Table des matières

Chapitre 1

Chapter 1 : Introduction to decidability

1.1 Introduction

For many centuries now, mathematicians have been describing and using computational methods to solve their problems. However, until a relatively recent date (1934), they did not know precisely what a computational method is.

The remarkable ability of mathematics to transform what constitutes their methods and techniques brings them closer to philosophy. This self-reflective capacity, for instance, allows for treating mathematical theories as objects of study (and therefore as theorems), just as integers and partial differential equations are objects of study and theorems.

Indeed, regarding the issue of computational methods, the self-reflectivity of mathematics has worked, giving rise to a series of concepts and results that are among the most profound and fruitful of the twentieth century.

Examples :
— Algorithms,
— Calculable functions,
— Decidable and undecidable problems.

Today, mathematicians have a precise definition of what we should call a computational method. They know exactly which problems these methods can address, and even more remarkably, they are aware that certain problems are not "computable." In fact, one of the most extraordinary aspects of these

studies is that they enable the establishment of negative results, in the form of : for a particular problem, not only is no method currently known, but none will ever be.

These problems for which it is demonstrated that there is no suitable computational method are called undecidable problems.

The initial undecidability results from the 1930s appeared artificial, but quickly it was realized that quite simple problems fell into the class of undecidable problems. In particular, in computer science, many natural questions that arise for programmers are found, after investigation, to correspond to undecidable problems. When one proves that a problem P is undecidable, it is deduced that : - one must give up solving it as is, and thus, - one needs to find a simplified version of P, then either succeed in solving it, or again establish that it is still undecidable and therefore simplify it further, etc. (recursive functions).

1.2 Brief history

The informal concept of an algorithm is extremely ancient, and the processes we learned in primary school, such as adding two numbers written in base ten or multiplying them, are algorithms.

In fact, at the beginning of the century, mathematicians did not suspect that it would be possible to rigorously define this concept, let alone demonstrate for certain problems that no algorithm exists.

The work of identifying and formulating the concept of an algorithm was carried out in several stages between 1931 and 1936 by mathematicians Church, Kleene, Turing, and Gödel.

They introduced several different classes of functions, which they later showed to coincide, and which they then recognized as the class of computable functions. A function is computable if there is a finite way to describe it that effectively allows one to compute all its values. The precise definition of the concept of a computable function simultaneously establishes the notion of an algorithm. Therefore, it can be said that in 1936, the exact formulation of the concept of an algorithm was established. Among all the ultimately equivalent definitions of the concept of an algorithm formulated in the 1930s and since, the one given by Turing in 1936 is the most practical for theorists,

and it is still used today. We will rely on it further to define the concept of an algorithm.

Every year, many decidability or undecidability results are established in numerous areas of mathematics, and given the multitude of open questions, it is certain that this trend is far from abating. Here, we will focus solely on the issue of the possibility or impossibility of an algorithm, and not on the problem of efficiency, which also constitutes a highly active area of research and reflection today.

1.3 Some decidable, semi-decidable, and undecidable problems

Problem A.
Let m and n be two given integers > 1. Is m a multiple of n ? We know that it is true that 12 is a multiple of 2, and it is false that 16 is a multiple of 5. We even know more than that ; we know how to go about determining for any n and m when it is true that "m is a multiple of n" and when it is false that "m is a multiple of n."

Indeed, it is sufficient to :
— Divide m by n.
— Look at the remainder obtained, r.
— If $r = 0$, then it is TRUE that "m is a multiple of n."
— Otherwise, it is FALSE that "m is a multiple of n."
This general and systematic calculation procedure constitutes an (informal) algorithm. It is an entirely reliable process that operates in a finite amount of time for all possible sets of data and always leads to the correct answer : it demonstrates that problem A is decidable.

It is important to note that, to be precise when talking about a decidable or undecidable problem, one must specify the problem's parameters. In our example, n and m are two integers > 1. In fact, we are not seeking to solve a single problem ; we are aiming to solve an infinite class of problems, encompassing all problems like this :

Is 2 a multiple of 2 ?
Is a 3 a multiple of 2 ?
.....

Is 10 a multiple of 3 ?
.....

Problem B.
The problem of prime numbers. Let n be a given integer > 1.
Is n a prime number ?

Problem C.
There are also problems that are semi-decidable.
For example, consider the following program P :
"3x + 1" Does the program below halt on a given x ?

while x > 1 do
{ if (x mod $2 = 0$) then x := x/2 ;
else x := 3x + 1 ; }

Answer. For this problem, one can use the following procedure : "Starting from the given x, execute this program, and if it halts, return 'YES.' However, if for a certain x the program '3x + 1' enters an infinite loop, this procedure can never provide the answer 'NO.' Such a procedure is called a semi-algorithm, and it can also be referred to as a semi-decidable program."

It is unknown whether there exists a decision algorithm for this problem that, for every given x, yields a correct answer of "YES" or "NO."

Problem D.
There are also undecidable problems.

Example :
Let algorithm A be.ıP(x) { While AP(x) (if x is even), then x := x*2 }.
It is evident that if you pass an even x as a parameter to this algorithm, it will never terminate, so it is undecidable.

Chapitre 2

Chapter 2 : A brief introduction to computability with the Turing machine and primitive recursive functions

2.1 Introduction

A program can be seen as the breakdown of a task into a sequence of elementary instructions (manipulating basic data) that can be understood by the programmable device you wish to use. However, each programmable device (in practice, each processor) is characterized by its own set of elementary instructions. In practice, this diversity is resolved through the existence of compilers or interpreters that translate the instructions from the (high-level) language provided to programmers into several elementary instructions (machine language) for the specific processor in use. However, this solution is not sufficient for the needs of theoretical computer science, which requires a unified representation of the concept of a programmable device, allowing for the demonstration of general results (decidability, complexity, etc.) that hold true for all concrete programmable devices that one can consider. To this end, the notion of a Turing machine was developed, which serves as an abstraction (and formalization) of the concept of a programmable device.

The Turing machine is a fundamental theoretical model for the theory of computability and complexity. Despite its extreme simplicity, it can be proven to simulate any operation achievable by any processor, no matter how powerful. It succinctly encapsulates the concept of a computer and serves as

an ideal framework for reasoning about algorithmic concepts.

There are numerous formulations of this machine, and if the one that follows doesn't exactly match the one you are familiar with, it's not of great significance because it's very easy to demonstrate their equivalence.

2.2 The formal definition of a Turing machine (TM)

A Turing machine is presented as a quintuplet $\langle Q, A, q_0, b, d \rangle$ where :
— Q is a finite set of states,
— A is a finite set of symbols (alphabet),
— $q_0 \in Q$ represents the initial state,
— b is a symbol not belonging to A, called the blank symbol,
— d is a transition function from Q x B $\longrightarrow$ (B $\cup\{<, >\}$) x Q, with B = A $\cup\{b\}$.

We will provide a more physical description of how a Turing machine operates, in line with the very idea of Alan Turing (the creator of the machine now bearing his name). A Turing machine is nothing more than a modified typewriter. Instead of working on a sheet of paper, the Turing machine operates on an infinitely long tape to the left and right (indexed by the set of integers **Z**) using a Read/Write head. The tape is composed of multiple cells, with each cell holding a symbol from the alphabet A or a blank symbol. A cell is said to be empty if it contains the blank symbol. The Read/Write head is capable of :
— Writing a symbol in the current cell (the one the head is pointing to),
— Erasing the content of the current cell (equivalent to writing the blank symbol),
— Moving to the left or to the right one cell, ($< et >$ in the definition),
— Reading the symbol contained in the current cell.

Example. The machine shown in the following figure is a machine that, when in state p, reads a "b," transitions to state q, writes an "a" in place of the "b," and moves its read head to the left.

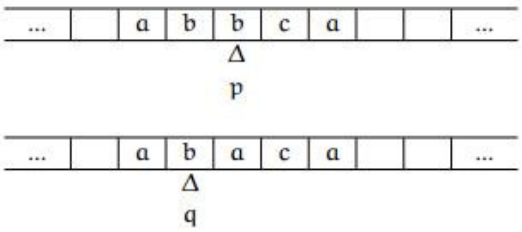


FIGURE 2.1 – Example of Turing machine

From a modeling perspective, the first three operations are represented by the set B $\cup\{<, >\}$, In other words, we write a symbol from set B (if this symbol is b, it is considered as erasing the cell), or we move the Read/Write head one cell to the left or right. Note that in the majority of works, these operations are represented by set B x $\{<, >\}$,i.e. We write a symbol from set B and move the Read/Write head to the left or right.

The transition function d and the initial tape content determine the sequence of operations executed by the machine. At the start of execution, the machine is in the initial state q_0, and, by convention, the Read/Write head is positioned on the cell indexed as 0 ;The equality d(q, s) = (o, q′) It reads as follows : If the Turing machine is in state q and the cell contains symbol s (the Read/Write head is said to read symbol s), then the Turing machine performs operation o and transitions to state q′.

An instruction of a Turing machine is called a quadruplet, which can be translated as follows : (q, s, o, q′) such that d(q, s) = (o, q′). To define a set of instructions, or quadruplets Q x B x (B $\cup\{<, >\}$) x Q This is equivalent to defining the graph of a transition function ; thus, the two approaches are equivalent. The set of instructions is called : *program* of the Turing machine. When the machine is in state q, it reads symbol s, and the function d is not defined for the pair (q, s), it is said that the machine halts, and state q is a halt state.

Example Write the Turing machine that recognizes words of even lengths, considering that the alphabet = $\{a, \#\}$, if the word's length is even, the machine writes T on the tape ; otherwise, it writes F.

Solution

q_0 a D q_1

q_1 a D q_2
q_2 # T q_3
q_3 T Halt
q_2 a D q_4
q_4 a D q_2
q_4 # F q_4
q_1 # F q_4
q_4 F Halt

2.3 Primitive recursive functions.

Primitive recursive functions were introduced by Gödel in 1934 in his work on incompleteness. They allow the description of functions for which it is clear that the computation always terminates. They correspond to functions that can be computed without the "While" instruction in a structured language like Pascal, meaning they can be computed using "if-then-else" and "for" constructs. The aim of this small class is to understand this expressiveness and observe some of its limitations.

Informally, these computable functions are those that can be defined over the set of natural numbers $\mathbb{N}$ by recursion :
$\bigcup_{k\in\mathbb{N}} F_k$ with $F_k = \mathbb{N}^k \longrightarrow \mathbb{N}$

They are as follows :
— The basic functions,
— Function composition,
— The recursion mechanism.

2.3.1 Definition of basic primitive recursive functions

— The zero() function or $\boldsymbol{O}() \in F_0$ such that $\boldsymbol{O}() = 0$,
— The successor function succ(x) or $\sigma(n) \in F_1$ such that $\sigma(n) = n+1$,
— The projection function $proj_i^k$ or $\pi_i^k \in F_k$, $k \geqslant 1, 1 \leqslant i \leqslant k$ such that $\pi_i^k(n_1, ..., n_k) = n_i$.
Example : $\pi_2^3(5, 6, 8) \longrightarrow 6$.

2.3.2 Definition of function composition

Let $g \in F_l$ et $h_1, ..., h_l \in F_k$. The composition of g and the functions $h_1, ..., h_l$ is defined as $f \in F_k$ such that :

f(n) = $g(h_1(n), ..., h_l(n))$ with n = $(n_1, ..., n_k)$
Example : In this example, we represent the composition as ∘,We can also represent it as Comp.
succ ∘ Zero ⟶ 1,
succ ∘ succ ∘ Zero ⟶ 2,
succ ∘ π_3^3(0, 2, 5)⟶ 6.

2.3.3 Definition of primitive recursion

Let $g \in F_k$ and $h \in F_{k+2}$, the function $f \in F_{k+1}$ such that :
— f(n, 0) = g(n),
— f(n, m+1)= h(n,m,f(n,m))
Is the function defined from g and h by primitive recursion.
Note : The function f is computable if g and h are computable.

We can also represent recursion with the following recursive diagram :

Function f(x, k)
If : k=0
F(x)
Else :
G(x, k-1, f(x, k-1))
End
End

We replaced the functions g with F and h with G to indicate that there are multiple formal representations. If F and G are primitive recursive, then f is primitive recursive. f recurses on the argument k, while the argument x is additional data.

2.3.4 Definition of primitive recursive functions

In general, primitive recursive functions are all functions that can be constructed from basic functions through composition and primitive recursion.

Example : Show that the function +(x, k) is primitive recursive.

Based on the recursive diagram seen previously, we can have :

Function +(x, k)
If : k=0
F(x)
Else :
G(x, k-1, f(x, k-1))
End
End

So for F, we take the identity function $\pi_1^1(x)$ and for the function G, we take : succ $\circ\pi_3^3$ that is :

Function +(x, k)
If : k=0
$\pi_1^1(x)$
Else :
succ $\circ\pi_3^3$(x, k-1, f(x, k-1))
End
End

Example : +(5,3)
+(5,3) = succ $\circ\pi_3^3$(5,2,+(5,2)) =
succ $\circ\pi_3^3$(5,2,succ $\circ\pi_3^3$(5,1,+(5,1)) =
succ $\circ\pi_3^3$(5,2,succ $\circ\pi_3^3$(5,1,succ $\circ\pi_3^3$(5,0,+(5,0))=
succ $\circ\pi_3^3$(5,2,succ $\circ\pi_3^3$(5,1,succ $\circ\pi_3^3$(5,0,π_1^1(5))))
As : π_1^1(5)=5 then :
succ $\circ\pi_3^3$(5,0,+(5,0)) = 6 and
succ $\circ\pi_3^3$(5,1,+(5,1)) = 7 finally
succ $\circ\pi_3^3$(5,2,+(5,2)) = 8

We can observe that we have successfully constructed the addition function using the recursion rule based on the two primitive recursive functions :

— The function F = π_1^1 which is a basic function (the projection)
— The function G = $succ \circ \pi_3^3$ which is a composition of two basic functions : projection and the successor function.

Note 1 : In recursion, the order of arguments for function G(x, k-1, f(x, k-1)) is not important ; you may find in other documents G(x,f(x, k-1), k-1).

Note 2 : We can also solve the problem using the recursion rule as follows :
+(x,0) = x + 0 = x = π_1^1
+(x, y+1) = +(x,y)+1 = succ(+(x,y)) = $succ(\pi_2^3(x, +(x, y), y)) = succ \circ$

$\pi_2^3(x, +(x, y), y)$.

Note 3 : The objective of the "Primitive Recursive Functions" section is firstly to explore another computational model than the Turing machine with its various forms and to demonstrate that primitive recursive functions and the Turing machine have the same expressive power, along with, of course λ-calcul (which is not covered in this program).

2.4 Notes for tutorials

To have a more algorithmic representation of the Turing machine, we will write the instructions as follows :

« current stat » « symbol » « operation » « next stat ».

« operation » can be a move to the right « D » or to the left « G » or the replacement of the read symbol with another. Example : q_0 s D q_1 which represents a move to the right after reading the symbol s, q_0 s s$'$ q_1 In this case, there is no movement ; however, there is a replacement of the symbol s by the symbol s$'$.

Initially, the Read/Write head is always positioned on a symbol other than the blank symbol, and we do not loop on the initial state.

2.5 Exercises

Exercise 1 Let the following instructions of a Turing machine be :

$q_0\ s_0\ D\ q_1$
$q_0\ s_1\ s_2\ q_2$
$q_1\ s_0\ D\ q_1$
$q_1\ s_2\ D\ q_1$
$q_1\ s_1\ D\ q_2$
$q_1\#$
$q_2\ s_2\ G\ q_3$
$q_3\ (s_0/s_1/s_2)\ G\ q_3$
$q_3\#$

Execute this Turing machine on the following sequence : $\#s_0s_0s_2s_1s_2s_1s_2s_0s_2\#$ considering that # represents the blank symbol.

Exercise 2 Write the Turing machine that, given a word on the tape composed of symbols a and b, determines if the word ends with a "b" or not. It writes a "T" at the end of the word if true and an "F" otherwise. Blank symbol = $.

Exercise 3 Modify the previous Turing machine to check if the input word ends with the same starting symbol (whether it is "a" or "b").

Exercise 4 The decimal values corresponding to the ASCII codes of the letters are :

— A : 65 ; B : 66 ; C : 67 ; etc,
— a : 97 ; b : 98 ; c : 99 ; etc.

To convert the ASCII code of a lowercase letter to the corresponding uppercase letter, you just need to change the 3rd bit from the left, from 1 to 0. Furthermore, the first two bits are always equal to "01" and the last 5 bits remain unchanged.

A : 65 = 01000001 and a : 97 = 01100001
C : 67 = 01000011 and c : 99 = 01100011
Write the Turing machine that transforms a lowercase letter into an uppercase one.

Exercise 5 Write the Turing machine that recognizes the sequence 0001 in a given word, considering that the tape contains multiple words and the alphabet $\Sigma = \{0, 1, \#\}$. There are multiple words on the tape separated by a single #. Two consecutive # symbols indicate the end of the sequence.

Exercise 6 Write the Turing machine that checks if a given word on the tape contains the following character sequence : « aab ». The blank symbol = # and there are multiple words on the tape separated by a single #. Two consecutive # indicate the end of the sequence. A = $\{a, b, \#\}$. We write a "T" or a "F".

Exercise 7 Write the Turing machine that replaces the "0" that comes after two "1" with a "1". A = $\{0, 1, \#\}$, q_0 is the initial stat and the tape contains one word.

Exercise 8 Write the Turing machine that transforms the word on the tape written in alphabet $\{a, b, \#\}$ so that all "a"s are at the beginning. Example : aabbaba becomes aaaabbb.

Exercise 9 Show that the following functions are primitive recursive :

1. The Plus function, plus = x + y,
2. The Sigma function, Sigma = $\Sigma_{i=0}^{x} i$,
3. The predecessor function (pred(x)),
4. The subtraction function (sub(x,y)) such that sub(x,y) = $\begin{cases} x-y \; si \; x>y \\ 0 \; si \; x \leq y \end{cases}$,
5. The absolute difference function $|x - y| = \begin{cases} x-y \; si \; x \geq y \\ y-x \; si \; x<y \end{cases}$,
6. The Alpha function such that $\alpha(x) = \begin{cases} 1 \; si \; x=0 \\ 0 \; si \; x \neq 0 \end{cases}$,
7. The multiplication function, mult = x * y,
8. The factorial function, Fact(x) = $x!$.

Chapitre 3

Chapter 3 : Introduction to formal systems

The expression "formal systems" consists of two words :

— **Formal :** which mainly concerns the form and does not take into account the content.

— **System :** which is defined as a set of material or non-material elements that depend on each other in a way to form a coherent and organized whole.

3.1 Definition

Axiomatic-Deductive systems or formal systems (FS) provide a general framework to express and rigorously study the notions of axiomatics and deductive mechanisms in a mathematical manner.

A formal system is, therefore, a mechanical process for constructing sentences in a language, an ideal entity that generates, in the form of *theorems* all the consequences that follow according to specified criteria (rules) from a set of initial propositions considered as primary truths (axioms). The expressions in a formal system have no inherent meaning and result from the operational possibilities specified in the system's rules. When constructing a formal system, it is generally done with the intention of representing an informal theory within that system. The goal of formalization is then to enable a precise and systematic study of the structural aspects of scientific theories.

3.2 Components

A formal system S is composed of a quadruplet :
— A finite, countable set of symbols called an alphabet,
— A recursive subset W of the set of finite sequences of S called "Set of well-formed formulas" constructed from the alphabet,
— A subset "A" of "W" called a set of axioms,
— A finite set R of rules (also called rules of inference, deduction or derivation) such that R = $\{R_1, R_2, ..., R_n\}$.

Concept of demonstration : Establishing a proof in S amounts to finding a finite sequence of wff, f_1, f_2,...., f_k such that f_i ($1 \leqslant i \leqslant k$) is either an axiom of S, or an wff obtained from another wff by application of an inference rule R.

Concept of theorem : A formula of a formal system is called a *theorem* , if it is an axiom, or if the formula is obtained by applying a rule of inference to a theorem.

A proof of a formal system is a series of formulas which are either axioms or formulas deduced from previous formulas.

Let f_1, f_2,...., f_k and g be wff, we notice f_1, f_2,...., $f_k \vdash$ g and we will read that from the formulas f_1, f_2,...., f_k we can deduce the formula g.

Example : Consider the PEU system with :
— The alphabet S = $\{p, e, u\}$,
— W is the set of wff formed from the alphabet (sequence of p,e,u),
— The set of axioms A = $\{upueuu\}$,
— The inference rules are defined as follows :
  — R_1 : If an expression of the form AeB is a theorem $\Longrightarrow$ uAeBu is a theorem.
  — R_2 : If an expression of the form AeB is a theorem $\Longrightarrow$ AueuB is a theorem.

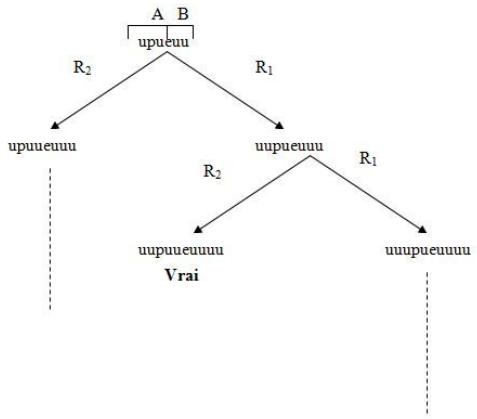
Questions

Q1 : Is "uupuueuuuu" a theorem ?

Q2 : Is "upupueuuu" a theorem ?

Solutions

Q1 : We can prove that "uupuueuuuu" is a theorem by constructing the derivation tree.



Q2 : upupueuuu is not a theorem because a theorem cannot contain more than one symbol p, and this can be proved by induction.

Reasoning by induction : Mathematical reasoning is generally done through derivation or deduction, as exemplified in Q1. However, there is also another type of reasoning called induction, particularly suitable when asked to prove properties involving n parameters. For example, attempting to prove the following equality :

1+2+3+4+....+n = $\frac{(n)(n+1)}{2}$ for any given number n, Of course, Gauss was clever as he was able to prove this formula based on the sum of the first n positive integers.
S_n = 1 + 2 + 3 + 4 +....+(n-1) + n
S_n = n + (n-1) + (n-2)+ (n-3) +....+ 2 + 1

If we do the sum column by column, we find :

$2S_n$ = (n+1)+(n+1)+(n+1)+(n+1)+(n+1) = n(n+1) hence the formula $S_n = \frac{(n)(n+1)}{2}$

We can also fall back on reasoning by induction to prove this formula and there are two demonstrations to be made :

— Show that the formula is true for n = 1, indeed 1 = (1.2)/2

— We assume that the formula is true at a certain rank n then we show that it remains true at the next rank.

for example if we take S_{n+1} = 1+2+3+4+....+n+(n+1)= S_n + (n+1) = $\frac{(n)(n+1)}{2}$ +(n+1), the formula being assumed to be true at rank n. It remains to factor and reduce to the same denominator which gives at the end $\frac{(n+1)(n+2)}{2}$ which is the formula at rank n+1.

With these two proofs done, we can assert that the property holds for any rank n.
Returning now to question 2 (Q2), to prove that "upupueuuu" is not a theorem, we consider the axiom of the system, which is "upueuu". We observe that there is only one "p" in the axiom, while the proposed word "upupueuuu" contains two "p". Moving now to level 1 in the derivation tree, we observe that there is always only one "p" in the two generated words, and so on, considering that the transition from one level to another is based on the two rules R_1 and R_2.

3.3 The formal system properties

The consistency property A formal system is consistent (i.e., non-contradictory) if one cannot prove both that "A" is a theorem and its negation as well ($\vdash$ A and $\vdash \neg$ A) or $\vdash$ A and $\neg \vdash$ A. A non-consistent system is said to be inconsistent.

The property of completeness A formal system is said to be complete if it provides a strategy for reaching the solution if it exists. That is to say, he has the ability to be able to prove all valid theorems.

The decidability property We say that a formal system is decidable if there exists a mechanical procedure to determine in finite time whether a word in the language is or is not a theorem.

The saturation of a formal system An SF is saturated if and only if by adding a formula f which is not a theorem, the SF becomes inconsistent.

3.4 Exercises

Exercise 1
1. Define a formal system such that we can produce the theorems kst, kstst, kststst,..... from an axiom k.

2.Define a formal system such that can produce the theorems ca, caba, cababa, cabababa, cababababa,..., etc. The axiom is c.

3. Define a formal system such that we can produce theorems b, ba, baa, baaa, baaaa,..., etc. The axiom is b.

Exercise 2 Consider the MIU system which includes :
- The alphabet S = $\{M, I, U\}$
- The axiom A = $\{MI\}$
- the rules :
  - R_1 : if a string ends with an I we can add a U at the end,
  - R_2 : If we have a string Mx, we can form Mxx (where x is any string),
  - R_3 :we can replace III with a U in a string,
  - R_4 : we can delete any UU pair.

1. Prove that MUIUI is a theorem.
2. Is UM a theorem ?
3. Is MU a theorem ?

Exercise 3 Let the formal system p-q
S= $\{p, /, q\}$ A= $\{pq\}$ R =
- a- $x \rightarrow /x/$
- b- $xpy \rightarrow xp/y/$ (x and y are system words)

Can we derive the following strings : $//p/q///$;$/p//q/$;$//////p///q/////////$?

Exercise 4 Consider a formal system composed of :
An alphabet $\{A, B, C, D\}$,
Axioms : D, DD,
Deductive rules :
- a- add C to the end of any string.
- b- add an A at the beginning and end of any string.

— c- replace a C with a B in a string.

Which of the following strings are theorems ? Give the proofs
DC, DCCC, DCCA, AAADAAA, AAADAAAA, AADCCCABBA.

Exercise 5 Let the formal system S (Σ, A, W, R) such that :

— Σ : it is the alphabet set such that $\Sigma = \{a, b, c\}$,
— A :it is the set of axioms which have the following form $A = \{a^{2i+1}bc^{2i-1} | i \geq 1\}$,
— W : represents the set of wffs generated from the axioms and wffs already generated,
— R : it is the set of rules such as $R = \{r_1 : (a^k bc^m, a^p bc^n) \longrightarrow a^{k+n}bc^{m+p}\}$

Q1 : Are the following formulas theorems $a^4bc^4, a^5bc^5, a^6bc^6$?

Q2 : Give the different possible forms of theorems.

Chapitre 4

Chapter 4 : Propositional logic

4.1 Introduction

In the context of classical logic, a proposition is a declarative statement (an assertion) to which we can assign the value "true" or "false," but not both : this is called propositional logic.

In the context of propositional logic, two approaches to resolution are possible : the first is a deductive approach, sometimes called proof theory, which determines whether a formula is provable or not ; the second is a semantic approach, sometimes called model theory, which involves the concept of interpreting a formula.

In this chapter, we will present these two approaches, and we conclude the chapter with the notion of completeness, which establishes the equivalence between the two approaches.

4.2 Language

4.2.1 Alphabet

Definition 1 : The alphabet of propositional logic is made up of :

— A countable set of propositional variables (or atomic formulas, or even atoms).

A propositional atom (atomic formula) is understood to be an indivisible statement, for example : it's raining, the road is wet, this cat is white, ...,

more abstractly : P, Q,..... . We use P, Q, R, P_1, P_2,....., for propositional variables.

— The connectors $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ and parentheses.

The negation $\neg A$: "non A" (i.e. A is not the case), the negation connector is a unary connector. If we take the proposition A = "this cat is white", the negation of A is not "this cat is black", but quite simply "this cat is not white".

The conjunction A $\wedge$ B : "A and B", this connector is binary.

The disjunction A $\vee$ B : "A or B", this connector is binary.

The implication A $\rightarrow$ B : "If A then B", this connector is binary.

The equivalence A $\leftrightarrow$ B : "A if and only if B", this connector is binary.

— Separators or parentheses (and).

4.2.2 Well-formed formulas

Definition 2 : the set of formulas or well-formed formulas (wff) of propositional logic is the smallest set of words built on the alphabet such as :

— If A is an atomic formula then A is a well-formed formula,

— $\neg A$ is a well-formed formula if A is a well-formed formula,

— (A $\wedge$ B) is a well-formed formula if A and B are well-formed formulas,

— (A $\vee$ B) is a well-formed formula if A and B are well-formed formulas,

— (A $\rightarrow$ B) is a well-formed formula if A and B are well-formed formulas,

— (A $\leftrightarrow$ B) is a well-formed formula if A and B are well-formed formulas,

To avoid ambiguities and minimize the use of parentheses, a priority is introduced among the connectors. After parentheses, the negation connector has the highest priority, followed by conjunction, disjunction, implication, and finally equivalence.

Example : By putting the parentheses, determine the order of priority of the connectors of the following formula :
$P \wedge \neg Q \vee R \rightarrow S \leftrightarrow X \vee Y$

Solution : $P \wedge \neg Q \vee R \rightarrow S \leftrightarrow X \vee Y = (((P \wedge \neg(Q)) \vee R) \rightarrow S) \leftrightarrow (X \vee Y)$

4.3 The proof theory

We will now focus on the deductive method, which allows us to decide whether a well-formed formula (wff) is a theorem or not. Note that an axiom is a wff assumed to be a theorem without proof, and a theorem is a wff demonstrable from the axioms using the rules of inference. The entire problem of deduction consists of generating theorems from other theorems (including the axioms), and this is accomplished through rules of inference.

4.3.1 The axioms

They are obtained from the following axiom schemas by replacing A, B, and C with any wff.

Here are the axiom schemas of propositional logic :
— **1a.** $(A \to (B \to A))$
— **1b.** $(A \to B) \to ((A \to (B \to C)) \to (A \to C))$
— **1c.** $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$
— **1d.** $(A \to B) \to ((B \to C) \to (A \to C))$
— **2.** $A \to (B \to A \wedge B)$
— **3a.** $A \wedge B \to A$
— **3b.** $A \wedge B \to B$
— **4a.** $A \to A \vee B$
— **4b.** $B \to A \vee B$
— **5.** $(A \to C) \to ((B \to C) \to (A \vee B \to C))$
— **6.** $B \to ((B \to C) \to C)$
— **7.** $A \to A$
— **8.** $(\neg A \to \neg B) \to (B \to A)$
— **9.** $(A \to B) \to ((B \to A) \to (A \leftrightarrow B))$

4.3.2 Inference rule

In propositional logic, there is a single inference rule called **modus-ponens**.

If P,
Then P $\to$ Q,
We conclude Q

If we note $\vdash Q$ this indicates that the formula Q is a theorem.

4.3.3 Concept of demonstration

A formula Q is a theorem if and only if there exists a proof with Q as its last formula.

The concept of proof (or derivation) can be extended to a "proof from assumptions." Thus, proving that a proposition P is a theorem is equivalent to seeking a proof whose last formula is P.

Example 1 : Let us demonstrate that $A \to A$ is a theorem.
Solution :
1. $\vdash A \to (A \to A)$ sh 1a
2. $\vdash (A \to (A \to A)) \to ((A \to ((A \to A) \to A)) \to (A \to A))$ sh 1b replace the B with $A \to A$ and the C with A
3. $\vdash ((A \to ((A \to A) \to A)) \to (A \to A)$ m.p 1,2
4. $\vdash A \to ((A \to A) \to A)$ sh 1a replace the B with $A \to A$
5. $\vdash A \to A$ m.p 3,4

Example 2 : Establish the following deduction :

$A \to (B \to C), B \vdash A \to C$
Solution :

1. $\vdash$ B 2nd Hyp
2. $\vdash B \to (A \to B)$ sh.1a
3. $\vdash A \to B$ m.p 1,2
4. $\vdash (A \to B) \to ((A \to (B \to C)) \to (A \to C))$ sh.1b
5. $\vdash (A \to (B \to C)) \to (A \to C)$ m.p 3,4
6. $\vdash A \to (B \to C)$ 1st Hyp
7. $\vdash A \to C$ m.p 5,6

4.3.4 Deduction theorem

Definition 3 : If P is a wff and E is a set of wffs, we say that P is a consequence of E or P is deducible from E, where the elements of E are called the hypotheses or premises of the deduction. We will denote "P is a consequence of E" by $E \vdash P$.

If E = $\{E_1, E_2, ..., E_n\}$, Then, the notation is : $E_1, E_2, ..., E_n \vdash P$. This allows stating the deduction theorem.

Deduction theorem : If E is a set of wffs and P and Q are wffs :
If $E, P \vdash Q$ then $E \vdash P \rightarrow Q$ (E, P denotes E $\cup\{P\}$ ($\cup$ it is the ensemble union)).

In the case where E = $\emptyset$, this would give us : If $P \vdash Q$ then $\vdash P \rightarrow Q$.
A particular and interesting case of the deduction theorem is the following : If $E_1, E_2, ..., E_n \vdash P$ then $E_1, E_2, ..., E_{n-1} \vdash (E_n \rightarrow P)$.

4.4 Model theory

In the deductive approach, we are interested in the notion of provable theorem, using axioms and rules. The second historically approach is based on the notion of validity and unverifiability of an wff. The objective is to be able to assign truth values to an wff.

The theory of models or the semantic aspect appeals to the interpretation notion.
In logic, an "interpretation" refers to the assignment of meanings or truth values to the various components of a logical expression, such as propositions and connectives. It establishes the correspondence between the abstract symbols of a logical language and the real-world concepts they represent.

A "model" in logic is a specific interpretation that satisfies a given set of logical expressions. In other words, a model is an assignment of truth values to propositions that makes a logical formula true. If a formula is true in a model, it is said to be satisfied by that model.

4.4.1 Interpretation and model notions

Definition 4 : An interpretation I (or a valuation) is a mapping from the set of propositional variables to the set of truth values. $\{T, F\}$.

A formula containing n atoms will have 2^n interpretations. In propositional calculus, the number of interpretations is always finite.

We can denote that an interpretation I of a wff A having n components with $I = \{V_1, V_2, ..., V_n\}$ such that V_i for i $\in \{1, .., n\}$ represents an atom or

its negation.

Definition 5 : A given interpretation I can be extended to all formulas by :

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \rightarrow B$	$A \leftrightarrow B$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

Example : Give the truth table of the following formula : $P \vee (Q \rightarrow R)$

P	Q	R	$Q \rightarrow R$	$P \vee (Q \rightarrow R)$
T	T	T	T	T
T	T	F	F	T
T	F	T	T	T
T	F	F	T	T
F	T	T	T	T
F	T	F	F	F
F	F	T	T	T
F	F	F	T	T

Definition 6 : When an interpretation I satisfies a wff F, we say that I is a model of the formula F.

Definition 7 : If E is a set of wffs $\{F_1, F_2, ..., F_n\}$,I isa *model* of E, if and only if I is a model for all formulas of E. So :
I is a model of E $\leftrightarrow$ I is a model of $F_1 \wedge F_2 \wedge ... \wedge F_n$.

Definition 8 (logical consequence) : A formula A is a *logical consequence* of $A_1, A_2, ..., A_n$ note $A_1, A_2, ..., A_n \models A$ if and only if any model of $A_1, A_2, ..., A_n$ is a model of A.

4.4.2 validity concept

Definition 9 : A wff is called *valid* (or tautology) if and only if it is true for all its interpretations, independently of the truth value of the atoms that

compose it ; In other words, it is the combination of connectives that makes it a tautology.

Definition 10 : A propositional logic formula is said to be *satisfiable* if there exists at least one interpretation I for which the formula is true. Similarly, we will say that a set E of propositional logic formulas is satisfiable if and only if there exists an interpretation I that is a model of E.

Definition 11 : A propositional logic formula is said to be *unsatisfiable* if and only if it is false for all its interpretations.

Theorem 1 : A propositional logic formula Q is a valid consequence of P (or logically follows from P) if and only if any interpretation satisfying P also satisfies Q.
$P \models Q$ will be used to denote that Q is a valid consequence of P.

4.5 Equivalence of two formulas and normal forms

Theorem 2 : Two propositional logic formulas P and Q are said to be equivalent (or P is equivalent to Q) if and only if the truth values of P and Q are the same for every interpretation of P and Q.

Example : The formulas P and Q are equivalent.
P = $A \vee B$ and Q = $B \vee A$
Furthermore, if the sets of propositional symbols in P and Q are not the same, it seems inappropriate to use this formula ; we will then say to establish equivalence :
Two propositional logic formulas P and Q are equivalent if and only if we have both : $P \models Q$, and $Q \models P$.This is equivalent to saying : $P \equiv Q$ is valid ($P \equiv Q$ is an abbreviation for 'P is equivalent to Q').

Theorem 3 : For all formulas P, Q and R, the following pairs of formulas are equivalent :

— **Idempotence :** $(P \vee P) \equiv P$ and $(P \wedge P) \equiv P$.

— **Commutativity :** $(P \vee Q) \equiv (Q \vee P)$ and $(P \wedge Q) \equiv (Q \wedge P)$.

— **Associativity :** $((P \vee Q) \vee R) \equiv (P \vee (Q \vee R))$ and $((P \wedge Q) \wedge R) \equiv (P \wedge (Q \wedge R))$.

— **Absorption :** $(P \vee (P \wedge R)) \equiv P$ and $(P \wedge (P \vee R)) \equiv P$.

— **Distributivity :** $(P \wedge (Q \vee R)) \equiv ((P \wedge Q) \vee (P \wedge R))$ and $(P \vee (Q \wedge R)) \equiv ((P \vee Q) \wedge (P \vee R))$.

— **Complementarity :** $\neg\neg P \equiv P$.

— **Morgan's Laws :** $\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$ and $\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$.

— **Tautology :** $(F \vee G) \equiv F$ if F is a tautology.
$(F \wedge G) \equiv G$ if F is a tautology.

— **unsatisfiability :** $(F \vee G) \equiv G$ If F is unsatisfiable.
$(F \wedge G) \equiv F$ If F is unsatisfiable.

This theorem, used in conjunction with theorem 2, allows us to simplify the formulas.

Example : Let F = $((A \vee \neg(B \wedge A)) \wedge (C \vee (D \vee C)))$, simplify the formula F.

1. $((A \vee \neg(B \wedge A)) \wedge (C \vee (D \vee C))$
2. $\equiv ((A \vee (\neg B \vee \neg A)) \wedge (C \vee (D \vee C))$ 2nd Morgan's law
3. $\equiv ((A \vee (\neg A \vee \neg B)) \wedge (C \vee (D \vee C))$ commutativity of $\vee$
4. $\equiv ((A \vee \neg A) \vee \neg B) \wedge (C \vee (D \vee C))$ associativity of $\vee$
5. $\equiv (A \vee \neg A) \wedge (C \vee (D \vee C))$ 1st tautology.
6. $\equiv (C \vee (D \vee C))$ 2nd tautology.
7. $\equiv (C \vee (C \vee D))$ commutativity of $\vee$
8. $\equiv (C \vee C) \vee D)$ associativity of $\vee$
9. $\equiv (C \vee D)$ idempotence.

Definition 12 : A literal is an atom or the negation of an atom.

Definition 13 : A conjunction of literals is a formula composed of literals connected by the operator $\wedge$.

Definition 14 : A disjunction of literals is a formula composed of literals connected by the operator $\vee$.

Definition 15 : A formula is called under **conjunctive normal form** if and only if, it is a conjunction of disjunction of literals.

Definition 16 : A formula is called under **disjunctive normal form** if and only if, it is a disjunction of conjunction of literals.

Theorem 4 : For any propositional calculus formula, we can find at least one equivalent formula in conjunctive normal form, and another in disjunctive normal form. There exists a transformation procedure which for any formula f, transforms this formula into conjunctive or disjunctive normal form.

This procedure consists of :

— Eliminate the logical operators $\leftrightarrow$ and $\rightarrow$ by using the laws :
$P \leftrightarrow Q = (P \rightarrow Q) \wedge (Q \rightarrow P)$
$P \rightarrow Q = \neg P \vee Q$

— Reduce the scope of negation through the repetitive application of complementarity and Morgan's laws.

— Use the laws of distributivity, and the equivalences between formulas.

Example : find the CNF (conjunctive normal form) of the following formula : $\neg((A \wedge B) \vee (C \wedge \neg D))$.

Solution :

1. $\neg((A \wedge B) \vee (C \wedge \neg D))$
2.$\equiv (\neg(A \wedge B) \wedge \neg(C \wedge \neg D))$ Morgan's 1st law
3. $\equiv ((\neg A \vee \neg B) \wedge (\neg C \vee D))$ it is in CNF form.

4.6 Equivalence between the two approaches

In conclusion, we give some results that have been established for propositional logic, notably the equivalences between the deductive approach and the semantic approach.

- **The propositional logic safety :** The propositional logic is safe, we also say correct, in the sense that, if a formula P is demonstrable (deductive approach) then it is valid (semantic approach).
  If $\vdash P$ then $\models P$.
- **The propositional logic completeness :** The propositional logic is complete : if a WFF is valid, then it is a theorem.
  If $\models P$ then $\vdash P$.
- **The propositional logic decidability :** The propositional logic is decidable. There exists an algorithm making it possible to decide in a finite number of steps, for any input formula, whether this formula is or is not a theorem : such an algorithm is a decision procedure.
- **The propositional logic coherence :** the propositional logic is coherent ; For any wff P, we cannot deduce both P and $\neg P$.

4.7 Exercises

Exercise 1
1. Let A be the following proposition : « All men are bearded ». Check the correct formulations of proposition $\neg A$.

☐ « Not all men are bearded.»
☐ « No man is bearded.»
☐ « There exists a man who is not bearded.»
☐ « There is at least one man who is not bearded.»
☐ « There is only one man who is not bearded.»

2. Here is a list of simple propositions A and B, the truth values of which you know. In each case, express the truth value of proposition $A \wedge B$.

T	F	
☐	☐	A : « Paris is the capital of France.» and B : « 1+1 = 2».
☐	☐	A : « A cat has five legs.» and B : « A square has four sides.».
☐	☐	A : « A right-angled triangle has a right angle.» and B : « Two parallel lines intersect at a point.».
☐	☐	A : « 3 * 8 = 32» and B : « Paris is the capital of France».
☐	☐	A : « Berlin is the capital of Spain.» and B : « A right-angled triangle has three equal sides.».
☐	☐	A : « A fly can fly.» and B : « Canada is a country in the North American continent.».

3.Here is a list of simple propositions A and B, the truth values of which you know. In each case, express the truth value of proposition $A \vee B$.

T	F	
☐	☐	A : « Paris is the capital of France.» and B : « 1+1 = 2».
☐	☐	A : « A cat has five legs.» and B : « A square has four sides.».
☐	☐	A : « A right-angled triangle has a right angle.» and B : «Two parallel lines intersect at a point.».
☐	☐	A : « 3 * 8 = 32» and B : « Paris is the capital of France.».
☐	☐	A : « Berlin is the capital of Spain.» and B : « A right-angled triangle has three equal sides.».
☐	☐	A : «A fly can fly.» and B : « Canada is a country in the North American continent.».

Exercise 2 Are the following expressions a well-formed formulas ?
1. $p \wedge \neg q$, 2. $p \vee \vee r$, 3. $(p \vee (\neg p))$, 4. $(p \vee \neg p)$.

Exercise 3 Give the truth tables for the following formulas, and indicate the equivalences between these formulas.
1. $\neg(p \wedge q)$, 2. $\neg p \vee \neg q$, 3. $\neg(p \vee q)$, 4. $\neg p \wedge \neg q$, 5. $p \vee (p \wedge q)$, 6. $p \wedge (p \vee q)$, 7. p

Exercise 4 Let p and q be two propositional variables denoting respectively« 'It is cold » and « 'It is raining ». Write a simple sentence corresponding to each of the following statements :
1. $\neg p$, 2. $p \wedge q$, 3. $p \vee q$, 4. $q \vee \neg p$, 5. $\neg p \wedge \neg q$, 6. $\neg\neg q$

Exercise 5 Let p be : « 'Eric reads "Match" » , q be «'Eric reads "L'express" » and r be : « Eric reads "Les Echos" ».
Provide a logical formula for each of the following sentences :
1. Eric reads Match or L'Express, but does not read Les Echos ».
2. Eric reads Match and L'Express, or he reads neither Match nor Les Echos.
3. It is not true that Eric reads Match but not Les Echos.
4. It is not true that Eric reads Les Echos or L'Express but not Match.

Exercise 6 Enigma : Three colleagues, Ahmed, Ali, and Mostafa, have lunch together every working day. The following statements are true :

1. If Ahmed orders a dessert, Ali orders one to,
2. Every day, either Mostafa or Ali, but not both, orders a dessert,
3. Every day, either Ahmed or Mostafa, or both, order a dessert,
4. If Mostafa orders a dessert, Ahmed does the same.

Questions :

1. Express the data of the problem as propositional formulas.
2. What can be deduced about who orders a dessert ?
3. Can we reach the same conclusion by removing one of the four statements ?

Exercise 7 Sheffer's connector is defined, denoted as | (Sheffer bar) which is the NAND by p| q $\equiv \neg(p \wedge q)$.

1. Give the truth table for the formula (p | q).

2. Give the truth table for the formula ((p | q)|(p | q)).
3. Express the connectors $\neg$, $\vee$ and $\rightarrow$ by using the Sheffer bar.

Exercise 8 Establish the truth tables for the following formulas and determine if they are valid, satisfiable, or unsatisfiable.

a. $(\neg P \wedge \neg Q) \rightarrow (\neg P \vee R)$
b. $P \wedge (Q \rightarrow P) \rightarrow P$
c. $(P \vee Q) \wedge \neg P \wedge \neg Q$
d. $(P \rightarrow Q) \wedge (Q \vee R) \wedge P$
e. $((P \vee Q) \rightarrow R) \leftrightarrow P$

Exercise 9 Find the disjunctive normal forms :
a. $(A \vee B \vee C) \wedge (C \vee \neg A)$
b. $(A \vee B) \wedge (C \vee D)$
c. $\neg((A \vee B) \rightarrow C)$

Exercise 10 Find the conjunctive normal forms :
a. $(A \vee B) \rightarrow (C \wedge D)$
b. $(A \vee (\neg B \wedge (C \vee (\neg D \wedge E))))$
c. $A \leftrightarrow (B \wedge \neg C)$

Exercise 11 Prove that the following formulas are theorems :
a. $\vdash A \leftrightarrow A$,Knowing that $A \rightarrow A$ should not be taken as an axiom.
b. $\vdash \neg B \rightarrow (B \rightarrow A)$

Exercise 12 Establish the following deductions :
a. $A \rightarrow (B \rightarrow C), A \wedge B \vdash C$
b. $A \rightarrow (B \rightarrow C), B \vdash A \rightarrow C$
c. $A, B \wedge C, A \wedge C \rightarrow E \vdash E$
d. $E, E \rightarrow (A \wedge D), D \vee F \rightarrow G \vdash G$

The axioms of propositional logic are :

- **1a.** $(A \rightarrow (B \rightarrow A))$
- **1b.** $(A \rightarrow B) \rightarrow ((A \rightarrow (B \rightarrow C)) \rightarrow (A \rightarrow C))$
- **1c.** $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
- **1d.** $(A \rightarrow B) \rightarrow ((B \rightarrow C) \rightarrow (A \rightarrow C))$
- **2.** $A \rightarrow (B \rightarrow A \wedge B)$
- **3a.** $A \wedge B \rightarrow A$

— **3b.** $A \wedge B \to B$
— **4a.** $A \to A \vee B$
— **4b.** $B \to A \vee B$
— **5.** $(A \to C) \to ((B \to C) \to (A \vee B \to C))$
— **6.** $B \to ((B \to C) \to C)$
— **7.** $A \to A$
— **8.** $(\neg A \to \neg B) \to (B \to A)$
— **9.** $(A \to B) \to ((B \to A) \to (A \leftrightarrow B))$

Chapitre 5

Chapter 5 : First order logic (predicate logic)

5.1 Introduction

The aim of predicate logic is to generalize the propositional logic. We can consider a predicate as a general statement in which variables appear. For example : “The lecture hall X is large” “If X is the father of Y and Z then Y and Z are brothers”.

If we replace all the predicate variables with defined values we obtain a proposition to which we can associate an interpretation (true, false). Thus, X = A1, in the first example gives "The lecture hall A1 is large". A predicate therefore represents a class of propositions.

By introducing quantifiers we can represent that a statement is true for all possible values or that there exists at least one value of the variables which makes the statement true. For example : "For all X, if X is a man then X is mortal".Since variables can take their values in infinite sets, quantifiers allow us to state facts corresponding to an infinity of propositions.

5.2 Language

5.2.1 Alphabet

The alphabet of predicate logic consists of :

— A countable set of predicate symbols with 0, 1, or multiple arguments, denoted as p, q, r, ..., man, mortal, father,....
— A countable set of object variables (or individual variables), denoted as x, y, z, x_1, x_2,.....
— A countable set of functions with 0, 1, or multiple arguments, denoted as f, g, ..., father-of,....
— Quantifiers $\forall$ and $\exists$.
— Connectors $\neg, \wedge, \vee, \rightarrow$ As well as the propositional logic parentheses.

Note

Functions with 0 arguments are called constants (often denoted as a, b, ..., Socrates,...).
Predicates with 0 arguments are propositional variables.

Example : Express in the formal language of first-order logic the following assertions :

All cats are black : $\forall x(Cat(x) \rightarrow Black(x))$
Some cats are black : $\exists x(Cat(x) \wedge Black(x))$
No cat is black : $\forall x \neg(Cat(x) \wedge Black(x))$
In the natural numbers domain, for every natural number, there exists a prime number greater than it : $\forall x \exists y(Pri(y) \wedge Greater(y, x))$

5.2.2 Definitions

A term : the set of terms is the smallest set of words built on the alphabet of predicate logic such that :

— Every variable is a term,
— f($t_1, ..., t_n$) is a term if "f" is a function with n arguments and $t_1, ..., t_n$ are terms.

Atomic formula : If P is a predicate with n arguments and $t_1, ..., t_n$ are terms then P($t_1, ..., t_n$) is an atomic formula. When n = 0, the atomic formula is a proposition.

Well-formed formula : An atomic formula is a wff.
If P and Q are wffs and x a variable, then : $\neg P$, $P \wedge Q, P \vee Q, P \rightarrow Q, P \leftrightarrow Q, (\forall x)P, et(\exists x)P$ are wffs.
The priority between connectives is the same as in propositional logic ; the quantifiers "$\forall$" and "$\exists$" have the same priority as "$\neg$".

Literal : If L is an atomic formula, (L) and $(\neg L)$ are called literals. L and $\neg L$ are called complementary literals.

5.2.3 The quantifier scope

A variable is **bound** in a formula if and only if it is in the scope of a quantifier. In wffs $(\forall x)$P, and $(\exists x)$P, P is the quantifiers **scope** .

A variable that is not bound in a formula is called **free**. A variable can therefore be free and bound in the same formula.

In the wff : $(P(x) \vee \forall x Q(x))$, x is free in P(x), but it is bound in Q(x).

A **closed** wff is a wff which does not contain free variables (it only contains constants or variables quantified universally or existentially), otherwise it is **open**.

If P is a wff and t a term, t is said to be free for x in P, if and only if when we replace a free occurrence of x by t no variable of t becomes bound. In the following, we only consider formulas without free variables.

The closure of a wff P, is defined as the closed wff obtained from P, by prefixing it with universal quantifiers relating to the free variables in P.

Example : The closure of : $P(x, y, z) \rightarrow (\exists y)Q(x, y, z)$ is
$\forall x \forall y \forall z(P(x, y, z) \rightarrow (\exists y)Q(x, y, z))$

5.2.4 The congruent formulas

Consider the formula : $\forall x(P(x) \wedge \exists x Q(x, z) \rightarrow \exists y R(x, y)) \vee Q(z, x)$ (1)

In the $\exists x Q(x,z)$ part, each x is related by $\exists x$, which can be indicated by attaching an index 1 to each of the x's to show that they belong together. We will do the same for the other variables. When putting the indices we must always start from the inside, proceeding according to the order followed in the formula construction. For the sake of normality, we choose to start numbering at each step with the leftmost quantifier. Consider formula (1) :

1- $\forall x(P(x) \wedge \exists x Q(x,z) \rightarrow \exists y R(x,y)) \vee Q(z,x)$
2- $\forall x_3(P(x_3) \wedge \exists x_1 Q(x_1,z) \rightarrow \exists y_2 R(x_3,y_2)) \vee Q(z,x)$

The occurrences left without indices are free. By erasing the bound variables, we obtain :

3-$\forall 3(P(3) \wedge \exists 1 Q(1,z) \rightarrow \exists 2 R(3,2)) \vee Q(z,x)$

Consider formula (2) :

1- $\forall y(P(y) \wedge \exists x Q(x,z) \rightarrow \exists z R(y,z)) \vee Q(z,x)$
2- $\forall y_3(P(y_3) \wedge \exists x_1 Q(x_1,z) \rightarrow \exists z_2 R(y_3,z_2)) \vee Q(z,x)$
3- $\forall 3(P(3) \wedge \exists 1 Q(1,z) \rightarrow \exists 2 R(3,2)) \vee Q(z,x)$

This shows that formulas (1) and (2) are **congruent**.
There is also another technique to detect congruent formulas, this technique is based on syntax trees and it goes as follows :

1. construct the syntax trees of the appropriate formulas,
2. For each tree, it is necessary to number the universally or existentially quantified variables starting from the leaves of the tree and from left to right, then ascend to the root,
3. construct the formulas by replacing the variables with their numbers,
4. If the resulting formulas are equivalent, then the initial formulas are said to be congruent.

5.2.5 Substitution and instantiation

The substitution operation consists of replacing certain free variables of an open formula F by terms.

If F is a formula where $x_1, x_2, ..., x_n$ are free variables, and σ the substitution $(t_1/x_1, ..., t_n/x_n)$, Fσ denotes the obtained formula by replacing all

occurrences of x_i in F by t_i such that i = 1 to n.

Example : F = $Q(x, y_1) \leftrightarrow \forall x(R(x, z_1) \vee S(x, y_1))$
$\sigma = (z_8/x, g(a, b)/y_1)$
F$\sigma = Q(z_8, g(a, b)) \leftrightarrow \forall x(R(x, z_1) \vee S(x, g(a, b)))$

We will say that a substitution instantiates x if it replaces x by a term where no variable appears. Thus, the substitution $(z_8/x, g(a, b)/y_1)$ instantiates y_1 but not x.

5.3 The deductive method : Proof theory

In the same way as for propositional logic, we give the axioms and inference rules of first order logic.

5.3.1 The axioms

We keep the propositional logic axioms and we add :

— The universal schema : $\forall x P(x) \rightarrow P(a)$
— The existential schema : $P(a) \rightarrow \exists x P(x)$

5.3.2 The inference rules

Modus Ponens : P,$P \rightarrow Q \vdash Q$

The universal rule : It allows transitioning from the form $C \rightarrow A(x)$ to the form $C \rightarrow \forall x A(x)$.

The existential rule : It allows transitioning from the form $A(x) \rightarrow C$ to the form $\exists x A(x) \rightarrow C$ where C contains no free occurrences of x.

Deduction theorem : If E is a set of closed formulas and P and Q are closed formulas : If E, P $\vdash$ Q, then E $\vdash P \rightarrow Q$ (E, P denotes E $\cup$ { P }).

Example : Establish the following deduction : $\forall y(R \rightarrow P(y)) \vdash R \rightarrow \forall x P(x)$

1- $\forall y(R \rightarrow P(y))$ Hyp
2- $\forall y(R \rightarrow P(y)) \rightarrow (R \rightarrow P(y))$ sh $\forall$
3-$R \rightarrow P(y)$ m.p 1,2
4- $R \rightarrow \forall y P(y)$ rule $\forall$
5- $(R \rightarrow \forall y P(y)) \rightarrow ((R \rightarrow (\forall y P(y) \rightarrow \forall x P(x))) \rightarrow (R \rightarrow \forall x P(x)))$ sh.1b
6- $(R \rightarrow (\forall y P(y) \rightarrow \forall x P(x))) \rightarrow (R \rightarrow \forall x P(x))$ m.p 4,5
7- $\forall y P(y) \rightarrow P(x)$ sh.$\forall$
8- $\forall y P(y) \rightarrow \forall x P(x)$ rule$\forall$
9- $(\forall y P(y) \rightarrow \forall x P(x)) \rightarrow (R \rightarrow (\forall y P(y) \rightarrow \forall x P(x)))$ sh.1a
10- $R \rightarrow (\forall y P(y) \rightarrow \forall x P(x))$ m.p 8,9
11- $R \rightarrow \forall x P(x)$ m.p 6,10

5.4 The semantic approach

We follow the same approach as that adopted in the presentation of propositional logic. After determining whether a formula F is provable or not (using the deductive approach), we now ask whether a formula WFF is valid or not. We cannot proceed as in propositional logic, and one of the challenges of first-order logic will be to find procedures for establishing that a formula is unverifiable.

5.4.1 Interpretation concept

In propositional logic, we noted that the interpretation of a formula consists of assigning truth values true or false to each of the atoms and evaluating the whole based on the truth tables of the different connectors. In first-order logic, an interpretation I of a formula P is defined by :

— A non-empty set D called the interpretation domain,
— A correspondence between the constants and the elements of D,
— A correspondence between each n-ary function and a mapping from D^n to D,
— A correspondence between each n-ary predicate and a mapping from D^n to $\{V, F\}$.

Interpretation of variables : To each variable, we associate an element of D.

Interpretation of terms :

— The variables are interpreted as described above,

— To each constant, we associate an element of D,

— If $t'_1, t'_2, ..., t'_n$ are the interpretations of the terms $t_1, t_2, ..., t_n$ and f' is the interpretation of f, then the element $f'(t'_1, t'_2, ..., t'_n)$ of D is the interpretation of the term $f(t_1, t_2, ..., t_n)$.

Interpretation of formulas in $\{True, False\}$:

— If the formula $P(t_1, t_2, ..., t_n)$ is an atom, its truth value is the truth value of $P'(t'_1, t'_2, ..., t'_n)$ where P' is the function associated with P and t'_i is the interpretation of the term t_i,

— If the formula has the form $\neg P, P \wedge Q, P \vee Q, P \rightarrow Q, P \leftrightarrow Q$, then the truth value of this formula is determined by the truth table of the respective logical connectives,

— If the formula has the form $\exists x P$, its truth value is "true" if there exists an element d in D such that P is true according to the interpretations of variables, terms, and formulas, with x replaced by d. Otherwise, the truth value is "false",

— If the formula has the form $\forall x P$, then its truth value is "true" if for every element d in D, P is true with respect to I, with x replaced by d (denoted as P(x/d)), otherwise the truth value is "false".

The truth value of a closed formula does not depend on the interpretation of variables. Thus, one can talk about the truth value of a closed formula.

Unlike propositional logic, there are in general an infinite number of interpretations for a formula of first-order predicate logic ; There is therefore no equivalent to the notion of a truth table.

Example : Consider the formula $F = \forall x(\exists x P(x) \rightarrow P(x)) \wedge P(y)$, to be interpreted over the domain D = $\{1, 2\}$.

First, we seek to identify the free variables and the bound variables. Thus, we have :
$\forall x_2(\exists x_1 P(x_1) \rightarrow P(x_2)) \wedge P(y)$, The variable y is free in the formula, so we provide interpretations relative to x :

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
1	T	T	F	F
2	T	F	T	F

The truth table is :

P(x)	y	$\forall x(\exists x P(x) \rightarrow P(x)) \wedge P(y)$
f_1	1	T
f_1	2	
f_2	1	
f_2	2	
f_3	1	F
f_3	2	
f_4	1	
f_4	2	

The truth value of the first row is as follows :

$$\forall x \quad T \left\{ \begin{array}{l} x = 1 \quad T \quad \exists x \quad T \left\{ \begin{array}{lll} f_1(1) & T & \\ or & & \\ f_1(2) & T & \rightarrow f_1(1) \quad T \end{array} \right. \\ and \\ x = 2 \quad T \quad \exists x \quad T \left\{ \begin{array}{lll} f_1(1) & T & \rightarrow f_1(2) \quad T \\ or & & \\ f_1(2) & T & \end{array} \right. \end{array} \right\} \wedge f_1(1) \quad T$$

5.4.2 validity and model notions

Let F be a closed first-order logic formula, and $S = \{P_1, P_2, ..., P_n\}$ be a finite set of closed first-order logic formulas.

Definition 1 : F is **verifiable** if and only if there exists an interpretation I such that the truth value of F relative to I is "true".

If F is true for interpretation I, we say that I is a model for F, and that I satisfies F. S is **verifiable** if there exists an interpretation I that is a model for each formula in S. Thus : I is a model for S if and only if I is a model for $P_1 \wedge P_2 \wedge .... \wedge P_n$; otherwise, S is **unverifiable**.

Definition 2 : F is **valid** if and only if every interpretation I for F satisfies F. S is valid if each of its interpretations is a model for S.

Definition 3 : F is called a **valid consequence** of S if for every interpretation I, if I is a model of S, then I is a model for F.

F is a **logical consequence** of S if and only if $P_1 \wedge P_2 \wedge .... \wedge P_n \rightarrow F$ is valid.

Theorem 2 : F is a valid consequence of S if and only if $S \cup \{\neg F\}$ is unverifiable.

5.5 The equivalence between the two approaches

Similar to propositional logic, we conclude the chapter with the equivalence between the deductive and the semantic approaches.

Soundness and completeness : First-order logic is sound, if a formula is provable, then it is valid.

$$\vdash P \rightarrow \models P$$

First-order logic is complete. A formula P is a theorem if and only if it is valid (completeness theorem established by Gödel).

$$\models P \leftrightarrow \vdash P$$

The completeness of predicate logic is significant, as it allows us to demonstrate the validity of a formula using proof theory.

Decidability : Predicate logic is **semi-decidable**, as there exists a proof procedure to establish that a formula is a theorem (by applying inference rules to axioms and theorems). For formulas that are not theorems, these procedures do not provide answers in a finite number of steps.

5.6 The clausal form of a formula

In propositional logic, for any formula, we can find an equivalent formula in CNF form and another in DNF form. In predicate logic, to transform a WFF "F" into CNF, the problem is to perform certain modifications on F while preserving its verifiability.
The transformation algorithm is as follows :

conversion into prenex form A WFF is said to be in prenex form if it has the following structure : $Q_1x_1Q_2x_2....Q_nx_nM$, where each Q_i, i = 1,..,n is either a $\forall$ or $\exists$ quantifier and M is a WFF that does not contain quantifiers. The Q_ix_i are called prefixes and M is the matrix.
The transformations into prenex form are as follows :

1. $\neg\forall xF$ into $\exists x\neg F$,
2. $(\forall xF) \wedge G$ into $\forall x(F \wedge G)$,
3. $(\forall xF) \vee G$ into $\forall x(F \vee G)$,
4. $(\forall xF) \rightarrow G$ into $\exists x(F \rightarrow G)$,
5. $G \wedge (\forall xF)$ into $\forall x(G \wedge F)$,
6. $G \vee (\forall xF)$ into $\forall x(G \vee F)$,
7. $G \rightarrow (\forall xF)$ into $\forall x(G \rightarrow F)$,
8. $\neg\exists xF$ into $\forall x\neg F$,
9. $(\exists xF) \wedge G$ into $\exists x(F \wedge G)$,
10. $(\exists xF) \vee G$ into $\exists x(F \vee G)$,
11. $(\exists xF) \rightarrow G$ into $\forall x(F \rightarrow G)$,
12. $G \wedge (\exists xF)$ into $\exists x(G \wedge F)$,
13. $G \vee (\exists xF)$ into $\exists x(G \vee F)$,
14. $G \rightarrow (\exists xF)$ into $\exists x(G \rightarrow F)$,

The variable x must have no free occurrences in G ; otherwise, x must be renamed to a new variable not appearing freely in the formulas F and G. This step can be done before the conversion into prenex form step.

The elimination of existential quantifiers (Skolemization) The Skolem form of a formula F put into prenex form is obtained by eliminating all existential quantifiers $\exists$ and the corresponding variable name from the prefix, replacing each quantified variable x_i with $\exists$ by $f_i(x_{j1}, x_{j2}, .., x_{jn})$ where $x_{j1}, x_{j2}, .., x_{jn}$ are variables quantified by $\forall$ placed before the $\exists x_i$.
The functional symbols f_i introduced are different from those used in the formula.
In the case where there are no $\forall$ preceding the $\exists x_i$, the symbol introduced is a zero-ary functional symbol, meaning a constant.

Universal quantifier elimination The universal quantifiers $\forall$ can become implicit without any change.

Elimination of implication and equivalence symbols We replace $(P \leftrightarrow Q)$ with $(P \rightarrow Q) \wedge (Q \rightarrow P)$ and $(P \rightarrow Q)$ with $(\neg P \vee Q)$.

Reducing the negation scope By applying the Morgan's laws.

Conversion to CNF form By applying the propositional logic properties.

5.7 Exercises

Exercise 1 Model the following sentences in predicate logic, specify the vocabulary.

— All students love logic.
— Not all students like a module.
— Students who get a good grade in logic are the best.

Exercise 2 Consider the following sentences :
a- All men are mortal.
b- Socrate is a mortal.
c- Socrate is a man.

— Identify the predicates.
— Express in first order logic a, b and c.
— can we deduce the statement b from a and c ? justify.

Exercise 3 Consider the following statements :

1. People who have influenza A should take Tamiflu.
2. People who have a fever and cough have influenza A.
3. Those who have a temperature above 38 have a fever.
4. Mohamed coughs and has a temperature above 38.
5. Mohamed has to take Tamiflu.

Model the above statements in first-order logic using the following predicates :

— influenza (x) : x has influenza A.
— take (x,y) : x must take y.
— fever (x) : x has a fever.
— cough (x) : x coughs.
— temp (x, t) : x has the temperature t.
— sup (x, y) : x is greater than y.

use also the following constants : 38, Mohamed, Tamiflu.

Exercise 4 Which of the formulas below are congruent ? Justify the answer (by applying the syntactic tree method).

— $F_1 : \forall x \exists y (\forall y P(x, y) \rightarrow \exists x Q(x, y))$.
— $F_2 : \forall v \exists z (\forall u P(z, u) \rightarrow \exists u Q(u, v))$.
— $F_3 : \forall z \exists x (\forall x P(z, x) \rightarrow \exists z Q(z, x))$.

Exercise 5 Say if the following formulas are true or false, knowing that the domain D = $\{c1, c2, c3\}$, the subsets of the domain are :

— R = $\emptyset$,

— P = $\{c1, c3\}$,

— Q = $\{c1, c2, c3\}$.

and the interpretations of the constants are :

I(a) = c1.

The formulas are :

1. $\forall x \neg Q(x)$, **2.** $\forall x P(x)$, **3.** $\forall x(P(x) \rightarrow Q(x))$, **4.** $\forall x(P(x) \wedge Q(x))$, **5.** $\exists x(Q(x) \wedge \neg P(x))$, **6.** $\exists x(\neg Q(x) \rightarrow P(x))$, **7.** $\forall x Q(x) \rightarrow \neg(\exists x R(x))$, **8.** $P(a) \rightarrow R(a)$. Justify your answer.

Exercise 6 Establish the truth table of the following formulas : (knowing that the interpretation domain is D = $\{1, 2\}$)

a- $\forall x(P(x) \rightarrow \exists x Q(x))$.

b- $\forall x(P(x, y) \wedge \exists x P(x))$.

Exercise 7

a- Establish the following deductions :

— $\forall x \forall y A(x, y) \vdash A(x, y)$.

— $\forall x(P(x) \rightarrow Q(x)), \forall x P(x) \vdash \forall x Q(x)$.

— $P(a), \forall x(P(x) \rightarrow Q(x)) \vdash Q(a)$.

— $\forall x S(x) \wedge \forall x R(x) \vdash \exists x(S(x) \wedge R(x))$.

b- Demonstrate that : $\vdash \forall x P(x) \rightarrow \exists x P(x)$.

5.8 Bibliographical note

The definitions in Chapter 2 were extracted from the book of Jean Yves Girard, translated from English by Julien Basch and Patrice Blanchard [Gir99]. Chapter 3 incorporates some notes from the course of MrsKempf [Kem07] "Formal Logic", from the book of Salem "Introduction to Formal Logic" [Sal91], and from the book of Gérard Chazal [Cha96], some exercises were extracted from the tutorial series of Mrs. Bahi [Bah06].

The definitions in Chapters 4 and 5 were extracted from the books "Mathematical Logic and Formalized Theories" [L.R74], "Mathematical Logic" [KCD03], and "Elements of Mathematical Logic : Model Theory" [KK67].

Bibliographie

[Bah06] H. Bahi. *systeme formel serie de TD*. Université Badji Mokhtar annaba, 2006.

[Cha96] Gérard Chazal. *Elements de logique formelle, édition Hermes*. Hermès Lavoisier, 1996.

[Gir99] Jean Ives Girard. *La machine de turing*. Seuil, 1999.

[KCD03] J.L. Krivine, R. Cori, and D.Lascar. *Logique mathématique 1- Calcul propositionel, Algebre de Boole, Calcul des prédicats*. DUNOD Paris, 2003.

[Kem07] Kempf. *Logique formelle*. 2007.

[KK67] G. Kreisel and J.L. Krivine. *Elements de la logique mathématique théorie des modèles*. DUNOD Paris, 1967.

[L.R74] Robert L.Rogers. *Mathematical logic and formalized theorie*. Elsevier, 1974.

[Sal91] Jean Salem. *Introduction à la logique formelle et symbolique*. Nathan université, 1991.