Chap 2

PROPOSITIONAL LOGIC

In Chap 1, we have proved a wide range of statements. For example, we proved the correctness of the sequent $\vdash (\phi \to (\psi \to (\phi \wedge \psi)))$ (Exp 1.4.3).
Strictly this is not correct because ϕ and ψ are not statements; they are variables ranging over statements. What we proved in Exp 1.4.3 was that

Every statement of the form $(\phi \to (\psi \to (\phi \wedge \psi)))$ is true.

Likewise the natural deduction 'proof' of $(\phi \to (\psi \to \phi))$ is strictly not a proof of a statement; it is a pattern of infinitely many proofs of different statements. Such patterns are called formal proofs.

So we are studying patterns that infinitely many different statements could have. These patterns are the real subjects matter of mathematical logic. To study them closer, we work with formal languages which are designed to express the patterns that are important for arguments.

To design a language, we need to describe three things.

- The lexicon is the set of 'words' of the language. In a formal language the words are called symbols.
- The syntax describes how the words are built up into sentences. A sentence of a formal language is called a formula.
- The semantics is the correlation between symbols and their meanings. The symbols fall into two classes:
  1) 'and', 'not', 'for all', etc. These are 'pattern' words.
  2) Symbols with no fixed meaning, but we have ways of attaching temporary meanings to them.

In the formal semantics of logical languages, we give formal necessary and sufficient conditions for a statement to be true, depending on how the temporary meanings are assigned.

2.1. LP, the language of propositions

The first step in constructing LP is to choose a set of symbols that can stand for statements. We call this set the signature. We should avoid putting into it symbols that already have other uses. For example, we should certainly not put into it any of the symbols

(2.1) $\quad \wedge \quad \vee \quad \rightarrow \quad \leftrightarrow \quad \neg \quad \bot$

(we call these the truth function symbols) and the parentheses

(2.2) $\quad (\quad)$

The symbols in the signature are called propositional symbols.

The usual custom is to choose propositional symbols to be lower-case letters near p, sometimes with indices or dashes, for example,

(2.3) $$p \quad q \quad r \quad p_1 \quad q_{215} \quad r' \quad r''''$$

In computer science applications of logic one often meets propositional symbols that are several letters long, like MouseEvent. The infinite set of symbols

(2.4) $$p_0, p_1, p_2, \ldots$$

will serve as a default signature in this chapter.

For each choice of signature σ, there is a dialect $LP(\sigma)$ of LP.

<u>Definition 2.1.1</u>

For each signature σ:

(a) The <u>lexicon</u> of $LP(\sigma)$ is the set of symbols consisting of the truth function symbols (2.1), the parentheses (2.2) and the symbols in σ.

(b) An <u>expression</u> of $LP(\sigma)$ is a string of one or more symbols from the lexicon of $LP(\sigma)$. The <u>length</u> of the expression is the number of occurrences of symbols in it. (once the same symbol will occur more than once.)

Metavariables

It will be useful to be able to say things like 'If ϕ is an expression then so is $(\neg\phi)$', where ϕ is not an expression itself but a variable ranging over expressions.

Definition 2.1.2 When we studying a language L, we distinguish between

(1) symbols of L, and

(2) symbols like ϕ above, that are not in L but are used to range over expressions of L.

The symbols in (2) are called metavariables. They will usually be Greek letters such as ϕ, ψ, χ.

Scanné avec CamScanner

Formulas of LP(σ)

We need to say which of the expressions of LP(σ) are 'grammatical sentences' of the language. These expressions are known as <u>formulas</u>. There is a short and sweet definition, as follows.

(2.5)
- $\bot$ is a formula of LP(σ), and so is every symbol in σ.
- If ϕ is a formula of LP(σ) the so is $(\neg\phi)$.
- If ϕ and ψ are formulas of LP(σ), then so are $(\phi\wedge\psi)$, $(\phi\vee\psi)$, $(\phi\to\psi)$ and $(\phi\leftrightarrow\psi)$.
- Nothing else is a formula of LP(σ).

The previous definition of 'formula' is not necessarily the most helpful. It presents the formulas as a set of strings and it hides te fact that formulas (like the sentences of any language) have a grammatical structure that underpins the uses we make of them. So in this and the next section we will follow an approach that has become standard in the study of natural languages, taking the grammatical structure as fundamental.

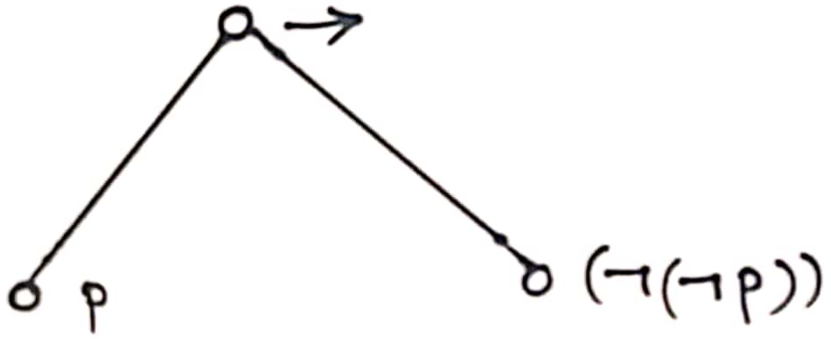
We begin with a particular formula of $LP(\sigma)$, where σ contains p :

(2.6) $$(p \to (\neg(\neg p))) .$$

This formula comes from Exp 1.6.1, by putting p in place of ϕ throughout.
It has the form $(\phi \to \psi)$ with p for ϕ and $(\neg(\neg p))$ for ψ.

We can write this analysis as a diagram:

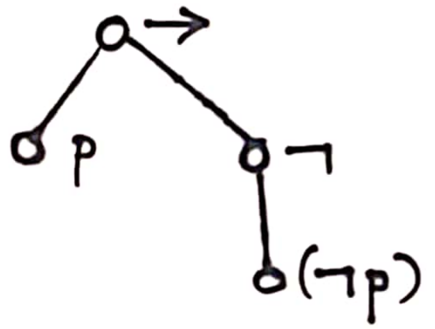
(2.7)



The formula on the left in (2.7) cannot be analysed further. But on the right, $(\neg(\neg p))$ has the form $(\neg\phi)$, where ϕ is $(\neg p)$.

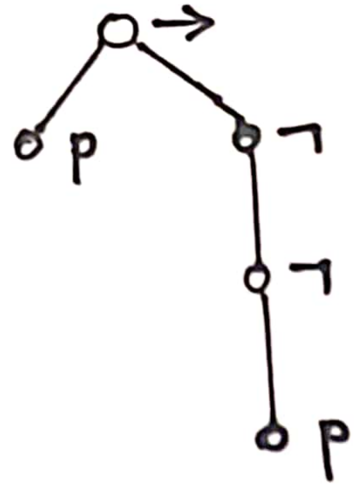
So we can extend the diagram downwards :

(2.8)



One more step analyses $(\neg p)$ as the result of negating p :

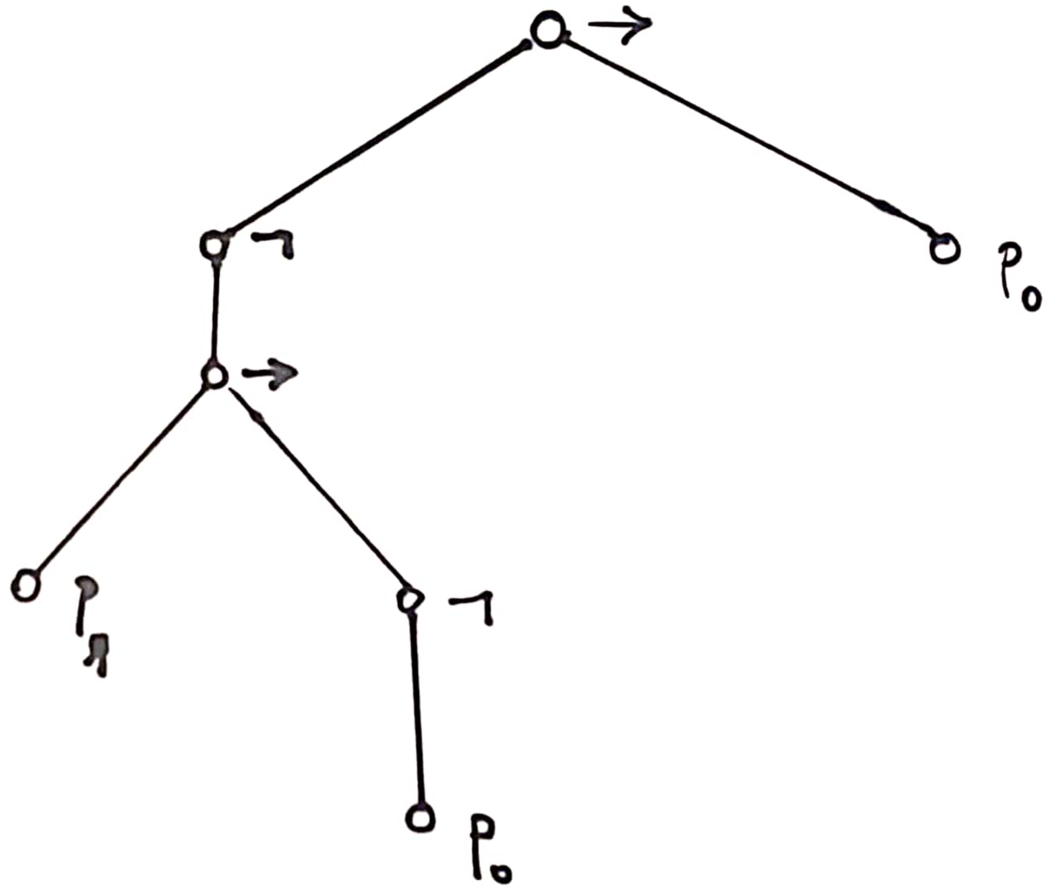
(2.9)



So we have a diagram (2.9) of small circles (called *nodes*) joined by lines, with labels on the right of the nodes. The labels on the two bottom nodes are one-symbol formulas. The other nodes carry the labels $\to$, $\neg$ and $\neg$.
The diagram with its labels analyses how formula (3.6) was put together. This kind of grammatical analysis is traditionally known as *parsing*.
The diagram branches downward, rather like a family tree, and so it will be called the *parsing tree* of the formula.

Given the tree diagram, we can reconstruct the formula that it came from. We illustrate this with a different tree that uses the signature $\{p_0, p_1\}$:

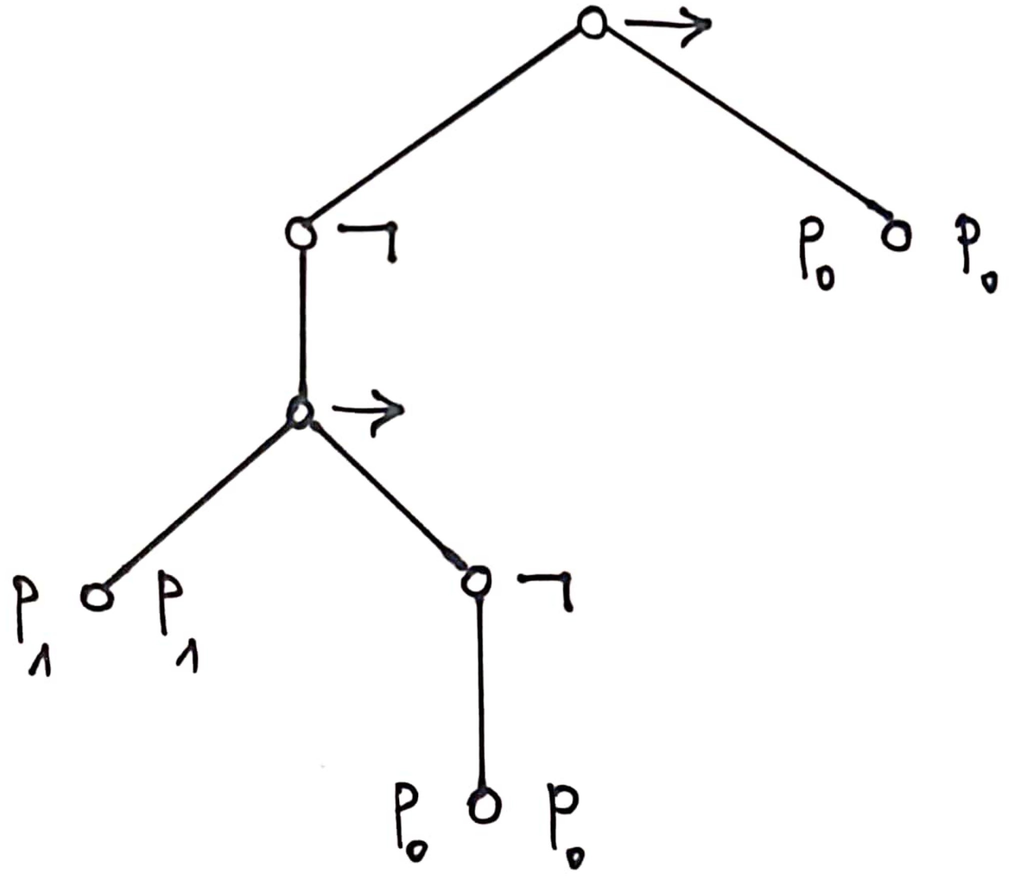
(2.10)



Scanné avec CamScanner

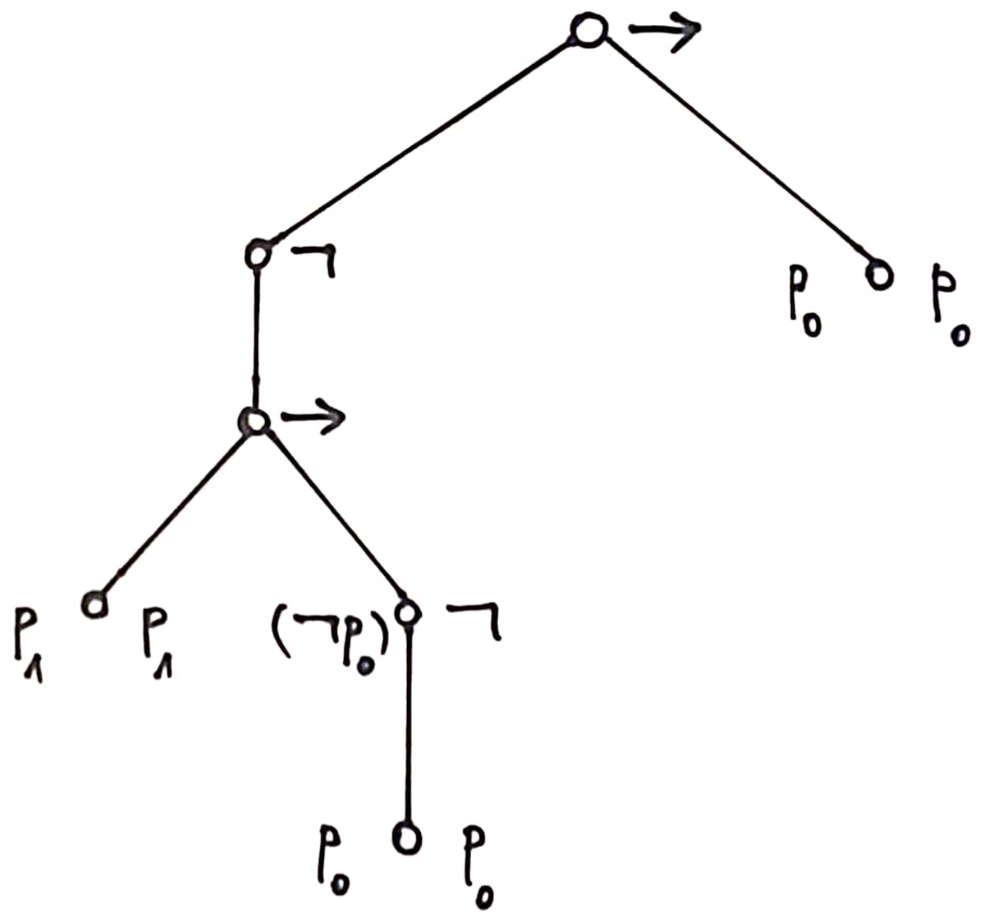
We reconstruct the formula by starting at the bottom tips of the tree and working upwards. As we go, we record our progress by writing labels on the <u>left</u> side of the tree nodes. the first step is to copy each propositional symbol on the left side of its node:

(2.11)



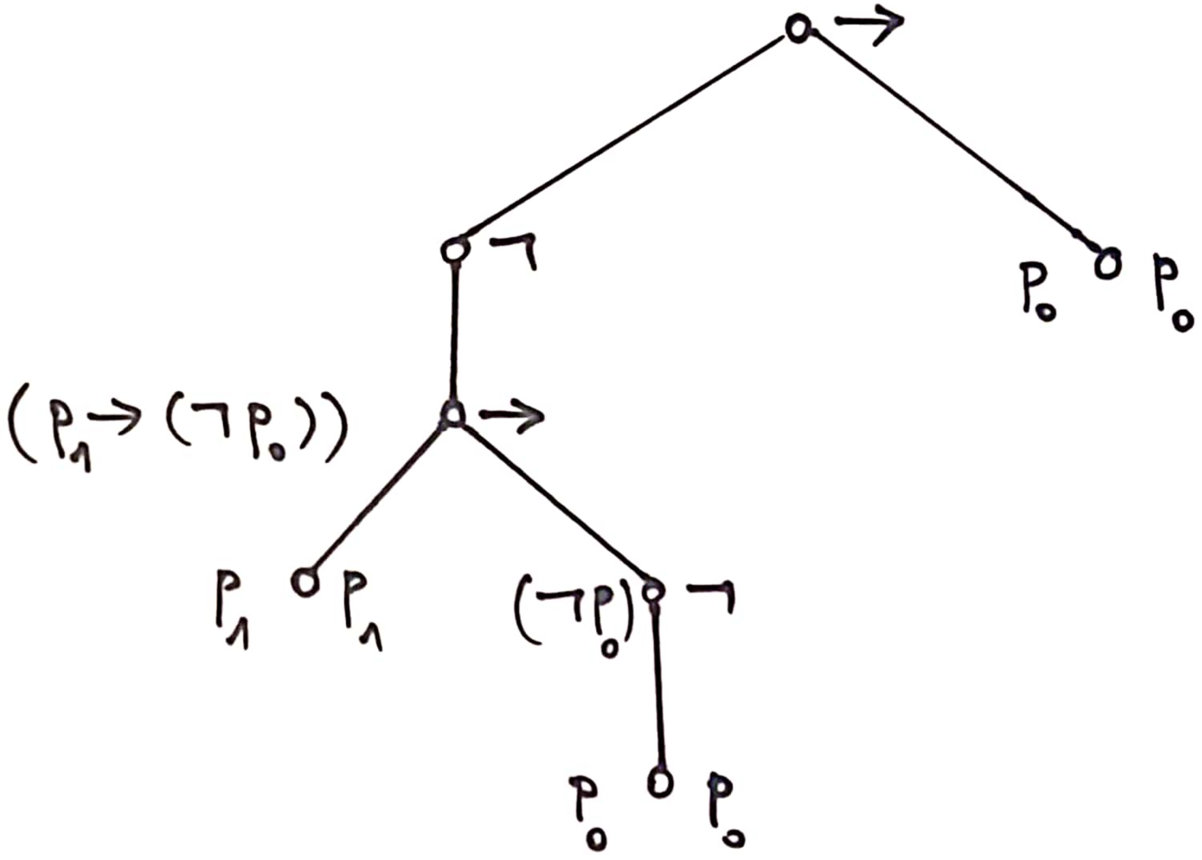
In the middle branch, the $\neg$ on the node just above p_0 shows that we form the formula $(\neg p_0)$. We write this formula on the left of the node:

(2.12)



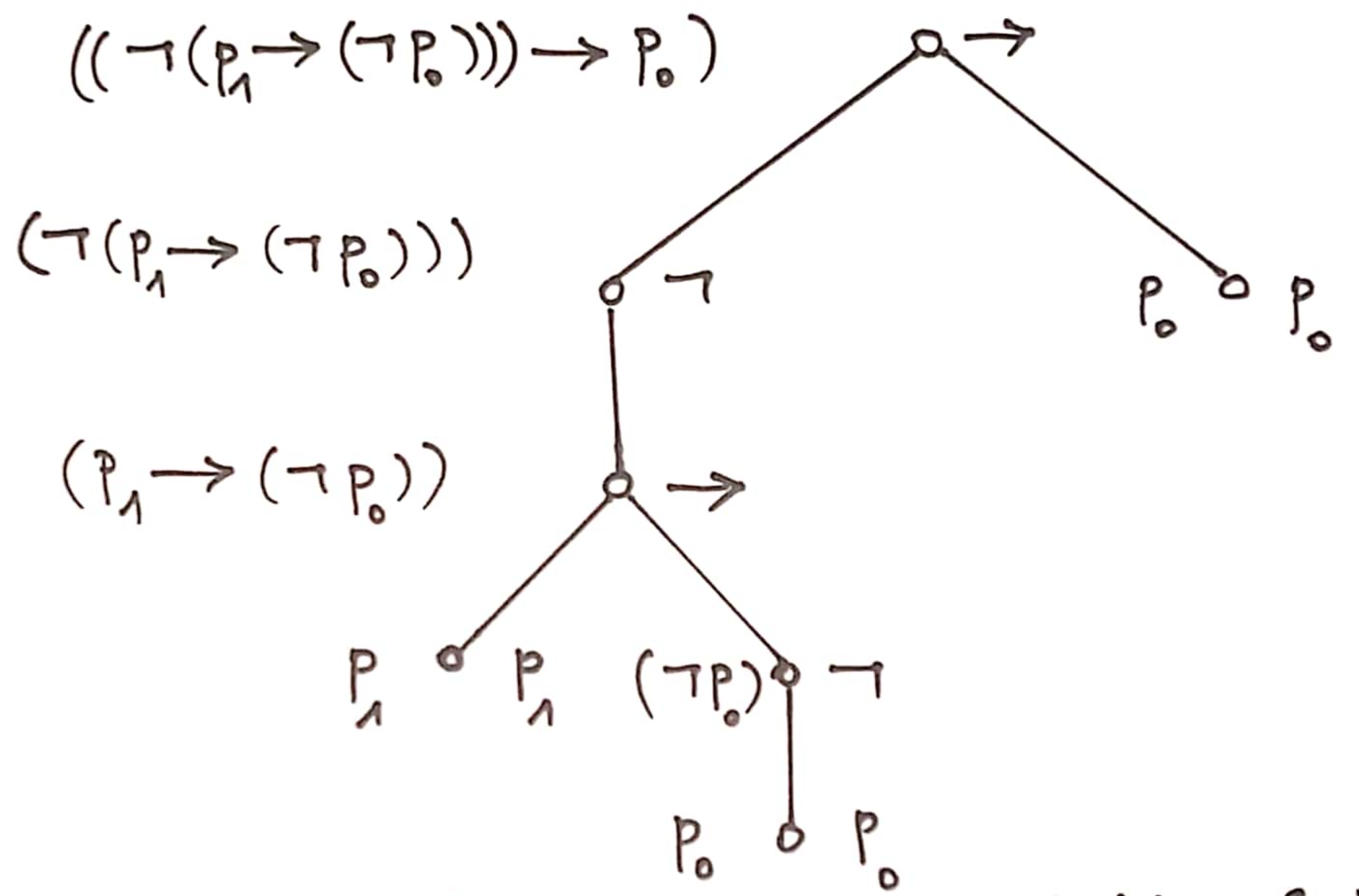
Now, the $\rightarrow$ on the left joins p_1 and $(\neg p_0)$ together to give $(p_1 \rightarrow (\neg p_0))$, and we attach this on the left of the node that carries the $\rightarrow$:

(2.13)



With two more moves to incorporate the remaining $\neg$ and $\rightarrow$, we finally reach the tree

(2.14)



$((\neg(p_1 \rightarrow (\neg p_0))) \rightarrow p_0)$
$\rightarrow$
$(\neg(p_1 \rightarrow (\neg p_0)))$
$\neg$
p_0
p_0
$(p_1 \rightarrow (\neg p_0))$
$\rightarrow$
p_1
p_1
$(\neg p_0)$
$\neg$
p_0
p_0

with $((\neg(p_1 \rightarrow (\neg p_0))) \rightarrow p_0)$ on the left of the top node. This is the formula constructed by the tree.

If we started with this formula and decomposed it to form a tree, the result would be (2.10). So the tree and the formula go hand in hand; we say the formula is *associated with* the tree, and the tree is the *parsing tree* of the formula. The labels on the left sides of nodes in (2.14) form the *syntax labelling* of the parsing tree; the next section will define this more rigorously.

In our example, we started at the bottom of the tree and worked upwards. At each node we knew how to write the label on its left, depending only on the symbol written on its right and the left-hand labels on the nodes (if any) one level down.

A set of instructions for writing a left labelling in this way is called a <u>compositional definition</u>. If δ is a compositional definition and we apply it to a parsing tree π, the thing we are trying to find is the left label on the top node; we call this label the <u>root label</u>, and we write it $\delta(\pi)$.

In the example above, the labels on the left nodes were expressions. But there are also compositional definitions that put other kinds of label on the left of nodes. For example, this is a compositional définition:

(2.15) Put 0 on the left of each node at the bottom of the tree. Next, if you have left labelled the nodes immediately below a given node ν, say with numbers $m_1, \dots, m_n$, then write

$$\max\{m_1, \dots, m_n\} + 1$$

on the left of ν.

This definition does not involve labels on the right at all. The label that it puts on the left of a node is called the <u>height of the node</u>; the root label is called the <u>height of the tree</u>.

Scanné avec CamScanner

Remark 2.1.3

Parsing trees were first designed by Frege in 1879 for his Begriffsschrift. Here is Frege's own version of (2.10) :

(2.16)