Logic Rondo

An Introduction to Logic for Humans and Computers

Henning Basold

Base revision 3f4832c, (HEAD -> main) from 2023-04-24

DRAFT
Do not distribute

© Copyright © 2020–2023 Henning Basold, under a Creative Commons Attribution-ShareAlike 4.0 International License: https://creativecommons.org/licenses/by-sa/4.0/.

Typeset with XeLaTeX

“Robot” on page 70 by William Hollowell licensed under Creative Commons BY, URL: https://thenounproject.com/term/r2d2/10452/
“obstacle” on page 70 by Annette Spithoven licensed under Creative Commons BY, URL: https://thenounproject.com/term/obstacle/211167
“Heart” on page 70 by Bohdan Burmich licensed under Creative Commons BY, URL: https://thenounproject.com/term/heart/396287

Contents

1. Introduction

2. Introduction to Propositional Logic

2.1. Motivation

Isaac: Ok, I see why you would want to study logic. But where do we start?

Clara: I have an idea. Let us go to the cradle of philosophy and mathematics in Europe.

Aristotle: Who are you, entering my house with that metal construction?

Isaac: I'm not a metal construction! I'm a living and breathing robot!

Aristotle: By Zeus! It can speak!

Isaac: More than that: without my Trans-O-Matic you would not be able to understand me or my friend.

Aristotle: A robot, mh? What is that supposed to mean?

Clara: Of course, you cannot know. You may think of an αὐτόματον (automaton) as Hephaestus created them, only that a robot can make decisions for itself and act like a human, all within limits certainly.

Aristotle: How fascinating! May I keep you here for studying my friend?

Isaac: Isaac is the name and I would prefer to be studied, if you don't mind. Rather, Clara and I are trying to understand the roots of my being together. We decided to start of quest with studying the logic underlying my artificial brain and my decisions. But we are not sure where to start. That's why we are here with you, one of the founders of formal logic.

Aristotle: I see, I see. Start where all logic starts: with simple deduction. Take, for example, the following deduction. "If it rains and Socrates has no umbrella, then he gets wet. Socrates has no umbrella and is not wet. Therefore, it does not rain."

Clara: Do you always have to use old male Philosophers as examples?

Aristotle: These are very illustrative examples, are they not?

Clara: Well... Never mind! The point is thus, that we infer knowledge from hypotheses and facts?

Aristotle: Yes, exactly. We shall therefore begin with the study of a simple logic that allows us to express the relation between propositions and infer knowledge from these relations. Let me introduce you to the syntax of a logic, as modern logic more than 2000 years from now will understand it.

This logic is called *propositional logic*, and comprises *propositional variables* and *logical connectives* that allow us to express relations between these variables as *formulas*. In section 2.2, we will see how these formulas are formed precisely and how we can use them to formalise hypotheses and facts.

For the time being, we will help Isaac to identify the relevant fragments of Aristotle's example. Let us **highlight** in the example the propositions that can be either true or false.

> "If **it rains** and **Socrates has no umbrella**, then **he gets wet**. **Socrates has no umbrella** and **he does not get wet**. Therefore, **it does not rain**."

If we write instead

- r for **it rains**,
- u for **Socrates has no umbrella**, and
- w for **he gets wet**,

then we can easily rewrite the first sentence to

> "If r and u, then w."

and can read it still in exactly the same way by replacing r, u and w by the corresponding phrases. We call r, u and w *propositional variables*, as they stand for propositions that can be either true or false. This is very important to keep in mind: we reason about such variables independently of their truth, and arguments need to be able to account for any possibility!

You will have noticed that the second sentence cannot directly written with the three propositional variables because it says "he does *not* get wet", while we only have the variable w that stands for "he gets wet". We will allow

ourselves to write "not" in front of a variable to negate what it says, that is, "not w" stands for "he does *not* get wet". In natural language, words have to appear in one or another order. This can create ambiguities, which is exactly what formal logic tries to prevent. With this in mind, we can write the second and third sentence as

> "u and not w. Therefore, not r."

What is left of the original sentence are only the words "if", "then", "and", "therefore" and "not". These are *logical connectives*, and we will introduce formal notations for them soon.

But let us appreciate for a moment that we have replaced in the original example certain phrases by variables and obtained an argument that relies only on the *structure* of the sentences, rather than their *meaning*. For instance, we could reinterpret the variables like so:

- r — **Isaac's battery is empty**,
- u — **no charger is in reach**, and
- w — **Isaac stops working**,

This gives us another argument that is clearly valid:

> "If **Isaac's battery is empty** and **no charger is in reach**, then **Isaac stops working**. **No charger is in reach** and **Isaac does not stop working**. Therefore, **Isaac's battery is not empty**."

2.2. Syntax of Propositional Logic

Syntax is the pillar of formal logic that allows us to express propositions *unambiguously*. As we saw earlier, it consists for propositional logic of two parts: propositional variables and logical connectives that can be put together into formulas. Propositional variables are syntactic entities that represent propositions, but have *no intrinsic meaning* and only serve as *placeholders*. We shall assume to be given a fixed set of such variables.

Assumption 2.1

Assume that PVar is a countable set of *propositional variables*. Elements of PVar are denoted by lower-case letters, possibly with index: $p, q, \dots, p_0, p_1, \dots$

As for the logic connectives, we will introduce symbols and formulas that allow us to unambiguously express propositions. For instance, we will write

$\wedge$ for "and" and $\neg$ for "not". With these notations and the variables u and w from above, the phrase

"Socrates has no umbrella and he does not get wet."

becomes

$$u \wedge \neg w\,.$$

Note that there are cases, where it is not clear how a sentence has to be read and we may have to use parentheses to disambiguate the reading. For example, we will write $\rightarrow$ for "if ...then ...". Then the sentence

"If it rains and Socrates has no umbrella, then he gets wet."

can be written as

$$(r \wedge u) \rightarrow w\,.$$

However, such parentheses can get in the way and we would like to have some reading conventions. We read, for example, the proposition $u \wedge \neg w$ intuitively already as $u \wedge (\neg w)$. Thus, part of the definition of formulas will also be a reading convention that allows us to unambiguously determine the structure of formulas.

Definition 2.2

The well-formed formulas (wff) φ of propositional logic are generated by the following context free grammar, in which p ranges over the propositional variables PVar.

$$\varphi ::= p \mid \bot \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid (\varphi)$$

Ambiguities in the grammar are resolved by the use of parentheses and the following reading conventions.

- $\wedge$ and $\vee$ have precedence over $\rightarrow$
- all connectives associate to the right

The set of all propositional wff is denoted by PForm. We denote elements of PForm by small Greek letters $\varphi, \psi, \gamma, \ldots$ possibly with a subscript index.

As reading and understanding formulas can be difficult, especially on first sight, let me provide some help. In table 2.1, all the connectives with their name, pronunciation and intuitive meaning are gathered. The name is how we will refer to a connective by itself, outside of formulas. To pronunciation

Connective	Name	Pronunciation	Intuitive Meaning
$\bot$	Absurdity	bottom	$\bot$ never holds
$\wedge$	Conjunction	φ and ψ	both hold
$\vee$	Disjunction	φ or ψ	φ or ψ or both hold
$\to$	Implication	φ implies ψ	if φ holds, then ψ holds

Table 2.1.: Logical connectives of propositional logic

we will use, of course, the pronunciation column. If you have trouble with Greek letters, then have a look at appendix A. Finally, the last column indicates how the connectives can be understood intuitively. Keep in mind though that this is only an intuition and the interpretation can vary radically for different applications and semantics, one of which we will discuss in chapter 3.

"But wait", Clara intervenes, "we are missing the negation, aren't we?" Yes indeed, we are. However, our intuition would dictate that the formula $\neg\varphi$ should hold only if φ does not hold. Or, in other words, whenever φ holds, something went wrong and we discovered an absurd situation. We can express this by the formula $\varphi \to \bot$, saying that φ implies absurdity. Similarly, we can also express other common logical connectives in terms of the basic connectives.

Definition 2.3: Derived connectives

We define three derived connectives as short-hand notation for the formula in second column of the following table.

Connective	Definition	Name
$\neg\varphi$	$\varphi \to \bot$	Negation
$\top$	$\neg\bot$	Truth or Top
$\varphi \leftrightarrow \psi$	$(\varphi \to \psi) \wedge (\psi \to \varphi)$	Bi-implication

We also adopt the following reading conventions: negation has precedence over $\vee$ and $\wedge$, thus also over $\to$; and bi-implication has the same precedence as $\to$.

Let us now come back to the original example.

Formula	With parentheses
$p \to q \to r$	$p \to (q \to r)$
$p \wedge q \wedge r$	$p \wedge (q \wedge r)$
$p \vee q \vee r$	$p \vee (q \vee r)$
$p \wedge q \to r$	$(p \wedge q) \to r$
$p \vee q \to r$	$(p \vee q) \to r$
$p \to q \wedge r$	$p \to (q \wedge r)$

Table 2.2.: Leaving out parentheses by using the precedences of connectives

Example 2.4

The deduction that it did rain by experimenting with Socrates' misery consists of the following three formulas.

$$r \wedge u \to w \quad \text{and} \quad u \wedge \neg w \quad \text{and} \quad \neg r$$

Putting these all together as one deduction, we obtain:

$$(r \wedge u \to w) \wedge (u \wedge \neg w) \to \neg r$$

Note that the first two formulas are put together with a conjunction, while the last comes after an implication. This way, we have resolved all the ambiguity in the original deduction. The sentence starting with "Therefore" signifies the right-hand side, the *conclusion* of an implication and everything before are the *assumptions* that are made in the deduction.

You should also appreciate that we can leave out parentheses by using our reading conventions. Without them, the formula in example 2.4 would look like this:

$$\Big(\big((r \wedge u) \to w\big) \wedge \big(u \wedge (\neg w)\big)\Big) \to (\neg r)$$

What an abomination! I wish, I had these tools in the debates with my contemporary philosophers in the ancient Greek times.

Table 2.2 shows some more example, in which the reading conventions allow us to leave out parentheses Note that there is no convention about mixing $\wedge$ and $\vee$, as this would cause more confusion than it helps. For example, the formula $p \wedge q \vee r$ is considered to be ambiguous and should be written either as $(p \wedge q) \vee r$ or $p \wedge (q \vee r)$.

2.3. Parse Trees

Isaac: I have the feeling that there may still be some ambiguity. How can I know in which order I have to process a formula?

Aristotle: Feelings? How ...?

Isaac: Hey, no need to insult me!

Aristotle: My apologies! But you are a curious thing and I would like to ask you so many questions. In any case, there is a way to make everything absolutely unambiguous by two-dimensional trees, the kind you have seen as data structures.

Formula	Top-Level Connective	Direct Subformulas
p	p	–
$\bot$	$\bot$	–
$\varphi \wedge \psi$	$\wedge$	φ, ψ
$\varphi \vee \psi$	$\vee$	φ, ψ
$\varphi \to \psi$	$\to$	φ, ψ

Table 2.3.: Top-Level Connectives and Direct Subformulas

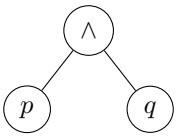
Definition 2.5

The *top-level connective* and *direct subformulas* of formulas are given as in table 2.3. A formula is called *atomic* if it has no direct subformulas, that is, if it is of the shape $\bot$ or p for $p \in \mathrm{PVar}$.
Given a formula φ, the *parse tree* of φ is a tree, in which

i) the root is labelled by the top-level connective of φ, and

ii) the children of the root are parse trees of the direct subformulas of φ.

When we picture such trees, we typically draw a circle for every node and write the label inside this node. This allows to picture, for example, the formula $p \wedge q$ as



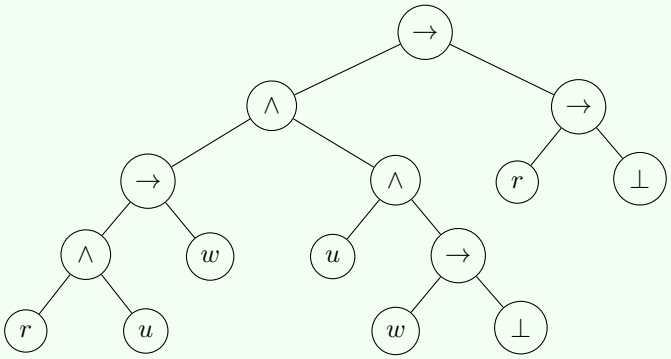
Our formula from earlier can serve as a more elaborate example.

Example 2.6

Recall the formula

$$(r \wedge u \to w) \wedge (u \wedge \neg w) \to \neg r$$

from example 2.4. The parse tree of this formula is given as follows.



Note that the parse tree does not contain negations because this is a derived connective. Instead, it is represented by the defining formula. For instance, $\neg r$ becomes $r \to \bot$ in the parse tree.

With parse trees, we can clearly circumvent any ambiguities.

2.4. Formula Iteration and Induction

Isaac: What a great contribution of computer science to the world! But what If I would like to make any formal statements or definitions for formulas? Are parse trees then not a bit too informal?

Aristotle: In fact, they are not. As parse trees give a unique representation for formulas, we can derive a proof principle that is familiar from the natural numbers.

Clara: Are you speaking about induction?

Aristotle: Indeed, I am. However, let me state first the iteration principle for formulas, as this allows us to define maps on formulas.

Theorem 2.7: Principle of Formula Iteration

Let A be a set together with $f_\bot \in A$ and four maps

$$f_P \colon \mathrm{PVar} \to A \qquad f_\wedge \colon A \times A \to A$$
$$f_\vee \colon A \times A \to A \qquad f_\to \colon A \times A \to A\,,$$

where $A \times A$ is the set-theoretic product (appendix B.1). Then there is a unique map $f \colon \mathrm{PForm} \to A$, such that the following equations hold.

$$f(\bot) = f_\bot \qquad f(\varphi \wedge \psi) = f_\wedge(f(\varphi), f(\psi))$$
$$f(p) = f_P(p) \qquad f(\varphi \vee \psi) = f_\vee(f(\varphi), f(\psi))$$
$$f(\varphi \to \psi) = f_\to(f(\varphi), f(\psi))$$

Proof. The idea of the proof is simple. Given the element $f_\bot$ and the four maps, we define $f(\varphi)$ by traversing the parse tree T depth-first from left to right. This, in turn, is done by iteration on the height of the tree T.

For atomic formulas, T has height 0 and we can directly define $f(\varphi) = f_\bot$ ($\varphi = \bot$) or $f(\varphi) = f_P(\varphi)$ ($\varphi \in \mathrm{PVar}$). If φ is not atomic, for example, if $\varphi = \varphi_1 \wedge \varphi_2$, then T is labelled at the root with $\wedge$ and the root has trees T_1 and T_2 as children. These children are themselves parse trees of smaller height and we can assume $f(\varphi_k)$ to be given. Thus, we can define $f(\varphi) = f_\wedge(f(\varphi_1), f(\varphi_2))$. Clearly, f fulfils the required equations.

If we are given a map $g \colon \mathrm{PForm} \to A$ that also fulfils the equations, then we can prove that $f = g$ also by induction on the height of parse trees. For atomic formulas φ, we clearly have $g(\varphi) = f(\varphi)$. If φ has a parse tree of height $n+1$, say, $\varphi = \varphi_1 \wedge \varphi_2$, then we can assume as induction hypothesis that $g(\varphi_k) = f(\varphi_k)$ for $k = 1, 2$. But then

$$g(\varphi) = f_\wedge(g(\varphi_1), g(\varphi_2) = f_\wedge(f(\varphi_1), f(\varphi_2) = f(\varphi).$$

Thus, by induction on the tree height of φ, we have that $g(\varphi) = f(\varphi)$ for all formulas φ and thereby that f is unique. □

Just like for natural numbers (theorem B.2), the usual induction principle can be obtained as a special case of the iteration principle.

Corollary 2.8: Formula Induction

Let P be a property of formulas, that is, $P \subseteq \text{PForm}$. If

i) P contains all atomic formulas ($\text{PVar} \subseteq P$ and $\bot \in P$), and

ii) for all formulas φ: if P contains all direct subformulas of φ, then $\varphi \in P$,

then P contains all formulas ($\text{PForm} \subseteq P$). We refer to item i) as the **base case** and to item ii) as the **induction step**.

As you can see, formula iteration and induction follow directly from the fact that formulas can be represented as parse trees with finite height. However, formula iteration and induction are much easier to use than induction on the height of trees because we can directly refer to the structure of the tree. This is why induction principles like that in corollary 2.8 are also referred to as *structural induction*.

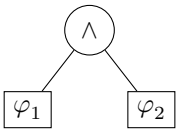
What can we use these principles for? For example, we can use it to find *all* subformulas, and not just the direct ones, of a given formula.

Definition 2.9

Let φ be a formula. The set of **subformulas** of φ is given by the following iterative definition.

$$\begin{aligned}
\text{Sub}(p) &= \{p\} \\
\text{Sub}(\bot) &= \{\bot\} \\
\text{Sub}(\varphi_1 \wedge \varphi_2) &= \{\varphi_1 \wedge \varphi_2\} \cup \text{Sub}(\varphi_1) \cup \text{Sub}(\varphi_2) \\
\text{Sub}(\varphi_1 \vee \varphi_2) &= \{\varphi_1 \vee \varphi_2\} \cup \text{Sub}(\varphi_1) \cup \text{Sub}(\varphi_2) \\
\text{Sub}(\varphi_1 \to \varphi_2) &= \{\varphi_1 \to \varphi_2\} \cup \text{Sub}(\varphi_1) \cup \text{Sub}(\varphi_2)
\end{aligned}$$

By "iterative" in definition 2.9 we mean that formula iteration can in principle be used to define Sub. This is not difficult but rather tedious, just in the same way it is tedious to define the factorial function on natural numbers by iteration (see appendix B.2). But we can very easily understand how Sub operates in terms of parse trees. Suppose φ is given by the following parse tree, in which rectangles represent again parse trees.



Then the subformulas of φ are the whole formula φ itself and the subformulas of the formulas that are represented by the smaller parse trees. In other words, we can list all subformulas by recursively walking through the parse tree.

Example 2.10

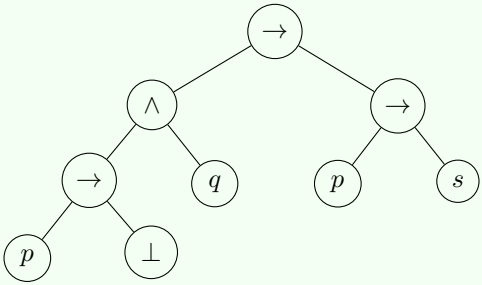
Let $\varphi = \neg p \wedge q \to p \to s$. Then we formally obtain the subformulas of φ by

$$\begin{aligned}\mathrm{Sub}(\varphi) &= \{\varphi\} \cup \mathrm{Sub}(\neg p \wedge q) \cup \mathrm{Sub}(p \to s) \\ &= \{\varphi\} \cup \{\neg p \wedge q, \neg p, p, \bot, q\} \cup \{p \to s, p, s\} \\ &= \{\varphi, \neg p \wedge q, \neg p, p, \bot, q, p \to s, s\}\,,\end{aligned}$$

where we use that

$$\begin{aligned}\mathrm{Sub}(\neg p \wedge q) &= \{\neg p \wedge q\} \cup \mathrm{Sub}(\neg p) \cup \mathrm{Sub}(q) \\ &= \{\neg p \wedge q\} \cup \{\neg p\} \cup \mathrm{Sub}(p) \cup \mathrm{Sub}(\bot) \cup \mathrm{Sub}(q) \\ &= \{\neg p \wedge q\} \cup \{\neg p\} \cup \{p\} \cup \{\bot\} \cup \{q\} \\ &= \{\neg p \wedge q, \neg p, p, \bot, q\}\,.\end{aligned}$$

If we look at the parse tree of φ, then the subformulas are evident because every node contributes as subformula the formula that is represents.



This gives us the following list of distinct subformulas by traversing

the tree depth-first from left to right.

$\neg p \wedge q \to p \to s$	$p \to s$	s
$\neg p \wedge q$	q	
$\neg p$	$\bot$	
p		

Clara: These are a lot of formalities for expressing very simple things! Is all of this really necessary?

Aristotle: If you know how computers work, then you should be able to appreciate the clarity of representing formulas as parse trees and formula induction as recursive visits of trees already. A more elaborate answer would be that we have seen throughout history everything from misunderstandings to lies because assertions have been misunderstood, wrong or implicit assumptions have been made etc. The formal language of propositional logic does away with all of this, as there is no ambiguity in assertions being made. If someone makes a mistake, then you can find that by inspecting the claimed formula.

Isaac: Just by looking at the formula? What if I don't know what the propositional variables should mean or how one part of a formula relates to another?

Aristotle: Very well observed, my little brass friend!

Isaac: Brass friend???

Aristotle: To get a proper answer to your question, I will have to refer you to another logician, one who studied the meaning of formulas rigorously.

3. Semantics of Propositional Logic

Peirce: Who on Earth are you two fellows bursting into my house and disturbing my studies! Is this tin can another concoction by Newcomb or the Southerners to disturb me?

Clara: Our apologies for coming in uninvited, Professor Pierce. My friend Isaac and I set out to learn logic, and we were told that you may help us with that.

Peirce: That is certainly possible. How may I help you?

Clara: My friend Isaac here is a robot and his functioning is founded in pure logic, but we don't understand it.

Peirce: Are you saying that this creation adheres to reason?

Isaac: Even better, all my decisions are based on what I can deduce from the facts that I know.

Peirce: ..., which makes you inherently limited.

Isaac: Does it? I would like to understand that better.

Peirce: Good, you are following the first rule of logic: the desire to learn.[1] Let us see, where do we start?

Clara: We paid a visit to Aristotle and learned about the syntax of propositional logic.

[1] "Upon this first, and in one sense this sole, rule of reason, that in order to learn you must desire to learn, and in so desiring not be satisfied with what you already incline to think, there follows one corollary which itself deserves to be inscribed upon every wall of the city of philosophy: Do not block the way of inquiry.", from C.S. Peirce, *F.R.L. First Rule of Logic*, 1899.

Peirce: How did you ...? I guess this whole encounter is illogical in any case. Well, if you know about syntax, then you know already how to avoid the fallacies of natural language. But do you know what formulas mean and how they relate?

Clara: Kind of, I think we know intuitively what they mean. But how they relate is unclear, at least to me.

Peirce: In that case, let us talk about semantics and understand the signs of logical formulas. Once we have that, we can talk about the relation of formulas and how Isaac may attempt logical deduction.

3.1. Truth Values

For every propositional formula, we can try to understand under what conditions it represents a correct or incorrect, a *true* or *false*, proposition. This is simplest way of assigning *truth values* to formulas.

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

Table 3.1.: Truth table of conjunction

Example 3.1

Let φ be the formula $p \wedge q$ for some propositional variables p and q. What would the truth value of φ? Is φ true or false? That depends entirely on the truth values of p and q, since propositional variables have no intrinsic meaning. Let us, for simplicity, write 0 instead of false and 1 instead of true. We could choose another notation, but this particular notation is short and will turn out to be beneficial, not the least because it corresponds to bits and voltage levels in your circuits, Isaac!
Now think about the case that p and q are both true, thus we assume they have both 1 as truth value. What would be the truth value of the conjunction $p \wedge q$? Clearly, as p *and* q are true, so should be $p \wedge q$ (read $\wedge$ out as "and"). Thus, if p and q have the truth value 1, the φ should

have as well the truth value 1.
How about that case that one of them, say q, is false and has the truth value 0? Then we end up with the question whether "true and false" should be true or false. As it is not possible that something is true and false at the same time, we deduce that $p \wedge q$ is false in this case and the formula φ has the truth value 0.
We can continue like this and prepare the small table 3.1. This table lists all the possible values that the variables p and q can take, together with the truth value of $p \wedge q$ that results from these values. Thus, the first two columns list all possibilities, while the last column is *computed* [Ane12] or *deduced* from the first column.

?

Suppose we prepare a truth table for the formula $p \wedge q \wedge r$ with three different propositional variables. How many rows would the table have?

Surely, we could devise truth tables for all kinds of formulas by hand, but what are truth tables in general and how do they relate to the meaning of formulas? To understand this, let us consider each row of the truth table in table 3.1 separately. Every row assigns truth values to the variables that appear in the formula and then determines the truth values of the overall formula. Note that it does not matter what other variables, which do not appear in the formula, have as truth value. The following definition can be thought of formalising the values of variables in one row of a truth table.

Definition 3.2

We define the set of *truth values* $\mathbb{B}$ by $\mathbb{B} = \{0, 1\}$. A *valuation* is a map $v \colon \mathrm{PVar} \to \mathbb{B}$, that is, an assignment of unique truth values to all propositional variables.

You will have noticed that definition 3.2 leads to infinite truth tables. This is of course rather inconvenient on paper and we will rectify this later. However, in theoretical investigations this definition is easy to work with.

3.2. Boolean Semantics of Propositional Logic

"Was the reason to introduce truth tables not to determine the truth value of formulas?", Isaac wonders. "But definition 3.2 only speaks about the values assigned to variables, doesn't it?". Absolutely, and the next step is to calculate from a valuation the truth value of a formula. For this, we will use that 0 and 1 are ordered as numbers, that is, we have $0 \leq 1$, $0 \leq 0$ and $1 \leq 1$. Using this information, we can reduce the calculation of truth values to familiar operations on numbers.

Definition 3.3

We define the **semantic implication** as binary operation $\Longrightarrow\colon \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ on truth values $x, y \in \mathbb{B}$ by the following case distinction.

$$x \Longrightarrow y = \begin{cases} 1, & x \leq y \\ 0, & \text{otherwise} \end{cases}$$

Given a valuation v, we define the **Boolean propositional semantics** of formulas iteratively as follows.

$$\begin{aligned} [\![-]\!]_v &\colon \mathrm{PForm} \to \mathbb{B} \\ [\![p]\!]_v &= v(p) \\ [\![\bot]\!]_v &= 0 \\ [\![\varphi \wedge \psi]\!]_v &= \min\{[\![\varphi]\!]_v, [\![\psi]\!]_v\} \\ [\![\varphi \vee \psi]\!]_v &= \max\{[\![\varphi]\!]_v, [\![\psi]\!]_v\} \\ [\![\varphi \to \psi]\!]_v &= [\![\varphi]\!]_v \Longrightarrow [\![\psi]\!]_v \end{aligned}$$

x	y	$x \Longrightarrow y$
0	0	1
0	1	1
1	0	0
1	1	1

Table 3.2.: Values of Semantic Implication

Clara: Wait, wait, Professor ... Why do we need the semantic implication and what are these strange double brackets?

Peirce: The semantic implication is not strictly necessary, but it clarifies the intention in the semantics of the *syntactic* implication. Also, it makes

it easier to understand how implication works, as we can make a table with all possible arguments of the operation and the results, see table 3.2.

Clara: Is this not a truth table?

Peirce: It indeed looks suspiciously like one and the truth table for implication would have exactly the same entries.

Clara: So why do we need the semantic implication then?

Peirce: Look at the definition, it tells you that the semantic implication is only true if the *antecedent*, the first argument x, has a truth value that is below that of the *consequent* y. In particular, if x is true, y must be true. By just looking at the truth table, you must be particularly confused by the first row, but our definition says that implication is just the same as the order of numbers!

Clara: Ok, but what about these funny brackets?

Peirce: These double brackets, called Scott brackets in honour of Dana Scott, are commonly used in denotational semantics. They take in a syntactic object, here formulas, and map that to an element of the semantic domain, here the truth values in $\mathbb{B}$. These brackets have two important features: First, they define a map, which means that *every* formula gets assigned a *unique* truth value. Second, they are defined by iteration on formulas. This means that the truth value of a formula is determined by the truth of its direct subformulas.

Isaac: That reminds me of the principle of iteration that Aristotle explained to us.

Peirce: Very well observed! In fact, the definition of subformulas has proceeded in exactly the same way as the definition of the semantics.

But enough chatter! Let us discuss us some examples.

Example 3.4

Let $p, q \in \mathrm{PVar}$ and let $v\colon \mathrm{PVar} \to \mathbb{B}$ be the valuation defined by $v(p) = v(q) = 1$ and $v(r) = 0$ for all other $r \in \mathrm{PVar}$. Then we have

$$[\![p \wedge q]\!]_v = \min\{[\![p]\!]_v, [\![q]\!]_v\} = \min\{v(p), v(q)\} = \min\{1, 1\} = 1\,.$$

You could have read this, of course, already off table 3.1. So let us try something more complex.

Example 3.5

Suppose p and q are distinct variables and we define a valuation v by $v(p) = 0$, $v(q) = 1$ and $v(r) = 0$ for all other $r \in \mathrm{PVar}$. Then the semantics of the formula $p \vee (q \to p) \to q \to p$ with respect to v are given as follows.

$$\begin{aligned}
&[\![p \vee (q \to p) \to q \to p]\!]_v \\
&= [\![p \vee (q \to p)]\!]_v \Longrightarrow [\![q \to p]\!]_v \\
&= [\![p \vee (q \to p)]\!]_v \Longrightarrow ([\![q]\!]_v \Longrightarrow [\![p]\!]_v) \\
&= [\![p \vee (q \to p)]\!]_v \Longrightarrow (v(q) \Longrightarrow v(p)) \\
&= \max\{v(p), v(q) \Longrightarrow v(p)\} \Longrightarrow (v(q) \Longrightarrow v(p)) \\
&= \max\{0, 1 \Longrightarrow 0\} \Longrightarrow (1 \Longrightarrow 0) \\
&= \max\{0, 0\} \Longrightarrow 0 \\
&= 0 \Longrightarrow 0 \\
&= 1
\end{aligned}$$

3.3. Back to Truth Tables

Isaac insists: "Ok, we can calculate now the semantics of formulas for some valuations. But what about truth tables? These seem to be rather convenient." They are indeed! The following theorem shows that we can reduce the calculation of the semantics to truth tables. Have you noticed in the examples 3.4 and 3.5 that the valuations were defined to be 0 on the variables that did not occur in the formulas? In fact, we could have given them any value, as they did not appear in the formulas. We also say that the semantics of formulas is locally determined.

Theorem 3.6: Local Determination

Let φ be a formula. If v_1 and v_2 are valuations, such that $v_1(p) = v_2(p)$ for all variables $p \in \mathrm{var}(\varphi)$ that appear in φ, then $[\![\varphi]\!]_{v_1} = [\![\varphi]\!]_{v_2}$.

Proof. We proceed by induction on formulas. In the base cases p and $\bot$, we have

$$[\![p]\!]_{v_1} = v_1(p) = v_2(p) = [\![p]\!]_{v_2}$$

and

$$[\![\bot]\!]_{v_1} = 0 = [\![\bot]\!]_{v_2}\,.$$

For the induction step, we assume that the induction hypothesis (IH) holds for φ_1 and φ_2, that is, $[\![\varphi_k]\!]_{v_1} = [\![\varphi_k]\!]_{v_2}$ for $k = 1, 2$. We then have

i) for the conjunction that

$$\begin{aligned}
[\![\varphi_1 \wedge \varphi_2]\!]_{v_1} &= \min\{[\![\varphi_1]\!]_{v_1}, [\![\varphi_2]\!]_{v_1}\} && \text{by definition}\\
&= \min\{[\![\varphi_1]\!]_{v_2}, [\![\varphi_2]\!]_{v_2}\} && \text{by IH}\\
&= [\![\varphi_1 \wedge \varphi_2]\!]_{v_2} && \text{by definition}
\end{aligned}$$

ii) the analogous argument for disjunction, and

iii) for implication that

$$\begin{aligned}
[\![\varphi_1 \to \varphi_2]\!]_{v_1} &= [\![\varphi_1]\!]_{v_1} \Longrightarrow [\![\varphi_2]\!]_{v_1} && \text{by definition}\\
&= [\![\varphi_1]\!]_{v_2} \Longrightarrow [\![\varphi_2]\!]_{v_2} && \text{by IH and because } \Longrightarrow \text{ is a map}\\
&= [\![\varphi_1 \to \varphi_2]\!]_{v_2} && \text{by definition}
\end{aligned}$$

Thus, for all formulas φ, the semantics $[\![\varphi]\!]_{v_1}$ and $[\![\varphi]\!]_{v_2}$ agree, meaning that they are determined only by the variables that appear in φ. □

The result of theorem 3.6 allows us to provide the semantics of any formula φ in terms of truth tables. To this end, we make a table that has one column for every variable that appears in φ and one column for the semantics of φ. The rows of the table will be determined by all the possible values that the variables can attain and the evaluation of the semantics of φ under a valuation that is compatible with values of the variables in a row. For instance, suppose that only the variables p and q appear in φ. The truth table would then start like this, where v is any valuation with $v(p) = v(q) = 0$:

p	q	φ
0	0	$[\![\varphi]\!]_v$
⋮	⋮	⋮

We are now able to construct a truth table for some more complex example.

Example 3.7

Recall that we calculated the semantics of $p \vee (q \to p) \to q \to p$ in example 3.5 for a specific valuation. Using the above recipe, we can now construct the truth table for this formula. It can be helpful for complex formula like this one to also add columns for subformulas and determine their semantics first. Like this, the overall calculation becomes apparent from the table. We indicate this by drawing a single vertical line between the columns of the variables and the subformulas, which in turn are separated by a double vertical line from the formula. For the above formula, we then obtain the following truth table.

p	q	$q \to p$	$p \vee (q \to p)$	$p \vee (q \to p) \to q \to p$
0	0	1	1	1
0	1	0	0	1
1	0	1	1	1
1	1	1	1	1

3.4. Entailment, Satisfiability, Tautologies

"Great, now we understand what formulas *mean*! But hold on, there seems to be something funny going on in example 3.7: all the rows in the table have the same truth value for the formula", Isaac inquires. "Is this correct?" It is! Such formulas have a special status and we call them *tautologies*. Before we go there, let me introduce you to another formulation of the semantics for formulas that is convenient whenever we want to state that a formula is true.

Definition 3.8

For Γ a set of formulas, that is $\Gamma \subseteq \mathrm{PForm}$, and a valuation v we define the semantics of Γ by

$$[\![\Gamma]\!]_v = \min\{[\![\psi]\!]_v \mid \psi \in \Gamma\}$$

with $[\![\Gamma]\!]_v = 1$ if $\Gamma = \emptyset$. For a formula φ, we say that Γ **entails** φ, written as $\Gamma \vDash \varphi$, if we have for all valuations v that $[\![\Gamma]\!]_v \leq [\![\varphi]\!]_v$. The set Γ contains the *premises* of the entailment. In the case that Γ is empty, we write $\vDash \varphi$ instead of $\emptyset \vDash \varphi$.

Compare this to the definition of the semantic implication, which holds if and only if the antecedent is below the consequent. You will see that, intuitively, $\Gamma \vDash \varphi$ holds if the conjunctions of all formulas in Γ implies φ. It might be the case that Γ is not a finite set, in which case we cannot form such a conjunction, but the entailment works even in this case. Even though an infinite set Γ entails a formula φ, only a finite amount of formulas contributes to the entailment. Proving this needs, however, tools that are not yet available to us.

What we can do though is to use definition 3.8 to deduce formulas from given premises. The following theorem 3.9 provides some rules that simplify the deduction process.

Theorem 3.9

For all formulas φ and ψ and $\Gamma \subseteq \mathrm{PForm}$ the following holds.

- if $\varphi \in \Gamma$, then $\Gamma \vDash \varphi$
- if $\Gamma \vDash \bot$, then $\Gamma \vDash \varphi$
- $\Gamma \vDash \varphi \wedge \psi$ iff $\Gamma \vDash \varphi$ and $\Gamma \vDash \psi$
- $\Gamma \vDash \varphi \vee \psi$ iff $\Gamma \vDash \varphi$ or $\Gamma \vDash \psi$ (or both)
- $\Gamma \vDash \varphi \to \psi$ iff, whenever $\Gamma \vDash \varphi$ holds, then $\Gamma \vDash \psi$
- $\Gamma \vDash \varphi \to \psi$ iff $\Gamma \nvDash \varphi$ or $\Gamma \vDash \psi$ (or both)
- $\Gamma \vDash \neg\varphi$ iff $\Gamma \nvDash \varphi$

Proof. We can analyse all the cases separately. For instance, suppose $\varphi \in \Gamma$. By definition 3.8, we have for any valuations v that

$$[\![\Gamma]\!]_v = \min\{[\![\psi]\!]_v \mid \psi \in \Gamma\} \leq [\![\varphi]\!]_v \,,$$

which proves our claim.

We shall also prove the case of conjunction. Let v be a valuation. Then

$$\begin{aligned} [\![\Gamma]\!]_v \leq [\![\varphi \wedge \psi]\!]_v &= \min\{[\![\varphi]\!]_v, [\![\psi]\!]_v\} \\ \text{iff } [\![\Gamma]\!]_v \leq [\![\varphi]\!]_v &\text{ and } [\![\Gamma]\!]_v \leq [\![\psi]\!] \,. \end{aligned}$$

As this holds for any valuation $\Gamma \vDash \varphi \wedge \psi$ iff $\Gamma \vDash \varphi$ $\Gamma \vDash \psi$.

All the other cases are proven analogously. □

"Wait, there are two items for the implication?", Clara objects. Yes, this is quite special about the Boolean semantics. The implication can be expressed in this semantics by using negation and disjunction. But don't be misled, there are also other possible semantics of propositional logic, like Kripke semantics, that we will unfortunately not touch upon here. In these semantics, implication cannot be expressed with negation and disjunction.

But we are digressing. Let us come back to the formula from example 3.7 that was always true. There are a few classifications of formulas and relations between formulas that allow us to say how to prove propositions, which means to establish that such propositions are true. As you may imagine, this is not always possible.

Definition 3.10

Let $\varphi, \psi \in \mathrm{PForm}$. We say that

- φ is *satisfiable* if there is a valuation v with $[\![\varphi]\!]_v = 1$. Otherwise, we say that φ is *unsatisfiable*;
- φ is a *tautology* if $\vDash \varphi$, that is, $\emptyset \vDash \varphi$; and
- φ and ψ are *semantically equivalent*, written $\varphi \equiv \psi$, if for all valuations v, $[\![\varphi]\!]_v = [\![\psi]\!]_v$.

Let me give you some examples to bring this list of terminology to life.

Example 3.11

1. Let $v(p) = v(q) = 1$ and $v(r) = 0$ otherwise. Then $[\![p \wedge q]\!]_v = 1$ and therefore $p \wedge q$ is satisfiable.
2. Let $\varphi = p \wedge \neg p$. Then for any valuation v, we have

$$[\![\varphi]\!]_v = \min\{v(p), v(p) \Longrightarrow 0\} = \left\{ \begin{matrix} \min\{0,1\}, & v(p) = 0 \\ \min\{1,0\}, & v(p) = 1 \end{matrix} \right\} = 0$$

   and thus φ is unsatisfiable.
3. The formula $p \to p$ is a tautology: Let v be any valuation. Then

$$[\![p \to p]\!]_v = v(p) \Longrightarrow v(p) = \left\{ \begin{matrix} 1, & v(p) \leq v(p) \\ 0, & \text{otherwise} \end{matrix} \right\} = 1$$

4. Finally, we have $p \wedge q \equiv q \wedge p$: For any valuation v we have that

$$[\![p \wedge q]\!]_v = \min\{v(p), v(q)\} = \min\{v(q), v(p)\} = [\![q \wedge p]\!]_v$$

3.5. Semantic Deduction

Pierce's excursion has left Isaac is confused: "I can see what tautologies and satisfiable formulas are, but how does definition 3.10 relate to deducing propositions?" First of all, you will appreciate that semantic equivalence says that two formulas have the same meaning and that one can be replaced by the other, whenever we speak about the semantics of formulas. For instance, all tautologies are semantically equivalent.

More importantly, however, we have not talked about the entailment relation $\Gamma \vDash \varphi$ between sets of formulas Γ and formulas φ, yet. The idea is that we may want to make some assumptions, like $r \wedge u \to w$ and $u \wedge \neg w$, and deduce a formula like $\neg r$. "Ooh, that looks familiar! We have seen something like this before in example 2.4 during our discussion with Aristotle.", Clara's eyes sparkle in delight. Yes, you can see proofs with assumptions in two ways: either you prove that

$$(r \wedge u \to w) \wedge (u \wedge \neg w) \to \neg r$$

is a tautology, or you prove the entailment

$$\{r \wedge u \to w, u \wedge \neg w\} \vDash \neg r\,.$$

Theorem 3.12

Let $\Gamma, \Delta \subseteq \mathrm{PForm}$ and $\varphi, \psi \in \mathrm{PForm}$. Then the following holds.

i) If $\Gamma \subseteq \Delta$ and $\Gamma \vDash \varphi$, then $\Delta \vDash \varphi$. (*Monotonicity*)

ii) If $\Gamma \vDash \varphi$ and $\Gamma \cup \{\varphi\} \vDash \psi$, then $\Gamma \vDash \psi$. (*Transitivity* or *semantic cut*)

iii) $\Gamma \vDash \varphi \to \psi$ iff $\Gamma \cup \{\varphi\} \vDash \psi$. (*Semantic deduction*)

iv) If $\Gamma \vDash \varphi$ and $\Gamma \vDash \varphi \to \psi$, then $\Gamma \vDash \psi$. (*Modus ponens*)

Proof. Let $\Gamma, \Delta \subseteq \mathrm{PForm}$ and $\varphi, \psi \in \mathrm{PForm}$. For a valuation v, we will write $\Gamma^v = \{[\![\psi]\!]_v \mid \psi \in \Gamma\}$ to simplify notation.

i) Suppose that $\Gamma \subseteq \Delta$ and $\Gamma \vDash \varphi$. We obtain from that first assumption that $\Gamma^v \subseteq \Delta^v$ and thus $[\![\Delta]\!]_v = \min \Delta^v \leq \min \Gamma^v = [\![\Gamma]\!]_v$ (note the reversed order). Combined with the second assumption, we get $[\![\Delta]\!]_v \leq [\![\Gamma]\!]_v \leq [\![\varphi]\!]_v$ and thus $\Delta \vDash \varphi$ as desired.

ii) Suppose that $\Gamma \vDash \varphi$ and $\Gamma \cup \{\varphi\} \vDash \psi$, and let v be a valuation. We get

$$\begin{aligned} &[\![\Gamma]\!]_v \\ &= \min\{[\![\Gamma]\!]_v, [\![\varphi]\!]_v\} && \text{by the assumption } [\![\Gamma]\!]_v \leq [\![\varphi]\!]_v \\ &\leq [\![\psi]\!]_v && \text{by the second assumption} \end{aligned}$$

and thus $\Gamma \vDash \psi$.

iii) This item requires an intermediate result that is left as an exercise: For all $x, y, z \in \mathbb{B}$, we have that

$$x \leq y \Longrightarrow z \text{ iff } \min\{x, y\} \leq z\,.$$

Using this result and by applying the definition 3.8 twice, we obtain

$$\begin{aligned} &[\![\Gamma]\!]_v \leq [\![\varphi \to \psi]\!]_v = [\![\varphi]\!]_v \Longrightarrow [\![\psi]\!]_v \\ &\text{iff } [\![\Gamma \cup \{\varphi\}]\!]_v = \min(\Gamma \cup \{\varphi\})^v = \min\{[\![\Gamma]\!]_v, [\![\varphi]\!]_v\} \leq [\![\psi]\!]_v \end{aligned}$$

and thus $\Gamma \vDash \varphi \to \psi$ iff $\Gamma \cup \{\varphi\} \vDash \psi$.

iv) This follows from items ii) and iii). □

Clara gets excited: "We should be able to use theorem 3.12 to formalise Aristotle's syllogism, shouldn't we?". Yes indeed, this is possible.

Example 3.13

Let us define $\Gamma = \{r \wedge u \to w, u \wedge \neg w\}$ and show that the entailment

$$\Gamma \vDash \neg r$$

holds. By definition of negation and theorem 3.12.iii), we can instead prove $\Gamma \cup \{r\} \vDash \bot$. We shall denote $\Gamma \cup \{r\}$ henceforth by Γ_1. By applying theorem 3.9 repeatedly, we obtain

$$\Gamma_1 \vDash u \text{ and } \Gamma_1 \vDash r$$

and therefore

$$\Gamma_1 \vDash u \wedge r\,.$$

Since $r \wedge u \to w \in \Gamma_1$ we can use modus ponens (theorem 3.12.iv)) to obtain

$$\Gamma_1 \vDash w\,. \tag{3.1}$$

On the other hand, we can apply theorem 3.9 to obtain

$$\Gamma_1 \vDash \neg w \tag{3.2}$$

because $u \wedge \neg w \in \Gamma_1$. By applying modus ponens to eqs. (3.1) and (3.2), we find that $\Gamma_1 \vDash \bot$ and thus $\Gamma \vDash \neg r$.

Isaac: This is all good and well, but example 3.13 has too much natural language for my taste! How could this kind of reasoning underlie any of my inner workings?

Clara: Indeed, the reasoning requires quite some ingenuity, no offence Isaac, and does not lend itself to automatic computation.

Isaac: No offence taken!

Clara: Even for me, the reasoning is hard to follow.

Peirce: You are absolutely right and I would advise not to carry out deductions this way. Even though the principles established in theorems 3.9 and 3.12 are useful, they do not tell us how to organise proofs.

Clara: Then we should learn how to organise deductions! Can you help us to do that?

Peirce: I'm afraid that your time here is up and you I will have to refer you to two other logicians. The first logician will introduce you to the exciting world of proof theory, which offers powerful tools to organise formal deductions and make them computer-verifiable. The other logician is more interested in letting computers figure out deductions automatically. This is an ideal task to test the limits of your computing machinery, Isaac.

Then, off you go! I wish you farewell and great continuation of your voyage!

4. Proof Theory of Propositional Logic

Isaac: What is going on here? Who are all these people sitting in the fields, on the hills and in the garden of this villa?

Peano: Welcome to my home! These are the women working in the cotton mills of Torino. They are asking for their rights: limiting working days to 10 hours, lunch breaks, costs for working material should be covered by the factory owner etc. etc.

Clara: Professor Peano, what a pleasure to meet you! These are indeed very reasonable requests.

Isaac: It seems that some of these requests have been reversed in our time.

Clara: Yes, that is unfortunately true. I guess that we could at some point use logic and computers to prove that rights will remain unstable, unless workers control their workspace and students control their educational institution.

Peano: What a wonderful idea! Although, students used to be in charge at the Universities of Bologna and Paris in the Middle Ages. In any case, what brings you two to my home?

Clara: Our visits to Aristotle and Charles Sanders Peirce have taught us how to formalise propositions as formulas and how semantically derive formulas from sets of assumptions. However, this left us a bit dissatisfied.

Peano: In which way?

Clara: Deductions via entailment depend on a particular semantics, which is contrary to the idea that understanding logic lies in understanding the differences of meaning through studying different models.

Isaac: And deductions made with entailment still use a lot of natural language.

Peano: In short, you are looking for is a formal syntax for proofs, not only for formulas.

Clara: Indeed, I haven't thought about it that way.

Peano: Formal proofs are what has guided my *Formulario* project. Please, come to my terrace and enjoy the worm evening with me over a glass of wine. I will tell you about a system that was conceived by Gerhard Gentzen [Gen35], with whom I disagree politically, but who paved the way for natural approaches to deduction.

Let us begin with what an informal, but rigorous, proof consists of. One of the main issues, which had been identified already by the Indian logicians [Gan04, Sec. 3.6 and Chap. 4], is that we need to be absolutely clear about the assumptions that we make and what we want to prove. Otherwise, we can easily prove anything, like the existence of any kind of god, without providing any actual evidence. Once this is stated, we can go state the proof method and steps. You may enumerate the ingredients of a proofs thus as follows, but be careful that this list is only an informal statement and may vary. It might be beneficial to learn about proofs also from other people [Sol13].

I) Fix the background theory (axioms, definitions, deduction rules etc.)

II) State all the assumptions of the proposition ("Suppose that ...")

III) State the proposition to be proved

IV) State the proof method ("By induction on ...", "By contraposition ...") and provide the proof steps that deduce the proposition from the assumptions (involving the background theory)

Peano: You have seen informal proofs already on your journey, I understand, and you should be able to match the above scheme to those proofs. Only that it will be difficult to pin down the exact background theory used in those proof. How can we, therefore, be sure that those proofs are correct?

Isaac: If they were is unambiguous format, then I could check all the steps and, assuming that the computations did not go wrong anywhere, could say with certainty that they are correct.

Clara: That would be great! But at the same time I would like to be able read and write these proofs easily, as I would like to communicate them still with other humans.

Peano: This is a very good point! Before we get to such *natural deduction* systems, let me first start with making proofs unambiguous and introduce you to deductive systems.

4.1. Deductive Systems

Over time, logicians have developed a vast range of deduction methods. What I would like to focus on here are deductive systems that formalise proofs of syntactic formulas without, a priori, referring to semantics. Here is a list, certainly incomplete, of such methods.

- Axiomatic proof theory
- **Natural deduction**
- **Sequent calculus**
- **Tableaux methods**
- **Uniform and focalised proofs**
- Algebraic proof theory
- Type theory
- Category theory

The methods **marked** in this list are based on a tree representation of proofs. This makes them easy to understand, while still being suitable for automation. We typically refer to the study of those as *Structural Proof Theory*.

In order to use a deductive systems in the formalisation of proofs, as outlined above, we have to

I) fix a deductive system I with background theory,

II) list our assumptions Γ,

III) represent the proposition as formula φ, and

IV) construct a proof for φ from Γ in I, which is written as

$$\Gamma \vdash_I \varphi\,.$$

We call this relation a *(hypothetical) judgement* [PD01].

"Why hypothetical?", Isaac is puzzled. Because we cannot expect to find proofs for any combination of Γ and φ. You may read $\Gamma \vdash_I \varphi$ as the result that we wish to prove and we will be able to say that this judgement *holds*, once we have found a proof.

Now the question is of course how a deductive system looks like and we will soon get to that. But we should first understand that not any deductive system may be useful. For example, we may find a deductive system questionable if it allows us to prove that all cats fly or $1 + 1 = 1$ in the natural numbers. As a first step, we may require that it is not possible to prove anything absurd in I, that is,

$$\vdash_I \bot \text{ is not provable in } I\,. \tag{Consistency}$$

This requirement is, unfortunately, often to weak or maybe even undesirable in some situations [Gan04; PTW18]. Thus, we usually give a reference that determines which judgements may be provable. Fortunately, you know already such a reference: the Boolean semantics. We thus require that

$$\text{if } \Gamma \vdash_I \varphi \text{ is provable, then } \Gamma \vDash \varphi\,. \tag{Soundness}$$

This says that any valid judgement must also give a valid entailment in the semantics. At this point, you may rightfully ask if we should not also ask that any valid entailment should give a proof. This is a quite strong requirement that many people lost a lot of sweat over:

$$\text{if } \Gamma \vDash \varphi \text{ then } \Gamma \vdash_I \varphi \text{ is provable.} \tag{Completeness}$$

Completeness is not only difficult to prove but it can also fail; sometimes spectacularly, sometimes for mundane reasons. It has inspired in any case the work of generations of logicians!

4.2. Natural Deduction

Clara: Fine, we know now what deductive systems are and what we desire of them, but how does such a system look like *concretely*?

Peano: No worries, we get to that. We will start with a deductive system that is very natural [Pra06] in the sense that deductions in that system correspond to intuitive reasoning and, more importantly, deductions can be reduced to very direct proofs. This latter sense of natural enables automatic deduction methods, but that is for another time. For the moment, let us think about how we would naturally prove propositions.

The deductive system that will allows this is Gentzen's system of *natural deduction*. This system is based on judgements of the form $\Gamma \vdash \varphi$, where Γ is a list of (propositional) formulas and φ is a single formula.

Conjunction Let us begin with the case of conjunction and suppose that we want to prove $\varphi \wedge \psi$. Intuitively, we would expect this proposition to be true if φ and ψ are separately true. In the terminology of deductive systems, we should thus be able to derive from the judgements

$$\Gamma \vdash \varphi \quad \text{and} \quad \Gamma \vdash \psi$$

that the judgement

$$\Gamma \vdash \varphi \wedge \psi$$

holds. As it is such an natural step, we will make it a *deduction rule*:

$$\frac{\Gamma \vdash \varphi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\ (\wedge\mathrm{I})$$

This rules consists of hypotheses, the conclusion and a label to name the rule. The hypotheses above the line are what we have to prove before we can apply the rule. Once we have proved the hypotheses, we can deduce the conclusion using the rule. We use labels to identify in proofs the rules that we use, which helps both readability and allows us to verify proofs. In the case of the rule above, we used the label ($\wedge$I) and read it as "conjunction introduction". The conclusion is a conjunction that has thus been introduced, hence the name of the rule. You will have noticed that we used formulas φ and ψ but did not say what these formulas exactly are. In fact, the rule works for any formula and we can see deduction rules as *schemes* that are agnostic to the structure of φ and ψ. With the rule ($\wedge$I), we are now able to prove conjunctions.

More generally, deduction rules are of the form

$$\frac{J_1 \qquad J_2 \qquad \cdots \qquad J_n}{J}\ (\mathrm{L})$$

where all the hypotheses $J_1, \ldots, J_n$ and the conclusion J are judgements, and L is a rule label. The label indicates whether we are introducing a connective, like ($\wedge$I) above, or are *eliminating* a connective, which means that the connective appears among the hypotheses.

Why would we need to eliminate a connective? Suppose that we know, for example from the assumptions of our proposition, that $\varphi \wedge \psi$ holds. In this

case, we also know that φ and ψ hold separately. This leads us to the following two rules.

$$\frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}\ (\wedge E_1) \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}\ (\wedge E_2)$$

In these rules, the "E" stands for elimination and the number indicate which of the conjuncts we would like to access.

Implication I hope that you recall that result called Modus Ponens from theorem 3.12, which states that if $\Gamma \models \varphi \to \psi$ and $\Gamma \models \varphi$ holds, then also $\Gamma \models \psi$ holds. The intuition is that if we can prove the condition (antecedent) of an implication, then we know that the conclusion of the implication holds. To use this in natural deduction proofs, we turn this idea into a rule:

$$\frac{\Gamma \vdash \varphi \to \psi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\ (\to E)$$

But how can we prove an implication? The same theorem 3.12 gives us the answer in item item iii) because it says that $\Gamma \models \varphi \to \psi$ holds if $\Gamma \cup \{\varphi\} \models \psi$ holds. As you can see, the theory Γ gets extended with φ. The use of the union of sets does not exactly match with our management of assumptions in judgements as lists. The main difference is that lists can contain several times the same formulas, while sets cannot. This is not strictly needed but becomes important in automatic deduction. Let us, thus, write Γ, φ for the *concatenation* of the list Γ and the formula φ. This also saves us some braces! With this notation for lists, we can now provide the introduction rule for implication.

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\ (\to I)$$

What this rule does is to turn the obligation to prove an implication into an obligation to prove ψ with φ as extra assumption.

This raises the question how such an assumption can be used. Let us write $\varphi : \Gamma$ if the formula φ appears in the list Γ. The rule for using an assumption allows us to pick any formula from Γ and use it as proven proposition.

$$\frac{\varphi : \Gamma}{\Gamma \vdash \varphi}\ (\text{Assum})$$

Isaac: These are various rules for proving propositions, but how does this help us to organise proofs any better than the semantic deduction did?

Peano: Think about what a deduction is: a chain of reasoning steps.

Clara: Like a list of rules?

Peano: Almost, except that some rules have more than one hypothesis and we will have to use trees instead of lists.

Definition 4.1: Natural deduction for propositional logic

The system **ND** of natural deduction for propositional logic is given by the rules in fig. 4.1, where Γ is a list of formulas and φ, ψ and δ are formulas. We write the judgement $\Gamma \vdash \varphi$ as $\vdash \varphi$ is Γ is the empty list.

$$\frac{\varphi : \Gamma}{\Gamma \vdash \varphi}\ (\text{Assum}) \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}\ (\bot\text{E})$$

$$\frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}\ (\wedge\text{E}_1) \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}\ (\wedge\text{E}_2) \qquad \frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\ (\wedge\text{I})$$

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}\ (\vee\text{I}_1) \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}\ (\vee\text{I}_2)$$

$$\frac{\Gamma \vdash \varphi \vee \psi \quad \Gamma, \varphi \vdash \delta \quad \Gamma, \psi \vdash \delta}{\Gamma \vdash \delta}\ (\vee\text{E})$$

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\ (\to\text{I}) \qquad \frac{\Gamma \vdash \varphi \to \psi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\ (\to\text{E})$$

Figure 4.1.: Deduction Rules of the natural deduction system **ND**

Figure 4.1 contains the rules for the two connectives that we did not discuss, yet. The rule ($\bot$E) allows us to deduce *anything* from absurdity. This rule is sometimes also referred to as the *principal of explosion* because, once absurdity has been proven, anything is possible. Finally, there are the rules for disjunction. It should be rather obvious why the introduction rules ($\vee\text{I}_1$) and ($\vee\text{I}_2$) make sense, as $\varphi \vee \psi$ holds whenever φ or ψ holds. The elimination rule ($\vee$E) is a bit more complex. Underlying it is the idea that we want to prove δ, knowing that $\varphi \vee \psi$ holds. As we cannot know which of the two it will be, we have to make a case distinction and prove that δ holds in either case. If we succeed, then δ can be deduced from $\varphi \vee \psi$.

Clara: This is like an if-then-else branch in programming, isn't it?

Peirce: Indeed, branching or case distinction are very closely related to the elimination of disjunction. You can think of it this way: The rule builds a proof of δ that takes a proof of $\varphi \vee \psi$ as input. To proceed, you make a case distinction on this input and check whether it proves φ or ψ. You can then use this information to prove δ for both cases separately.

Clara: Mh, that would suggest that we can write proofs as programs.

Peirce: It does! And you may be delighted to hear that the aforementioned area of type theory concerns itself with interpreting proofs as programs. Very fascinating, but we shall continue for the moment with natural deduction.

The missing piece of the puzzle is a way of assembling the rules together into complex proofs.

Definition 4.2

Let Γ be a list of formulas and φ a formula. We call the pair $\Gamma \vdash \varphi$ a *(hypothetical) judgement* of **ND**. A *deduction* (or *proof tree*) for $\Gamma \vdash \varphi$ is a finite tree, such that

i) each node is labelled with a judgement and a rule label, such that the children of the node are labelled with the hypotheses of that rule;

ii) the root of the tree is labelled with $\Gamma \vdash \varphi$; and

iii) the leaves of the tree are (necessarily) labelled with (Assum).

We say that φ holds under hypotheses Γ in **ND**, if there is a deduction of $\Gamma \vdash \varphi$. A formula φ is a *theorem* if $\vdash \varphi$ holds.

Definition 4.2 may look intimidating at first sight, but it really just formalises our intuition of proofs. To illustrate this, let us go through some simple examples.

Example 4.3

In this example, we show how the (Assum)-rule and ($\to$I)-rule can be used together to deduce $\vdash p \to q \to p$. The formula $p \to q \to p$ also allows us to "store" knowledge and is one of the building blocks of

combinatory logic in form of the K-combinator. Here is the proof tree for the deduction:

$$\dfrac{\dfrac{\dfrac{p : p, q}{p, q \vdash p}\,(\text{Assum})}{p \vdash q \to p}\,(\to\text{I})}{\vdash p \to q \to p}\,(\to\text{I})$$

As writing $\varphi : \Gamma$ in (Assum) is tedious, we allow ourselves to leave this out whenever $\varphi : \Gamma$ obviously holds. Moreover, as the label (Assum) is also quite long, we will typically shorten the rule to

$$\dfrac{}{\Gamma \vdash \varphi}$$

if the application of (Assum) is obvious. In non-obvious cases, we will still indicate the occurrence of φ in Γ.

Example 4.4

We deduce $p \wedge q \to r \vdash p \to q \to r$. This process is also known as *Currying*. As the judgements can get quite lengthy, let us name the assumptions throughout the proof by defining $\Gamma = p \wedge q \to r$.

$$\dfrac{\dfrac{\dfrac{\dfrac{p \wedge q \to r : \Gamma}{\Gamma, p, q \vdash p \wedge q \to r} \qquad \dfrac{\dfrac{}{\Gamma, p, q \vdash p} \quad \dfrac{}{\Gamma, p, q \vdash q}}{\Gamma, p, q \vdash p \wedge q}\,(\wedge\text{I})}{\Gamma, p, q \vdash r}\,(\to\text{E})}{\Gamma, p \vdash q \to r}\,(\to\text{I})}{\Gamma \vdash p \to q \to r}\,(\to\text{I})$$

As you can see, we have simplified the application of (Assum) in the two right leaves, but added some information on the left leaf to make the proof easier to read.

"Aristotle gave us an example that Prof. Peirce said we could easily prove with what we learn here.", Clara says. "This example use negation and, if I remember correctly, then negation was defined in terms of implication and absurdity. Can we thus use the deduction rules for those connectives to reason about negation and prove the said example?"

Absolutely! But I would advise to make your life simpler and first establish some specific rules for negation inside the system **ND**. These rules would not

be part of the system itself but can be derived from it and can be used *like* proof rules: they are admissible.

Definition 4.5

A rule

$$\frac{J_1 \quad J_2 \quad \cdots \quad J_n}{J}\ (\mathrm{L})$$

is *admissible*, if there it is possible to construct from any deduction of the judgements $J_1, \ldots, J_n$ a deduction for the judgement J.

Now we can make your life easier by providing admissible rules that determine how negation can be handled in **ND**.

Theorem 4.6

The following two rules are admissible in **ND**.

$$\frac{\Gamma, \varphi \vdash \bot}{\Gamma \vdash \neg\varphi}\ (\neg\mathrm{I}) \qquad \frac{\Gamma \vdash \varphi \quad \Gamma \vdash \neg\varphi}{\Gamma \vdash \psi}\ (\neg\mathrm{E})$$

Proof. Let φ and ψ be formulas. First, we suppose that $\Gamma, \varphi \vdash \bot$ holds. We then obtain immediately a proof tree

$$\frac{\Gamma, \varphi \vdash \bot}{\Gamma \vdash \varphi \to \bot}\ (\to\mathrm{I})$$

and thus a proof tree for $(\neg\mathrm{I})$ because $\neg\varphi$ is defined to be $\varphi \to \bot$.

Next, suppose that there are proof trees for $\Gamma \vdash \varphi$ and $\Gamma \vdash \neg\varphi$. We can then build the following proof tree by using again the definition of $\neg\varphi$ in terms of $\varphi \to \bot$.

$$\frac{\dfrac{\Gamma \vdash \varphi \quad \Gamma \vdash \neg\varphi}{\Gamma \vdash \bot}\ (\to\mathrm{E})}{\Gamma \vdash \psi}\ (\bot\mathrm{E})$$

This shows that both, $(\neg\mathrm{I})$ and $(\neg\mathrm{E})$, are admissible in **ND**. □

Using these two rules, we can easily prove judgements that involve negation. In particular, we can prove the statement by Aristotle!

Example 4.7: It does not rain for Socrates

Let $\Gamma = r \wedge u \to w, u \wedge \neg w$. The following is a proof for $\Gamma \vdash \neg r$.

$$\dfrac{\dfrac{\dfrac{}{\Gamma, r \vdash r \wedge u \to w} \quad \dfrac{\dfrac{}{\Gamma, r \vdash r} \quad \dfrac{\dfrac{}{\Gamma, r \vdash u \wedge \neg w}\,(\wedge \mathrm{E}_1)}{\Gamma, r \vdash u}}{\Gamma, r \vdash r \wedge u}\,(\wedge \mathrm{I})}{\Gamma, r \vdash w}\,(\to \mathrm{E}) \quad \dfrac{\dfrac{u \wedge \neg w : \Gamma}{\Gamma, r \vdash u \wedge \neg w}\,(\wedge \mathrm{E}_2)}{\Gamma, r \vdash \neg w}}{\dfrac{\Gamma, r \vdash \bot}{\Gamma \vdash \neg r}\,(\neg \mathrm{I})}\,(\neg \mathrm{E})$$

"I would like to see also some proofs that involve the elimination of disjunction.", says Isaac. "Could we go over that as well?"

Most certainly. Here is an interesting example that shows that, if we can exclude on alternative of a disjunction, then the other necessarily has to hold.

Example 4.8

We put $\Gamma = p \vee q, \neg q$ and prove $p \vee q, \neg q \vdash p$ by means of the following deduction.

$$\dfrac{\dfrac{}{\Gamma \vdash p \vee q} \quad \dfrac{}{\Gamma, p \vdash p} \quad \dfrac{\dfrac{}{\Gamma, q \vdash q} \quad \dfrac{\neg q : \Gamma}{\Gamma, q \vdash \neg q}\,(\text{Assum})}{\Gamma, q \vdash p}\,(\neg \mathrm{E})}{\Gamma \vdash p}\,(\vee \mathrm{E})$$

Note that ($\vee$E) acts like a case distinction and we show in the second, were we suppose that q holds, that this case is absurd. This allows us to show that $\Gamma \vdash p$ must hold.

In the final example, we show that the negation of a disjunction is very similar to the conjunction of the negation of the subformulas of the disjunction. This is one side of one of de Morgan's law, which holds in the Boolean semantics in that $\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$. You can try to prove the other direction yourself.

Example 4.9

We prove $\neg(\varphi \vee \psi) \vdash \neg\varphi$ for all formulas φ and ψ. Let $\Gamma = \neg(\varphi \vee \psi)$.

$$\dfrac{\dfrac{\dfrac{}{\Gamma, \varphi \vdash \neg(\varphi \vee \psi)} \quad \dfrac{\dfrac{}{\Gamma, \varphi \vdash \varphi}}{\Gamma, \varphi \vdash \varphi \vee \psi}\,(\vee \mathrm{I}_1)}{\Gamma, \varphi \vdash \bot}\,(\neg \mathrm{E})}{\Gamma \vdash \neg\varphi}\,(\neg \mathrm{I})$$

"What about the equivalence $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$?", Clara wonders. That equivalence is surprisingly more delicate! Keep it in mind until we come to section 4.5. In that section, we will prove properties of proof trees and the system **ND**. Let me, before we continue with examples, tell you how we can do that. Recall that we showed in corollary 2.8 how properties of formulas can be proved by formula induction. A similar result can also be stated for proof trees.

Theorem 4.10: Proof tree induction

Let P be a property of proof trees. Suppose further that

i) P holds for all single-node proof trees labelled by (Assum) (P contains all instances of the rule (Assum)) and;

ii) for all proof trees $T_1, \dots, T_n$ for judgement $J_1, \dots, J_n$ that match a rule (L) with conclusion J, if $P(T_1), \dots, P(T_n)$ hold, then P also holds for the proof tree

$$\dfrac{J_1 \quad \cdots \quad J_n}{J}\,(\mathrm{L})$$

Under these conditions, the property P holds for all proof trees.

Analogously to induction on natural numbers and formulas, we will refer to item i) as base case and item ii) as the induction step. We will not go over this, but you may imagine how theorem 4.10 can be proven by induction over the tree height and by breaking proof trees down in subtrees.

4.3. Fitch-Style Natural Deduction

Clara: The system **ND** seems to indeed formalise our intuition of proofs, as we can clearly see the proof unfolding and check each step ...

Isaac: ... probably even automatically with a computer.

Clara: Indeed! However, I find it a bit worry-some that the trees grow so much in width. How can I bigger proofs on an A4 page?

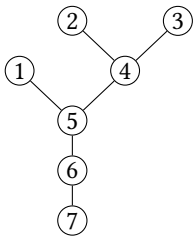
Why not consider using an interactive presentation of proof trees, in which you can focus on parts of the tree? But if you insist on using paper, then there are several ways to represent trees in a way that they only grow in one rather than two dimensions. This makes it possible to represent large proof trees on paper, albeit with two disadvantages:

1. the proofs are harder to read because the branches of proof trees will be typically be scattered over the page, and
2. writing proofs requires either good planning or computer support.

With a bit of discipline and experience, the second issue will have negligible impact. The first issue is more severe, but we shall introduce a format for proofs that allows for a bit of two-dimensional structure and features numbered references to help readability.

Consider, for example, the proof tree from example 4.4. This proof tree is shown below on the left. Let us draw a tree that corresponds to this proof tree but only has numbers as labels. You may find this tree on the right:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, p, q \vdash p \wedge q \to r} \qquad \dfrac{\dfrac{}{\Gamma, p, q \vdash p} \quad \dfrac{}{\Gamma, p, q \vdash q}}{\Gamma, p, q \vdash p \wedge q}\,(\wedge\mathrm{I})}{\Gamma, p, q \vdash r}\,(\to\mathrm{E})}{\Gamma, p \vdash q \to r}\,(\to\mathrm{I})}{\Gamma \vdash p \to q \to r}\,(\to\mathrm{I})$$



As you can imagine, we can make a list of all the nodes and refer to them by their numbers instead of drawing the edges. If you have ever implemented trees with pointers or arrays, then this will be very familiar. Also, if you ever read some classical text on mathematical logic, then you will see that proofs are often represented in this way. But just listing the proof steps and using numbered references misses the introduction of premises. For instance, we

introduce a new assumption in node 7 that we will have to record. This leads us to *Fitch-style proofs*, which combine numbered proof steps and so-called *flags* that indicate the introduction of assumptions. Let me show you how the above proof tree looks like in Fitch-style:

1	$p \wedge q \to r$	
2	$\quad p$	
3	$\quad\quad q$	
4	$\quad\quad p \wedge q$	$\wedge$I, 2, 3
5	$\quad\quad r$	$\to$E, 1, 4
6	$\quad q \to r$	$\to$I, 3–5
7	$p \to q \to r$	$\to$I, 2–6

Fitch-style proofs use as numbering schemes the line numbers of a proof, going from top to bottom. The mentioned flags are in line 1, 2 and 3, and we say that we *open flags* in those lines. A flag indicates that everything proved within it, requires the premises written on top of that flag. For instance, the whole proof ranging from line 2 to 7 requires the premise $p \wedge q \to r$ in line 1, the proof ranging from line 3 to 6 requires the premise p in line 2 and so forth. We also indent flags a bit and retain some two-dimensional element, which helps readability a lot. Except for premises made, we indicate on the right of a Fitch-style proof the proof rule that we use to obtain the formula in that line together with the involved line numbers.

Isaac: This is a very compact presentation of proofs! But why do we write "2,3" in line 4 but "3–5" in line 6?

Peano: The answer lies in the proof rules that are used. Notice that ($\wedge$I) does not introduce new premises and merely combines two proven formulas into a conjunction. In contrast, the introduction of implication makes a new hypothesis: to prove $q \to r$, we assume q and deduce r. This what the flag ranging from line 3 to 5 indicates. Thus, the conclusion of $q \to r$ in 6 needs to refer to the whole *sub-proof* that ranges from line 3 to 5.

Clara: In other words: Whenever we change our premises Γ and prove the resulting hypothetical judgement, then we have to create a sub-proof by opening a flag, and then we have to refer to that whole sub-proof?

Peano: Precisely!

Clara: Looking back at the rules of **ND** in fig. 4.1, it appears that also the ($\vee$E)-rule changes the premises, but for two sub-proofs! How do we deal with that?

Peano: In exactly the same way.

Recall that we proved $p \vee q, \neg q \vdash p$ in example 4.8. First of all, note that this judgement has two premises, which we will write on two separate lines in the proof. The goal of the proof is then to prove p from the these two premises:

1	$p \vee q$	
2	$\neg q$	
3	$\quad p$	
4	$\quad p$	Assum, 3
5	$\quad q$	
6	$\quad p$	$\neg$E, 5, 2
7	p	$\vee$E, 1, 3–4, 5–6

The ($\vee$E)-rule has been used in this proof on line 7 and refers to the disjunction in line 1 that is eliminated and to the two sub-proofs in lines 3–4 and 5–6.

In general, a proof in Fitch-style for a hypothetical judgement $\varphi_1, \dots, \varphi_n \vdash \psi$ will always have the following outline.

1	φ_1	
2	$\vdots$	
n	φ_n	
$n+1$	$\vdots$	
	$\quad \gamma$	
	$\quad \vdots$	
m	$\quad \delta$	
$m+1$	$\vdots$	
k	ψ	L

Everything above the horizontal line that opens a flag are the assumptions or hypotheses, which can be used inside the flag and will be *discharged*, once the flag closes. For example, $\varphi_1, \dots, \varphi_n$ are the assumptions under which ψ holds, and γ is the assumption that is used to prove δ. With line m, γ will be discharged is no longer usable from line $m+1$ on. This corresponds to proving the judgement $\varphi_1, \dots, \varphi_n, \gamma \vdash \delta$ by means of a rule like ($\to$I), as we did in the first Fitch-style proof above in lines 6 and 7. In the same proof, you can also see that flags can be nested arbitrarily. Finally, every line, which is not a hypothesis, needs to be given a label L that states used the rule and the lines that contain the information necessary to apply this rule.

Peano: I refrain from giving you a precise definition of Fitch-style proofs because this is rather cumbersome. Instead, I trust that you are able to understand intuitively how such proofs are formed

Clara: Is there no way to ensure that a Fitch-style proof is correct?

Peano: Of course, Fitch-style proofs are just a different way of writing **ND**-proofs. Thus, if it is possible to translate a given Fitch-style proof into **ND**, then it will be correct.

Isaac: Would it be possible to get one more example, on which I can apply my learning algorithm?

Peano: Surely.

Example 4.11: It does not rain for Socrates — Fitch-style

In example 4.7, we have proved the judgement $r \wedge u \to w, u \wedge \neg w \vdash \neg r$

using proof trees. Here is now the same proof in Fitch-style:

1	$r \wedge u \to w$	
2	$u \wedge \neg w$	
3	$\quad r$	
4	$\quad u$	$\wedge$E, 2
5	$\quad r \wedge u$	$\wedge$I, 3, 4
6	$\quad w$	$\to$E, 1, 5
7	$\quad \neg w$	$\wedge$E, 2
8	$\quad \bot$	$\neg$E, 6, 7
9	$\neg r$	$\neg$I, 3–8

You should be able to match all the steps to the original proof. One small difference is that we do not indicate any longer which conjunct we want to obtain from the application of the elimination rules for the conjunction. However, it is easy to recover the indices by inspecting the involved formulas. We will allow ourselves the same simplification for the introduction rules of the disjunction.

?

Which of the following two Fitch-style proof attempts is correct? If any, which hypothetical judgement does it prove?

1	p	
2	q	
3	p	Assum, 1
4	$q \to p$	$\to$I, 2–3

1	p	
2	$\quad r$	
3	$\quad r \wedge p$	$\wedge$I, 1, 2
4	$r \to p \wedge r$	$\to$I, 2–3

4.4. Soundness and Consistency

Isaac: Speaking of correctness, I found myself wondering why the natural deduction proofs that we have constructed so far are correct.

Clara: Also, what does it even mean for a proof to be correct?

Peano: This is indeed a valid point that we have glossed over so far. Remember that you have seen the semantics of formulas in terms of Boolean truth values. Using these semantics, you could say that a set of formulas entails another formula, written $\Gamma \vDash \varphi$. This means that, inside the Boolean model, the formula φ is true whenever all the formulas in Γ are true.

Clara: I guess, it is not a coincidence that the entailment symbol and the judgement turnstile look very similar.

Peano: Indeed, the idea is that we can say that the deductive system **ND** is correct with respect to the Boolean model, which means that if $\Gamma \vdash \varphi$ is provable, then $\Gamma \vDash \varphi$. This property is also called *soundness*.

Clara: Does this mean that we cannot generally say that deductive system is correct?

Peano: No, soundness is always relative to a chosen model or a class of model with some properties.

Here, we will focus on soundness relative to the Boolean model.

Theorem 4.12: Soundness of ND

If $\Gamma \vdash \varphi$ is derivable in **ND**, then $\Gamma \vDash \varphi$.

Proof. The proof proceeds by induction on proof trees for judgements.

i) In the base case, we have that $\Gamma \vDash \varphi$ must be deduced by (Assum) and therefore that φ appears in Γ. Thus, $\Gamma \vDash \varphi$ by theorem 3.9.

ii) For the induction step, we proceed by case distinction on the applied rule.

- Suppose that we have a proof tree for $\Gamma \vdash \varphi \to \psi$ labelled by ($\to$I). This means that the judgement $\Gamma, \varphi \vdash \psi$ is also derivable and the induction hypothesis gives us that $\Gamma, \varphi \vDash \psi$. From item iii) of theorem 3.12 we obtain $\Gamma \vDash \varphi \to \psi$, as desired.

- Suppose that we have a proof tree for $\Gamma \vdash \varphi$ labelled by ($\vee$E) and hypotheses $\Gamma \vdash \psi_1 \vee \psi_2$, $\Gamma, \psi_1 \vdash \varphi$ and $\Gamma, \psi_2 \vdash \varphi$. The induction hypothesis gives us $\Gamma \vDash \psi_1 \vee \psi_2$, $\Gamma \cup \{\psi_1\} \vDash \varphi$ and $\Gamma \cup \{\psi_2\} \vDash \varphi$. From the first hypothesis and theorem 3.9 we get that $\Gamma \vDash \psi_1$ or $\Gamma \vDash \psi_2$ holds. In the first case, we use theorem 3.12.ii) (semantic cut) and the second hypothesis $\Gamma \cup \{\psi_1\} \vDash \varphi$ to obtain $\Gamma \vDash \varphi$. The second case is analogous. Thus $\Gamma \vDash \varphi$ holds in either case and we are done
- The cases for the other rules can be dealt with similarly and only require a bit of work.

By induction on proof trees we get that $\Gamma \vdash \varphi$ implies $\Gamma \vDash \varphi$. □

In particular, theorems proven **ND** are always true.

Corollary 4.13

If φ is a theorem of **ND**, that is $\vdash \varphi$ is derivable, then φ is a tautology: $[\![\varphi]\!]_v = 1$ for all valuations v.

A further important consequence is that natural deduction is *consistent*.

Corollary 4.14: Consistency

There is no deduction for $\vdash \bot$ in **ND**.

Proof. Suppose that $\vdash \bot$ is derivable. By corollary 4.13, we would have that $[\![\bot]\!]_v = 1$ for all valuations. This contradicts the definition of $[\![\bot]\!]_v$ and therefore $\vdash \bot$ cannot be derivable. □

4.5. Classical Logic and Completeness

Isaac: This is great news: that means we can prove any entailment using natural deduction and know that such a proof is correct by construction!

Peano: No! There is a misunderstanding here. It is correct to say that any proof in **ND** guarantees that the proven judgements yields an entailment in the Boolean model. However, it is not true that any entailment can be proven! This is what we would call *completeness*.

Clara: What is then all the fuzz about with the formal proofs if we cannot even prove everything that is true?

Peano: There is a good reason to rule out certain true formulas, but we will have to leave that for another time. For the moment, let me show you how to fix the issue of completeness.

First, let us identify the issue.

Theorem 4.15

The judgement $\vdash \neg\neg p \to p$ is not provable in **ND**, but $\neg\neg p \to p$ is a tautology in the Boolean model.

Proof. We have for all valuations v that

$$[\![\neg\neg p \to p]\!]_v = (1 - (1 - [\![p]\!]_v)) \Longrightarrow [\![p]\!]_v = [\![p]\!]_v \Longrightarrow [\![p]\!]_v = 1$$

and thus $\neg\neg p \to p$ is a tautology. To show that $\vdash \neg\neg p \to p$ is not provable in **ND**, we would need to provide a different model and show that **ND** is also sound for that model, while the formula would not be a tautology in that model. As we have already taken quite a bit of time, we shall leave it at that now, but you can find some details in appendix C. □

Theorem 4.15 tells us that there are entailments that hold in the Boolean model, but are not provable. Is there a way to fix this? It turns out that it suffices to turn the formula $\neg\neg p \to p$ into a proof rule to obtain a complete deductive system.

Definition 4.16

The system **cND** (*classical* natural deduction) is obtained by adding to the system **ND** the following rule for *proofs by contradiction*.

$$\frac{\Gamma, \neg\varphi \vdash \bot}{\Gamma \vdash \varphi}\ \text{(Contra)}$$

The rule (Contra) formalises the commonly know proof principle, called "proof by contradiction", that proceeds by proving that the assumption that a property φ does not hold is absurd and then concluding that φ must therefore hold.

With this rule, we can prove in the system **cND** every proposition that is true in the Boolean model.

Theorem 4.17: Soundness and Completeness of cND

A hypothetical judgement $\Gamma \vdash \varphi$ is derivable in **cND** if and only if $\Gamma \vDash \varphi$ holds in the Boolean semantics.

Proof. Soundness is proved like in theorem 4.12, only that we also have to account for the (Contra)-rule. That this rule is correct follows essentially theorem 4.15. Completeness can be proved either by contradiction or by providing an algorithm that constructs a proof trees for hypothetical judgements that correspond to valid entailments [Gal87, sec. 3.4.7]. □

Clara: Why is the proof-by-contradiction rule so special? It seems like a perfectly good proof principle to me.

Peano: There are people who doubt that it can be justified easily. For once, there are perfectly reasonable semantics for propositional logic, in which this proof principle is not valid (appendix C). But already on its own, this proof principle is not very intuitive because it proves a *positive* statement from a *negative* one. This is a story of its own and started a whole school of logic called "constructive logic", which is also closely related to our earlier discussion of the interpretation of proofs as programs. I'm afraid though that this topic will take us too far and we will have to stop here. Before you leave, let me briefly show you what **cND** can prove.

Example 4.18

We can derive the *law of excluded middle (LEM)* in **cND**:

1	$\neg(p \vee \neg p)$	
2	$\neg p$	example 4.9, 1
3	$\neg\neg p$	symmetric example 4.9, 1
4	$\bot$	¬E, 2, 3
5	$p \vee \neg p$	Contra, 1–4

The LEM states that every formula has to be either true or false, which

is exactly what we have in the Boolean model, and that a third possibility is not allowed.

Peano: My friends, we had wonderful evening but I am getting tired now. It seems that all the workers have gone home by now and I will have to clean up my garden. But it was certainly worth it! (*Peano laughs in great content.*)

Clara: Thank you so much for your time and patience Professor Peano. We have greatly enjoyed our time with you and learned many things, even beyond formal logic.

Isaac: If only all humans were as kind and generous as you are Professor, then I may grow to like your kind.

Peano: I don't know what you have seen in your future, but never lose hope that the good in humans will prevail and a society of true solidarity will emerge. You, as a robot, may have the chance to exist that long.

Isaac: I suppose that you have never seen the Film WALL-E?

Peano: Now, please excuse me. I wish you all the best for your journey!

Clara: Before we leave: do you have an idea what we should learn next about logic?

Peano: I would suggest to expand your cultural horizon. Let me send you to an interesting place and time. Goodbye!

Isaac and You: Goodbye, Professor!

5. Automatic Deduction for Propositional Logic

[*Clara and Isaac find themselves on a densely packed market, full of merchants with tents, and people pushing and squeezing to move forward. Everybody is wearing long colourful tunics in various colours, in white, blue and brown. The air is hot and dry, infused with sand and smells. Music is flowing out of a corner of the market, through the alleys. The drum fills the gaps between the people, the flute dances between them and the lute moves makes Clara want to dance.*]

al-Khwārizmī: حاجة !تحرق الدم العمى

Clara: What does he say?

Isaac: He says, "hajat tuharuq aldam! al'ama". He seems to be very annoyed.

Clara: Incredible! All these smells: cardamon, cinnamon, pepper, and turmeric! Look at all these bags full of ingredients. Chickpeas, dates, fresh figs, olives, melons! Hey, what instruments are the people at the corner playing?

al-Khwārizmī: The wind instrument is a *Duduk*, the guitar-like instrument an *Oud*, and the drum is a *Tombak*. But these instruments are not an as unusual appearance here, as you two are. May I help you?

Clara: We are indeed not from here. We are on a journey to learn about logic and to understand the origins of my friend Isaac.

al-Khwārizmī: How is his origin related to logic?

Isaac: You see, I function purely by means of electronics and make decisions based on logic reasoning.

al-Khwārizmī: Remarkable! You need to tell me more about this.

Clara: We do not fully understand his inner working ourselves, yet.

al-Khwārizmī: Then you have come to the right place. Welcome to Baghdad, my friends! Please follow me.

Isaac: This crowd is very dense! Let us try to keep up with him.

al-Khwārizmī: Do you see the bridge ahead? It will take us over the Tigris to the Madīnat as-Salām, the *City of Peace*, where the palace is located.

Clara: The palace?

al-Khwārizmī: Yes, the palace is surrounded by houses, of which one hosts my workplace in the *House of Wisdom*.

[*The three cross the bridge and arrive at a gate that leads through massive stone walls that surround the* City of Peace.]

Isaac: Do we pass this gate?

al-Khwārizmī: Yes, please. Go right in!

Clara: Beautiful, all these trees that fill the area! And it is suddenly much quieter. This is a welcome change.

al-Khwārizmī: Please come in. Welcome to the House of Wisdom.

Clara: Who are all these people?

al-Khwārizmī: They are working on the translation and study of mathematical texts.

Clara: Isaac? Are you all right?

Isaac: Yes, my translation processor was just overloaded. I had to turn it to single-language mode. There are just too many languages being spoken in here.

al-Khwārizmī: Hohoho! Technology is not always a blessing!

Clara: Why were you so annoyed when we met you earlier on the *Souk al-Shorja* market?

al-Khwārizmī: Ah, this donkey insisted that my calculations using Hindu numerals are incorrect and that we should use the Roman system instead. Such ignorance! You know, it is important to choose the right representation, as you will otherwise have great difficulties doing even the simplest computations. But you are not here to listen to my complaints. So let us explore how Isaac is able to carry out automatic deductions.

5.1. Methods of Semantic Deduction

As you have learned in theorem 4.17, it is possible to reduce the task of proving $\Gamma \vdash \varphi$ in **cND** to showing that the entailment $\Gamma \models \varphi$ holds in the Boolean semantics. How can we prove such an entailment? You could, of course, attempt to use a truth table. This will work but requires potentially an enormous amount of work because you have to go through all 2^n possible combinations of truth values, if Γ and φ have n distinct propositional variables. Such would be an attempt of a fool, someone who would try to calculate with Roman numerals!

Instead, let us try to simplify the problem, at least in some cases. First of all, notice for $\Gamma = \varphi_1, \dots, \varphi_n$ that

$$\Gamma \models \varphi \text{ iff } \varphi_1 \wedge \cdots \wedge \varphi_n \wedge \neg\varphi \text{ is unsatisfiable} \tag{5.1}$$
$$\text{iff } \neg\varphi_1 \vee \cdots \vee \neg\varphi_n \vee \varphi \text{ is a tautology} \tag{5.2}$$

The first equivalence (5.1) is easy to prove and understand: $\Gamma \models \varphi$ holds if and only if $\min[\![\Gamma]\!]_v \leq [\![\varphi]\!]_v$ for all valuations v. Suppose now v is any valuation. We then have that

$$\begin{aligned}
\min[\![\Gamma]\!]_v \leq [\![\varphi]\!]_v \text{ iff } & \min[\![\Gamma]\!]_v = 0 \text{ or } [\![\varphi]\!]_v = 1 \\
\text{iff } & [\![\varphi_1 \wedge \cdots \wedge \varphi_n]\!]_v = 0 \text{ or } [\![\neg\varphi]\!]_v = 0 \\
\text{iff } & \min\{[\![\varphi_1 \wedge \cdots \wedge \varphi_n]\!]_v, [\![\neg\varphi]\!]_v\} = 0 \\
\text{iff } & [\![\varphi_1 \wedge \cdots \wedge \varphi_n \wedge \neg\varphi]\!]_v = 0\,.
\end{aligned}$$

Thus, $\varphi_1 \wedge \cdots \wedge \varphi_n \wedge \neg\varphi$ is unsatisfiable exactly when $\Gamma \models \varphi$.

What does the equivalence (5.1) give us? Suppose we have an efficient algorithm to decide whether a formula is satisfiable, then we could use this algorithm to also efficiently decide the entailment $\Gamma \models \varphi$. The problem of deciding whether a formula is satisfiable is also known as SAT and is, generally, an NP-complete problem. This is not really better than writing truth tables! However, there are certain classes formulas that can be efficiently checked for satisfiability. We will see one such class in section 5.4.

The equivalence (5.2) could also be proven directly, but let me introduce you to a different style of reasoning: equational reasoning.

5.2. Algebra of Boolean Logic

As you know very well, we can use equations to prove relations of numbers. For instance, for all (real) numbers a and b, we know that $a + b = b + a$. We call this the law of *commutativity*. Such laws are vital to simplify calculations and prove facts about numbers. For example, we can use commutativity to rearrange and simplify calculations: $a + b + (-a) = a + (-a) + b = b$.

On other occasions, one needs to rearrange terms over real numbers to solve problems. An example might be to find all numbers x, such that $x^2+x-2 = 0$. Such x are called *roots*. You could now go and find a formula to solve this problem, but by the time you find the right formula, you may have already solved this problem by doing a simple calculation:

$$x^2 + x - 2 = x^2 + 2x - x - 2 = (x + 2)(x - 1)\,.$$

This immediately tells you that the roots are $x = -2$ and $x = 1$, as only then the outcome of the multiplication can be zero. And that makes the laws of arithmetic a powerful tool in solving problems with numbers.

Clara: What does this now have to do with logic?

al-Khwārizmī: Think about what it means for a formula φ to be a tautology.

Clara: The formula φ has to be evaluated to 1 under any valuation.

al-Khwārizmī: Indeed. And can you think of any other logic connective that has this property?

Clara: The truth connective $\top$?

al-Khwārizmī: Perfect!

This means that we can reduce the question of whether a formula φ is a tautology to asking whether $\varphi \equiv \top$. Now remember that we found the roots by rearranging a term into a form, where the problem became easy. We will do exactly that in section 5.3, but before we will need to discuss the algebraic laws that govern propositional logic in the Boolean semantics.

Theorem 5.1: Algebraic Laws of Propositional Logic

The following semantic equivalences holds for all propositional formulas φ, ψ, δ in the Boolean semantics.

$$\varphi \wedge (\psi \wedge \delta) \equiv (\varphi \wedge \psi) \wedge \delta \qquad \text{Associativity for } \wedge$$

$$
\begin{aligned}
\varphi \vee (\psi \vee \delta) &\equiv (\varphi \vee \psi) \vee \delta && \text{Associativity for } \vee \\
\varphi \vee (\psi \wedge \delta) &\equiv (\varphi \vee \psi) \wedge (\varphi \vee \delta) && \text{Distributivity } \vee \text{ over } \wedge \\
\varphi \wedge (\psi \vee \delta) &\equiv (\varphi \wedge \psi) \vee (\varphi \wedge \delta) && \text{Distributivity } \wedge \text{ over } \vee \\
\varphi \wedge \psi &\equiv \psi \wedge \varphi && \text{Commutativity of } \wedge \\
\varphi \vee \psi &\equiv \psi \vee \varphi && \text{Commutativity of } \vee \\
\neg\neg\varphi &\equiv \varphi && \text{Double negation elimination (DNE)} \\
\neg(\varphi \wedge \psi) &\equiv \neg\varphi \vee \neg\psi && \text{De Morgan's law for } \wedge \\
\neg(\varphi \vee \psi) &\equiv \neg\varphi \wedge \neg\psi && \text{De Morgan's law for } \vee \\
\varphi \wedge \varphi &\equiv \varphi && \text{Idempotence for } \wedge \\
\varphi \vee \varphi &\equiv \varphi && \text{Idempotence for } \vee \\
\neg\top \equiv \bot \quad &\text{and} \quad \neg\bot \equiv \top && \text{Complements} \\
\varphi \to \psi &\equiv \neg\varphi \vee \psi && \text{Reduction of } \to
\end{aligned}
$$

Proof. The proof of all equivalences follows easily from the definitions and properties of min and max. As as example, let us prove De Morgan's law for $\wedge$. To that end, suppose that v is a valuation. We then have that

$$
\begin{aligned}
[\![\neg(\varphi \wedge \psi)]\!]_v &= 1 - \min\{[\![\varphi]\!]_v, [\![\psi]\!]_v\} \\
&= \max\{1 - [\![\varphi]\!]_v, 1 - [\![\psi]\!]_v\} \\
&= [\![\neg\varphi \vee \neg\psi]\!]_v \,.
\end{aligned}
$$

As this holds for any valuation, we obtain De Morgan's law for $\wedge$. □

These laws seem very familiar from arithmetic, but be careful: the connectives $\wedge$ and $\vee$ are treated as dual through De Morgan's laws, unlike multiplication and sum. Also, idempotence and implication have no analogue in arithmetic. Nevertheless, we can use these laws as a powerful tool to rearrange and simplify formulas. As a first test, let us prove the equivalence (5.2).

$$
\begin{aligned}
&\varphi_1 \wedge \cdots \wedge \varphi_n \wedge \neg\varphi \text{ is unsatisfiable} \\
&\text{iff } \varphi_1 \wedge \cdots \wedge \varphi_n \wedge \neg\varphi \equiv \bot \\
&\text{iff } \neg(\varphi_1 \wedge \cdots \wedge \varphi_n \wedge \neg\varphi) \equiv \neg\bot \\
&\text{iff } \neg\varphi_1 \vee \cdots \vee \neg\varphi_n \vee \varphi \equiv \top && \text{theorem 5.1} \\
&\text{iff } \neg\varphi_1 \vee \cdots \vee \neg\varphi_n \vee \varphi \text{ is a tautology}
\end{aligned}
$$

?

Can you identify the laws used in the above proof?

As you can see, we can now can use algebraic and logical reasoning interchangeably. Moreover, (5.1) and (5.2) allow us to simplify the problem of deduction, if we are able to identify classes of formulas for which we can easily decide whether they are tautologies or satisfiable.

5.3. Conjunctive Normal Forms

al-Khwārizmī: Do you remember how we found the roots of an arithmetical term?

Clara: We rearranged it so that we multiplied terms, for which we could easily find roots.

al-Khwārizmī: Very good. Now how do you think we can identify *tautologies*?

Isaac: Didn't we say that φ is a tautology if $\varphi \equiv \top$?

al-Khwārizmī: That is correct. Now can you think of an arrangement in a formula that allows you to break down the problem, like we did for finding the roots?

Clara: The point was that if we multiply any number with zero, the result will be zero, wasn't it? It seems that this does not work of conjunction because $\varphi \wedge \psi \equiv \bot$ does not necessarily hold if just $\varphi \equiv \bot$ or $\psi \equiv \bot$. For example, $p \wedge \neg p$ is contradictory but p and $\neg p$ on their own are not.

al-Khwārizmī: Excellent! We will indeed have to properly adapt the method for polynomials to logic.

Our goal is to find a normal form of formulas, for which the tautology problem becomes simple. In the case of polynomials, we use products of irreducible polynomials that make it easy to find roots by finding roots for each multiplicand. Since the Boolean semantics have precisely two truth values, we have that $\varphi \wedge \psi \equiv \top$ if and only if $\varphi \equiv \top$ and $\psi \equiv \top$. This is analogous to the product in polynomials, and what remains to find is something that corresponds to irreducible polynomials.

Definition 5.2

We say that a formula is in *conjunctive normal form* (CNF) if it can be generated by the non-terminal C in the following grammar.

$$\begin{aligned} L &::= p \mid \neg p \\ D &::= L \mid L \vee D \\ C &::= D \mid D \wedge C \end{aligned}$$

Formulas of the shape L are called *literals*.

In this formula, the disjunctions D play the role of irreducible polynomials. We will see that it is easy to decide whether they are tautologies.

Note that the grammar is just a formal way of stating that any formula in conjunctive normal form must be a conjunction of disjunctions of literals. But the grammar also allows that we do not use conjunctions or disjunctions at all, as the following example shows.

Example 5.3

Here are some formulas that are in CNF.

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

However, in a CNF, we may not have negations of complex formulas:

$$\neg(r \vee q) \text{ and } \neg\neg p \text{ are not in CNF}$$

We may also not use implication directly, as $p \to q$ is not in CNF. However, $p \to q$ is semantically equivalent to $\neg p \vee q$. Finally, the disjunction may only appear between literals:

$$p \vee (\neg q \wedge r) \text{ is not in CNF}$$

The goal is to transfer our knowledge from algebra to determine whether a

formula is a tautology. Remember that we proceeded by first turning the arithmetic term into a *normal form*, which consequently allowed us to find roots easily. We repeat this here and show that we can determine for formulas in CNF easily whether they are tautologies and then that any formula can be brought into this shape. Let us begin with the first part.

Theorem 5.4

1. A disjunction of literals $L_1 \vee \cdots \vee L_n$ is a tautology, if and only if there are i and j with $1 \leq i, j \leq n$ and a propositional variable p, such that $L_i = p$ and $L_j = \neg p$. We call (L_i, L_j) a *match*.
2. Formula in conjunctive normal form $D_1 \wedge \cdots \wedge D_m$ is a tautology if and only if every disjunct D_k has a match.

Proof. 1. We prove both directions of the statement separately. One direction is easy: If the formula $L_1 \vee \cdots \vee L_n$ has a match, then it is clearly a tautology by the law of excluded middle.

For the other direction, assume that $L_1 \vee \cdots \vee L_n$ is a tautology. Suppose, towards a contradiction, that this formula has no matching literals. We define a valuation v by

$$v(p_k) = \begin{cases} 0, & L_k = p \\ 1, & L_k = \neg p \end{cases}$$

Since all literals L_k use either distinct variables or are the same, we immediately have

$$[\![L_1 \vee \cdots \vee L_n]\!]_v = \max\{0\} = 0\,.$$

This is in contradiction to the assumption that the formula is a tautology, and therefore we must have matching literals.

2. The follows immediately from our discussion that $\varphi \wedge \psi \equiv \top$ if and only if $\varphi \equiv \top$ and $\psi \equiv \top$, and the result from item 1. □

You should now be able to distil yourself an algorithm from theorem 5.4 that decides whether a formula in CNF is a tautology in linear time [HR04].

Example 5.5

Let us go through the formulas listed in example 5.3:

1. The formula p as no matching literals
2. Neither does the formula $\neg p$.
3. The formula $\neg p \vee q \vee p$ has the match $(\neg p, p)$ and is therefore a tautology.
4. Also the formula $(\neg p \vee q \vee p) \wedge (r \vee \neg r)$ is a tautology. The first conjunct has again $(\neg p, p)$ as match, while the second conjunct has $(r, \neg r)$ as match.
5. Finally, $(\neg p \vee q \vee p) \wedge (\neg p \vee r) \wedge q$ is not a tautology because neither the second nor the third conjunct have matches.

And now comes the big step: We need to transform formulas into conjunctive normal forms.

Theorem 5.6

Every propositional formula can be transformed into a semantically equivalent (not necessarily unique) CNF.

Proof. A CNF for a formula φ can be computed by the following

1. Reduce all occurrences of implication (that do not correspond to negations) to negation and disjunction, using the last law in theorem 5.1.
2. Push all negations to literals by using De Morgan's laws and DNE from left to right. This gives, what we may call, a negation normal form that only consists of literals, conjunction and disjunction.
3. Finally, we apply the distributivity of $\vee$ over $\wedge$ from left to right to obtain a conjunctive normal form. □

Again, you can turn the proof of theorem 5.6 into an algorithm [HR04]. To do so, it is important to note that we used the laws in theorem 5.1 only **from left to right**! Were you to attempt to mix directions, then you would not reach a normal form. Also, using commutativity is not a good idea because then you may end up swapping formulas around for eternity. If you stick to these

rules, then you end up with a *strongly normalising rewriting system* [Klo92]. And this is what will give you an algorithm.

But instead of going through the tedious process of proving this formally, let us rather see how CNFs are computed on some examples.

Example 5.7

We shall compute a CNF of the formula $(\neg p \wedge q) \to p \wedge (r \to q)$.

$$
\begin{aligned}
&(\neg p \wedge q) \to p \wedge (r \to q) && \\
&\equiv (\neg p \wedge q) \to p \wedge (\neg r \vee q) && \text{Reduction of } \to \\
&\equiv \neg(\neg p \wedge q) \vee (p \wedge (\neg r \vee q)) && \text{Reduction of } \to \\
&\equiv (\neg\neg p \vee \neg q) \vee (p \wedge (\neg r \vee q)) && \text{De Morgan} \\
&\equiv (p \vee \neg q) \vee (p \wedge (\neg r \vee q)) && \text{DNE} \\
&\equiv ((p \vee \neg q) \vee p) \wedge ((p \vee \neg q) \vee (\neg r \vee q)) && \text{Distributivity} \\
&\equiv (p \vee \neg q \vee p) \wedge (p \vee \neg q \vee \neg r \vee q) && \text{Associativity}
\end{aligned}
$$

I have coloured for you the formulas that are shuffled around in the use of distributivity. This should help to understand this, probably, most difficult step. The application of associativity is not strictly necessary but simplifies the resulting formula somewhat. Again not strictly necessary, but you may even simplify further:

$$
\begin{aligned}
&\dots && \\
&\equiv (p \vee p \vee \neg q) \wedge (p \vee \neg q \vee \neg r \vee q) && \text{Commutativity} \\
&\equiv (p \vee \neg q) \wedge (p \vee \neg q \vee \neg r \vee q) && \text{Idempotence}
\end{aligned}
$$

This reasoning is correct, albeit not needed to obtain a CNF. You also need to be aware that the use of commutativity here is, as we discussed above, not necessarily something that can be directly automatised. Rather, you would need to develop another optimisation algorithm that computes a **minimal** CNF.

I think that finding a negation normal form, by reducing implication, applying De Morgan's laws and DNE should not pose a problem to you. The distributive law is somewhat more complex. Let me therefore give you another example, in which you have to apply it multiple times.

Example 5.8

Suppose we were asked to find the CNF of $\neg(p \vee \neg q) \vee (p \vee r \to s)$. This first steps to find the negation normal form are simple:

$$\begin{aligned}
&\neg(p \vee \neg q) \vee (p \vee r \to q) \\
&\equiv \neg(p \vee \neg q) \vee (\neg(p \vee r) \vee q) && \text{Reduce } \to \\
&\equiv (\neg p \wedge \neg\neg q) \vee ((\neg p \wedge \neg r) \vee q) && \text{De Morgan} \\
&\equiv (\neg p \wedge q) \vee ((\neg p \wedge \neg r) \vee q) && \text{DNE}
\end{aligned}$$

The next step is to apply the distributive law on the **innermost** combination of conjunction and disjunction:

$$\begin{aligned}
&(\neg p \wedge q) \vee ((\neg p \wedge \neg r) \vee q) \\
&\equiv (\neg p \wedge q) \vee ((\neg p \vee q) \wedge (\neg r \vee q)) && \text{Distributivity}
\end{aligned}$$

So far so good. Now things get a bit more tricky, as you have to do careful bookkeeping. As the formula has a conjunction between two disjunctions, we will have to apply the distributive law *twice*! In the first step, I will help again with some colour. You should be able to match the formulas in the second step.

$$\begin{aligned}
&(\neg p \wedge q) \vee ((\neg p \vee q) \wedge (\neg r \vee q)) \\
&\equiv ((\neg p \wedge q) \vee (\neg p \vee q)) \wedge ((\neg p \wedge q) \vee (\neg r \vee q)) && \text{Distr.} \\
&\equiv (\neg p \vee \neg p \vee q) \wedge (q \vee \neg p \vee q) \wedge (\neg p \vee \neg r \vee q) \wedge (q \vee \neg r \vee q) && \text{Distr.} \\
&\equiv (\neg p \vee q) \wedge (q \vee \neg p) \wedge (\neg p \vee \neg r \vee q) \wedge (q \vee \neg r) && \text{Comm. \& Idem.}
\end{aligned}$$

The last step is again not necessary because the second to last line is already a CNF, but simplifications are always welcome!

Clara: Using conjunctive normal forms to prove entailment seems very elegant and powerful. I have a feeling that there is a catch. The formula that we obtained in example 5.7 seems rather big.

al-Khwārizmī: Well observed, my friend! The problem lies in the application of distributivity to find a CNF of a formula. This law duplicates subformulas and can cause the computed CNF to be exponentially larger than the original formula. Thus, even though you may be able to decide in

linear time whether a formula in CNF is a tautology, the transformation causes the running time to be still potentially exponential for general formulas. And we have not even touched upon the SAT-problem, which is NP-complete!

Isaac: That is all quite discouraging! But considering that it does not take me years to make decisions, there must surely be a way to improve on that…

al-Khwārizmī: There are numerous ways. For once, the SAT-problem can often be solved efficiently for problems that you find in reality, in contrast to the ones that we would construct to show NP-completeness. But we can also be smarter and think about the knowledge that you have and how you use it to make decisions. Before that, I need a short break! Let us go outside and have a tea in the gardens.

?

Let us call a subformula of shape D in a CNF a *disjunctive clause.* How many disjunctive clauses do CNFs for the formulas $(p_1 \wedge q_1) \vee (p_2 \wedge q_2)$ and $(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee (p_3 \wedge q_3)$, respectively, have?

5.4. Horn Clause Theories

[*Al-Khwārizmī, Clara and Isaac go outside into the gardens of the* City of Peace. *Trees line the many small houses and in the middle rises the* Palace of the Golden Gate *with its green dome. Around are numerous people sitting and holding a book in one hand, while writing with the other. The three sit down on small carpets that al-Khwārizmī brought with him and drink tea from small cups.*]

Isaac: What are these people doing?

al-Khwārizmī: They are translating texts from China, India and Greece, interpret them, critique them, add new insights.

Clara: Does that mean that they study throughout their life?

al-Khwārizmī: Of course! The key to progress is to understand, scrutinise and question existing knowledge.

Let us now talk about how you can be smarter about *representing* logical knowledge, thereby enabling Isaac to work.

Most of the knowledge that you gather can be divided into **facts**, things that you know for sure, and **inference rules**, which allow deduction of new knowledge. If we restrict ourselves to formulas that follow this idea, so-called *Horn clauses*, we obtain a fragment of propositional logic for which satisfiability can be decided in linear time. This is great for knowledge representation and inference, as you may find it in you, Isaac, or in expert systems that can be used in medicine, or other areas where large knowledge bases have to be queried. The caveat is, of course, that this fragment does not cover all of propositional logic but suffices for most computations with knowledge.

Definition 5.9

We say that a propositional formula is a *Horn clause* or *definite clause* if it is generated by the non-terminal D of the following grammar.

$$\begin{aligned} A &::= \bot \mid \top \mid p && \text{(Atoms)} \\ G &::= A \mid G \wedge G && \text{(Goals)} \\ D &::= G \to A && \text{(Horn Clauses)} \end{aligned}$$

A finite set of Horn clauses is called a *Horn theory* or a *(propositional) logic program*.

The idea is that a Horn theory represents a knowledge base, composed of inference rules with premises, called goal here, and atomic conclusions. The main restriction, compared to, say, natural deduction rules is that the conclusion and premises must be atoms. Nevertheless, you can represent positive facts by Horn clauses of the form

$$\top \to p$$

and negative facts by

$$p \wedge q \to \bot\,.$$

It is also possible to have disjunctions in goals by using the equivalence

$$p \vee q \to r \equiv (p \to r) \wedge (q \to r)\,,$$

which we can assemble as Horn theory $\{p \to r, q \to r\}$

Let us see some concrete examples and counterexamples.

Example 5.10

The following formulas are Horn clauses.

1. $p \wedge q \wedge s \to p$
2. $q \wedge r \to p$
3. $p \wedge s \to s$
4. $p \wedge q \wedge s \to \bot$
5. $\top \to s$

Note that the second to last formula is semantically equivalent to the formula

$$\neg(p \wedge q \wedge s)$$

which shows you how to represent the negation of facts. The last formula is equivalent to s, and we can thus also represent simple facts as Horn clauses. Putting any combination of these formulas together, we obtain a Horn theory. For example,

$$\{p \wedge q \wedge s \to p, q \wedge r \to p, p \wedge s \to s\}$$

and

$$\{p \wedge q \wedge s \to \bot, q \wedge r \to p, \top \to s\}$$

are both Horn theories.
We can even represent negative facts with disjunction by using

$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

and turning this into a Horn theory:

$$\{\neg p, \neg q\}\,.$$

The other way around, we have

$$\neg p \vee \neg q \equiv \neg(p \wedge q) = p \wedge q \to \bot\,,$$

which is a Horn clause.
In contrast, if we do not know something definitively, then we cannot represent this in Horn clauses. For instance,

$$p \vee q$$

cannot be represented in Horn theories.

What makes Horn theories so interesting is that we can efficiently decide whether a Horn theory is satisfiable. We will thus obtain an efficient algorithm that decides whether a Horn theory Γ with $\Gamma = \{\varphi_1, \dots, \varphi_n\}$, seen as knowledge base, entails an atom by using eq. (5.1) from our earlier discussion:

$$\Gamma \models p \text{ iff } \bigwedge \Gamma \wedge \neg p \text{ is unsatisfiable,}$$

where

$$\bigwedge \Gamma = \varphi_1 \wedge \cdots \wedge \varphi_n \,.$$

This means that we can query a Horn theory Γ with a goal p by checking whether the Horn theory $\Gamma \cup \{\neg p\}$ is unsatisfiable.

How can we check whether a Horn theory Γ is satisfiable or not? The idea is quite simple: we iterate over all clauses in Γ and use them to deduce whether an atom can be derived, while remembering the atoms that we could already derive. If we are in the end able to derive $\bot$, then the theory is not satisfiable. Otherwise, we can find this way a valuation that makes the Horn theory true. Algorithm 1 formalises this idea.

Algorithm 1: HornSat

Input: Horn theory Γ
Output: Is $\bigwedge \Gamma$ satisfiable?
begin
  let val *be an array indexed by atoms in* Γ *to* $\mathbb{B}$*, initialised to* 0 *everywhere*
  $\mathsf{val}[\top] \longleftarrow 1$
  while *there is a clause* $A_1 \wedge \cdots \wedge A_n \to A \in \Gamma$ *with* $\mathsf{val}[A] = 0$ *and* $\mathsf{val}[A_k] = 1$ *for all* $1 \leq k \leq n$ **do**
    $\mathsf{val}[A] \longleftarrow 1$
  end
  if $\mathsf{val}[\bot] = 1$ **then**
    return *No*
  else
    return *Yes*
  end
end

The following theorem assures us that algorithm 1 works correctly and is efficient.

Theorem 5.11

The algorithm HornSat terminates for any Horn theory Γ within $n+1$ steps for n being the number of distinct atoms in Γ, and it outputs Yes only if $\bigwedge \Gamma$ is satisfiable.

This algorithm is easily implemented on a computer, but for humans it is useful to use a table that shows the state of the val array.

Example 5.12

Let Γ be the Horn theory

$$\{\underbrace{p \wedge q \wedge s \to \bot}_{1}, \underbrace{q \wedge s \to p}_{2}, \underbrace{\top \to s}_{3}, \underbrace{\top \to q}_{4}\},$$

where we numbered the clauses for easy reference. This is not necessary, and if the clauses are short, then it is easier to leave out the numbering. The following table shows the steps taken by HornSat with the content of val for each atom and the clause that is selected in the loop.

Step	$\top$	p	q	s	$\bot$	Clause
0	1	0	0	0	0	
1	1	0	0	1	0	3
2	1	0	1	1	0	4
3	1	1	1	1	0	2
4	1	1	1	1	1	1

Since the algorithm stops with 1 in the column of $\bot$, we find that the theory Γ is not satisfiable. And indeed, any valuation has to assign 1 to s and q by clause 3 and 4, thus also to p by clause 2 and therefore contradicts the first clause.
Remember that we claimed that HornSat should allow us to prove entailment. Indeed, if you read $p \wedge q \wedge s \to \bot$ as $\neg(p \wedge q \wedge s)$ and put $\Delta = \{q \wedge s \to p, s, q\}$, then

$$\Gamma \text{ is unsatisfiable iff } \Delta \vDash \neg(p \wedge q \wedge s).$$

The run of the algorithm above shows therefore that $q \wedge s \to p$, s and q together entail $p \wedge q \wedge s$. An obvious fact, proven by a rather convoluted but fully automatic procedure!

Let us contrast example 5.12 to an example, in which the algorithm terminates

without marking $\bot$ as true in the end.

Example 5.13

Let $\Gamma = \{p \wedge q \wedge s \to p, \top \to s, s \to q, s \wedge q \to r\}$. The following table shows again all the steps taken by HornSat and the val array evolves.

Step	$\top$	p	q	s	r	$\bot$	Clause
0	1	0	0	0	0	0	
1	1	0	0	1	0	0	$\top \to s$
2	1	0	1	1	0	0	$s \to q$
3	1	0	1	1	1	0	$s \wedge q \to r$

After step 3, the algorithm terminates because no clause with a goal marked as true and conclusion marked as false is available. Note that we have used all clauses, except the first. This clause has p in the goal $p \wedge q \wedge s$, which is marked with 0 and makes this clause thus unusable. From the table, we can also read off a valuation v that satisfies Γ:

$$v(p) = 0 \quad v(q) = 1 \quad v(s) = 1 \quad v(r) = 1.$$

We find then immediately that

$$\min[\![\Gamma]\!]_v = \min\{1, 1, 1, 1\} = 1.$$

al-Khwārizmī: I think this is as far as we will go today. The last few sun rays are touching the walls and we haven't even eaten, yet! May I invite you to my home for dinner?

Clara: That would be lovely, a bit of rest before we continue our journey. May I ask one last question?

al-Khwārizmī: Of course, of course!

Clara: We have learned on our previous stop about formal proofs. But why do we need those, if we could let a computer prove everything for us?

al-Khwārizmī: This is a good question with many answers.

Let me give you three of them. And the first will be concerned with efficiency. The following table lists what we have discovered together.

Problem	CNF	Horn	General wff
Satisfiability (SAT)	NP	$\Theta(n)$	NP
Tautology (TAUT)	$\Theta(n)$	$\Theta(n)$	NP
Conversion from wff	$O(2^n)$	Not always possible	—

The notation $\Theta(n)$ means thereby that these problems are bounded in complexity linearly and asymptotically from below and above, while $O(2^n)$ says that there are formulas that have only an exponentially larger CNF. This table tells you that the complexity situation is quite good, whenever you are presented with a formula in conjunctive normal form or a Horn theory. Outside of these realms, the situation is pretty dire, although many clever algorithms have been developed to improve the efficiency for the SAT-problem on CNFs. In general, however, it might be a good idea to just provide a proof yourself, possibly assisted by a computer that carries out the trivial steps and helps with complex ones.

The second reason why formal proofs are still interesting is that they give an insight into **why** a formula can be derived. This is important in many applications of logic like, for instance, an artificial intelligence for which you would like to be sure that it is working correctly and that you can change when things go wrong. It is possible to automatically construct proofs for a valid entailment, see theorem 4.17, but this process is far from efficient. There are better ways to construct proofs for derivations from Horn theories, as you will see later.

al-Khwārizmī: Finally, learning about proof theory now for the fairly simple propositional logic will help you greatly with learning proof theory for more a complex logic. But I'm taking the excitement away from your journey! Let us celebrate our present achievements over our meal.

Isaac: Thank you very much for these insights!

Clara: Yes, thank you! I am looking forward to the meal and a bit of rest.

[*Clara and Isaac follow al-Khwārizmī, leaving the* City of Peace *and the palace behind. The sun has set and the clear sky shows an abundance of stars, as they have never seen before.*]

6. Introduction to First-Order Predicate Logic

[*Clara and Isaac are in front of a grey stone temple, made from of stacked segments that narrow down with every level. Each segment is decorated with ornaments and large arcs invite visitors to enter. Trees grow around and above the temple. They provide shade and coolness, protecting from the hot midday sun. Clara and Isaac sit down at the foot of one of the trees.*]

Clara: We had quite the journey so far! What brings us here?

Isaac: During our dinner with al-Khwārizmī, I realised that, even though we have learned how to reason with propositions, we do not know how to reason about objects and elements of this world. How can we state that objects have equal properties or relate in other ways?

Clara: What do you mean exactly?

Isaac: I can get up, and walk towards the temple without running into any tree, person or other obstacle. This means I am able to reason about my position relative to any number of obstacles and any size of an environment.

Clara: And you are saying that this is not possible with propositional logic?

Isaac: Look, let me give an example.

6.1. The Need for a Richer Language

[*Isaac draws the board in fig. 6.1 in the sand between him and Clara.*]

Clara: Do you think you are a star?

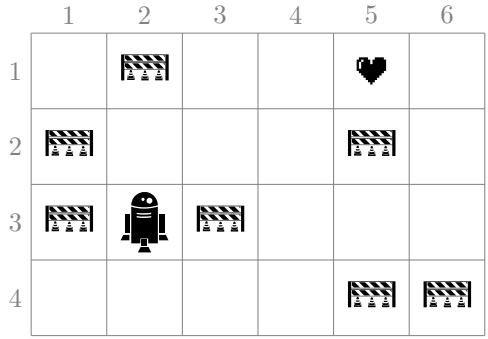


Figure 6.1.: Robot trying to find a heart

Isaac: At least you recognise that this is a robot and what it wants! What you see on this board is me, some obstacles and a goal that I am trying to reach. In other words, we are trying to find a route from the initial position to the goal.

Clara: Ok, let me attempt to solve this using propositional logic.

First, we introduce propositional variables for all the positions (x, y) in your drawing, where x ranges from 1 to 6, and y from 1 to 4.

r-x-y	Robot in position (x, y)
o-x-y	Obstacle in position (x, y)
g-x-y	Goal in position (x, y)

We can then specify, for example, that you are in position $(2, 3)$ by stipulating that the variable r-2-3 is true. Or something more complex: The requirement that you should never collide with an obstacle can be expressed by the following formula.

$$(\text{r-1-1} \rightarrow \neg\text{o-1-1}) \wedge (\text{r-1-2} \rightarrow \neg\text{o-1-2}) \wedge (\text{r-1-3} \rightarrow \neg\text{o-1-3}) \wedge \cdots \quad (6.1)$$

Oh, I see the problem! This specification consists of 24 conjuncts and only works for this specific board size. Is there a better way to do this?

Isaac: I don't know. My CPU is running hot! This is not the season for me to be in India.

Clara: Shall we go inside? The temple should allow you to cool down.

Isaac: Lets go!

[*Clara and Isaac enter the temple through the main arc in the middle. Music and discussions immediately enshroud and drag them in.*]

Clara: These drums, lutes and pipes sound very familiar.

Isaac: The Muslim culture has expanded far to the east!

Raghunātha: I'm not here for *Jalpa*! This is a serious issue and there is no place for wrangling! Let us continue this later …

Who are you two strange figures??

Clara: My friend Isaac and I came inside to cool down a bit from thinking about a difficult logic problem.

Raghunātha: Logic? I hope you don't mean the kind of logic employed in the debate over there: arguing purely for winning and not for the good of our society, and entirely neglecting facts and arguments?

Isaac: You seem to be upset …

Raghunātha: Please excuse my temper!

Clara: We are in fact here because we do not understand some aspect of Isaac's inner machinery.

Raghunātha: Is this a …?

Clara: Yes.

Raghunātha: Marvellous! Tell me, what is your problem?

[*They explain to Raghunātha the problem that they had just discussed outside the temple.*]

Raghunātha: I see. Let us sit down in the quiet corner over there and enjoy a friendly discussion over some refreshments.

What you need is a language for talking about the properties of objects and not just propositions. This is the language of *first-order predicate logic* (FOL). Predicates allow us to express properties of objects. For example, instead of having a propositional variable for every position in fig. 6.1 and every property of that position, we have just *one predicate per property* that classifies the positions. Concretely, the variables r-x-y, o-x-y and g-x-y can be replaced in FOL with predicates R, O and G on positions. For the moment, we will hide

the x- and y-coordinates and just write p for a position (x, y). The predicates R, O and G on positions are then used and read as follows.

$R(p)$	Robot in position p
$O(p)$	Obstacle in position p
$G(p)$	Goal in position p

You could now naively write the formula (6.1) in the same way, but then we would have not gained anything. Instead, the power of predicate logic comes from the fact that there are logical connectives that allow you to talk about *objects*, which are in this case positions on the board. In particular, eq. (6.1) states that Isaac should not collide with an obstacle on *all positions*:

$$\begin{aligned}&\text{For all positions } p\text{, if the robot is in } p,\\&\text{then there should be no obstacle in } p.\end{aligned} \tag{6.2}$$

Clearly, we could use propositional connectives and the predicates R and O to represent if the robot is in p, then there should be no obstacle in p by the formula $R(p) \to \neg O(p)$. What remains is to find a formal representation for the phrase "For all positions p". First-order predicate logic uses a logical connective, called the *universal quantifier*, that provides exactly this representation. This quantifier is written as $\forall$, a turned around A, and comes with an *object variable* behind that it binds. We will also use a dot to separate the quantifier with the bound variables from the remaining formula. The sentence (6.2) becomes then this symbolic expression

$$\forall p.\, R(p) \to \neg O(p)\,, \tag{6.3}$$

which we read as "for all p, if $R(p)$ then not $O(p)$".

Clara: Can you explain a bit more why we use the dot?

Raghunātha: It indicates the *scope* of the universal quantifier: the quantifier and the object variable p refer to everything on the right of the dot. This helps to reduce the use of parentheses, as we would have to write $\forall p.\, (R(p) \to \neg O(p))$ otherwise. You also see that people write $(\forall p)(R(p) \to \neg O(p))$, but the number of parentheses gets out of control with this approach. That's why we use the dot with the convention that everything on its right is in its scope.

Clara: What is actually an *object* variable?

Raghunātha: Excellent questions! We will come to these soon. For the time being, think of object variables as placeholders for objects of unknown nature. These can be positions, numbers, letters in an alphabet or any other mathematically presentable object.

Clara: Alright… Isaac and I began with the problem of finding a route on the board fig. 6.1. Can we use FOL for this as well?

Raghunātha: Yes, of course! In fact, this is a problem that shows that FOL can express things that are *impossible* to express with propositional logic.

Suppose, we want to specify what a route from the starting position to the goal is in fig. 6.1. If you attempt this in propositional logic, then you will find that there are infinitely many possible routes, even for this finite board, because the robot can run in loops. Therefore, a formula that describes how a route looks like would have to be infinite in propositional logic, which is clearly non-sense! Now, think about how you would describe a route: it is path on the board leads from the starting point to the goal, with the crucial property that

There is no obstacle anywhere on a path. (6.4)

Let us attempt to formalise (6.4) using FOL. First, we have to choose a way to talk about paths. There are many possibilities to do that, but let us see a path as a list of positions on the board that we can access by indices, just like arrays in programming. Thus, we find at the index 0 the initial position of a path, at index 1 the next position etc. Formally, we use three more predicates:

$P(r)$	r is a path
$N(k)$	k is an index (natural number)
$S(r, k, p)$	Position p occurs at index k in r

Using these predicates, we can express that there is no obstacle on a path, as in (6.4), by the following formula.

$$\forall r.\, P(r) \to \forall k.\, \forall p.\, N(k) \land S(r, k, p) \to \neg O(p) \tag{6.5}$$

In words, this formula is read as

> "For all paths r and for all k and p, if k is an index and p is the position in r at index k, then there is no obstacle at position p."

Raghunātha: Easy, right?

Clara: Looks rather complicated! How does this work with the dots again?

The easiest way is to imagine that you add parentheses around everything on the right of the dot, until you get to some parentheses. If you do this, starting from the outside, then you get the formula

$$\forall r.\left(P(r) \to \Big(\forall k.\, \forall p.\, (N(k) \wedge S(r,k,p) \to \neg O(p))\Big)\right).$$

Isaac: So far so good. But now I would like to interpret, at least intuitively, this formula. Suppose I assume that $S(r, 0, (2, 3))$ is true, which is the initial position on the board in fig. 6.1 and that $S(r, 1, (2, 2))$ *and* $S(r, 1, (3, 3))$ are true. Does this mean that I am in some kind of undecided quantum state, occupying both positions $(2, 2)$ and $(3, 3)$ after the first step? Even worse, in one case I hit my head against an obstacle!

Raghunātha: This is a very good observation and is a flaw in our naive formalisation.

Fortunately, first-order predicate logic has a way out. Notice that p is *uniquely determined* by r and k in $S(r, k, p)$, and thus S is a map. In FOL, we allow ourselves to write such maps in the form of so-called *function symbols*. For this particular example, we can use one symbol s with the following meaning.

$$s(r, k) \text{ returns the position at index } k \text{ in path r}$$

We can then simplify eq. (6.5) to

$$\forall r.\, P(r) \to \forall k.\, N(k) \to \neg O(s(r, k)) \tag{6.6}$$

and thereby avoid that a path can be ambiguous. Note, however, that such ambiguities might be wanted and that this is the crucial difference between using predicates and functions:

- use a predicate (also called relation) S, if there can be multiple positions (or none) at index k on the path r, and
- use a map s, if there is exactly one position for every index k on all paths r.

Isaac: Oh, isn't there then a problem with the formula (6.6)?

?

Can you see what problem Isaac spotted? And would you have an idea how to solve it? Hint: The predicate N only ensures that k is a natural number, but does not relate k to the length of the path r.

Isaac: Ok, so that problem is solvable. We talked initially about routes, which supposedly lead from the starting point to the goal, but now we diverted to paths. How can we put this together?

Raghunātha: Which question is it that we are actually trying answer?

Clara: We began with the routing problem, which asks for a route from the initial position of the robot to the heart.

Raghunātha: Yes indeed. It is important to realise that this problem asks for *the existence* of a path. So far, we have only dealt with *universal* statements, statements that require something to hold *for all* instances of an object variable. But existence is qualitatively different.

To be precise, let us formulate the routing problem first in natural language:

There is a path leading from the initial to the goal position. (6.7)

The phrase "there is" indicates that we are looking for the existence of something. First-order predicate logic allows us to express such statements with the so-called *existential quantifier*, written as "$\exists$" (a turned around "E"). To formulate the sentence (6.7) using this quantifier, let us use two more function symbols:

$\underline{0}$	represents the number 0
$e(r)$	last index in path r

With these function symbols, we can present the sentence (6.7) as the following formula.

$$\exists r.\, P(r) \wedge R(s(r, \underline{0})) \wedge G(s(r, e(r))) \tag{6.8}$$

This formula can be read like this:

$$\underbrace{\exists r.\, P(r)}_{\text{there is a path}} \quad \wedge \quad \underbrace{R(s(r, \underline{0}))}_{\text{starting at the robot position}} \quad \wedge \quad \underbrace{G(s(r, e(r)))}_{\text{ending at the goal}}$$

With this at hand, you should be able to easily match the formula with the informal sentence that describes the routing problem.

Clara: Whew! This seems all quite reasonable, but what is first-order predicate logic then actually?

Raghunātha: I'm glad you should ask. Let us try to distil a formal definition of first-order formulas.

6.2. The Language of First-Order Logic

In propositional logic, we only deduced logical facts and we were thus content with formulas that were made up of propositional variables and logical connectives. In first-order predicate logic, or FOL, we additionally wish to reason about the relations of objects. Thus, FOL will have two building blocks:

- terms, built from object variables and function symbols, and
- formulas, built from predicate symbols and logical connectives.

Terms will thereby be the syntactical representation of the objects that we aim to reason about. As you can see, both terms and formulas mention *symbols*. Looking back at the function and predicate symbols that we used in the formalisation of the routing problem, you will see that every symbol can be applied to a certain number of arguments. For instance, s was a symbol that can be applied to two arguments, while O only took one argument. We refer to the number of arguments of a symbol as its *arity*. Formally, the symbols that can appear in FOL formulas come from so-called signatures:

Definition 6.1

A first-order **signature** $\mathcal{L}$ is a triple $(\mathcal{F}, \mathcal{R}, \mathrm{ar})$, where $\mathcal{F}$ and $\mathcal{R}$ are disjoint sets ($\mathcal{F} \cap \mathcal{R} = \emptyset$) and ar is a map $\mathcal{F} \cup \mathcal{R} \to \mathbb{N}$. The elements of $\mathcal{F}$ are called **function symbols** and those of $\mathcal{R}$ **predicate symbols** or *relation symbols*. The map ar assigns to each symbols its **arity** , which is the number of argument the symbol expects. We write $\mathcal{F}_n = \{f \in \mathcal{F} \mid \mathrm{ar}(f) = n\}$ and $\mathcal{R}_n = \{P \in \mathcal{R} \mid \mathrm{ar}(P) = n\}$. The elements of $\mathcal{F}_0$ are called *constants*.

Note that $\mathcal{F}$ (and $\mathcal{R}$) is partitioned into the sets $\mathcal{F}_n$, that is, $\mathcal{F} = \bigcup_{n \in \mathbb{N}} \mathcal{F}_n$ and $\mathcal{F}_n \cap \mathcal{F}_m = \emptyset$ for $m \neq n$. This is because ar is a map and thus assigns to every symbol a unique arity.

Let us illustrate this definition on the routing problem.

Example 6.2

The signature $\mathcal{L} = (\mathcal{F}, \mathcal{R}, \mathrm{ar})$ of all symbols that we used in section 6.1

is given as follows.

$$\mathcal{F} = \{s, \underline{0}, e\} \qquad \mathcal{R} = \{R, G, O, P, N, S\}$$
$$\mathrm{ar}(\underline{0}) = 0 \qquad \mathrm{ar}(e) = 1 \qquad \mathrm{ar}(s) = 2$$
$$\mathrm{ar}(R) = \mathrm{ar}(G) = \mathrm{ar}(O) = \mathrm{ar}(P) = \mathrm{ar}(N) = 1$$
$$\mathrm{ar}(S) = 3$$

Thus, this signature has 3 function, 6 predicate symbols, and $\underline{0}$ as the only constant. The partitions of the signature are given by

$$\mathcal{F}_0 = \{\underline{0}\} \quad \mathcal{F}_1 = \{e\} \quad \mathcal{F}_2 = \{s\} \quad \mathcal{F}_3 = \emptyset$$
$$\mathcal{R}_0 = \emptyset \quad \mathcal{R}_1 = \{R, G, O, P, N\} \quad \mathcal{R}_2 = \emptyset \quad \mathcal{R}_3 = \{S\}$$

and for $k > 3$ by $\mathcal{F}_k = \mathcal{R}_k = \emptyset$.

Clara: The signature from example 6.2 has only a finite number of symbols. Is this missing in definition 6.1?

Raghunātha: No, you can have arbitrarily large signatures. For instance, the set $\{\underline{n} \mid n \in \mathbb{N}\}$ could be a perfectly valid set of function symbols. The usefulness of such a large signature is, however, another question. It is often better to represent natural numbers in the language of FOL, but for that we need to first talk about terms.

Definition 6.3

Let $\mathcal{L}$ be a signature $(\mathcal{F}, \mathcal{R}, \mathrm{ar})$ and $\boxed{\mathrm{Var}}$ a (countable) set of object variables. The set $\boxed{\mathrm{Term}}$ or $\boxed{\mathrm{Term}(\mathcal{L})}$ of **terms** or *$\mathcal{L}$-terms*, is the set closed under the following three rules

$$\frac{x \in \mathrm{Var}}{x \in \mathrm{Term}} \qquad \frac{c \in \mathcal{F}_0}{c \in \mathrm{Term}} \qquad \frac{f \in \mathcal{F}_n \quad t_1 \in \mathrm{Term} \quad \cdots \quad t_n \in \mathrm{Term}}{f(t_1, \ldots t_n) \in \mathrm{Term}}$$

and that fulfils the following *iteration property*: for any set X together with maps $I_{\mathrm{Var}} \colon \mathrm{Var} \to X$, $I_0 \colon \mathcal{F}_0 \to X$ and $I_f \colon X^{\mathrm{ar}(f)} \to X$ for all $f \in \mathcal{F}$, there is a unique map $I \colon \mathrm{Term} \to X$ with

$$I(x) = I_{\mathrm{Var}}(x)$$
$$I(c) = I_0(c)$$
$$I(f(t_1, \ldots, t_n)) = I_f(I(t_1), \ldots, I(t_n)) \,.$$

Clara: Oh my ...

Raghunātha: Don't worry, I will slowly walk you through this definition.

First off, you could think of terms as given by a context free grammar:

$$t ::= x \mid c \mid f(t, \dots, t)$$

The problem with this grammar is that it does not take the arity of the symbol f into account. For instance, if you take the symbol e from example 6.2, then this grammar would allow you to write $e(\underline{0}, e(\underline{0}))$. This clearly does not make a lot of sense, as e is supposed to be a function symbol with one argument and not one or two! In definition 6.3, we used instead the three rules to indicate exactly the arity of the function symbols. For instance, we can say $\underline{0} \in \text{Term}$ by the second rule and therefore $e(\underline{0}) \in \text{Term}$ by the third rule. However, $e(\underline{0}, e(\underline{0})) \in \text{Term}$ does not follow from those rules because $e \notin \mathcal{F}_2$. Therefore, $e(\underline{0}, e(\underline{0})) \notin \text{Term}$

Now, this last statement, that the construction is *not* a term, would require some proof. Such a proof is possible by using the iteration principle, as we will see below. But the iteration principle gives us even more: it allows us to define maps on terms, just as you saw for propositional formulas. For instance, we can define maps that count variables or function symbols in terms, or manipulate terms. This will be extremely important once we talk about proof theory for FOL.

For the time being, let us define a map that finds all variables in a term. Recall that $\mathcal{P}(\text{Var})$ is the powerset of Var, given by $\mathcal{P}(\text{Var}) = \{U \mid U \subseteq \text{Var}\}$.

Lemma 6.4

Let $\mathcal{L}$ be a signature. There is a map $\boxed{\text{var}\colon \text{Term}(\mathcal{L}) \to \mathcal{P}(\text{Var})}$ with

$$\text{var}(x) = \{x\} \qquad \text{var}(c) = \emptyset \qquad \text{var}(f(t_1, \dots, t_n)) = \bigcup_{k=1}^{n} \text{var}(t_k)$$

that computes the variables that appear in $\mathcal{L}$-terms.

Proof. Suppose that the function symbols in $\mathcal{L}$ are $\mathcal{F}$. To use the iteration on terms, we define maps $\text{var}_{\text{Var}}\colon \text{Var} \to \mathcal{P}(\text{Var})$, $\text{var}_0\colon \mathcal{F}_0 \to \mathcal{P}(\text{Var})$ and

$\mathrm{var}_f \colon \mathcal{P}(\mathrm{Var})^{\mathrm{ar}(f)} \to \mathcal{P}(\mathrm{Var})$ for all $f \in \mathcal{F}$ with $\mathrm{ar}(f) = n$, by

$$\mathrm{var}_{\mathrm{Var}}(x) = \{x\} \qquad \mathrm{var}_0(c) = \emptyset \qquad \mathrm{var}_f(U_1, \dots, U_n) = \bigcup_{k=1}^{n} U_k$$

This gives us, by the iteration principle, the map var with the properties that we were looking for. □

Example 6.5

With the signature $\mathcal{L}$ from example 6.2 and $r, s \in \mathrm{Var}$, we have that $s(r, \underline{0})$ and $s(r, e(s))$ are $\mathcal{L}$-terms. The map lemma 6.4 allows us to compute the variables in those terms:

$$\begin{aligned} \mathrm{var}(s(r, \underline{0})) &= \mathrm{var}(r) \cup \mathrm{var}(\underline{0}) \\ &= \{r\} \cup \emptyset \\ &= \{r\} \end{aligned} \qquad \begin{aligned} \mathrm{var}(s(r, e(s))) &= \mathrm{var}(r) \cup \mathrm{var}(e(s)) \\ &= \{r\} \cup \mathrm{var}(s) \\ &= \{r\} \cup \{s\} \\ &= \{r, s\} \end{aligned}$$

Note that there are also other terms that may not have any sensible meaning, given the intuition we assigned to the symbols. For instance, $e(s(r, \underline{0}))$ is a valid term, but what is "the last index in the position at the beginning of a path"?

The way we defined the map var in lemma 6.4 is akin to recursive definitions that you may find in functional programming. We will usually just define maps on terms through equations that can *in principle* be obtained by defining the map by appealing to the formal iteration property, but without explicitly using this principle. For instance, we can define the height of terms as a map $h \colon \mathrm{Term} \to \mathbb{N}$ by three equations:

$$\begin{aligned} h(x) &= 0 \\ h(c) &= 0 \\ h(f(t_1, \dots, t_n)) &= 1 + \max\{h(t_k) \mid 1 \leq k \leq n\} \end{aligned}$$

Clearly, h could be obtained by iteration of appropriate maps but these three equations are easier to read and convey directly the behaviour of h.

From the iteration principle, we can also obtain an induction principle.

Theorem 6.6: Term Induction

Let $P \subseteq \mathrm{Term}(\mathcal{L})$ be a property of $\mathcal{L}$-terms. Suppose that $\mathrm{Var} \subseteq P$, $\mathcal{F}_0 \subseteq P$, and for all $f \in \mathcal{F}$ and $t_1, \dots, t_{\mathrm{ar}(f)} \in P$ we have $f\big(t_1, \dots, t_{\mathrm{ar}(f)}\big) \in P$. Then $\mathrm{Term}(\mathcal{L}) \subseteq P$.

Proof. We put $I_{\mathrm{Var}}(x) = x$, $I_0(c) = c$ and $I_f(t_1, \dots, t_{\mathrm{ar}(f)}) = f(t_1, \dots, t_{\mathrm{ar}(f)})$. By the iteration principle, we get a map $I\colon \mathrm{Term}(\mathcal{L}) \to P$ with $I(t) = t$ for all terms t, and by uniqueness I must be an inclusion map. Therefore, $\mathrm{Term}(\mathcal{L}) \subseteq P$. □

Let us come back to example 6.5. We saw at the end that there are terms that we may not care about or that we consider as "non-sense". This shows that terms should really be only thought of as providing a way of representing objects, or rather shapes of objects, that we may want to reason about. And for that, we will need to talk about formulas in first-order logic. There is a slight complication with variables in FOL, but you will have to wait until the next chapter of your journey to see this issue come up. For the moment, we make only a first attempt.

Definition 6.7: A first attempt to define FOL formulas

Let $\mathcal{L}$ be a signature with predicate symbols $\mathcal{R}$. The set Form or $\mathrm{Form}(\mathcal{L})$ of *(first-order) formulas* or *$\mathcal{L}$-formulas* is closed under the following rules

$$\frac{P \in \mathcal{R}_n \quad t_1 \in \mathrm{Term} \quad \cdots \quad t_n \in \mathrm{Term}}{P(t_1, \dots t_n) \in \mathrm{Form}} \qquad \frac{\varphi \in \mathrm{Form}}{(\varphi) \in \mathrm{Form}}$$

$$\frac{}{\bot \in \mathrm{Form}} \qquad \frac{\varphi \in \mathrm{Form} \quad \psi \in \mathrm{Form} \quad \square \in \{\wedge, \vee, \to\}}{\varphi \,\square\, \psi \in \mathrm{Form}}$$

$$\frac{x \in \mathrm{Var} \quad \varphi \in \mathrm{Form}}{\forall x.\, \varphi \in \mathrm{Form}} \qquad \frac{x \in \mathrm{Var} \quad \varphi \in \mathrm{Form}}{\exists x.\, \varphi \in \mathrm{Form}}$$

and fulfils an iteration property, similar to that of terms. The binding precedences extend that of propositional logic:

- $\wedge$ and $\vee$ bind stronger than $\to$, and
- $\forall x.$ and $\exists x.$ extend to the right as far as possible.

Formula	With parentheses
$\forall x.\, R(x) \to O(x)$	$\forall x.\, (R(x) \to O(x))$
$\forall x.\, \exists y.\, R(x) \to O(y)$	$\forall x.\, (\exists y.\, (R(x) \to O(y)))$
$\forall x.\, R(x) \to \exists y.\, O(y)$	$\forall x.\, (R(x) \to (\exists y.\, O(y)))$

Table 6.1.: Leaving out parentheses in formulas with quantifiers

We will not worry too much about the iteration principle, but you will see how it works in a moment. The iteration principle implies also again an induction principle for formulas. However, we will not need this principle at this stage.

Let us see some examples of formulas and applications of the precedences.

Example 6.8

All the formulas from eqs. (6.3), (6.5), (6.6) and (6.8) are all formulas over the signature from example 6.2.

As for the precedences, the table 2.2 from propositional logic still applies, but we additionally have the rule for quantifiers, which is illustrated in table 6.1. The rule that quantifiers extend as far as possible to the right implies that we have to use parentheses if a quantifier should stay under another logical connective. For instance, the formulas

$$\forall x.\, R(x) \to O(x) \qquad \text{and} \qquad (\forall x.\, R(x)) \to O(x) \tag{6.9}$$

are different and also express very different properties.

This becomes clear when we distinguish between *bound* and *free* variables.

Definition 6.9

Let $\mathcal{L}$ be a signature. We define maps on $\mathcal{L}$-formulas

$$\mathrm{fv} \colon \mathrm{Form} \to \mathcal{P}(\mathrm{Var}) \qquad \text{and} \qquad \mathrm{bv} \colon \mathrm{Form} \to \mathcal{P}(\mathrm{Var})$$

by

$$\mathrm{fv}(P(t_1,\dots,t_n)) = \bigcup_{k=1}^{n} \mathrm{var}(t_k) \qquad \mathrm{bv}(P(t_1,\dots,t_n)) = \emptyset$$

$$\begin{aligned} \mathrm{fv}(\bot) &= \emptyset & \mathrm{bv}(\bot) &= \emptyset \\ \mathrm{fv}(\varphi \,\square\, \psi) &= \mathrm{fv}(\varphi) \cup \mathrm{fv}(\psi) & \mathrm{bv}(\varphi \,\square\, \psi) &= \mathrm{bv}(\varphi) \cup \mathrm{bv}(\psi) \\ \mathrm{fv}(\forall x.\, \varphi) &= \mathrm{fv}(\varphi) \setminus \{x\} & \mathrm{bv}(\forall x.\, \varphi) &= \mathrm{bv}(\varphi) \cup \{x\} \\ \mathrm{fv}(\exists x.\, \varphi) &= \mathrm{fv}(\varphi) \setminus \{x\} & \mathrm{bv}(\exists x.\, \varphi) &= \mathrm{bv}(\varphi) \cup \{x\} \end{aligned}$$

where $\setminus$ denotes set difference and $\square \in \{\wedge, \vee, \to\}$. We say that a variable x is *free* (resp. *bound*) in φ, if $x \in \mathrm{fv}(\varphi)$ (resp. $x \in \mathrm{bv}(\varphi)$).

Example 6.10

Suppose we have a signature $\mathcal{L}$ with unary predicate symbols (one argument) P and Q, and that $\varphi = (\forall x.\, P(x) \wedge Q(y)) \to P(x) \vee Q(y)$. The variable y is clearly nowhere bound, while x is bound *and* free:

$$\underbrace{(\forall x.\, P(x) \wedge Q(x))}_{x \text{ bound}} \to \underbrace{P(x) \vee Q(y)}_{x,y \text{ free}}$$

Formally, we have

$$\begin{aligned} \mathrm{fv}(\varphi) &= \mathrm{fv}(\forall x.\, P(x) \wedge Q(x)) \cup \mathrm{fv}(P(x) \vee Q(y)) \\ &= (\mathrm{fv}(P(x) \wedge Q(x)) \setminus \{x\}) \cup \mathrm{fv}(P(x)) \cup \mathrm{fv}(Q(y)) \\ &= (\mathrm{fv}(P(x)) \cup \mathrm{fv}(Q(x)) \setminus \{x\}) \cup \{x\} \cup \{y\} \\ &= (\{x\} \setminus \{x\}) \cup \{x, y\} \\ &= \emptyset \cup \{x, y\} \\ &= \{x, y\} \end{aligned}$$

and

$$\begin{aligned} \mathrm{bv}(\varphi) &= \mathrm{bv}(\forall x.\, P(x) \wedge Q(x)) \cup \mathrm{bv}(P(x) \vee Q(y)) \\ &= (\mathrm{bv}(P(x) \wedge Q(x)) \cup \{x\}) \cup \mathrm{bv}(P(x)) \cup \mathrm{bv}(Q(y)) \\ &= (\mathrm{bv}(P(x)) \cup \mathrm{bv}(Q(x)) \cup \{x\}) \cup \emptyset \cup \emptyset \\ &= (\emptyset \cup \{x\}) \cup \emptyset \\ &= \{x\}\,. \end{aligned}$$

Example 6.10 shows how bound and free variables for a formula can be computed, and that variables can occur both bound and free. This is a bit misleading, however, because *at every individual occurrence* a variable is either bound or free, but never both. If we apply definition 6.9 to the formulas in eq. (6.9), then we find that the formula on the left has no free variables and x as bound variable, while the formula on the right has x both as free and bound variable. This should give a clear indication that the two formulas are different.

Clara: But can the formulas still not mean the same thing?

Raghunātha: Keep in mind that we are talking about syntax for the time being, which has no intrinsic meaning! That being said, you can read the formula $\forall x.\, R(x) \to O(x)$ as "$R(x)$ implies $O(x)$ for all x". Compare this to the reading of the formula $(\forall x.\, R(x)) \to O(x)$ as "if $R(y)$ holds for all y, then $O(x)$ holds for whatever x is".

Clara: But now you changed bound x in the second formula to a y. Of course they read differently then!

Raghunātha: This is indeed a delicate part of first-order logic that we will not be able to fully resolve at this stage. But think of it this way: the formula $\forall x.\, R(x)$ and $\forall y.\, R(y)$ should express the same thing, namely that R holds for all objects. Therefore, I renamed x to y. This resolves the naming conflict and we can clearly distinguish the two formulas. But this also shows that definition 6.7 is missing an important aspect of variable binding. The night is setting in. I'm afraid that we have to end our discussion for today and you will have to wait for the resolution of this problem.

Clara: Thank very much! This is a lot to take in.

Raghunātha: Indeed, but most of what we have discussed is quite straightforward: just remember that first-order predicate logic allows us to reason about objects, where objects are described syntactically as terms, and we use predicate symbols to describe properties of objects. Finally, quantifiers are used to describe universal and existential properties. You can find an overview over the connectives in table 6.2.

Isaac: I think that I understand now roughly how the routing problem can be formally described, but I hope that we will learn more about how it can actually be solved!

Raghunātha: For that, my friend, you will have to continue your journey. I will now go home and take a bit of rest.

[*Clara and Isaac leave the temple through one of the side arcs. Some children have played with their drawing under the tree. The robot has gotten a heart and left the board, leaving all constraints behind but those scribbled on the side of the board. As the wind has started to blow gently, these became unreadable but for a fragment reading: "...laws of robotics". Birds came out to use the cooler night to find food and comrades. They are making loud chirping noises, while Clara and Isaac leave the temple area for the next chapter of their journey.*]

Basic connectives

Connective	Name	Pronunciation
$\bot$	Absurdity/Falsity	
$\wedge$	Conjunction	φ and ψ
$\vee$	Disjunction	φ or ψ
$\to$	Implication	φ implies ψ
$\forall x.$	Universal quantifier	for all x, φ holds
$\exists x.$	Existential quantifier	for some x, φ holds

Derived connectives

Connective	Name	Definition
$\neg$	Negation	$\neg\varphi = \varphi \to \bot$
$\top$	Truth	$\top = \neg\bot$
$\leftrightarrow$	Bi-implication	$\varphi \leftrightarrow \psi = (\varphi \to \psi) \wedge (\psi \to \varphi)$
$\exists!x.$	Uniqueness quantifier (introduced in chapter 9)	$\exists!x.\, \varphi = \exists x.\, \varphi \wedge \text{unique}_x(x, \varphi)$

Table 6.2.: Logical connectives of first-order logic

7. Proof Theory of First-Order Predicate Logic

7.1. Substitution in First-Order Logic

In these notes, we will discuss an important operation of first-order logic: the substitution of a term for a variable. This operation will replace any occurrence of a variable in a formula by the given term. For instance, we will be able to substitute the term $f(m, x)$ for the variable y in the formula $P(y) \wedge Q(r, y)$ to obtain

$$P(f(m, x)) \wedge Q(r, f(m, x)).$$

However, the substitution operation is surprisingly complex, as we need to deal with variables that are bound by quantifiers. The aim of these notes is to give a rigorous presentation of variables and binding that allows us to safely carry out substitutions.

7.1.1. The Difficulty of Names and Variables

Let us first discuss two questions that arise in FOL:

1. Are the formulas $\forall x.\, P(x)$ and $\forall y.\, P(y)$ expressing the same?
2. What is the scope of variables?

Formula Equality and Renaming The first question can be answered by reading the formula without explicitly naming variables:

> "P holds for all objects".

Clearly, this sentence corresponds to both formulas $\forall x.\, P(x)$ and $\forall y.\, P(y)$, and we should consider both formulas to be the same, already *syntactically*! This gives us a first rule that we will have to adhere to for FOL:

> Two formulas are considered to be the same, if we can *bijectively rename* the variables of one formula to obtain the other.

For instance, consider the formulas φ and ψ given by $\varphi = \forall x.\, \forall y.\, Q(x, y)$ and $\psi = \forall w.\, \forall z.\, Q(w, z)$. Then, a bijective renaming would be to rename x to w and y to z, which allows us to transform φ into ψ. However, renaming x to r and y to r is not bijective. Thus, we do not consider the formula $\forall r.\, \forall r.\, Q(r, r)$ to be same as φ. Note, that in the latter formula, the r refers to the inner-most quantifier and the outer quantifier has no effect!

Scoping The second question concerns the scope of variables and it requires us to determine which objects a variable refers to. For instance, the variable x in the sub-formula $P(x)$ of $\forall x.\, P(x)$ refers to what the quantifier ranges over. We say that x is *in the scope* of $\forall x$ in this formula. However, the variable y in $\forall x.\, Q(x, y)$ is in the scope of no quantifier and is thus a global reference. If we naively substituted x for y in this formula, then we would obtain $\forall x.\, Q(x, x)$. Here, the scope of the second argument of Q, and with it the meaning of the formula, has suddenly changed. This leads to a second rule:

> Substituting a term in a formula should not change variable scoping.

7.1.2. De Bruijn Trees

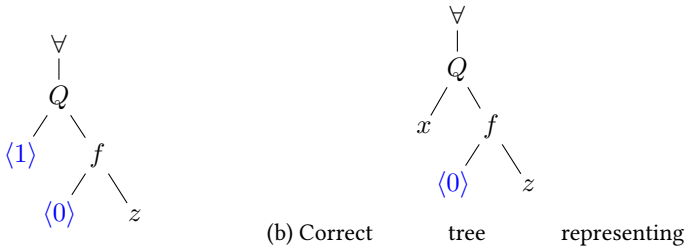
There are several ways to deal with variables, binding and substitution. An intuitively understandable way is to represent terms and formulas as *de Bruijn trees*. The idea is that bound variables are represented by a number that points to the quantifier that binds this variables. All free variables keep their name.

Figure 7.1 shows the (de Bruijn-) tree representation of $\forall x.\, \forall y.\, Q(x, f(y, z))$, where the left figure shows the actual tree and the right figure indicates to which quantifier the numerical name refers. We see that the bound variables are numbered starting from the *inner-most* quantifier. Also note that the variable z is free and thus keeps its name in the tree representation.

(a) The plain tree (b) Tree with indication of the variable references

Figure 7.1.: de Bruijn-Tree Representations of $\forall x.\, \forall y.\, Q(x, f(y, z))$

Since the nesting depth of quantifiers is important, we also note that tree representations are *not closed under subtrees*! For instance, the tree in fig. 7.2a is not a valid tree representation because $\langle 1 \rangle$ is a dangling reference. Instead,



(a) Invalid de Bruijn-Tree (b) Correct tree representing $\forall y.\, Q(x, f(y, z))$

Figure 7.2.: Attempts of finding trees that represent the subformula $\forall y.\, Q(x, f(y, z))$ of $\forall x.\, \forall y.\, Q(x, f(y, z))$

if we want to remove the outer quantifier, we need to pick a name that we replace $\langle 1 \rangle$ with, say x, and then obtain the tree representation in fig. 7.2b.

In what follows, we will not work explicitly with tree representations, but will use another approach, see section 7.1.3 below. However, the tree representation shows how we can solve the initial problems:

1. The formulas $\forall x.\, P(x)$ and $\forall y.\, P(y)$ have the same tree representation, see fig. 7.3.

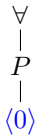


Figure 7.3.: De Bruijn-Tree representing both formulas $\forall x.\, P(x)$ and $\forall y.\, P(y)$

(a) Representation before substitution (b) Representation after substitution

Figure 7.4.: Substitution of the free variable x for the free variable y by using the tree representation of $\forall x.\, Q(x, y)$

2. Substitutions can be carried out without changing the binding of variables, see fig. 7.4. Note that the *bound* variable x is represented by $\langle 0 \rangle$ and thus there is no danger of substituting the *unbound* variable x for the *unbound* variable y. Hence, the problem of accidentally binding a variable does not exist for the tree representation.

7.1.3. Axiomatising Terms and Substitutions

The tree representations of formulas that we saw in section 7.1.1 solves our problems but it is difficult to work with. In fact, these trees are good way of *implementing* FOL on a computer, but not for humans to work with. The aim of this section is to introduce a bunch of axioms that formulas and substitution have to fulfil. These axioms are fulfilled if we were to represent terms by de Bruijn trees, but for the remainder of the course, we will leave unspecified how formulas are implemented.

We begin with the definition of substitutions and how they can be applied to terms.

Definition 7.1

A **substitution** is a map $\sigma\colon \mathrm{Var} \to \mathrm{Term}$. Given $t \in \mathrm{Term}$ and $x \in \mathrm{Var}$, we write $\sigma[x := t]$ for the **updated substitution** defined by

$$(\sigma[x := t])(y) = \begin{cases} t, & x = y \\ \sigma(y), & x \neq y \end{cases}$$

and $\eta\colon \mathrm{Var} \to \mathrm{Term}$ for the **identity substitution** given by $\eta(y) = y$. For convenience, we write $[x := t]$ as shorthand for $\eta[x := t]$. Given a term t, we write $t\,\sigma$ for the **application** of t to the substitution σ, defined by iteration as follows.

$$\begin{aligned} x\,\sigma &= \sigma(x) \\ c\,\sigma &= c \\ f(t_1, \ldots, t_n)\,\sigma &= f(t_1\,\sigma, \ldots, t_n\,\sigma) \end{aligned}$$

The notation $\sigma[x := t]$ to update a substitution σ can be read like an assignment in an imperative programming language: the term that σ assigned to x will be overwritten by t. Similarly, the notation $[x := t]$ starts with a storage in which all variables y have the default value y, except for x which gets t assigned as initial value.

?

Can you make sense of the name *identity substitution*? What is the result of applying a term to η?

The following example illustrates these definitions.

Example 7.2

Let $g(x, y)$ be a term with two free variables x and y, and one binary function symbol g. Moreover, let $\sigma = [x := y][y := x]$.

1. The substitution σ exchanges the two variables:

$$g(x, y)\,\sigma = g(x\,\sigma, y\,\sigma) = g(\sigma(x), \sigma(y)) = g(y, x)$$

This example shows that the substitution $[x := y][y := x]$ is *not* the same as η. In fact, σ is substituting for x and y *in parallel* and not in sequence. Therefore, we also cannot obtain σ by repeated application:

$$g(x, y)\,[x := y][y := x] \neq (g(x, y)\,[x := y])\,[y := x] = g(x, x)$$

2. Let now $\tau = \sigma[y := f(x)]$. In τ, the assignment of x to y is overwritten with the assignment of $f(x)$ to y. We thus have the following.

$$g(x, y)\,\tau = g(x\,\tau, y\,\tau) = g(y, f(x))$$

Next, we need to be able to apply substitutions to formulas. To circumvent the difficulties described in section 7.1.1, we *assume* that there is a set of terms on which we can carry out substitutions. This set of terms can *in principle* be defined by appealing to the tree representation. However, this is tedious and instead we just assume that formulas and the application of substitutions fulfil certain *axioms*.

Definition 7.3: Extension of definition 6.7

Given a formula φ, a variable x and a substitution σ, we say that x is **fresh for φ** , written $x \# \varphi$, if $x \notin \mathrm{fv}(\varphi)$. Similarly, we say that x is **fresh for σ** , written $x \# \sigma$, if

$$x \notin \mathrm{var}(\sigma(y)) \text{ for all } y \in \mathrm{Var} \setminus \{x\}\,.$$

Additionally to the closure rules in definition 6.7, we assume that we can **apply a FOL formula φ to a substitution σ** , written $\varphi\,\sigma$. Further, we assume for all $\Diamond \in \{\forall, \exists\}$ and $\Box \in \{\wedge, \vee, \to\}$ that the equality

on formulas fulfils the following six axioms.

$$\varphi \,\square\, \psi = \varphi' \,\square\, \psi' \text{ iff } \varphi = \varphi' \text{ and } \psi = \psi' \tag{EC}$$
$$\Diamond x.\, \varphi = \Diamond y.\, \psi \text{ iff } y \# \varphi \text{ and } \psi = \varphi[x := y] \tag{EQ}$$
$$\bot\, \sigma = \bot \tag{SB}$$
$$P(t_1, \dots, t_n)\, \sigma = P(t_1\, \sigma, \dots, t_n\, \sigma) \tag{SP}$$
$$(\varphi \,\square\, \psi)\, \sigma = \varphi\, \sigma \,\square\, \psi\, \sigma \tag{SC}$$
$$(\Diamond x.\, \varphi)\, \sigma = \Diamond x.\, \varphi\, \sigma[x := x] \text{ if } x \# \sigma \tag{SQ}$$

?

Do you know why we need to exclude in the definition of $x \# \sigma$ the variable x from the quantification?

Let us explain the intuition behind these axioms. First of all, there are two groups of axioms: those for the equality on formulas (starting with E) and those that define the action of substitution on formulas (starting with S). The axiom (EQ) allows us to bijectively rename bound variables without illegally binding other variables. For instance, we have

$$\forall x.\, Q(x, y) = \forall z.\, Q(z, y)$$

because z is fresh for $Q(x, y)$, that is $z \notin \mathrm{fv}(Q(x, y))$. However, we have

$$\forall x.\, Q(x, y) \neq \forall y.\, Q(y, y)$$

because $y \in \mathrm{fv}(Q(x, y))$. The axiom (EC) allows us to carry out this renaming also in complex formulas that involve other connectives than quantifiers. Implicitly, we also use equality on terms and atoms, in the sense that

$$P(t_1, \dots, t_n) = P'(t'_1, \dots, t'_n) \text{ iff } P = P' \text{ and } t_k = t'_k \text{ for all } 1 \leq k \leq n$$

and that equality is an equivalence relation (reflexive, symmetric and transitive).

The second group of axioms describes how substitution can be computed iteratively. Axioms (SB), (SP) and (SC) are doing what we would expect: no action on the atom $\bot$, reduce subsitution on predicates to substitution in terms, and distribute substitution over propositional connectives. Complications arise

only in the axiom (SQ), which has to make sure that the *use of a bound variable is not changed* and that *variables are not accidentally bound.* On the one hand, that the use of a bound variable is not changed is achieved by updating the substitution σ to $\sigma[x := x]$. For instance, if $\sigma(x) = g(y)$, then naively carrying out this substitution on $\forall x.\, P(x)$ would lead to $\forall x.\, P(g(y))$, which is certainly not what we want! Instead, we have by (SQ) and (SP)

$$(\forall x.\, P(x))\,\sigma = \forall x.\, P(x)\,\sigma[x := x] = \forall x.\, P(x).$$

Accidental binding is prevented, on the other hand, by the precondition that x must be fresh for σ. This condition ensures that none of the terms we want to substitute for the (free) variables in φ contains the variable x, which would become then bound by the quantifier.

These rules and their interaction are best illustrated through some examples.

Example 7.4

In the following, we use the substitution σ given by

$$\sigma = [y := x][z := m].$$

1. Let $\varphi = \forall z.\, Q(z, y)$. First, we note that $\mathrm{var}(\sigma(y)) = \{x\}$ and thus $z \notin \mathrm{fv}(\sigma(v))$ for all $v \neq z$. Thus, $z \# \sigma$. This allows us to carry out the substitution:

$$\begin{aligned} \varphi\,\sigma &= \forall z.\, Q(z, y)\,\sigma[z := z] && \text{(SQ)} \\ &= \forall z.\, Q(z\,\sigma[z := z], y\,\sigma[z := z]) && \text{(SP)} \\ &= \forall z.\, Q(z, x) \end{aligned}$$

   Note that $\sigma[z := z] = [y := x]$ and the substitution of m for z in σ was "forgotten" when we applied the substitution under the quantifier. This is intuitively expected, as the bound variable z in φ is a local reference, while the z in σ refers to a global variable z that has the same name but is distinct from the local variable.

2. Let $\psi = \forall x.\, Q(x, y)$. First, we note that $\mathrm{var}(\sigma(y)) = \{x\}$. Thus, $x \in \mathrm{fv}(\sigma(y))$ and x is *not* fresh. However, we have $z \# Q(x, y)$ and $Q(x, y)[x := z] = Q(z, y)$. This allows us to rename the bound variable x in ψ to z and then carry out the substitution as above:

$$\begin{aligned} (\forall x.\, Q(x, y))\,\sigma &= (\forall z.\, Q(z, y))\,\sigma && \text{(EQ)} \\ &= \forall z.\, Q(z, x) && \text{by 1.} \end{aligned}$$

> Note that we cannot safely rename z back to x, as we would otherwise illegally bind x.

In the remainder, we will not make explicit use of the axioms provided in definition 7.3. Instead, we will rename *bound* variables, if necessary, before carrying out substitutions. For instance, we would just write

$$(\forall x.\, Q(x, y))\,[y := x] = \forall z.\, Q(z, x)$$

without explicitly referring to the axioms. However, we know that in cause of doubt, we can always go back to the axioms and formally carry out the renaming and substitution.

Our equality axioms for formulas and the possibility of renaming variables whenever necessary has as consequence that bound variables are *internal* to formulas and cannot be observed. The supposed map bv that extracts bound variables from a formula that we introduced in definition 6.9 is in fact not well-defined for formulas that fulfil the axioms of definition 7.3. For instance, the formulas $\forall x.\, P(x)$ and $\forall y.\, P(y)$ are equal according to the axioms but

$$\mathrm{bv}(\forall x.\, P(x)) = \{x\} \neq \{y\} = \mathrm{bv}(\forall y.\, P(y))$$

and thus bv cannot be a map. In general, whenever we want to define a map on formulas, then such a map has to respect the equality and thus be invariant under renaming. More precisely, if $f \colon \mathrm{Form}(\mathcal{L}) \to A$ is a map and $\varphi = \psi$, according to definition 7.3, then $f(\varphi) = f(\psi)$.

7.2. Natural Deduction for FOL

Note: This section has no explanation. Please refer to the lectures.

Similarly to propositional logic, we want *syntactic proofs* for FOL.

7.2.1. The Intuitionistic System $\mathbf{ND}_1$

As we reason about objects in FOL, we need a new definition of sequents:

Definition 7.5

A **first-order sequent** for a signature $\mathcal{L}$ is a triple

$$\Delta \mid \Gamma \vdash \varphi\,,$$

where Δ is a list of variables in Var, Γ is a list of $\mathcal{L}$-formulas, φ is a $\mathcal{L}$-formula, and $\mathrm{fv}(\Gamma) \cup \mathrm{fv}(\varphi) \subseteq |\Delta|$ for $|\Delta| = \{x \in \mathrm{Var} \mid x : \Delta\}$. If Δ is the empty list $\cdot$, we write

$$\Gamma \vdash \varphi$$

instead of $\cdot \mid \Gamma \vdash \varphi$.

Example 7.6

- This is a sequent:

  $$x, y \mid \forall z.\, Q(z, y) \vdash P(x)$$

  because $\mathrm{fv}(\forall z.\, Q(z, y)) \cup \mathrm{fv}(P(x)) = \{x, y\}$.

- This is a sequent:

  $$x, y, z \mid \forall z.\, Q(z, y) \vdash P(x)$$

  because $\mathrm{fv}(\forall z.\, Q(z, y)) \cup \mathrm{fv}(P(x)) = \{x, y\} \subseteq \{x, y, z\}$.

- This is not a sequent:

  $$x \mid \forall z.\, Q(z, y) \vdash P(x)$$

  because $\mathrm{fv}(\forall z.\, Q(z, y)) \cup \mathrm{fv}(P(x)) \not\subseteq \{x\}$.

Definition 7.7: Intuitionistic natural deduction for FOL

The **system $\mathbf{ND}_1$** for FOL is given by the rules in fig. 7.5, where the label $x \notin \Delta$ in the rules ($\forall$I) and ($\exists$E) are side-conditions that have to be fulfilled to apply those rules. However, these side-condition will not be displayed in proof trees.

$$\frac{\varphi : \Gamma}{\Delta \mid \Gamma \vdash \varphi}\ (\text{Assum}) \qquad \frac{\Delta \mid \Gamma \vdash \bot}{\Delta \mid \Gamma \vdash \varphi}\ (\bot\text{E})$$

$$\frac{\Delta \mid \Gamma \vdash \varphi \wedge \psi}{\Delta \mid \Gamma \vdash \varphi}\ (\wedge\text{E}_1) \qquad \frac{\Delta \mid \Gamma \vdash \varphi \wedge \psi}{\Delta \mid \Gamma \vdash \psi}\ (\wedge\text{E}_2)$$

$$\frac{\Delta \mid \Gamma \vdash \varphi \qquad \Delta \mid \Gamma \vdash \psi}{\Delta \mid \Gamma \vdash \varphi \wedge \psi}\ (\wedge\text{I})$$

$$\frac{\Delta \mid \Gamma \vdash \varphi}{\Delta \mid \Gamma \vdash \varphi \vee \psi}\ (\vee\text{I}_1) \qquad \frac{\Delta \mid \Gamma \vdash \psi}{\Delta \mid \Gamma \vdash \varphi \vee \psi}\ (\vee\text{I}_2)$$

$$\frac{\Delta \mid \Gamma \vdash \varphi \vee \psi \qquad \Delta \mid \Gamma, \varphi \vdash \delta \qquad \Delta \mid \Gamma, \psi \vdash \delta}{\Delta \mid \Gamma \vdash \delta}\ (\vee\text{E})$$

$$\frac{\Delta \mid \Gamma, \varphi \vdash \psi}{\Delta \mid \Gamma \vdash \varphi \to \psi}\ (\to\text{I}) \qquad \frac{\Delta \mid \Gamma \vdash \varphi \to \psi \qquad \Delta \mid \Gamma \vdash \varphi}{\Delta \mid \Gamma \vdash \psi}\ (\to\text{E})$$

$$(x \notin \Delta)\ \frac{\Delta, x \mid \Gamma \vdash \varphi}{\Delta \mid \Gamma \vdash \forall x.\, \varphi}\ (\forall\text{I}) \qquad \frac{\Delta \mid \Gamma \vdash \forall x.\, \varphi}{\Delta \mid \Gamma \vdash \varphi\,[x := t]}\ (\forall\text{E})$$

$$\frac{\Delta \mid \Gamma \vdash \varphi\,[x := t]}{\Delta \mid \Gamma \vdash \exists x.\, \varphi}\ (\exists\text{I}) \qquad (x \notin \Delta)\ \frac{\Delta \mid \Gamma \vdash \exists x.\, \varphi \qquad \Delta, x \mid \Gamma, \varphi \vdash \psi}{\Delta \mid \Gamma \vdash \psi}\ (\exists\text{E})$$

Figure 7.5.: Deduction Rules of the natural deduction system $\mathbf{ND}_1$

Note about fig. 7.5:

- In ($\forall$E) and ($\exists$I), the definition of a sequent ensures in $\Delta \mid \Gamma \vdash \varphi[x := t]$ that all the free variables of t appear in Δ.

- Similarly, it is ensured in ($\exists$E) that the variable x does not occur freely in ψ because $\Delta \mid \Gamma \vdash \psi$ is a sequent.

7.2.2. Fitch-Style Deduction for $\mathbf{ND}_1$

Example 7.10

We prove $\neg\exists x.\,\varphi \vdash \forall x.\,\neg\varphi$.

1	$\neg\exists x.\,\varphi$	
2	x	
3	φ	
4	$\exists x.\,\varphi$	$\exists$I, 3
5	$\bot$	$\neg$E, 1, 4
6	$\neg\varphi$	$\neg$I, 3–5
7	$\forall x.\,\neg\varphi$	$\forall$I, 2–6

7.2.3. The Classical System $\mathbf{cND}_1$

Definition 7.11

The **system $\mathbf{cND}_1$** of natural deduction for *classical* first-order logic is given by the system $\mathbf{ND}_1$ together with the contradiction rule:

$$\frac{\Delta \mid \Gamma, \neg\varphi \vdash \bot}{\Delta \mid \Gamma \vdash \varphi}\ \text{(Contra)}$$

7.3. Exercises

The following exercises allow you to practise the material of this chapter.

Exercise 1

Give the de Bruijn-tree representations of the following formulas.

a) $Q(x, g(m))$
b) $\forall x.\, Q(x, g(m))$
c) $(\exists x.\, Q(x, g(m))) \wedge \forall y.\, Q(y, z)$
d) $\exists y.\, Q(y, g(z)) \wedge \forall x.\, Q(y, f(x, z))$
e) $\forall x.\, \exists x.\, P(x)$

Exercise 2

Let σ be given by

$$\sigma = [x := g(x)][y := x][z := y].$$

Evaluate the following applications of terms to σ.

a) $x\,\sigma$
b) $f(x, y)\,\sigma$
c) $(y\,\sigma)\,\sigma$
d) $x\,(\sigma[x := x])$

Exercise 3

Let σ be given by

$$\sigma = [x := g(x)][y := x][z := y]$$

and φ by

$$\varphi = \forall x.\, Q(x, y, z).$$

Determine which of the following relations hold.

a) $x \# \sigma$
b) $y \# \sigma$
c) $z \# \sigma$
d) $x \# \varphi$
e) $y \# \varphi$
f) $z \# \varphi$

Exercise 4

Let σ be given by

$$\sigma = [x := g(x)][y := x][z := y].$$

Carry out the following substitutions.

a) $Q(x, g(m))\,\sigma$
b) $(\forall x.\, Q(x, g(m)))\,\sigma$
c) $((\exists x.\, Q(x, g(m))) \wedge \forall y.\, Q(y, z))\,\sigma$
d) $(\exists y.\, Q(y, g(z)) \wedge \forall x.\, Q(y, f(x, z)))\,\sigma$

8. Semantics of First-Order Logic

In this chapter, we will discuss the following two questions:

1. What are objects and their properties and when is a formula valid?
2. Are all derivable sequents valid entailments?

The first question requires us to find a mathematical model of function and predicate symbols, which we can extend to semantics of terms and formulas. We will focus here on Boolean semantics for simplicity. The second question will be answered by proving that both our proof systems $\mathbf{ND}_1$ and $\mathbf{cND}_1$ are sound with respect to the Boolean semantics.

Let us begin by finding out what an appropriate model of function and predicate symbols may be.

8.1. Models of First-Order Logic

Recall that in propositional logic a valuation $v\colon \mathrm{PVar} \to \mathbb{B}$ on propositional variables determined the truth value of formulas. In first-order logic we need to provide interpretations instead for object variables, terms and predicates. Finding the right structures to do so is the subject of this section.

First, we recall some notation.

Notation 8.1
Given a set A and $n \in \mathbb{N}$, we write A^n for the n-fold **finite product** of A, consisting of n-tuples of elements in A: as follows.

$$A^n = \{(a_1, \dots, a_n) \mid a_k \in A \text{ for } k = 1, \dots, n\}$$

We identify A^1 with A for simplicity. Note also that A^0 consists only of the single element $()$, which is a tuple with no entries.

For example, the first few finite products look as follows.

$$A^0 = \{()\} \qquad A^1 = A \qquad A^2 = \{(a, b) \mid a, b \in A\}$$

Definition 8.2

Let $\mathcal{L}$ be a signature with $\mathcal{L} = (\mathcal{F}, \mathcal{R}, \mathrm{ar})$. An **$\mathcal{L}$-model** $\mathcal{M}$ consists of

- a set U, the **universe** of $\mathcal{M}$
- for each $f \in \mathcal{F}$ a map
$$\mathcal{M}(f) \colon U^{\mathrm{ar}(f)} \to U$$
- for each $P \in \mathcal{R}$ a predicate
$$\mathcal{M}(P) \subseteq U^{\mathrm{ar}(P)}$$

We also write $|\mathcal{M}|$ for U.

Note that the universe in in definition 8.2 may be empty. This leads to some subtleties that are often swept under the rug. We will discuss this further after definition 8.11.

Further, observe that if $c \in \mathcal{F}$ is a constant, then $\mathcal{M}(c)$ is a map of type $U^0 \to U$. Since U^0 has one element, providing such a map corresponds to providing one element $a \in U$:

$$\mathcal{M}(c)() = a$$

Let us unfold the definition for some specific arities. Suppose our signature $\mathcal{L}$ consists of three function symbols c, f and g with arity 0, 1 and 2, respectively, and three predicate symbols P, Q, R also of arity 0, 1 and 2. An $\mathcal{L}$-model $\mathcal{M}$ consists of a universe U, maps and predicates as in table 8.1.

We see that $\mathcal{M}(c)$ denotes one element $\mathcal{M}(c)(*) \in U$, as discussed above; $\mathcal{M}(f)$ is a unary map; and $\mathcal{M}(g)$ is a binary map. Moreover, we find that $\mathcal{M}(P)$ is either the empty set $\emptyset$ or the singleton set U^0. In other words, P is nothing but a *propositional variable*! Finally, $\mathcal{M}(Q)$ is a predicate, or unary relation, while $\mathcal{M}(R)$ is a binary relation.

	Symbol s	ar(s)	Interpretation type
Function Symbols	c	0	$\mathcal{M}(c)\colon U^0 \to U$
	f	1	$\mathcal{M}(f)\colon U \to U$
	g	2	$\mathcal{M}(g)\colon U^2 \to U$
Predicate Symbols	P	0	$\mathcal{M}(P) \subseteq U^0$
	Q	1	$\mathcal{M}(Q) \subseteq U$
	R	2	$\mathcal{M}(R) \subseteq U^2$

Table 8.1.: Data of a model for the indicated signature

?

Let $\mathcal{L} = (\emptyset, \mathcal{R}, \mathrm{ar})$ with $\mathcal{R} = \{P\}$ and $\mathrm{ar}(P) = 0$. How many possibilities are there to make a model for $\mathcal{L}$?

In the next example, we discuss a signature and two models that occur "in the wild".

Example 8.3

Let $\mathcal{L}$ be given as follows.

$$\mathcal{F} = \{\underline{0}, \underline{1}, p\} \qquad \mathcal{R} = \{I, L\}$$
$$\mathrm{ar}(\underline{0}) = \mathrm{ar}(\underline{1}) = 0 \qquad \mathrm{ar}(p) = \mathrm{ar}(I) = \mathrm{ar}(L) = 2$$

The interpretation of this signature could be that $\underline{0}$ and $\underline{1}$ stand for the numbers 0 and 1, p for addition (plus), I for equality (identity), and L for less-than. Indeed, we can give such an interpretation, which leads to the model $\mathcal{M}_a$ in table 8.2 with universe $|\mathcal{M}_a| = \mathbb{N}$.

It is not necessary to interpret $\mathcal{L}$ as in example 8.3 and we can give different meanings to the symbols and even the universe.

Symbol	Interpretation type	Interpretation
$\underline{0}$	$\mathcal{M}_a(\underline{0})\colon \mathbb{N}^0 \to \mathbb{N}$	$\mathcal{M}_a(\underline{0})() = 0$
$\underline{1}$	$\mathcal{M}_a(\underline{1})\colon \mathbb{N}^0 \to \mathbb{N}$	$\mathcal{M}_a(\underline{1})() = 1$
p	$\mathcal{M}_a(p)\colon \mathbb{N}^2 \to \mathbb{N}$	$\mathcal{M}_a(p)(n,m) = n + m$
I	$\mathcal{M}_a(I) \subseteq \mathbb{N}^2$	$\mathcal{M}_a(I) = \{(n,m) \mid n = m\}$
L	$\mathcal{M}_a(L) \subseteq \mathbb{N}^2$	$\mathcal{M}_a(L) = \{(n,m) \mid n \leq m\}$

Table 8.2.: Arithmetic model $\mathcal{M}_a$ over universe $\mathbb{N}$

Example 8.4

We show in this example how to use the same signature $\mathcal{L}$ as in example 8.3 to reason about formal languages, where we use the notation introduced in appendix B.4: The function symbols $\underline{0}$, $\underline{1}$ and p correspond to, respectively, the empty language, the language containing only the empty and union of languages. The predicate symbols, on the other hand, can be interpreted as language equality and language inclusion. All of this is summed up in table 8.3. Note that that $\underline{0}$ and p behave similarly to 0 and addition under this interpretation. We will discuss this later in more depth.

Symbol	Interpretation type	Interpretation
$\underline{0}$	$\mathcal{M}_l(\underline{0})\colon \mathcal{P}(A^*)^0 \to \mathcal{P}(A^*)$	$\mathcal{M}_l(\underline{0})() = \emptyset$
$\underline{1}$	$\mathcal{M}_l(\underline{1})\colon \mathcal{P}(A^*)^0 \to \mathcal{P}(A^*)$	$\mathcal{M}_l(\underline{1})() = \{\varepsilon\}$
p	$\mathcal{M}_l(p)\colon \mathcal{P}(A^*)^2 \to \mathcal{P}(A^*)$	$\mathcal{M}_l(p)(L_1, L_2) = L_1 \cup L_2$
I	$\mathcal{M}_l(I) \subseteq \mathcal{P}(A^*)^2$	$\mathcal{M}_l(I) = \{(L_1, L_2) \mid L_1 = L_2\}$
L	$\mathcal{M}_l(L) \subseteq \mathcal{P}(A^*)^2$	$\mathcal{M}_l(L) = \{(L_1, L_2) \mid L_1 \subseteq L_2\}$

Table 8.3.: Language model $\mathcal{M}_l$ for $\mathcal{L}$ over universe $\mathcal{P}(A^*)$

Clearly, the two models $\mathcal{M}_a$ and $\mathcal{M}_l$ in example 8.3 and example 8.4 are completely different interpretations of $\mathcal{L}$. This illustrates the power of first-order

logic: one language we can reason about an enormous variety of different structures. However, as we will in chapter 9, this power can also become a weakness of first-order logic.

8.2. Valuations and the Interpretion of FOL

Just as propositional variables in propositional logic, the object variables in first-order formulas have no intrinsic meaning. Instead, we have to give meaning to them externally through valuations, which assign to each variable an element of a given universe.

Definition 8.5

Given a signature $\mathcal{L}$ and an $\mathcal{L}$-model $\mathcal{M}$, an *$\mathcal{M}$-valuation*, or simply *valuation*, is a map v of the following type.

$$v \colon \mathrm{Var} \to |\mathcal{M}|$$

With interpretations of variables at our disposal, we can understand also the meaning of terms.

Definition 8.6

A valuation v in a model $\mathcal{M}$ extends to the *semantics of terms* $[\![-]\!]_v^{\mathcal{M}} \colon \mathrm{Term} \to |\mathcal{M}|$ by iteration on terms as follows.

$$\begin{aligned} [\![x]\!]_v^{\mathcal{M}} &= v(x) \\ [\![c]\!]_v^{\mathcal{M}} &= \mathcal{M}(c)() \\ [\![f(t_1, \dots, t_n)]\!]_v^{\mathcal{M}} &= \mathcal{M}(f)([\![t_1]\!]_v^{\mathcal{M}}, \dots, [\![t_n]\!]_v^{\mathcal{M}}) \end{aligned}$$

If $\mathcal{M}$ is clear from the context, then we just write $[\![-]\!]_v$ instead of $[\![-]\!]_v^{\mathcal{M}}$.

Let us demonstrate valuations and the term semantics for the models in examples 8.3 and 8.4.

Example 8.7

We begin with the arithmetic model $\mathcal{M}_a$ from example 8.3. Let $x \in \text{Var}$ be some variable and define

$$v(y) = \begin{cases} 5, & x = y \\ 0, & x \neq y \end{cases}$$

Under this valuation, the term $p(\underline{0}, p(x, \underline{1}))$ gets the following semantics assigned.

$$\begin{aligned} [\![p(\underline{0}, p(x, \underline{1}))]\!]_v &= [\![\underline{0}]\!]_v + [\![p(x, \underline{1})]\!]_v \\ &= [\![\underline{0}]\!]_v + ([\![x]\!]_v + [\![\underline{1}]\!]_v) \\ &= \mathcal{M}_a(\underline{0})() + (v(x) + \mathcal{M}_a(\underline{1})()) \\ &= 0 + (5 + 1) \\ &= 6 \end{aligned}$$

The next example provides semantics for the same term but in the language model.

Example 8.8

Let $x \in \text{Var}$ be some variable and define

$$v(y) = \begin{cases} A, & x = y \\ \emptyset, & x \neq y \end{cases}$$

Under this valuation, the term $p(\underline{0}, p(x, \underline{1}))$ gets the following semantics assigned.

$$\begin{aligned} [\![p(\underline{0}, p(x, \underline{1}))]\!]_v &= [\![\underline{0}]\!]_v \cup [\![p(x, \underline{1})]\!]_v \\ &= [\![\underline{0}]\!]_v \cup ([\![x]\!]_v \cup [\![\underline{1}]\!]_v) \\ &= \mathcal{M}_l(\underline{0})() \cup (v(x) \cup \mathcal{M}_l(\underline{1})()) \\ &= \emptyset \cup (A \cup \{\varepsilon\}) \\ &= \{w \in A^* \mid \text{length}(w) \leq 1\} \end{aligned}$$

Here, $\text{length}(w)$ is the length of the word w.

Now that we have an interpretation of terms, we can further extend it to an

interpretation on formulas. Recall that we needed to update substitutions to eliminate universal quantifiers and introduce existential quantifiers. A similar operation on valuations is necessary in the semantics of FOL formulas.

Definition 8.9

Let v be an $\mathcal{M}$ valuation, $x \in \mathrm{Var}$ and $a \in |\mathcal{M}|$. We define the *update of valuations* on v by the following equation.

$$(v[x \mapsto a])(y) = \begin{cases} a, & y = x \\ v(y), & y \neq x \end{cases}$$

Let us briefly illustrate the update of valuations.

Example 8.10

We start with the valuation $v \colon \mathrm{Var} \to \mathbb{N}$ given by $v(y) = 0$ for all $y \in \mathrm{Var}$. Given variables $x, z \in \mathrm{Var}$ with $x \neq z$, we have

$$\begin{aligned}
(v[x \mapsto 1])(x) &= 1 \\
(v[x \mapsto 1])(z) &= 0 \\
(v[x \mapsto 1][z \mapsto 2])(x) &= 1 \\
(v[x \mapsto 1][z \mapsto 2])(z) &= 2 \\
(v[x \mapsto 1][z \mapsto 2][x \mapsto 3])(x) &= 3 \\
(v[x \mapsto 1][z \mapsto 2][x \mapsto 3])(z) &= 2
\end{aligned}$$

The update operation of valuations allows us now to give semantics to quantifiers and thereby to all formulas.

Definition 8.11

Let $\mathcal{L}$ be a signature and $\mathcal{M}$ an $\mathcal{L}$-model. We define for all $\mathcal{M}$-valuations v a map

$$[\![-]\!]_v \colon \mathrm{Form} \to \mathbb{B}$$

by iteration on formulas.

$$
\begin{aligned}
[\![\bot]\!]_v &= 0 \\
[\![P(t_1, \dots, t_n)]\!]_v &= \begin{cases} 1, & ([\![t_1]\!]_v, \dots, [\![t_n]\!]_v) \in \mathcal{M}(P) \\ 0, & \text{otherwise} \end{cases} \\
[\![\varphi \wedge \psi]\!]_v &= \min\{[\![\varphi]\!]_v, [\![\psi]\!]_v\} \\
[\![\varphi \vee \psi]\!]_v &= \max\{[\![\varphi]\!]_v, [\![\psi]\!]_v\} \\
[\![\varphi \to \psi]\!]_v &= [\![\varphi]\!]_v \implies [\![\psi]\!]_v \\
[\![\forall x.\, \varphi]\!]_v &= \min\{[\![\varphi]\!]_{v[x \mapsto a]} \mid a \in |\mathcal{M}|\} \\
[\![\exists x.\, \varphi]\!]_v &= \max\{[\![\varphi]\!]_{v[x \mapsto a]} \mid a \in |\mathcal{M}|\}
\end{aligned}
$$

First of all, note that definition 8.11 is correct for quantifiers because renaming of variables does not affect the result:

$$[\![\forall x.\, \varphi]\!]_v = [\![\forall y.\, \varphi\,[x := y]]\!]_v \text{ for all } y \# \varphi$$

because

$$\min\{[\![\varphi]\!]_{v[x \mapsto a]} \mid a \in |\mathcal{M}|\} = \min\{[\![\varphi\,[x := y]]\!]_{v[y \mapsto a]} \mid a \in |\mathcal{M}|\}$$

Thus, the definition of semantics is compatible with the equality of formulas that we have required to hold in chapter 7.

Recall that we allow the universe of a model to be empty. This creates some subtleties. For instance, if φ is a propositional tautology, then $\exists x.\, \varphi$ is *not* a tautology, unless the signature has constants! This is because in an empty model, we have

$$[\![\exists x.\, \varphi]\!]_v = \max \emptyset = 0$$

The possibilities that arise in the semantics of quantifiers are summed up in table 8.4 together with the corresponding quantification. Note, however, that the first row does not appear if there are constants in the signature.

Let us now calculate the truth value of some formulas.

$\{[\![\varphi]\!]_{v[x\mapsto a]} \mid a \in \lvert\mathcal{M}\rvert\}$	$[\![\forall x.\, \varphi]\!]_v$	$[\![\exists x.\, \varphi]\!]_v$	Interpretation
$\emptyset$	1	0	the model is empty
$\{0\}$	0	0	φ holds for no a
$\{1\}$	1	1	φ holds for all a
$\{0, 1\}$	0	1	φ holds for some a

Table 8.4.: Possibilities for quantifier semantics

Example 8.12

Recall the signature $\mathcal{L}$ and arithmetic model $\mathcal{M}_a$ from example 8.3. Consider the formula

$$\forall x.\, I(p(x, \underline{0}), x)$$

that expresses under the arithmetical interpretation that $x + 0 = x$. In other words, the formula should be true for any valuation over $\mathcal{M}_a$. Indeed, given a valuation v and a natural number n, we have

$$[\![p(x, \underline{0})]\!]_{v[x\mapsto n]} = 0 + n = n$$

and thus

$$\begin{aligned}
[\![I(p(x, \underline{0}), x)]\!]_{v[x\mapsto n]} &= \begin{cases} 1, & \left([\![p(x, \underline{0})]\!]_{v[x\mapsto n]}, [\![x]\!]_{v[x\mapsto n]}\right) \in \mathcal{M}_a(I) \\ 0, & \text{otherwise} \end{cases} \\
&= \begin{cases} 1, & n = n \\ 0, & \text{otherwise} \end{cases} \\
&= 1.
\end{aligned}$$

As expected, this gives us

$$\begin{aligned}
[\![\forall x.\, I(p(x, \underline{0}), x)]\!]_v &= \min\{[\![I(p(x, \underline{0}), x)]\!]_{v[x\mapsto n]} \mid n \in \mathbb{N}\} \\
&= \min\{1\} \\
&= 1
\end{aligned}$$

and thus $\forall x.\, I(p(x, \underline{0}), x)$ is true in $\mathcal{M}_a$.
As a second example, consider the formula

$$\forall x.\, \exists y.\, L(x, y) \wedge \neg I(x, y)$$

that states that for every number there is a strictly larger number. This formula is true in the arithmetic model because for every $n \in \mathbb{N}$ we have that $n < n + 1$. In other words, for every valuation v and $n \in \mathbb{N}$

$$[\![L(x, y) \land \neg I(x, y)]\!]_{v[x \mapsto n][y \mapsto n+1]} = 1.$$

Thus,

$$\begin{aligned}
&[\![\exists y.\, L(x, y) \land \neg I(x, y)]\!]_{v[x \mapsto n]} \\
&= \max\{[\![L(x, y) \land \neg I(x, y)]\!]_{v[x \mapsto n][y \mapsto m]} \mid m \in \mathbb{N}\} \\
&= 1 \qquad \text{because } n + 1 \in \mathbb{N}
\end{aligned}$$

and

$$\begin{aligned}
&[\![\forall x.\, \exists y.\, L(x, y) \land \neg I(x, y)]\!]_v \\
&= \min\{[\![\exists y.\, L(x, y) \land \neg I(x, y)]\!]_{v[x \mapsto n]} \mid n \in \mathbb{N}\} \\
&= \min\{1\} \\
&= 1
\end{aligned}$$

?

Is the formula $\forall x.\, \exists y.\, L(x, y) \land \neg I(x, y)$ from example 8.12 true in the language model $\mathcal{M}_l$?

8.3. Entailment and Satisfiability for FOL

In section 8.2, we have defined the functional interpretation of first-order formulas in terms of the mapping $[\![-]\!]$. As for propositional logic, we can also give a relational interpretation of formulas. This allows us to easily define satisfiability and tautologies.

Definition 8.13

Let φ be an $\mathcal{L}$-formula and Γ a set of $\mathcal{L}$-formulas. We define the *semantic entailment*, where $[\![\Gamma]\!]_v = \min\{[\![\psi]\!]_v \mid \psi \in \Gamma\}$, as follows.

$$\begin{array}{rll} \Gamma \vDash_{\mathcal{M}} \varphi & \text{if } [\![\Gamma]\!]_v^{\mathcal{M}} \leq [\![\varphi]\!]_v^{\mathcal{M}} \text{ for all valuations } v & (\Gamma \textit{ entails } \varphi \text{ in } \mathcal{M}) \\ \Gamma \vDash \varphi & \text{if } [\![\Gamma]\!]_v^{\mathcal{M}} \leq [\![\varphi]\!]_v^{\mathcal{M}} & (\Gamma \textit{ entails } \varphi) \\ & \text{for all models } \mathcal{M} \text{ and valuations } v & \end{array}$$

We say that $\mathcal{M}$ and v *satisfy* φ if $[\![\varphi]\!]_v^{\mathcal{M}} = 1$, and that φ is *satisfiable*, if there is a model $\mathcal{M}$ and an $\mathcal{M}$-valuation v that satisfy φ. The formula φ is a *tautology*, written $\vDash \varphi$, if $\emptyset \vDash \varphi$. Finally, we say that $\mathcal{M}$ *validates* φ, if $\emptyset \vDash_{\mathcal{M}} \varphi$, that is, φ is a tautology only within the model $\mathcal{M}$.

Let us give some examples in the arithmetic model.

Example 8.14

Let φ_e be the formula $\exists y.\, I(x, p(y, y))$ ("x is even"), and let v_1 and v_2 be given as follows.

$$v_1(z) = \begin{cases} 2, & z = x \\ 1, & z \neq x \end{cases} \qquad v_2(z) = \begin{cases} 3, & z = x \\ 1, & z \neq x \end{cases}$$

Then $\mathcal{M}_a$ and v_1 satisfy φ_e, but $\mathcal{M}_a$ and v_2 do not. Therefore φ_e is satisfiable but not validated by $\mathcal{M}_a$.
Let $\Gamma = \{\varphi_e\}$ and $\varphi_o = \exists y.\, I(x, p(p(y, y), \underline{1}))$. Then

$$\Gamma \vDash_{\mathcal{M}_a} \varphi_o\,[x := p(x, \underline{1})]$$

holds: If $[\![\Gamma]\!]_v^{\mathcal{M}_a} = 1$, then $v(x)$ must be an even number. Thus, $v(x)+1$ is an odd number and $[\![\varphi_o\,[x := p(x, \underline{1})]]\!]_v = 1$. Formally,

$$\begin{array}{ll} & [\![\Gamma]\!]_v = 1 \\ \text{iff} & [\![\varphi_e]\!]_v = 1 \\ \text{iff} & \min\{[\![I(x, p(y, y))]\!]_{v[y \mapsto n]} \mid n \in \mathbb{N}\} = 1 \\ \text{iff} & [\![I(x, p(y, y))]\!]_{v[y \mapsto n]} = 1 \text{ for some } n \in \mathbb{N} \qquad \text{(see table 8.4)} \\ \text{iff} & v(x) = n + n \text{ for some } n \in \mathbb{N} \\ \text{iff} & v(x) \text{ even} \end{array}$$

and under this assumption

$$\begin{aligned}
[\![\varphi_o[x := p(x, \underline{1})]]\!]_v &= [\![(\exists y.\, I(x, p(p(y, y), \underline{1})))[x := p(x, \underline{1})]]\!]_v \\
&= [\![\exists y.\, I(p(x, \underline{1}), p(p(y, y), \underline{1}))]\!]_v \\
&= \max\{[\![I(p(x, \underline{1}), p(p(y, y), \underline{1}))]\!]_{v[y \mapsto n]} \mid n \in \mathbb{N}\} \\
&\geq [\![I(p(x, \underline{1}), p(p(y, y), \underline{1}))]\!]_{v[y \mapsto v(x)/2]} \\
&= 1,
\end{aligned}$$

where we use the identity $v(x) + 1 = (v(x)/2 + v(x)/2) + 1$ in the last line.

To show that a formula is not a tautology, it can be convenient or even necessary to choose the interpretation of function or predicate symbols appropriately, as the following example shows.

Example 8.15

We have seen in example 8.12 that $[\![\forall x.\, I(p(x, \underline{0}), x)]\!]_v^{\mathcal{M}_a} = 1$ for any valuation v and therefore $\mathcal{M}_a$ validates $\forall x.\, I(p(x, \underline{0}), x)$. However, this formula is not a tautology because it is not satisfied by the model $\mathcal{M}$ with $|\mathcal{M}| = \mathbb{N}$, $\mathcal{M}(\underline{0}) = 1$ and otherwise the same interpretation as in the arithmetic model $\mathcal{M}_a$.

In chapter 7, we have seen formulas that were derivable in $\mathbf{ND}_1$, which are a good source of tautologies, see also section 8.4 below.

Example 8.16

We claim that the formula $(\forall x.\, \varphi) \to \neg\exists x.\, \neg\varphi$ is a tautology for any formula φ. Indeed, let $\mathcal{M}$ be a model for the signature $\mathcal{L}$, over which φ is a formula, and v a valuation in $\mathcal{M}$. We obtain from table 8.4 that

$$\begin{aligned}
[\![\neg\exists x.\, \neg\varphi]\!]_v = 1 &\text{ iff } [\![\exists x.\, \neg\varphi]\!]_v = 0 \\
&\text{iff } [\![\neg\varphi]\!]_{v[x \mapsto a]} = 0 \text{ for all } a \in |\mathcal{M}| \\
&\text{iff } [\![\varphi]\!]_{v[x \mapsto a]} = 1 \text{ for all } a \in |\mathcal{M}| \\
&\text{iff } [\![\forall x.\, \varphi]\!]_v = 1
\end{aligned}$$

This implies that

$$[\![\forall x.\, \varphi]\!]_v \leq [\![\neg\exists x.\, \neg\varphi]\!]_v$$

and thus

$$\begin{aligned}&[\![(\forall x.\, \varphi) \to \neg\exists x.\, \neg\varphi]\!]_v \\ &= [\![(\forall x.\, \varphi)]\!]_v \implies [\![\neg\exists x.\, \neg\varphi]\!]_v \\ &= 1.\end{aligned}$$

Hence, $(\forall x.\, \varphi) \to \neg\exists x.\, \neg\varphi$ is a tautology.
It should be noted that implication and entailment are, like in propositional logic, closely related. In this example, we have $\forall x.\, \varphi \vDash \neg\exists x.\, \neg\varphi$ and the proof proceeds in essentially the same way: First, we have that $\forall x.\, \varphi \vDash \neg\exists x.\, \neg\varphi$ holds if and only if $[\![\forall x.\, \varphi]\!]_v \leq [\![\neg\exists x.\, \neg\varphi]\!]_v$ for all models and valuations. If $[\![\forall x.\, \varphi]\!]_v = 0$, we have nothing to prove, while if $[\![\forall x.\, \varphi]\!]_v = 1$ we use the reasoning above to obtain that $[\![\neg\exists x.\, \neg\varphi]\!]_v = 1$. Therefore, $[\![\forall x.\, \varphi]\!]_v \leq [\![\neg\exists x.\, \neg\varphi]\!]_v$ holds for all models and valuations and $\forall x.\, \varphi \vDash \neg\exists x.\, \neg\varphi$.

8.4. Soundness of Natural Deduction for FOL

We come now to the second initial question: Are all derivable formulas tautologies? Using definition 8.13, we can state this question precisely by asking: if there is a proof for the sequent $\Gamma \vdash \varphi$ in one of the systems from chapter 7, is φ then entailed semantically by Γ? The answer to this question is the main result of this chapter.

Theorem 8.17: Soundness

For every formula φ and list of formulas Γ over a signature $\mathcal{L}$

1. if $\Gamma \vdash \varphi$ is derivable in $\mathbf{ND}_1$, then $\Gamma \vDash \varphi$.
2. if $\Gamma \vdash \varphi$ is derivable in $\mathbf{cND}_1$, then $\Gamma \vDash \varphi$.

Instantiating theorem 8.17 with the empty list of assumptions, we obtain the following corollary.

Corollary 8.18

For every formula φ over a signature $\mathcal{L}$ the following holds.

1. If $\vdash \varphi$ is derivable in $\mathbf{ND}_1$, then φ is a tautology.
2. If $\vdash \varphi$ is derivable in $\mathbf{cND}_1$, then φ is a tautology.

The most important use of theorem 8.17 is to show that a formula is not provable.

Example 8.19

Recall from example 8.15 that $\forall x.\, I(p(x, \underline{0}), x)$ is not a tautology. Thus, this formula cannot be provable in $\mathbf{ND}_1$, as we would obtain a contradiction with corollary 8.18.

The proof of theorem 8.17 requires some interesting results that essentially show that substitutions are the syntactic counterpart of valuations. This is proved below in lemma 8.20 for terms and in lemma 8.21 for formulas.

Lemma 8.20

For all substitutions σ, terms t, valuations v and variables x, where x is fresh for σ, the following equation holds.

$$[\![t\,\sigma]\!]_v = [\![t\,(\sigma[x := x])]\!]_{v[x \mapsto [\![\sigma(x)]\!]_v]}$$

Proof. We let $\sigma' = \sigma[x := x]$ and $v' = v[x \mapsto [\![\sigma(x)]\!]_v]$, which means that we have to prove

$$[\![t\,\sigma]\!]_v = [\![t\,\sigma']\!]_{v'}$$

This proof proceeds by induction on the term t.

- In the base case we have for a variable y:

$$\begin{aligned}
[\![y\,\sigma]\!]_v &= [\![\sigma(y)]\!]_v && \text{def. substitution}\\
&= \begin{cases} [\![\sigma(x)]\!]_v, y = x \\ [\![\sigma(y)]\!]_v, y \neq x \end{cases} \\
&= \begin{cases} [\![x]\!]_{v'}, y = x \\ [\![\sigma(y)]\!]_v, y \neq x \end{cases} && \text{def. } v' \\
&= \begin{cases} [\![x]\!]_{v'}, y = x \\ [\![\sigma(y)]\!]_{v'}, y \neq x \end{cases} && x \text{ fresh for } \sigma \\
&= [\![y\,\sigma']\!]_{v'} && \text{def. substitution update}
\end{aligned}$$

- In the induction step, we have

$$\begin{aligned}
&[\![f(t_1, \dots, t_n)\,\sigma]\!]_v \\
&= [\![f(t_1\,\sigma, \dots, t_n\,\sigma)]\!]_v && \text{def. substitution}\\
&= \mathcal{M}(f)([\![t_1\,\sigma]\!]_v, \dots, [\![t_n\,\sigma]\!]_v) && \text{def. semantics}\\
&= \mathcal{M}(f)([\![t_1\,\sigma']\!]_{v'}, \dots, [\![t_n\,\sigma']\!]_{v'}) && \text{induction hypothesis}\\
&= [\![f(t_1, \dots, t_n)\,\sigma']\!]_{v'} && \text{def. semantics and subst.}
\end{aligned}$$

Thus, by induction on t, the sought after identity $[\![t\,\sigma]\!]_v = [\![t\,\sigma']\!]_{v'}$ holds. □

The following lemma extends lemma 8.20 to formulas.

Lemma 8.21

For all substitutions σ, formulas φ, valuations v and variables x, where x is fresh for σ, the following equation holds.

$$[\![\varphi\,\sigma]\!]_v = [\![\varphi\,(\sigma[x := x])]\!]_{v[x \mapsto [\![\sigma(x)]\!]_v]}$$

Proof. As in lemma 8.20, we let $\sigma' = \sigma[x := x]$ and $v' = v[x \mapsto [\![\sigma(x)]\!]_v]$, and then prove

$$[\![\varphi\,\sigma]\!]_v = [\![\varphi\,\sigma']\!]_{v'}$$

by induction on φ.

- In the predicate base case, we have

$$\begin{aligned}
[\![P(t_1,\dots,t_n)\,\sigma]\!]_v &= [\![P(t_1\,\sigma,\dots,t_n\,\sigma)]\!]_v && \text{def. substitution}\\
&= \mathcal{M}(P)([\![t_1\,\sigma]\!]_v,\dots,[\![t_n\,\sigma]\!]_v) && \text{def. semantics}\\
&= \mathcal{M}(P)([\![t_1\,\sigma']\!]_{v'},\dots,[\![t_n\,\sigma']\!]_{v'}) && \text{lemma 8.20}\\
&= [\![P(t_1,\dots,t_n)\,\sigma']\!]_{v'} && \text{def. semantics and subst.}
\end{aligned}$$

- The base case for $\bot$ is trivial: $[\![\bot\,\sigma]\!]_v = 0 = [\![\bot\,\sigma']\!]_{v'}$.
- The cases for conjunction, disjunction and implication are immediate by the induction hypothesis. We write $\mathbb{B}_\wedge = \min$, $\mathbb{B}_\vee = \max$ and $\mathbb{B}_\to(x,y) = x \Rightarrow y$, which are the binary Boolean functions for their corresponding connective. This gives us

$$\begin{aligned}
[\![(\varphi_1 \square \varphi_2)\,\sigma]\!]_v &= \mathbb{B}_\square([\![\varphi_1\,\sigma]\!]_v, [\![\varphi_2\,\sigma]\!]_v) && \text{def. subst. and semantics}\\
&= \mathbb{B}_\square([\![\varphi_1\,\sigma']\!]_{v'}, [\![\varphi_2\,\sigma']\!]_{v'}) && \text{induction hyp.}\\
&= [\![(\varphi_1 \square \varphi_2)\,\sigma']\!]_{v'} && \text{def. semantics and subst.}
\end{aligned}$$

- For quantifiers assume that y is fresh for σ.

  Before we continue, observe that for any $a \in U$, we can define $w = v[y \mapsto a]$ and $w' = w[x \mapsto [\![\sigma(x)]\!]_v]$. By using the induction hypothesis for ψ with w, we obtain $[\![\psi\,\sigma]\!]_w = [\![\psi\,\sigma']\!]_{w'}$. Since x and y are fresh, we have that $w' = v'[y \mapsto a]$. Thus $[\![\psi\,\sigma]\!]_w = [\![\psi\,\sigma']\!]_{v'[y\mapsto a]}$.

  With this observation, we have

$$\begin{aligned}
[\![(\forall y.\,\psi)\,\sigma]\!]_v &= [\![\forall y.\,\psi\,\sigma]\!]_v && \text{(SQ)}\\
&= \min\{[\![\psi\,\sigma]\!]_{v[y\mapsto a]} \mid a \in U\} && \text{def. semantics}\\
&= \min\{[\![\psi\,\sigma']\!]_{v'[y\mapsto a]} \mid a \in U\} && \text{see above}\\
&= [\![(\forall y.\,\psi)\,\sigma']\!]_{v'} && \text{def. semantics and (SQ)}
\end{aligned}$$

  The same reasoning, replacing min by max, gives us the result also for the existential quantifier.

This concludes the induction and thereby the proof. □

Proof of theorem 8.17. We generalise the statement and prove that $\Delta \mid \Gamma \vdash \varphi$ implies $\Gamma \vDash \varphi$. To this end, we proceed by induction on the proof tree for $\Delta \mid \Gamma \vdash \varphi$ in $\mathbf{cND}_1$. The statement for $\mathbf{ND}_1$ follows from this.

Thus, assume that we are given a proof tree for $\Delta \mid \Gamma \vdash \varphi$. We have to show for all models $\mathcal{M}$ and valuations v in $\mathcal{M}$ that $[\![\Gamma]\!]_v \leq [\![\varphi]\!]_v$. Most of the cases are dealt with in the same way as for propositional logic and we only treat the rules for quantifiers here.

- Suppose the proof tree ends in

$$\frac{\Delta, x \mid \Gamma \vdash \varphi}{\Delta \mid \Gamma \vdash \forall x.\, \varphi} \text{ (}\forall\text{I)}$$

where x does not appear in Δ. We now have

$$\begin{aligned} [\![\forall x.\, \varphi]\!]_v &= \min\{[\![\varphi]\!]_{v[x \mapsto a]} \mid a \in |\mathcal{M}|\} && \text{def. semantics} \\ &\geq \min\{[\![\Gamma]\!]_{v[x \mapsto a]} \mid a \in |\mathcal{M}|\} && \text{by IH} \\ &= [\![\Gamma]\!]_v, && x \text{ not in } \Gamma \end{aligned}$$

where we use the induction hypothesis (IH) for $\Delta, x \mid \Gamma \vdash \varphi$ with model $\mathcal{M}$ and valuation $v[x \mapsto a]$. This identity gives us $\Gamma \models \forall x.\, \varphi$.

- Next, suppose that last rule that is used in the tree is

$$\frac{\Delta \mid \Gamma \vdash \forall x.\, \varphi}{\Delta \mid \Gamma \vdash \varphi\,[x := t]} \text{ (}\forall\text{E)}$$

for some term t with variables in Δ. By the induction hypothesis, we have that $[\![\Gamma]\!]_v \leq [\![\forall x.\, \varphi]\!]_v$. Using the substitution lemma with the substitution $\sigma = [x := t]$, we obtain the desired result:

$$\begin{aligned} [\![\Gamma]\!]_v &\leq [\![\forall x.\, \varphi]\!]_v && \text{by IH} \\ &= \min\{[\![\varphi]\!]_{v[x \mapsto a]} \mid a \in |\mathcal{M}|\} && \text{by def.} \\ &\leq [\![\varphi]\!]_{v[x \mapsto [\![t]\!]_v]} && \text{property of min} \\ &= [\![\varphi\,[x := t]]\!]_v && \text{lemma 8.21} \end{aligned}$$

- The proof for the introduction of existential quantifiers

$$\frac{\Delta \mid \Gamma \vdash \varphi\,[x := t]}{\Delta \mid \Gamma \vdash \exists x.\, \varphi} \text{ (}\exists\text{I)}$$

follows a similar argument:

$$\begin{aligned} [\![\Gamma]\!]_v &\leq [\![\varphi\,[x := t]]\!]_v && \text{by IH} \\ &= [\![\varphi]\!]_{v[x \mapsto [\![t]\!]_v]} && \text{lemma 8.21} \\ &\leq \max\{[\![\varphi]\!]_{v[x \mapsto a]} \mid a \in |\mathcal{M}|\} && \text{property of max} \\ &= [\![\exists x.\, \varphi]\!]_v && \text{by def.} \end{aligned}$$

- Finally, the elimination of existential quantifiers is probably the most tricky part. Suppose that the tree ends with this rule application:

$$(x \notin \Delta)\ \frac{\Delta \mid \Gamma \vdash \exists x.\, \varphi \qquad \Delta, x \mid \Gamma, \varphi \vdash \psi}{\Delta \mid \Gamma \vdash \psi}\ (\exists\mathrm{E})$$

  By the induction hypothesis, we have

  I) $[\![\Gamma]\!]_v \leq [\![\exists x.\, \varphi]\!]_v = \max\{[\![\varphi]\!]_{v[x \mapsto a]} \mid a \in |\mathcal{M}|\}$, and

  II) for all $a \in |\mathcal{M}|$

$$\begin{aligned} \min\{[\![\Gamma]\!]_v, [\![\varphi]\!]_{v[x \mapsto a]}\} &= [\![\Gamma, \varphi]\!]_{v[x \mapsto a]} && x \notin \Delta \\ &\leq [\![\psi]\!]_{v[x \mapsto a]} && \text{by IH} \\ &= [\![\psi]\!]_v && x \notin \Delta \end{aligned}$$

  Putting these together, we obtain

$$\begin{aligned} &[\![\Gamma]\!]_v \\ &\leq \min\{[\![\Gamma]\!]_v, \max\{[\![\varphi]\!]_{v[x \mapsto a]} \mid a \in |\mathcal{M}|\}\} \\ &\qquad\qquad \text{by I and monotonicity of min} \\ &\leq [\![\psi]\!]_v && \text{by II} \end{aligned}$$

  as required.

This induction in proof trees shows that $\Delta \mid \Gamma \vdash \varphi$ in $\mathbf{cND}_1$ implies $\Gamma \vDash \varphi$. □

9. Extensions and Limits of First-Order Logic

First-order logic together with the proof system $\mathbf{cND}_1$ is a strong and expressive logic, but at the same time also severely limited. In this chapter, we will see where this seeming contradiction comes from.

Before we get to that, we will first extend the syntax of first-order logic by a special predicate that allows us to reason about the identity of objects.

9.1. First-Order Logic with Equality

Recall that we used in example 8.3 a binary predicate symbol I to express that two objects are equal. In the arithmetic model $\mathcal{M}_a$, we also interpreted I as the equality of numbers. However, a general model $\mathcal{M}$ is not forced to give this interpretation to I, it can interpret I as any binary relation. For example, the interpretation as inequality is perfectly valid but is the exact opposite of our intention: $\mathcal{M}(I) = \{(a, b) \mid a \neq b\}$.

This problem can be fixed by giving equality a special status and making it part of the syntax and proof system of first-order logic. To this end, we extend the first-order formulas by one extra kind atomic formula of the form $t \doteq s$ for terms s and t, which is a logical formula with the intent of expressing that t is equal to s. We use this special notation the to carefully distinguish the **syntactic equality** $\doteq$ in the logic, from the equality that we use elsewhere to express general equality of mathematical objects. In particular, we previously wrote $s = t$ to say that s and t are equal as terms. For instance, we reasoned about identities like $f(x)\,[x := c] = f(c)$. The "dotted" notation is different from this equality as it is just a syntactic formula. For example, we can form the formula $f(x) \doteq c$. This formula may be true or not, but the notation with the dot does not assign any intrinsic meaning to such statements, whereas $f(x) = c$ cannot be true as identity of terms. The meaning of $\doteq$ will rather come from the proof system and the semantics. It may happen that $f(x) \doteq c$

becomes true in a model, for example, by interpreting f as the successor map with $\mathcal{M}(f)(n) = n + 1$, x as the number 1 and c as the number 2. Thus, it is important to keep in mind that $=$ and $\doteq$ express generally different things.

Definition 9.1

Let $\mathcal{L}$ be a signature. The set $\mathrm{Form}^{=}$ or $\mathrm{Form}(\mathcal{L}^{=})$ of (first-order) **formulas with equality** or *$\mathcal{L}^{=}$-formulas* is closed under the following rules.

$$\frac{t \in \mathrm{Term} \qquad s \in \mathrm{Term}}{t \doteq s \in \mathrm{Form}^{=}}$$

$$\frac{P \in \mathcal{R}_n \qquad t_1 \in \mathrm{Term} \qquad \cdots \qquad t_n \in \mathrm{Term}}{P(t_1, \ldots t_n) \in \mathrm{Form}^{=}} \qquad \frac{\varphi \in \mathrm{Form}^{=}}{(\varphi) \in \mathrm{Form}^{=}}$$

$$\frac{}{\bot \in \mathrm{Form}^{=}} \qquad \frac{\varphi \in \mathrm{Form}^{=} \qquad \psi \in \mathrm{Form}^{=} \qquad \square \in \{\wedge, \vee, \to\}}{\varphi \,\square\, \psi \in \mathrm{Form}^{=}}$$

$$\frac{x \in \mathrm{Var} \qquad \varphi \in \mathrm{Form}^{=}}{\forall x.\, \varphi \in \mathrm{Form}^{=}} \qquad \frac{x \in \mathrm{Var} \qquad \varphi \in \mathrm{Form}^{=}}{\exists x.\, \varphi \in \mathrm{Form}^{=}}$$

The map $\mathrm{fv} \colon \mathrm{Form}^{=} \to \mathcal{P}(\mathrm{Var})$ is defined exactly as the map as on Form, with the following extra case for equality.

$$\boxed{\mathrm{fv}(s \doteq t) = \mathrm{fv}(s) \cup \mathrm{fv}(t)}$$

Finally, the $\mathcal{L}^{=}$-formulas fulfil, additionally to the axioms from definition 7.3, the following axiom for substitution.

$$\boxed{(s \doteq t)\,\sigma = (s\,\sigma \doteq t\,\sigma)} \qquad \text{(SE)}$$

Note that $\mathrm{Form}^{=}$ is defined in the same way as Form with one extra case added for $s \doteq t$ and that there is no difference for $\mathcal{L}^{=}$-formulas that do not involve the equality relation, see also lemma 9.7 below.

Also we need to be careful to distinguish in the axiom (SE) between the atomic equality formulas and the equality of formulas themselves. The equality of formulas, without the dot, compares the two formulas that are made up of the equality connective, with the dot.

Let us revisit the examples involving statements about natural numbers.

Example 9.2: Arithmetic with equality

In example 8.14, we have reasoned about even and odd numbers by using a predicate I that was supposed to represent the equality of numbers. As we have mentioned in the introduction of this section, this approach does not work very well. Let us instead consider the following signature $\mathcal{L}$.

$$\mathcal{F} = \{\underline{0}, \underline{1}, p\} \qquad \mathcal{R} = \{L\}$$
$$\mathrm{ar}(\underline{0}) = \mathrm{ar}(\underline{1}) = 0 \qquad \mathrm{ar}(p) = \mathrm{ar}(L) = 2$$

The symbols in this signature still have the same intent as in example 8.3: $\underline{0}$ and $\underline{1}$ represent the numbers 0 and 1, p addition, and L less-than. We can now express in $\mathrm{Form}(\mathcal{L}^{=})$ the formulas that appeared in examples 8.12 and 8.14 more naturally:

$$\forall x.\, x \doteq p(x, \underline{0}) \qquad (\underline{0} \text{ is neutral})$$
$$\exists y.\, x \doteq p(y, y) \qquad (x \text{ is even})$$
$$\exists y.\, x \doteq p(p(y, y), \underline{1}) \qquad (x \text{ is odd})$$

Recall from table 8.4 that the existential quantifier in $\exists x.\, \varphi$ requires that there is at least one object that has the property φ. Suppose we want to express that there is *exactly one* such object, that is, that there is a unique object with property φ. We can use equality to do exactly that. To say that x is unique with the property φ is expressed by requiring that x is equal to any other object with the same property:

$$\forall y.\, \varphi[x := y] \to x \doteq y$$

Using this expression of uniqueness, we can define unique existential quantification, typically written as $\exists! x.\, \varphi$, by the following formula.

$$\exists x.\, \varphi \wedge (\forall y.\, \varphi[x := y] \to x \doteq y)$$

As uniqueness is one of the most important applications of equality, it deserves its own definition.

Definition 9.3: Uniqueness

Let φ be a formula, t a term and x a free variable in φ. We define a formula $\text{unique}_x(t, \varphi)$ that expresses that t is uniquely among all objects that can be placed in the formula φ for x:

$$\text{unique}_x(t, \varphi) = \forall y.\, \varphi[x := y] \to t \doteq y$$

The variable y is chosen to be fresh for φ and t. Using uniqueness, we can define the **uniqueness quantifier** as follows.

$$\exists!x.\, \varphi \quad = \quad \exists x.\, \varphi \wedge \text{unique}_x(x, \varphi)$$

Another useful application of equality is that it allows us to count how many things there are with a certain property.

Example 9.4: Expressing finite quantities

Not only can we use equality to enforce uniqueness as in definition 9.3, but we can even state that there must be at least or exactly a certain amount of objects with some property. To do so, we need to express that two objects are not identical, which we define here as the negation of equality:

$$s \not\doteq t \quad = \quad \neg(s \doteq t).$$

We can express that there are *at least two* objects with property φ by

$$\exists x.\, \exists y.\, \varphi[x := x] \wedge \varphi[x := y] \wedge x \not\doteq y$$

and that there must be *exactly two* objects for which φ holds by

$$\exists x.\, \exists y.\, \varphi[x := x] \wedge \varphi[x := y] \wedge x \not\doteq y \wedge (\forall z.\, \varphi[x := z] \to z \doteq x \vee z \doteq y)$$

This is called *counting quantification.*

Besides counting objects, we can also use equality to make sure that objects are given by a certain pattern. Let us demonstrate this by specifying the commands of a simple protocol.

Example 9.5: Commands of a vending machine

Suppose we were to specify the interface of a vending machine with a coin slot and button to select the product. Since vending machines occur across computer science curricula, it is worthwhile to figure out how to formalise this specification.

We begin with the signature $\mathcal{L}_v$, where v stands for "vending machine", with predicate symbols Cmd and Prod of arity one for classifying commands, products, a constant coin, and a unary function symbol sel for the product selection command. This model assumes that our vending machine does not distinguish between coins and possible surpluses will be donated. Formally, we define $\mathcal{L}_v = (\mathcal{R}, \mathcal{F}, \mathrm{ar})$, where $\mathcal{R} = \{\mathrm{Cmd}, \mathrm{Prod}\}$, $\mathcal{F} = \{\mathrm{coin}, \mathrm{sel}\}$, $\mathrm{ar}(\mathrm{coin}) = 0$, and $\mathrm{ar}(\mathrm{Cmd}) = \mathrm{ar}(\mathrm{Prod}) = \mathrm{ar}(\mathrm{sel}) = 1$,

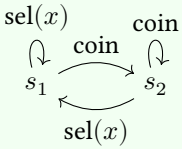
Within the signature $\mathcal{L}_v$, we can now use equality to specify that a command must either be the input of a coin or the selection of a product:

$$\forall x.\, \mathrm{Cmd}(x) \leftrightarrow x \doteq \mathrm{coin} \vee \exists y.\, x \doteq \mathrm{sel}(y) \wedge \mathrm{Prod}(y)$$

This formula can be recognised as a typical data type declaration, here in Haskell-style:

```
data Cmd = Coin | Sel Prod
```

Given that we know how commands look like, we can specify the behaviour of our vending machine. Our vending machine is going to be a bit greedy and not very user friendly: It has two states s_1 and s_2, one that waits for a coin and one that dispenses the chosen product. When the machine is in the first state it will ignore any selection commands, while in the second state it will continue accepting coins, even after already receiving one. This is displayed in the following state diagram.



Before we can reason about this machine, we have to carry out the mundane task of formally specifying this state diagram. To do so, we have to extend the signature $\mathcal{L}_v$ with constants s_1 and s_2, a unary predicate symbol St to classify states, and a binary function symbol f

that represents that transitions of the machine. Our goal is to describe f by a formula, such that

$$\forall x.\, \forall y.\, \mathrm{St}(x) \wedge \mathrm{Cmd}(y) \to \exists! z.\, \mathrm{St}(z) \wedge f(x, y) \doteq z$$

holds. The uniqueness quantifier is what makes f a map, which means that f assigns to every state x and command y a unique state z. Typically, we would specify f by *pattern matching*:

$$\begin{aligned} f(s_1, \mathrm{coin}) &= s_2 & \qquad f(s_1, \mathrm{sel}(x)) &= s_1 \\ f(s_2, \mathrm{coin}) &= s_2 & \qquad f(s_2, \mathrm{sel}(x)) &= s_1 \end{aligned}$$

Such a pattern matching definition ensures automatically that f is well-defined, if we know that there are exactly two states s_1 and s_2, and the two commands coin and sel. We have specified already how commands look like above. Similarly, we can specify that there are only two states:

$$\forall x.\, \mathrm{St}(x) \leftrightarrow x \doteq s_1 \vee x \doteq s_2$$

Such pattern matching definition can be specified in first-order logic with equality as follows.

$$\begin{aligned} \forall x.\, \forall y.\, \mathrm{St}(x) \wedge \mathrm{Cmd}(y) \to \\ \quad & (x \doteq s_1 \wedge y \doteq \mathrm{coin} \to f(x, y) = s_2) \\ \wedge\, & (x \doteq s_2 \wedge y \doteq \mathrm{coin} \to f(x, y) = s_2) \\ \wedge\, & (x \doteq s_1 \wedge \exists z.\, \mathrm{Prod}(z) \wedge y \doteq \mathrm{sel}(z) \to f(x, y) = s_1) \\ \wedge\, & (x \doteq s_2 \wedge \exists z.\, \mathrm{Prod}(z) \wedge y \doteq \mathrm{sel}(z) \to f(x, y) = s_1) \end{aligned}$$

That this formulas specifies f uniquely as a map, follows from the formulas that declare the "data types" Cmd and St. We will, however, refrain from proving this here because such a proof would be quite long and extremely boring. A better approach is to use an extension of first-order logic with types [And02; Jac99], that allows the specification of Cmd, St and f directly by pattern matching. That being said, this example shows that we can *in principle* handle data types and function definitions in first-order logic with equality.

We could now go further and also specify and reason about the behaviour of the vending machine, but at that point we should be really using a computer and a proof assistant.

9.1.1. Semantics of FOL with Equality

The intention of the new symbol $\doteq$ is that it expresses that two objects are identical. With this in mind, we can easily extend the semantics of formulas to account for equality.

Definition 9.6: Formula semantics with equality

Let $\mathcal{L}$ be a signature and $\mathcal{M}$ an $\mathcal{L}$-model. The $\mathcal{L}^{=}$*-semantics* or just semantics of $\mathcal{L}^{=}$-formulas is given for a valuation $v\colon \mathrm{Var} \to |\mathcal{M}|$ by the map

$$[\![-]\!]_v^{=}\colon \mathrm{Form}^{=} \to \mathbb{B}$$

that is defined by iteration on formulas as follows.

$$\begin{aligned}
[\![\bot]\!]_v^{=} &= 0 \\
[\![P(t_1, \dots, t_n)]\!]_v^{=} &= \begin{cases} 1, & ([\![t_1]\!]_v, \dots, [\![t_n]\!]_v) \in \mathcal{M}(P) \\ 0, & \text{otherwise} \end{cases} \\
[\![s \doteq t]\!]_v^{=} &= \begin{cases} 1, & [\![s]\!]_v = [\![t]\!]_v \\ 0, & \text{otherwise} \end{cases} \\
[\![\varphi \wedge \psi]\!]_v^{=} &= \min\{[\![\varphi]\!]_v^{=}, [\![\psi]\!]_v^{=}\} \\
[\![\varphi \vee \psi]\!]^{=} v &= \max\{[\![\varphi]\!]_v^{=}, [\![\psi]\!]_v^{=}\} \\
[\![\varphi \to \psi]\!]_v^{=} &= [\![\varphi]\!]_v^{=} \implies [\![\psi]\!]_v^{=} \\
[\![\forall x.\, \varphi]\!]_v^{=} &= \min\{[\![\varphi]\!]_{v[x \mapsto a]}^{=} \mid a \in |\mathcal{M}|\} \\
[\![\exists x.\, \varphi]\!]_v^{=} &= \max\{[\![\varphi]\!]_{v[x \mapsto a]}^{=} \mid a \in |\mathcal{M}|\}
\end{aligned}$$

If it is clear from the context that $[\![-]\!]_v^{=}$ is applied to an $\mathcal{L}^{=}$-formula φ, then we just write $[\![\varphi]\!]_v$ instead of $[\![\varphi]\!]_v^{=}$. Semantic entailment for formulas and assumptions in $\mathrm{Form}^{=}$ is adapted accordingly: if Γ is a set of formulas and φ a single formula in $\mathrm{Form}^{=}$, then we write,

$$\Gamma \vDash^{=} \varphi \text{ if } [\![\Gamma]\!]_v^{=} \leq [\![\varphi]\!]_v^{=} \text{ for all models } \mathcal{M} \text{ and valuations } v.$$

Note that $[\![-]\!]_v^{=}$ differs from the semantics of formulas without equality only

by the extra case for $s \doteq t$. This means that whenever a formula does not use the equality symbol, then its semantics is given by definition 8.11. The following lemma makes this idea precise.

Lemma 9.7: Preservation of semantics

There is a map $e : \mathrm{Form}(\mathcal{L}) \to \mathrm{Form}(\mathcal{L}^{=})$, such that for all models $\mathcal{M}$, valuations v and formulas $\varphi \in \mathrm{Form}(\mathcal{L})$ the following identity holds.

$$[\![e(\varphi)]\!]^{=}_{v} = [\![\varphi]\!]_{v}$$

Proof. The map e is defined by iteratively mapping a formula in $\mathrm{Form}(\mathcal{L})$ to the same formula in $\mathrm{Form}(\mathcal{L}^{=})$: $e(\bot) = \bot$, $e(P(t_1, \dots, t_n)) = P(t_1, \dots, t_n)$, $e(\varphi \wedge \psi) = e(\varphi) \wedge e(\psi)$ etc. That the semantics of $e(\varphi)$ and φ agree is easily proved by induction. □

This result allows us also to use table 8.4 to determine the semantics of $\mathcal{L}^{=}$-formulas with quantifiers.

Let us go through the semantics of some formulas that use the equality predicate.

Example 9.8

Recall that the arithmetic model $\mathcal{M}_a$ in example 8.3 had to account for the predicate I that modelled equality explicitly. We can now forget about the interpretation of I and use $\mathcal{M}_a$ to give semantics to $\mathcal{L}^{=}$-formulas for the signature $\mathcal{L}$ from example 9.2. For instance, we have for a given valuation $v \colon \mathrm{Var} \to \mathbb{N}$ that

$$\begin{aligned}
[\![\forall x.\, x \doteq p(x, \underline{0})]\!]^{=}_{v} &= \begin{cases} 1, & [\![x \doteq p(x, \underline{0})]\!]_{v[x \mapsto n]} \text{ for all } n \in \mathbb{N} \\ 0, & \text{otherwise} \end{cases} \\
&= \begin{cases} 1, & n = n + 0 \text{ for all } n \in \mathbb{N} \\ 0, & \text{otherwise} \end{cases} \\
&= 1,
\end{aligned}$$

where we used table 8.4 for the first identity. Similarly, we have

$$\begin{aligned}
[\![\exists y.\, x \doteq p(y,y)]\!]^{=}_{v} &= \begin{cases} 1, & [\![x \doteq p(y,y)]\!]_{v[y\mapsto n]} = 1 \text{ for some } n \in \mathbb{N} \\ 0, & \text{otherwise} \end{cases} \\
&= \begin{cases} 1, & v(x) = 2n \text{ for some } n \in \mathbb{N} \\ 0, & \text{otherwise} \end{cases} \\
&= \begin{cases} 1, & v(x) \text{ even} \\ 0, & \text{otherwise} \end{cases}
\end{aligned}$$

9.1.2. Natural Deduction for FOL with Equality

The last step to complete the picture of first-order logic with equality is to give the corresponding proof system. Keeping lemma 9.7 in mind, we expect that such a proof system has the same rules as $\mathbf{ND}_1$ or $\mathbf{cND}_1$ for all connectives and only adds rules for equality. Thus, let us briefly think about how we use equality and try to deduce the proof rules from this intuition.

A common use of equality is equational reasoning. For instance, if a and b are numbers, then

$$ab + a(-b) = a(b - b) = a0 = 0$$

establishes that $ab + a(-b)$ is identical to 0. In fact, if we calculate with concrete numbers, say 2 and 5, then $2 \cdot 5 + 2 \cdot (-5)$ is just another way of writing 0. This leads us to the most basic identity, the identity of an object with itself:

$$x = x$$

This identity is called *reflexivity* and will be the first rule that we adapt into our proof system.

Reflexivity by itself is, however, not enough. We often replace "equals by equals" in equational reasoning. For instance, if we know that $c = 0$ then we can infer

$$d = 0 + d = c + d.$$

From this and the previous identity, we can infer the following much more complex identity.

$$d = (ab + a(-b)) + d$$

The process of replacing equal objects can be generalised to arbitrary formulas and leads us the so-called *replacement rule*.

Formally, the intuitionistic and classical natural deduction proof systems for first-order logic with equality are given in the following definition.

Definition 9.9: Natural deduction with equality

The systems $\mathbf{ND}_1^=$ and $\mathbf{cND}_1^=$ are given by extending, respectively, $\mathbf{ND}_1$ and $\mathbf{cND}_1$ with the following two rules.

$$\frac{}{\Delta \mid \Gamma \vdash t \doteq t}\ (\text{Refl}) \qquad \frac{\Delta \mid \Gamma \vdash s \doteq t \qquad \Delta \mid \Gamma \vdash \varphi[x := s]}{\Delta \mid \Gamma \vdash \varphi[x := t]}\ (\text{Repl})$$

The rule (Refl) is called *reflexivity* and (Repl) is called *replacement*.

We follow in definition 9.9 the traditional naming for the rules, although we could also use the naming scheme of introduction and elimination rules that we employed for other logical connectives. Under this naming scheme, (Refl) would be an introduction rule, while (Repl) would be an elimination rule.

Surprisingly, the two rules (Refl) and (Repl) are enough to fully characterise equality. To give a flavour of the power of the replacement rule, let us prove that $\doteq$ is symmetric and transitive. These two properties form the basis of equational reasoning.

Theorem 9.10: Equality is an equivalence relation

The following two rules of *symmetry* and *transitivity* are admissible in $\mathbf{ND}_1^=$ and $\mathbf{cND}_1^=$.

$$\frac{\Delta \mid \Gamma \vdash s \doteq t}{\Delta \mid \Gamma \vdash t \doteq s}\ (\text{Sym}) \qquad \frac{\Delta \mid \Gamma \vdash s \doteq t \qquad \Delta \mid \Gamma \vdash t \doteq r}{\Delta \mid \Gamma \vdash s \doteq r}\ (\text{Trans})$$

Proof. To derive transitivity, we will have to use the replacement rule. The difficulty in using this rule lies in finding the correct formula φ, in which we replace equal terms. If we take a look at the conclusion of the transitivity rule, then we have several options of choosing such a formula φ. After a bit of trial and error, we can come up with

$$s \doteq x$$

for φ. This formula works because $\varphi[x := r]$ is the conclusion $s \doteq r$ of (Trans) and $\varphi[x := t]$ is the first premise $s \doteq t$ of (Trans). Thus, (Trans) is given by the following application of (Repl), albeit with swapped premises.

$$\frac{\Delta \mid \Gamma \vdash \varphi[x := t] \qquad \Delta \mid \Gamma \vdash t \doteq r}{\Delta \mid \Gamma \vdash \varphi[x := r]} \text{ (Repl)}$$

By the above discussion, this a proof tree for

$$\frac{\Delta \mid \Gamma \vdash s \doteq t \qquad \Delta \mid \Gamma \vdash t \doteq r}{\Delta \mid \Gamma \vdash s \doteq r} \text{ (Trans)}$$

by carrying out the substitution. This shows that the transitivity rule is derivable from (Repl). □

?

Can you find a derivation for the symmetry rule in theorem 9.10 in $\mathbf{ND}_1^=$?

At this point, one may wonder why we have to introduce equality as a new symbol and give new proof systems. Why can we not just add equality to the signature and find axioms Γ so that that $\Gamma, \Gamma' \vdash \varphi$ is derivable in $\mathbf{ND}_1$ if and only if $\Gamma' \vdash \varphi$ is derivable in $\mathbf{ND}_1^=$? For instance, the formula $\forall x.\, x \doteq x$ could serve as an axiom that represents reflexivity. The problem is that the replacement rule ranges over all formulas, which would mean that we have to add an axiom for each formula, including our axioms. This leads to a problem that we cannot solve in first-order logic, if we want to use only finitely many axioms. Alternatively, we can add for all function and predicate symbols in the signature axioms that state that they respect the equality relation. For example, if $f \in \mathcal{F}_1$ and $P \in \mathcal{R}_1$ we would need

$$\text{resp}_f = \forall x.\, \forall y.\, x \doteq y \to f(x) \doteq f(y)$$
$$\text{resp}_P = \forall x.\, \forall y.\, x \doteq y \to P(x) \to P(y)$$

One can then prove that for Γ_0 consisting of axioms for reflexivity, transitivity, symmetry, resp_f and resp_P for all f and P in $\mathcal{L}$ that $\Gamma_0, \Gamma \vdash \varphi$ is provable in $\mathbf{ND}_1$ over the signature $\mathcal{L}^=$, which has $\mathcal{R}^= = \mathcal{R} \cup \{\doteq\}$, iff $\Gamma \vdash \varphi$ is provable in $\mathbf{ND}_1^=$. However, this proof requires induction over formulas and leads thus to quite large proof trees. For this reason we give equality a special status in our logic.

In our semantics of $\mathcal{L}^{=}$-formulas, we interpreted the syntactic equality $\doteq$ as equality in the universe. This make the proof of the following soundness statement an easy extension of soundness of the systems $\mathbf{ND}_1$ and $\mathbf{cND}_1$.

Theorem 9.11

If the sequent $\Gamma \vdash \varphi$ is derivable in $\mathbf{ND}_1^{=}$ or $\mathbf{cND}_1^{=}$, then $\Gamma \vDash \varphi$.

Since the unique quantifier in definition 9.3 is a derived logical connective, we can equip it with introduction and elimination rules. Especially the introduction rule shortens proofs considerably.

Lemma 9.12: Rules for uniqueness quantifier

The following rules are admissible in $\mathbf{ND}_1^{=}$, where the variable y in (∃!I) has to be fresh.

$$\frac{\Delta \mid \Gamma \vdash \varphi[x := t] \qquad \Delta, y \mid \Gamma, \varphi[x := y] \vdash t \doteq y}{\Delta \mid \Gamma \vdash \exists!x.\, \varphi} \ (\exists!\mathrm{I})$$

$$\frac{\Delta \mid \Gamma \vdash \exists!x.\, \varphi \qquad \Delta, x \mid \Gamma, \varphi, \mathrm{unique}_x(x, \varphi) \vdash \psi}{\Delta \mid \Gamma \vdash \psi} \ (\exists!\mathrm{E})$$

?

Can you derive the two rules in lemma 9.12?

We finish this section with a few example proofs in the systems $\mathbf{ND}_1^{=}$ and $\mathbf{cND}_1^{=}$. As before, we will use Fitch-style proofs to make the proofs readable.

Example 9.13

In this example, we will formally prove the incredibly difficult fact that the successor of an even number is odd. To this end, we use the formulas φ_e and φ_o given by

$$\varphi_e = \exists y.\, x \doteq p(y, y) \quad \text{and} \quad \varphi_o = \exists z.\, x \doteq p(p(z, z), \underline{1})$$

to describe even and odd numbers. We then derive the sequent

$$\vdash \forall x.\, \varphi_e \to \varphi_o[x := p(x, \underline{1})]$$

in $\mathbf{ND}_1^=$ by using Fitch-style.

1	x		$\exists y.\, x \doteq p(y,y)$	
2			$y \mid x \doteq p(y,y)$	
3			$p(x,\underline{1}) \doteq p(x,\underline{1})$	Refl
4			$p(x,\underline{1}) \doteq p(p(y,y),\underline{1})$	Repl, 2, 3
5			$\exists z.\, p(x,\underline{1}) \doteq p(p(z,z),\underline{1})$	$\exists$I, 4
6			$\varphi_o[x := p(x,\underline{1})]$	$\exists$E, 1, 2–5
7		$\varphi_e \to \varphi_o[x := p(x,\underline{1})]$		$\to$I, 1–6
8	$\forall x.\, \varphi_e \to \varphi_o[x := p(x,\underline{1})]$			$\forall$I, 1–7

In the proof, we use the reflexivity rule (Refl) in line 3 and the replacement rule (Repl) in line 4. The rule (Repl) is thereby applied to the formula ψ given by $p(x,\underline{1}) \doteq p(u,\underline{1})$ and the substitutions $[u := x]$ and $[u := p(y,y))]$:

$$\frac{\Delta \mid \Gamma \vdash x \doteq p(y,y) \qquad \Delta \mid \Gamma \vdash \psi[u := x]}{\Delta \mid \Gamma \vdash \psi[u := p(y,y)]}$$

where $\Delta = x, y$ and $\Gamma = \varphi_e, x \doteq p(y,y)$ are the context and assumptions introduced in lines 1 and 2.

The next example shows how equality can be used to prove uniqueness of dividers

Example 9.14: Even numbers have unique divider

We all know that division gives unique results, whenever it is defined. In particular, if a number n is even, then there should be a unique number k with $n = 2k$. Formally, we use the formula

$$\forall x.\, \varphi_e \to \exists! z.\, x \doteq p(z,z) \qquad (*)$$

to express uniqueness of divisors. In this example, we will prove uniqueness of divisors, as expressed by $(*)$, in $\mathbf{ND}_1^=$.
Underlying uniqueness of division by two is the fact that doubling a

number is an injective function, that is, $2n = 2m$ implies $m = n$ for all natural numbers m and n:

$$\varphi_{\text{double-inj}} \quad = \quad \forall m.\, \forall n.\, p(m, m) \doteq p(n, n) \to m \doteq n$$

Thus, what we will prove is that uniqueness of divisors is derivable from injectivity of doubling:

$$\varphi_{\text{double-inj}} \vdash \forall x.\, \varphi_e \to \exists! z.\, x \doteq p(z, z) \tag{†}$$

To make injectivity of doubling better usable in the proof of (†) below, let us derive for some context Δ, and terms r, s and t the following sequent.

$$\Delta \mid \varphi_{\text{double-inj}}, r \doteq p(s, s), r \doteq p(t, t) \vdash s \doteq t \tag{♭}$$

This sequent allows us to use an intermediate term t to relate the doubling of s and of t to derive the equality of s and t, and is proven as follows. Note that the proof uses symmetry and transitivity that we derived in theorem 9.10, and that the steps 4 and 5 are akin to the chain of equations $p(s, s) \doteq x \doteq p(t, t)$.

1	$\varphi_{\text{double-inj}}$	
2	$x \doteq p(s, s)$	
3	$x \doteq p(t, t)$	
4	$p(s, s) \doteq x$	Sym, 2
5	$p(s, s) \doteq p(t, t)$	Trans, 4, 3
6	$\forall n.\, p(s, s) \doteq p(n, n) \to s \doteq n$	∀E, 1
7	$p(s, s) \doteq p(t, t) \to s \doteq t$	∀E, 6
8	$s \doteq t$	→E, 7, 5

Using the identity, we can prove the uniqueness of the divisor of even numbers. The derivation of (∗) uses a typical combination: assume that an object with some property exists and prove that this object is unique. In the course of this, we use existential elimination (line 4-7) to inspect

the object that we know to exist, and then introduce the uniqueness quantifier in line 7 by appealing to the above identity (♭).

1	$\varphi_{double-inj}$	
2	x	
3	$\exists y.\, x \doteq p(y,y)$	
4	$y \mid x \doteq p(y,y)$	
5	$y' \mid x \doteq p(y',y')$	
6	$y \doteq y'$	(♭), 4, 5
7	$\exists!z.\, x \doteq p(z,z)$	$\exists!$I, 4, 5–6
8	$\exists!z.\, x \doteq p(z,z)$	$\exists$E, 4–7
9	$(\exists y.\, x \doteq p(y,y)) \to \exists!z.\, x \doteq p(z,z)$	$\to$I, 3–8
10	$\forall x.\, \varphi_e \to \exists!z.\, x \doteq p(z,z)$	$\forall$I, 2–9

Example 9.14 shows how $\mathbf{ND}_1^=$ can be used to do equational reasoning, as it occurs in computer science and mathematics. Clearly, the proofs are fairly lengthy because we have to make every step explicit, which we typically not do on paper. However, the amount of boilerplate can be reduced by using a computer to automate some, or even all, of the steps, while still retaining the certainty of a formal proof as the one above.

9.2. Completeness

The strength of the classical natural deduction systems for FOL are in their completeness theorems. Recall from theorems 8.17 and 9.11 that $\mathbf{cND}_1$ and $\mathbf{cND}_1^=$ are sound, meaning that any statement that is provable is also true semantically:

> If $\Gamma \vdash \varphi$ is derivable in $\mathbf{cND}_1$ or $\mathbf{cND}_1^=$, then $\Gamma \vDash \varphi$.

The completeness theorem establishes the other direction of this implication.

Theorem 9.15: Completeness of $\mathbf{cND}_1$ and $\mathbf{cND}_1^=$

Let $\Gamma \vdash \varphi$ be a first-order sequent.

1. If Γ is a list of $\mathcal{L}$-formulas, φ a $\mathcal{L}$-formula and $\Gamma \vDash \varphi$, then $\Gamma \vdash \varphi$ is derivable in $\mathbf{cND}_1$.
2. If Γ is a list of $\mathcal{L}^=$-formulas, φ a $\mathcal{L}^=$-formula and $\Gamma \vDash^= \varphi$, then $\Gamma \vdash \varphi$ is derivable in $\mathbf{cND}_1^=$.

Just as in the case of propositional logic, this remarkable result tells us that there is a proof tree for any semantically true statement. Unfortunately, the proof of this result is not effective, that is, we cannot extract an actual proof tree from the proof of theorem 9.15 and we only know that such a proof tree has to exist. The reason for this is that the proof of theorem 9.15 is a proof by contradiction and therefore uses classical logic in an essential way. This cannot be avoided, which renders the completeness theorem fairly useless from a computational perspective. If we wanted to use it to derive $\Gamma \vdash \varphi$, then we would have to quantify over all models and establish for each model $\mathcal{M}$ that the Γ entails φ in $\mathcal{M}$. As Γ and φ will likely contain quantifiers, we will then have to quantify over all elements of the universe of $\mathcal{M}$, cf. example 8.16. This is not only difficult, but generally undecidable. Even though the soundness and completeness theorems establish $\mathbf{cND}_1$ as a good proof system for first-order logic, finding a proof tree remains a difficult problem. In the next chapter, we will see a proof system that can prove less than $\mathbf{cND}_1$ and $\mathbf{ND}_1$, but allows us to do proof search.

9.3. Compactness and its Consequences

Closely related to completeness is also compactness. Where the completeness theorem 9.15 told us that any semantically true formula has a formal proof, the compactness theorem will establish limits on properties that we can express as formulas in first-order logic.

Note that (first-order) sequents of the form $\Gamma \vdash \varphi$ always require that Γ is a *finite list* of formulas, whereas the semantic entailment $\Gamma \vDash^= \varphi$ allows that Γ can be *any set*, even an infinite set, of formulas. Theorem 9.15 can be formulated differently to allow for such a general set Γ of assumption. However, one then has to show that $\Gamma \vDash^= \varphi$ uses only a finite number of assumptions from Γ to obtain completeness of $\mathbf{cND}_1^=$. This is because a the proof trees of

$\mathbf{cND}_1^=$ are finite and can therefore only use finitely many assumptions. As a consequence, we get the following result.

Theorem 9.16: Compactness of first-order logic

Let Γ be an *arbitrary set* of first-order formulas. Then $\Gamma \models^= \varphi$ if and only if there is a *finite* $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \models^= \varphi$.

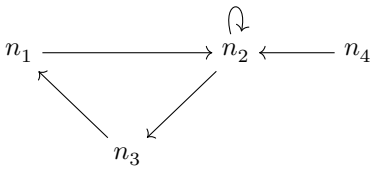
The right-to-left direction of the equivalence between the two conditions is easily established: Observe that if $\Gamma_0 \subseteq \Gamma$, then for any valuation v we have

$$[\![\Gamma]\!]_v^= \leq [\![\Gamma_0]\!]_v^= \leq [\![\varphi]\!]_v^=.$$

This shows that $\Gamma_0 \models^= \varphi$ implies $\Gamma \models^= \varphi$ for any (finite) subset Γ_0 of Γ. The other direction is what makes theorem 9.16 so interesting, and it is this direction that we use now to establish some limits of first-order logic with $\mathbf{cND}_1^=$.

9.3.1. Expressiveness of First-Order Logic

A *(simple directed) graph* G is given by a finite set N of nodes and a relation $E \subseteq N^2$ of edges. Graphs are typically drawn as diagrams like this:



This graph G_1 is given by the set $N_1 = \{n_1, n_2, n_3, n_4\}$ of nodes and the relation E_1 of edges given by

$$E_1 = \{(n_1, n_2), (n_2, n_2), (n_2, n_3), (n_3, n_1), (n_4, n_2)\}.$$

A typical problem in graph theory is reachability:

> Given a graph G and nodes m and n in G, can we reach n from m by traversing along edges of G?

In the above example, we have that n_3 is reachable from n_1 by using the edges (n_1, n_2) and (n_2, n_3). Contrary to that, n_4 is not reachable from n_1 in G_1.

This problem seems to lend itself to formalisation in first-order logic: Let $\mathcal{L}$ be the signature $(\emptyset, \mathcal{R}, \mathrm{ar})$ with $\mathcal{R} = \{R\}$ and $\mathrm{ar}(R) = 2$. The intention is that the relation symbol R represents a binary relation and therefore the edges of a graph. Indeed, given a graph G with nodes N and edges E, we obtain an $\mathcal{L}$-model $\mathcal{M}_G$ by defining

$$|\mathcal{M}_G| = N \quad \text{and} \quad \mathcal{M}_G(R) = E.$$

We can now ask if there is a first-order formula φ over $\mathcal{L}$ with two free variables x and y, such that for all graphs G, all nodes m and n in G and valuations v we have

$$[\![\varphi]\!]^{\mathcal{M}_G}_{[x \mapsto m][y \mapsto n]} = 1 \quad \text{if and only if } n \text{ is reachable from } m \text{ in } G.$$

It is important to note that we ask for a formula that works for all graphs! Let us try to write down such a formula. How can we reach a node y from x? Either x is already y, or an edge connects them, or we make a transition via other nodes. The problem is that there is a priori no upper bound on the number of nodes that we have to visit to get from x to y because the formula has to work for any graph. Thus, we are left with the following attempt to write down a formula, in which the dots indicate that such a formula would have to be infinitely long, which is clearly not what we consider a formula.

$$\begin{aligned} & x \doteq y \\ \vee\ & R(x, y) \\ \vee\ & (\exists z.\, R(x, z) \wedge R(z, y)) \\ \vee\ & (\exists z_1.\, \exists z_2.\, R(x, z_1) \wedge R(z_1, z_2) \wedge R(z_2, y)) \\ \vee\ & \cdots \end{aligned}$$

How do we get out of this? In just first-order logic with equality, we don't!

Theorem 9.17: Graph reachability cannot be expressed

There is no $\mathcal{L}^{=}$-formula φ with $\mathrm{fv}(\varphi) = \{x, y\}$, such that φ is true if and only if y is reachable from x via R.

Proof. Assume that there is an $\mathcal{L}^{=}$-formula with $\mathrm{fv}(\varphi) = \{x, y\}$, such that for every graph $G = (N, E)$ with $\mathcal{M}_G$ as above and for all $n, m \in N$

$$[\![\varphi]\!]^{\mathcal{M}_G}_{[x \mapsto m][y \mapsto n]} = 1 \quad \text{if and only if } n \text{ is reachable from } m \text{ in } G.$$

We define now formulas φ_k with $\mathrm{fv}(\varphi_k) = \{x, y\}$, such that

$$[\![\varphi_k]\!]^{\mathcal{M}_G}_{[x \mapsto m][y \mapsto n]} = 1 \quad \text{if and only if there is a path from } n \text{ to } m \text{ of length } k.$$

Concretely, we define φ_k by iteration on k:

$$\begin{aligned} \varphi_0 &= x \doteq y \\ \varphi_{k+1} &= \exists z.\, R(x, z) \wedge \varphi_k[x := z] \end{aligned}$$

Let now $\Gamma = \{\neg\varphi_k \mid k \in \mathbb{N}\}$, which expresses that there is no path of any length and we clearly have $\Gamma \vDash \neg\varphi$. By the compactness theorem 9.16, we get a finite $\Gamma_0 \subseteq \Gamma$ with $\Gamma_0 \vDash \neg\varphi$. Let $k \in \mathbb{N}$ be the largest number, such that $\varphi_k \in \Gamma_0$. Thus, any path longer than k is not forbidden by Γ_0!

To use this fact, we define a graph $G = (N, E)$ by $N = \{n_0, n_1, \dots, n_{k+1}\}$ and $E = \{(n_i, n_{i+1}) \mid i = 0, \dots, k\}$. This graph G looks essentially like a list

$$n_0 \longrightarrow n_1 \longrightarrow \cdots \longrightarrow n_{k+1}$$

with only one path from n_0 to n_{k+1}, which is furthermore of length $k + 1$. In other words, for this graph G, we have

$$[\![\Gamma_0]\!]^{\mathcal{M}_G}_{[x \mapsto n_0][y \mapsto n_{k+1}]} = 1$$

but

$$[\![\neg\varphi]\!]^{\mathcal{M}_G}_{[x \mapsto n_0][y \mapsto n_{k+1}]} = 0,$$

which contradicts $\Gamma_0 \vDash \neg\varphi$. Hence, the formula φ cannot exist. □

Theorem 9.17 is a severe limitation of first-order logic. Isaac was counting on FOL to find a route that leads it to its heart! However, since routing is essentially graph reachability, FOL will not be able to help 😔 But there is hope! In the exercises and in the next chapter, we will see that our Isaac can still be helped to find its heart.

9.4. Exercises

Exercise 5

a) Formalise the sentence

> "Pavel owes money to everyone but himself"

as a formula φ in first-order logic with equality. You need one constant p for "Pavel" and one binary predicate symbol O for "owes to".

b) Derive for your formula φ the following sequent in $\mathbf{ND}_1^=$ using a Fitch-style proof.

$$\varphi \vdash \neg O(p, p)$$

Exercise 6

Let $\mathcal{L}$ be a signature with a unary predicate symbol P. We define P_1 to be the formula

$$P_1 \quad = \quad \forall y.\, \forall z.\, P(y) \land P(z) \to y \doteq z,$$

which expresses that there can be maximally one object that fulfils P. Prove the following logical equivalence in $\mathbf{ND}_1^=$:

$$\vdash (\exists! x.\, P(x)) \leftrightarrow ((\exists x.\, P(x)) \land P_1)$$

To approach the proof of this formula, do both implications in separate proofs and refer to them in the proof of the logical equivalence. Furthermore, use the derived rules for the uniqueness quantifier from lemma 9.11 in the lecture notes.

Exercise 7 *Graph Reachability*

We have seen that compactness prevents us from giving a *formula* that expresses reachability in graphs. In this exercise, we will see that reachability can be expressed by appropriately defining a *predicate*. Let $\mathcal{L}$ be the signature $(\{E, R\}, \{n_1, \dots, n_4\}, \mathrm{ar})$ with $\mathrm{ar}(E) = \mathrm{ar}(R) = 2$ and $\mathrm{ar}(n_k) = 0$ for $k = 1, \dots 4$. The intention is that $E(x, y)$ holds if there is an edge between x and y in a given graph, and $R(x, y)$ holds if the node y is reachable from x. We will use the constants n_k later to model the nodes of a concrete graph.

Reachability R is the reflexive and transitive closure of the edge relation E. In other words, each node x must be related to itself via E (reflexivity), and if there is an edge from x to some z and y is reachable from z, then y is also reachable from x (transitivity).

a) Give two formulas φ_r and φ_t with free variables x and y that express, respectively, reflexivity and transitivity. That is to say, that the formula φ_R given by

$$\forall x.\, \forall y.\, R(x,y) \leftrightarrow \varphi_r \vee \varphi_t$$

expresses that R is the reachability relation in the graph with edges E.

b) Let $\varphi_{R,1}$, $\varphi_{R,2}$ and $\varphi_{R,3}$ be given by

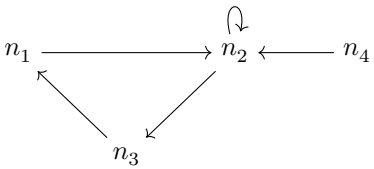
$$\begin{aligned}
\varphi_{R,1} &= \forall x.\, \forall y.\, R(x,y) \to \varphi_r \vee \varphi_t \\
\varphi_{R,2} &= \forall x.\, \forall y.\, \varphi_r \to R(x,y) \\
\varphi_{R,3} &= \forall x.\, \forall y.\, \varphi_t \to R(x,y)
\end{aligned}$$

Derive the following sequent in $\mathbf{ND}_1^=$ using Fitch-style.

$$\varphi_{R,1}, \varphi_{R,2}, \varphi_{R,3} \vdash \varphi_R$$

Note that the proof merely uses the rules for quantifiers and propositional connectives, not those for equality.

c) Consider the following graph G.



Give formulas $\varphi_{E,1}, \dots, \varphi_{E,5}$ that describe the edge of G.

d) Give a proof in $\mathbf{ND}_1^=$ using Fitch-style of $\varphi_R, \varphi_{E,1}, \dots, \varphi_{E,5} \vdash R(n_4, n_3)$. Use b) to simplify the task.

Exercise 8

A *group* G is given by a binary map $\dot{+} \colon G \times G \to G$ and an element $\dot{0}$ of G, such that

1. for all x in G, $x \dot{+} \dot{0} = \dot{0} \dot{+} x = x$,
2. for all x, y, z in G, $x \dot{+} (y \dot{+} z) = (x \dot{+} y) \dot{+} z$, and
3. for every x in G there is a y in G, such that $x \dot{+} y = \dot{0}$ and $y \dot{+} x = \dot{0}$.

The equations 1 - 3 are called the *group axioms*. Groups appear everywhere in computer science and mathematics, with very popular applications in cryptography. The goal of this exercise is to formally reason about groups in first-order logic with equality.

Let $\mathcal{L}$ be the signature with function symbols $\dot{+}$ and $\dot{0}$ of arity 2 and 0, respectively. Let us write, as above, the symbol $\dot{+}$ in infix notation, that is, we write $s \dot{+} t$ instead of $\dot{+}(s, t)$ for terms s and t. We define $\mathcal{L}^{=}$-formulas $\varphi_{1,l}$ and $\varphi_{1,r}$ by

$$\varphi_{1,l} = \forall x.\, \dot{0} \dot{+} x \doteq x \quad \text{and} \quad \varphi_{1,r} = \forall x.\, x \dot{+} \dot{0} \doteq x$$

that formalise together the first group axiom.

a) Give $\mathcal{L}^{=}$-formulas φ_2 and φ_3 that formalise the axioms 2 and 3 from above.

b) Give a formula φ_u that expresses the uniqueness of y in the third group axiom. The element y is called the *inverse* of x.

c) Prove in $\mathbf{ND}_1^{=}$ that the inverse y of x in the third axiom is unique, that is, derive $\vdash \varphi_u$ for your formula in $\mathbf{ND}_1^{=}$. Use Fitch-style as usual.

10. Incompleteness and Undecidability

Note: Incomplete lecture notes with little explanation. Please refer for this chapter to the lecture.

The following definition refers to "computation procedures", which we do not define precisely here. Think of them as a computer programs, a sequence of instructions that can be unambiguously run by a machine. A more precise definition can be given, for example, in terms of Turing machines or by any other equivalent notion of computation.

Definition 10.1

We define the set $\overline{\mathbb{N}}$ of *partial numbers* by $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$, where the symbol ∞ can be understood as *diverging computation* or non-halting computation. A map $f\colon \mathbb{N}^n \to \overline{\mathbb{N}}$ is called an (n-ary) *partial function* or just *function*. The *domain* of f is given by the set all of all inputs on which f converges: $\boxed{\mathrm{dom}(f) = \{x \in \mathbb{N}^n \mid f(x) \neq \infty\}}$. If $\mathrm{dom}(f) = \mathbb{N}^n$, then f is called *total*.
We say that a function f is *computable* if there is a computation procedure that halts on all inputs $x \in \mathrm{dom}(f)$ with output $f(x)$, and does not halt if $f(x) = \infty$. A set $P \subset \mathbb{N}^n$ is *semi-decidable* (or *recursively enumerable*, r.e.) if there is a computable function f with $\mathrm{dom}(f) = P$. Such a set P is *decidable*, if the characteristic function χ_P with

$$\chi_P(x) = \begin{cases} 0, & x \in P \\ 1, & x \notin P \end{cases}$$

is total computable.

The name "recursively enumerable" for semi-decidable sets comes from the fact that a set $P \subseteq \mathbb{N}$ is semi-decidable if and only if P is empty or there is a

total computable function $f\colon \mathbb{N} \to \mathbb{N}$ with $P = \{f(n) \mid n \in \mathbb{N}\}$. Thus, there is a way to *enumerate* the elements of P as $f(0), f(1), \ldots$

Theorem 10.2

The set $\{\varphi \mid \vdash \varphi \text{ derivable in } \mathcal{D}\}$ is for $\mathcal{D} \in \{\mathbf{ND}_1, \mathbf{cND}_1, \mathbf{ND}_1^=, \mathbf{cND}_1^=\}$ semi-decidable but not decidable.

Definition 10.3

A term t or formula φ is *closed*, if $\mathrm{var}(t) = \emptyset$ or $\mathrm{fv}(\varphi) = \emptyset$, respectively.

Definition 10.4

A set Γ of closed formulas is a *theory*, if it is closed under deduction: If φ is closed and $\Gamma \vdash \varphi$ in $\mathbf{cND}_1^=$, then $\varphi \in \Gamma$. We call Γ *axiomatisable* if there is a r.e. set $\Gamma_0 \subseteq \Gamma$, such that $\Gamma = \{\varphi \mid \Gamma_0 \vdash \varphi \text{ in } \mathbf{cND}_1^=\}$. The elements of Γ_0 are called *axioms* and Γ_0 an *axiomatisation*.

Example 10.5

$\Gamma_0 = \{\forall x.\, \neg(s\, x \doteq \underline{0})\}$ axiomatises that zero is not the successor of any number.

Definition 10.6

We shall write $\boxed{\mathcal{L}(\Gamma)}$ for the signature containing all symbols in Γ. Let Γ and Γ' be axiomatised by Γ_0 and Γ_0', respectively. Γ' contains Γ, if $\mathcal{L}(\Gamma) \subseteq \mathcal{L}(\Gamma')$ and $\Gamma_0 \subseteq \Gamma_0'$.

Definition 10.7

The set PR of *primitive recursive* (p.r.) functions is the smallest set containing the following functions.

- Constants: $\mathrm{K}^n\colon \mathbb{N}^n \to \mathbb{N}$ with $\mathrm{K}^n(x) = 0$
- Projection: $\Pi_k^n\colon \mathbb{N}^n \to \mathbb{N}$ with $\Pi_k^n(x_1, \ldots, x_n) = x_k$ for $1 \leq k \leq n$
- Successor: $\mathrm{S}\colon \mathbb{N} \to \mathbb{N}$ with $\mathrm{S}(x) = x + 1$

- Composition: $\mathrm{C}^n_m(g, h_1, \ldots, h_n) \colon \mathbb{N}^m \to \mathbb{N}$ for all p.r. functions $g \colon \mathbb{N}^n \to \mathbb{N}$ and $h_1, \ldots, h_n \colon \mathbb{N}^m \to \mathbb{N}$ defined by

$$\mathrm{C}^n_m(g, h_1, \ldots, h_n)(x) = g(h_1(x), \ldots, h_n(x))$$

- Iteration: $\mathrm{I}^n(g, h) \colon \mathbb{N}^{n+1} \to \mathbb{N}$ for all p.r. functions $g \colon \mathbb{N}^n \to \mathbb{N}$ and $h \colon \mathbb{N}^{n+2} \to \mathbb{N}$ defined by

$$\begin{aligned} \mathrm{I}^n(g, h)(0, x) &= g(x) \\ \mathrm{I}^n(g, h)(x + 1, y) &= h(\mathrm{I}^n(g, h)(x, y), x, y) \end{aligned}$$

Example 10.8

1. The truncated predecessor function $\mathrm{pred} \colon \mathbb{N} \to \mathbb{N}$, specified by

$$\mathrm{pred}(n) = \begin{cases} 0, & n = 0 \\ n - 1, & n > 0 \end{cases},$$

   is primitive recursive, as we have that $\mathrm{pred} = \mathrm{I}^0(\mathrm{K}^0, \Pi^2_2)$. That this identity holds can be seen by evaluating the right-hand side as far as possible and match it with the specification of pred. Since the function is given by iteration, we distinguish the base case and the step case:

$$\begin{aligned} \mathrm{I}^0(\mathrm{K}^0, \Pi^2_2)(n) &= \begin{cases} \mathrm{K}^0(), & n = 0 \\ \Pi^2_2(\mathrm{I}^0(\mathrm{K}^0, \Pi^2_2)(k), k), & k = n + 1 \end{cases} \\ &= \begin{cases} 0, & n = 0 \\ k, & n = k + 1 \end{cases} \\ &= \begin{cases} 0, & n = 0 \\ n - 1, & n > 0 \end{cases} \end{aligned}$$

   This corresponds to our specification of the predecessor.

2. Note that the predecessor is defined by case distinction on its argument. We can use the same idea to show that case distinction is primitive recursive in general. Given $f \colon \mathbb{N}^n \to \mathbb{N}$ and

$g\colon \mathbb{N}^{n+1} \to \mathbb{N}$, the function $[f,g]\colon \mathbb{N}^{n+1} \to \mathbb{N}$, given by

$$[f,g](n,x) = \begin{cases} f(x), & n = 0 \\ g(k,x), & n = k+1 \end{cases},$$

is primitive recursive because

$$[f,g] = \mathrm{I}^n(f, \mathrm{C}(g, \Pi_2^{n+2}, \dots, \Pi_{n+2}^{n+2}))\,.$$

3. Neither of the two examples above has used proper recursion, as both ignored the recursion parameter in their use of iteration. To show that addition is primitive recursive, we observe that it obeys two recursive equations: $0 + m = m$ and $(n+1) + m = (n+m) + 1$. We can emulate these with iteration and define $p\colon \mathbb{N}^2 \to \mathbb{N}$ as follows.

$$p = \mathrm{I}^1(\Pi_1^1, \mathrm{C}_3^1(S, \Pi_1^3))$$

With this definition we have for all $n, m \in \mathbb{N}$ that

$$p(0,m) = \Pi_1^1(m) = m$$

and

$$\begin{aligned} p(n+1,m) &= \mathrm{C}_3^1(S, \Pi_1^3)(p(n,m), n, m) \\ &= S(\Pi_1^3(p(n,m), n, m)) \\ &= S(p(n,m)) \\ &= p(n,m) + 1\,. \end{aligned}$$

Therefore, p fulfils the same recursive equations as $+$, which one can use to prove that $p(n,m) = n + m$.

Note: Remainder of lecture missing!

11. First-Order Horn Clauses and Automatic Deduction

11.1. Automatic Deduction and the Cut-Rule

When approaching proofs of mathematical theorems, we usually decompose the problem into intermediate results that are easier to prove on their own. In the natural deduction systems, the approach is justified by the so-called *cut rule*. This rule allows one to prove first a formula φ, often called a *lemma*, then prove a formula ψ under the assumption of φ, and finally the conclude that ψ holds without explicit reference to the lemma φ. The rule is formulated in the following theorem.

Theorem 11.1: Admissible Cut

The following *cut rule* is admissible in $\mathbf{ND}_1$.

$$\frac{\Gamma \vdash \varphi \qquad \Gamma, \varphi \vdash \psi}{\Gamma \vdash \psi}\ (\text{Cut})$$

Proof. The cut rule is given by the following proof tree.

$$\dfrac{\Gamma \vdash \varphi \qquad \dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\ (\to\text{I})}{\Gamma \vdash \psi}\ (\to\text{E})$$

□

The cut rule is very useful in structuring proofs, but horrendous for automatic deduction because it requires ingenuity for inventing the intermediate lemma φ to prove ψ. There are techniques for generating such lemmas but we will take another route here.

A first step to automatic deduction is to avoid the use of the cut rule and thereby the introduction of an implication that is immediately followed by

an implication elimination, as in the proof of the cut rule. Fortunately, it is possible to avoid such detours and we can proceed without having to make up formulas out of thin air. This is captured by the following theorem.

Theorem 11.2: Cut Elimination

Any proof tree in $\mathbf{ND}_1$ for a sequent $\Gamma \vdash \varphi$ can be transformed into a proof tree in $\mathbf{ND}_1$ for the same sequent, which contains no instances of the cut rule.

Proof. We shall only discuss the idea how detours introduced by a cut can be removed, as its implementation is rather technical. Details can be found in texts on proof theory [TS00; TvD88]

In the (Cut)-rule, we split the proof a lemma φ from its use in a proof of ψ. To avoid a cut, we can thus use the proof of φ directly everywhere we use the assumption φ in the proof of ψ. More precisely, suppose we have a proof tree

$$\dfrac{\vdots}{\Gamma \vdash \varphi}$$

for the sequent $\Gamma \vdash \varphi$. Any proof tree for $\Gamma, \varphi \vdash \psi$ can only use φ through the (Assum)-rule and the tree must be of the following form, where we display only one use of the (Assum)-rule.

$$\dfrac{\dfrac{\dfrac{}{\Gamma, \varphi \vdash \varphi}\,(\text{Assum}) \qquad \cdots}{\Gamma, \varphi \vdash \delta} \qquad \vdots}{\Gamma, \varphi \vdash \psi}$$

We can now replace everywhere the use of the (Assum)-rule applied to φ directly by the proof of φ, as in the following tree.

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \varphi} \qquad \cdots}{\Gamma \vdash \delta} \qquad \vdots}{\Gamma \vdash \psi}$$

Note that the assumption φ goes away. This procedure can be carried out more precisely by induction over proof trees, allowing us substitute the proof tree for φ into that of ψ and thus eliminating the application of the (Cut)-rule. □

This is good news for automatic deduction because we can limit at each step the rules that may be applied by inspecting the formula φ that we have to prove. The approach of inspecting φ in finding a proof for $\Gamma \vdash \varphi$ is also called *goal-oriented* because the search for the proof is driven by the goal φ. However, restricting ourselves to proofs that avoid the cut rule is not enough for fully automatic deduction because there are other choice-points. For instance, if we try to prove $\exists x.\, P(x) \vdash \neg\forall x.\, \neg P(x)$ do we first use the introduction of the negation or the elimination of the existential quantifier? The problem is that the principle formula of elimination rules, the formula that gives a rule its name, appears among the premises of those rules. This means for the goal-oriented construction of a proof that we have to guess the right formula to eliminate from the goal, while there may be a non-trivial relation between the two. In the above example, we have to guess that we have to eliminate $\exists x.\, P(x)$ to prove $\neg\forall x.\, \neg P(x)$, which is difficult for human intelligence, let alone for a computer. One way out of this is to limit the class of formulas that may appear in proofs and thereby reduce the amount of guessing to a manageable amount. Combined with limiting proof rules, we obtain a reasonable fragment of first-order logic for automatic deduction.

11.2. First-Order Horn Clauses and Logic Programming

In chapter 5, we have seen a class of propositional formulas that were called Horn clauses. It turns out that there is an extremely useful generalisation to first-order logic.

Definition 11.3: Horn Clauses and Theories in FOL

A *predicate atom* over a signature $\mathcal{L}$ is a formula of the form $P(t_1, \dots, t_n)$, where P is an n-ary predicate symbol in $\mathcal{L}$ and $t_1, \dots, t_n$ are $\mathcal{L}$-terms. Predicate atoms are denoted by letters $A, B, C, \dots$ A *Horn clause* is a closed formula of the form

$$\forall x_1.\, \cdots \forall x_m.\, A_1 \wedge \cdots \wedge A_n \to A_0$$

for $m, n \in \mathbb{N}$ and predicate atoms $A_0, A_1, \dots, A_n$. The atom A_0 is called the *head* of the clause and the set $\{A_1, \dots, A_n\}$ is called the *body* of the clause. We call a list Γ of Horn clauses a *Horn clause theory* or *logic program*.

The body of a clause may be empty, that is, n may be zero, in which case we leave out the implication in the Horn clause in definition 11.3 and just write $\forall x_1. \cdots \forall x_m. A_0$. This is convenient when writing logic programs that contain Horn clauses without assumptions, so called *base facts*. In the abstract treatment of Horn clauses later in this chapter it will, however, not be necessary to differentiate between base facts and general Horn clauses.

It is also important to note that a Horn clause must be closed, thus only the variables $x_1, \dots, x_m$ may appear in the head and body of a Horn clause. These are then all bound by the quantifiers on the outside.

What makes Horn clause theories so useful? As the name "logic program" indicates, we can use such theories for programming. In fact, we can describe *any* Turing machine by a logic program Γ. A computation corresponds to giving a predicate atom A with free variables $x_1, \dots, x_m$ and asking for a derivation of $\Gamma \vdash \exists x_1. \cdots \exists x_m. A$ in $\mathbf{ND}_1$. If the Turing machine halts, then such a derivation exists. This correspondence makes it possible to automatically find such a derivation if it exists. Thus, finding proofs is also for Horn clause theories only semi-decidable but, as we will see, it is still much easier in many cases to find proofs from Horn clause theories rather than arbitrary theories.

Let us demonstrate the programming aspect of Horn clauses. Recall that in primitive recursive arithmetic and other approaches to arithmetic, we have always assumed that all objects were numbers. Often, we need to reason about different *types* of objects. The following example shows how we can axiomatise with Horn clauses a predicate that characterises natural numbers.

Example 11.4

In this example, we will use that every natural number is either zero or the successor of another natural number. More specifically, the constant $\underline{0}$ represents in the following zero and the unary function symbol s represents the successor of a number. With these two symbols, we can enumerate all the natural numbers:

$$\underline{0},\ s(\underline{0}),\ s(s(\underline{0})),\ s(s(s(\underline{0}))), \dots$$

Writing natural numbers in this way is called *unary encoding*, which is a very bad encoding from the perspective of complexity but very useful for explanatory purposes.

In this example, we will be working with a signature $\mathcal{L}$ that contains a constant $\underline{0}$, a unary function symbol s, a unary predicate symbol N, a

binary predicate symbol L, and a ternary predicate symbol P. We use the unary predicate symbol N to pin down the natural numbers, in the sense that $N(t)$ holds if t is the representation of a natural number. The following two Horn clauses implement precisely the initial description of natural numbers.

$$N(\underline{0})$$
$$\forall n.\, N(n) \to N(s(n))$$

We can now continue and implement arithmetic for natural numbers. To use Horn clauses for this task, we have to switch from thinking in terms of function to thinking in terms of relations. More specifically, we use a ternary predicate symbol P with the intent that $P(m, n, k)$ holds if k is the sum of m and n. To provide Horn clauses that implement addition, recall the primitive recursive specification of addition: $0 + m = m$ and $(n + 1) + m = (n + m) + 1$. Assuming that n and m are given in the above representation of natural numbers, we can represent this recursive specification of addition by the following two Horn clauses.

$$\forall m. \quad P(\underline{0}, m, m)$$
$$\forall n.\, \forall m.\, \forall k.\, P(n, m, k) \to P(s(n), m, s(k))$$

In a similar spirit, we can define other recursive relations on natural numbers. For instance, to specify that the binary predicate L represents the less or equal relation, we would use the following two Horn clauses.

$$\forall n. \quad L(n, n)$$
$$\forall n.\, \forall m.\, L(n, m) \to L(n, s(m))$$

The clauses say that every natural number is less or equal to itself (reflexivity) and that increasing the number on the right preserves the less or equal relation.

?

Can you find two Horn clauses that describe the computation of multiplication as ternary predicate symbol M analogously to P in example 11.4?

What makes logic programming so interesting compared to other programming paradigms? The answer lies in its declarative nature, which allows us to specify what the result of a computation should be rather than how the result is computed. In his seminal work, Kowalski [Kow79] has expressed this in the following equation.

Algorithm = Logic + Control

This equation essentially means that we can design algorithms by providing logical formulas that describe the result of an algorithm and a control mechanism that controls the use of these formulas to compute the result. A particular property of this approach is that new algorithms can be obtained by exchanging the control mechanism, while preserving the logical properties and correctness of the computed result. In terms of the above equation, we may have a logical specification L and algorithms A_1 and A_2 given by $A_1 = L + C_1$ and $A_2 = L + C_2$ for some control mechanisms C_1 and C_2. These two algorithms will compute results with the same logical properties, but may have different computational behaviour. For example, A_2 may be more efficient than A_1.

Let us return to concrete logic programming approach and illustrate it on our initial robot example.

Example 11.5: Path Finding

Remember the board from fig. 6.1 on page 70 that Isaac drew? In section 6.1, we used first-order logic to describe what moves on the board are allowed and how a route towards the heart looks like. The aim of this example is to use Horn clauses to give a logical description of paths that will allow us, with an appropriate control mechanism, to derive an efficient algorithm for routing.

In section 9.3.1, we have seen that reachability in graphs cannot be expressed by a first-order formula. Since path finding for the robot can be seen as reachability in a graph, we cannot directly express paths as a formula. In exercise 3 of chapter 9, we saw how graph reachability could still be expressed by introducing a new predicate that represented the reflexive, transitive closure of the edge relation of a graph. And even better, the two formulas that expressed this are Horn clauses and we can therefore formulate graph reachability as a logic program! Let us now use this approach to help Isaac break this limitation of first-order logic and to finally get to the heart.

For this purpose, we will use the signature $\mathcal{L}$ with constants $\underline{1}, \underline{2}, \ldots$ for all positive natural numbers, one binary function symbol pos, three unary predicate symbols R, H and F, and two binary predicate symbols A and C. Let us explain the intention of all these symbols. We will use pos to describe positions on the board and will denote by $\mathrm{pos}(x, y)$ the position with horizontal coordinate x and vertical coordinate y. For instance, the robot is in fig. 6.1 in position $\mathrm{pos}(\underline{2}, \underline{3})$. The unary predicate symbols R, H and F describe, respectively, the position of the **R**obot, the position of the **H**eart and **F**ree positions. Since the robot may only move between adjacent positions, will use the binary predicate symbol A to express when a position is **A**djacent to another position. Finally, the binary predicate symbol C is what we are after: it will relate two positions if they are **C**onnected by a path via free and adjacent positions.

Our goal is to describe the situation in fig. 6.1 and the predicate symbol C as a logic program, that is, a finite list of Horn clauses. All of the following is provided in appendix D as a Prolog program that can be directly run in your favourite Prolog interpreter.

Let us begin with the easy part: the position of the robot and the heart. These are given by the following two formulas.

$$R(\mathrm{pos}(\underline{2}, \underline{3})) \quad \text{and} \quad H(\mathrm{pos}(\underline{5}, \underline{1})) \tag{11.1}$$

This two formulas are base facts, albeit with no quantifiers and an empty body, that is, m and n in definition 11.3 are both zero.

Next, we describe the free position that the robot may visit by providing a formula for each free position:

$$\begin{array}{cccc} F(\mathrm{pos}(\underline{1}, \underline{1})) & F(\mathrm{pos}(\underline{2}, \underline{2})) & F(\mathrm{pos}(\underline{3}, \underline{1})) & \\ F(\mathrm{pos}(\underline{1}, \underline{4})) & F(\mathrm{pos}(\underline{2}, \underline{3})) & F(\mathrm{pos}(\underline{3}, \underline{2})) & \ldots \\ & F(\mathrm{pos}(\underline{2}, \underline{4})) & F(\mathrm{pos}(\underline{3}, \underline{4})) & \end{array} \tag{11.2}$$

You may have noticed that we have initially talked in section 6.1 about the obstacles on the board and now we talk about free positions instead. The reason for this is that obstacles are an inherently negative description of the allowed moves that the robot may make. More precisely, we have the relation

$$\forall p.\, F(p) \leftrightarrow \neg O(p)\,,$$

where O was the predicate symbol that we used for describing the positions of obstacles. This negative relation between F and O would

become a problem when we ask if the robot can move to a certain position because the first-order Horn clauses in definition 11.3 may not use $\bot$ and therefore no negation! Thus, we would not be able to formulate the routing problem as a logic program if we describe positions with obstacles instead of free positions.

To finish the board description, we have to give all the adjacent positions by providing formulas that specify the predicate symbol A. This predicate should be read as, if $A(p, q)$ holds then position p and q are adjacent. For instance, we should have that $A(\text{pos}(\underline{1}, \underline{1}), \text{pos}(\underline{1}, \underline{2}))$ holds. However, neither $A(\text{pos}(\underline{1}, \underline{1}), \text{pos}(\underline{2}, \underline{2}))$ nor $A(\text{pos}(\underline{6}, \underline{1}), \text{pos}(\underline{7}, \underline{1}))$ should not hold. In the first case, the two positions $\text{pos}(\underline{1}, \underline{1})$ and $\text{pos}(\underline{2}, \underline{2})$ are not adjacent, as we do not allow diagonal steps. In the second case, the term $\text{pos}(\underline{7}, \underline{1})$ is not a valid position on the board because the horizontal coordinates range from 1 to 6 only. Formalising all of this as Horn clauses is a bit tedious because we have to enumerate for each position all its four neighbours, except for positions at the boundary that have two or three neighbours:

$$\begin{aligned}
&A(\text{pos}(\underline{x}, \underline{y}), \text{pos}(\underline{x+1}, \underline{y})), \quad 1 \leq x \leq 5, \quad 1 \leq y \leq 4 \\
&A(\text{pos}(\underline{x}, \underline{y}), \text{pos}(\underline{x}, \underline{y+1})), \quad 1 \leq x \leq 6, \quad 1 \leq y \leq 3 \\
&A(\text{pos}(\underline{x}, \underline{y}), \text{pos}(\underline{x-1}, \underline{y})), \quad 2 \leq x \leq 6, \quad 1 \leq y \leq 4 \\
&A(\text{pos}(\underline{x}, \underline{y}), \text{pos}(\underline{x}, \underline{y-1})), \quad 1 \leq x \leq 6, \quad 2 \leq y \leq 4
\end{aligned} \tag{11.3}$$

The first line of (11.3) states that every position $\text{pos}(\underline{x}, \underline{y})$ is adjacent to its east neighbour $\text{pos}(\underline{x+1}, \underline{y})$. Note that $\text{pos}(\underline{x}, \underline{y})$ only has a neighbour in the east if it is not a boundary position, that is, if $x \leq 5$. In the remaining three lines, we analogously provide the specification of the southern, western and northern neighbours.

With the board layout described through the Horn clauses in eqs. (11.1) to (11.3), we can now try to find a way for Isaac to the heart. We achieve this by providing a description of the predicate symbol C with the intention that $C(p, q)$ holds if position q is reachable from p via free and adjacent positions. Reachability can be described by two Horn clauses, one that states that every position can be reached from itself and one for making an intermediate step via a free position:

$$\forall p. \quad C(p, p) \tag{11.4}$$

$$\forall p.\, \forall q.\, \forall r.\, C(q, r) \wedge A(p, q) \wedge F(q) \rightarrow C(p, r) \tag{11.5}$$

These two Horn clauses formulate reachability as backward search. That is to say, we start with the final position we want to reach and then try to find positions that make a path to the initial position.

11.3. Uniform Proofs

In example 11.5, we have used logical formulas to describe how a path from the starting position to the goal looks like. What is missing to get an algorithm for routing is the control part of Kowalski's equation, which allows us to search for a path. We will introduce in this section a control mechanism based on proof theory, called uniform proofs.

The idea of uniform proofs is that we devise a proof system tailored towards proving goal formulas of a very specific shape only from Horn clauses. This restriction of goals and assumptions has a two-fold effect:

1. we remove all the choices appearing in elimination rules, and
2. we concentrate the choices in one proof rule.

Removing elimination rules in general and the cut rule in particular, rules out an infinitude of choices and makes the search for proof rules more goal-oriented. As we will see, the second aspect of concentrating choice, reduces the amount of branching in proof search drastically to just a few options. All together, uniform proofs provide an operational perspective on proof search and will guide us in the construction of proofs, also in $\mathbf{ND}_1$, relative to logic programs.

Before we come to the actual definition of uniform proofs, we will need to talk about specific terms and substitutions that arise in proof search. Recall that a term with no variables is called closed. In the simplest case, which will be the only one that we treat here, we can prove an existentially quantified formula $\exists x.\, \varphi$ by providing a closed term t and prove $\varphi[x := t]$. For instance, let φ be $R(\mathrm{pos}(x, \underline{3}))$ and let t be $\underline{2}$, as in example 11.5. Under the assumption of eq. (11.1), we have that $\varphi[x := t]$ holds and thus also $\exists x.\, (\mathrm{pos}(x, \underline{3}))$. Note that we substituted the closed term $\underline{2}$ for x and thereby obtained a closed formula.[1] We will restrict the proof rules ($\exists$I) and ($\forall$E) of $\mathbf{ND}_1$ to allow only substitutions that result into closed formulas, which in turn will eliminate all free variables from proofs. The following definition formalises the needed substitutions.

[1] Terminology: Closed terms and formulas are often also called ground terms and formulas.

Definition 11.6: Closing substitution

Let $X \subseteq \mathrm{Var}$ be a set of variables. An *X-closing substitution* is a substitution $\sigma\colon \mathrm{Var} \to \mathrm{Term}$, such that $\sigma(x)$ is closed for all $x \in X$.

The purpose of X-closing substitutions is to turn any formula, whose free variables are among those in X, into a closed formula.

Example 11.7

As we have $\mathrm{fv}(P(\underline{0}, x, x)) = \{x\}$, any X-closing substitution with $x \in X$ will allow use to close this formula. For instance, $s(\underline{0})$ is a closed term over the signature of example 11.4 and $[x := s(\underline{0})]$ is a $\{x\}$-closing substitution. Indeed, we obtain the closed formula $P(\underline{0}, x, x)\,[x := s(\underline{0})] = P(\underline{0}, s(\underline{0}), s(\underline{0}))$.

?

Do signatures without constants admit closed terms?

We mentioned initially that we obtain proof search by restricting the formulas φ that can appear as a goal, that is, in a sequent $\Gamma \vdash \varphi$ that we wish to prove. Out of the many choices one can make to restrict formulas, we will be using the one given in definition 11.8 below. Before we come to that definition, let us briefly think about what a reasonable set of goals, for which proofs can be searched, would be. First of all, we of course want to be able to prove facts about predicates, hence predicate atoms of the form $P(t_1, \ldots, t_n)$ should certainly be allowed as goals. Next, conjunction does certainly not pose any problem because finding a proof for $\Gamma \vdash \varphi \wedge \psi$ only requires us to find proofs for $\Gamma \vdash \varphi$ and $\Gamma \vdash \psi$ separately by the introduction rule ($\wedge$I). To be able to state and prove more interesting properties, we will also allow disjunction and existential quantification. These two connectives will make proof search more difficult. Recall that the introduction rules ($\vee \mathrm{I}_1$) and ($\vee \mathrm{I}_2$) for disjunction require us to find proofs for *either* $\Gamma \vdash \varphi$ *or* $\Gamma \vdash \psi$ in order to prove $\Gamma \vdash \varphi \vee \psi$. Thus, to find a proof for the disjunction $\varphi \vee \psi$ we have to try to prove one of the options, say φ, and if we fail to prove this option, then we try the other. Trying out these different options is not difficult, but can be costly, depending on how many proof steps we have to carry out before we find out that φ is not provable. Finally, the proof of an existential quantifier $\exists x.\, \varphi$ requires us to find a term t, such that $\varphi[x := t]$ holds. As it turns out, it is possible

to devise procedure that construct a closed term t, if it exists, automatically. Thus, we will also allow existential quantifiers in proof goals. The following definition 11.8 sums up the formulas that we allow as goals.

Definition 11.8: Goal formulas

The set Goal of all *goal formulas* φ_G is the set of $\mathcal{L}$-formulas over a signature $\mathcal{L}$ generated by the following grammar.

$$\varphi_G ::= P(t_1, \dots, t_n) \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \exists x.\, \varphi_G$$

Introduction Rules

$$\frac{\Gamma \vdash_u \varphi_1 \quad \Gamma \vdash_u \varphi_2}{\Gamma \vdash_u \varphi_1 \wedge \varphi_2}\ (\wedge\mathrm{I}) \qquad \frac{\Gamma \vdash_u \varphi[x := t] \quad t \text{ closed}}{\Gamma \vdash_u \exists x.\, \varphi}\ (\exists\mathrm{I})$$

$$\frac{\Gamma \vdash_u \varphi_1}{\Gamma \vdash_u \varphi_1 \vee \varphi_2}\ (\vee\mathrm{I}_1) \qquad \frac{\Gamma \vdash_u \varphi_2}{\Gamma \vdash_u \varphi_1 \vee \varphi_2}\ (\vee\mathrm{I}_2)$$

Backchaining Rule

$$\frac{(\forall x_1.\, \cdots \forall x_m.\, A_1 \wedge \cdots A_n \to A_0) : \Gamma \quad \Gamma \vdash_u A_1[\sigma] \ \cdots\ \Gamma \vdash_u A_n[\sigma]}{\Gamma \vdash_u A_0[\sigma]}\ (\mathrm{B})$$

where σ is an $\{x_1, \dots, x_m\}$-closing substitution.

Figure 11.1.: Rules of Uniform Proofs

Let us now come to uniform proofs, which are given by a proof system that allows us to prove goals φ (definition 11.8) from Horn clause theories Γ (definition 11.3). To distinguish sequents for uniform proofs from those of $\mathbf{ND}_1$, we will write $\Gamma \vdash_u \varphi$ for a sequent that we intent to find a uniform proof for. We have already explained the general approach to prove $\Gamma \vdash_u \varphi$ above for the case of conjunction, disjunction and existential quantification. This leads us to the introduction rules in the upper part of fig. 11.1, which are exactly the same rules as in $\mathbf{ND}_1$, just restricted to the introduction of goal formulas. Let us consider the goal φ given by $\exists u.\, \exists v.\, R(u) \wedge A(u, v) \wedge F(v)$ over the

signature from the robot path finding example 11.5. This formula says that the robot is in some position on the board that has a free adjacent position. Such positions exist, namely if we pick $p = \text{pos}(\underline{2}, \underline{3})$ for the robot position and the adjacent position $q = \text{pos}(\underline{2}, \underline{4})$. Using only the introduction rules from fig. 11.1, we can start a proof for φ:

1	$\vdots$	
2	$A(p, q)$	??
3	$F(q)$	??
4	$A(p, q) \wedge F(q)$	$\wedge$I, 2, 3
5	$R(p)$	??
6	$R(p) \wedge A(p, q) \wedge F(q)$	$\wedge$I, 5, 4
7	$\exists v.\, R(p) \wedge A(p, v) \wedge F(v)$	$\exists$I, 6
8	$\exists u.\, \exists v.\, R(u) \wedge A(u, v) \wedge F(v)$	$\exists$I, 7

However, we are not able to fill in the question marks, yet. Note that the formulas that we have to prove in lines 2, 3 and 5 are closed predicate atoms. It is easy to see that any proof attempt that starts with a goal formula and only uses the introduction rules, must eventually reach predicate atoms. Thus, we need a rule to prove such predicate atoms from a Horn clause theory. And that is the heart of logic programming: given a predicate atom, find a Horn clause that matches the atom, and continue with the premises of the Horn clause. This process is called *backchaining* and is captured by the rule (B) in fig. 11.1. The idea of the backchaining rule is that, whenever we have to prove a predicate atom, we choose a Horn clause from our logic program in such a way that the head of the clause (definition 11.3) matches the atom. If we find such a matching clause, then we continue by proving all the premises of the chosen Horn clause. Note that the premises of a Horn clause are always predicate atoms, thus we will continue using the backchaining rule until we can finish the proof by selecting a fact, which is a Horn clause without premises.

Let us use this procedure to fill in the question marks in the proof above. For the purpose of this, let us denote by Γ_1 the logic program that consists of all the Horn clauses in eqs. (11.1) to (11.3). In line 5, we have to prove $R(\text{pos}(\underline{2}, \underline{3}))$, which is exactly the fact that we assumed in (11.1). Thus, we can use the

following instance of the backchaining rule, where both n and m are 0 and thus the substitution σ is irrelevant.

$$\frac{R(\text{pos}(\underline{2}, \underline{3})) : \Gamma_1}{\Gamma_1 \vdash_u R(\text{pos}(\underline{2}, \underline{3}))} \text{ (B)}$$

The proofs for lines 2 and 3 are given analogously, which then results in the following completed uniform proof in Fitch-style.

1	Γ_1	
2	$A(p, q)$	B, 1
3	$F(q)$	B, 1
4	$A(p, q) \land F(q)$	$\land$I, 2, 3
5	$R(p)$	B, 1
6	$R(p) \land (A(p, q) \land F(q))$	$\land$I, 5, 4
7	$\exists v.\, R(p) \land (A(p, v) \land F(v))$	$\exists$I, 6
8	$\exists u.\, \exists v.\, R(u) \land (A(u, v) \land F(v))$	$\exists$I, 7

Before we continue to explain the backchaining rule for more interesting cases, let us formally define what we mean by a uniform proof.

Definition 11.9: Uniform proofs

Let Γ be a Horn clause theory and φ a closed goal formula. We say that φ can be proven from Γ by a *uniform proof*, if a proof for $\Gamma \vdash_u \varphi$ can be derived from the rules in fig. 11.1.

Note that the uniform proofs have no separate elimination rules. Instead, backchaining combines the assumption rule (Assum), the universal quantifier elimination ($\forall$E) and the implication elimination ($\to$E) into one rule. This works because the only assumptions that we can use are Horn clauses.

Example 11.10

Suppose we want to show that the robot can reach *some* position in

fig. 6.1, thus we want to find a proof for

$$\Gamma \vdash_u \exists u.\, C(\mathrm{pos}(\underline{2},\underline{3}), u)\,,$$

where C is the predicate for connected positions from example 11.5 and Γ the logic program that consists of eqs. (11.1) to (11.5). As above, we will refer to the logic program given by eqs. (11.1) to (11.3) as Γ_1. This allows us refer explicitly to the Horn clauses in eqs. (11.4) and (11.5). There are many choices for the position u, but to make the example non-trivial and not too lengthy, let us use $\mathrm{pos}(\underline{2},\underline{4})$. The uniform proof for the above sequent goes for this choice as follows.

1	Γ_1	
2	$\forall p.\, C(p,p)$	
3	$\forall p.\, \forall q.\, \forall r.\, C(q,r) \wedge A(p,q) \wedge F(q) \to C(p,r)$	
4	$C(\mathrm{pos}(\underline{2},\underline{4}), \mathrm{pos}(\underline{2},\underline{4}))$	B, 2
5	$A(\mathrm{pos}(\underline{2},\underline{3}), \mathrm{pos}(\underline{2},\underline{4}))$	B, 1
6	$F(\mathrm{pos}(\underline{2},\underline{4}))$	B, 1
7	$C(\mathrm{pos}(\underline{2},\underline{3}), \mathrm{pos}(\underline{2},\underline{4}))$	B, 3, 4, 5, 6
8	$\exists u.\, C(\mathrm{pos}(\underline{2},\underline{3}), u)$	∃I, 7

You may notice that there is another choice hidden in example 11.10: To apply the backchaining rule in line 7, we had to choose an "intermediate" position, the variable q in the transitivity rule from line 3. We chose the position $\mathrm{pos}(\underline{2},\underline{4})$ because it brought us directly to the position that we wanted to go to. However, we could have chosen to take a completely different path, for instance via $\mathrm{pos}(\underline{2},\underline{2}), \mathrm{pos}(\underline{3},\underline{2}), \mathrm{pos}(\underline{4},\underline{2}), \mathrm{pos}(\underline{4},\underline{3}), \mathrm{pos}(\underline{4},\underline{4})$ and $\mathrm{pos}(\underline{3},\underline{4})$. This is where we have to find an appropriate control mechanism to guide the search for a path. In appendix D, an implementation of example 11.5 is given in a language called Prolog, a programming language based on Horn clause theories. The control mechanism chosen there is "tabling" combined with the search for a shortest path, which gives a very efficient search strategy for path finding. We will not explain the details of this control mechanism or Prolog here. For our purposes, it suffices to say that uniform proofs provide a foundation for the reasoning steps that Prolog takes and that an algorithm can be

obtained from the uniform proof system by complementing it with an appropriate search strategy for the substitution σ in the backchaining rule.

Remark. The uniform proof system has quite a few less rules than $\mathbf{ND}_1$: all the elimination rules and the introduction rules for implication and universal quantification are not present. We have already explained why the general introduction and elimination rules for implication cause troubles. It is possible to add restricted versions of the introduction rules for implication and universal quantification [MN12; Mil+91]. However, the general elimination rules need to be treated completely differently.

Solutions

Answers to the Quizzes of Chapter 3

Answer to quiz on page 17 For each of the three variables, there are two possibilities 0 and 1. Thus, there are $2^3 = 8$ different combinations of truth values and each of them gets its own row.

Answers to the Quizzes of Chapter 4

Answer to quiz on page 45 The first proof attempt is not correct because the ($\to$I)-rule uses a hypothesis but line 2 does not open a new flag. A correct proof would look as follows.

1	p	
2	$\quad q$	
3	$\quad p$	Assum, 1
4	$q \to p$	$\to$I, 2–3

The second attempt is correct and proves the judgement $p \vdash r \to p \wedge r$.

Answers to the Quizzes of Chapter 5

Answer to quiz on page 56 First, we have

$$\begin{aligned} &\neg(\varphi_1 \wedge \cdots \wedge \varphi_n \wedge \neg\varphi) \\ &\equiv \neg\varphi_1 \vee \cdots \vee \neg\varphi_n \vee \neg\neg\varphi && \text{by De Morgan's laws} \\ &\equiv \neg\varphi_1 \vee \cdots \vee \neg\varphi_n \vee \varphi && \text{by DNE} \end{aligned}$$

and then we use $\neg\top = \bot$ to obtain the proof.

Answer to quiz on page 62 You can easily find that

$$(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \equiv (p_1 \vee p_2) \wedge (p_1 \vee q_2) \wedge (q_1 \vee p_2) \wedge (q_1 \vee q_2).$$

This formula has $2^2 = 4$ disjunctive clauses. For the other formula, you will find that you will have to have every combination of p_i and q_j in the disjunctive clauses:

$$(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee (p_3 \wedge q_3) \equiv (p_1 \vee p_2 \vee p_3) \wedge (p_1 \vee p_2 \vee q_3) \wedge \cdots \wedge (q_1 \vee q_2 \vee q_3).$$

This leads to $2^3 = 8$ disjunctive clauses and shows the exponential blow-up of the CNF.

Answers to the Quizzes of Chapter 6

Answer to quiz on page 74 In eq. (6.6), we used that $s(r, k)$ is supposedly uniquely defined for any k. But typically, path would be of finite length, say l. If k is larger than l, the map s cannot be defined for $s(r, k)$.

There are several ways to solve this. A more elegant one will be presented in chapter 11, but that approach uses a different definition of paths and indexing entirely. For our present definition, we can introduce a function symbol len, which returns the **len**gth of a path, and a predicate symbol B, where $B(k, n)$ intuitively holds if k is natural number below the **B**ound n. The formula (6.6) can then be improved to

$$\forall r.\, P(r) \to \forall k.\, B(k, \mathrm{len}(r)) \to \neg O(s(r, k))\,.$$

Answers to the Quizzes of Chapter 7

Answer to quiz on page 89 If t is any term, then it is easy to see that $t\,\eta = t$. Thus, the identity substitution is the identity for the application of terms to substitutions. Formally, this can be proven by induction over terms.

In fact, this goes even deeper. We can compose substitutions, just like we compose maps: For substitutions σ and τ, let us denote by $\tau \odot \sigma$ their composition, which is given by applying to every term $\sigma(x)$ the substitution τ:

$$(\tau \odot \sigma)(x) = \sigma(x)\,\tau$$

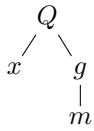
Since $\sigma(x)\,\eta = \sigma(x)$, we thus obtain $\eta \odot \sigma = \sigma$. The other way round, we also have $(\sigma \odot \eta)(x) = x\sigma = \sigma(x)$. Hence, composition with η does not change a substitution and thus the identity substitution acts like the identity map.

Answer to quiz on page 91 The reason is that in the axiom (SP), we would otherwise exclude even the identity substitution because $x \in \mathrm{var}(\eta(x))$. However, the present definition gives us $x \# \eta$ and we could prove $\varphi\,\eta = \varphi$.

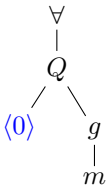
Solutions to the Exercises of Chapter 7

Solution 1

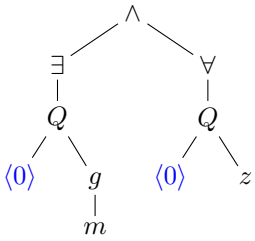
a)



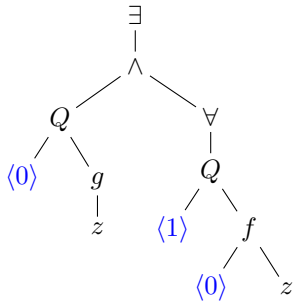
b)



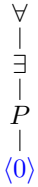
c)



d)



e)



Solution 2

a) $x\,\sigma = \sigma(x) = g(x)$

b) $f(x,y)\,\sigma = f(\sigma(x),\sigma(y)) = f(g(x),x)$

c) $(y\,\sigma)\,\sigma = x\,\sigma = g(x)$

d) $x\,(\sigma[x := x]) = (\sigma[x := x])(x) = x$

Solution 3

We have $\mathrm{fv}(\varphi) = \{y, z\}$, $\mathrm{var}(\sigma(x)) = \{x\}$, $\mathrm{var}(\sigma(y)) = \{x\}$, $\mathrm{var}(\sigma(z)) = \{y\}$, and $\mathrm{var}(\sigma(v)) = \{v\}$ for all other variables v. This gives us

a) x is not fresh for σ, since $x \in \mathrm{var}(\sigma(y))$.

b) y is not fresh for σ, since $y \in \mathrm{var}(\sigma(z))$.

c) z is fresh for σ, since $z \notin \mathrm{var}(\sigma(v))$ for any variable v.

d) x is fresh for φ, since $x \notin \mathrm{fv}(\varphi)$.

e) y is not fresh for φ, since $y \in \mathrm{fv}(\varphi)$.

f) z is not fresh for φ, since $z \in \mathrm{fv}(\varphi)$.

Solution 4

a) $Q(x, g(m))\,\sigma = Q(g(x), g(m))$

b) Note that x is not fresh for σ and we cannot directly apply carry out the substitution, even though y never appears in the formula. Instead, we do the following.

$$\begin{aligned}
(\forall x.\, Q(x, g(m)))\,\sigma &= (\forall x'.\, Q(x', g(m)))\,\sigma && \text{(EQ)}\\
&= \forall x'.\, Q(x', g(m))\,(\sigma[x' := x']) && \text{(SQ)}\\
&= \forall x'.\, Q(x', g(m)) && \text{(SP)}\\
&= \forall x.\, Q(x, g(m)) && \text{(EQ)}
\end{aligned}$$

c) Note that $\sigma[x := x] = [y := x][z := y]$, which means that this updated substitution does not affect the occurrences of x in the third step below.

$$\begin{aligned}
&((\exists x.\, Q(x, g(m))) \wedge \forall y.\, Q(y, z))\,\sigma\\
&= ((\exists x.\, Q(x, g(m))))\,\sigma \wedge (\forall y.\, Q(y, z))\,\sigma && \text{(SC)}\\
&= (\exists x.\, Q(x, g(m))) \wedge (\forall y.\, Q(y, z))\,\sigma && \text{(SQ)}\\
&= (\exists x.\, Q(x, g(m))) \wedge (\forall y'.\, Q(y', z))\,\sigma && \text{(EQ)}\\
&= (\exists x.\, Q(x, g(m))) \wedge (\forall y'.\, Q(y', z)\,\sigma) && \text{(SQ)}\\
&= (\exists x.\, Q(x, g(m))) \wedge (\forall y'.\, Q(y', y)) && \text{(SP)}
\end{aligned}$$

d)

$$
\begin{aligned}
&(\exists y.\, Q(y, g(z)) \land \forall x.\, Q(y, f(x,z)))\,\sigma \\
&= (\exists y'.\, Q(y', g(z)) \land \forall x.\, Q(y', f(x,z)))\,\sigma && \text{(EQ)} \\
&= \exists y'.\, (Q(y', g(z)) \land \forall x.\, Q(y', f(x,z)))\,\sigma && \text{(SQ)} \\
&= \exists y'.\, Q(y', g(z))\,\sigma \land (\forall x.\, Q(y', f(x,z)))\,\sigma && \text{(SC)} \\
&= \exists y'.\, Q(y', g(y)) \land (\forall x.\, Q(y', f(x,z)))\,\sigma && \text{(SP)} \\
&= \exists y'.\, Q(y', g(y)) \land (\forall x'.\, Q(y', f(x',z)))\,\sigma && \text{(EQ)} \\
&= \exists y'.\, Q(y', g(y)) \land \forall x'.\, Q(y', f(x',z))\,\sigma && \text{(SQ)} \\
&= \exists y'.\, Q(y', g(y)) \land \forall x'.\, Q(y', f(x',y)) && \text{(SP)} \\
&= \exists y'.\, Q(y', g(y)) \land \forall x.\, Q(y', f(x,y)) && \text{(EQ)}
\end{aligned}
$$

In the last step, we used that $x \# Q(y', f(x', y))$. This step is not strictly necessary, but shows that σ does not affect the binding of x because no variable is replaced with a term that contains x.

Answers to the Quizzes of Chapter 8

Answer to quiz on page 101 There are exactly two possibilities because $\mathcal{M}(P)$ must be a subset of A^0. Thus, either $\mathcal{M}(P) = \emptyset$ or $\mathcal{M}(P) = A^0$.

Answer to quiz on page 108 The formula $\forall x.\, \exists y.\, L(x,y) \land \neg I(x,y)$ translates in the language model $\mathcal{M}_l$ to "for every language U there is language V that strictly contains U: $U \subset V$." This is not true because the total language A^* is maximal. Indeed, formally we have for all valuations v and $V \subseteq A^*$ with $w = v[x \mapsto A^*]$ that

$$
[\![L(x,y) \land \neg I(x,y)]\!]_{w[y \mapsto V]} = \begin{cases} 1, & A^* \subseteq U \text{ and } A^* \neq V \\ 0, & \text{otherwise} \end{cases} = 0.
$$

This gives $[\![\exists y.\, L(x,y) \land \neg I(x,y)]\!]_w = 0$ and thus

$$
\begin{aligned}
&[\![\forall x.\, \exists y.\, L(x,y) \land \neg I(x,y)]\!]_v \\
&= \min\{[\![\exists y.\, L(x,y) \land \neg I(x,y)]\!]_{v[x \mapsto U]} \mid U \in A^*\} \\
&\leq [\![\exists y.\, L(x,y) \land \neg I(x,y)]\!]_w \\
&= 0
\end{aligned}
$$

Answers to the Quizzes of Chapter 9

Answer to quiz on page 127 To derive symmetry, we apply the replacement rule to the formula $x \doteq s$, which we refer to as φ, as follows. First, we note that $\varphi[x := t] = t \doteq s$, which means that we can prove symmetry by showing that $\varphi[x := t]$ is derivable from $s \doteq t$. Second, we note that $\varphi[x := s] = s \doteq s$, whence $\varphi[x := s]$ is provable by (Refl). Putting this together, we obtain the following derivation for symmetry of $\doteq$.

$$\dfrac{\Delta \mid \Gamma \vdash s \doteq t \qquad \dfrac{}{\Delta \mid \Gamma \vdash \varphi[x := s]}\,(\text{Refl})}{\Delta \mid \Gamma \vdash \varphi[x := t]}\,(\text{Repl})$$

Solutions to the Exercises of Chapter 9

Solution 5

We use the constant p to represent "Pavel" and the predicate symbol O to represent "owes money to". The sentence is the formalised by the formula

$$\forall x.\, x \not\doteq p \leftrightarrow O(p, x)$$

The proof of $\varphi \vdash \neg O(p, p)$ goes as follows.

1	$\forall x.\, x \not\doteq p \leftrightarrow O(p, x)$	
2	$\quad O(p, p)$	
3	$\quad p \not\doteq p \leftrightarrow O(p, p)$	$\forall$E, 1
4	$\quad O(p, p) \rightarrow p \not\doteq p$	$\wedge$E, 3
5	$\quad p \not\doteq p$	$\rightarrow$E, 4, 2
6	$\quad p \doteq p$	Refl
7	$\quad \bot$	$\neg$E, 5, 6
8	$\neg O(p, p)$	$\neg$I, 2–7

Solution 6

We prove each direction separately, starting with the sequent

$$\vdash (\exists!x.\, P(x)) \to ((\exists x.\, P(x)) \wedge P_1)\,.$$

1		$\exists!x.\, P(x)$	
2		x | $P(x)$	
3		$\forall y.\, P(y) \to x \doteq y$	
4		$(\exists x.\, P(x))$	$\exists$I, 2
5		y | z |	
6		$P(y) \wedge P(z)$	
7		$P(y)$	$\wedge$E, 6
8		$P(y) \to x \doteq y$	$\forall$E, 3
9		$x \doteq y$	$\to$E, 8, 7
10		$P(z)$	$\wedge$E, 6
11		$P(z) \to x \doteq z$	$\forall$E, 3
12		$x \doteq z$	$\to$E, 11, 10
13		$y \doteq x$	Sym, 9
14		$y \doteq z$	Trans, 13, 12
15		$P(y) \wedge P(z) \to y \doteq z$	$\to$I, 6–14
16		$\forall z.\, P(y) \wedge P(z) \to y \doteq z$	$\forall$I, 5–15
17		$\forall y.\, \forall z.\, P(y) \wedge P(z) \to y \doteq z$	$\forall$I, 5–16
18		$(\exists x.\, P(x)) \wedge P_1$	$\wedge$I, 4, 17
19		$(\exists x.\, P(x)) \wedge P_1$	$\exists!$E, 2–18
20	$(\exists!x.\, P(x)) \to ((\exists x.\, P(x)) \wedge P_1)$		$\to$I, 1–19

Next, we prove $\vdash ((\exists x.\, P(x)) \wedge P_1) \to (\exists! x.\, P(x))$:

1	$(\exists x.\, P(x)) \wedge P_1$	
2	$(\exists x.\, P(x))$	$\wedge$E, 1
3	P_1	$\wedge$E, 1
4	$x \mid P(x)$	
5	$y \mid P(y)$	
6	$\forall z.\, P(x) \wedge P(z) \to x \doteq z$	$\forall$E, 3
7	$P(x) \wedge P(y) \to x \doteq y$	$\forall$E, 6
8	$P(x) \wedge P(y)$	$\wedge$I, 4, 5
9	$x \doteq y$	$\to$E, 7, 8
10	$\exists! x.\, P(x)$	$\exists!$I, 4, 5–9
11	$\exists! x.\, P(x)$	$\exists$E, 2, 4–10
12	$((\exists x.\, P(x)) \wedge P_1) \to (\exists! x.\, P(x))$	$\to$I, 1–11

Putting both proofs together, we get $\vdash (\exists! x.\, P(x)) \leftrightarrow ((\exists x.\, P(x)) \wedge P_1)$ by

1	$(\exists! x.\, P(x)) \to ((\exists x.\, P(x)) \wedge P_1)$	
2	$((\exists x.\, P(x)) \wedge P_1) \to (\exists! x.\, P(x))$	
3	$(\exists! x.\, P(x)) \leftrightarrow ((\exists x.\, P(x)) \wedge P_1)$	$\wedge$I, 1, 2

Solution 7

a) The formulas are

$$\varphi_r = x \doteq y$$
$$\varphi_t = \exists z.\, E(x, z) \wedge R(z, y)$$

b) The derivation of $\varphi_{R,1}, \varphi_{R,2}, \varphi_{R,3} \vdash \varphi_R$ goes as follows.

1	$\forall x.\, \forall y.\, R(x,y) \to \varphi_r \vee \varphi_t$							
2	$\forall x.\, \forall y.\, \varphi_r \to R(x,y)$							
3	$\forall x.\, \forall y.\, \varphi_t \to R(x,y)$							
4	x	y						
5			$R(x,y)$					
6			$\forall y.\, R(x,y) \to \varphi_r \vee \varphi_t$					$\forall$E, 1
7			$R(x,y) \to \varphi_r \vee \varphi_t$					$\forall$E, 6
8			$\varphi_r \vee \varphi_t$					$\to$E, 7, 5
9			$R(x,y) \to \varphi_r \vee \varphi_t$					$\to$I, 5–8
10			$\varphi_r \vee \varphi_t$					
11				φ_r				
12				$\forall y.\, \varphi_r \to R(x,y)$				$\forall$E, 2
13				$\varphi_r \to R(x,y)$				$\forall$E, 12
14				$R(x,y)$				$\to$E, 13, 11
15				φ_t				
16				$\forall y.\, \varphi_t \to R(x,y)$				$\forall$E, 3
17				$\varphi_t \to R(x,y)$				$\forall$E, 16
18				$R(x,y)$				$\to$E, 17, 15
19			$R(x,y)$					$\vee$E, 10, 11–14, 15–18
20			$\varphi_r \vee \varphi_t \to R(x,y)$					$\to$I, 10–19
21			$R(x,y) \leftrightarrow \varphi_r \vee \varphi_t$					$\wedge$I, 9, 20
22		$\forall y.\, R(x,y) \leftrightarrow \varphi_r \vee \varphi_t$						$\forall$I, 4–21
23	$\forall x.\, \forall y.\, R(x,y) \leftrightarrow \varphi_r \vee \varphi_t$							$\forall$I, 4–22

c) The graph G is described by the following 5 formulas.

$$\varphi_{E,1} = E(n_1, n_2) \qquad \varphi_{E,4} = E(n_3, n_1)$$
$$\varphi_{E,2} = E(n_2, n_2) \qquad \varphi_{E,5} = E(n_4, n_2)$$
$$\varphi_{E,3} = E(n_2, n_3)$$

d) To derive $\varphi_R, \varphi_{E,1}, \dots, \varphi_{E,5} \vdash R(n_4, n_3)$, we first derive as follows the sequent $\varphi_{R,1}, \dots, \varphi_{R,3}, \varphi_{E,1}, \dots, \varphi_{E,5} \vdash R(n_4, n_3)$.

1	$\forall x.\, \forall y.\, R(x, y) \to \varphi_r \vee \varphi_t$	
2	$\forall x.\, \forall y.\, \varphi_r \to R(x, y)$	
3	$\forall x.\, \forall y.\, \varphi_t \to R(x, y)$	
4	$\varphi_{E,1}, \dots, \varphi_{E,5}$	
5	$n_3 \doteq n_3$	Refl
6	$\forall y.\, n_3 \doteq y \to R(n_3, y)$	$\forall$E, 2
7	$n_3 \doteq n_3 \to R(n_3, n_3)$	$\forall$E, 6
8	$R(n_3, n_3)$	$\to$E, 7, 5
9	$E(n_2, n_3)$	Assum, 4
10	$E(n_2, n_3) \wedge R(n_3, n_3)$	$\wedge$I, 9, 8
11	$\exists z.\, E(n_2, z) \wedge R(z, n_3)$	$\exists$I, 10
12	$\forall y.\, \varphi_t[x := n_2] \to R(n_2, y)$	$\forall$E, 2
13	$\varphi_t[x := n_2][y := n_3] \to R(n_2, n_3)$	$\forall$E, 12
14	$R(n_2, n_3)$	$\to$E, 13, 11
15	$E(n_4, n_2)$	Assum, 4
16	$E(n_4, n_2) \wedge R(n_2, n_3)$	$\wedge$I, 15, 14
17	$\exists z.\, E(n_4, z) \wedge R(z, n_3)$	$\exists$I, 16
18	$\forall y.\, \varphi_t[x := n_4] \to R(n_4, y)$	$\forall$E, 2
19	$\varphi_t[x := n_4][y := n_3] \to R(n_4, n_3)$	$\forall$E, 18
20	$R(n_4, n_3)$	$\to$E, 19, 17

Using b) and the cut rule, we obtain $\varphi_R, \varphi_{E,1}, \dots, \varphi_{E,5} \vdash R(n_4, n_3)$ from this proof.

Solution 8

a) The formulas for associativity and the inverse are given by

$$\varphi_2 = \forall x.\, \forall y.\, \forall z.\, x \dot{+} (y \dot{+} z) \doteq (x \dot{+} y)$$
$$\varphi_3 = \forall x.\, \exists y.\, x \dot{+} y = \dot{0} \wedge y \dot{+} x = \dot{0}$$

b) Uniqueness can be expressed in several ways. The quickest is to replace in φ_3 the existential by the uniqueness quantifier:

$$\varphi_u = \forall x.\, \exists! y.\, x \dot{+} y = \dot{0} \wedge y \dot{+} x = \dot{0}$$

c) Before we give the formal proof that inverses are unique, let us first prove uniqueness by standard equational reasoning. Assume that we have inverses and y and z, then the following chain of equations shows that $y = z$, where we indicate the used assumption and proof rules on the right.

$$\begin{aligned}
y &\doteq y \dot{+} \dot{0} && \text{by } \varphi_{1,r} \\
&\doteq y \dot{+} (x \dot{+} z) && \text{by (Repl) and } x \text{ being an inverse} \\
&\doteq (y \dot{+} x) \dot{+} z && \text{by } \varphi_2 \\
&\doteq \dot{0} \dot{+} z && \text{by (Repl) and } y \text{ being an inverse} \\
&\doteq z && \text{by } \varphi_{1,l}
\end{aligned}$$

Let us formalise first this equational proof in $\mathbf{ND}_1^=$ by deriving the sequent

$$x, y, z \mid \varphi_{1,l}, \varphi_{1,r}, \varphi_2, y \dot{+} x \doteq \dot{0}, x \dot{+} z \doteq \dot{0} \vdash y \doteq z \qquad (*)$$

The proof goes as follows.

1	x, y, z	$\varphi_{1,l}$	
2		$\varphi_{1,r}$	
3		φ_2	
4		$y \dot{+} x \doteq \dot{0}$	
5		$x \dot{+} z \doteq \dot{0}$	
6		$y \dot{+} \dot{0} \doteq y$	$\forall$E, 2
7		$y \doteq y \dot{+} \dot{0}$	Sym, 6
8		$\dot{0} \doteq x \dot{+} z$	Sym, 5
9		$y \doteq y \dot{+} (x \dot{+} z)$	Repl, 8, 7
10		$\forall u. \forall z. y \dot{+} (u \dot{+} z) \doteq (y \dot{+} u) \dot{+} z$	$\forall$E, 3
11		$\forall z. y \doteq y \dot{+} (x \dot{+} z) \doteq (y \dot{+} x) \dot{+} z$	$\forall$E, 10
12		$y \dot{+} (x \dot{+} z) \doteq (y \dot{+} x) \dot{+} z$	$\forall$E, 11
13		$y \dot{+} (x \dot{+} z) \doteq 0 \dot{+} z$	Repl, 4, 12
14		$\dot{0} \dot{+} z \doteq z$	$\forall$E, 1
15		$y \dot{+} (x \dot{+} z) \doteq z$	Trans, 13, 14
16		$y \doteq z$	Trans, 9, 15

The following now formalises the proof that the group axioms imply the unique

existence of the inverse, that is, it shows $\varphi_{1,l}, \varphi_{1,r}, \varphi_2, \varphi_3 \vdash \varphi_u$.

1	$\varphi_{1,l}$	
2	$\varphi_{1,r}$	
3	φ_2	
4	φ_3	
5	x |	
6	$\exists y.\, x \dot{+} y \doteq \dot{0} \wedge y \dot{+} x \doteq \dot{0}$	$\forall$E, 4
7	y | $x \dot{+} y \doteq \dot{0} \wedge y \dot{+} x \doteq \dot{0}$	
8	$y \dot{+} x \doteq \dot{0}$	$\wedge$E, 7
9	z | $x \dot{+} z \doteq \dot{0} \wedge z \dot{+} x \doteq \dot{0}$	
10	$x \dot{+} z \doteq \dot{0}$	$\wedge$E, 9
11	$y \doteq z$	$(*)$, 8, 10
12	$\exists! y.\, x \dot{+} y \doteq \dot{0} \wedge y \dot{+} x \doteq \dot{0}$	$\exists!$I, 7, 9–11
13	$\exists! y.\, x \dot{+} y \doteq \dot{0} \wedge y \dot{+} x \doteq \dot{0}$	$\exists$E, 6, 7–12
14	$\forall x. \exists! y.\, x \dot{+} y \doteq \dot{0} \wedge y \dot{+} x \doteq \dot{0}$	$\forall$I, 5–13

Answers to the Quizzes of Chapter 11

Answer to quiz on page 147 The idea is that multiplication is given recursively by $0 \cdot n = 0$ and $(m+1) \cdot n = (m \cdot n) + n$. This can be translated into the following two Horn clauses.

$$\forall n. \quad M(0, n, 0)$$
$$\forall m. \forall n. \forall i. \forall k.\, M(m, n, i) \wedge P(i, n, k) \to M(s(m), n, k)$$

Answer to quiz on page 152 No, if a signature $\mathcal{L}$ has no constants, then it is not possible to obtain an $\mathcal{L}$-term without variables. For example, suppose $\mathcal{L}$ is a signature with only one unary function symbol f. Then the only terms we can build over this signature are of the form $x, f(x), f(f(x)), \ldots$ for variables

x. Thus, it is not possible to obtain a closed term without variables from $\mathcal{L}$. This can formally be proven by induction over first-order terms.

Bibliography

[And02] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. 2nd ed. Applied Logic Series. Springer Netherlands, 2002. ISBN: 978-1-4020-0763-7. DOI: 10.1007/978-94-015-9934-4. (Visited on 10/05/2020).

[Ane12] Irving H. Anellis. 'Peirce's Truth-functional Analysis and the Origin of the Truth Table'. In: *History and Philosophy of Logic* 33.1 (2012), pp. 87–97. ISSN: 0144-5340. DOI: 10.1080/01445340.2011.621702.

[Gal87] Jean H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Wiley, 1987. 511 pp. ISBN: ISBN 978-0-471-61546-0.

[Gan04] Jonardon Ganeri. 'Indian Logic'. In: *Handbook of the History of Logic*. Ed. by Dov M. Gabbay and John Woods. Vol. 1. North-Holland, 2004, pp. 309–395. ISBN: 1874-5857. DOI: 10.1016/S1874-5857(04)80007-4.

[Gen35] Gerhard Gentzen. 'Untersuchungen Über Das Logische Schließen. I'. In: *Mathematische Zeitschrift* 39.1 (1935), pp. 176–210. ISSN: 1432-1823. DOI: 10.1007/BF01201353.

[HR04] Michael Huth and Mark Dermot Ryan. *Logic in Computer Science - Modelling and Reasoning about Systems*. 2nd ed. Cambridge University Press, 2004, p. 427. I pp.

[Jac99] Bart Jacobs. *Categorical Logic and Type Theory*. Studies in Logic and the Foundations of Mathematics 141. Amsterdam: North Holland, 1999.

[Kle74] Stephen C. Kleene. *Introduction to Metamathematics*. 7. repr. Bibliotheca Mathematica. Groningen: Wolters-Noordhoff and North-Holland, 1974. ISBN: 978-0-444-10088-7.

[Klo92] Jan Willem Klop. *Term Rewriting Systems*. 1992.

[Kow79] Robert A. Kowalski. 'Algorithm = Logic + Control'. In: *Commun. ACM* 22.7 (1979), pp. 424–436. DOI: 10.1145/359131.359136.

[MN12] Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, 2012. 306 pp. ISBN: ISBN 978-0-521-87940-8. URL: http://www.cambridge.org/de/academic/subjects/computer-science/programming-languages-and-applied-logic/programming-higher-order-logic?format=HB.

[Mil+91] Dale Miller, Gopalan Nadathur, Frank Pfenning and Andre Scedrov. 'Uniform Proofs as a Foundation for Logic Programming'. In: *Ann. Pure Appl. Logic* 51.1-2 (1991), pp. 125–157. DOI: 10.1016/0168-0072(91)90068-W.

[PD01] Frank Pfenning and Rowan Davies. 'A Judgmental Reconstruction of Modal Logic'. In: *Math. Struct. Comput. Sci.* 11.4 (2001), pp. 511–540. DOI: 10.1017/S0960129501003322.

[Pra06] Dag Prawitz. *Natural Deduction: A Proof-Theoretical Study*. Dover Books on Mathematics. Dover Publications, 2006. ISBN: 978-0-486-44655-4.

[PTW18] Graham Priest, Koji Tanaka and Zach Weber. 'Paraconsistent Logic'. In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Summer 2018. Metaphysics Research Lab, Stanford University, 2018. URL: https://plato.stanford.edu/archives/sum2018/entries/logic-paraconsistent/.

[Sol13] Daniel Solow. *How to Read and Do Proofs: An Introduction to Mathematical Thought Processes*. 6th edition. Hoboken, New Jersey: Wiley, 2013. 336 pp. ISBN: 978-1-118-16402-0.

[TS00] Anne Sjerp Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. 2nd ed. Cambridge Tracts in Theoretical Computer Science. Cambridge: Cambridge University Press, 2000. ISBN: 0-521-77911-1.

[TvD88] Anne Sjerp Troelstra and Dirk van Dalen. *Constructivism in Mathematics: An Introduction*. Vol. 1 & 2. Studies in Logic and the Foundations of Mathematics 121 & 123. North-Holland, 1988. 384 pp. ISBN: 978-0-444-70266-1.

A. Greek Letters

Letter	Name	LaTeX command
α A	alpha	`\alpha \Alpha`
β B	beta	`\beta \Beta`
γ Γ	gamma	`\gamma \Gamma`
δ Δ	delta	`\delta \Delta`
ϵ E	epsilon	`\epsilon \Epsilon`
ζ Z	zeta	`\zeta \Zeta`
θ Θ	theta	`\theta \Theta`
ι I	iota	`\iota \Iota`
κ K	kappa	`\kappa \Kappa`
λ Λ	lambda	`\lambda \Lambda`
μ M	mu	`\mu \Mu`
ν N	nu	`\nu \Nu`
ξ Ξ	xi	`\xi \Xi`
o O	omicron	`\omicron \Omicron`
π Π	pi	`\pi \Pi`
ρ P	rho	`\rho \Rho`
σ Σ	sigma	`\sigma \Sigma`
τ T	tau	`\tau \Tau`
υ Υ	upsilon	`\upsilon \Upsilon`
φ Φ	phi	`\varphi \Phi`
χ X	chi	`\chi \Chi`
ψ Ψ	psi	`\psi \Psi`
ω Ω	omega	`\omega \Omega`

Table A.1.: Greek Letters and Their Pronunciation

B. Tools

B.1. Sets and Maps

Given sets A and B, we denote by $A \times B$ their **product** consisting of ordered pairs

$$A \times B = \{(a, b) \mid a \in A, b \in B\}$$

and B^A the **set of maps** $A \to B$. Given two maps $f\colon A \to B$ and $g\colon B \to C$, we denote by $g \circ f$ the **composition** of g and f with $(g \circ f)(x) = g(f(x))$. The composition can be pronounced as "g composed with f" or "g after f".

We will also be using the **powerset** of sets A, denoted by $\mathcal{P}(A)$, which is the set of all subsets of A:

$$\mathcal{P}(A) = \{U \mid U \subseteq A\}$$

The **empty set** is denoted by $\emptyset$.

The set of natural numbers starting at 0 is denoted by $\mathbb{N}$. Given a natural number $n \in \mathbb{N}$, we write $[n]$ for the **set of the first n natural numbers** :[1]

$$[n] = \{k \in \mathbb{N} \mid k < n\}\,.$$

In particular, we have $[0] = \emptyset$, $[1] = \{0\}$ etc. Generally, $[n]$ has exactly n elements.

Exercise 9

Show for any set A that the sets $A^{[2]}$ and $A \times A$ are isomorphic, that is, there is a map $A^{[2]} \to A \times A$ with an inverse.

[1] Warning: In some contexts, $[n]$ includes also n and has thus $n+1$ elements. For our purposes, the present interpretation is the most useful though.

Exercise 10

Show that there is exactly one map $\emptyset \to X$ for any set X.

Exercise 11

Suppose that X is a finite set with m elements and let $n \in \mathbb{N}$. Show that $X^{[n]}$ has m^n elements.

B.2. Induction on Natural Numbers

We denote by $\mathbb{N}$ the set of **natural numbers** starting at 0. But what is this set exactly? Naively, one could define it by saying $\mathbb{N} = \{0, 1, 2, ...\}$. However, making precise the meaning of the dots is quite difficult. A better way is to define the natural numbers as some set with a certain property, as follows.

Definition B.1: Natural numbers and iteration principle

The natural numbers are a set $\mathbb{N}$ with an element $0 \in \mathbb{N}$ and a map $\mathrm{suc}\colon \mathbb{N} \to \mathbb{N}$, such that for any set A with an element a_0 and a map $f\colon A \to A$, there is a unique map $g\colon \mathbb{N} \to A$ such that $g(0) = a_0$ and $g(\mathrm{suc}(n)) = f(g(n))$ for all $n \in \mathbb{N}$. We say that g is defined by iteration of (a_0, f).

To make our life simple, we will write $n+1$ instead of $\mathrm{suc}(n)$ for the successor of the number n. We can the express any natural number by $1 = 0 + 1$, $2 = 1+1, 3 = 2+1$ and so forth. When we wrote $\mathbb{N} = \{0, 1, 2, ...\}$ above, we intuitively understood that $\mathbb{N}$ should consist of exactly the numbers expressed in this way and that there should be no spurious elements like catfish $\in \mathbb{N}$. That this is the case is expressed by the iteration property because it tells us that every element in $\mathbb{N}$ is either of the form 0 or $n + 1$ for some $n \in \mathbb{N}$. This principle allows us to count from any number down to 0. One can say that

elements of $\mathbb{N}$ are *static* representations of numbers, while iteration reflect the *dynamics* of counting.

If we consider a pair (a_0, f) given as in definition B.1, then one can compare this to the following imperative program, in which a for-loop is used to repeatedly apply f to the initial value a_0.

```
def g(n):
  res = a_0
  for i in [1, ..., n]:
    res = f(res)
  return res
```

Using the iteration principle can be a nuisance because we have to explicitly specify the pair (a_0, f), which only gets worse for the more complicated structures that we encounter in, for example, chapter 2. For instance, suppose that we wish to define exponentiation in terms of multiplication. Our intuition would be to proceed with the following definition.

$$a^0 = 1 \quad \text{and} \quad a^{n+1} = a^n \cdot a \tag{B.1}$$

This definition arises by applying the iteration principle to the pair $(1, f_a)$ with $f_a(x) = x \cdot a$. However, defining exponentiation via the formal iteration principle is clearly inconvenient, and we will use the equational style in eq. (B.1) whenever possible and *we are sure that the equations can be reduced to the iteration principle*. Such a reduction is not always easy. For instance, the factorial function can be expressed by the following equations.

$$0! = 1 \quad \text{and} \quad (n+1)! = n! \cdot (n+1) \tag{B.2}$$

If we tried to use the iteration principle directly to define ! as map $\mathbb{N} \to \mathbb{N}$ via iteration, then we run into the problem that $n + 1$ is not available for the multiplication. Instead, we have to define first a map $\mathrm{fac} \colon \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ by iteration from $(1, 1)$ and $f(k, r) = (k + 1, r \cdot k)$. We then find that $\mathrm{fac}(n) = (n + 1, n!)$. This shows that the factorial function can be obtained by iteration but this requires considerable overhead. Therefore, we typically prefer specifications like in eq. (B.2).

The iteration principle the natural numbers can be used to define most maps on the natural numbers that we may be interested in.[2] From the iteration principle, we can also derive the usual induction principle.[3]

[2] Defining exact set of definable maps requires one to set up a formal theory, like primitive recursion, Peano arithmetic or Gödel's system T.

[3] We do not specify formally at this point what a property is, but we can think of subsets $P \subseteq \mathbb{N}$.

Theorem B.2: Induction principle for $\mathbb{N}$

Let P be a property of $\mathbb{N}$, where we write $P(n)$ for the property P at $n \in \mathbb{N}$. To prove that $P(n)$ holds for all $n \in \mathbb{N}$, it suffices to show that

- $P(0)$ holds (**base case**), and
- for all $n \in \mathbb{N}$, assuming that $P(n)$ holds, that $P(n+1)$ holds (**induction step**).

Proof. The proof of the induction principle needs a concrete definition of what a property is. For simplicity, we assume that a property is a subset, that is, $P \subseteq \mathbb{N}$ where $P(n)$ means that $n \in P$. Our goal is then to show that all natural numbers are contained in P. In turn, we obtain this by iterating $(0, \mathrm{suc})$ to get a map $g\colon \mathbb{N} \to P$ because 0 is in P and for any $n \in P$, also $n+1 \in P$. Since g is uniquely defined by $g(0) = 0$ and $g(n+1) = g(n) + 1$, we obtain that $\mathbb{N} \subseteq P$, as required. □

The assumption $P(n)$ in the induction step is called the **induction hypothesis**, and it should be noted that it is only assumed for *each individual* n. Sometimes, the induction principle is mistakenly stated with the induction hypothesis separated as follows.

Incorrect "induction principle": To prove that $P(n)$ holds for all $n \in \mathbb{N}$,

1. prove that $P(0)$ holds,
2. assume that $P(n)$ holds for all $n \in \mathbb{N}$, and
3. prove that $P(n+1)$ holds for all $n \in \mathbb{N}$.

This, however, is incorrect because the second point implies immediately the first and third, thereby allowing us to prove that any property! Clearly, we do not want this and theorem B.2 is the correct principle. In chapter 6, we will see how first order logic allows us to resolve such ambiguities of natural language.

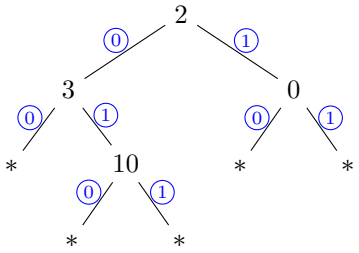


Figure B.1.: Example of a binary tree labelled in $\mathbb{N}$. The root is labelled with the number 2 and has two children. The circled blue numbers indicate the number of the outgoing branching.

B.3. Trees and Induction

There are various kinds of trees: binary trees, which have two children below every node; lists, which have just one child at any node; trees with arbitrary branching, where every node may have an arbitrary number of children. Besides the branching, trees usually have labels. For instance, in a list every node is labelled with the corresponding list element. The aim of this section is to give a general account of labelled trees.

We begin by characterising the labels and branching a tree may have.

Definition B.3

We call a pair (A, B) a **branching type** if A is a set and B an A-indexed family of sets, that is, for every $a \in A$ we are given a set B_a.

Note that the terminology of "branching type" also tacitly includes labels. The intuition of this definition is that a tree of type (A, B) will be labelled in A and will have at a node with label $a \in A$ one branch for every element in B_a.

Example B.4

Binary trees (not balanced!) labelled in $\mathbb{N}$ are induced by the branching type $(\mathbb{N} \cup \{*\}, B)$ with $B_n = [2]$ and $B_* = \emptyset$. The idea is that an inner node can be labelled with a number, while a leaf is labelled with $*$. An inner node has then exactly two children and a leaf has none.

Figure B.1 shows an example of a binary tree, which we wish to capture with the branching type given in example B.4. The following definition shows how general trees can be constructed and what their defining property is.

Definition B.5: Trees as inductive structures

Trees with branching type (A, B) are given by a set $\mathcal{T}$, or $\mathcal{T}(A, B)$, together with a family tr of maps $\mathrm{tr}_a \colon \mathcal{T}^{B_a} \to \mathcal{T}$ indexed by elements $a \in A$, such that the following iteration principle is fulfilled: for any set X and family f of maps with $f_a \colon X^{B_a} \to X$, there is a *unique* map $\bar{f} \colon \mathcal{T} \to X$ with

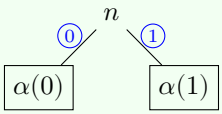
$$\bar{f}(\mathrm{tr}_a(\alpha)) = f_a(g \circ \alpha)$$

for all $a \in A$ and $\alpha \colon B_a \to \mathcal{T}$. We say that $\bar{f}$ is defined by **iteration** from f or that $\bar{f}$ is the **inductive extension** of f.

This definition packs a lot. Let us unfold it in the case of binary trees.

Example B.6: Binary trees

Let us write $\mathcal{B} = \mathcal{T}(\mathbb{N} \cup \{*\}, B)$ for a set of trees for the branching type that we introduced in example B.4. This set comes with a family tr of maps indexed by $\mathbb{N} \cup \{*\}$, that is, we get for every $n \in \mathbb{N}$ a map $\mathrm{tr}_n \colon \mathcal{B}^{[2]} \to \mathcal{B}$ and one map $\mathrm{tr}_* \colon \mathcal{B}^{\emptyset} \to \mathcal{B}$. From the exercises in appendix B.1, we know that giving a map α in $\mathcal{B}^{[2]}$ amounts to giving a pair of elements in $\mathcal{B}$. Thus, we can represent for such an α the resulting tree $tr_n(\alpha)$ like so, where a box indicates a whole subtree:



Let us denote by ℓ the tree $\mathrm{tr}_*(\varepsilon)$, where ε is the only element of $\mathcal{B}^{\emptyset}$, see exercise 10. This tree represents a leaf in a binary tree. The tree from fig. B.1 can then be represented in $\mathcal{B}$ by

$$t = \mathrm{tr}_2(\mathrm{tr}_3(\ell, \mathrm{tr}_{10}(\ell, \ell)), \mathrm{tr}_0(\ell, \ell))$$

where we use the pair notation to create elements of $\mathcal{B}^{[2]}$.
So much for the construction of tree. The interesting part is what we can do with them though. This is where the iteration principle comes in, which allows us to traverse a tree. For instance, we can sum up all

the labels in a tree by using the family s given by

$$\begin{aligned} s_n &\colon \mathbb{N}^{[2]} \to \mathbb{N} & s_* &\colon \mathbb{N}^{\emptyset} \to \mathbb{N} \\ s_n(\alpha) &= n + \alpha(0) + \alpha(1) & s_*(\alpha) &= 0 \end{aligned}$$

this gives us a map $\bar{s} \colon \mathcal{B} \to \mathbb{N}$. Running this map on the tree t from above, we have get the following.

$$\begin{aligned} \bar{s}(t) &= s_2(\bar{s}(\mathrm{tr}_3(\ell, \mathrm{tr}_{10}(\ell, \ell))), \bar{s}(\mathrm{tr}_0(\ell, \ell))) \\ &= 2 + \bar{s}(\mathrm{tr}_3(\ell, \mathrm{tr}_{10}(\ell, \ell))) + \bar{s}(\mathrm{tr}_0(\ell, \ell)) \\ &= 2 + (3 + \bar{s}(\ell) + \bar{s}(\mathrm{tr}_{10}(\ell, \ell))) + (0 + \bar{s}(\ell) + \bar{s}(\ell)) \\ &= 2 + (3 + 0 + (10 + \bar{s}(\ell) + \bar{s}(\ell))) + (0 + 0 + 0) \\ &= 2 + (3 + 0 + (10 + 0 + 0)) + 0 \\ &= 2 + (3 + 0 + 10) + 0 \\ &= 2 + 13 + 0 \\ &= 15 \end{aligned}$$

Thus, we have traversed the trees depth-first and summed up all the intermediate results.

Exercise 12

Define a map $c \colon \mathcal{B} \to \mathbb{N}$ that counts the number of labelled nodes in a binary tree by using the iteration principle.

Exercise 13

Define by iteration a map $h \colon \mathcal{B} \to \mathbb{N}$ that computes that height of a tree, where the leaves should have height 0 and labelled nodes height at least 1.

So far, have used the iteration principle only to define maps but not to prove anything. Just like for the natural numbers, we can also obtain an induction principle.

Theorem B.7: Tree Induction

Let $\mathcal{T}$ be a set of tree with branching type (A, B) and P a property of $\mathcal{T}$, that is $P \subseteq \mathcal{T}$. If for all $a \in A$ and $\alpha \colon B_a \to P$ we have $\mathrm{tr}_a(\alpha) \in P$, then P holds for all trees in $\mathcal{T}$ (i.e., $P = \mathcal{T}$).

Exercise 14

Use the tree induction principle to prove that $c(t) \leq 2^{h(t)}$ for all $t \in \mathcal{B}$.

B.4. Formal Languages

Recall that A^* denotes the set of *words* over an alphabet A. Concretely, the set of words is given by

$$A^* = \{\varepsilon\} \cup \{a_0 a_1 \cdots a_n \mid n \in \mathbb{N}, a_k \in A\},$$

where ε is the empty word. For instance, if $A = \{a, b\}$, then A^* contains the singleton words a and b, and longer words like $abbaa$. The set of *languages* over A is the powerset $\mathcal{P}(A^*)$, that is, the set of all subsets of A^*. Given two words $v, w \in A^*$, we denote by vw or $v +\!\!+ w$ the **concatenation** of the words v and w, that is, considering v and w as one word by reading their letters in order.

$$(v_0 \cdots v_n) +\!\!+ (w_0 \cdots w_m) = v_0 \cdots v_n w_0 \cdots w_m$$

We will often make use of **context-free grammars**, which generate languages. These grammars will be notated in so-called **Backus-Naur form**. Let us start with an example.

Example B.8

Suppose we want to define a language of arithmetic expressions, in which one can use numbers in $\mathbb{N}$, addition and negation. In this case, we would say that such expressions e are generated by the following grammar.

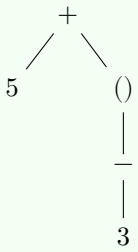
$$e ::= n\,, n \in \mathbb{N} \mid e + e \mid -e \mid (e) \tag{B.3}$$

This grammar can be read as follows. The symbol $::=$ says that an

expression e can be generated by using any of the four options on the right-hand side, where the options are separated by the vertical bar. The first option is that e can be any natural number n, which is the only case where we can start to generate an expression. To generate larger expressions, we have to use any of the other two options. For instance, if we have generated already expressions e_1 and e_2, then the second option allows us to generate the expression "$e_1 + e_2$", the third option gives us "$-e_1$" and the fourth introduces parentheses "(e_1)". It is important to realise that "$+$" and "$-$" have no meaning, they are just syntax. In the terminology of (context-free) grammars, "(", ")", "$+$", "$-$" and "n" are called **terminal symbols**, while e in the grammar is called a **non-terminal symbol**.
We can now define the language generated by the grammar, call it L, as a subset of all words over the alphabet $A = \mathbb{N} \cup \{+, -, (,)\}$ by

$$E = \{e \in A^* \mid \text{ generated by eq. (B.3)}\}\,.$$

But what does "generated by" mean exactly? We can think of eq. (B.3) as a way of describing trees of a certain shape. For instance, the expression $5 + (-3)$ can be seen as a linear, textual description of the following tree.



In appendix B.3, we have already seen how to describe such trees. The labels are numbers and the operators, that is, we put

$$L = \mathbb{N} \cup \{+, -, ()\}$$

and the branching width B is given by

$$B_n = \emptyset \qquad B_+ = [2] \qquad B_- = [1] \qquad B_{()} = [1]\,,$$

which indicates that the numbers are leaves, "$+$" has two children, and "$-$" and "$()$" have one. An expression can be seen as a tree of branching

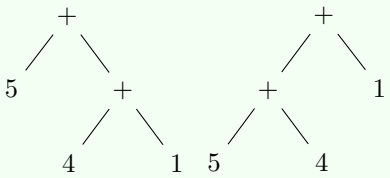
type (L, B). To get a language, we define a map flat$\colon \mathcal{T}(L, B) \to A^*$ by iteration of the family $\{f_x\}_{x \in L}\colon (A^*)^{B_x} \to A^*$ defined by

$$\begin{aligned} f_n(\alpha) &= n \\ f_+(\alpha) &= \alpha(0) +\!\!+ \text{"+"} +\!\!+ \alpha(1) \\ f_+(\alpha) &= \text{"}-\text{"} +\!\!+ \alpha(0) \\ f_{()}(\alpha) &= \text{"("} +\!\!+ \alpha(0) +\!\!+ \text{")"} \end{aligned}$$

This gives us that expressions are given as the image of the map flat:

$$E = \{\text{flat}(t) \mid t \in \mathcal{T}(L, B)\}$$

There is something peculiar about trees compared to expressions: The latter need parentheses to disambiguate, as we do not know how to generate the word $5 + 4 + 1$ with our grammar and there are two different trees that flat maps to this word:



To resolve this ambiguity, we normally denote these expressions by, respectively, $5 + (4 + 1)$ and $(5 + 4) + 1$. This tells us that trees do not need the parentheses and all ambiguity is removed. In fact, we can see this already in the branching type, in which the parentheses do not add any branching and merely reflect the parentheses in the grammar. Often parentheses can be left out by constructing a grammar more cleverly than what we did, but we leave that for our specific uses of grammars.

Definition B.9

Let A be an alphabet (a set). A **context-free grammar** G over A is a tuple (V, R) where V is a finite set of non-terminal symbols and $R \subseteq V \times (A \cup V)$ is a relation, the **production rules** of G.

Example B.10

Taking $V = \{e\}$ and

$$R = \{(e, n) \mid n \in \mathbb{N}\} \cup \{(e, e + e)\} \cup \{(e, -e)\} \cup \{(e, (e))\}$$

is the grammar given in example B.8.

C. Three-Valued Logic

In what follows, we describe the so-called 3-valued Heyting logic or algebra. Let $\mathbb{T}$ be the set $\{0, X, 1\}$. Intuitively, we understand 0 and 1 as true and false, as in chapter 3, while the third element X of $\mathbb{T}$ should be seen as an unknown truth value. This can occur, for example, in computer when the voltage of a logical signal is not high enough or fluctuates and thereby creates an undefined logic state. We will see that $\mathbb{T}$ can be used a domain for modelling propositional logic. First of all, we define an order $\sqsubseteq$ on $\mathbb{T}$ by

$$0 \sqsubseteq X\,, \quad X \sqsubseteq 1\,, \quad 0 \sqsubseteq 1\,, \quad 0 \sqsubseteq 0\,, \quad X \sqsubseteq X \quad \text{and} \quad 1 \sqsubseteq 1\,.$$

This order allows us to use min and max as usual, and if we use them to interpret conjunction and disjunction, then we will see that they conform to our expectation of an unknown value as input to logic gates:

$\min(a, b)$ (interpretation of $\wedge$) and $\max(a, b)$ (interpretation of $\vee$)

a \ b	0	X	1
0	0	0	0
X	0	X	X
1	0	X	1

a \ b	0	X	1
0	0	X	1
X	X	X	1
1	1	1	1

We can also define a semantic implication $\Longrightarrow_{\mathbb{T}}$ on $\mathbb{T}$ just as we did for the Boolean semantics:

$$a \Longrightarrow_{\mathbb{T}} b = \begin{cases} 1, & a \sqsubseteq b \\ b, & \text{otherwise} \end{cases}$$

The following table lists all the possibilities for $\Longrightarrow_{Tri}$ and the resulting negation:

Tables of $a \Longrightarrow_{\mathbb{T}} b$ and negation of a

a \ b	0	X	1
0	1	1	1
X	0	1	1
1	0	X	1

a	$a \Longrightarrow_{\mathbb{T}} 0$
0	1
X	0
1	0

Putting this all together, we can define for valuations $v\colon \mathrm{PVar} \to \mathbb{T}$ a map $[\![-]\!]_v^{\mathbb{T}}\colon \mathrm{PForm} \to \mathbb{T}$ analogously to definition 3.3. We can use this map to also gives us an entailment relation $\vDash_{\mathbb{T}}$ by defining

$$\Gamma \vDash_{\mathbb{T}} \varphi \text{ if for all v we have } [\![\Gamma]\!]_v^{\mathbb{T}} \leq [\![\varphi]\!]_v^{\mathbb{T}}\,.$$

Also similarly to the Boolean model (theorem 4.12), one can prove the following soundness result.

Theorem C.1

If $\Gamma \vdash \varphi$ is derivable in **ND**, then $\Gamma \vDash_{\mathbb{T}} \varphi$.

However, the similarity with the Boolean model stops when we move to classical logic. Indeed, it is easy to see that $\nvDash_{\mathbb{T}} p \vee \neg p$. Let v be the valuation that is equal to X everywhere. We then have

$$[\![p \vee \neg p]\!]_v = \max\{[\![p]\!]_v, [\![\neg p]\!]_v\} = \max\{X, 0\} = X \neq 1\,.$$

This shows that $\neg \nvDash_{\mathbb{T}} p \vee \neg p$ and therefore the (Contra)-rule from definition 4.16 cannot be sound for this three-valued model.

There are other possibilities for interpreting the implication, see Łukasiewicz's or Kleene's three-valued logic [Kle74, § 64], but different proof systems than **ND** and **cND** are needed to handle those interpretations.

D. Logic Programming

```
% :- table path(_,_,lattice(shortest/3))
% :- table conn/2

% Partial order of lists by length; used in tabled execution
shortest(P1, P2, P) :-
   length(P1, L1),
   length(P2, L2),
   (L1 < L2 -> P = P1; P = P2).

% Right
adjacent(pos(X1,Y1), pos(X2, Y1)) :- succ(X1, X2), X1 < 6.
% Down
adjacent(pos(X1,Y1), pos(X1, Y2)) :- succ(Y1, Y2), Y1 < 4.
% Left
adjacent(pos(X1,Y1), pos(X2, Y1)) :- succ(X2, X1).
% Up
adjacent(pos(X1,Y1), pos(X1, Y2)) :- succ(Y2, Y1).

% Can we go from U to V?
step(U, V) :- adjacent(U, V), free(V).

% conn(U, V) holds if two positions U and V connected.
conn(U, U).
conn(U, V) :-
  conn(W, V),
  step(U, W).

% Can our robot reach the goal?
connr :- robot(U), goal(V), conn(U, V).

% path(U, V, P) holds if P is a path from U to V. A path is here a list of
    positions.
path(U, U, [U]).
path(U, V, [U|P]) :-
  path(W, V, P),
  step(U, W).

% A path P with route(P) leads our robot from the initial position to the
    goal.
route(P) :- robot(U), goal(V), path(U, V, P).

```

```
% Initial position of robot.
robot(pos(2, 3)).

% Position of goal.
goal(pos(5,1)).

% All the positions that do not contain an obstacle.
free(pos(1,1)).
free(pos(1,4)).

free(pos(2,2)).
free(pos(2,3)).
free(pos(2,4)).

free(pos(3,1)).
free(pos(3,2)).
free(pos(3,4)).

free(pos(4,1)).
free(pos(4,2)).
free(pos(4,3)).
free(pos(4,4)).

free(pos(5,1)).
free(pos(5,3)).

free(pos(6,1)).
free(pos(6,2)).
free(pos(6,3)).
```

List of Notation

General Notation

$g \circ f$	composition of g after f	See p. 177
$\emptyset$	empty set that contains no elements	See p. 177
$[n]$	set of first n natural numbers	See p. 177
A^n	set of first n-tuples over A	See p. 99
$\mathbb{N}$	set of natural numbers	See p. 178
$\mathcal{P}(A)$	powerset of A	See p. 177
$A \times B$	product of the sets A and B	See p. 177
B^A	set of maps from A to B	See p. 177

Logic Syntax

ar	arity of function and predicate symbols in first-order signature	See p. 76
$\mathrm{Form}(\mathcal{L}^{=})$	first-order formulas with equality $\doteq$	See p. 118
$\mathcal{F}$	set of function symbols in a first-order signature	See p. 76
$\mathcal{R}$	set of predicate symbols of a first-order signature ("$\mathcal{R}$" stands for "relation")	See p. 76
$\mathcal{L}$	(first-order) signature consisting of function and predicate symbols	See p. 76
$\bot$	syntactic absurdity ("falsity")	See p. 7
$\leftrightarrow$	syntactic bi-implication ("if and only if" or "iff")	See p. 7
$\wedge$	syntactic conjunction ("and")	See p. 7
$\vee$	syntactic disjunction ("to")	See p. 7
$s \doteq t$	syntactic equality ("s is provably equal to t")	See p. 117
$\rightarrow$	syntactic implication ("implies")	See p. 7

$\neg$	syntactic negation ("not")	See p. 7
$\exists x.\, \varphi$	syntactic existential quantification ("there is x such that φ")	See p. 80
$\exists! x.\, \varphi$	syntactic uniqueness quantification ("there is a unique x such that φ")	See p. 120
$\forall x.\, \varphi$	syntactic universal quantification ("for all x φ")	See p. 80
$\top$	syntactic truth ("true" or "top")	See p. 7
$\mathrm{Term}(\mathcal{L})$	terms over a signature $\mathcal{L}$	See p. 77

Operations on Formulas

$\mathrm{bv}(\varphi)$	bound variables appearing in formula φ	See p. 81
$\mathrm{fv}(\varphi)$	free variables appearing in formula φ	See p. 81
$\mathrm{Sub}(\varphi)$	subformulas of φ	See p. 12
$\mathrm{var}(t)$	variables appearing in term t	See p. 78

Semantics

$\mathcal{M}$	$\mathcal{L}$-model for a first-order signature $\mathcal{L}$	See p. 100
$\lvert\mathcal{M}\rvert$	universe of an $\mathcal{L}$-model	See p. 100
$[\![\varphi]\!]_v$	Boolean semantics of φ under valuation v	See p. 18
$[\![\Gamma]\!]_v$	Boolean semantics of context Γ under valuation v	See p. 22
$\Gamma \vDash \varphi$	Propositional entailment: Γ entails φ	See p. 22
$\Longrightarrow$	semantic implication	See p. 18

Proof Theory

$t\,\sigma$	application of the term t to substitution σ	See p. 89
$\varphi\,\sigma$	application of the formula φ to substitution σ	See p. 90
$\Delta \mid \Gamma \vdash \varphi$	first-order sequent consisting of variable context Δ, assumptions Γ x and an FOL formula φ	See p. 94

$\mathbf{ND}_1$	intuitionistic natural deduction for first-order logic	See p. 94
$\mathbf{cND}_1$	classical natural deduction for first-order logic	See p. 97
$\mathbf{ND}_1^=$	intuitionistic natural deduction for first-order logic with equality	See p. 126
$\mathbf{cND}_1^=$	classical natural deduction for first-order logic with equality	See p. 126
$x \# \varphi$	variable x is fresh for formula φ	See p. 90
$x \# \sigma$	variable x is fresh for substitution σ	See p. 90
η	substitution changes nothing	See p. 89
σ	substitutions are maps $\sigma\colon \mathrm{Var} \to \mathrm{Term}$ that assign first-order terms to variables	See p. 89
$\sigma[x := t]$	updates the substitution σ to assign t to variable x	See p. 89