalism, because its regular aspect allows for more efficient solutions.

2.2.5.1 Branching heuristic

The choice of the next variable to be affected has a significant impact on the size of the search tree developed by $\widehat{\text{DPLL}}$. Hence, the choice of branching heuristic is decisive in the algorithms based on DPLL. Branching heuristics are often based on syntactic arguments such as the size of clauses, number of occurrences, , *etc*. Among the most well-known heuristics, we can mention:

MOMS : The *MOMS* heuristic (for Maximum Occurences in Clauses of Minimum Size) consists, as its name indicates, in selecting the variable with the most occurrences in the shortest clauses. The choices made by this heuristic favour unit propagation.

JW : The *JW* heuristic (for Jeroslow-Wang) also uses the length of clauses. Indeed, a weight is assigned to each literal depending on the size of the clauses where it appears. The possibility of a variable being selected by JW is inversely proportional to the sum of the sizes of the clauses where it appears.

UP : Unlike the two previous heuristics, the *UP* heuristic (for Unit Propagation) is not based on syntactic arguments. This heuristic tries to select a variable predicting its effect on model search. It consists in computing a weight for each variable according to the simplifications it generates by unit propagation.

2.2.5.2 Adding clauses

The idea is to add clauses, in pre-treatment, to the initial set of clauses. The new set of clauses remains equivalent to the initial set. For example, we can mention *restricted resolution* which consists in adding resolvents with limited size. This kind of method allows to reduce the size of the search tree by bringing contradictions earlier in the search.

2.2.5.3 Conflict analysis and non-chronological backtracking

When the DPLL algorithm reaches a contradiction (a conflict), it returns (chronologically) to the last variable branching it performed. However, this branching is not necessarily one of the decisions that led to the contradiction. Conflict analysis addresses this problem by focusing on decisions at the origin of the conflict. This analysis allows to perform a non-chronological backtracking, i.e. a to return directly to the level, higher in the tree, where the conflict originated without testing intermediate levels.

2.2.5.4 Learning

Learning consists in analysing and learning the reasons for a contradiction: when a partial assignment leads to a contradiction, it is possible to isolate a subset of assignments and a subset of clauses, which are responsible for the conflict. From these assignments it is possible to build (learn) a clause which is implied by these clauses. This new clause is added to the set of clauses of the problem. The relevant assignments are determined by a dependency graph between clauses and assignments, called *implication graph*. The learned clauses allow not to repeat several times the same "errors" in the search tree.

2.2.5.5 Restart

Most current SAT solvers use a restart strategy, which consists in restarting the search taking into account new clauses learned during the previous search. Most solvers perform a restart after a threshold of learned clauses is reached. Thus, after each restart, the search space traversal changes because the set of clauses is modified. It should be noted that most solvers regularly erases the learned clauses. Thus, mixing restarts and erasels may in some cases compromise the completeness of solvers. A setting of restarts is therefore necessary to keep this latest property. The Chaff solver, for example, increases at each restart the threshold beyond which a clause is forgotten. As a result, more and more clauses are being kept, which ensures completeness.

2.2.5.6 Lazy data structures

In modern SAT solvers, 80% of the computing time is used up by unit propagation. Thus, so-called "lazy" data structures were introduced with the aim of optimizing unit propagation.

2.3 Exercises

Exercise 31 (Resolvent) *Through resolution, we have:*

$$\frac{a+b \quad \overline{a}+\overline{b}}{b+\overline{b}}$$

Show that: $b+\overline{b} \not\models (a+b).(\overline{a}+\overline{b})$. □

Exercise 32 (Resolvent) *We recall that two clauses are equal if and only if they have the sets of literals.*

— *Are the clauses* $p+q+\overline{r}+q+p+s+q+\overline{r}$ *and* $s+q+\overline{r}+p$ *equal?*
— *Are the clauses* $p+q+r+p$ *and* $q+r+q+r+q+r$ *equal? Is one included in the other? Is one the consequence of the other? Are they equivalent?*
— *Give all the resolvents of the clauses* $a+\overline{b}+c$ *and* $a+b+\overline{c}$. *Are these resolvents valid?*

□

Exercise 33 (Proof) *The following sets of formulas are unsatisfiable.*

— $\{a, a \Rightarrow b, \overline{b}\}$.
— $\{a+b, \overline{a}+c, \overline{a}+\overline{d}, d+\overline{c}, \overline{b}+a\}$.
— $\{a+b+c, \overline{a}+b, \overline{b}+c, \overline{c}+a, \overline{a}+\overline{b}+\overline{c}\}$.

Prove that fact using resolution. □

Exercise 34 (Formalization and Resolution,*) *We note that: " x unless y " can be formalized as* $\overline{x \Leftrightarrow y}$. *In a haunted house, ghosts show their presence in two ways:* ***obscene chants*** *and* ***sardonic laughs***. *However, we can influence their behaviour by playing the organ or burning incense. Given the following:*

(i) The ghosts don't do obscene chants unless we play the organ while the ghosts are not laughing.
(ii) If we burn incense, the ghosts laugh if and only if they are also chanting.
(iii) At the moment, the ghosts are chanting, but not laughing.

And our conclusion:

(iv) At the moment, we are playing the organ, but not burning incense.

We define the following:

— *c: the ghosts are chanting.*
— *o: we are playing the organ.*
— *l: the ghosts are laughing.*
— *i: we are burning incense.*

1. *Transform the expression* $\overline{x \Leftrightarrow y}$ *into a product of clauses.*
2. *Formalize the hypotheses and the negation of the conclusion into a product of clauses.*
3. *Using resolution, prove that the reasoning is correct.*

In other words, transform the conjunction of the hypotheses and the negation of the conclusion into a product of clauses and deduce the empty clause. □

Exercise 35 (Proof,*) *Show, using a proof by resolution, the correctness of the following reasoning:*

$$r+q \Rightarrow t, t.q \Rightarrow r, q \models t \Leftrightarrow r.$$

□

Exercise 36 (Formalization and Proof,*) *Show, using resolution, that the following reasoning is correct:*

— *The weather is nice unless it is snowing.*
— *It is raining unless it is snowing.*
— *The weather is nice or it is raining.*
— *Hence, it is not snowing.*

□

Exercise 37 (↪Defining a Clause) *A clause is either the empty clause, or a (non-empty) disjunction of literals. Give a inductive definition of a clause and define by recurrence the function s such that $s(C)$ is the set of literals in clause C.* □

Exercise 38 (↪Proof) *Prove property 2.1.14 on page 38 on the monotonicity and composition.* □

Exercise 39 (↪Property of Resolution) *Prove property 2.1.8 on page 37.* □

Exercise 40 (↪Resolution does not add literals,*) *Let Γ be a set of clauses. A literal in Γ is a literal that appears in one of the clauses of Γ. Show that any clause deduced from Γ only contains literals in Γ.* □

Exercise 41 (Formalization and resolution) *Consider the following hypotheses:*

1. *If Peter misses his tournament then Peter will be depressed.*
2. *If the weather is nice then Peter will go to the swimming pool.*
3. *If Peter does not go to the swimming pool he will be depressed.*
4. *When at the swimming pool, Peter does not train.*
5. *Peter will miss his tournament if he does not train.*

We would like to demonstrate that under such hypotheses, the following conclusion can be derived:

6. *Peter will be depressed.*

You will proceed as follows:

- *Formalize the hypotheses and the negation of the conclusion.*
- *Deduce a set of clauses equivalent to your formal statements.*
- *Show that the conclusion can be correctly derived from the hypotheses, by proving using the resolution method that the set of clauses is contradictory.*

□

Exercise 42 (Formalisation and resolution)

The Beatles were a rock band formed in the 60's in Liverpool. It was composed of four boys: Ringo Starr, Paul MacCartney, John Lennon and Georges Harrison. It has been quite hard to decide who would play which instrument, as the following extract from their discussion will show:

Paul said: *"If Ringo doesn't play the guitar, then I will play the bass guitar and John the guitar."*

Georges said: *"I will play the guitar if, and only if, John plays it too."*

John said: *"If Paul plays the bass guitar, then Georges will play the guitar."*

Ringo said: *"I will play the drums and therefore, not the guitar."*

After their talk, they settled on the following power four:

- *Ringo would play the drums,*
- *Paul would play the bass guitar*
- *John and Georges would both play the guitar.*

We will now show that every member of the band is happy with this conclusion.

1. *Formalise the 4 hypotheses and the conclusion using the following propositional variables:*
   - *RD: "Ringo plays the drums",*
   - *RG: "Ringo plays the guitar",*
   - *PB: "Paul plays the bass guitar",*
   - *JG: "John plays the guitar",*
   - *GG: "Georges plays the guitar".*
2. *Transform the hypotheses and the negation of the conclusion into clauses.*
3. *Prove, using resolution, that the conclusion is a consequence of the hypotheses.*

□

Exercise 43 (Reduction, X1603 Andrews) *Soit l'ensemble de clauses :*

$$\{p+q,\overline{p}+r+\overline{q}+p,p+\overline{r},q+\overline{p}+\overline{q},q+\overline{r}+p,r+q+\overline{p}+\overline{r},\overline{r}+q\}.$$

1. *Reduce this set (reduction is defined in 2.1.4 on page 40).*
2. *Determine whether this set is satisfiable using resolution.*

□

Exercise 44 (DPLL) *Consider the following set of clauses:*

$$\{\overline{a}+\overline{b}+\overline{f},a+b+f,e+\overline{a},\overline{a}+\overline{b},\overline{a}+c,d+a+\overline{d},a+b,\overline{a}+\overline{c}+\overline{d},d\}.$$

— *Use algorithm Algo_DPLL on this set of clauses.*
— *Conclude whether or not it is satisfiable.*
— *If it is satisfiable, give a model obtained from the trace of the algorithm.*

In the tree of calls, identify the steps as follows:

— *Elimination of valid clauses (VAL).*
— *Reduction (RED).*
— *Pure literal elimination (PLE).*
— *Unit resolution (UR).*

In addition, note the assignments made at each step of the algorithm so that the model can easily be recovered. □

Exercise 45 (DPLL) *Use algorithm Algo_DPLL to determine whether or not the following sets of clauses are satisfiable:*

— $\{a+b+c+d+e+f,\overline{a}+b,\overline{b}+a,\overline{c}+d,\overline{d}+c,\overline{b}+\overline{c},\overline{b}+c,b+\overline{c},\overline{e},\overline{f}\}$.
— $\{a+b+c+d+f,\overline{a}+b,\overline{b}+a,\overline{c}+d,\overline{d}+c\}$.
— $\{b+j+\overline{a},a+j+\overline{b},b+a+j,a+j,j+b,\overline{b}+\overline{j},\overline{j}+b,j+s,\overline{s}+\overline{b}\}$.
— $\{a+\overline{c}+d,\overline{b}+c+f,b+\overline{e}+f,\overline{c}+e+\overline{f},e+f,c+d,\overline{a},\overline{e}+\overline{f}\}$.

Give a trace of the algorithm. □

Exercise 46 (Complete Strategy) *Let* Γ *be the following product of clauses:*

$$(a+b+\overline{c}).(b+\overline{c}).(c+\overline{c}).(b+c).(\overline{a}+\overline{b}+c).(a+\overline{b}+c).$$

Determine using the complete resolution strategy whether Γ *is unsatisfiable, or if it has a model. Give a trace of the algorithm. Give the proof(s) you obtained. If* Γ *has a model, show it.* □

Exercise 47 (Complete Strategy) *Consider the following set of clauses:*

$$\{p+q,\overline{p}+s,\overline{s}+t,\overline{t},\overline{q}+r,\overline{r},\overline{r}+p+t,q+z+\overline{z},\overline{q}+r+s\}.$$

Use the complete strategy on this set of clauses and determine whether it is satisfiable or not. □

Exercise 48 (Complete Strategy) *Consider the function* f *such that* $f(x,y,z)=0$ *if and only if there is an even number of arguments with value* 1*. Express* f *as a product of clauses using the method described in subsection 1.6 page 32, then simplify* f *using the complete strategy.* □

Exercise 49 (↪Proof) *Prove Lemma 2.2.2 on page 41.* □

Exercise 50 (↪Proof) *Prove Lemma 2.2.5 on page 41.* □

Exercise 51 (From SAT to 3-SAT,*)** *SAT (short for Satisfiability) is a decision problem stated as follows: "given a set of clauses* Γ*, determine if* Γ *has a model". 3-SAT is a restriction of this problem in which each clause in* Γ *contains exactly three literals.*

SAT is an NP-complete problem [5]. In this exercise, we study a polynomial-time reduction from SAT to 3-SAT, thereby showing that 3-SAT is also NP-complete[2].

A polynomial-time reduction from SAT to 3-SAT is a polynomial-time transformation of the set of clauses Γ *into a set of clauses* Γ' *verifying the following two properties:*

[2] We note that the k-SAT is not NP-complete for any value of k: 2-SAT can be solved in polynomial time.

(a) all the clauses in Γ' contain three distinct literals.

(b) Γ has a model if and only if Γ' has a model.

Usually, such reductions also satisfy the following additional property:

(c) Any model for Γ' is also a model for Γ.

The goal of this exercise is to show that the polynomial-time transformation below verify these three properties. Note that the transformation potentially introduces new variables.

Let $\Gamma = \{c_1, \ldots, c_m\}$ be a set of clauses. The reduction consists in replacing every clause $c_i = z_1 + \ldots + z_k$ of Γ (where each z_i is a literal) by a set of clauses C'_i constructed as follows, depending on the value of k:

$k = 1$: $C'_i = \{z_1 + y_1 + y_2,\ z_1 + y_1 + \overline{y_2},\ z_1 + \overline{y_1} + y_2,\ z_1 + \overline{y_1} + \overline{y_2}\}$ where y_1 and y_2 are new variables (not present anywhere else).

$k = 2$: $C'_i = \{z_1 + z_2 + y_1,\ z_1 + z_2 + \overline{y_1}\}$ where y_1 is a new variable.

$k = 3$: $C'_i = \{c_i\}$.

$k > 3$: $C'_i = \{z_1 + z_2 + y_1,\ \overline{y_1} + z_3 + y_2,\ \overline{y_2} + z_4 + y_3,\ \overline{y_3} + z_5 + y_4,\ \ldots,\ \overline{y_{k-3}} + z_{k-1} + z_k\}$ where $y_1, \ldots, y_{k-3}$ are new variables.

Finally, $\Gamma' = \bigcup_{i=1}^{m} C'_i$.

By construction, the reduction certainly verify property (a). The answers to the following questions allow us to prove properties (b) and (c):

1. *Show (without a truth table) that any assignment will give the same value to c_i as to the conjunction of all the clauses in C'_i when c_i consists of only one literal.*
2. *Show (without a truth table) that any assignment will give the same value to c_i as to the conjunction of all the clauses in C'_i when c_i consists of two literal.*
3. *Let c_i be a clause with more than 3 literals. Show that if c_i has a model, then C'_i also has a model.*
4. *Let c_i be a clause with more than 3 literals. Show that any model for C'_i is also a model for c_i.*
5. *The reduction above maintains satisfiability. Show that the reduction does not maintain the meaning of the set of formulas. That is, the conjunction of all the clauses in Γ is not equivalent to the conjunction of all the formulas in Γ'.*

□

Chapter 3

Natural Deduction

Contents

A proof in mathematics courses is a decomposition of a reasoning into obvious elementary steps. In this situation, without knowing it we perform *natural deduction*. In this chapter, we present a formal system which is a particular modeling of natural deduction. Natural deduction and sequent calculus were introduced in 1934 by Gerhard Gentzen (1909-1945) [12, 13]. The proofs in our system are not trees as in Gehard Gentzen's original system but look more like the mathematicians' proofs. We also explain how to link these two formalisms by building trees close to those originally introduced. We choose to talk only about *classical logic*, as opposed to *intuitionistic logic* which is obtained in *omitting* the *Reductio ad absurdum* rule (the last rule in Table 3.1 on page 53).

Preliminary remark: To simplify the study of natural deduction, we consider that true, negation and equivalence are *abbreviations* defined as follows:

— $\top$ abbreviates $\bot \Rightarrow \bot$.
— $\neg A$ abbreviates $A \Rightarrow \bot$.
— $A \Leftrightarrow B$ abbreviates $(A \Rightarrow B) \wedge (B \Rightarrow A)$.

Two formulas will be considered *equal*, if the formulas obtained by eliminating abbreviations are identical. For example, the formulas $\neg\neg a$, $\neg a \Rightarrow \bot$ and $(a \Rightarrow \bot) \Rightarrow \bot$ are equal. It is clear that two equal formulas, as defined above, are equivalent (see exercise 55 on page 61) and we implicitly use this property when we replace a formula with another formula equal up to the abbreviations.

Outline: We first describe our system of natural deduction rules and we introduce the concept of proof in a context. Then we present tactics to help the reader write proofs. We show the consistency and the completeness of our system. Finally, we present a tool for automatically building proof starting from a formula, or to display a counter-example.

3.1 The formal system of natural deduction

First, we present the rules of the natural deduction system. They constitute the admissible basic reasonings. Then we explain the notion of proof which is built by successive applications of these rules.

Definition 3.1.1 (Rule) *A* rule *consists of formulas called premises $H_1, \ldots, H_n$ and a conclusion. The premises are written above a line and the only formula below that line is the conclusion of the rule. The name of a rule is written at the same level as the line separating premises and conclusion.*

$$\frac{H_1 \ldots H_n}{C}\ R$$

The natural deduction we present features ten deduction rules that are grouped in table 3.1 on the facing page. The fundamental rule is: if we can deduce a formula B from an hypothesis A, then we can deduce $A \Rightarrow B$ without this hypothesis. We describe each of the ten rules of the deductive system we consider. The application of these rules, and only these rules, will allow us to prove the correctness of reasonings. So it's important to understand fully the way each of them works. We have two sets of rules for each connective: rules of introduction and rules of elimination. The intuition behind the introduction rules is to generate a logical symbol in order to gradually build the formula to be proved. The elimination rules are used to simplify a reasoning or to obtain a new formula based on the reasonings already performed and thus make a logical connective disappear. Furthermore, we have two special rules, which we will describe last, one to eliminate two successive occurrences of negation, and another to deduce anything from falsehood.

3.1.1 Conjunction rules

The conjunction introduction rule, noted $\wedge$I, means simply that if we have a proof of A and a proof of B, then we can deduce a proof of the proposition $A \wedge B$. We can also say that a proof of the proposition $A \wedge B$ can be broken down into a proof of A and a proof of B.

$$\frac{A \quad B}{A \wedge B}\ \wedge I$$

The conjunction elimination rules, noted $\wedge E1$ and $\wedge E2$, allow to deduce, from the conjunction of two formulas $A \wedge B$, either the formula A or the formula B. Thus we eliminated the conjunction connective.

$$\frac{A \wedge B}{A}\ \wedge E1 \qquad \frac{A \wedge B}{B}\ \wedge E2$$

3.1.2 Disjunction rules

The rules for disjunction introduction are the dual for the conjunction elimination rules. The two introduction rules, noted $\vee$I1 and $\vee$I2, create a disjunction of two formulas starting from one of the two formulas. These rules mean simply that if A is true then $A \vee B$ and $B \vee A$ are true too, regardless of proposition B.

$$\frac{A}{A \vee B}\ \vee I1 \qquad \frac{A}{B \vee A}\ \vee I2$$

The disjunction elimination rule, noted $\vee E$, is more complex. To deduce C from the disjunction $A \vee B$ one must also prove the following premises: A implies C and B implies C. This rule formalizes the notion of *proof by cases* used in mathematics: assume that in a given environment, we want to prove C while two cases, A or B, are possible; so we focus on the case where A is satisfied and we prove C, then we focus on the case where B is satisfied and we prove C.

$$\frac{A \vee B \quad A \Rightarrow C \quad B \Rightarrow C}{C}\ \vee E$$

3.1.3 Rules of implication

The rule of implication elimination $\Rightarrow E$ says if we are able to get a proof of A and a proof of $A \Rightarrow B$, then we have a proof of B. This rule corresponds to the *modus ponens*.

$$\frac{A \quad A \Rightarrow B}{B}\ \Rightarrow E$$

The fundamental rule of our system is the implication introduction rule $\Rightarrow I$: if we can deduce a formula B from a hypothesis A, then we can deduce $A \Rightarrow B$ without this hypothesis. This rule is summarized by the diagram below, where the notation $[A]$ indicates that if A is an assumption in the proof of B, this assumption is *removed* from the proof of $A \Rightarrow B$, i.e. it no longer serves to prove $A \Rightarrow B$.

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \Rightarrow B} \Rightarrow I$$

3.1.4 Two special rules

Finally, we have two special rules:

— The first one allows to deduce anything from falsehood: this is the "falsehood rule",, noted Efq, for the Latin formula "ex falso, quodlibet", indicating that from falsehood, we can deduce what we want.

$$\frac{\bot}{A} \; Efq$$

— The second one is the reduction to absurdity rule, noted *RAA* for "reductio ad absurdum". It eliminates two successive occurrences of negation.

$$\frac{\neg\neg A}{A} \; RAA$$

We group all the rules of natural education in table 3.1.

Introduction	Elimination
Falsehood rule	
Reduction to absurdity	

Table 3.1: Set of natural deduction rules.

3.1.5 Proofs in natural deduction

Intuitively, a proof is the successive application of natural deduction rules. In the original system introduced by Gerhard Gentzen, a proof is represented by a tree of formulas (some formulas may be be marked as removed). We chose to represent a proof by a succession of reasoning steps based on the natural deduction rules, in order to be closer to the proofs in mathematics. In the previous chapter, a proof by resolution consisted of a list of clauses. In natural deduction, during a proof, we may add hypotheses and remove them, which makes the definition of a proof more complex. Therefore, we introduce several intermediate concepts in order to define clearly the concept of proof in natural deduction. First we introduce the components of a proof, i.e. "proof lines", then the the concept of proof draft, and finally the contexts of these proof lines in order to define a proof in natural deduction.

3.1.5.1 Proof draft

A proof consists of a sequence of lines which are either a formula, or a formula preceded by the keyword "`Assume` " or "`Therefore`".

Definition 3.1.2 (Proof line) *A proof* line *is one of the following three:*
- — `Assume` formula,
- — formula,
- — `Therefore` formula.

A proof draft is a proof under construction, intuitively it has at least as many `Assume` as it has `Therefore`.

Definition 3.1.3 (Proof draft) *A proof* draft *is a sequence of lines such that, in any prefix of the sequence, the number of lines beginning with the keyword* `Assume` *is at least equal to the number of lines starting with the keyword* `Therefore`.

Example 3.1.4 *In the example below, the sequence of lines 1 to 3 is a proof draft. On the other hand, the sequence of lines 1 to 5* is not a proof draft *because in the sequence of lines 1 to 4, the number of lines beginning with the keyword* `Therefore` *exceeds the number of lines beginning with the key word* `Assume`.

number	*line*
1	`Assume` a
2	$a \vee b$
3	`Therefore` $a \Rightarrow a \vee b$
4	`Therefore` $\neg a$
5	`Assume` b

3.1.5.2 Context of the lines in a proof draft

In order to be able to apply the implication introduction rule, we need to know at each step of a proof the formulas that are usable assumptions. To do this, let us associate each line of a proof draft to a *context*, which is the sequence of the assumptions introduced by the `Assume` lines and not removed by the `Therefore` lines. Since in any prefix of a proof draft, the number of lines starting with the keyword `Assume` is at least equal to that of lines beginning with the keyword `Therefore`, the context of the line preceding a `Therefore` line is not empty, and consequently the context of each line in a proof draft is well-defined. In the following, we represent the contexts of each line by replacing each hypothesis in the context with the number of the line where this hypothesis was introduced.

Definition 3.1.5 (Context) *We number the lines of a proof draft from* 1 *to n. For i from* 1 *to n, the list of formulas* Γ_i *is the* context *of line i. The list* Γ_0 *is empty and the lists of formulas* Γ_i *are defined as follows:*
- — *If line i is "*`Assume` *A", then* $\Gamma_i = \Gamma_{i-1}, i$.
- — *If line i is "A" then* $\Gamma_i = \Gamma_{i-1}$
- — *If line i is "*`Therefore` *A" then* Γ_i *is obtained by removing the last formula in* Γ_{i-1}

The list Γ_i *is the* context *of line i.*

We present our proofs in tabular form, allowing us to number the lines in a proof, and to trace the usable contexts and the rules used to generate a new line.

Example 3.1.6 *Here we illustrate the management of contexts on simple proof draft.*

context	*number*	*line*	*rule*
1	*1*	`Assume` a	
1,2	*2*	`Assume` b	
1,2	*3*	$a \wedge b$	$\wedge I$ *1,2*
1	*4*	`Therefore` $b \Rightarrow a \wedge b$	$\Rightarrow I$ *2,3*
1,5	*5*	`Assume` e	

3.1.5.3 Proofs

We specify the concept of conclusion and usable formula in order to define and manipulate proofs in natural deduction.

Definition 3.1.7 (Conclusion, usable formula) *We define the concept of conclusion and usable formula:*

— *The formula on a line of a proof draft is the* conclusion *of the line.*

— *The conclusion of a line is* usable *as long as its context (i.e. the assumptions that allowed to deduce it) is present.*

In other words, the conclusion of line i is usable on line i and on all the following lines whose context has the context of line i as a prefix.

Example 3.1.8 *In the example below, the conclusion of line 2 is usable on line 2, and not beyond, because on line 3, the assumption that allowed it to be deduced is removed.*

context	*number*	*line*	*rule*
1	*1*	`Assume` a	
1	*2*	$a \vee b$	$\vee I$ *1 1*
	3	`Therefore` $a \Rightarrow a \vee b$	$\Rightarrow I$ *1,2*
4	*4*	`Assume` c	
4	*5*	$c \vee a$	$\vee I$ *4*
	6	`Therefore` $c \Rightarrow c \vee a$	$\Rightarrow I$ *4,5*
	7	$(a \Rightarrow a \vee b) \wedge (c \Rightarrow c \vee a)$	$\wedge I$ *3,6*

We define a proof in an environment. An environment is a set of formulas assumed to be "true". Intuitively, to perform a proof in an environment is to be able to use the formulas in the environment without having to prove them.

Definition 3.1.9 (Proof) *Let* Γ *be a set of formulas,* a proof in the environment Γ *is a proof draft having the following properties:*

1. *For any "*`Therefore` A*" line, the formula A is equal to $B \Rightarrow C$, where B is the last formula in the previous line context, and where C is a formula usable on the previous line or equal to an element of the environment* Γ.
2. *For any "A" line, the formula A is the conclusion of a rule (other than the implication introduction rule) whose premises are usable on the previous line or are elements of the environment* Γ.

The "`Therefore` A" line is an application of the implication introduction rule. Indeed, C is deduced from Γ or assumptions in the previous line. Since the list of assumptions of the previous line ends with B, we can deduce $B \Rightarrow C$, without the hypothesis B.

Definition 3.1.10 (Proof of a formula) *A proof of the formula A in the environment Γ is either the empty proof when A is an element of Γ, or a proof whose last line has A as a conclusion and an empty context.*

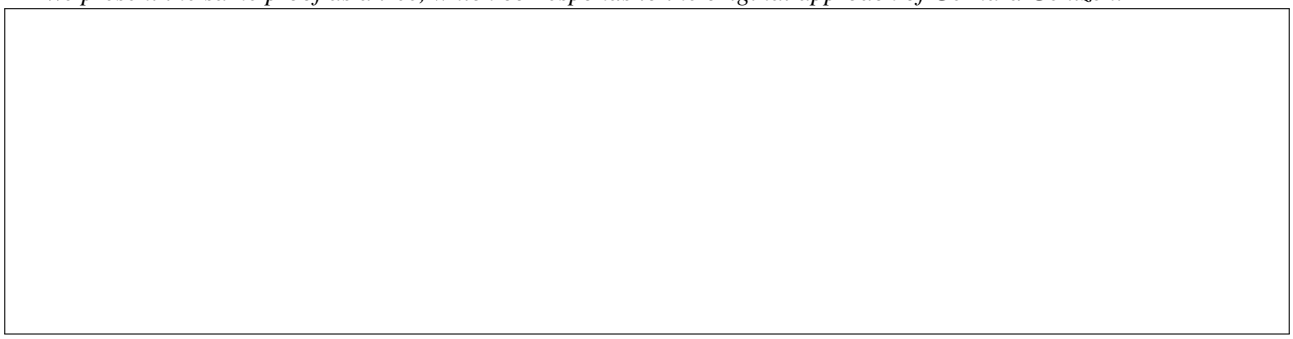
We note $\Gamma \vdash A$ the fact that there is a proof of A in the environment Γ and $\Gamma \vdash P : A$ the fact that P is a proof of A in the environment Γ. When the environment is empty, we omit it, in other words we abbreviate $\emptyset \vdash A$ into $\vdash A$. When we ask for a proof of a formula without indicating an environment, we assume the environment is empty.

Example 3.1.11 *Prove* $(a \Rightarrow b) \Rightarrow (\neg b \Rightarrow \neg a)$.

context	*number*	*proof*	*rule*
	1		
	2		
	3		
	4		
	5		
	6		
	7		
	8		

The proof itself is what appears in the "proof" column. We added the numbering of the proof lines, the justifications which indicate the rules we used and the contexts of each proof line.

We present the same proof as a tree, which corresponds to the original approach of Gerhard Gentzen.

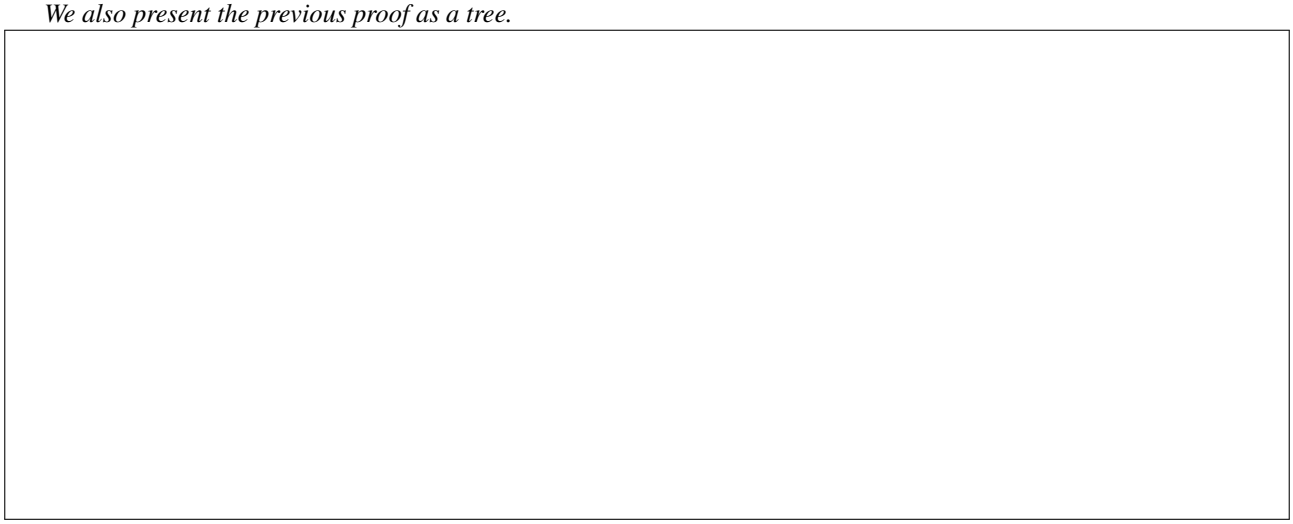


We notice that to lift a hypothesis it is customary to cross out this hypothesis.

Example 3.1.12 *We give a proof of $\neg A \vee B$ in the environment $A \Rightarrow B$. We suggest numbering i, ii, iii, etc. the formulas in the environment to distinguish these numbers and the numbers of the proof lines.*

environment			
reference		*formula*	
i		$A \Rightarrow B$	
context	*number*	*proof*	*rule*
	1		
	2		
	3		
	4		
	5		
	6		
	7		
	8		
	9		
	10		

We also present the previous proof as a tree.



As was pointed out in the preliminary remarks, consider equal two formulas which are identical up to abbreviations. So in the previous example, we identified $A \Rightarrow \bot$ and $\neg A$ as well as $(\neg A \vee B) \Rightarrow \bot$ and $\neg(\neg A \vee B)$.

3.2 Proof tactics

We give advice on how to build a proof. This advice is the basis of the software presented in paragraph 3.5 on page 59. To prove the formula A in an environment Γ, we apply *in order* the following rules, where the order was chosen so as to delay as far as possible the use of the *RAA* rule:

1. If $A \in \Gamma$, then the obtained proof is empty.
2. If A is the conclusion of a rule whose premises are in Γ, then the obtained proof is "A".
3. If Γ contains a contradiction, i.e. a formula B and a formula $\neg B$, then the obtained proof is "$\bot$, A".
4. If $A = B \wedge C$, then:
   — prove B: Let P be the proof obtained for B,
   — prove C: Let Q be the proof obtained for C.
   The proof obtained for A is "P, Q, A".
   Proofs may fail (if one asks to prove a formula that cannot be proved in the given environment): If proof of B or C fails, so does the proof of A. To simplify the following, we no longer stress the failure cases, unless they have to be followed by another proof.
5. If $A = B \Rightarrow C$, then prove C under hypothesis B (which is added to the environment). Let P be the proof obtained for C, the proof obtained for A is "`Assume` B, P, `Therefore` A".
6. If $A = B \vee C$, then prove B: If P is the proof obtained for B, then "P, A" is the proof obtained for A. If the proof of B fails, then prove C: If P is the proof obtained for C, then "P, A" is the proof obtained For A. *If the proof of C fails, try the next rules*.
7. If $B \wedge C$ is in the environment, then prove A using the formulas B, C, which replace $B \wedge C$ in the environment and let P be the result of this proof. Then "B, C, P" is proof of A in the initial environment.
8. If $B \vee C$ is in the environment then:
   — prove A in the environment where B replaces $B \vee C$: Let P be the obtained proof,
   — prove A in the environment where C replaces $B \vee C$: Let Q be the obtained proof.
   The proof of A is then "`Assume` B, P, `Therefore` $B \Rightarrow A$, `Assume` C, Q, `Therefore` $C \Rightarrow A$, A".
9. If $\neg(B \vee C)$ is in the environment, then we deduce $\neg B$ by proof $P4$ and $\neg C$ by proof $P5$ (proofs requested in exercise 59 on page 62). Let P be the proof of A in the environment where $\neg B$, $\neg C$ replace the formula $\neg(B \vee C)$. The proof of A is "$P4$, $P5$, P".
10. If $A = B \vee C$, then prove C under the assumption $\neg B$: let P be the obtained proof. "`Assume` $\neg B$, P, `Therefore` $\neg B \Rightarrow C$" is proof of the formula $\neg B \Rightarrow C$ which is equivalent to A. To obtain the proof of A, just add the proof $P1$, requested in exercise 59 on page 62, of A in the environment $\neg B \Rightarrow C$. The proof obtained for A is therefore "`Assume` $\neg B$, P, `Therefore` $\neg B \Rightarrow C$, $P1$".
11. If $\neg(B \wedge C)$ is in the environment, then we deduce $\neg B \vee \neg C$ by proof $P3$ requested in exercise 59 on page 62, and we reason by cases as follows:
   — prove A in the environment where $\neg B$ replaces $\neg(B \wedge C)$: let P be the obtained proof,
   — prove A in the environment where $\neg C$ replaces $\neg(B \wedge C)$: let Q be the obtained proof.
   The proof of A is "$P3$, `Assume` $\neg B$, P, `Therefore` $\neg B \Rightarrow A$, `Assume` $\neg C$, Q, `Therefore` $\neg C \Rightarrow A$, A".
12. If $\neg(B \Rightarrow C)$ is in the environment, then we deduce B by proof $P6$, $\neg C$ by proof $P7$ (proofs requested in exercise 59 on page 62). Let P be the proof of A in the environment where B, $\neg C$ replace the formula $\neg(B \Rightarrow C)$. The proof of A is "$P6$, $P7$, P".
13. If $B \Rightarrow C$ is in the environment and if $C \neq \bot$, i.e. if $B \Rightarrow C$ is not equal to $\neg B$, then we deduce $\neg B \vee C$ in the environment $B \Rightarrow C$ by proof $P2$ requested in exercise 59 on page 62 then we reason by cases:
   — prove A in the environment where $\neg B$ replaces $B \Rightarrow C$: Let P be the obtained proof,
   — prove A in the environment where C replaces $B \Rightarrow C$: Let Q be the obtained proof.
   The proof of A is "$P2$, `Assume` $\neg B$, P, `Therefore` $\neg B \Rightarrow A$, `Assume` C, Q, `Therefore` $C \Rightarrow A$, A".

Example 3.2.1 *We apply these tactics to the proof of Peirce*[1]*'s law:* $((p) \Rightarrow q) \Rightarrow p) \Rightarrow p$. *This formula is not provable in*

[1] Charles Sanders Peirce was an American logician and philosopher (1839 - 1914).

intuitionistic logic, which is, as we said in the beginning of the chapter, classical logic without the Reductio ad absurdum *rule. We can prove, using intuitionistic logic,that this formula is equivalent to the law of excluded middle.*

Given the form of the formula, we must apply tactic 5. The proof of Peirce's formula is therefore of the following form:

Proof Q:
`Assume` $(p \Rightarrow q) \Rightarrow p$
Q_1 *proof of* p *in environment* $(p \Rightarrow q) \Rightarrow p$
`Therefore` $(p \Rightarrow q) \Rightarrow p) \Rightarrow p$

The proof Q_1 *must use tactic 13. So this proof is written: In the environment* $B \Rightarrow C$ *where* $B = p \Rightarrow q$, $C = p$.

Proof Q_1*:*
$Q_{11} = P2$ *where* $P2$ *is the proof of* $\neg B \vee C$ *in the environment* $B \Rightarrow C$*, see exercise 59 on page 62*
`Assume` $\neg B$
Q_{12} *proof of* $A = p$ *in environment* $\neg B$
`Therefore` $\neg B \Rightarrow A$
`Assume` C
Q_{13} *proof of* $A = p$ *in environment* C
`Therefore` $C \Rightarrow A$
A

Q_{13} *is the empty proof, because* $A = C = p$.
Q_{12} *is the proof of* $A = p$ *in environment* $\neg(p \Rightarrow q)$*. This proof is the proof* $P6$ *requested in exercise 59 on page 62, where* $B = p$ *and* $C = q$*. Assembling the pieces* Q_1, Q_{11}, Q_{12}, Q_{13}*, we get the proof Q, that we don't copy here, because it can be automatically obtained by the software developed by Michel Levy, available at:* `http://teachinglogic.univ-grenoble-alpes.fr/DN/`

We show below how to find proof Q_{12} *without using tactics. The only rule that does not lead to a dead-end, is the reduction to absurdity. So this proof has shape:*

Proof Q_{12} *of* p *in environment* $\neg(p \Rightarrow q)$
`Assume` $\neg p$
Q_{121} *proof of* $\perp$ *in the environment* $\neg(p \Rightarrow q)$, $\neg p$
`Therefore` $\neg\neg p$
p

To get a contradiction (a proof of $\perp$*), we need to deduce* $p \Rightarrow q$*. So the proof* Q_{121} *is:*

`Assume` p
$\perp$
q
`Therefore` $p \Rightarrow q$
$\perp$

3.3 Consistency of natural deduction

The proof of consistency of natural deduction consists in proving that each proof built from a set of formulas Γ gives a formula that is deducible from Γ.

Theorem 3.3.1 (Consistency of deduction) *If a A formula is deduced from an environment* Γ *(*$\Gamma \vdash A$*), then A is a consequence of* Γ *(*$\Gamma \models A$*).*

Proof : Let Γ be a set of formulas. Let P be a proof of A in this environment. Let C_i be the conclusion and H_i the context of the i^{th} line of proof P. We take $H_0 = \emptyset$, the empty list. If the proof P is empty, we take $C_0 = A$. Let us note Γ, H_i the set of all formulas in Γ and in the list H_i. We prove by induction that for any k we have Γ, $H_k \models C_k$, which implies that for the last line n of this proof, as by definition H_n is the empty list and $C_n = A$, we get $\Gamma \models A$.

Base case: Assume A is deduced from Γ through the empty proof. Then A is an element of Γ, so $\Gamma \models A$. Since $H_0 = \emptyset$, we can conclude: $\Gamma, H_0 \models A$, so $\Gamma, H_0 \models C_0$.

Induction: Assume that for any line $i < k$ of the proof we have $\Gamma,\ H_i \models C_i$. Let us show that $\Gamma,\ H_k \models C_k$. Assume the line k is:

1. "`Assume` C_k". The formula C_k is the last formula of H_k, so $\Gamma,\ H_k \models C_k$.
2. "`Therefore` C_k". The formula C_k is equal to the formula $B \Rightarrow D$, and B is the last formula of H_{k-1}. We distinguish two cases, either D is an element of Γ, or D is usable on the previous line.
   (a) In the first case, since D is equal to a formula of Γ, D is a consequence of Γ, thus a consequence of $\Gamma,\ H_k$. Since $B \Rightarrow D$ is a consequence of D, as a result $\Gamma,\ H_k \models C_k$.
   (b) In the second case, D is usable on the previous line. So there is an $i < k$ such that $D = C_i$ and H_i is a prefix of H_{k-1}. By *induction hypothesis*, $\Gamma,\ H_i \models D$. Since H_i is a prefix of H_{k-1}, we have $\Gamma,\ H_{k-1} \models D$. Since B is the last formula of H_{k-1}, we have $H_{k-1} = H_k,\ B$ and so $\Gamma,\ H_k,\ B \models D$, which implies $\Gamma,\ H_k \models B \Rightarrow D$. Finally, since C_k is equal to $B \Rightarrow D$ and two equal formulas are equivalent, we have $\Gamma,\ H_k \models C_k$.
3. "C_k". This formula is the conclusion of a rule in Table 3.1 on page 53, applied to its premises usable on the previous line or to elements of Γ. Let us consider only the case of rule $\wedge I$, the other cases being similar. The formula C_k is equal to $(D \wedge E)$ and the premises of the rule are D and E. Since D and E are elements of Γ or usable on the previous line, as in the previous case, using the induction hypothesis, we have: $\Gamma,\ H_{k-1} \models D$ and $\Gamma,\ H_{k-1} \models E$. Since the line k does not change the hypotheses, we have $H_{k-1} = H_k$, so $\Gamma,\ H_k \models D$ and $\Gamma,\ H_k \models E$. Since C_k is equal to $(D \wedge E)$, we have: $D,\ E \models C_k$. As a result $\Gamma,\ H_k \models C_k$.

□

3.4 Completeness of natural deduction

We prove the completeness of the rules only for formulas with the logical symbols $\bot$, $\wedge$, $\vee$, $\Rightarrow$. The completeness for formulas obtained by adding logical symbols $\top$, $\neg$ and $\Leftrightarrow$ results immediately from the fact that these symbols can be considered as abbreviations.

Given the absence of negation, we define a literal as a variable or an implication between a variable and $\bot$. Let x be a variable; x and $x \Rightarrow \bot$ (which can be shortened to $\neg x$) are complementary literals. In the following we distinguish a list of formulas Γ and $s(\Gamma)$ the set of formulas in the list. To simplify the notations, we use the comma to add an element at the beginning or at the end of the list as well as to concatenate two lists, whether these lists are lists of formulas or proofs.

Theorem 3.4.1 *Let Γ be a finite set of formulas and A a formula, if $\Gamma \models A$ then $\Gamma \vdash A$.*

Remark 3.4.2 *We admit this theorem in this handout. One can provide a constructive proof, that is, it gives a complete set of tactics to build proofs of a formula in an environment. However, these tactics can give long proofs. In particular if we have to prove a formula $(B \vee C)$, it is generally better to follow the tactics given in paragraph 3.2 on page 57, i.e. try to prove B, then try to prove C and* only in case of a failure *use the tactic given in the proof of completeness, which "reduces" this proof to a proof of C by adding the hypothesis $\neg B$.*

3.5 Tools

We indicate two softwares to become more familiar with natural deduction. The first automatically builds proofs like the ones we presented, the second illustrates interactively on some examples how to draw tree-like proofs.

3.5.1 Automatic Proof Building Software

To automatically produce proofs, we recommend use the software:

`http://teachinglogic.univ-grenoble-alpes.fr/DN/`

Thanks to an intuitive syntax, this software allows to type a formula and automatically generate:

— if the formula is (syntactically) incorrect, an error message (in red) is produced above the formula

— if the formula is provable, its proof (without annotations) is generated by the software.
— if the formula is correct but not provable, a countermodel is proposed above the formula.

If the formula is provable, it is also possible to obtain an annotated version of the proof.

3.5.2 Drawing tree-like proofs

Those who love tree-like proofs may use the following software developed by Laurent Théry:

`http://www-sop.inria.fr/marelle/Laurent.Thery/peanoware/Nd.html`

or its Android version:

`https://play.google.com/store/apps/details?id=org.inria.peanoware`

This software is presented as a game, it offers formulas to be proved and allows to apply the rules of natural deduction, interactively, to build tree-like proofs. When you succeed in building the proof of a formula, you see the photo of Dag Prawitz, one of the main logicians who studied the properties of natural deduction.

3.6 Exercises

Exercise 52 (Proof Sketch) *The following sequence of lines is not a proof sketch. Find the smallest number i such that lines 1 to $i-1$ is a proof draft, but lines 1 to i is not. Give the context of lines 1 to $i-1$.*

context	*number*	*proof*
	1	`Suppose` a
	2	`Suppose` b
	3	c
	4	`Thus` d
	5	`Suppose` e
	6	f
	7	`Thus` g
	8	h
	9	i
	10	`Thus` j
	11	`Thus` k
	12	l

□

Exercise 53 (Proof of Formulas Requiring Special Rules) *Give a proof of the following formulas:*

— $a \Rightarrow \neg\neg a$,
— $\neg\neg a \Rightarrow a$,
— $a \Leftrightarrow \neg\neg a$,
— $a \vee \neg a$.

□

Exercise 54 (Simple Proofs of Formulas) *Give a proof of the following formulas:*

— $a \Rightarrow c$ *in environment* $a \Rightarrow b$, $b \Rightarrow c$.
— $(a \Rightarrow b) \wedge (b \Rightarrow c) \Rightarrow (a \Rightarrow c)$.
— $(a \Rightarrow b) \Rightarrow ((b \Rightarrow c) \Rightarrow (a \Rightarrow c))$.

□

Exercise 55 (↷Abbreviations) *Let A be a formula and $ufld(A)$ be the formula obtained by replacing in A the negations and equivalences (in abbreviated form) by their definition. $ufld(A)$ is the formula obtained by "unfolding", hence the name $ufld$ chosen for the unfolding function.*

— *Define the function $ufld$ by recurrence.*
— *Show that the formulas A and $ufld(A)$ are equivalent.*
— *Deduce that two formulas, equal up to abbreviations, are equivalent.*

□

Exercise 56 (Proof of Formulas) *Give a proof of the following formulas:*

1. $a \Rightarrow (b \Rightarrow a)$.
2. $a \wedge b \Rightarrow a$.
3. $\neg a \Rightarrow (a \Rightarrow b)$.
4. $(a \Rightarrow (b \Rightarrow c)) \Rightarrow ((a \Rightarrow b) \Rightarrow (a \Rightarrow c))$.
5. $a \wedge b \Rightarrow b \wedge a$.
6. $a \vee b \Rightarrow b \vee a$.
7. $(a \Rightarrow (b \Rightarrow c)) \Rightarrow (a \wedge b \Rightarrow c)$.
8. $(a \wedge b \Rightarrow c) \Rightarrow (a \Rightarrow (b \Rightarrow c))$.
9. $(a \Rightarrow b) \wedge (c \Rightarrow d) \Rightarrow (a \wedge c \Rightarrow b \wedge d)$.

□

Exercise 57 (About implication) *Give a proof of the following formulas:*

1. *(**)* $a \Rightarrow b$ *in the environment* $\neg a \vee b$.
2. $(\neg b \Rightarrow \neg a) \Rightarrow (a \Rightarrow b)$.

□

Exercise 58 (Boolean algebra) *Give a proof of the following formulas:*

1. $\neg(a \wedge \neg a)$.
2. $a \vee a \Rightarrow a$.
3. $a \wedge a \Rightarrow a$.
4. $a \wedge (b \vee c) \Rightarrow a \wedge b \vee a \wedge c$.
5. $a \wedge b \vee a \wedge c \Rightarrow a \wedge (b \vee c)$.
6. $a \vee b \wedge c \Rightarrow (a \vee b) \wedge (a \vee c)$.
7. $(a \vee b) \wedge (a \vee c) \Rightarrow a \vee b \wedge c$.

□

Exercise 59 (Proof of Formulas with Environment) *Give a proof of the following formulas:*

— *P1 :* $B \vee C$ *in environment* $\neg B \Rightarrow C$.
— *P2 :* $\neg B \vee C$ *in environment* $B \Rightarrow C$.
— *P3 :* $\neg B \vee \neg C$ *in environment* $\neg(B \wedge C)$.
— *P4 :* $\neg B$ *in environment* $\neg(B \vee C)$.
— *P5 :* $\neg C$ *in the same environment* $\neg(B \vee C)$.
— *P6 :* B *in the same environment* $\neg(B \Rightarrow C)$.
— *P7 :* $\neg C$ *in the same environment* $\neg(B \Rightarrow C)$.

□

Exercise 60 (Proof of Formulas) *Give a natural deduction proof of the following formulas:*

1. $\neg(a \vee b) \Rightarrow (\neg a \wedge \neg b)$.
2. $(\neg a \wedge \neg b) \Rightarrow \neg(a \vee b)$.
3. *(*)* $(\neg a \vee \neg b) \Rightarrow \neg(a \wedge b)$.
4. *(**)* $(a \vee b) \wedge (\neg a \vee b) \Rightarrow b$.
5. *(***)* $\neg(a \wedge b) \Rightarrow (\neg a \vee \neg b)$.
6. *(***)* $(a \vee b) \wedge (\neg b \vee c) \Rightarrow a \vee c$.

□

Exercise 61 (Midterm 2011) *Prove the following formulae using natural deduction :*

— $(p \Rightarrow \bot) \wedge (\neg p \wedge q \Rightarrow \bot) \Rightarrow \neg q$
— $((p \vee q) \vee (p \vee r)) \wedge \neg p \Rightarrow q \vee r$
— *(*)* $(p \Rightarrow \bot) \vee (p \wedge q \Rightarrow \bot) \Rightarrow \neg q \vee \neg p$

□

Exercise 62 (Midterm 2013) *Give a proof of the following formulae using the natural deduction in the table format :*

— $(p \vee q) \Rightarrow (\neg p \wedge \neg q) \Rightarrow r$.
— $((p \Rightarrow q) \wedge (q \Rightarrow r)) \wedge \neg r \Rightarrow \neg p$.
— $(p \Rightarrow q) \Rightarrow ((p \wedge q) \vee \neg p)$.

□

Exercise 63 (Questions from various midterm exams)

1. *Prove the following formula using natural deduction:* $(p \vee q) \wedge (p \Rightarrow r) \Rightarrow q \vee r$

2. *We consider the hypotheses:*
   - *(H1) : If Peter is tall, then John isn't Peter's son.*
   - *(H2) : If Peter isn't tall, then John is Peter's son.*
   - *(H3) : If John is Peter's son, then Mary is John's sister.*

   Formalize and prove that from these we can deduce conclusion (C) : "Mary is John's sister or Peter is tall or both" using natural deduction.

□

Part II

First-order logic

Chapter 4

First-order logic

Contents

THIS famous syllogism [1] cannot be formalized in propositional logic:

All men are mortal;
Socrates is a man;
Hence Socrates is mortal.

To formalize such reasoning we need to enrich the propositional logic with new connectives called *quantifiers*, this extended logic is called first-order logic. It makes it possible to talk about structures with a single non-empty domain (unlike propositional logic, this domain can have more than two values), functions and relations over that domain. The language of this logic consists of several categories: terms that represent the elements of the domain or functions over these elements, relations that connect terms together and formulas that describe the interactions between the relations thanks to connectives and quantifiers. For example, the relation $mortal(x)$ means that x is mortal, the relation $man(x)$ means that x is a man. In addition, $Socrates$ is a constant of the domain (i.e., its value is an element of the domain) and $man(Socrates)$ means that Socrates is a man. Finally, the formula $\forall x\ (man(x) \Rightarrow mortal(x))$ indicates that all men are mortal. From these two hypotheses, it is possible to conclude that Socrates is mortal ($mortal(Socrates)$) by various methods we will detail in this second part. This same reasoning also applies to prove the following syllogism:

[1] The notion of syllogism was introduced by Aristotle and means literally "word (which goes) with (another)", also known in Latin by *modus ponendo ponens* = "how to assert, to establish by affirming" and more briefly *modus ponens*.

Cheap horses are rare.
Everything that is rare is expensive.
Hence a cheap horse is expensive.

Despite the conclusion which seems contradictory, the reasoning is correct. In order to obtain a contradiction one must add the hypothesis: $\forall x($ cheap $(x) \Leftrightarrow \neg$ expensive (x). This reflects the commonly accepted relation between the notions of cheap and expensive, i.e. everything that is expensive is not cheap and vice versa. With this additional hypothesis we can show that this reasoning is contradictory, without this hypothesis the reasoning is correct. This elucidates the paradox of this syllogism.

Our aim in this chapter is to introduce the concepts and basic notions of first-order logic in order to be able to propose methods of reasoning in the following chapters.

Overview: We begin by describing the syntax of first-order logic. Then we define the notions of *free* and *bound* which are essential for interpreting the meaning of formulas. Then we define the meaning of the formulas that we can build from the introduced syntax. Finally, we state remarkable properties of first-order logic, which can be used in the reasonings.

4.1 Syntax

We add two symbols to propositional logic: the *existential* symbol ($\exists$) and the *universal* symbol ($\forall$). The existential symbol means that there is an element having a certain property, whereas the universal symbol allows to talk about all elements having a property. These changes imply that an element of this language can now depend on several variables. It is therefore necessary to add a variable delimiter to the syntax. Classically, we chose the comma. These changes introduce new symbols in addition to variables, these symbols can for example be numbers on which we can create functions (such as the addition) or define relations (such as equality). All these changes make the presentation of the syntax slightly more subtle than propositional logic.

4.1.1 Strict formulas

To write the formulas of first-order logic we extend the syntax of the propositional logic, so we have the following vocabulary:

- Two propositional constants: $\bot$ and $\top$ representing respectively the false and the true.
- Variables: sequence of letters and digits starting with a lowercase letter among u, v, w, x, y, z.
- Connectives: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$.
- Quantifiers: $\forall$ the universal quantifier and $\exists$ the existential quantifier.
- Punctuations: the comma "," and the opening and closing brackets "(" and ")".
- Symbol:
  - ordinary: sequence of letters and digits not beginning with a lowercase letter among u, v, w, x, y, z.
  - special: $+, -, *, /, =, \neq, <, \leq, >, \geq$

The set of special symbols can be enriched with new elements, depending on the mathematical fields we study, provided we respect the constraint that these new elements are not in the sets of propositional constants, variables, connectives, quantifiers, punctuations or ordinary symbols.

Example 4.1.1 *Below, we illustrate these notions:*

- $x, x1, x2, y$ *are variables,*
- *man, parent, succ, 12, 24,* $f1$ *are ordinary symbols; ordinary symbols will represent functions (numerical constants or functions with multiple arguments) or relations (propositional variables or relations with multiple arguments).*
- $x = y$, $z > 3$ *are examples of an application of special symbols.*

We broadly define the notion of a term which is essential in first-order logic. We will extend later this concept by defining what is a term associated with a set of symbols.

Definition 4.1.2 (Term) *A* term *is inductively defined by:*

- *an ordinary symbol is a term,*
- *a variable is a term,*
- *if* $t_1, \ldots, t_n$ *are terms and if* s *is a symbol (ordinary or special) then* $s(t_1, \ldots, t_n)$ *is a term.*

Example 4.1.3

are terms, however

is not a term. Note that $42(1,y,3)$ *is also a term, but it is customary that the names of functions and relations are ordinary symbols starting with letters.*

Definition 4.1.4 (Atomic formula) *We define an atomic formula inductively by:*

— $\top$ *and* $\bot$ *are atomic formulas*

— *an ordinary symbol is an atomic formula*

— *if* $t_1,\ldots,t_n$ *are terms and if* s *is a symbol (ordinary or special) then* $s(t_1,\ldots,t_n)$ *is an atomic formula.*

Example 4.1.5

are atomic formulas, also notice that

are not atomic formulas.

Notice that the set of terms and the set of atomic formulas are not disjoint. For example, $p(x)$ is both a term and an atomic formula. When t is both a term and a atomic formula, we distinguish $[\![t]\!]$ the meaning of t seen as a term, from $[t]$ the meaning of t seen as a formula (see paragraph 4.3.4 on page 74).

Definition 4.1.6 (Formula) *We define a* formula *inductively by:*

— *an atomic formula is a formula,*

— *if* A *is a formula then* $\neg A$ *is a formula,*

— *if* A *and* B *are formulas and if* $\circ$ *is one of the connectives* $\vee,\wedge,\Rightarrow,\Leftrightarrow$ *then* $(A\circ B)$ *is a formula,*

— *if* A *is a formula and if* x *is* any *variable then* $\forall x\ A$ *and* $\exists x\ A$ *are formulas*[2].

Example 4.1.7

are atomic formulas, so they are formulas. However

is a formula that is not atomic.

The notion of *subformula* (definition 1.1.4) can naturally be extended to first-order logic. The notion of *strict formula size* requires a new definition:

Definition 4.1.8 (Size of a formula) *The* size of a formula A*, noted* $|A|$*, is defined inductively by:*

— $|\top|=0$ *and* $|\bot|=0$.

— *If* A *is an atomic formula then* $|A|=0$.

— $|\neg A|=1+|A|$.

— $|Q x\ A|=1+|A|$ *if* Q *is one of the quantifiers* $\forall$ *or* $\exists$.

— $|(A\circ B)|=|A|+|B|+1$.

[2] Warning, x may not be present in A.

4.1.2 Prioritized formulas

We use the precedences of propositional logic for connectives, and we add a precedence for the quantifiers identical to that of the negation. In table 4.1 we give the precedences for symbols and connectives in decreasing order from top to bottom. We only specify two rules to build prioritized formulas, which distinguish them from completely parenthesized formulas and allow to omit parentheses or add new parentheses.

To shorten the writing of terms, some function symbols $+,-,*,/$ and some relation symbols $=,\neq,<,\leq,>,\geq$ can be written in an infix manner, that is, the usual way.

Example 4.1.9 *We* abbreviate *the term* $+(x,*(y,z))$ *into* $x+y*z$ *and* $\leq(*(3,x),+(y,5))$ *into* $3*x\leq y+5$. *The reverse transformation is defined by giving the symbols* $=,\neq,<,\leq,>,\geq$ *lower precedence than the symbols* $+,-,*,/$.

OPRATIONS	
$-+$ unary	
$*,/$	left associative
$+,-$ binary	left associative
RELATIONS	
$=,\neq,<,\leq,>,\geq$	
NEGATION, QUANTIFIERS	
$\neg,\forall,\exists$	
BINARY CONNECTIVES	
$\wedge$	left associative
$\vee$	left associative
$\Rightarrow$	**right** associative
$\Leftrightarrow$	left associative

Table 4.1: Precedence of connectives and symbols.

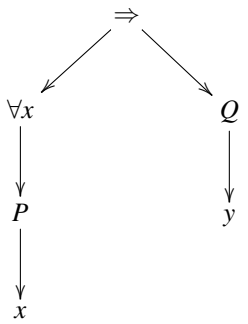
Definition 4.1.10 (Prioritized formula) *A* prioritized formula *is:*

— *An atomic formula.*
— *If A is a prioritized formula then* $\neg A$ *is a prioritized formula.*
— *If A and B are prioritized formulas then* $A\circ B$ *is a prioritized formula.*
— *If A is a prioritized formula and if x* any *variable then* $\forall x\, A$ *and* $\exists x\, A$ *are prioritized formulas.*
— *If A is a prioritized formula then* (A) *is a prioritized formula.*

Example 4.1.11 *The (prioritized) formula* $\forall xP(x)\wedge\forall xQ(x)\Leftrightarrow\forall x(P(x)\wedge Q(x))$ *can be seen as an* abbreviation *of the formula* $((\forall xP(x)\wedge\forall xQ(x))\Leftrightarrow\forall x(P(x)\wedge Q(x)))$.

The (prioritized) formula $\forall x\forall y\forall z(x\leq y\wedge y\leq z\Rightarrow x\leq z)$ *can be seen as an* abbreviation *of the formula* $\forall x\forall y\forall z((\leq(x,y)\wedge\leq(y,z))\Rightarrow\leq(x,z)$.

Example 4.1.12 *The precedence of* $\forall$ *is higher than* $\Rightarrow$, *in formula* $\forall xP(x)\Rightarrow Q(y)$. *Thus, the left operand of the immlication is* $\forall xP(x)$. *The structure of the formula will then be represented by the following tree:*



As for propositional logic, the size of a prioritized formula will be equal to the size of the strict formula it abbreviates. Similarly, for subformulas, we always consider the strict formula abbreviated by the prioritized formula.

4.2 To be free or bound

The meaning of the formula $x+2=4$ depends on x: the formula is true (in arithmetic) only if $x=2$. The variable x is free in that formula.

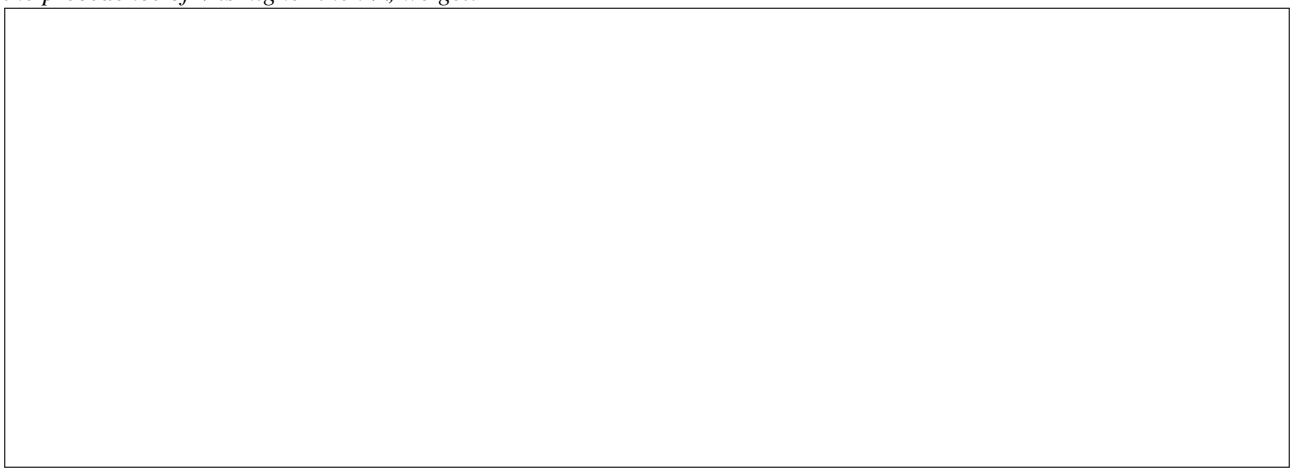
On the opposite, still in arithmetic, $\forall x(x+2=4)$ is a false formula and $\exists x(x+2=4)$ is a true formula because $2+2=4$. For each of these two formulas, there is no need to choose a value for x to determine their respective value: these two formulas have no free variables.

4.2.1 Free and bound occurrences

On a tree representation, where we draw the structures of formulas representing $\forall x$ and $\exists x$ as tree nodes, a *bound occurrence* of the variable x is an occurrence below a $\exists x$ node or a $\forall x$ node. An occurrence of x that is not below such a node is *free*.

Definition 4.2.1 (Binding scope, free occurrence, bound occurrence) *Let x be a variable and A a formula. In a formula $\forall x\, A$ or $\exists x\, A$, the* binding scope *for x is A. An occurrence of x in a formula is* free *if it is not within the scope of a binding for x, otherwise it is said to be* bound.

Example 4.2.2 *To see the occurrences of the variables, we draw the structure of the formula $\forall x P(x,y) \wedge \exists z R(x,z)$, since the precedence of $\forall$ is higher then $\wedge$, we get:*



The occurrence of x in $P(x,y)$ is bound, the occurrence of x in $R(x,z)$ is free. The occurrence of z is bound.

4.2.2 Free and bound variables

Definition 4.2.3 (Free variable, bound variable, closed formula) *The variable x is a* free variable *of a formula if and only if there is a free occurrence of x in the formula. A variable x is a* bound variable *of a formula if and only if there is a bound occurrence of x in the formula. A formula without free variables is also called a* closed formula.

Remark 4.2.4 *A variable may be both free and bound. For example, in the formula $\forall x P(x) \vee Q(x)$, x is both free and bound.*

Remark 4.2.5 *By definition, a variable that does not appear in a formula (0 occurrence) is a NON-free variable of the formula.*

Example 4.2.6 *The free variables of the formula in the example 4.2.2 are x and y.*

4.3 Meaning of the formulas

In this section, we define the meaning of connectives, quantifiers and *equality* in first-order logic. In order to define the meaning of a formula, it is sufficient to give the value of free variables and the meaning of its symbols. First we explain how we declare a symbol. Then we introduce the notion of signature which allows us to define the associated terms and formulas. Once all these notions are stated we define interpretations in order to give a meaning to the formulas of first-order logic.

4.3.1 Declaring a symbol

In order to give a meaning to the formulas of first-order logic we must declare the symbols we use. In addition to variables, we distinguish function symbols, e.g. $g(x,y)$, and relation symbols like $parent(x,y)$.

Definition 4.3.1 (Symbol declaration) *A* symbol declaration *is a triplet noted s^{gn} where s is a symbol, g is either f (for function) or r (for relation), and n is a natural integer denoting the number of arguments of this symbol, n is also called the* arity *of s.*

Example 4.3.2 *The functions with arity* 0 *are constants, for example* 1 *and* 2 *are constants with a zero arity, but $parent^{r2}$ is a notation meaning that parent is used as a relation with arity* 2*. Similarly $*^{f2}$ is a declaration stating that $*$ is a* 2*-ary function. The A^{r0} symbol has arity* 0 *and means that A is a propositional variable. The symbol man^{r1} is* 1*-ary or* unzry *and the symbol g^{f2} has arity* 2 *or is* 2*-ary (binary).*

Remark 4.3.3 *When the context or the usual conventions convey an implicit declaration of a symbol, we omit the exponent. For example, the equal symbol is always used as a relation with two arguments, so we shorten the symbol declaration $=^{r2}$ into $=$.*

4.3.2 Signature

In first-order logic we can choose the name of the variables we use but we also have the ability to build our own constants (functions without arguments), functions, propositional variables (zero arity relations) and relations. A signature is therefore the set of declarations for the symbol we accept when building formulas. It allows to define the symbols whose meaning is not fixed *a priori*, unlike for example the constants $\top$ and $\bot$ whose meaning is always fixed to 1 and 0 respectively.

Definition 4.3.4 (Signature, constant, function symbol, propositional variable and relation) *A* signature Σ *is a set of symbol declarations of the form s^{gn}. Let n be a strictly positive integer and Σ a signature, the symbol s is:*

1. *a* constant *of the signature if and only if $s^{f0} \in \Sigma$*
2. *a* function symbol with *n* arguments *of the signature, if and only if $s^{fn} \in \Sigma$*
3. *a* propositional variable *of the signature if and only if $s^{r0} \in \Sigma$*
4. *a* relation symbol with *n* arguments *of the signature if and only if $s^{rn} \in \Sigma$*

Instead of saying that $0^{f0}, 1^{f0}, +^{f2}, *^{f2}, =^{r2}$ is a signature for arithmetic, we say more simply that a possible signature for arithmetic includes 0, 1, $+$ (with two arguments), $*$ and $=$. In this example, we must point out that the symbol *plus* is used with two arguments (because we can also find the symbol *plus* with one argument).

Example 4.3.5 *A possible signature for set theory is $\in, =$. Notice that all other operations on sets can be defined using these two symbols.*

We now define the terms associated with signature. We consider a countable set of variables disjoint from the set of signature symbols.

Definition 4.3.8 (Term over a signature) *Let Σ be a signature, a* term *over Σ is either:*

— *a variable,*
— *a constant s where $s^{f0} \in \Sigma$,*
— *a term of the form $s(t_1,\ldots,t_n)$, where $n \geq 1$, $s^{fn} \in \Sigma$ and $t_1,\ldots,t_n$ are terms over Σ.*

The set of terms over signature Σ is noted T_Σ.

Definition 4.3.9 (Atomic formula over a signature) *Let Σ be a signature, an* atomic formula *over Σ is either:*
- *one of the constants $\top, \bot$,*
- *a propositional variable s where $s^{r0} \in \Sigma$,*
- *of the form $s(t_1, \ldots, t_n)$ where $n \geq 1$, $s^{rn} \in \Sigma$ and where $t_1, \ldots, t_n$ are terms on Σ.*

Note that variables are not atomic formulas.

Definition 4.3.10 (Formula over a signature) *A* formula *over a signature Σ is a formula whose atomic subformulas are atomic formulas over Σ (according to definition 4.3.9).*

The set of formulas over signature Σ is noted F_Σ .

Example 4.3.11 *$\forall x\ (p(x) \Rightarrow \exists y\ q(x,y))$ is a formula over the signature $\Sigma = \{p^{r1}, q^{r2}, h^{f1}, c^{f0}\}$. But it iss also a formula over the signature $\Sigma' = \{p^{r1}, q^{r2}\}$, since the symbols h and c are not included in the formula.*

Definition 4.3.12 (Signature associated with a formula) *The* signature associated with a formula *is the smallest signature Σ such that the formula is an element of F_Σ, it is the smallest signature that allows to write the formula.*

Example 4.3.13 *The signature Σ associated with the formula $\forall x\ (p(x) \Rightarrow \exists y\ q(x,y)$ is $\Sigma = \{p^{r1}, q^{r2}\}$.*

Definition 4.3.14 (Signature associated with a set of formulas) *The* signature associated with a set of formulas *is the union of the signatures associated with each formula of the set.*

Example 4.3.15 *The signature Σ associated with the set consisting of the two formulas $\forall x(p(x) \Rightarrow \exists y\ q(x,y)), \forall u\ \forall v\ (u + s(v) = s(u) + v)$ is $\Sigma = \{p^{r1}, q^{r2}, +^{f2}, s^{f1}, =^{r2}\}$.*

4.3.3 Interpretation

In propositional logic, the meaning of formulas is only fixed by the values of variables, in first-order logic the meaning of the formulas also depends on the meaning of the functions and the relations. The meaning of functions and relations is fixed by an interpretation.

Definition 4.3.16 (Interpretation) *An* interpretation *I over a signature Σ is defined by a non-empty domain D and an application that maps each symbol declaration $s^{gn} \in \Sigma$ to its value s_I^{gn} as follows:*

1. *s_I^{f0} is an element of D.*
2. *s_I^{fn} where $n \geq 1$ is a function from D^n to D, in other words, a function with n arguments.*
3. *s_I^{r0} is either 0 or 1.*
4. *s_I^{rn} where $n \geq 1$ is a subset of D^n, in other words a relation with n arguments.*

Example 4.3.17 *Let I be the interpretation with domain $D = \{1,2,3\}$ where the binary relation* friend *is true for the couples $(1,2)$, $(1,3)$ and $(2,3)$, i.e. $friend_I^{r2} = \{(1,2),(1,3),(2,3)\}$. $friend(2,3)$ is true in the interpretation I. On the other hand, $friend(2,1)$ is false in the interpretation I.*

Remark 4.3.18 *In* any *interpretation I, the value of the symbol $=$ is the set $\{(d,d) \mid d \in D\}$, in other words, in any interpretation the meaning of equality is* the identity *over the domain of the interpretation.*

Example 4.3.19 *Consider the following signature.*
- *$Anne^{f0}$, $Bernard^{f0}$ and $Claude^{f0}$: the first names Anne, Bernard, and Claude denote constants,*
- *ℓ^{r2} : the letter ℓ denotes a relation with two arguments (we read $\ell(x,y)$ as x loves y) and*
- *s^{f1}: the letter s denotes a function with one argument (we read $s(x)$ as the spouse of x).*

A possible interpretation over this signature is the interpretation I with domain $D = \{0,1,2\}$ where:
- *$Anne_I^{f0} = 0$, $Bernard_I^{f0} = 1$, and $Claude_I^{f0} = 2$.*
- *$\ell_I^{r2} = \{(0,1),(1,0),(2,0)\}$.*

— $s_I^{f1}(0) = 1, s_I^{f1}(1) = 0, s_I^{f1}(2) = 2$. *Notice that the function* s_I^{f1} *has domain D, which requires us to define artificially* $s_I^{f1}(2)$*: Claude, denoted by* 2*, has no spouse.*

Definition 4.3.20 (Interpretation of a set of formulas) The interpretation of a set of formulas *is an interpretation that only defines the meaning of the signature associated with the set of formulas.*

Definition 4.3.21 (State) *A* state *e of an interpretation is an application from the set of variables to the domain of the interpretation.*

Definition 4.3.22 (Assignment) *An* assignment *is a couple* (I,e) *composed of an interpretation I and a state e.*

Example 4.3.23 *Let* $D = \{1,2,3\}$ *be a domain and I be the interpretation where the binary relation* friend *is true only for couples* $(1,2)$*,* $(1,3)$ *and* $(2,3)$*, i.e.,* $friend_I^2 = \{(1,2),(1,3),(2,3)\}$*. Let e be the state that maps x to* 2 *and y to* 1*. The assignment* (I,e) *makes the relation* $friend(x,y)$ *false.*

4.3.4 Meaning of the formulas

We now explain how to evaluate a formula from an assignment, that is, an interpretation and a state. We need to notice that in some cases the state of the assignment is unnecessary to give the meaning of a formula.

Remark 4.3.24 *The value of a formula depends only on its free variables and its symbols, thus to evaluate a formula without free variables, the state of the variables is useless. We have then two possibilities:*

— *For a formula* without free variables*, it is sufficient to give an interpretation I of the symbols in the formula. In this case, the assignments* (I,e) *and* (I,e') *will yield the same value for the formula for any states e and* e'*. So for any state e, we will identify* (I,e) *and I. Depending on the context, I will be regarded as either an interpretation or an assignment whose state is irrelevant.*

— *For a formula* with free variables*, we need an assignment.*

Let Σ be a signature, I an interpretation over Σ with domain D and e a state of this interpretation. We want to know the value (0 or 1) of any A formula over Σ in the assignment (I,e). This value will be noted $[A]_{(I,e)}$. In order to do this, we first need to define the meaning of each term t over Σ in assignment (I,e), noted $[\![t]\!]_{(I,e)}$. Then we have to fix the meaning of each atomic formula B over Σ in the same assignment, noted $[B]_{(I,e)}$.

4.3.4.1 Meaning of the terms over a signature

Everyone intuitively knows how to evaluate a term: we replace the variables with their values, the function symbols with the associated functions and we apply the functions. But to reason about the meaning of the terms or to write a program for evaluating the terms, we must formalize this evaluation.

Definition 4.3.25 (Evaluation) *We give the inductive definition of the evaluation of a term t:*

1. *if t is a variable, then* $[\![t]\!]_{(I,e)} = e(t)$*,*
2. *if t is a constant then* $[\![t]\!]_{(I,e)} = t_I^{f0}$*,*
3. *if* $t = s(t_1,\ldots,t_n)$ *where s is a symbol and* $t_1,\ldots,t_n$ *are terms, then* $[\![t]\!]_{(I,e)} = s_I^{fn}([\![t_1]\!]_{(I,e)},\ldots,[\![t_n]\!]_{(I,e)})$*.*

In the examples, we replace the variables with their values, we assimilate symbols with their meaning.

Example 4.3.26 *Let the signature be* a^{f0}, f^{f2}, g^{f2}*.*
Let I be the interpretation with domain $\mathbb{N}$ *where:*

— $a_I = 1$*;*

— f_I^{f2} *is the product;*

— g_I^{f2} *is the sum.*

Let e be the state such that $x = 2, y = 3$*. Let us compute* $[\![f(x,g(y,a))]\!]_{(I,e)}$*.*

4.3.4.2 Meaning of the atomic formulas over a signature

Definition 4.3.27 (Meaning of atomic formulas) *The meaning of atomic formulas is given by the following inductive rules:*

1. $[\top]_{(I,e)} = 1, [\bot]_{(I,e)} = 0$. *In the examples, we may replacer* $\top$ *with its value* 1 *and* $\bot$ *with its value* 0.
2. *Let s be a propositional variable,* $[s]_{(I,e)} = s_I^{r0}$.
3. *Let* $A = s(t_1,\ldots,t_n)$ *where s is a symbol and* $t_1,\ldots,t_n$ *are terms. If* $([\![t_1]\!]_{(I,e)},\ldots,[\![t_n]\!]_{(I,e)}) \in s_I^{rn}$ *then* $[A]_{(I,e)} = 1$ *otherwise* $[A]_{(I,e)} = 0$.

Remark 4.3.28 *We must distinguish between evaluating a term* $[\![\ ]\!]$ *and evaluating an atomic formula* $[\ ]$*, because in our presentation, these two syntactic categories are not dijoint. This is because* $s(t_1,\ldots,t_n)$ *may be the application of a function (*$[\![\ ]\!]$*) or a relation (*$[\ ]$*). But when the context is unambiguous, we can omit this distinction.*

Example 4.3.29 *Using again example 4.3.19 on page 73, we get:*

— $[\ell(Anne, Bernard)]_I =$

— $[\ell(Anne, Claude)]_I =$

Let e be the state $x = 0, y = 2$. *We have:*

— $[\ell(x, s(x))]_{(I,e)} =$

— $[\ell(y, s(y))]_{(I,e)} =$

Be careful to distinguish (depending on the context), the elements of the domain 0, 1 *and the truth values* $0, 1$. *The following examples show the interpretation of equality as an identity. In the interpretation I, we have:*

— $[(Anne = Bernard)]_I =$

— $[(s(Anne) = Anne)]_I =$

— $[(s(s(Anne)) = Anne)]_I =$

4.3.4.3 Meaning of formulas over a signature

We can now add the meaning of quantifiers and also that of the connectives which remains identical to the one given for propositional logic (see subsection 1.2.1 on page 14).

Definition 4.3.30 (Meaning of formulas) *The meaning of the formulas is given by:*

1. *Propositional connectives have the same meaning as in propositional logical. Let B and C be formulas, let us only recall the meaning of implication: if* $[B]_{(I,e)} = 0$ *then* $[(B \Rightarrow C)]_{(I,e)} = 1$ *otherwise* $[(B \Rightarrow C)]_{(I,e)} = [C]_{(I,e)}$.

2. *Let x be a variable and B a formula.* $[\forall xB]_{(I,e)} = 1$ *if and only if* $[B]_{(I,f)} = 1$ for every *state f identical to e, except for x. Let $d \in D$. Let us note $e[x=d]$ the state identical to e, except for the variable x, which is mapped to the value d by the state $e[x=d]$. The above definition can be rephrased as follows:*

$$[\forall xB]_{(I,e)} = min_{d\in D}[B]_{(I,e[x=d)]} = \prod_{d\in D}[B]_{(I,e[x=d])},$$

where the product is the Boolean product.
This definition allows to compute the value of $\forall xB$ in the case where D is a finite domain. But when D is not finite, it is only a translation of the universal quantifier from one formalism into another.

3. $[\exists xB]_{(I,e)} = 1$ *if and only if* there is a *state f identical to e, except for x, such that* $[B]_{(I,f)} = 1$*. The above definition can be rephrased as follows:*

$$[\exists xB]_{(I,e)} = max_{d\in D}[B]_{(I,e[x=d])} = \sum_{d\in D}[B]_{(I,e[x=d])},$$

where the sum is the Boolean sum.

Example 4.3.31 *Let I be the interpretation with domain $D = \{1,2,3\}$ where the binary relation* friend *is true for couples $(1,2)$, $(1,3)$ and $(2,3)$, i.e. $friend_I^r{}^2 = \{(1,2),(1,3),(2,3)\}$. The formula $firend(1,2) \wedge friend(2,3) \Rightarrow friend(1,3)$ is true in the interpretation I,* i.e., $[friend(1,2) \wedge friend(2,3) \Rightarrow friend(1,3)]_I = 1$.

Example 4.3.32 *Let us Use the interpretation I given in example 4.3.19 on page 73. According to the meaning of the existential quantifier, we must evaluate $\ell(x,x)$ for $x = 0$, $x = 1$, and $x = 2$. We simplify this computation and the following computations by immediately replacing variables with their values: this avoids us using the states.*

— $[\exists x\, \ell(x,x)]_I =$

— $[\forall x \exists y\, \ell(x,y)]_I = ([\ell(0,0)]_I + [\ell(0,1)]_I + [\ell(0,2)]_I).\ ([\ell(1,0)]_I + [\ell(1,1)]_I + [\ell(1,2)]_I).\ ([\ell(2,0)]_I + [\ell(2,1)]_I + [\ell(2,2)]_I)$
$= (false + true + false).(true + false + false).(true + false + false) = true.true.true = true.$

— $[\exists y \forall x\, \ell(x,y)]_I =$

Remark 4.3.33 *In the above interpretation, the formulas $\forall x \exists y\, \ell(x,y)$ and $\exists y \forall x\, \ell(x,y)$ do not have the same value. Thus, when inverting an existential quantifier with a universal quantifier, we do not preserve the meaning of formulas.*

4.3.5 Model, validity, consequence, equivalence

These concepts are defined as in propositional logic. But whereas in propositional logic, an assignment is an application from propositional variables into $\{0,1\}$, in first-order logic an assignment is a couple consisting of a interpretation of the symbols on the one hand and the state of the variables on the other hand.

4.3.6 Instantiation

Definition 4.3.34 (Instantiation) *Let x be a variable, t a term and A a formula.*

1. *$A<x := t>$ is the formula obtained by replacing every free occurrence of x with the term t in the formula A.*
2. *The term t is free for x in A if the variables of t are not bound in the free occurrences of x in A.*

Example 4.3.35 *The term z is free for x in the formula $\exists y p(x,y)$. On the other hand the term y, like any term including the variable y, is not free for x in this formula. Let A be the formula $(\forall x P(x) \vee Q(\mathbf{x})$, the formula $A<x := b>$ is*

Theorem 4.3.36 *Let A be a formula and t a term free for the variable x in A. Let I be an interpretation and e a state of the interpretation. We have $[A<x := t>]_{(I,e)} = [A]_{(I,e[x=d])}$, where $d = [\![t]\!]_{(I,e)}$.*

In other words, the value of $A<x := t>$ in an assignment is the same as A in an identical assignment, except that it maps x to the value of the term t. This theorem, whose result is obvious, can be proved by an induction (which we will not detail) over the size of the formulas. We only show that the condition on t is mandatory by observing an example where this condition is not respected.

Example 4.3.37 *Let I be the interpretation with domain $\{0,1\}$ with $p_I = \{(0,1)\}$ and e, a state where $y = 0$. Let A be the formula $\exists y p(x,y)$ and t the term y.* This term is not free for x in A. *We have:*

— $A<x := y>=$

and $[A<x := y>]_{(I,e)} =$

— *Let $d = [\![t]\!]_{(I,e)} = [\![y]\!]_{(I,e)} = 0$. In the assignment $(I, e[x = d])$, we have $x = 0$. So we get:*
$[A]_{(I,e[x=d])} =$

Thus, $[A<x := t>]_{(I,e)} \neq [A]_{(I,e[x=d])}$, with $d = [\![t]\!]_{(I,e)}$.

Corollary 4.3.38 *Let A be a formula and t a term free for x in A. The formulas $\forall x A \Rightarrow A<x := t>$ and $A<x := t> \Rightarrow \exists x A$ are valid.*

This corollary is an immediate consequence of the previous theorem.

4.3.7 Finite interpretation

We show how to look for finite models of closed formulas (i.e. without free variables). A finite model of a closed formula is an interpretation of the formula with a finite domain, which makes the formula true. It is clear that the name of the domain elements is irrelevant, so when we are looking for a model with a domain of n items, the domain we will use will be that of (natural) integers less than n. To find out whether a closed formula has a model with the domain $\{0,\ldots,n-1\}$, it is sufficient to list all possible interpretations of the signature associated with the formula and to evaluate the formula for these interpretations. But this method is unusable in practice, because the number of these interpretations is enormous. If we have a signature with a constant, a function symbol with one argument and a symbol of relation with two arguments (and possibly the equality which has a fixed meaning), over a domain with 5 elements, this signature has $5 \times 5^5 \times 2^{25} = 524,288,000,000$ interpretations. Thus, we show first, in the case *where a formula has no function symbol and no constant, except for representations of integers less than n*, how we can look for its models with n elements by reduction to the propositional case. In the presence of function symbols and constants, we can enumerate their values, then apply the methods below. To find clever enumerations, we recommend reading the manual for Prover9 [22].

4.3.7.1 Integers and their representations

We distinguish in the folowing an integer n and its representation $\underline{n}$. For example the integer 3 can be represented in base 10 by 3, in base 2 by 11, in Roman numerals per III, or as the Greek digit γ. As is usual, we choose the representation in base 10. In the following, implicitly, all the interpretations we consider give to each representation of an integer, the value of the integer represented.

4.3.7.2 Expansion of a formula

In some cases it is sufficient to look at the expansions of finite size in order to find a model or counter-model for a given formula.

Definition 4.3.39 (n-expansion) *Let A be a formula and n an integer. The* n-expansion of A *is the formula obtained by replacing any subformula of A of the form $\forall xB$ with the conjunction $(\prod_{i<n} B{<}x := \underline{i}{>})$ and any subformula of A of the form $\exists xB$ with the disjunction $(\sum_{i<n} B{<}x := \underline{i}{>})$ where $\underline{i}$ is the decimal representation of the integer i.*

Example 4.3.40 *The 2-expansion of the formula $\exists xP(x) \Rightarrow \forall xP(x)$ is the formula*

Theorem 4.3.41 *Let n be an integer and A a formula containing only representations of integers lesser than n. Let A' be the n-expansion of A. Any interpretation with domain $\{0,\ldots,n-1\}$ assigns the same value to A and A'.*

The condition on A is necessary because if A includes a representation of an integer at least equal to n, the value of this representation will not be in the domain of the interpretation. The proof of the theorem is an induction on the size of formulas, that we will not detail. We merely show that the elimination of a universal quantifier in the formula $\forall xB$ retains the value of this formula.

Let (I,e) be an interpretation and a state with domain $\{0,\ldots,n-1\}$ giving each representation of an integer, the value of that integer. By definition of the meaning of the universal quantifier: $[\forall xB]_{(I,e)} = \prod_{i<n}[B]_{(I,e[x=i])}$. According to theorem 4.3.36 on the previous page and the fact that the value of the representation of the integer i is i, we have: $[B]_{(I,e[x=i])} = [B{<}x := \underline{i}{>}]_{(I,e)}$. As a result: $[\forall xB]_{(I,e)} = \prod_{i<n}[B{<}x := \underline{i}{>}]_{(I,e)} = [\prod_{i<n} B{<}x := \underline{i}{>}]_{(I,e)}$.

4.3.7.3 Interpretation and propositional assignment

Let n be an integer and A a closed formula, with no quantifier, no equality, no function symbol, and no constant except for representations of integers less than n. Let P be the set of atomic formulas in A (except $\top$ and $\bot$ whose meaning is fixed).

From assignment to interpretation.

Theorem 4.3.42 *Let v be a propositional assignment from P to $\{0,1\}$ then there is an interpretation I of A such that $[A]_I = [A]_v$.*

Proof : Let v be a propositional assignment from P to $\{0,1\}$. Let I be the following interpretation of A:

1. Domain: $\{0,\ldots,n-1\}$
2. $s_I^{r0} = v(s)$ if and only if $s \in P$
3. Let $p \geq 1$ and s^{rp} be a relation symbol of A. Then $s_I^{rp} = \{(k_1,\ldots,k_p)\underline{k_p}) \in P$ and $v(s(\underline{k_1}),\ldots,\underline{k_p}) = 1\}$

By definition of I, the assignment v and the interpretation I give the same value to the atomic subformulas of A, therefore (by induction on the formulas) the same value to the formula A. □

Example 4.3.43 *The assignment v, defined by $p(0) = 1$ and $p(1) = 0$, gives the value 0 to the following formula :*

$$(p(0) + p(1)) \Rightarrow (p(0).p(1)).$$

Thus, I defined by $p_I = \{0\}$ also gives the value 0 to that same formula. This example shows that v and I are two similar ways of presenting an interpretation, the second often being more concise.

From interpretation to assignment.

Theorem 4.3.44 *Let I be an interpretation of A then there is a assignment v of P such that*

$$[A]_I = [A]_v.$$

Proof : Let I be an interpretation of A and v the following propositional assignment from P to $\{0,1\}$: for any $B \in P$, $v(B) = [B]_I$. By definition of v, the assignment v and the interpretation I give the same value to the atomic subformulas of A, hence the same value to the formula A. □

4.3.7.4 Search for a finite model of a closed formula

Closed formula with no function symbol: Let A be a closed formula with no function symbol or constant, except for representations of integers less than n. To find an interpretation I model of A with domain $\{0,\ldots,n-1\}$ giving to each representation of an integer the value of that integer, we proceed as follows:

1. We replace A with its n-expansion B.
2. In the formula B, we replace the equalities with their value, that is, $\underline{i} = \underline{j}$ is replaced with 0 if $i \neq j$ and by 1 if $i = j$. Moreover, we recommended to eliminate these truth values through the identities $x+0=x$, $x+1=1$, $x*0=0$, $x*1=x$. Let C be the formula obtained after these replacements and simplifications.
3. We are looking for a propositional assignment v of the atomic formulas in C, which is a model of C: if such an assignment does not exist, A has no model, otherwise the interpretation I deduced from v as shown in theorem 4.3.42 on the facing page is a model of A.

Let us prove the correctness of this method:

1. Assume there is no propositional assignment model of C, but A has a model I. According to theorem 4.3.41 on the preceding page, I is a model of B, hence a model of C, and according to theorem 4.3.44 on the facing page, there is a propositional assignment model of C. From this contradiction, we deduce that A has no model with n elements.
2. Assume the propositional assignment v is a model of C. Thus, the interpretation I built as indicated in theorem 4.3.42 on the preceding page is a model of C, so it is model of B, so also according to theorem 4.3.41 on the facing page, it is a model of A.

Example 4.3.45 *Let A be the formula $\exists x P(x) \wedge \exists x \neg P(x) \wedge \forall x \forall y (P(x) \wedge P(y) \Rightarrow x = y)$. It is clear that this formula does not have a one-element model because this element should satisfy both the property P and its negation. The 2-expansion of A is:*

Closed formula with function symbols: Let A be a closed formula that may include representations of integers less than n. As in the previous case, we replace A with its expansion, we remove the equalities. Then we enumerate the choices of values for the symbols as in *the algorithm* `DPLL`, propagating as much as possible each of the choices we made.

Example 4.3.46 *Let A be the formula $\exists y\ P(y) \Rightarrow P(a)$, for which we look for a 2-element counter-model (which is equivalent to finding a model of the negation of A).*

Example 4.3.47 *We're looking for a 2-element model of the formulas $P(a)$, $\forall x(P(x) \Rightarrow P(f(x))$, $\neg P(f(b))$.*

4.3.8 Substitution and replacement

The replacement and substitution properties already stated for propositional logic (see 1.3.1 on page 19 and 1.3.10 on page 21) can be extended to first-order logic[3]. More specifically, the application of a substitution to a *propositionally* valid formula gives a valid formula. For example let σ be the propositional substitution such that $\sigma(p) = \forall x\, q(x)$. Since the formula $p \vee \neg p$ is valid, so is the formula $\sigma(p \vee \neg p) = \forall x\, q(x) \vee \neg \forall x\, q(x)$. The principle of replacement also extends to first-order logic with the same statement as for propositional logic because it is deduced from the following basic properties. For any formulas A and B and any variable x:

— $(A \Leftrightarrow B) \models (\forall xA \Leftrightarrow \forall xB)$.
— $(A \Leftrightarrow B) \models (\exists xA \Leftrightarrow \exists xB)$.

4.4 Important equivalences

In this section we give the rules for simplifying the formulas of first-order logic. Notice that the simplification rules of propositional logic are of course still applicable in the first-order.

4.4.1 Relationship between $\forall$ and $\exists$

Lemma 4.4.1 *Let A be a formula and x a variable.*

1. $\neg \forall xA \equiv \exists x \neg A$.
2. $\forall xA \equiv \neg \exists x \neg A$.

[3]Notice that in Stephen Cole Kleene's book [19], this notion of substitution over propositional variables is extended to a broader substitution applicable to all the relation symbols.

3. $\neg\exists xA \equiv \forall x\neg A$.
4. $\exists xA \equiv \neg\forall x\neg A$.

Let us prove the first two identities, the other two are asked in exercise 78 on page 86:

1. Let I be an interpretation with domain D and let e be a state.

$$\begin{array}{lcll} [\neg\forall xA]_{(I,e)} & \equiv & 1-[\forall xA]_{(I,e)} & \\ & \equiv & 1-\prod_{d\in D}[A]_{(I,e[x=d])} & \text{based on the meaning of } \forall \\ & \equiv & \sum_{d\in D}\left(1-[A]_{(I,e[x=d])}\right) & \text{through generalized De Morgan's laws} \\ & \equiv & \sum_{d\in D}[\neg A]_{(I,e[x=d])} & \text{by definition of the meaning of } \neg \\ & \equiv & [\exists x\neg A]_{(I,e)} & \text{by definition of the meaning of } \exists \end{array}$$

   When critically examining this evidence, we see that it uses generalized De Morgan's laws, which is the expression into another formalism of a similar law. It is therefore necessary to formalize the use of quantifiers to avoid these proofs that are actually only changes in notations. We will do this by adding to natural deduction some rules about the quantifiers.

2. The other equivalences are provable from the first by the principle of replacement. We only prove the second identity:

$$\begin{array}{lcll} \forall xA & \equiv & \neg\neg\forall xA & \text{identity of double negation} \\ & \equiv & \neg\exists x\neg A & \text{by identity 1 on the facing page} \end{array}$$

4.4.2 Moving quantifiers

We give, without proof, equivalences that apply to all formulas (with or without free variables) and which allow to find the usual laws for moving quantifiers. We assume that x, y are variables and that A, B are formulas.

1. $\forall x\forall yA \equiv \forall y\forall xA$.
2. $\exists x\exists yA \equiv \exists y\exists xA$.
3. $\forall x(A\wedge B) \equiv (\forall xA\wedge\forall xB)$.
4. $\exists x(A\vee B) \equiv (\exists xA\vee\exists xB)$.
5. Let Q be one of the quantifiers $\forall,\exists$, and $\circ$ one of the operations $\wedge,\vee,\Rightarrow$. Assume x is not a free variable of A.
   (a) $QxA \equiv A$,
   (b) $Qx(A\circ B) \equiv (A\circ QxB)$.

Example 4.4.2 *We eliminate from these two formulas the useless quantifiers:*

— $\forall x\exists xP(x) \equiv$

— $\forall x(\exists xP(x)\vee Q(x)) \equiv$

4.4.3 Renaming of bound variables

Theorem 4.4.3 *Let Q be one of the quantifiers $\forall,\exists$. Assume y is a variable that does not appear in QxA then: $QxA \equiv QyA<x := y>$.*

Example 4.4.4 *The formula $\forall x\ p(x,z)$ is equivalent to the formula $\forall y\ p(y,z)$ (according to the above theorem) but it is not equivalent to the formula $\forall z\ p(z,z)$, where the change of variable does not satisfy the conditions of the theorem.*

Definition 4.4.5 (Formulas equal up to renaming of bound variables) *Two formulas are* equal up to renaming of bound variables *if we can get one from the other by replacing subformulas of the form QxA with $QyA<x := y>$ where Q is a quantifier and y is a variable that does not appear in QxA. Two formulas equal up to renaming of bound variables, are also said to be copies of each other or even* α-equivalent.

Theorem 4.4.6 *If two formulas are equal up to renaming of bound variables then they are equivalent.*

This result is clearly a consequence of theorem 4.4.3 on the previous page. We will not write a full proof, which is long and tedious. We only outline this proof with an example.

Example 4.4.7 *We show that the formulas $\forall x \exists y P(x,y)$ and $\forall y \exists x P(y,x)$ are equal up to renaming of bound variables, according to definition 4.4.3 on the preceding page and are therefore equivalent, indeed:*

Our definition of equality between two formulas up to renaming of bound variables is not effective, as the definition does not give a test. But it is simple to see whether two formulas are equal up to renaming of bound variables: we draw lines between each quantifier and the variables it binds and we erase the names of bound variables. If after this transformation, both formulas become identical, it means that they are equal up to renaming of bound variables.

Example 4.4.8 *The two formulas $\forall x \exists y P(y,x)$ and $\forall y \exists x P(x,y)$ are converted into $\forall|\exists|\underline{P(|,|)}$.*

4.5 Exercises

Exercise 64 (Structure and free variables)
For each formula hereafter, indicate its structure and its free variables.

1. $\forall x(P(x) \Rightarrow \exists y Q(x,y))$.
2. $\forall a \forall b(b \neq 0 \Rightarrow \exists q \exists r(a = b*q+r \wedge r < b))$[4].
3. $Even(x) \Leftrightarrow \exists y(x = 2*y)$.
4. $Divides(x,y) \Leftrightarrow \exists z(y = z*x)$.
5. $Prime(x) \Leftrightarrow \forall y(Divides(y,x) \Rightarrow y = 1 \vee y = x)$.

□

Exercise 65 (Formalization, function and relation symbols) *Consider* $\Sigma = \{b^{r2}, u^{r2}, c^{r2}, o^{r2}, r^{f0}, f^{f1}\}$, *a signature with the following meaning:*

— $b(x,y) :=$ *x is one of y's brothers.*
— $u(x,y) :=$ *x is one of y's uncles.*
— $c(x,y) :=$ *x is one of y's cousins.*
— $o(x,y) :=$ *x is older than y.*
— *r is Robert's nickname.*
— $f(x)$ *returns x's father.*

Formalize the following sentences in first order logic using signature Σ.

1. *Every brother of Robert's father is Robert's uncle.*
2. *If the fathers of two children are brothers, then these children are cousins.*
3. *One of Robert's cousins is younger than one of Robert's brothers.*

Translate in English the following logical propositions:

1. $\neg\exists x\, o(x, p(x))$
2. $\forall x \forall y(f(f(x)) = f(f(y)) \Rightarrow c(x,y))$
3. $\forall x \exists y(b(x,y) \wedge o(y,x))$
4. $\exists x \exists y(b(x,y) \wedge \neg(f(x) = f(y)))$

□

Exercise 66 (Formalization) *Consider signature* $\Sigma = \{g^{f0}, f^{f0}, P^{r2}, W^{r2}\}$, *whose symbols have the following meaning:*

— $g :=$ *Germany's team.*
— $f :=$ *France's team.*
— $P(x,y) :=$ *x played against y.*
— $W(x,y) :=$ *x won against y.*

Formalize the following sentences in first order logic using siganture Σ.

1. *France's team won a match and lost a match.*
2. *France's and Germany's team tied.*
3. *A team won all its matches.*
4. *No team lost all its matches.*
5. *Consider the following assertion: "All the teams who played against a team that won all its matches won at least one match". Which of the following formulas formalize this assertion, and which are equivalent?*
   (a) $\forall x \exists y(P(x,y) \wedge \forall z(P(y,z) \Rightarrow W(y,z)) \Rightarrow \exists v W(x,v))$.
   (b) $\forall x(\exists y(P(x,y) \wedge \forall z(P(y,z) \Rightarrow W(y,z))) \Rightarrow \exists v W(x,v))$.
   (c) $\exists x(\forall y(P(x,y) \Rightarrow W(x,y)) \Rightarrow \forall z(P(x,z) \Rightarrow \exists v W(x,v)))$.
   (d) $\forall x \forall y(P(x,y) \wedge \forall z(P(y,z) \Rightarrow W(y,z)) \Rightarrow \exists v W(x,v))$.
   (e) $\forall x(\forall y(P(x,y) \wedge \forall z(P(y,z) \Rightarrow W(y,z))) \Rightarrow \exists v W(x,v))$.

□

[4] In order to respect the usual notation for the Euclidean Algorithm, we exceptionally use a, b, q and r as variable names

Exercise 67 (Formalization,)** *We define constants s for Serge, t for Tobby, $L(x,y)$ for x loves y, D(x) for x is a dog, $P(x)$ for x is a pet, $K(x)$ for x is a kid, $B(x)$ for x is a bird and $S(x,y)$ for x is scared of y. Describe the signature Σ associated for these symbols, and formalize the following statements in first order logic using that signature.*

1. *Dogs and birds are pets.*
2. *Tobby is a dog that loves kids.*
3. *Birds do not like dogs.*
4. *Serge loves all pets except dogs.*
5. *All kids are not scared of dogs.*
6. *Some dogs love kids.*
7. *Some dogs love kids, and vice versa.*
8. *Kids love some dogs.*

□

Exercise 68 (Evaluation of unary predicates) *Let I be the interpretation on domain $D = \{0,1\}$ with $P_I = \{0\}$ and $Q_I = \{1\}$.*

1. *Evaluate the following formulas with I: $\forall x P(x)$ and $\forall x(P(x) \vee Q(x))$.*
2. *Are the formulas $\forall x P(x) \vee \forall x Q(x)$ and $\forall x(P(x) \vee Q(x))$ equivalent?*
3. *Evaluate the following formulas with I: $\exists x P(x)$ and $\exists x(P(x) \wedge Q(x))$.*
4. *Are the formulas $\exists x P(x) \wedge \exists x Q(x)$ and $\exists x(P(x) \wedge Q(x))$ equivalent?*
5. *Evaluate the following formulas with I: $\forall x(P(x) \Rightarrow Q(x))$ and $\forall x P(x) \Rightarrow \forall x Q(x)$.*
6. *Are the formulas $\forall x(P(x) \Rightarrow Q(x))$ and $\forall x P(x) \Rightarrow \forall x Q(x)$ equivalent?*

□

Exercise 69 (Interpretation,*) *Consider the following formulas:*

1. $\forall x \exists y(y = x+1)$.
2. $\exists y \forall x(y = x+1)$.
3. $\forall x \exists y(y = x+1) \Rightarrow \exists y \forall x(y = x+1)$.
4. $\forall x \exists y(x = y+1)$.
5. $\exists x \forall y(y = x+y)$.
6. $\exists x(x \neq 0 \wedge x+x = x)$.

and the following interpretations:

1. *I1 is Boolean Algebra on $\{0,1\}$.*
2. *I2 is the usual arithmetic on natural numbers.*
3. *I3 is the usual arithmetic on rational numbers.*
4. *I4 is the Boolean Algebra on $\mathcal{P}(X)$ where constants 0 and 1 denote the sets $\emptyset$ and X, and addition represents set union.*

Indicate if these interpretations are models or counter-models of the formulas above. □

Exercise 70 (Unary predicats and equality) *Let I be the interpretation on the domain $D = \{0,1,2\}$ with $P_I = \{0,1\}, Q_I = \{1,2\}, R_I = \{\}$. Evaluate the following formulas with this interpretation:*

1. $\exists x R(x)$.
2. $\forall x(P(x) \vee Q(x))$.
3. $\forall x(P(x) \Rightarrow Q(x))$.
4. $\forall x(R(x) \Rightarrow Q(x))$.
5. $\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$[5].

[5] This formula means that there is only one element satisfying *P*.

6. $\exists x(P(x) \wedge Q(x) \wedge \forall y(P(y) \wedge Q(y) \Rightarrow x = y))$.

□

Exercise 71 (Evaluation, equality) *We use the unary function symbol f and the constant a. Let $x \neq y$ be a shorthand for $\neg(x = y)$. We define interpretations I_1, I_2 on domain $\{0,1,2\}$ as follows:*
$a_{I_1} = a_{I_2} = 0$.

x	$f_{I_1}(x)$	$f_{I_2}(x)$
0	0	1
1	0	2
2	2	0

In interpretation I_1, then in I_2, evaluate the following formula:

1. $f(a) = a$.
2. $f(f(a)) = a$.
3. $f(f(f(a)) = a$.
4. $\exists x(f(x) = x)$, *(f has a fixed point).*
5. $\forall x(f(f(f(x))) = x)$.
6. $\forall y \exists x(f(x) = y)$, *($f$ is surjective, or onto).*
7. $\forall x \forall y(f(x) = f(y) \Rightarrow x = y)$, *($f$ is injective, or one-to-one.*
8. $\neg \exists x \exists y(f(x) = f(y) \wedge x \neq y)$.

□

Exercise 72 (Formalization and evaluation) *We use the following notation:*

— $P(x)$ *means x passed his exam*
— $Q(x,y)$ *means x called y*

Give the signature associated with these symbols, and formalize the following statements:

1. *Someone failed the exam and was not called by anyone.*
2. *All those who passed the exam received a call.*
3. *Nobody called those who passed the exam.*
4. *All those cho called someone else called someone who passed the exam.*

We define an interpretation with domain $D = \{0,1,2,3\}$. Let Anatoli, Boris, Catarina and Denka be four constants with value 0, 1, 2, 3 respectively. Anatoli and Boris are boys, Catarina and Denka are girls. In our interpretation, only Boris and Catarina passed the exam, every boy called every girl, Denka called Boris, Catarina called Denka and no other calls were made. Evaluate the statements above in this interpretation.
Indication *: to facilitate the evaluation, it might help to represent the interpretation pictorially, by circling people who passed the exam and drawing an arrow from x to y if x called y.* □

Exercise 73 (Expansion and counter-models) *Using the method of expansion, find counter-models to the following formulas:*

1. $\exists x P(x) \Rightarrow \exists x(P(x) \wedge Q(x))$.
2. $\forall x(P(x) \Rightarrow Q(x)) \Rightarrow \exists x Q(x)$.
3. $\forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\exists x P(x) \Rightarrow \forall x Q(x))$.
4. $(\exists x F(x) \Rightarrow \exists x G(x)) \Rightarrow \forall x(F(x) \Rightarrow G(x))$.
5. $\forall x \exists y R(x,y) \Rightarrow \exists x R(x,x)$.
6. $\forall x \forall y(R(x,y) \Rightarrow R(y,x)) \Rightarrow \forall x R(x,x)$.

Hint *: it should be enough to build 1 or 2 expansions.* □

Exercise 74 (Incorrect reasoning) *Consider the following hypotheses:*

1. $\exists x P(x)$.

2. $\exists x Q(x)$.
3. $\forall x(P(x) \wedge Q(x) \Rightarrow R(x))$.

Show that it is incorrect to deduce from these that $\exists x R(x)$. □

Exercise 75 (Counter-model with a relation) *Construct counter-models for the following formulas, where* F *is a relation:*

1. $\forall x \exists y(x = y) \Rightarrow \exists y \forall x(y = x)$.
2. $F(a) \wedge (a \neq b) \Rightarrow \neg F(b)$.
3. $\exists x \exists y(F(x) \wedge F(y) \wedge x \neq y) \Rightarrow \forall x F(x)$.
4. $\forall x \forall y(F(x,y) \Rightarrow x = y) \Rightarrow \exists x F(x,x)$.

□

Exercise 76 (Counter-model with a function) *Construct counter-models for the following formulas, where* f *is a function and* P *is a relation :*

1. $\forall y \exists x(f(x) = y)$.
2. $\forall x \forall y(f(x) = f(y) \Rightarrow x = y)$.
3. $\exists x \forall y(f(y) = x)$.
4. $\forall x(P(x) \Rightarrow P(f(x))$.

□

Exercise 77 (Equivalences) *Prove that*

1. $\neg \forall x \exists y P(x,y) \equiv \exists y \forall x \neg P(y,x)$.
2. $\exists x(P(x) \Rightarrow Q(x)) \equiv \forall x P(x) \Rightarrow \exists x Q(x)$.
3. *The sentence "no sick person likes charlatans" was formalized in first order logic by two students as follows:*
   — $\forall x \forall y((S(x) \wedge L(x,y)) \Rightarrow \neg C(y))$.
   — $\neg(\exists x(S(x) \wedge (\exists y(L(x,y) \wedge C(y)))))$.
   Show that both students are saying the same thing, that is, these two formulas are equivalent.

□

Exercise 78 (, Proofs,*) *Prove the following two equivalences from Lemma 4.4.1 on page 80:*

— $\neg \exists x A \equiv \forall x \neg A$.
— $\exists x A \equiv \neg \forall x \neg A$.

Hint *: use properties 1 and 2 from Lemma 4.4.1 on page 80.* □

Exercise 79 (Proof) *We know that* $(\forall x(A \wedge B)) \equiv (A \wedge (\forall x B))$ *under the condition that* x *is not a free variable in* A*, as mentioned in paragraph 4.4.2 on page 81. Show that this condition is necessary by giving an assignment that give different values to the formulas* $\forall x(P(x) \wedge Q(x))$ *and* $P(x) \wedge (\forall x Q(x))$ *when this condition is not met.* □

Chapter 5

Basis of proof automation

Contents

It is important to emphasize that first-order logic is *undecidable*, which means there is no algorithm for *determining* whether a formula is valid or invalid. This result was established by Alonzo Church and Alan Turing in 1936 and 1937 [4, 24]. They also showed that this logic is *semi-decidable*, that is to say we can write a program, which takes a formula and has the following behavior:

1. If it ends then it correctly decides whether the formula is valid or not. When the formula is valid, the decision usually comes with a proof.
2. If the formula is valid, then it ends. However, the execution can be long!

Notice that *if the formula is not valid, the termination of this program is not guaranteed*. Indeed if the program ended for any formula, then, according to the first point, it would be an algorithm to determine whether a formula is valid or not. However, as stated above, Church and Turing have proved that such an algorithm did not exist.

Outline: In this chapter we start with a method introduced by Jacques Herbrand, allowing to enumerate a finite set of formulas in order to find a model for a formula. Then we present the transformation due to Thoralf Albert Skolem which transforms a formula into a "Skolem form" where the existence of a model is preserved. By then transforming this Skolem form into an equivalent "clausal form", we can apply resolution. Then we take the concept of resolution introduced in chapter 2 on page 35 and we adapt it in first-order logic. In 1929, Kurt Gödel (1906 - 1978) proved that first-order logic is complete and consistent. We present this result for resolution, *i.e.* there is a proof by resolution in first-order logic of a formula F from a set of formulas Γ if and only if F is a consequence of Γ.

5.1 Herbrand's method

Jacques Herbrand (1908-1931) was a French mathematician and logician. In 1930 he established a link between the predicate calculus and the propositional calculus.

5.1.1 The Herbrand universe and base

We first introduce the notion of universal closure which makes it possible to transform any formula into a closed formula while maintaining satisfiability.

Definition 5.1.1 (Universal closure) *Let C be a formula with free variables $x_1, \ldots, x_n$. The* universal closure *of C, noted $\forall(C)$, is the formula $\forall x_1 \ldots \forall x_n C$. This notion is defined up to the order of the free variables in C. Let Γ be a set of formulas, $\forall(\Gamma) = \{\forall(A) \mid A \in \Gamma\}$.*

Example 5.1.2 *The universal closure of $P(x) \wedge R(x,y)$ is*

We generalize definition 1.3.1 on page 19 for first-order logic.

Definition 5.1.3 (Substitution) *A substitution is a mapping from variables to terms. Let A be a formula and σ a substitution. $A\sigma$ is the formula obtained by replacing every free occurrence of a variable by its image in the mapping. The formula $A\sigma$ is an* instance *of A.*

In the remainder of this chapter, we will only consider formulas which contain neither the equality symbol nor the propositional constants $\top$ and $\bot$, *because their meaning is fixed in every interpretation*: $\top$ is 1, $\bot$ is 0, and the $=$ symbol is the identity over the domain of the interpretation (see Note 4.3.18 on page 73). In addition we consider that any signature shall include at least one constant. Thus the signature of a set of formulas having no constant, will be arbitrarily completed by the constant a.

Definition 5.1.4 (Herbrand universe and base)

1. *The* Herbrand universe *for Σ is the set of closed terms (*i.e. *without variables) of this signature, denoted by D_Σ.*
2. *The* Herbrand base *for Σ is the set of closed atomic formulas of this signature, denoted by B_Σ.*

Notice that the Herbrand universe of a signature without constant is empty. Since all signatures considered in this chapter must have at least one constant, this domain is not empty.

Example 5.1.5 *Herbrand base computations:*

1. *Let Σ be the signature containing only the 2 constants a, b and the 2 unary relation symbols P, Q. We have $D_\Sigma = \{a, b\}$ and $B_\Sigma =$*
2. *Let Σ be the signature containing only the constant a, the unary function symbol f and the unary relation symbol P. We have $D_\Sigma = \{f^n(a) \mid n \in \mathbb{N}\}$ and $B_\Sigma =$*

5.1.2 Herbrand interpretation

Using the Herbrand base and universe of a formula, we are able to build a so-called "Herbrand interpretation".

Definition 5.1.6 (Herbrand interpretation) *Let Σ be a signature and $E \subseteq B_\Sigma$. The* Herbrand interpretation *$H_{\Sigma,E}$ has the domain D_Σ and gives symbols the following meaning:*

1. *If the s symbol is a constant of the signature, its value is itself in this interpretation.*
2. *If s is a function symbol of the signature with $n \geq 1$ arguments and if $t_1, \ldots, t_n \in D_\Sigma$ then $s^{fn}_{H_{\Sigma,E}}(t_1, \ldots, t_n) = s(t_1, \ldots, t_n)$.*
3. *If the s symbol is a propositional variable, its value is* 1, i.e. *it is true if and only if $s \in E$.*
4. *If s is a relation symbol signature with $n \geq 1$ arguments and if $t_1, \ldots, t_n \in D_\Sigma$ then $s^{rn}_{H_{\Sigma,E}} = \{(t_1, \ldots, t_n) \mid t_1, \ldots, t_n \in D_\Sigma \wedge s(t_1, \ldots, t_n) \in E\}$.*

The following result shows that a Herbrand interpretation can be assimilated to the set of closed atomic formulas it satisfies.

Property 5.1.7 (Property of a Herbrand interpretation) *Let Σ be a signature and $E \subseteq B_\Sigma$. In the Herbrand interpretation $H_{\Sigma,E}$:*

1. *The value of a term without variable is itself.*
2. *The interpretation is a model of a closed atomic formula if and only if it is an element of E.*

The proof is a direct consequence of the definition of a Herbrand interpretation.
Notice here, with an example, why we assumed that the formulas do not contain the relation symbols $\top, \bot, =$, whose meaning is fixed in every interpretation. Assume on the contrary that $\top$ is an element of the base but not an element of E. According to point 2, the interpretation $H_{\Sigma,E}$ would give $\top$ the value 0, while $\top$ has value 1 in any interpretation.

Example 5.1.8 *Let Σ be the signature consisting of the* 2 *constant a, b and the* 2 *unary relation symbols P, Q. The set $E = \{P(b), Q(a)\}$ defines the Herbrand interpretation $H_{\Sigma,E}$ with domain $D_\Sigma = \{a, b\}$*

here the constants a and b have value themselves and where $P_{H_{\Sigma,E}} = \{b\}$ and $Q_{H_{\Sigma,E}} = \{a\}$.

Theorem 5.1.16 (Universal closure and Herbrand model) *Let Γ be a set of formulas without quantifier over the signature Σ. The universal closure $\forall(\Gamma)$ has a model if and only if $\forall(\Gamma)$ has a model which is a Herbrand interpretation of Σ.*

This theorem is used in the proof of theorem 5.2.17 on page 94 .

5.1.3 Herbrand's theorem

Herbrand's theorem establishes a link between first-order logic and propositional logic.

Theorem 5.1.17 (Herbrand's theorem) *Let Γ be a set of formulas without quantifier over the signature Σ.*
$\forall(\Gamma)$ has a model if and only if every finite set of closed instances over the signature Σ of formulas of Γ has a propositional model mapping the Herbrand base B_Σ to $\{0, 1\}$.

Corollary 5.1.18 *Let Γ be a set of formulas without quantifier over the signature Σ. The universal closure $\forall(\Gamma)$ is unsatisfiable if and only if there is a finite unsatisfiable set of closed instances of the formulas of Γ over the signature Σ.*

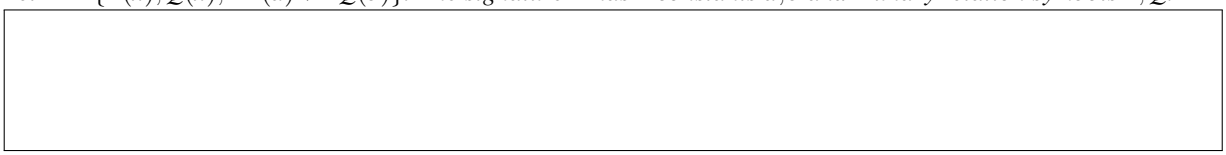
Proof : The corollary is obtained by replacing each part of the equivalence of Herbrand's theorem with its negation. □

In the case where Γ is a *finite* set of formulas, this corollary provides the grounds for a *semi-decision* procedure to know whether $\forall(\Gamma)$ is unsatisfiable or not: we enumerate the set of closed instances of the formulas in Γ over the signature Σ and we stop this enumeration as soon as we reach an unsatisfiable set, or when this enumeration is over without contradiction, or when we are "tired":

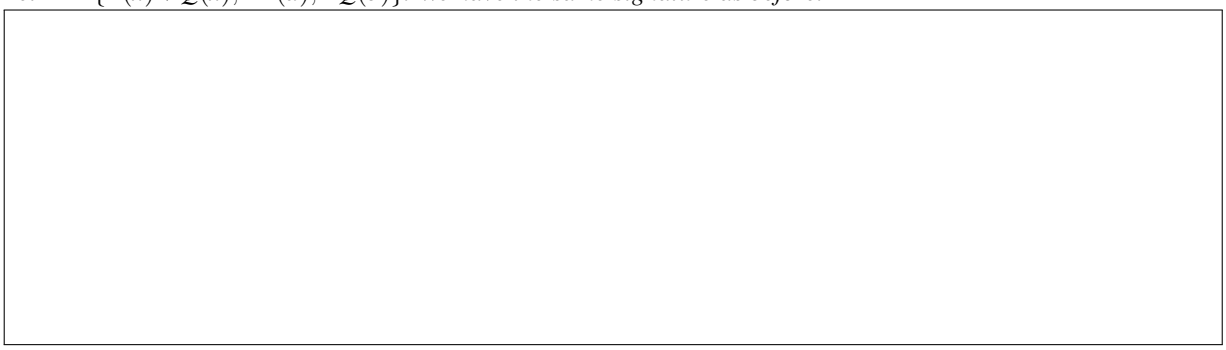
1. In the first case, the procedure answers (applying the corollary) that $\forall(\Gamma)$ is unsatisfiable.
2. The second case can only occur if the Herbrand universe only contains constants, the procedure answers (applying the corollary) that $\forall(\Gamma)$ is satisfiable and it gives a model.
3. In the third case, obviously we cannot conclude: the corollary tells us that if $\forall(\Gamma)$ is unsatisfiable, and if we had been more persevering, we would have reached a contradiction.

Example 5.1.19

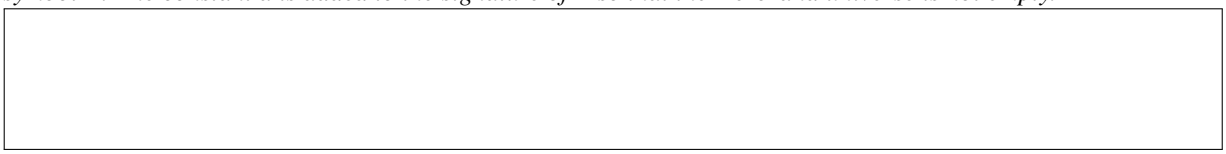
1. *Let* $\Gamma = \{P(x), Q(x), \neg P(a) \vee \neg Q(b)\}$. *The signature* Σ *has 2 constants* a, b *and 2 unary relation symbols* P, Q.



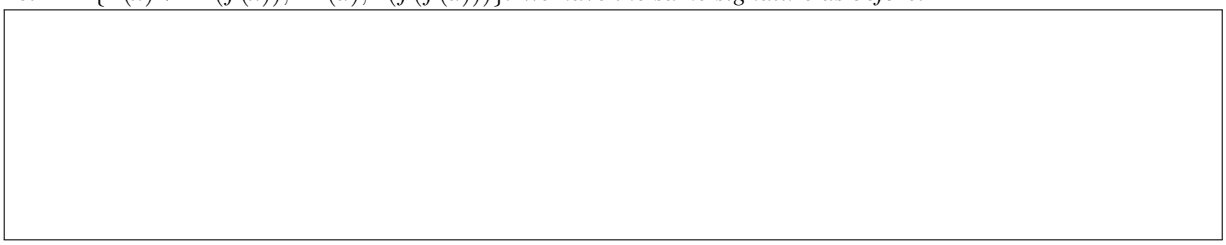
2. *Let* $\Gamma = \{P(x) \vee Q(x), \neg P(a), \neg Q(b)\}$. *We have the same signature as before.*



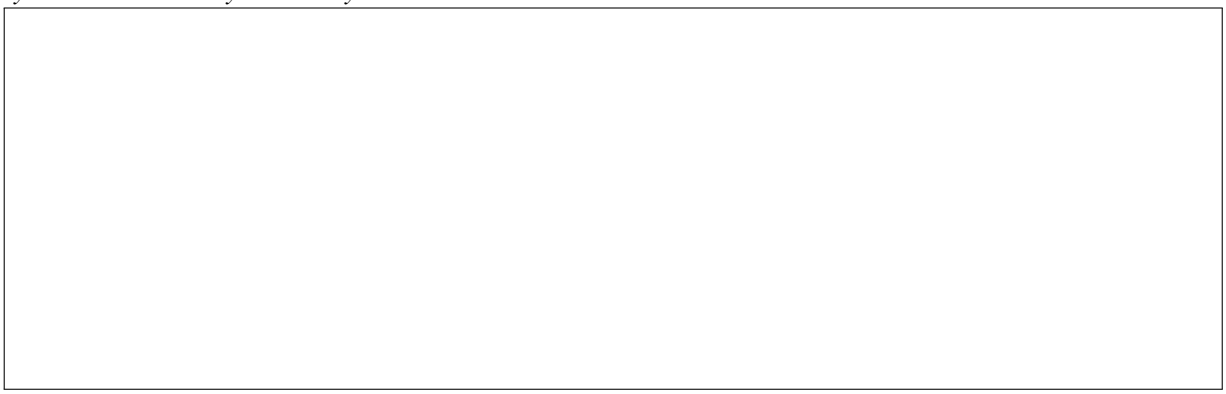
3. *Let* $\Gamma = \{P(x), \neg P(f(x)\}$. *The signature* Σ *consists of a constant a, a unary function symbol* f *and a unary relation symbol P. The constant a is added to the signature of* Γ *so that the Herbrand universe is not empty.*



4. *Let* $\Gamma = \{P(x) \vee \neg P(f(x)), \neg P(a), P(f(f(a)))\}$. *We have the same signature as before.*



5. *Let* $\Gamma = \{R(x, s(x)), R(x,y) \wedge R(y,z) \Rightarrow R(x,z), \neg R(x,x)\}$. *The signature* Σ *contains a constant a, a unary function symbol s and a binary relation symbol R.*



5.2 Skolemization

Herbrand's theorem can be applied to the universal closure of a set of formulas without quantifier. We study a transformation, called *Skolemization*, which turns a set of closed formulas into the universal closure of a set of formulas without quantifier and *which preserves the existence of a model.* This transformation is due to Thoralf Albert Skolem (1887 -

1963), Norwegian mathematician and logician. It is needed to be able to apply resolution to a set of first-order logic formulas. We begin by giving the intuition of Skolemization before looking at the result of this transformation on simple examples in order to observe the properties of Skolemization. Skolemization provides a way of eliminating the existential quantifiers and turns a closed formula A into a formula B such that:

— A is a consequence of B.

— every model of A gives a model of B.

Hence A has a model if and only if B has a model: Skolemization preserves the existence of a model, we also say that it preserves satisfiability. Let's see how it works on two simple examples.

Example 5.2.1 *The formula $\exists x P(x)$ is Skolemised into $P(a)$. Let us observe the relationship between these two formulas:*

1. *$\exists x P(x)$ is a consequence of $P(a)$.*
2. *$P(a)$ is not a consequence of $\exists x P(x)$, but a model of $\exists x P(x)$ "gives" a model of $P(a)$. Indeed let I be a model of $\exists x P(x)$. So there is an element $d \in P_I$. Let J be the interpretation such that $P_J = P_I$ and $a_J = d$. J is a model of $P(a)$.*

Example 5.2.2 *The formula $\forall x \exists y Q(x,y)$ is Skolemised into $\forall x Q(x, f(x))$. We observe the relationship between these two formulas:*

1. *$\forall x \exists y Q(x,y)$ is a consequence of $\forall x Q(x, f(x))$.*
2. *$\forall x Q(x, f(x))$ is not a consequence of $\forall x \exists y Q(x,y)$, but every model of $\forall x \exists y Q(x,y)$ "gives" a model of $\forall x Q(x, f(x))$. Indeed let I be a model of $\forall x \exists y Q(x,y)$ and D the domain of I. For every $d \in D$, the set $\{e \in D \mid (d,e) \in Q_I\}$ is not empty, so there is a function $g : D \to D$ [1] such that for any $d \in D$, $g(d) \in \{e \in D \mid (d,e) \in Q_I\}$. Let J be the interpretation such that $Q_J = Q_I$ and $f_J = g$: J is a model of $\forall x Q(x, f(x))$.*

5.2.1 Skolemization algorithm

We first define the elements necessary to present the Skolem form, then we give a Skolemization algorithm.

Definition 5.2.3 (Proper formula) *A closed formula is said to be* proper*, if it does not contain a variable bound by two distinct quantifiers.*

Example 5.2.4 *The formulas $\forall x P(x) \vee \forall x Q(x)$ and $\forall x (P(x) \Rightarrow \exists x Q(x) \wedge \exists y R(x,y))$ are not proper.*
The formulas $\forall x P(x) \vee \forall y Q(y)$ and $\forall x (P(x) \Rightarrow \exists y R(x,y))$ are proper.

We extend the notion of normal form of propostional logic to first-order logic by saying that a formula is in normal form if it is without equivalence, nor implication, and its negations are only applied to atomic formulas.

Definition 5.2.5 (Skolem form) *Let A be a closed formula and B the normal form without quantifier obtained by the transformation below: B is the* Skolem form *of A.*

There are several ways of Skolemizing a closed formula. We propose a compromise between efficiency and simplicity of the transformation. Our algorithm consists in the following sequence of four steps:

1. **Transformation into normal form**: Transformation of the closed formula into another equivalent closed formula in normal form.
2. **Transformation into a proper formula**: Transformation of the closed normal formula into an equivalent closed, normal and proper formula.
3. **Elimination of existential quantifiers**: This transformation preserves *only the existence of a model*, as we have seen on the examples.
4. **Transformation into Skolem form**: Transformation of the closed, normal, proper formula without existential quantifier into a normal formula without quantifier.

Let us detail each of these four steps.

[1] By stating the existence of g, we implicitly use the axiom of choice.

5.2.1.1 Transformation into a normal formula

As in the propositional case, we remove equivalences and implications and we move the negations towards atomic formulas using the following equivalences from left to right:

— $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$.
— $A \Rightarrow B \equiv \neg A \vee B$.
— $\neg\neg A \equiv A$.
— $\neg(A \wedge B) \equiv \neg A \vee \neg B$.
— $\neg(A \vee B) \equiv \neg A \wedge \neg B$.
— $\neg\forall xA \equiv \exists x\neg A$.
— $\neg\exists xA \equiv \forall x\neg A$.

Remark 5.2.6 *By combining moving a negation and eliminating an implication, i.e. replacing $\neg(A \Rightarrow B)$ with $A \wedge \neg B$, we can perform the transformation into an equivalent normal formula faster than by limiting ourselves to the above equivalences.*

Example 5.2.7 *The formula $\forall y(\forall xP(x,y) \Leftrightarrow Q(y))$ is transformed through elimination of equivalence and implication into:*

5.2.1.2 Transformation into a proper formula

Using the results on variable renaming presented in section 4.4.3 on page 81, we change the names of the variables bound by two quantifiers, for example by choosing new variables for each name change.

Example 5.2.8 *The formula $\forall xP(x) \vee \forall xQ(x)$ is changed into*

The formula $\forall x(P(x) \Rightarrow \exists xQ(x) \wedge \exists yR(x,y))$ is changed into

5.2.1.3 Elimination of existential quantifiers

Let A be a normal and proper closed formula featuring an occurrence of the subformula $\exists yB$. Let $x_1,\ldots,x_n$ be the set of free variables of $\exists yB$, where $n \geq 0$. Let f be a symbol that *does not appear in A*. We replace this occurrence of $\exists yB$ with $B<y := f(x_1,\ldots,x_n)>$[2]

Theorem 5.2.9 *Let A' be the formula obtained by applying the elimination of an existential quantifier in the formula A. The formula A' is a closed formula in normal and proper form verifying:*

1. *A is a consequence of A' ($A' \models A$).*
2. *If A has a model then A' has a model identical to the model of A except for the meaning of f.*

[2] If $n = 0$, f is a constant.

Remark 5.2.10 *In theorem 5.2.9, it must be noted that the formula A' obtained from formula A by elimination of a quantifier remains closed, in normal and proper form. So, by "applying" several times the theorem, which involves choosing a* new *symbol for each eliminated quantifier, we turn a closed, normal and proper formula A into a closed, normal and proper formula B* without existential quantifier *such that:*

— *the formula A is a consequence of the formula B,*

— *if A has a model, then B has an identical model except for the meaning of the new symbols.*

Example 5.2.11 *By eliminating the existential quantifiers in the formula $\exists x\forall y P(x,y) \wedge \exists z\forall u \neg P(z,u)$ we get*

Therefore, it is mandatory to use a new symbol for each elimination of an existential quantifier.

Example 5.2.12 *By eliminating the existential quantifiers in the formula $\exists x\forall y\exists z P(x,y,z)$ we get two possible solutions:*

— $\forall y P(a,y,f(a,y))$

— $\forall y P(a,y,f(y))$

This depends on the order in which we apply the elimination of existential quantifiers. But theorem 5.2.9 shows that one of these transformations has a model if and only if the other also has a model.

5.2.1.4 Transformation into universal closure

We can finally remove the universal quantifiers in order to obtain a Skolem form.

Theorem 5.2.13 *Let A be a closed formula, in normal form, proper and without existential quantifier. Let B be the formula obtained by removing every universal quantifier from A. The formula A is equivalent to the universal closure of B.*

Proof : With the conditions imposed on A, the transformation of A into $\forall(B)$ amounts to performing all possible replacements of subformulas of the following form:

— $(\forall xC) \wedge D$ into $\forall x(C \wedge D)$, where x not free in D,

— $(\forall xC) \vee D$ into $\forall x(C \vee D)$, where x not free in D,

— $D \wedge (\forall xC)$ into $\forall x(D \wedge C)$, where x not free in D,

— $D \vee (\forall xC)$ into $\forall x(D \vee C)$, where x not free in D.

Since each of these replacements turns a formula into another equivalent formula, the formulas A and $\forall(B)$ are equivalent. □

5.2.2 Properties of the Skolem form

Property 5.2.14 *Let A be a closed formula and B the Skolem form of A.*

— *The formula A is a consequence of $\forall(B)$,*

— *if A has a model then $\forall(B)$ has a model.*

Thus A has a model if and only if $\forall(B)$ has a model.

Proof : Let C be the closed formula in normal and proper form, obtained after the first two steps of Skolemizing A. Let D be the result of the elimination of existential quantifiers applied to C. According to remark 5.2.10 we have:

— the formula C is a consequence of the formula D,

— if C has a model then D has a model.

Since the first two steps turn formulas into equivalent formulas, A and C are equivalent. According to theorem 5.2.13, D is equivalent to $\forall(B)$. Thus we can replace D with $\forall(B)$ and C with A in the above statement, which proves the theorem. □

Example 5.2.15 *Let $A = \forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\forall xP(x) \Rightarrow \forall xQ(x))$. We Skolemize $\neg A$.*

1. *$\neg A$ is transformed into the normal form:*

2. *The formula in normal form is transformed into the proper formula:*

3. *The existential quantifier is "replaced" with a constant:*

4. *The universal quantifiers are removed:*

This formula is the Skolem form of $\neg A$ according to definition 5.2.5 on page 91.

Let us instantiate the Skolem form of $\neg A$ by replacing x and y with a. We get the formula $(\neg P(a) \vee Q(a)) \wedge P(a) \wedge \neg Q(a)$ which is unsatisfiable. Hence $\forall((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(a))$ is unsatisfiable. Since, according to property 5.2.14 on the previous page, Skolemization preserves the existence of a model, $\neg A$ is unsatisfiable, thus A is valid.

To Skolemize a set of formulas, we Skolemize each formula of the set as indicated above, taking care to choose a new symbol for each existential quantifier elimination during the third step of Skolemization. The reasons for choosing a new symbol appear clearly in the proof of theorem 5.2.9 on page 92 and in the example 5.2.11 on the previous page.

Corollary 5.2.16 *By Skolemizing a set of closed formulas Γ, we obtain a set of formulas Δ without quantifiers such that:*
- *Every model of $\forall(\Delta)$ is a model of Γ.*
- *If Γ has a model then $\forall(\Delta)$ has a model which differs from the model of Γ only by the meaning of the new symbols.*

Skolemization has practical consequences: it facilitates the proof of unsatisfiability of a formula. It also has theoretical consequences, which we examine.

Theorem 5.2.17 *Let Γ be a countable set of closed formulas. If Γ has a model then Γ has a countable model.*

Proof : We Skolemise Γ into a set Δ of formulas without quantifiers. Assume Γ has a model. Then $\forall(\Delta)$ has a model according to corollary 5.2.16. According to theorem 5.1.16 on page 89, $\forall(\Delta)$ has a Herbrand model over the signature of Δ. Since the set Γ is countable, its signature is too, so Δ's signature too and consequently the domain of this Herbrand model is countable. □

Remark 5.2.18 *Let's call* theory*, a set of formulas. After the theorem, every countable theory which has a model, has a countable model. In other words with a first-order theory, we can talk about the properties of real numbers or sets, but we cannot force our theory to have only uncountable models, though the set of all real numbers or the class of all sets are uncountable.*

5.2.3 Clausal form

After the Skolemization of formulas and before being able to apply a generalization of the resolution method to first-order, we need to turn the Skolem forms into clauses as we did in propositional logic. We therefore extend the concepts key to this transformation before giving the rules allowing this transformation.

Definition 5.2.19 (Positive, negative literal and clause) *A* positive literal *is an atomic formula. A* negative literal *is the negation of an atomic formula.*

Notice that every literal is either positive or negative, and remind that a clause is a sum of literals.

Definition 5.2.20 (Clausal form of a formula) *Let A be a closed formula, the* clausal form *of A is a set of clauses obtained in two steps:*

1. *Skolemizing of A, i.e. building B the Skolem form of A.*
2. *Replacing B with an equivalent set* Γ *of clauses obtained by distributing the sum over the product.*

Property 5.2.21 *The universal closure of the clausal form of a closed formula has a model if and only the formula has a model. More specifically:*
— *any formula is a consequence of the universal closure of its clausal form,*
— *if the formula has a model then the universal closure of its clausal form has a model too.*

Proof : Let A be a closed formula, B its Skolem form and Γ its clausal form. According to the properties of Skolemization:
— A is a consequence of $\forall(B)$.
— If A has a model then $\forall(B)$ has a model.
Since Γ is obtained by distributivity, B and Γ are equivalent so $\forall(B)$ and $\forall(\Gamma)$ are also equivalent. As a result in both properties above, we can replace $\forall(B)$ with $\forall(\Gamma)$. □

Definition 5.2.22 (Clausal form of a set of formulas) *Let* Γ *be a set of closed formulas. We define the* clausal form *of* Γ *as the union of the clausal forms of each formula in* Γ*, taking care during Skolemization to eliminate each occurrence of an existential quantifier using a new symbol.*

Corollary 5.2.23 *Let* Γ *be a set of closed formulas and* Δ *the clausal form of* Γ*. We have:*
— Γ *is a consequence of* $\forall(\Delta)$*, and*
— *if* Γ *has a model then* $\forall(\Delta)$ *has a model.*

Theorem 5.2.24 (Adaptation of Herbrand's theorem to clausal forms) *Let* Γ *be a set of closed formulas and* Δ *the clausal form of* Γ*. The set* Γ *is unsatisfiable if and only if there is an unsatisfiable finite subset of instances of clauses of* Δ *over the signature of* Δ*.*

Proof : According to 5.2.23, Skolemization preserved satisfiability, so: Γ is unsatisfiable if and only if $\forall(\Delta)$ is unsatisfiable. According to the corollary of Herbrand's theorem 5.1.18 on page 89, $\forall(\Delta)$ is unsatisfiable if and only if there is a unsatisfiable finite subset of instances of clauses of Δ over the signature of Δ. □

Example 5.2.25 *Let* $A = \exists y \forall z (P(z,y) \Leftrightarrow \neg \exists x (P(z,x) \wedge P(x,z)))$*. Let's compute the clausal form of A.*

1. *Put A in normal form:*

$$\exists y \forall z ((\neg P(z,y) \vee \forall x (\neg P(z,x) \vee \neg P(x,z))) \wedge \exists x (P(z,x) \wedge P(x,z)) \vee P(z,y)).$$

2. *Make the result proper:*

$$\exists y \forall z ((\neg P(z,y) \vee \forall x (\neg P(z,x) \vee \neg P(x,z))) \wedge \exists u (P(z,u) \wedge P(u,z)) \vee P(z,y)).$$

3. *Eliminate existential quantifiers:*

$$\forall z ((\neg P(z,a) \vee \forall x (\neg P(z,x) \vee \neg P(x,z))) \wedge (P(z,f(z)) \wedge P(f(z),z)) \vee P(z,a)).$$

4. *Remove universal quantifiers, we get the Skolem form of A:*

$$((\neg P(z,a) \vee (\neg P(z,x) \vee \neg P(x,z))) \wedge (P(z,f(z)) \wedge P(f(z),z)) \vee P(z,a)).$$

5. *Transform into a product of sums of literals, we get the clausal form of A, which is the following set of clauses:*

According to the property above, A has no model if and only if there is an unsatisfiable finite set of instances of C_1, C_2, C_3 over the signature of these clauses. We are looking for these instances:

5.3 Unification

After putting a set of first-order logical formulas in clausal form and before generalizing resolution for these clauses, we need an algorithm to solve the equality between two terms. Given two terms, to find a substitution which, applied to these two terms, makes them equal is called the *unification problem* for these two terms. This problem has been studied by great names of computer science like Alan Robinson [23], Gerard Huet [15, 16], Alberto Martelli and Ugo Montanari [21], or more recently Franz Baader and Ayne Snyder [3].

5.3.1 Unifier

Definition 5.3.1 (Expression, solution) *Let us call* expression*, a term or a literal. A substitution σ (see definition 5.1.3 on page 88) is* a solution of the equation *$e_1 = e_2$ between two expressions, if the two expressions $e_1\sigma$ and $e_2\sigma$ are* identical*. A substitution is a solution of a set of equations if it is a solution of each equation in set.*

Definition 5.3.2 (Unifier) *Let σ be a substitution and E a set of expressions. $E\sigma = \{t\sigma \mid t \in E\}$. The substitution σ is a unifier of E if and only if the set $E\sigma$ contains only one element. Let $\{e_i | 1 \leq i \leq n\}$ be a finite set of expressions. The substitution σ is a unifier of this set if and only if it is a solution of the system of equations $\{e_i = e_{i+1} | 1 \leq i < n\}$.*

Definition 5.3.3 (Domain of a substitution) *The* domain *of a substitution σ is the set of variables x such that $x\sigma \neq x$. In the following, we are interested only in finite domain substitutions, that is to say that only change a finite number of variables. A finite support substitution σ is noted $< x_1 := t_1, \ldots, x_n := t_n >$ or more simply $x_1 := t_1, \ldots, x_n := t_n$ when there is no risk of ambiguity. The variables $x_1, \ldots, x_n$ are distinct and the substitution verifies:*

— *For i from 1 to n, $x_i\sigma = t_i$.*
— *For any variable y such that $y \notin \{x_1, \ldots, x_n\}$, we have: $y\sigma = y$.*

Example 5.3.4 *The equation $P(x, f(y)) = P(g(z), z)$ has a solution:*

The system of equations $x = g(z), f(y) = z$ has a solution:

Definition 5.3.5 (Composition of substitutions) *Let σ and τ be two substitutions, we note $\sigma\tau$ the substitution such that for any variable x, $x\sigma\tau = (x\sigma)\tau$. The substitution $\sigma\tau$ is an instance of σ. Two substitutions are equivalent if each of them is an instance of the other.*

Example 5.3.6 *Consider the following substitutions:*

— $\sigma_1 =< x := g(z), y := z >$.

— $\sigma_2 =< x := g(y), z := y >$.

— $\sigma_3 =< x := g(a), y := a, z := a >$.

We have the following relationships between these substitutions:

Definition 5.3.7 (Most general solution) *A solution of a system of equations is called* the most general *if any other solution is an instance of it. Notice that two "most general" solutions are equivalent.*

Example 5.3.8 *The equation $f(x, g(z)) = f(g(y), x)$ has solutions (among others):*

Definition 5.3.9 (Most general unifier) *Let E be a set of expressions. We recall that an expression is a term or literal. A unifier of E (see definition 5.3.2 on the preceding page) is called* the most general*, if any other unifier is an instance of it.*

Remark 5.3.10 (Most general unifier and most general solution) *Let $E = \{e_i \mid 1 \leq i \leq n\}$ be a set of expressions. In the definition of a unifier, we have indicated that σ is a unifier of E if and only if σ is a solution of the system $S = \{e_i = e_{i+1} \mid 1 \leq i < n\}$.*
Hence, the most general unifier of E is the most general solution of S.

5.3.2 Unification algorithm

The algorithm computing the solution of a system of equations is called *unification algorithm* because the search for a unifier of a set of expressions can be reduced to the search for a solution of a system of equations. The algorithm distinguishes equations to be solved, noted by an equality, from solved equations, noted by the symbol :=. Initially, there are no solved equations. The algorithm applies the the rules stated below. It stops when there are no more equations to be solved or when it declares that the system to be solved has no solution. When it stops without having declared the absence of solution, the list of solved equations is the most general solution of the initial system of equations.

5.3.2.1 The rules of the algorithm

Our unification algorithm consists in applying the six following rules:

1. **Deletion.** If both members of an equation are identical, the equation is deleted[3].
2. **Decomposition.** If both members of an equation are distinct and start with the same symbol then:
   — An equation of the form $\neg A = \neg B$ is replaced with $A = B$.

[3] We can limit ourselves to the removal of equations of the form $x = x$ where x is a variable.

— An equation of the form $f(s_1,\ldots,s_n) = f(t_1,\ldots,t_n)$ is replaced with the equations $s_1 = t_1,\ldots,s_n = t_n$. For $n = 0$, this decomposition removes the equation.

3. **Decomposition failure.** If an equation to be solved is of the form $f(s_1,\ldots,s_n) = g(t_1,\ldots,t_p)$ with $f \neq g$ or $n \neq p$ then the algorithm declares there is no solution. In particular there is obviously a failure if we are trying to solve an equation between a positive and a negative literal or if we have a constant and a function.
4. **Orientation.** If an equation is of the form $t = x$ where t is a term that is not a variable and x a variable, then we replace the equation with $x = t$.
5. **Elimination of a variable** : If an equation to be solved is of the form $x = t$ where x is a variable and t a term *not containing* x then:
   (a) Remove $x = t$ from the equations to be solved.
   (b) Replace x with t in every equation (to be solved *and solved*) [4].
   (c) Add $x := t$ to the solved part.
6. **Elimination failure**: If an equation to be solved is of the form $x = t$ where x is a variable and t a term distinct from x and *containing* x then the algorithm declares that there is no solution.

Example 5.3.11 *We apply this algorithm to solve the following equations:*

1. $f(x,g(z)) = f(g(y),x)$.

2. $f(x,x,a) = f(g(y),g(a),y)$.

3. $f(x,x,x) = f(g(y),g(a),y)$.

We first show that our algorithm is correct, that is to say, that it does compute a unifier. Then we prove that it always terminates.

[4] An effective algorithm avoids this systematic replacement.

5.3.2.2 Correctness of the algorithm

We can easily observe the following three facts:

— (1) : the rules of deletion, decomposition, and orientation and elimination of variables preserve the set of solutions.

— (2) : initially and after any rule application, the set of solved equations constitutes a substitution $x_1 := t_1, \ldots, x_n := t_n$ where $x_1, \ldots, x_n$ are distinct variables not appearing in any of the terms $t_1, \ldots, t_n$. Let σ be this substitution. It is clear that $t_i\sigma = t_i$ hence σ is a solution of the solved equations.

— (3): The rules of failure correctly declare the absence of solution.

From (1) and (2), it results that if the algorithm ends without failure, the substitution σ it returns is the most general solution of the *initial system*. Indeed:

— σ is a solution, since, according to (2), it is a solution of the *final system* and, according to (1), the set of solutions is preserved by the rules.

— σ is a most general solution since, according to (1), any other solution is a solution of the final system, hence an instance of the substitution defined by this system.

From (1) and (3), it results that if the algorithm declares a failure, then the initial system has no solution. Indeed, in this case:

— If the initial system had a solution, according to (1), it would be a solution of the final system,

— According to (3), the final system has no solution.

5.3.2.3 Termination of the algorithm

— The deletion and decomposition rules strictly decrease the sum of the lengths of the terms which appear in both members of the equations.

— The orientation rule strictly decreases the sum of the lengths of the terms which appear on the left of the equations.

— The elimination rule strictly decreases the number of variables in the equations to be solved.

It follows that the unification algorithm terminates.

5.4 First-order resolution

We now have all the ingredients to present first-order resolution. Of course we transform a set of formulas in clausal form. Then we can apply three rules enabling to "factorize" terms, to "copy" a clause and finally to apply a "binary resolution". These different rules use the unification algorithm presented above. In the following, we prove the consistency and the completeness of this formal system. The completeness is based on Herbrand's theorem in the version we gave in corollary 5.2.24 on page 95.

5.4.1 Three rules for resolution

Let Γ be a set of clauses, we assume that $\forall(\Gamma)$ has no model, then we give a formal system allowing to deduce $\bot$ from Γ with the following three rules:

1. *Factoring* which, from the premise $P(x, f(y)) \vee P(g(z), z) \vee Q(z, x)$, deduces $P(g(f(y)), f(y)) \vee Q(f(y), g(f(y)))$. The deduced clause is obtained by computing the most general solution $x := g(f(y)), z := f(y)$ of $P(x, f(y)) = P(g(z), z)$.
2. The *copy* rule which allows to rename the variables in a clause.
3. The *binary resolution* (BR) which, from the two premises without a common variable $P(x, a) \vee Q(x)$ and $\neg P(b, y) \vee R(f(y))$, deduces the resolvent $Q(b) \vee R(f(a))$, by computing the most general solution $x := b, y := a$ of $P(x, a) = P(b, y)$.

To simplify the presentation of the rules, we will identify a clause, which is a sum of literals, with the set of its literals. We formalize these three rules.

5.4.1.1 Factoring

Property 5.4.1 *Let A be a formula without quantifier and B an instance of A. We have:* $\forall(A) \models \forall(B)$.

Definition 5.4.2 *The clause C' is a* factor *of the clause C if $C' = C$ or if there is a subset E of C such that E has at least two elements, E is unifiable and $C' = C\sigma$ where σ is the most general unifier of E.*

Example 5.4.3 *The clause $\underline{P(x)} \vee Q(g(x,y)) \vee \underline{P(f(a))}$ has two factors:*

We immediately obtain the following property.

Property 5.4.4 (Consistency of the factoring rule) *Let C' be a factor of the clause C. We have: $\forall(C) \models \forall(C')$.*

Proof : Since C' is an instance of C, this property is an immediate consequence of property 5.4.1 on the previous page. □

5.4.1.2 Copy of a clause

Definition 5.4.5 *Let C be a clause and σ a substitution, which only changes the variables of C and whose* restriction *to the variables of C is a* bijection *between these variables and those of the clause $C\sigma$. The clause $C\sigma$ is a copy of the clause C. The substitution σ is also called a* renaming *of C.*

Definition 5.4.6 *Let C be a clause and σ a renaming of C. Let f be the restriction of σ to the variables of C and f^{-1} the inverse function of f. Let σ_C^{-1} be the substitution defined as follows for any variable x:*

- *If x is a variable of $C\sigma$ then $x\sigma_C^{-1} = xf^{-1}$ (for clarity concerns, we preferred to use the postfix notation xf^{-1} rather than the prefix notation $f^{-1}(x)$, more common).*
- *Otherwise $x\sigma_C^{-1} = x$.*

This substitution is called the inverse of the renaming σ *of C.*

Example 5.4.7 *Let $\sigma = <x := u, y := v>$. σ is a renaming of the literal $P(x,y)$. The literal $P(u,v)$, where $P(u,v) = P(x,y)\sigma$, is a copy of $P(x,y)$. Let $\tau = <u := x, v := y>$. τ is the inverse of the renaming σ of $P(x,y)$. Notice that $P(u,v)\tau = P(x,y)$: the literal $P(x,y)$ is a copy of $P(u,v)$ through the renaming tau.*

Property 5.4.8 *Let C be a clause and σ a renaming of C.*

1. *σ_C^{-1} is a renaming of $C\sigma$.*
2. *For any expression or clause E, whose variables are those of C, $E\sigma\sigma_C^{-1} = E$.*

Thus $C\sigma\sigma_C^{-1} = C$ and hence C is a copy of $C\sigma$.

Proof : Let f be the restriction of σ to the variables of C. By definition of renaming, f is a bijection between the variables of C and those of $C\sigma$.

1. By definition of σ_C^{-1}, this substitution only changes the variables of $C\sigma$ and its restriction to the variables of $C\sigma$ is the bijection f^{-1}. Thus, σ_C^{-1} is a renaming of $C\sigma$.
2. Let x be a variable of C. By definition of f, $x\sigma\sigma_C^{-1} = xff^{-1} = x$. Thus, by an induction over terms, literals and clauses we omit here, for any expression or clause E whose variables are those of C, we have $E\sigma\sigma_C^{-1} = E$.

□

Property 5.4.9 *If two clauses are copies of each other, their universal closures are equivalent.*

Proof : Let C' be a copy of C. By definition, C' is an instance of C and by the previous property, C is a copy of C', hence an instance of C'. Thus by property 5.4.1 on the preceding page, the universal closure of C is a consequence of that of C' and conversely. As a result, these two universal closures are equivalent. □

5.4.1.3 Binary resolution

Definition 5.4.10 *Let C and D be two clauses* without a common variable. *The clause E is a* binary resolvent *of C and D if there is a literal $L \in C$ and a literal $M \in D$ such that L and M^c (the complementary literal of M) are unifiable and if $E = ((C - \{L\}) \cup (D - \{M\}))\sigma$ where σ is the the most general solution of the equation $L = M^c$.*

Example 5.4.11 *Let $C = P(x,y) \vee P(y,k(z))$ and $D = \neg P(a, f(a,y_1))$.*

Property 5.4.12 (Consistency of the binary resolution rule) *Let E be a binary resolvent of clauses C and D, we have: $\forall(C), \forall(D) \models \forall(E)$.*

Proof : Let I be an interpretation. Since the three formulas $\forall(C), \forall(D), \forall(E)$ are without free variable, it is sufficient to show that if I is a model of $\forall(C)$ and $\forall(D)$ then I is a model of $\forall(E)$. Assume I is a model of $\forall(C), \forall(D)$ and let us prove that it is a model of $\forall(E)$. Let e be *any* state of this interpretation. (I,e) is a model of $C\sigma$ and $D\sigma$, for any substitution σ. By definition of the binary resolvent, there is a literal $L \in C$ and a literal $M \in D$ such that L and M^c (the complementary literal of M) are unifiable. Thus, $L\sigma$ and $M\sigma$ are two complementary literals, so (I,e) is a model of either of these two literals.

1. Assume (I,e) is a model of $L\sigma$. Since (I,e) is a counter-model of $M\sigma$ and a model of $D\sigma$, it's a model of $(D - \{M\})\sigma$, hence a model of E because $(D - \{M\})\sigma \subset E$.
2. Assume (I,e) is a counter-model of $L\sigma$. Since it is a model of $C\sigma$, it is a model of $(C - \{L\})\sigma$ hence a model of E because $(C - \{L\})\sigma \subset E$.

Thus (I,e) is a model of E. Since e can be any state, I is a model of $\forall(E)$. □

5.4.1.4 Proof by factoring, copy and binary resolution

Definition 5.4.13 *Let Γ be a set of clauses and C a clause. A* proof *of C from Γ is a sequence of clauses ending with C, where every clause in the proof is an element of Γ, a factor of a previous clause in the proof, a copy of a previous clause in the proof or a binary resolvent of two previous clauses in the proof.*

The fact that C is deduced from Γ at the first order by the 3 rules of factoring, copy and binary resolution is noted $\Gamma \vdash_{1fcb} C$. That means there is a proof of C starting from Γ. When there is no ambiguity on the formal system we used, we replace $\vdash_{1fcb}$ with $\vdash$.

5.4.2 Consistency of resolution

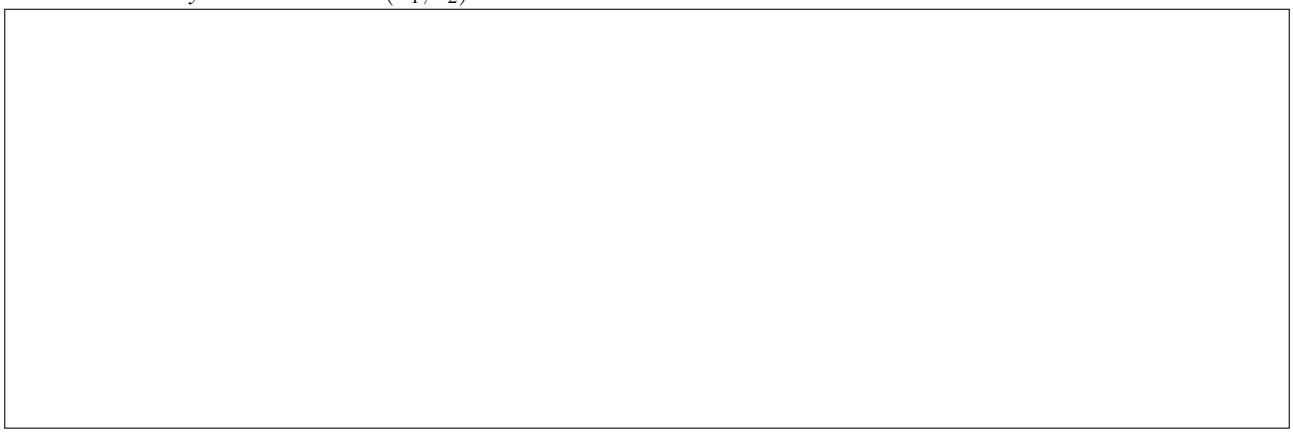
Property 5.4.14 (Consistency) *Let Γ be a set of clauses and C a clause. If $\Gamma \vdash_{1fcb} C$ then $\forall(\Gamma) \models \forall(C)$.*

Proof : This property is an immediate consequence of the consistency of factoring, copy and binary resolution. This proof is requested in exercise 97 on page 108. □

Example 5.4.15 *Let the two clauses:*

1. $C_1 = P(x,y) \vee P(y,x)$.
2. $C_2 = \neg P(u,z) \vee \neg P(z,u)$.

Let us show by resolution that $\forall(C_1, C_2)$ has no model.

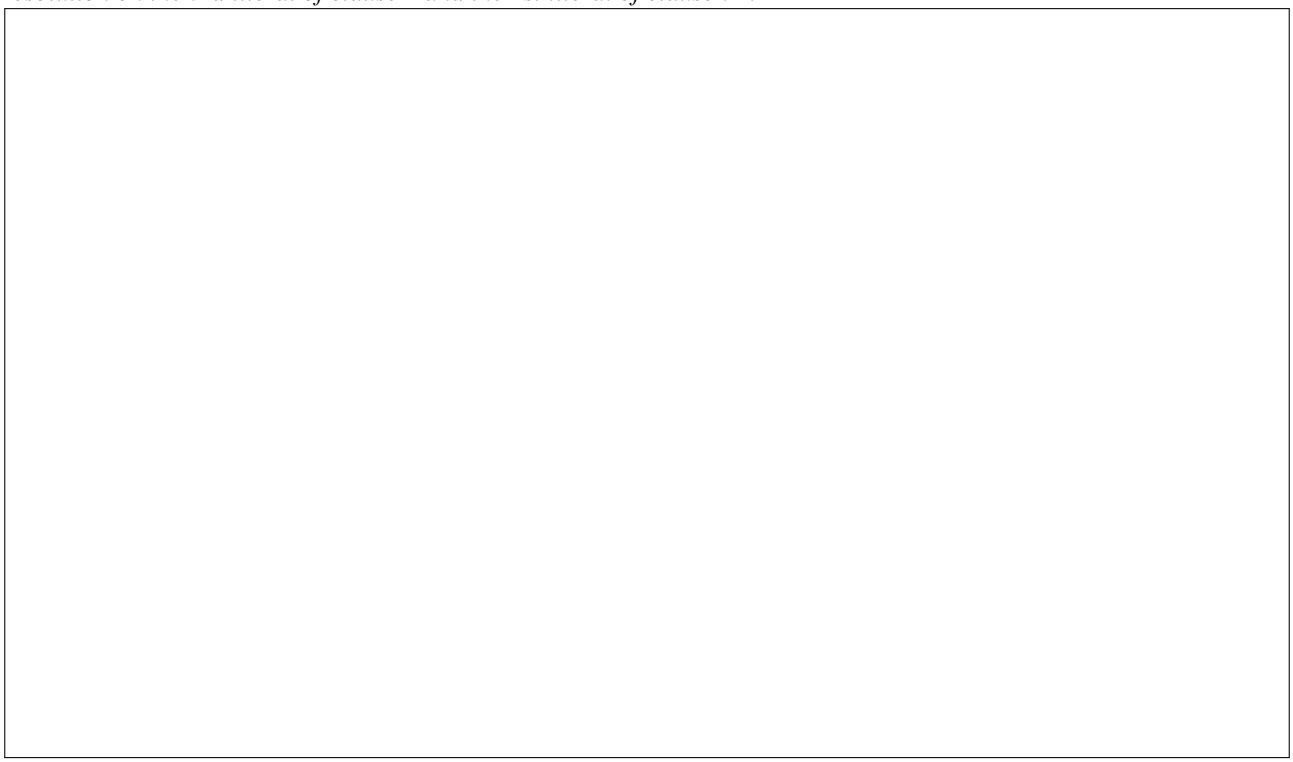


This example shows, a contrario*, that binary resolution alone is incomplete, without factoring, we cannot deduce the empty clause.*

Example 5.4.16 *Let the three clauses:*

1. $C_1 = \neg P(z,a) \vee \neg P(z,x) \vee \neg P(x,z)$.
2. $C_2 = P(z, f(z)) \vee P(z,a)$.
3. $C_3 = P(f(z),z) \vee P(z,a)$.

We provide a proof by resolution that $\forall(C_1, C_2, C_3)$ has no model. In this proof BR 1(3), 3(1) means "by binary resolution on the 3rd literal of clause 1 and the 1st literal of clause 3".



5.4.3 Completeness of resolution

We define a *new* rule, first-order resolution, which is a combination of the three rules of factoring, copy and binary resolution. This will allow us to prove the completeness of resolution.

Definition 5.4.17 *The clause E is a first-order resolvent of the clauses C and D if E is a binary resolvent of C' and D' where C' is a factor of C and D' is a copy of a factor of D without common variable with C'. The rule which, from C and D, deduces E is called first-order resolution.*

Example 5.4.18 *Let $C = \neg P(z,a) \vee \neg P(z,x) \vee \neg P(x,z)$ and $D = P(z,f(z)) \vee P(z,a)$.*

$C' = \neg P(a,a)$ is a factor of C. The clause $P(a,f(a))$ is a binary resolvent of C' and D (which is a factor of itself) hence

Let Γ be a set of clauses and C a clause, until now we have defined three notions of proof by resolution that we denote by:

1. $\Gamma \vdash_p C$ the fact that there is a proof of C starting from Γ obtained by propositional resolution, in other words without substitution.
2. $\Gamma \vdash_{1fcb} C$ the fact that there is a proof of C starting from Γ by factoring, copy and binary resolution.
3. $\Gamma \vdash_{1r} C$ the fact that there is a proof of C starting from Γ obtained by first-order resolution.

Since first-order resolution is a combination of the factoring, copy and binary resolution rules, we have immediately $\Gamma \vdash_{1r} C$ implies $\Gamma \vdash_{1fcb} C$.

Theorem 5.4.19 (Lifting theorem) *Let C and D be two clauses. Let C' be an instance of C and D' an instance of D. Let E' be a* propositional *resolvent of C' and D', there is a* first-order *resolvent E of C and D of which E' is an instance.*

Example 5.4.20 *Let $C = P(x) \vee P(y) \vee R(y)$ and $D = \neg Q(x) \vee P(x) \vee \neg R(x) \vee P(y)$.*
- *The clauses $C' = P(a) \vee R(a)$ and $D' = \neg Q(a) \vee P(a) \vee \neg R(a)$ are instances of C and D respectively.*
- *The clause $E' = P(a) \vee \neg Q(a)$ is a propositional resolvent of C' and D'.*
- *The clause $E = P(x) \vee \neg Q(x)$ is a first-order resolvent of C and D of which E' is an instance.*

Theorem 5.4.21 *Let Γ be a set of clauses and Δ a set of instances of clauses of Γ, and $C_1, \ldots, C_n$ a proof by propositional resolution starting from Δ, there is a proof $D_1, \ldots, D_n$ by first-order resolution starting from Γ such that for i from 1 to n, the clause C_i is an instance of D_i.*

Proof : We perform an induction on n. Let C_1, C_n, C_{n+1} be a proof by propositional resolution starting from Δ. By induction, there is a proof $D_1, \ldots, D_n$ by first-order resolution starting from Γ such that for i from 1 to n, the clause C_i is an instance of D_i. We distinguish two cases:

1. Assume $C_{n+1} \in \Delta$. There is $E \in \Gamma$ of which C_{n+1} is an instance so we take $D_{n+1} = E$.
2. Assume C_{n+1} is a propositional resolvent of C_j and C_k where $j,k \leq n$. According to the previous result, there is a first-order resolvent E of D_j and D_k: we take $D_{n+1} = E$.

□

We immediately deduce the following corollary.

Corollary 5.4.22 *Let Γ be a set of clauses and Δ a set of instances of clauses in Γ. Assume $\Delta \vdash_p C$. There is a D such that $\Gamma \vdash_{1r} D$ and C is an instance of D .*

Example 5.4.23 *Let the set of clauses $P(f(x)) \vee P(u), \neg P(x) \vee Q(z), \neg Q(x) \vee \neg Q(y)$. The universal closure of this set of clauses is unsatisfiable and we prove it in three manners.*

1. *By instantiation over the Herbrand universe $\{f^n(a), n \in \mathbb{N}\}$:*
   - *$P(f(x)) \vee P(u)$ is instantiated by $x := a, u := f(a)$ into $P(f(a))$*
   - *$\neg P(x) \vee Q(z)$ is instantiated by $x := f(a), z := a$ into $\neg P(f(a)) \vee Q(a)$*
   - *$\neg Q(x) \vee \neg Q(y)$ is instantiated by $x := a, y := a$ into $\neg Q(a)$*

   The set of these 3 instantiations if unsatisfiable, as shows the proof by propositional resolution below:

$$\dfrac{\dfrac{P(f(a)) \quad \neg P(f(a)) \vee Q(a)}{Q(a)} \qquad \neg Q(a)}{\bot}$$

2. *This proof by propositional resolution is lifted into a proof by the rule of first-order resolution:*

$$\dfrac{\dfrac{P(f(x)) \vee P(u) \quad \neg P(x) \vee Q(z)}{Q(z)} \qquad \neg Q(x) \vee \neg Q(y)}{\bot}$$

3. *Each rule of first-order resolution is decomposed into factoring, copy and binary resolution:*

$$\dfrac{\dfrac{\dfrac{P(f(x)) \vee P(u)}{P(f(x))}\ Fact \quad \dfrac{\neg P(x) \vee Q(z)}{\neg P(y) \vee Q(z)}\ Copie}{Q(z)}\ RB \qquad \dfrac{\neg Q(x) \vee \neg Q(y)}{\neg Q(x)}\ Fact}{\bot}\ RB$$

Theorem 5.4.24 (Refutational completeness of first-order resolution) *Let Γ be a set of clauses. The following propositions are equivalent:*

1. $\Gamma \vdash_{1r} \bot$.
2. $\Gamma \vdash_{1fcb} \bot$.
3. $\forall(\Gamma) \models \bot$.

Proof :

— We know that (1) implies (2), because first-order resolution is a combination of factoring, copy and binary resolution.

— We know that (2) implies (3), because factoring, copy and binary resolution are consistent.

— We still have to prove that (3) implies (1). Assume that $\forall(\Gamma) \models \bot$, i.e. $\forall(\Gamma)$ is unsatisfiable. According to Herbrand's theorem, there is Δ a finite set of instances without variable of clauses of Γ that has no propositional model. By completeness of propositional resolution, we have: $\Delta \vdash_p \bot$. According to the corollary to lifting 5.4.22 on the preceding page, there is a D such that $\Gamma \vdash_{1r} D$ and $\bot$ is an instance of D. In this case, we have $D = \bot$.

□

5.5 Software tool

We mention here a software to experiment with the deductive system described in this chapter:

`http://teachinglogic.univ-grenoble-alpes.fr/ResBinSc/`

This tool, in a similar way to the one proposed for propositional natural deduction in paragraph 3.5 on page 59, allows to type a list of first-order clauses. Then it automatically tries to deduce the empty clause:

— if this list of clauses is unsatisfiable and no time limit nor size limit has been set, then the empty clause will be deduced;

— it is possible to restrict the search space, by imposing a time or size limit on the clauses produced (but then we lose the prover's completeness).

If the list of clauses is unsatisfiable, we get a proof using copy, factoring and binary resolution rules, annotated by the unifiers used.

5.6 Exercises

Exercise 80 (Herbrand Universe) *Let Σ be the signature with constant a and function symbols f and g of arity one and two respectively.*

— *Give 5 distinct elements of the Herbrand universe of this signature.*
— *Give an inductive definition of this Herbrand universe.*

□

Exercise 81 (Signature, Herbrand Universe and Base) *For each of the following sets of formulas,*

— $\Gamma_1 = \{P(x) \vee Q(x) \vee R(x), \neg P(a), \neg Q(b), \neg R(c)\}$.
— $\Gamma_2 = \{P(x), \neg Q(x), \neg P(f(x)) \vee Q(f(x))\}$.
— $\Gamma_3 = \{P(x), Q(f(x)), \neg R(f(f(x))), \neg P(f(f(x))) \vee \neg Q(x) \vee R(f(x))\}$.

1. *Give the signature, and the corresponding Herbrand universe and base.*
2. *Prove whether or not their universal closure has a model.*

□

Exercise 82 (Herbrand's method) *Use Herbrand's method in order to prove that the following set of formulae is unsatisfiable:*

1. $\forall x R(x, f(x))$
2. $\forall x \forall y (\overline{S(x,y)} \vee R(x,y) \vee R(y,x))$
3. $\forall x \forall y (S(x,y) \vee \overline{R(x,y)})$
4. $\forall x \forall y (S(x,y) \vee \overline{R(y,x)})$
5. $\forall y \overline{S(y,a)}$

□

Exercise 83 (Herbrand's method,*) *Use Herbrand's method in order to prove that the following set of formulae is unsatisfiable:*

1. $\forall x (Q(x) \vee \neg P(f(x)))$
2. $\forall y (Q(y) \Rightarrow R(y))$
3. $\forall z (\neg P(z) \Rightarrow Q(z) \vee R(z))$
4. $\forall u \neg R(u)$

In particular, you will have to transform your set of unsatisfiable instances into an equivalent set of clauses. Then, you will prove that it is contradictory with a proof in propositional resolution. □

Exercise 84 (Herbrand's method,*) *Let Γ be the following set of formulae:*

1. $\overline{S(x,y)} + \overline{M(z,x)} + M(z,y)$
2. $S(x,y) + M(f(x,y),x)$
3. $S(x,y) + \overline{M(f(x,y),y)}$
4. $S(c,a)$
5. $S(a,b)$
6. $\overline{S(c,b)}$

Find a finite unsatisfiable set of closed instances of these formulae.
This enables us to deduce something about a set of first-order formulae : which set and what ? □

Exercise 85 (Herbrand Model,*)** *Let Δ be the following set of formulas:*

1. $x < y \wedge y < z \Rightarrow x < z$.
2. $\neg(x < x)$.

3. $x < y \Rightarrow x < f(x,y) \wedge f(x,y) < y$.
4. $a < b$.

— *Give a model of* $\forall(\Delta)$.
— *Does* $\forall(\Delta)$ *have a model on the Herbrand universe created from* a, b, f *?*

□

Exercise 86 (Skolemization) *Skolemize the following formulas (pay attention to the negations !) then transform them in clausal form.*

1. $\neg(\exists x P(x) \vee \exists x Q(x) \Rightarrow \exists x(P(x) \vee Q(x)))$.
2. $\neg(\forall x \forall y \forall z(e(x,y) \wedge e(y,z) \Rightarrow \neg e(x,z)) \Rightarrow \neg \exists x \forall y\, e(x,y))$.
3. $\neg(\neg \forall x P(x) \vee \neg \forall x Q(x) \Rightarrow \neg(\forall x P(x) \wedge \forall x Q(x)))$.
4. $\forall x((\exists y P(x,y) \Rightarrow \exists x Q(x)) \wedge \exists y P(x,y) \wedge \neg \exists x Q(x))$.
5. $\neg(\exists x \forall y \forall z((P(y) \Rightarrow Q(z)) \Rightarrow (P(x) \Rightarrow Q(x))))$

□

Exercise 87 (Unification) *Are the following terms unifiable? If so, give their most general unifier, if not, justify your answer.*

— $h(g(x), f(a,y), z)$ *et* $h(y, z, f(u,x))$.
— $h(g(x), f(a,y), z)$ *et* $h(y, z, f(u, g(x)))$.

□

Exercise 88 (Unification) *Give the most general unifier for each of the following terms if it exists.*

1. $R(a, f(x)) = R(y, f(g(y,b)))$,
2. $R(y, f(x)) = R(x, f(g(y,b)))$,
3. $Q(y, f(a), f(x)) = Q(g(a,b), x, f(y))$, *et*
4. $Q(y, f(x), g(y,x)) = Q(x, f(y), g(f(a), y))$.

□

Exercise 89 (Unification) *Give the most general unifier for each of the following terms if it exists.*

1. $pair(a, crypt(z,b))$ *et* $pair(x,y)$.
2. $pair(crypt(x,b), crypt(y,b))$ *et* $pair(crypt(a,b), z)$.
3. $crypt(pair(z,a), x)$ *et* $crypt(pair(y, crypt(x,b)), b)$.
4. $crypt(pair(a,z), x)$ *et* $crypt(pair(y, crypt(x,b)), b)$.
5. $f(x, y, g(a,a))$ *et* $f(g(y,y), z, z)$
6. $f(x,y,a)$ *et* $f(y, g(z,z), x)$

□

Exercise 90 (Unification with multiple solutions) *The equation* $f(g(y), y) = f(u,z)$ *has two most general unifiers (Recall: they are therefore equivalent). Give these two solutions.* □

Exercise 91 (Proof by resolution,*) *Show that the following formula is valid by transforming its* negation *in clausal form and by finding a contradictory set of instances of the clauses.*

$$\forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\forall x P(x) \Rightarrow \forall x Q(x)).$$

□

Exercise 92 (Proof by resolution,)** *Consider the following formulas:*

1. $H_1 = \exists x P(x) \Rightarrow \forall x P(x)$.
2. $H_2 = \forall x(P(x) \vee Q(x))$.

3. $C = \exists x \neg Q(x) \Rightarrow \forall x P(x)$.

We want to show that C is the consequence of H_1 and H_2 using instantiation and resolution.

1. *Transform the three formulas H_1, H_2, $\neg C$ in clausal form.*
2. *Find contradictory instances of the clauses obtained and show by propositional resolution that these instances are contradictory.*
3. *Give a direct proof of this contradiction using factorization, copy and binary resolution. It is possible that you only need the last rule.*

□

Exercise 93 (Proof by resolution,)**

— *Using a proof by factorization, copy and binary resolution, prove that the universal closure of the following set of clauses is unsatisfiable:*

1. $P(f(x)) \vee \neg Q(y,a)$.
2. $Q(a,a) \vee R(x,x,b) \vee S(a,b)$.
3. $S(a,z) \vee \neg R(x,x,b)$.
4. $\neg P(f(c)) \vee R(x,a,b)$.
5. $\neg S(y,z) \vee \neg S(a,b)$.

— *Using a proof by factorization, copy and binary resolution, prove that the universal closure of the following set of clauses is unsatisfiable:*

1. $P(x)$.
2. $\neg P(y) \vee Q(y,x)$.
3. $\neg Q(x,a) \vee \neg Q(b,y) \vee \neg Q(b,a) \vee \neg P(f(y))$.

— *Using a proof by factorization, copy and binary resolution, prove that the universal closure of the following set of clauses is unsatisfiable:*

1. $R(x,a)$.
2. $R(y,b)$.
3. $R(z,a+b)$.
4. $\neg(b < x+b) \vee (a+b < a)$.
5. $b < z$.
6. $\neg(a+b < x) \vee \neg R(y,b)$.

Recall : *The symbol $+$ has a higher priority than the symbol $<$, which itself has a higher priority than binary connectors.*

□

Exercise 94 (Unification, resolution) *Let $\Gamma_1 = \{P(x, f(x,b), u), \neg P(g(a), z, h(z))\}$ and $\Gamma_2 = \{P(x, f(x,b), u), \neg P(g(z), z, h(z))\}$ Are the sets $\forall(\Gamma_1)$ and $\forall(\Gamma_2)$ satisfiable or unsatisfiable?* □

Exercise 95 (Skolemization and First Order Resolution) *The goal of this exercise is to prove the following syllogism:*

$$\forall x(man(x) \Rightarrow mortal(x)) \wedge man(Socrates) \Rightarrow mortal(Socrates)$$

— *Skolemize the* **negation** *of the syllogism.*
— *Transform the obtained Skolem form in clausal form.*
— *Demonstrate by instantiation that the* ***negation*** *of the syllogism is a contradiction.*
— *Demonstrate using factorization, copy and binary resolution that the negation of the syllogism is a contradiction.*

□

Exercise 96 (Clausal Form and Resolution,)** *Consider the following formulas:*

1. $A_1 = \exists u \forall v(P(u) \wedge (R(v) \Rightarrow Q(u,v)))$.

2. $A_2 = \forall u \forall v(\neg P(u) \vee \neg S(v) \vee \neg Q(u,v))$.
3. $A_3 = \exists v(R(v) \wedge S(v))$.

Show that the set containing these three formulas is contradictory using resolution:

1. *Transorm A_1, A_2, A_3 in clausal form.*
2. *Find contradictory instances of the clauses you obtained and show this contradiction using propositional resolution.*
3. *Make a direct proof of $\bot$ using factorization, copy and binary resolution.*

□

Exercise 97 (Coherence of First Order Resolution) *Prove the coherence of first order resolution, that is:*

Let Γ be a set of clauses and C a clause. If $\Gamma \vdash_{1fcb} C$ then $\forall(\Gamma) \models \forall(C)$.

□

Exercise 98 (Clausal Form and Resolution,)** *Consider the following formulas:*

1. $E_1 = \forall x(x = x)$.
2. $E_2 = \forall x \forall y(x = y \Rightarrow y = x)$.
3. $E_3 = \forall x \forall y \forall z \forall t(x = y \wedge z = t \Rightarrow (x \in z \Rightarrow y \in t))$.
4. $C = \forall y \forall z(y = z \Rightarrow \forall x(x \in y \Leftrightarrow x \in z))$.

E_1, E_2, E_3 are axioms of equality: E_1 says equality is reflexive, E_2 says it is symmetric, E_3 says it is a congruence relative to $\in$. We do not use transitivity in this exercise. C says that two sets are equal if they have the same elements. We note that in this exercise, the symbol '=' is not treated as usual, as the symbol with fixed meaning describing the identity relation. The restriction on the appearance of the equality symbol in formulas applies only when it is considered as a symbol with fixed meaning. Show that C is a consequence of the other axioms on equality:

1. *Put the formulas E_1, E_2, E_3, $\neg C$ in clausal form.*
2. *Find contradictory instances of the clauses you obtained.*
3. *Show a direct proof of $\bot$ using factorization, copy and binary resolution.*

□

Exercise 99 (Clausal Form and Resolution,)** *Consider the following formulas:*

1. $H_1 = \forall u(\exists v R(u,v) \Rightarrow R(u, f(u)))$.
2. $H_2 = \forall u \exists v R(u,v)$.
3. $H_3 = \exists u R(f(f(u)), u)$.
4. $C = \exists u \exists v \exists w(R(u,v) \wedge R(v,w) \wedge R(w,u))$.

Show that C is the consequence of H_1, H_2, H_3.

1. *Transform the formulas H_1, H_2, H_3, $\neg C$ in clausal form.*
2. *Find contradictory instances of the clauses you obtained, and show the contradiction using propositional resolution.*
3. *Prove this contradiction directly using factorization, copy and binary resolution.*

□

Exercise 100 (Clausal Form and Resolution,)** *Let $F(x,y)$ be a formalization of " x is y's father ". Let $A(x,y)$ be a formalization of " x is y's ancestor ". Consider the following formulas:*

1. $H_1 = \forall x \exists y F(x,y)$.
2. $H_2 = \forall x \forall y \forall z(F(x,y) \wedge F(y,z) \Rightarrow A(x,z))$.
3. $C = \forall x \exists y A(x,y)$.

Show that C is the consequence of H_1, H_2 by transforming $H_1, H_2, \neg C$ into clausal form, and deriving the empty clause using factorization, copy and binary resolution. □

Exercise 101 (Bernays-Schonfinkel Formula,*)** *A BS (Bernays-Schonfinkel) formulais a closed formula* without *any function symbols, of the form $\exists x_1 \ldots \exists x_n \forall y_1 \ldots \forall y_p$ B, where B does not have any quantifiers. Show how to decide the satisfiability of such a formula.* □

Chapter 6

First-order natural deduction : quantifiers, copy and equality

Contents

In addition to the rules of chapter 3, we add the rules about quantifiers, copy and equality. Our system still has one rule for removing assumptions (the rule of introduction of implication). The definitions of proof draft, environment, context, usable formula remain unchanged. We prove the consistency of the rules of our system, but we admit without proof, that this system is complete: proofs of completeness for systems with similar rules can be found in books [2, 11]. Unlike in the propositional case, *there is no algorithm* for deciding whether a formula is valid or invalid (proof given by Alonzo Church and Alan Turing in 1936 and 1937 [4, 24]). In other words, admitting the equivalence between provable (without environment) and valid, there is no algorithm that, given a formula, can build a proof for it, or warn us that this formula has no proof.

Outline: First, we extend the rules of natural deduction introduced for propositional logic. Then we present two new tactics to help write proofs. Finally, we prove the consistency of our system.

6.1 Rules for first-order logic

Our rules system is divided into several sets of rules:

— the "propositional" rules,
— the introduction and elimination rules for the quantifiers,
— the rules governing equality,
— and a copy rule.

The set of rules defined in section 3 on page 51 can be used in our deduction system. We refer the reader to table 3.1 on page 53, which summarizes all these rules. In the remainder of this section, we only present the natural deduction rules specific to first-order logic.

6.1.1 Rules for the quantifiers

The introduction and elimination rules for the quantifiers are presented in table 6.1. In this table, A and B are formulas, x is a variable, t is a term. Notice that, unlike the "propositional" rules, the use of these new rules is subject to use conditions, which refer to the notions of *free variable* (Definition 4.2.3 on page 71) and *free term for a variable* (Definition 4.3.34 on page 76).

In the following, we detail these rules and illustrate their use on examples and counter-examples showing various errors resulting from non-compliance with the use conditions.

Rules	**Use conditions**
$\dfrac{\forall xA}{A<x:=t>}\ \forall E$	
$\dfrac{A}{\forall xA}\ \forall I$	
$\dfrac{\exists xA \quad (A \Rightarrow B)}{B}\ \exists E$	
$\dfrac{A<x:=t>}{\exists xA}\ \exists I$	

Table 6.1: Introduction and elimination rules for quantifiers.

6.1.1.1 Rules for the universal quantifier

The universal quantifier elimination rule, noted $\forall E$, means that if the formula A is true for every value of x then any instance of A where x is replaced with **a term** t **free for** x is true. Notice that this rule is a simple application of corollary 4.3.38 on page 77.

$$\frac{\forall xA}{A<x:=t>}\ \forall E$$

We justify below the use conditions of this rule with an example showing that an incorrect use of the rule may lead to prove an invalid formula.

Example 6.1.1 *We show that an incorrect use of the* ***rule*** $\forall E$ *leads to a "proof" of an invalid formula.*

context	*number*	*line*	*rule*
1	*1*	`Assume` $\forall x\exists yP(x,y)$	
1	*2*	$\exists yP(y,y)$	$\forall E$ *1, y ERROR*
	3	`Therefore` $\forall x\exists yP(x,y) \Rightarrow \exists yP(y,y)$	$\Rightarrow I$ *1,2*

On line 2, we did not comply with the use conditions of rule $\forall E$ *because the term y is not free for x in the formula* $\exists yP(x,y)$*. We give an interpretation which is a counter-model of the "conclusion": let I be the interpretation with domain* $\{0,1\}$ *and* $P_I =$

The introduction rule for the universal quantifier, noted $\forall I$, means that if we were able to deduce the formula A independently from the value of x, we can generalize by deducing $\forall xA$. Thus, it is necessary that x **is free neither in the proof environment, nor in the context of the line where we deduced** A.

$$\frac{A}{\forall xA}\ \forall I$$

We now illustrate the use of the introduction rule for the universal quantifier with an example of correct use (example 6.1.2) and an example of incorrect use (example 6.1.3) of the rule.

Example 6.1.2 *We prove* $\forall yP(y) \wedge \forall yQ(y) \Rightarrow \forall x(P(x) \wedge Q(x))$.

context	*number*	*line*	*rule*
1	*1*	`Assume` $\forall yP(y) \wedge \forall yQ(y)$	
1	*2*		
1	*3*		
1	*4*		
1	*5*		
1	*6*		
1	*7*		
	8	`Therefore` $\forall xP(x) \wedge \forall xQ(x) \Rightarrow \forall x(P(x) \wedge Q(x))$	$\Rightarrow$ *I 1,7*

Example 6.1.3 *We show that an incorrect use of* ***rule*** $\forall I$ *leads to a "proof" of an invalid formula.*

context	*number*	*line*	*rule*
1	*1*	`Assume` $P(x)$	
1	*2*	$\forall xP(x)$	$\forall I$ *1 ERROR*
	3	`Therefore` $P(x) \Rightarrow \forall xP(x)$	$\Rightarrow$ *I 1,2*

On line 2, we did not comply with the use conditions of rule $\forall I$ *because the premise* $P(x)$ *is established in context* $P(x)$*, which prohibits generalizing on x. We give a counter-model of the "conclusion":*

6.1.1.2 Rules for the existential quantifier

The elimination rule for the existential quantifier, noted $\exists E$, means that if $\exists xA$ is true and we can deduce $A \Rightarrow B$ independently of the value of x, then we can deduce B, if the formula B does not depend either on the value of x. Thus it is necessary that x **is free neither in the proof environment nor in** B**, nor in the context of the line** $A \Rightarrow B$**.**

$$\frac{\exists xA \quad (A \Rightarrow B)}{B}\ \exists E$$

Since the use condition of this rule is complex, we illustrate the use of the rule with two examples of incorrect use.

Example 6.1.4 *We show that an incorrect use of* ***rule*** $\exists E$ *leads to a "proof" of an invalid formula.*

context	*number*	*line*	*rule*
1	*1*	`Assume` $\exists xP(x) \wedge (P(x) \Rightarrow \forall yQ(y))$	
1	*2*	$\exists xP(x)$	$\wedge E1$ *1*
1	*3*	$P(x) \Rightarrow \forall yQ(y)$	$\wedge E2$ *1*
1	*4*	$\forall yQ(y)$	$\exists E$ *2,3 ERROR*
	5	`Therefore` $\exists xP(x) \wedge (P(x) \Rightarrow \forall yQ(y)) \Rightarrow \forall yQ(y)$	$\Rightarrow$ *I 1,4*

We did not comply with the condition that the context of the premise $P(x) \Rightarrow \forall y Q(y)$ must not depend on x. It is that the conclusion we reached is not valid. We give an assignment (I, e) which is a counter-model of this "conclusion":

Example 6.1.5 *We show that an incorrect use of* ***rule*** $\exists E$ *leads to a "proof" of an invalid formula.*

context	*number*	*line*	*rule*
1	*1*	`Assume` $\exists x P(x)$	
1,2	*2*	`Assume` $P(x)$	
1	*3*	`Therefore` $P(x) \Rightarrow P(x)$	$\Rightarrow$ *I 2,2*
1	*4*	$P(x)$	$\exists$*E 1,3 ERROR*
1	*5*	$\forall x P(x)$	$\forall$*I 4*
	6	`Therefore` $\exists x P(x) \Rightarrow \forall x\, P(x)$	$\Rightarrow$ *I 1,5*

The conclusion of the rule $\exists E$ is $P(x)$, in contradiction with the use condition of this rule which requires that the conclusion must not depend on x. We give an interpretation which is a counter-model of this "conclusion":

The introduction rule for the existential quantifier, noted $\exists I$, means that if an instance of the formula A where x is replaced with **a term** t **free for** x is true, then the formula $\exists xA$ is true. Notice that this rule is a simple application of corollary 4.3.38, page 77. This justifies the use condition. To convince yourself, it is sufficient to study example 4.3.37, preceding corollary 4.3.38.

Below we give an example of correct use of the introduction rule for the existential quantifier by proving one of De Morgan's laws.

Example 6.1.6 (De Morgan's laws) *We prove that for any formula A, $\neg\forall xA \Rightarrow \exists x\neg A$.*

context	*number*	*line*	*rule*
1	*1*	`Assume` $\neg\forall xA$	
1,	*2*		
1,	*3*		
1,	*4*		
1,	*5*		
1,	*6*		
1,	*7*		
1,	*8*		
1,	*9*		
1	*10*		
1	*11*		
	12	`Therefore` $\neg\forall xA \Rightarrow \exists x\neg A$	$\Rightarrow$ *I 1,11*

6.1.2 Copy

The *copy* rule consists in deducing, from a formula A, another formula A' equal up to renaming of bound variables, as specified in definition 4.4.5, page 81. For example, $\forall x \exists y P(x,y)$ is a copy of $\forall y \exists x P(y,x)$.

$$\frac{A'}{A}\ \text{Copy}$$

Notice that there is no use condition for this rule.

6.1.3 The rules of equality

Two rules characterize equality: a term equals itself (*reflexivity* rule) and if two terms are equal, we can replace one with the other (*congruence* rule).

$\frac{}{t=t}\ Reflexivity$	
$\frac{s=t \quad A<x:=s>}{A<x:=t>}\ Congruence$	

We notice that the first rule has no premise. This is what we also call an *axiom*. We also notice that the use conditions for the second rule are similar to the ones for rules $\forall E$ and $\exists I$. These use conditions are justified the same way as before.

We now give two examples of how equality rules are used.

Example 6.1.7 *Let us prove that $s=t \Rightarrow t=s$, i.e. prove that equality is symmetric.*

context	*number*	*line*	*rule*
1	*1*	`Assume` $s=t$	
1	*2*		
1	*3*		
	4	`Therefore` $s=t \Rightarrow t=s$	

Example 6.1.8 *Let us prove that $s=t \wedge t=u \Rightarrow s=u$, i.e. prove that equality is transitive.*

1	*1*	`Assume` $s=t \wedge t=u$	
1	*2*		
1	*3*		
1	*4*		
	5	`Therefore` $s=t \wedge t=u \Rightarrow s=u$	

6.2 Proof tactics

We introduce two new tactics for the application of the rules $\forall I$ and $\exists E$ and we illustrate these tactics with an example.

6.2.1 Reasoning forward with an existence hypothesis

Let Γ be a set of formulas, x a variable, A and C formulas. Assume we're looking for a proof of C in the environment $\Gamma, \exists xA$.

1. Assume x is not free in Γ, nor in C. In this case, the proof can always be written:

$$\begin{array}{|ll}
\texttt{Assume}\ A & \\
\hline
\text{proof of } C \text{ in the environment } \Gamma, A & \\
\hline
\texttt{Therefore}\ A \Rightarrow C & \Rightarrow I\ 1,_ \\
C & \exists E
\end{array}$$

2. Assume x is free in Γ or in C. We choose a "fresh" variable y, i.e. not free in Γ, C and absent from A, then we reduce to the previous case, via the copy rule. The proof is then written:

$$\begin{array}{|ll}
\exists yA\langle x := y\rangle & \text{Copy of } \exists xA \\
\texttt{Assume}\ A\langle x := y\rangle & \\
\hline
\text{proof of } C \text{ in the environment } \Gamma, A\langle x := y\rangle & \\
\hline
\texttt{Therefore}\ A\langle x := y\rangle \Rightarrow C & \Rightarrow I\ 1,_ \\
C & \exists E
\end{array}$$

The search for the initial proof was reduced to the search for a proof in a simpler environment. This is exactly the kind of reasoning applied in mathematics courses when we look for proof of a formula C with the hypothesis $\exists xP(x)$. We introduce a "new" constant a satisfying $P(a)$ and we prove C under the assumption $P(a)$.

6.2.2 Reasoning backward to generalize

We use the same notations as in the previous paragraph. Assume that we are looking for a proof of $\forall xA$ in the environment Γ.

1. Assume x is not free in Γ. In this case, the proof can always be written:

$$\begin{array}{|ll}
\hline
\text{proof of } A \text{ in the environment } \Gamma & \\
\hline
\forall xA & \forall I
\end{array}$$

2. Assume x is free in Γ. We choose a "fresh" variable y, i.e. not free in Γ, then we reduce to the previous case, via the copy rule. The proof is then written:

$$\begin{array}{|ll}
\hline
\text{proof of } A\langle x := y\rangle \text{ in the environment } \Gamma & \\
\hline
\forall yA\langle x := y\rangle & \forall I \\
\forall xA & \text{Copy of the previous formula}
\end{array}$$

The search for the initial proof was reduced to the search for a proof of a simpler formula in the same environment. It is precisely the proof method we use in a mathematics course when we are looking for a proof of $\forall xP(x)$. We introduce a "new" constant a and we prove $P(a)$. Then we add: since the choice of a is arbitrary, we have $\forall xP(x)$.

6.2.3 An example of application of tactics

We denote "there is one and only one x " by $\exists !x$. Formally, $\exists !xP(x)$ means $\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$. By separating the existence of x and its uniqueness, we can also define $\exists !xP(x)$ by $\exists xP(x) \wedge \forall x\forall y(P(x) \wedge P(y) \Rightarrow x = y)$. These two definitions are of course equivalent and we prove formally that the former implies the latter. Since the proof is long, we must learn how to decompose the proofs.

Proof plan We apply the following two tactics:
— To prove $A \Rightarrow B$, assume A and deduce B
— To prove $A \wedge B$, prove A and prove B.

1 `Assume` $\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$

proof of $\exists x P(x)$ in the environment (1)
proof of $\forall x \forall y(P(x) \wedge P(y) \Rightarrow x = y)$ in the environment (1)

$\exists x P(x) \wedge \forall x \forall y(P(x) \wedge P(y) \Rightarrow x = y)$ $\wedge I$
`Therefore` $\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y)) \Rightarrow \exists x P(x) \wedge \forall x \forall y(P(x) \wedge P(y) \Rightarrow x = y)$ $\Rightarrow I$

Application of the tactic using an existence hypothesis

We are looking for a proof of $\exists x P(x)$ in the environment $\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$. We apply the tactic "reason forward using an existential hypothesis".

reference	formula	
i	$\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$	
1	`Assume` $P(x) \wedge \forall y(P(y) \Rightarrow x = y)$	
2	$P(x)$	$\wedge E$1 1
3	$\exists x P(x)$	$\exists I$ 2, x
4	`Therefore` $P(x) \wedge \forall y(P(y) \Rightarrow x = y) \Rightarrow \exists x P(x)$	$\Rightarrow I$ 1,2
5	$\exists x P(x)$	$\exists E$ i, 3

Application of the tactic to reach a general conclusion

We are looking for a proof of $\forall x \forall y(P(x) \wedge P(y) \Rightarrow x = y)$ in the environment $\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$. We apply, in the order given below, the following tactics:
— "reason forward using an existential hypothesis".
— To prove $A \Rightarrow B$, assume A and deduce B
— "reason backward to get a general conclusion".

environment			
reference		formula	
i		$\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$	
context	number	proof	rule
1	1	`Assume` $P(x) \wedge \forall y(P(y) \Rightarrow x = y)$	
1	2		
1	3		
1	4		
1	5		
1	6		
1	7		
1	8		
1	9		
1	10		
1	11		
1	12		
1	13		
1	14		
	15		
	16	$\forall x \forall y(P(x) \wedge P(y) \Rightarrow x = y)$	

The tactic "use an existential hypothesis" is used to produce line 16. The tactic "introduce an implication" is used to produce line 15: it introduces hypothesis (1) in which x is a free variable. Thus, to apply the tactic "obtain a general conclusion", we need to rename a variable in line 2. Also notice that the formula on line 3 is instantiated on lines 5 and 8.

As can be seen in this example, the difficult part of proofs is focused around the rules $\forall E$ and $\exists I$:

— when reasoning forward, we must find the right instantiations of formulas starting with a universal quantifier.

— when reasoning backward, we must find the right instantiation allowing to deduce a formula beginning with an existential quantifier.

6.3 Consistency of the system

We start by proving two properties about existential and universal quantifiers. Finally we prove the consistency of our deductive system.

Property 6.3.1 *Let Γ be a set of formulas, x a variable and A a formula. Assume x is not free in Γ, so we have : $\Gamma \models A$ if and only if $\Gamma \models \forall XA$.*

Proof :

$\Rightarrow$ Assume $\Gamma \models A$. Let (I,e) be an assignment model of Γ. Since x is not free in Γ, for any state f identical to e except for the value of x, (I,f) and (I,e) give the same value to the formulas of Γ, hence (I,f) is a model of Γ. Since $\Gamma \models A$, for any state f identical to e except for the value of x, (I,f) is a model of A, so (I,e) is model of $\forall xA$.

$\Leftarrow$ Assume $\Gamma \models \forall xA$. Since the formula $\forall xA \Rightarrow A$ is valid (according to corollary 4.3.38), we have $\Gamma \models A$.

□

If we take a critical look at this proof, we observe that it is a paraphrase, in another formalism, of the $\forall I$ rule. The above equivalence explains that the tactic "reason backward to generalize" is a safe tactic. Indeed, admitting the completeness of the system, when x is not free in Γ, there is proof of A in the environment Γ if and only there is a proof of $\forall xA$ in the same environment.

Property 6.3.2 *Let Γ be a set of formulas, x a variable, A and B two formulas. Assume x is free neither in Γ, nor in B, then we have: $\Gamma \models A \Rightarrow B$ if and only if $\Gamma \models (\exists xA) \Rightarrow B$*

Proof :

$\Rightarrow$ Assume $\Gamma \models A \Rightarrow B$. Let (I,e) be an assignment model of Γ. Since x is not free in Γ, for any state f identical to e except for the value of x, (I,f) and (I,e) give the same value to formulas of Γ, hence (I,f) is a model of Γ. Since $\Gamma \models A \Rightarrow B$, for any state f identical to e except for the value of x, (I,f) is a model of $A \Rightarrow B$. Assume (I,e) is a model of $\exists xA$, there is a state g identical to e except for the value of x such that (I,g) is a model of A. Since for any f state identical to e except for the value of x, (I,f) is a model of $A \Rightarrow B$, then (I,g) is a model of B. Since x is not free in B, (I,e) is a model of B.

$\Leftarrow$ Assume $\Gamma \models (\exists xA) \Rightarrow B$. Since the formula $A \Rightarrow (\exists xA)$ is valid (according to corollary 4.3.38 on page 77), we have $\Gamma \models A \Rightarrow B$.

□

If we take a critical look at this proof, we see that it is a paraphrase, in another formalism, of the $\exists E$ rule. The above equivalence explains that the tactic "reason forward with an existential hypothesis", is a safe tactic. Indeed, admitting the completeness of the system, when x is not free in Γ, nor in B, there is a proof of B in the environment Γ, A if and only if there is a proof of B in the environment $\Gamma, \exists xA$.

Theorem 6.3.3 (Consistency of deduction) *If a formula is deduced from an environment then it is a consequence of that environment.*

Proof : The proof follows the same structure as the same theorem in the propositional case (see theorem 3.3.1 on page 58) and we use the same notations dealing only with new rules.

Let Γ be a set of formulas. Let P be a proof of A in this environment. Let C_i be the conclusion and H_i the context of the i-th line of proof P. Remind that the proof lines are numbered starting from 1 and H_0 is the empty list.

We denote by Γ, H_i the set of formulas from Γ and from the list H_i.

Base case: Assume A is deduced from Γ by the empty proof. Then A is an element of Γ, so $\Gamma \models A$. Since H_0 is the empty list, we can conclude: $\Gamma, H_0 \models A$.

Induction step: Assume that for every line $i < k$, $\Gamma, H_i \models C_i$. Let us prove that $\Gamma, H_k \models C_k$.

We only study the cases of the new rules and to simplify we do not make a distinction between two formulas equal up to the abbreviations near negation and equivalence.

— Assume that $C_k = \forall xA$ and this line was deduced by the $\forall I$ rule from formula A with $A = C_i$ and $0 \leq i < k$ or $A \in \Gamma$. If $A = C_i$ and $0 < i < k$, by induction hypothesis we have, $\Gamma, H_i \models A$. If $A \in \Gamma$ then $\Gamma \models A$. Since H_0 is the empty list, there is an i such that $0 \leq i < k$ and $\Gamma, H_i \models A$. Given the use conditions for the rule, x is not free in Γ, H_i. Thus, according to property 6.3.1 on the facing page, we also have $\Gamma, H_i \models \forall xA$. Since the line i is usable on line $k-1$ and H_0 is the empty list, H_i is a prefix of H_{k-1}. Since the context is not modified by the line k, we have $H_{k-1} = H_k$, so $\Gamma, H_k \models C_k$.

— Assume $C_k = A\langle x := t\rangle$ and this line was deduced by the $\forall E$ rule from the formula $\forall xA$ with $\forall xA = C_i$ and $0 < i < k$ or $\forall xA \in \Gamma$. By induction hypothesis or because H_0 is the empty list, there exists an i such that $0 \leq i < k$ and $\Gamma, H_i \models \forall xA$. According to the use conditions of the rule, the term t is free for the variable x in the formula A. Thus, according to corollary 4.3.38 on page 77, the formula $\forall xA \Rightarrow A\langle x := t\rangle$ is valid and as a result $\Gamma, H_i \models A\langle x := t\rangle$. Since the line i is usable on line $k-1$, and since H_0 is the empty list, H_i is a prefix of H_{k-1}. Since the context is not modified by line k, we have $H_{k-1} = H_k$, thus $\Gamma, H_k \models C_k$.

— Assume that $C_k = \exists xA$ and this line was deduced by the rule $\exists I$ from the formula $A\langle x := t\rangle$ with $A\langle x := t\rangle = C_i$ and $0 < i < k$ or $A\langle x := t\rangle \in \Gamma$. By induction hypothesis or because H_0 is the empty list, there is an i such that $0 \leq i < k$ and $\Gamma, H_i \models A\langle x := t\rangle$. According to the use conditions of the rule, the term t is free for the variable x in formula A. Thus, according to corollary 4.3.38 on page 77, the formula $A\langle x := t\rangle \Rightarrow \exists xA$ is valid and as a result $\Gamma, H_i \models \exists xA$. Since the line i is usable on line $k-1$, and since H_0 is the empty list, H_i is a prefix of H_{k-1}. Since the context is not modified by line k, we have $H_{k-1} = H_k$, thus $\Gamma, H_k \models C_k$.

— Assume $C_k = B$ and this formula was deduced by the $\exists E$ rule from the formula $\exists xA$ with $\exists xA = C_i$ and $0 < i < k$ or $\exists xA \in \Gamma$ and formula $A \Rightarrow B$ with $A \Rightarrow B = C_j$ and $0 < j < k$ or $A \Rightarrow B \in \Gamma$. By induction hypothesis or because H_0 is the empty list, there are integers i and j such that $0 \leq i < k$, $0 \leq j < k$, $\Gamma, H_i \models \exists xA$ and $\Gamma, H_j \models A \Rightarrow B$. Given the use conditions of the rule, x is not free in Γ, H_j, nor in B. Thus, according to property 6.3.2 on the preceding page, we also have $\Gamma, H_j \models (\exists xA) \Rightarrow B$. Since lines i and j are usable on line $k-1$, and since H_0 is the empty list, H_i and H_j are prefixes of H_{k-1}. Since the context is not modified by line k, we have $H_{k-1} = H_k$, so $\Gamma, H_k \models \exists xA$ and $\Gamma, H_k \models (\exists xA) \Rightarrow B$. As a result $\Gamma, H_k \models C_k$.

— Assume $C_k = A'$ and this formula was deduced, by the copy rule from formula A with $A = C_i$ and $0 < i < k$ or $A \in \Gamma$. By induction hypothesis or because H_0 is the empty list, there is an i such that $0 \leq i < k$, $\Gamma, H_i \models A$. Since, according to theorem 4.4.6 on page 82, the formulas A and A' are equivalent, we have $\Gamma, H_i \models A'$. Since line i is usable on line $k-1$, and since H_0 is the empty list, H_i is a prefix of H_{k-1}. Since the context is not modified by line k, we have $H_{k-1} = H_k$, thus $\Gamma, H_k \models C_k$.

— Assume $C_k = (t = t)$. Since this formula is valid (with the meaning given to equality), $\Gamma, H_k \models C_k$.

— Assume $C_k = A\langle x := t\rangle$ and this line was deduced, by the congruence rule, from formula $s = t$ with $(s = t) = C_i$ and $0 < i < k$ or $(s = t) \in \Gamma$ and from formula $A\langle x := s\rangle$ with $A\langle x := s\rangle = C_j$ and $0 < j < k$ or $A\langle x := s\rangle \in \Gamma$. By induction hypothesis or because H_0 is the empty list, there are integers i and j such that $0 \leq i < k$, $0 \leq j < k$, $\Gamma, H_i \models (s = t)$ and $\Gamma, H_j \models A\langle x := s\rangle$. Since lines i and j are usable on line $k-1$, and since H_0 is the empty list, H_i and H_j are prefixes of H_{k-1}. Since the context is not modified by line k, we have $H_{k-1} = H_k$, so $\Gamma, H_k \models (s = t)$ and $\Gamma, H_k \models A\langle x := s\rangle$. According to theorem 4.3.36 on page 77 and the use conditions of the rule, we have: $s = t, A\langle x := s\rangle \models A\langle x := t\rangle$. As a result $\Gamma, H_k \models C_k$.

Since the last line of a proof has an empty context, we obtain that its conclusion is a consequence of Γ. □

6.4 Exercises

Exercise 102 (Natural Deduction) *Prove the following formulas using natural deduction:*

1. *The famous syllogism, "All men are mortal, Socrates is a man, therefore Socrates is mortal", which we formalize as* $\forall x(H(x) \Rightarrow M(x)) \wedge H(socrates) \Rightarrow M(socrates)$.
2. $\forall x P(x) \Rightarrow \exists y P(y)$.
3. $\forall x P(x) \Rightarrow \exists x P(x)$.
4. $\forall x(P(x) \wedge Q(x)) \Rightarrow \forall x P(x) \wedge \forall x Q(x)$. *Note that example 6.1.2 on page 111 gives the proof of the reciprocal.*
5. $\forall x(P(x) \Rightarrow Q(x)) \wedge \exists x P(x) \Rightarrow \exists x Q(x)$.
6. $\exists x(P(x) \vee Q(x)) \Rightarrow \exists x P(x) \vee \exists x Q(x)$.
7. $\exists x P(x) \vee \exists x Q(x) \Rightarrow \exists x(P(x) \vee Q(x))$. *(**)*
8. $\forall x(P(x) \Rightarrow Q(x)) \wedge \forall x(Q(x) \Rightarrow R(x)) \Rightarrow \forall x(P(x) \Rightarrow R(x))$.
9. $\forall x\ (P(x) \Rightarrow Q(x)) \wedge \exists x\ \neg Q(x) \Rightarrow \exists x\ \neg P(x)$.

In this exercise, the formulas $P(x)$ *and* $Q(x)$ *can be replaced by* any *other formulas.* □

Exercise 103 (Natural Deduction) *Prove the following formulas:*

1. $\forall x \forall y P(x,y) \Rightarrow \forall y \forall x P(x,y)$.
2. $\exists x \exists y P(x,y) \Rightarrow \exists y \exists x P(x,y)$.
3. $\exists x \forall y P(x,y) \Rightarrow \forall y \exists x P(x,y)$.
4. $\forall x(Q(x) \Rightarrow \forall y(R(y) \Rightarrow P(x,y))) \Rightarrow \forall y(R(y) \Rightarrow \forall x(Q(x) \Rightarrow P(x,y)))$. *(*)*

In this exercise, the formula $P(x,y)$ *can be replaced by any other formula. However,* $Q(x)$ *can only be replaced by a formula that does not have y as a free variable, and* $R(y)$ *can only be replaced by a formula that does not have x as a free variable. Explain the reason for these constraints.* □

Exercise 104 (Find the Mistake) *Consider the following formula:*

$$\exists x\ P(x) \wedge \forall x\ Q(x) \Rightarrow \exists x\ (P(x) \wedge Q(x)).$$

Among the following three natural deduction proofs, only one is correct. Identify the correct proof, and justify why the other two are incorrect.

1.

context	*number*	*proof*	*justification*
1	*1*	`Suppose` $\exists x\ P(x) \wedge \forall x\ Q(x)$	
1	*2*	$\exists x\ P(x)$	$\wedge$*E1 1*
1	*3*	$\forall x\ Q(x)$	$\wedge$*E2 1*
1,4	*4*	`Suppose` $P(x)$	
1,4	*5*	$Q(x)$	$\forall$*E 3, x*
1,4	*6*	$P(x) \wedge Q(x)$	$\wedge$*I 4,5*
1,4	*7*	$\exists x(P(x) \wedge Q(x))$	$\exists$*I 6, x*
1	*8*	`Thus` $P(x) \Rightarrow \exists x(P(x) \wedge Q(x))$	$\Rightarrow$*I 4,7*
1	*9*	$\exists x(P(x) \wedge Q(x))$	$\exists$*E 2,8*
	10	`Thus` $\exists x\ P(x) \wedge \forall x\ Q(x) \Rightarrow \exists x(P(x) \wedge Q(x))$	$\Rightarrow$*I 1,9*

2.

context	*number*	*proof*	*justification*
1	*1*	`Suppose` $\exists x\, P(x) \wedge \forall x\, Q(x)$	
1	*2*	$\exists x\, P(x)$	$\wedge$*E1 1*
1	*3*	$\forall x\, Q(x)$	$\wedge$*E2 1*
1,4	*4*	`Suppose` $P(x)$	
1,4	*5*	$Q(x)$	$\forall$*E 3, x*
1,4	*6*	$P(x) \wedge Q(x)$	$\wedge$*I 4,5*
1	*7*	`Thus` $P(x) \Rightarrow P(x) \wedge Q(x)$	$\Rightarrow$*I 4,6*
1	*8*	$P(x) \wedge Q(x)$	$\exists$*E 2,7*
1	*9*	$\exists x(P(x) \wedge Q(x))$	$\exists$*I 9, x*
	10	`Thus` $\exists x\, P(x) \wedge \forall x\, Q(x) \Rightarrow \exists x(P(x) \wedge Q(x))$	$\Rightarrow$*I 1,9*

3.

context	*number*	*proof*	*justification*
1	*1*	`Suppose` $\exists x\, P(x) \wedge \forall x\, Q(x)$	
1	*2*	$\exists x\, P(x)$	$\wedge$*E1 1*
1	*3*	$\forall x\, Q(x)$	$\wedge$*E2 1*
1,4	*4*	`Suppose` $P(x)$	
1	*5*	`Thus` $P(x) \Rightarrow P(x)$	$\Rightarrow$*I 4,4*
1	*6*	$P(x)$	$\exists$*E 2,6*
1	*7*	$Q(x)$	$\forall$*E 3, x*
1	*8*	$P(x) \wedge Q(x)$	$\wedge$*I 7,8*
1	*9*	$\exists x(P(x) \wedge Q(x))$	$\exists$*I 9, x*
	10	`Thus` $\exists x\, P(x) \wedge \forall x\, Q(x) \Rightarrow \exists x(P(x) \wedge Q(x))$	$\Rightarrow$*I 1,10*

□

Exercise 105 (Copy) *Prove* $\forall x \forall y P(x,y) \Rightarrow \forall x \forall y P(y,x)$ *using natural deduction with as few copies as possible.* □

Exercise 106 (Natural Deduction) *Prove the following formulas using natural deduction (note that Q is a propositional variable):*

1. $\forall x(Q \wedge P(x)) \Rightarrow Q \wedge \forall x P(x)$.
2. $Q \wedge \forall x P(x) \Rightarrow \forall x(Q \wedge P(x))$.
3. $\forall x(Q \vee P(x)) \Rightarrow Q \vee \forall x P(x)$. *(**)*
4. $Q \vee \forall x P(x) \Rightarrow \forall x(Q \vee P(x))$.
5. $\exists x(Q \wedge P(x)) \Rightarrow Q \wedge \exists x P(x)$.
6. $Q \wedge \exists x P(x) \Rightarrow \exists x(Q \wedge P(x))$.
7. $\exists x(Q \vee P(x)) \Rightarrow Q \vee \exists x P(x)$.
8. $Q \vee \exists x P(x) \Rightarrow \exists x(Q \vee P(x))$. *(*)*

In this exercise, the formula $P(x)$ can be replaced by any formula. However, Q can be replaced only by a formula that does not have x as a free variable. □

Exercise 107 (Proof) *Prove the formula* $\neg \exists x P(x) \Rightarrow \forall x \neg P(x)$. *Verify that* $P(x)$ *can be replaced by any formula. Note: the reciprocal to this formula is proven in example 6.1.6 on page 112.* □

Exercise 108 (Equality) *Prove the following formulae:*

1. $R(a,c) \wedge (a = b) \Rightarrow R(b,c)$.
2. $x = y \Rightarrow f(x,z) = f(y,z)$.
3. $\forall x \exists y(x = y)$.

4. $\exists x \forall y\ x = y \Rightarrow \forall x \forall y\ x = y$. (*)

□

Exercise 109 (Equality,)** *Prove that the second definition of " there exists one and only one" (see subsection 6.2.3 on page 114) implies the first, that is, prove the formula:* $\exists x P(x) \wedge \forall x \forall y (P(x) \wedge P(y) \Rightarrow x = y) \Rightarrow \exists x (P(x) \wedge \forall y (P(y) \Rightarrow x = y))$.

□

Exercise 110 (Induction and natural deduction,*)** *We can define addition using the following formulas:*

(a) $\forall n (n + 0 = n)$.

(b) $\forall n \forall p (n + s(p) = s(n + p))$.

These two formula allow us to do additions: we can prove, among other, that $s(0) + s(0) = s(s(0))$. *But they do not allow the proof of more general properties of addition, which require the recurrence principle.*

1. *Show that from hypotheses (a) and (b), we cannot deduce* $\forall n (0 + n = n)$.
2. *We give the name* $P(n)$ *to the property above and describe the recurrence principle on this property as follows:*
   (c) $\forall n (P(n) \Leftrightarrow 0 + n = n)$.
   (d) $P(0) \wedge \forall n (P(n) \Rightarrow P(s(n))) \Rightarrow \forall n P(n)$.
   Prove using natural deduction that $(a) \wedge (b) \wedge (c) \wedge (d) \Rightarrow \forall n P(n)$.

□

Exercise 111 (Exam 2009) *All of the following proofs must be justified.*

1. *Prove using natural deduction that this formula is valid:*
   $(\exists x\ p(x) \Rightarrow \forall x\ q(x)) \Rightarrow \forall x (p(x) \Rightarrow q(x))$.
2. *Prove using natural deduction that this formula is valid:*
   $\exists x\ p(x) \wedge \forall x (p(x) \Rightarrow p(f(x))) \Rightarrow \exists x\ p(f(f(x)))$.
3. *We denote by* $f^n(x)$ *the term obtained applying* f *n times to x.*
   For instance, $f^0(x) = x, f^1(x) = f(x), f^2(x) = f(f(x))$.
   Let Γ, Δ *be two formula sets and* A, B *two formulas. We remind* $\Gamma \vdash A$ *is true if there is a proof of* A *in the* environment Γ.
   We give some trivial properties of the $\vdash$ *relation.*
   Monotonicity : if $\Gamma \vdash A$ *and* $\Gamma \subset \Delta$ *then* $\Delta \vdash A$.
   Composition : if $\Gamma \vdash A$ *and* $\Gamma \vdash A \Rightarrow B$ *then* $\Gamma \vdash B$.
   (a) Prove using natural deduction of the following property:
   $\forall x (p(x) \Rightarrow p(f(x))) \vdash \exists x p(f^n(x)) \Rightarrow \exists x\ p(f^{n+1}(x))$
   (b) Deduce from the above property, monotonicity and composition that for any *natural integer n :*
   $\exists x\ p(x), \forall x (p(x) \Rightarrow p(f(x))) \vdash \exists x\ p(f^n(x))$.

□

Exercise 112 (Exam 2012) *Prove the following formulas using first-order natural deduction:*

1. $\neg \forall x P(x) \vee \neg \exists y Q(y) \Rightarrow \neg (\forall x P(x) \wedge \exists y Q(y))$
2. $\forall x \forall y (P(y) \Rightarrow R(x)) \Rightarrow \exists y P(y) \Rightarrow \forall x R(x)$
3. $\neg \forall x \neg P(x) \Rightarrow \exists x P(x)$

□

Exercise 113 (Exam 2013) *Prove the following formulae using first order natural deduction.*

1. $\exists x (Q(x) \Rightarrow P(x)) \wedge \forall x Q(x) \Rightarrow \exists x P(x)$
2. $\forall x\ \forall y (R(x,y) \Rightarrow \neg R(y,x)) \Rightarrow \forall x \neg R(x,x)$

□

Exercise 114 (Questions from various exams) *Prove the following formulae using first order natural deduction.*

1. $\exists x (P(x) \vee Q(x)) \wedge \forall x \neg Q(x) \Rightarrow \exists x P(x)$
2. $\forall x (P(x) \Rightarrow Q(x)) \wedge \exists x (P(x) \wedge R(x)) \Rightarrow \exists x (Q(x) \wedge R(x))$
3. $\exists x \neg (P(x) \vee \neg P(x)) \Rightarrow \forall x P(x)$

□

Bibliography

[1] Collin Allen and Michael Hand. *Logic Primer*. MIT, 2001.

[2] Peter B. Andrews. *An introduction to mathematical logic : to truth through proof*. Academic Press, 1986.

[3] Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 445–532. Elsevier and MIT Press, 2001.

[4] Alonzo Church. A note on the entscheidungsproblem. *Journal of Symbolic Logic*, 1(1):40–41, 1936.

[5] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[6] René Cori and Daniel Lascar. *Logique mathématique Cours et exercices I. Calcul propositionnel, algèbres de Boole, calcul des prédicats*. Masson, 1993.

[7] René Cori and Daniel Lascar. *Logique mathématique Cours et exercices II. Fonctions récursives, théorème de Gödel, théorie des ensembles, théories des modèles*. Masson, 1993.

[8] René David, Karim Nour, and Christophe Raffali. *Introduction à la logique*. Dunod, 2001.

[9] Martin Davis, George W. Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Communications of The ACM*, 5:394–397, 1962.

[10] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.

[11] Herbert B. Enderton. *A mathematical Introduction to Logic*. Academic Press, 2001.

[12] Gerhard Gentzen. Untersuchungen über das logische Schließen I. *Mathematische Zeitschrift*, 39:176–210, 1935. 10.1007/BF01201353.

[13] Gerhard Gentzen. Untersuchungen über das logische Schließen II. *Mathematische Zeitschrift*, 39:405–431, 1935. 10.1007/BF01201363.

[14] Roger Godement. *Cours d'algèbre*. Hermann, 1973.

[15] Gérard P. Huet. unification in typed lambda calculus. In Corrado Böhm, editor, *Lambda-Calculus and Computer Science Theory, Proceedings of the Symposium Held in Rome, March 25-27, 1975*, volume 37 of *Lecture Notes in Computer Science*, pages 192–212. Springer, 1975.

[16] Gérard P. Huet. Higher order unification 30 years later. In Victor Carreño, César Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics, 15th International Conference, TPHOLs 2002, Hampton, VA, USA, August 20-23, 2002, Proceedings*, volume 2410 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2002.

[17] Michael Huth and Mark Ryan. *Logic in Computer Science*. Cambridge University Press, 2004.

[18] Jacquemin. *Logique et Mathématiques 109 exercices corrigées*. Masson, 1994.

[19] Stephen C. Kleene. *Logique Mathématique*. Armand Colin, 1971.

[20] Thierry Lucas, Isabelle Berlanger, and Isabelle De Greef. *Initiation à la logique formelle*. De Boeck, 2003.

[21] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4:258–282, April 1982.

[22] William McCune. Prover9 and mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.

[23] J. Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41, January 1965.

[24] Alan Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1937.

Index