NAIL062 Propositional and Predicate Logic: Lecture Notes

Jakub Bulín[1]

Winter Semester 2024

[1]KTIML MFF UK, jakub.bulin@mff.cuni.cz

The lecture is based on the textbook *A. Nerode, R. Shore: Logic for Applications* [3], with some parts taken from the book *M. Ben-ari: Mathematical Logic for Computer Science* [1]. The course structure and the majority of the content are borrowed from the lectures of Petr Gregor from previous years [2]; see also the lecture notes by Martin Pilát [4]. This version was translated from the original Czech version with the assistance of an LLM-based translator. If you find any typos or other errors, or if there are any parts that are difficult to understand, please let me know.

Contents

Chapter 1

Introduction to Logic

The word *logic* is used in two senses:

- A set of principles that underlie the organization of elements within a system (e.g., a computer program, an electronic device, a communication protocol)
- Reasoning conducted according to strict rules that preserve validity

In computer science, these two meanings converge: first, we formally describe the given system, and then we *formally reason* about it (in practical applications, this is done automatically), i.e., we derive valid *inferences* about the system using some *(formal) proof system.*

Practical applications of logic in computer science include software verification, logic programming, SAT solving, automated reasoning, database theory, knowledge-based representation, and many others. Moreover, logic (to a greater extent than mathematics) is a fundamental tool for describing theoretical computer science.

1.1 Propositional Logic

Now let's demonstrate logic in action with two real-life examples (from the lives of a treasure hunter and a theoretical computer scientist):

1.1.1 Example: Treasure Hunting

Example 1.1.1. While searching for treasure in a dragon's lair, we encountered a fork in the path with two corridors. We know that at the end of each corridor, there is either treasure or a dragon, but not both. A dwarf we met at the fork told us, "At least one of these two corridors leads to the treasure," and after some further urging (and a small bribe), she also said, "The first corridor leads to a dragon." It is well known that dwarves you meet in a dragon's lair either always tell the truth or always lie. Which way should we go?

1.1.2 Formalization in Propositional Logic

We will begin by formalizing the situation and our knowledge in propositional logic. A *proposition* is a statement to which we can assign a truth value: *True* (1) or *False* (0).

Some propositions can be expressed using simpler propositions and logical connectives, e.g., "(The dwarf is lying) *if and only if* (the second corridor leads to a dragon)" or "(The first corridor leads to treasure) *or* (the first corridor leads to a dragon)." If a proposition cannot be decomposed in this way, it is called a *simple proposition*, *atomic proposition*, or *propositional variable*.

Thus, we will describe the entire situation using *propositional variables*. We can also think of these as simple yes/no questions we need to answer to know everything about the given situation. Let's choose "The treasure is in the first corridor" (denote as p_1), and "The treasure is in the second corridor" (p_2). Other propositional variables could be considered, such as "There is a dragon in the first corridor" (d_1) or "The dwarf is telling the truth" (t). However, these can be expressed using $\{p_1, p_2\}$, e.g., t holds if and only if p_1 does not hold. That is, if we know the truth values of p_1, p_2, the truth values of d_1, t are uniquely determined. A smaller number of propositional variables means a smaller search space.

Next, we will express all our knowledge as (*compound*) propositions and write them in formal notation in the *language* of propositional logic over the set of atomic propositions $\mathbb{P} = \{p_1, p_2\}$, using symbols representing logical connectives: $\neg$ ("not X", *negation*), $\wedge$ ("X and Y", *conjunction*), $\vee$ ("X or Y", *disjunction*), $\rightarrow$ ("if X, then Y", *implication*), $\leftrightarrow$ ("X if and only if Y", *equivalence*), and parentheses (,). It is worth mentioning that disjunction is not exclusive; that is, "X or Y" holds even if both X and Y hold, and implication is purely logical: "if X, then Y" holds whenever X does not hold or Y holds.

The information that the corridor contains either a treasure or a dragon, but not both, is already encoded in our choice of propositional variables: the presence of a dragon is the same as the absence of treasure. The dwarf's statement that "The first corridor leads to a dragon" is thus expressed as "It is not the case that the treasure is in the first corridor," formally $\neg p_1$. The statement that "At least one of these two corridors leads to the treasure" is expressed as "The treasure is in the first corridor or the treasure is in the second corridor," formally $p_1 \vee p_2$. The information that dwarves either always tell the truth or always lie can be understood to mean that either both our propositions hold or the negations of both our propositions hold, formally:

$$(\neg p_1 \wedge (p_1 \vee p_2)) \vee (\neg(\neg p_1) \wedge \neg(p_1 \vee p_2))$$

Let us denote this proposition as φ (from the word "formula"; propositions are sometimes also called *propositional formulas*). In our example, all information can be expressed with a single proposition. But in practice, we often need more propositions, sometimes even infinitely many (for example, if we want to describe the execution of a computer program and we do not know in advance how many steps it will take). We then describe the situation using a set of propositions, called a *theory*, here $T = \{\varphi\}$. The propositions in T are also called *axioms* of the theory T[1].

1.1.3 Models and Consequences

Is our information sufficient to determine whether there is treasure in one particular corridor? In other words, we are asking if one of the propositions p_1 or p_2 is a logical *consequence* of the proposition φ (or the theory T). What does this mean?

Let's imagine that there are several "worlds" differing in what is at the end of the first and second corridors. For example, in one of the worlds, there is treasure at the end of the first

[1]Terminology in logic often comes from its application in mathematics.

corridor and a dragon at the end of the second corridor. We can describe this world using a truth valuation of the propositional variables: $p_1 = 1, p_2 = 0$. Such a valuation is called a *model* of the language $\mathbb{P} = \{p_1, p_2\}$ and we write it succinctly as $v = (1, 0)$ (v from the word "valuation"). Thus, we have a total of four different worlds, described by the *models of the language*:

$$\mathrm{M}_{\mathbb{P}} = \{(0,0),(0,1),(1,0),(1,1)\}.$$

Is the world described by the model $v = (1, 0)$ consistent with the information we have, i.e., does the proposition φ (or the theory T) *hold* in it? We can easily determine the truth value of the (compound) proposition φ in the model v, denote it $v(\varphi)$: We know that $v(p_1) = 1$ and $v(p_2) = 0$, so $v(\neg p_1) = 0$, and also $v(\neg p_1 \wedge (p_1 \vee p_2)) = 0$ (it is a conjunction of two propositions, and the first conjunct is false in the model v). Similarly, $v(p_1 \vee p_2) = 1$ (because $v(p_1) = 1$), so $v(\neg(p_1 \vee p_2)) = 0$, and $v(\neg(\neg p_1) \wedge \neg(p_1 \vee p_2)) = 0$. The proposition φ is a disjunction of two propositions, neither of which holds in the model v, so $v(\varphi) = 0$.

A keen reader will surely see the tree structure of the proposition φ and the step-by-step evaluation of $v(\varphi)$ from the leaves to the root. We will present a formal definition in the next chapter.

Similarly, we determine the truth values of the proposition φ in other models. We find that the set of *models of the proposition* φ (or the *models of the theory* T), i.e., the set of all models of the language in which φ (or all axioms of the theory T) holds, is

$$\mathrm{M}_{\mathbb{P}}(\varphi) = \mathrm{M}_{\mathbb{P}}(T) = \{(0,1)\}.$$

We see that our information uniquely determines the model $(0, 1)$, the world in which there is a dragon in the first corridor and treasure in the second corridor. In general, there can be more models; we only need to know that in every model of φ (or of T) the proposition p_2 holds, to conclude that p_2 is a *consequence* of the theory T; we also say that p_2 *holds* in the theory T.

1.1.4 Proof Systems

The approach we have chosen is very inefficient. If we have n propositional variables[2], there are 2^n models of the language, and it is practically impossible to check the validity of the theory in each of them. This is where *proof systems* come into play. In a given proof system, a *proof* of a proposition ψ from a theory T is a precisely, formally defined syntactic object that includes an easily (mechanically) verifiable "proof" (reason) that ψ holds in T, and which can be searched for (using a computer) purely based on the structure of the proposition ψ and the axioms of the theory T ("syntax"), i.e., without having to deal with models ("semantics").

We want two properties from a proof system:

- *soundness*, i.e., if we have a proof of ψ from T, then ψ holds in T, and
- *completeness*, i.e., if ψ holds in T, then there exists a proof of ψ from T,

where soundness is a necessity (without it, searching for proofs makes no sense), and completeness is a desirable property, but an efficient proof system can be useful even if not everything that holds can be proven.

[2]In practice, we commonly have thousands of variables.

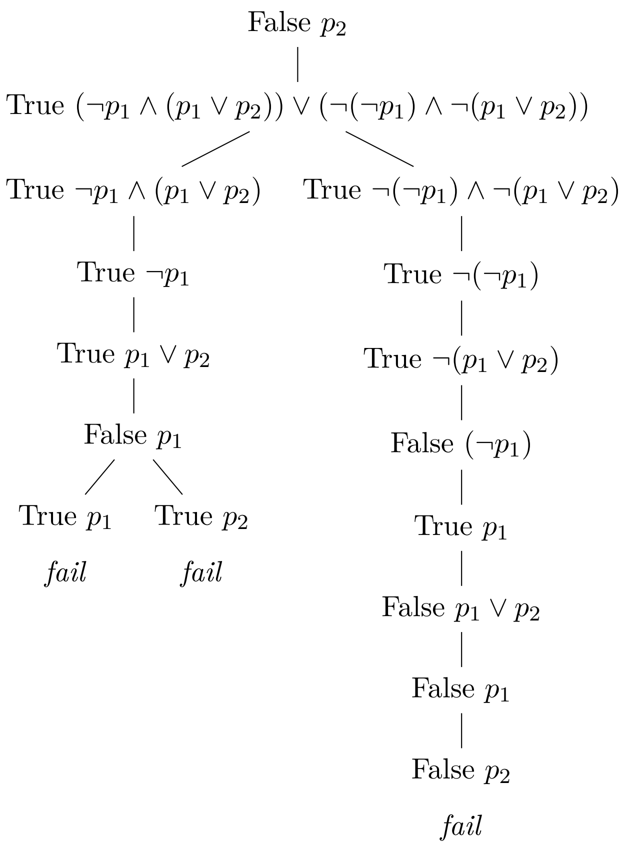

False p_2
True $(\neg p_1 \land (p_1 \lor p_2)) \lor (\neg(\neg p_1) \land \neg(p_1 \lor p_2))$
True $\neg p_1 \land (p_1 \lor p_2)$
True $\neg(\neg p_1) \land \neg(p_1 \lor p_2)$
True $\neg p_1$
True $p_1 \lor p_2$
False p_1
True p_1
True p_2
fail
fail
True $\neg(\neg p_1)$
True $\neg(p_1 \lor p_2)$
False $(\neg p_1)$
True p_1
False $p_1 \lor p_2$
False p_1
False p_2
fail

Figure 1.1: Tableau proof of the proposition p_2 from the theory T

Here we briefly outline two proof systems: the *method of analytic tableaux* and the *resolution method*. They will be formally introduced later, and we will prove soundness and completeness for both of them. Both of these proof systems are based on *proof by contradiction*, i.e., they assume the validity of the axioms from T and the negation of the proposition ψ, and try to reach a contradiction.

1.1.5 Tableau Method

In the method of analytic tableaux, the 'proof' is a *tableau*: a tree whose nodes are labeled with assumptions about the validity of propositions. Let's look at an example of a tableau in Figure 1.1.

We start with the assumption that the proposition p_2 does not hold (because we are proving by contradiction). Then we add the validity of all axioms of the theory T (in our case, there is only one: the proposition φ constructed above). We then build the tableau by simplifying the propositions in the assumptions according to certain rules that ensure the following invariant:

Every model of the theory T in which p_2 does not hold must agree *with one of the branches of the tableau (i.e., satisfy all assumptions on that branch).*

The proposition φ is a disjunction of two propositions, $\varphi = \varphi_1 \vee \varphi_2$. If it holds in some model, then either φ_1 holds in that model or φ_2 holds in it. We branch the tree according to these two possibilities. In the next step, we have the assumption that the proposition $\neg p_1 \wedge (p_1 \vee p_2)$ is true. In that case, both $\neg p_1$ and $p_1 \vee p_2$ must hold, so we add both of these assumptions to the end of the branch. The truth of $\neg p_1$ means the falsity of p_1, and so on.

We proceed in this manner until it is no longer possible to simplify the propositions in the assumptions, i.e., until they are just propositional variables. If we find a pair of opposite assumptions about some proposition ψ on one branch, i.e., that it both holds and does not hold, we know that no model can agree with this branch. Such a branch is called *contradictory.* Since we are proving by contradiction, a proof is a tableau in which every branch is contradictory. This ensures that there is no model of T in which p_2 does not hold. From this, it follows that p_2 holds in every model of T, in other words, it is a consequence of T, which is what we wanted to prove.

For now, we will be satisfied with understanding the basic idea of this method; the details will be presented later in Chapter 4.

1.1.6 Resolution Method

It is not difficult to program a systematic search for a tableau proof. In practice, however, another proof system is used that has a much simpler and more efficient implementation: the *resolution method.* This method dates back to 1965 and forms the basis of most *automated reasoning systems*, *SAT solvers*, or Prolog language interpreters (see Subsection 8.7.2).

The resolution method is based on the fact that every proposition can be equivalently expressed in a special form, called *conjunctive normal form (CNF).* A *literal* is a propositional variable p or its negation $\neg p$ (i.e., literals are propositions that just determine the value of a single propositional variable). A disjunction of several literals, e.g., $p \vee \neg q \vee \neg r$, is called a *clause.* A proposition is in CNF if it is a conjunction of clauses. For every proposition ψ, there is an *equivalent* proposition ψ' in CNF. Equivalent means having the same meaning (the same models), which we write as $\psi \sim \psi'$. Later, we will show two methods of conversion to CNF; for now, let's look at our example: in the proposition

$$(\neg p_1 \wedge (p_1 \vee p_2)) \vee (\neg(\neg p_1) \wedge \neg(p_1 \vee p_2))$$

we first replace $\neg(\neg p_1) \sim p_1$ and $\neg(p_1 \vee p_2) \sim (\neg p_1 \wedge \neg p_2)$:

$$(\neg p_1 \wedge (p_1 \vee p_2)) \vee (p_1 \wedge \neg p_1 \wedge \neg p_2)$$

and then repeatedly use the *distributivity* of $\vee$ over $\wedge$ (imagine $\vee$ as multiplication and $\wedge$ as addition):

$$(\neg p_1 \vee p_1) \wedge (\neg p_1 \vee \neg p_1) \wedge (\neg p_1 \vee \neg p_2) \wedge (p_1 \vee p_2 \vee p_1) \wedge (p_1 \vee p_2 \vee \neg p_1) \wedge (p_1 \vee p_2 \vee \neg p_2)$$

This proposition is already in CNF, but we further simplify it: we omit duplicate literals from clauses, and we realize that if a clause contains a pair of opposite literals $p, \neg p$, it is a *tautology* (true in every model), and we can remove it. We obtain the CNF proposition

$$\neg p_1 \wedge (\neg p_1 \vee \neg p_2) \wedge (p_1 \vee p_2)$$

which is equivalent to the original proposition φ. Since we want to prove p_2 by contradiction, we add the clause $\neg p_2$:

$$\neg p_1 \wedge (\neg p_1 \vee \neg p_2) \wedge (p_1 \vee p_2) \wedge \neg p_2$$

The proposition p_2 holds in the theory T if and only if this CNF proposition is *unsatisfiable* (has no model). Propositions in CNF will also be written in *set notation*[3]:

$$\{\{\neg p_1\}, \{\neg p_1, \neg p_2\}, \{p_1, p_2\}, \{\neg p_2\}\}$$

The *resolution rule* states that if we have a pair of clauses, one containing the literal p and the other the opposite literal $\neg p$, then their *resolvent*, that is, the clause formed by removing the literal p from the first clause and $\neg p$ from the second clause and taking the union of the remaining literals, is a logical consequence of the two clauses. For example, from $p \vee \neg q \vee \neg r$ and $\neg p \vee \neg q \vee s$, we can derive the resolvent $\neg q \vee \neg r \vee s$. *Resolution refutation* of a formula in CNF is then a sequence of clauses ending with the *empty clause* $\square$ (indicating a contradiction), where each clause is either from the given CNF formula or it is a resolvent of some two preceding clauses. In our case:

$$\{\neg p_1\}, \{p_1, p_2\}, \{p_2\}, \{\neg p_2\}, \square$$

The third clause is the resolvent of the first and second, and the fifth is the resolvent of the third and fourth. The resolution can also be naturally represented using a *resolution tree*, where the leaves are clauses from the given formula, and the inner nodes are resolvents of their children:

$$\begin{tikzcd}
\{\neg p_1\} \arrow[dr, dash] & & \{p_1, p_2\} \arrow[dl, dash] & \\
& \{p_2\} \arrow[dr, dash] & & \{\neg p_2\} \arrow[dl, dash] \\
& & \square &
\end{tikzcd}$$

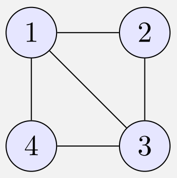
If the CNF formula had a model, the model would have to satisfy all of its clauses, and thus gradually all the resolvents in the sequence, and finally the empty clause. But no model can satisfy the empty clause (a disjunction of zero possibilities). We therefore have a proof by contradiction and thus know that we will find the treasure in the second corridor.

The details of the resolution method will be presented in Chapter 5.

1.1.7 Example: Graph Coloring

In the second example, we will get a bit closer to applications. Unlike logical puzzles in the form of word problems, various variants of the graph coloring problem appear in a variety of practical tasks, from scheduling problems to the design of physical and network systems to image processing.

Example 1.1.2. Find a vertex coloring of the following graph using three colors, i.e., assign to its vertices the colors R, G, B in such a way that no edge is monochromatic.



[3]In practical implementation, we could use a list of clauses, where each clause is a list of (unique) literals in some chosen order. Again, imagine thousands of propositional variables and clauses.

We represent the graph as a set of vertices and a set of edges, where each edge is a pair of vertices. It is easier to work with ordered pairs, so we choose an (arbitrary) orientation of the edges.

$$\mathcal{G} = \langle V; E\rangle = \langle \{1,2,3,4\}; \{(1,2),(1,3),(1,4),(2,3),(3,4)\}\rangle$$

We start again with formalization in propositional logic. Let the set of colors be $C = \{R, G, B\}$. The natural choice of propositional variables is "vertex v has color c", denoted as p_v^c, for each vertex $v \in V$ and each color $c \in C$. Our (ordered) set of propositional variables has 12 elements:

$$\mathbb{P} = \{p_v^c \mid c \in C, v \in V\} = \{p_1^R, p_1^G, p_1^B, p_2^R, p_2^G, p_2^B, p_3^R, p_3^G, p_3^B, p_4^R, p_4^G, p_4^B\}$$

We have a total of $|\,\mathrm{M}_{\mathbb{P}}\,| = 2^{12} = 4096$ models of the language (represented by 12-dimensional 0–1 vectors). Most of them cannot be interpreted as a coloring of the graph. For example, $v = (1,1,0,0,\dots,0)$ indicates that vertex 1 is colored both red and green. We start with a theory expressing that each vertex has at most one color. There are several ways to express this. We will state, for each vertex, that it must not have (at least) one color from each pair of colors. This gives us a theory in CNF:

$$\begin{aligned}
T_1 = \{&(\neg p_1^R \vee \neg p_1^G) \wedge (\neg p_1^R \vee \neg p_1^B) \wedge (\neg p_1^G \vee \neg p_1^B),\\
&(\neg p_2^R \vee \neg p_2^G) \wedge (\neg p_2^R \vee \neg p_2^B) \wedge (\neg p_2^G \vee \neg p_2^B),\\
&(\neg p_3^R \vee \neg p_3^G) \wedge (\neg p_3^R \vee \neg p_3^B) \wedge (\neg p_3^G \vee \neg p_3^B),\\
&(\neg p_4^R \vee \neg p_4^G) \wedge (\neg p_4^R \vee \neg p_4^B) \wedge (\neg p_4^G \vee \neg p_4^B)\}\\
= \{&(\neg p_v^R \vee \neg p_v^G) \wedge (\neg p_v^R \vee \neg p_v^B) \wedge (\neg p_v^G \vee \neg p_v^B) \mid v \in V\}
\end{aligned}$$

The theory T_1 could be called the theory of *partial* vertex colorings of the graph $\mathcal{G}$. The theory T_1 has $|\,\mathrm{M}_{\mathbb{P}}(T_1)| = 4^4 = 2^8 = 256$ models. (Why?) If we want complete colorings, we add the condition that each vertex has at least one color.[4]

$$\begin{aligned}
T_2 &= T_1 \cup \{p_1^R \vee p_1^G \vee p_1^B, p_2^R \vee p_2^G \vee p_2^B, p_3^R \vee p_3^G \vee p_3^B, p_4^R \vee p_4^G \vee p_4^B\}\\
&= T_1 \cup \{p_v^R \vee p_v^G \vee p_v^B \mid v \in V\}\\
&= T_1 \cup \{\bigvee_{c \in C} p_v^c \mid v \in V\}
\end{aligned}$$

The theory T_2 has $3^4 = 81$ models. It is an *extension* of the theory T_1, as every consequence of T_1 also holds in the theory T_2. In fact, $\mathrm{M}_{\mathbb{P}}(T_2) \subseteq \mathrm{M}_{\mathbb{P}}(T_1)$.[5] The remaining condition is to forbid monochromatic edges. For each edge and each color, we specify that at least one of the vertices of the edge must not have the given color. For illustration, we will write out the

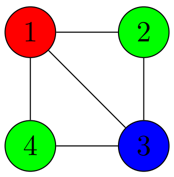
[4]The symbol $\bigvee$ is used similarly to the symbols $\sum$ for summation and $\prod$ for product: to simplify the notation of a proposition in the form of a disjunction. For example, if $v = 1$, then $\bigvee_{c \in C} p_v^c$ represents the proposition $p_1^R \vee p_1^G \vee p_1^B$. Analogously, $\bigwedge$ for conjunction.

[5]Here we see a typical example of the anti-monotonic relationship of the so-called *Galois correspondence*: the more properties (propositions) we require, the fewer objects (models) satisfy these properties.

complete list of propositions one last time; in the future, we will use the abbreviated notation.

$$\begin{aligned}
T_3 = T_2 \cup \{&(\neg p_1^R \vee \neg p_2^R) \wedge (\neg p_1^G \vee \neg p_2^G) \wedge (\neg p_1^B \vee \neg p_2^B),\\
&(\neg p_1^R \vee \neg p_3^R) \wedge (\neg p_1^G \vee \neg p_3^G) \wedge (\neg p_1^B \vee \neg p_3^B),\\
&(\neg p_1^R \vee \neg p_4^R) \wedge (\neg p_1^G \vee \neg p_4^G) \wedge (\neg p_1^B \vee \neg p_4^B),\\
&(\neg p_2^R \vee \neg p_3^R) \wedge (\neg p_2^G \vee \neg p_3^G) \wedge (\neg p_2^B \vee \neg p_3^B),\\
&(\neg p_3^R \vee \neg p_4^R) \wedge (\neg p_3^G \vee \neg p_4^G) \wedge (\neg p_3^B \vee \neg p_4^B)\}\\
= T_2 \cup \{ &\bigwedge_{c \in C} (\neg p_u^c \vee \neg p_v^c) \mid (u,v) \in E\}
\end{aligned}$$

The resulting theory T_3 is *satisfiable* (has a model) if and only if the graph $\mathcal{G}$ is 3-colorable. It has 6 models. The models are in one-to-one correspondence with 3-colorings of the graph $\mathcal{G}$. The model $v = (1,0,0,0,1,0,0,0,1,0,1,0)$ corresponds to the following coloring; other colorings can be obtained by permuting the colors.



Once we have the theory T_3 formalizing 3-coloring of the graph $\mathcal{G}$, we can easily solve related questions, such as finding all colorings where vertex 1 is blue and vertex 2 is green: these correspond to the models of the theory $T_3 \cup \{p_1^B, p_2^G\}$. Or we can prove that vertices 2 and 4 must be colored the same color. We can use the tableau method: at the root of the tableau will be the assumption

$$\text{False } (p_2^R \wedge p_4^R) \vee (p_2^G \wedge p_4^G) \vee (p_2^B \wedge p_4^B)$$

Or we can find a resolution refutation of the theory formed by converting the axioms of T_3 into CNF and adding the CNF equivalent of the *negation* of the proposition $(p_2^R \wedge p_4^R) \vee (p_2^G \wedge p_4^G) \vee (p_2^B \wedge p_4^B)$ (again, because it is a proof by contradiction, we assume for contradiction that the vertices *do not* have the same color).

1.2 Predicate Logic

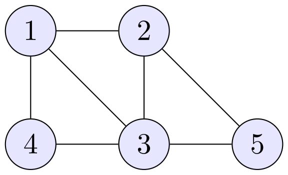
Now we will very briefly and informally introduce *predicate logic*. Predicate logic deals with properties of objects and relationships between objects. For example:

All humans are mortal.
Socrates is a human.
Socrates is mortal.

In fact, propositional logic emerged later (about a century) than Aristotle's predicate logic, and was largely forgotten for a long time.

1.2.1 Disadvantages of Formalization in Propositional Logic

The disadvantage of formalizing our graph coloring problem in propositional logic is that the resulting theory T_3 is quite large and was created ad hoc for the graph $\mathcal{G}$. Imagine we need to modify the graph $\mathcal{G}$, for example, by adding a new vertex 5 connected by edges to vertices 2 and 3:



To formalize the new problem, we need to add three new propositional variables to our language: $\mathbb{P}' = \mathbb{P} \cup \{p_5^R, p_5^G, p_5^B\}$, and create new theories T_1', T_2', T_3' by adding axioms related to vertex 5 and edges $(2,5), (3,5)$. The issue here is that the structure of the graph $\mathcal{G}$ and natural properties like "there is an edge from vertex u to vertex v" or "vertex u is green" have been ('unnaturally') 'hardcoded' into 0–1 variables. This drawback is addressed by *predicate logic.*

1.2.2 A Brief Introduction to Predicate Logic

A *model* in predicate logic is not a 0–1 vector but rather a *structure.* Examples of structures are our (oriented) graphs:

$$\mathcal{G} = \langle V^{\mathcal{G}}; E^{\mathcal{G}} \rangle = \langle \{1,2,3,4\}; \{(1,2),(1,3),(1,4),(2,3),(3,4)\} \rangle$$
$$\mathcal{G}' = \langle V^{\mathcal{G}'}; E^{\mathcal{G}'} \rangle = \langle \{1,2,3,4,5\}; \{(1,2),(1,3),(1,4),(2,3),(3,4),(2,5),(3,5)\} \rangle$$

Both graphs consist of a set of vertices and a binary relation on this set. These are structures in the *language of graph theory* $\mathcal{L} = \langle E \rangle$, where E is a binary *relation symbol.* The *language* of predicate logic specifies which relations (how many and of what arity — unary, binary, ternary, etc.) the structures should have, and which symbols we will use for them. Additionally, we use the equality symbol $=$, and structures can also contain functions and constants (such as the functions $+, -, \cdot$ and constants $0, 1$ in the *field* of real numbers), but we will leave these for later.

Predicate logic uses the same logical connectives as propositional logic, but the basic building blocks of *predicate formulas* are not propositional variables, but so-called *atomic formulas*, for example: $E(x, y)$ represents the statement that there is an edge from vertex x to vertex y in the graph. Here x, y are *variables* representing the vertices of the given graph. Additionally, we can use *quantifiers* in the formulas: $(\forall x)$ "for all vertices x" and $(\exists y)$ "there exists a vertex y".[6]

Now we can formalize statements that make sense for any graph. For example:

- "There are no loops in the graph":

$$(\forall x)(\neg E(x, x))$$

[6] We can think of quantifiers as "conjunction" or "disjunction" over all vertices of the graph.

- "There exists a vertex with out-degree 1":

$$(\exists x)(\exists y)(E(x,y) \wedge (\forall z)(E(x,z) \rightarrow y = z))$$

In the given graph $\mathcal{G}$ and under the assignment of vertex u to the variable x and vertex v to the variable y, we evaluate $E(x,y)$ as True if and only if $(u,v) \in E^{\mathcal{G}}$.

1.2.3 Formalization of Graph Coloring in Predicate Logic

Let us return to graph coloring. A natural way to formalize our 3-coloring problem is in the language $\mathcal{L}' = \langle E, R, G, B\rangle$, where E is binary and R, G, B are unary relation symbols, so $R(x)$ means "vertex x is red". A structure for this language is an (oriented) graph along with a triple of sets of vertices, e.g.,

$$\begin{aligned}\mathcal{G}_C &= \langle V^{\mathcal{G}_C}; E^{\mathcal{G}_C}, R^{\mathcal{G}_C}, G^{\mathcal{G}_C}, B^{\mathcal{G}_C}\rangle \\ &= \langle \{1,2,3,4\}; \{(1,2),(1,3),(1,4),(2,3),(3,4)\}, \{1\}, \{2,4\}, \{3\}\rangle\end{aligned}$$

represents the graph $\mathcal{G}$ with the valid coloring from the picture above. We will say that $\mathcal{G}_C$ is an *expansion* of the $\mathcal{L}$-structure $\mathcal{G}$ into the language $\mathcal{L}'$.

As in propositional logic, we first need to ensure that our models represent colored graphs. We start with the requirement that each vertex is colored with at most one color:

$$(\forall x)((\neg R(x) \vee \neg G(x)) \wedge (\neg R(x) \vee \neg B(x)) \wedge (\neg G(x) \vee \neg B(x)))$$

We express coloring with at least one color as follows:

$$(\forall x)(R(x) \vee G(x) \vee B(x))$$

And we formalize the edge condition using the predicate $E(x,y)$ as follows:

$$(\forall x)(\forall y)(E(x,y) \rightarrow ((\neg R(x) \vee \neg R(y)) \wedge (\neg G(x) \vee \neg G(y)) \wedge (\neg B(x) \vee \neg B(y))))$$

The models of the resulting theory represent oriented graphs with vertex 3-coloring.

1.3 Other Types of Logical Systems

Predicate logic where variables represent individual vertices is called *first-order* logic (abbreviated as *FO* logic). In *second-order* logic (abbreviated as *SO* logic), we also have variables representing sets of vertices or even sets of n-tuples of vertices (i.e., relations, functions). For example, the statement that a given graph is bipartite can be formalized in second-order logic with the following formula, where S is a *second-order variable* representing a set of vertices and $S(x)$ expresses that "vertex x is an element of set S":

$$(\exists S)(\forall x)(\forall y)(E(x,y) \rightarrow (S(x) \leftrightarrow \neg S(y)))$$

As another important example, consider the statement that every non-empty subset bounded from below has an infimum,[7] which can be formalized in second-order logic as follows:[8]

$$\begin{aligned}&(\forall S)((\exists x)S(x) \wedge (\exists x)(\forall y)(S(y) \rightarrow x \leq y) \rightarrow \\ &(\exists x)((\forall y)(S(y) \rightarrow x \leq y) \wedge (\forall z)((\forall y)(S(y) \rightarrow z \leq y) \rightarrow z \leq x)))\end{aligned}$$

[7] Which holds in the ordered set of real numbers but not in the rational numbers, e.g., $\{x \mid x^2 > 2, x > 0\}$.

[8] Even though this formula is quite complex, try to understand its individual parts.

In *third-order* logic, we even have variables representing sets of sets (which is useful, for example, in topology), etc.

Besides propositional and predicate logic, there are other types of logical systems, such as intuitionistic logic (which only allows constructive proofs), temporal logics (which talk about validity 'always', 'sometime in the future', 'until', etc.), modal logics ('it is possible', 'it is necessary'), and fuzzy logic (where we have statements that are '0.35 true'). These logics have important applications in computer science, e.g., in artificial intelligence (modal logics for reasoning of autonomous agents about their environment), in parallel programming (temporal logics), or in washing machines (fuzzy logics). In this course, we will limit ourselves to propositional logic and first-order predicate logic.

1.4 About the Lecture

The lecture is divided into three parts:

The first part deals with propositional logic. First, we introduce syntax and semantics, then the problem of satisfiability of CNF formulas (the well-known NP-complete problem SAT) and its polynomially solvable fragments (2-SAT, Horn-SAT). We will also demonstrate the practical use of a SAT solver. We will continue with the tableau method, proving its soundness and completeness, and several applications, such as the Compactness Theorem. Finally, we will introduce the resolution method in propositional logic.

In the second part, we will introduce predicate logic. Again, we start with syntax and semantics, show how to adapt the tableau method for predicate logic, and end with the resolution method. Additionally, we will mention several practical applications, such as database queries and logic programming (the Prolog language). The structure of the exposition in this part is closely tied to the previous part. Many definitions, theorems, proofs, and algorithms will be very similar to their counterparts in propositional logic. They will mainly differ at the low level, in technical details. Therefore, it is important to have a good understanding of propositional logic before moving on to predicate logic.

The third and smallest part is an introduction to model theory, axiomatizability, and algorithmic decidability. This is a taste of more advanced topics that one will encounter primarily in theoretical computer science and mathematical logic, although some have their place in applied computer science as well. Finally, we will introduce the famous Gödel's incompleteness theorems, which demonstrate the limits of formal methods (formal provability in an axiomatic system).

Part I

Propositional logic

Chapter 2

Syntax and Semantics of Propositional Logic

Syntax is a set of formal rules for creating well-formed sentences consisting of words (in the case of natural languages) or formal expressions consisting of symbols (e.g., statements in a programming language). In contrast, *semantics* describes the meaning of such expressions. The relationship between syntax and semantics is fundamental to all of logic and is therefore the key to its understanding.

2.1 Syntax of Propositional Logic

First, we define the formal 'statements' with which we will work in propositional logic.

2.1.1 Language

The *language* of propositional logic is determined by a non-empty set of *propositional variables* $\mathbb{P}$ (also called *atomic propositions* or *atoms*). This set can be finite or infinite, but it will usually be countable[1] (unless otherwise specified), and it will have a fixed ordering. For propositional variables, we will typically use the notation p_i (from the word "proposition"), but for better readability, especially if $\mathbb{P}$ is finite, we will also use $p, q, r, \ldots$. For example:

$$\begin{aligned}
&\mathbb{P}_1 = \{p, q, r\}\\
&\mathbb{P}_2 = \{p_0, p_1, p_2, p_3, \ldots\} = \{p_i \mid i \in \mathbb{N}\}
\end{aligned}$$

In addition to propositional variables, the language also includes *logical symbols*: symbols for logical connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ and parentheses $($, $)$. However, for simplicity, we will talk about the "language $\mathbb{P}$".

Remark 2.1.1. If we need to formally express the ordering of the propositional variables in $\mathbb{P}$, we imagine it as a bijection $\iota\colon \{0, 1, \ldots, n-1\} \to \mathbb{P}$ (for a finite, n-element language) or $\iota\colon \mathbb{N} \to \mathbb{P}$ (if $\mathbb{P}$ is countably infinite). In our examples, $\iota_1(0) = p$, $\iota_1(1) = q$, $\iota_1(2) = r$, and $\iota_2(i) = p_i$ for all $i \in \mathbb{N}$.[2]

[1]This is important in many applications in computer science, as uncountable sets cannot fit into a (even infinite) computer.

[2]The set of natural numbers $\mathbb{N}$ includes zero, see standard ISO 80000-2:2019.

2.1.2 Proposition

The basic building block of propositional logic is a *proposition*, also called *propositional formula.* It is a finite string composed of propositional variables and logical symbols according to certain rules. Atomic propositions are propositions, and we can further create propositions from simpler propositions and logical symbols: for example, for the logical connective $\wedge$, we write first the symbol '(', then the first proposition, the symbol '$\wedge$', the second proposition, and finally the symbol ')'.

Definition 2.1.2 (Proposition)**.** A *proposition* (*propositional formula*) in the language $\mathbb{P}$ is an element of the set $\mathrm{PF}_\mathbb{P}$ defined as follows: $\mathrm{PF}_\mathbb{P}$ is the smallest set satisfying[3]

- for every atomic proposition $p \in \mathbb{P}$, $p \in \mathrm{PF}_\mathbb{P}$,
- for every proposition $\varphi \in \mathrm{PF}_\mathbb{P}$, $(\neg\varphi)$ is also an element of $\mathrm{PF}_\mathbb{P}$,
- for every $\varphi, \psi \in \mathrm{PF}_\mathbb{P}$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$, and $(\varphi \leftrightarrow \psi)$ are also elements of $\mathrm{PF}_\mathbb{P}$.

Propositions are usually denoted by Greek letters φ, ψ, χ (φ from the word "formula"). To avoid listing all four binary logical connectives, we sometimes use a placeholder symbol $\square$. Thus, the third point of the definition could be expressed as:

- for every $\varphi, \psi \in \mathrm{PF}_\mathbb{P}$ and $\square \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, $(\varphi \mathbin{\square} \psi)$ is also an element of $\mathrm{PF}_\mathbb{P}$.

A *subproposition* (*subformula*) is a substring that is itself a proposition. Note that all propositions are necessarily finite strings, created by applying a finite number of steps from the definition to their subpropositions.

Example 2.1.3. The proposition $\varphi = ((p \vee (\neg q)) \leftrightarrow (r \rightarrow (p \wedge q)))$ has the following subpropositions: $p, q, (\neg q), (p \vee (\neg q)), r, (p \wedge q), (r \rightarrow (p \wedge q)), \varphi$.

A proposition in the language $\mathbb{P}$ does not have to contain all atomic propositions from $\mathbb{P}$ (and it cannot if $\mathbb{P}$ is an infinite set). Therefore, it will be useful to denote by $\mathrm{Var}(\varphi)$ the set of atomic propositions occurring in φ.[4] In our example, $\mathrm{Var}(\varphi) = \{p, q, r\}$.

We introduce abbreviations for two special propositions: $\top = (p \vee (\neg p))$ (*tautology*) and $\bot = (p \wedge (\neg p))$ (*contradiction*), where $p \in \mathbb{P}$ is fixed (e.g., the first atomic proposition from $\mathbb{P}$). Thus, the proposition $\top$ is always true, and the proposition $\bot$ is always false.

When writing propositions, we may omit some parentheses for better readability. For example, the proposition φ from Example 2.1.3 can be represented by the string $p \vee \neg q \leftrightarrow (r \rightarrow p \wedge q)$. We omit outer parentheses and use priority of operators: $\neg$ has the highest priority, followed by $\wedge, \vee$, and finally $\rightarrow, \leftrightarrow$ have the lowest priority. Furthermore, the notation $p \wedge q \wedge r \wedge s$ means the proposition $(p \wedge (q \wedge (r \wedge s)))$, and similarly for $\vee$.[5][6]

[3]This kind of definition is called *inductive.* It can also be naturally expressed using a *formal grammar*, see the course NTIN071 Automata and Grammars.

[4]If we do not specify the language of a proposition (and if it is not clear from the context), we mean that it is in the language $\mathrm{Var}(\varphi)$.

[5]Due to the associativity of $\wedge, \vee$, the placement of parentheses does not matter.

[6]Sometimes finer priorities are introduced, $\wedge$ often has higher priority than $\vee$, and $\rightarrow$ has higher priority than $\leftrightarrow$. Also, sometimes $p \rightarrow q \rightarrow r$ is written instead of $(p \rightarrow (r \rightarrow q))$, although $\rightarrow$ is not associative and so here the placement of parentheses does matter. We prefer to avoid both of those conventions.

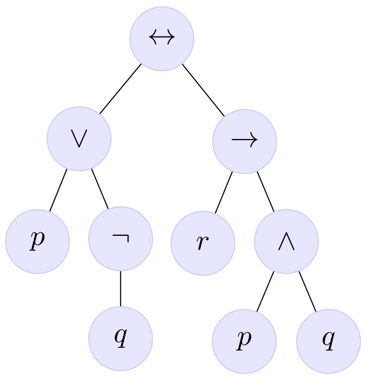


Figure 2.1: Tree of the proposition $\varphi = ((p \vee (\neg q)) \leftrightarrow (r \rightarrow (p \wedge q)))$

2.1.3 Tree of a Proposition

In the definition of a proposition, we chose *infix* notation (with parentheses) purely for human readability. Nothing would prevent us from using *prefix* ("Polish") notation, i.e., defining propositions as follows:

- every atomic proposition is a proposition, and
- if φ, ψ are propositions, then $\neg\varphi$, $\wedge\varphi\psi$, $\vee\varphi\psi$, $\rightarrow\varphi\psi$, and $\leftrightarrow\varphi\psi$ are also propositions.

The proposition $\varphi = ((p \vee (\neg q)) \leftrightarrow (r \rightarrow (p \wedge q)))$ would then be written as $\varphi = \leftrightarrow\vee p\neg q\rightarrow r\wedge pq$. We could also use *postfix* notation and write $\varphi = pq\neg\vee rpq\wedge\rightarrow\leftrightarrow$. The essential information about a proposition is actually contained in its tree structure, which captures how it is composed of simpler propositions, similar to the tree of an arithmetic expression.

Example 2.1.4. The tree of the proposition $\varphi = ((p \vee (\neg q)) \leftrightarrow (r \rightarrow (p \wedge q)))$ is illustrated in Figure 2.1. Notice also how the subpropositions of φ correspond to subtrees. The proposition φ is obtained by traversing the tree from the root, and at each node:

- if the label is an atomic proposition, write it out,
- if the label is a negation: write '$(\neg$', recursively call the child, write ')',
- otherwise (for binary logical connectives): write '(', call the left child, write the label, call the right child, write ')'.[7]

Now we will define the tree of a proposition formally, *by induction on the structure of the proposition*:[8]

Definition 2.1.5 (Tree of a Proposition)**.** The *tree of a proposition* φ, denoted Tree(φ), is a rooted ordered tree, defined inductively as follows:

[7]Prefix and postfix notations would be obtained similarly, but we do not write parentheses, and the label is written immediately upon entering or just before leaving the node.

[8]Once we have the tree of a proposition defined, we can understand induction on the structure of the proposition as induction on the depth of the tree. For now, understand it as induction on the number of steps in Definition 2.1.2 by which the proposition was created. Alternatively, induction on the length of the proposition or the number of logical connectives would work as well.

- If φ is an atomic proposition p, Tree(φ) contains a single node, and its label is p.
- If φ is of the form $(\neg\varphi')$, Tree(φ) has a root labeled $\neg$, and its single child is the root of Tree(φ').
- If φ is of the form $(\varphi' \square \varphi'')$ for $\square \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, Tree($\varphi$) has a root labeled $\square$ with two children: the left child is the root of the tree Tree(φ'), and the right child is the root of Tree(φ'').

Exercise 2.1. Prove that every proposition has a uniquely determined proposition tree, and vice versa.

2.1.4 Theory

In practical applications, we do not express the desired properties with a single proposition — it would have to be very long and complex and difficult to work with — but with many simpler propositions.

Definition 2.1.6 (Theory)**.** A *theory* in the language $\mathbb{P}$ is any set of propositions in $\mathbb{P}$, that is, any subset $T \subseteq \mathrm{PF}_{\mathbb{P}}$. The individual propositions $\varphi \in T$ are also called *axioms*.

Finite theories could be replaced by a single proposition: the conjunction of all their axioms. But that would not be practical. Moreover, we also allow infinite theories (a trivial example is the theory $T = \mathrm{PF}_{\mathbb{P}}$), and the empty theory $T = \emptyset$.[9]

2.2 Semantics of Propositional Logic

In our logic, the semantics are given by one of two possible values: *True* or *False*. (In other logical systems, semantics can be more interesting.)

2.2.1 Truth Value

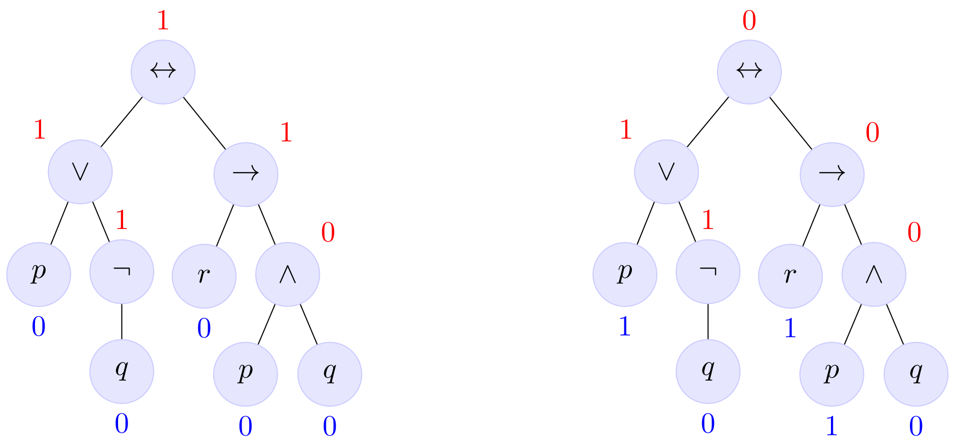
Propositions can be assigned one of two possible truth values: *True* (1) or *False* (0). Atomic propositions represent simple, indivisible statements (hence the name '*atomic*'); their truth value must be assigned to correspond to what we want to model (that is why we call them *propositional variables*). Once we *assign* truth values to the atomic propositions, the truth value of any compound proposition is uniquely determined and can be easily calculated according to the tree of the proposition:

Example 2.2.1. Let us calculate the truth value of the proposition $\varphi = ((p \vee (\neg q)) \leftrightarrow (r \rightarrow (p \wedge q)))$ for the assignments (a) $p = 0$, $q = 0$, $r = 0$ and (b) $p = 1$, $q = 0$, $r = 1$. We proceed from the leaves towards the root, similarly to evaluating an arithmetic expression. The proposition φ is *true* under assignment (a) and *false* under assignment (b). See Figure 2.2.

Logical connectives in the inner nodes are evaluated according to their *truth tables*, see Table 2.1.[10]

[9]Infinite theories are useful, for example, for describing the development of a system over (discrete) time steps $t = 0, 1, 2, \ldots$. The empty theory is not useful for anything, but it would be awkward to formulate statements about logic if theories had to be non-empty.

[10]Let us recall once again that disjunction is not exclusive, i.e., $p \vee q$ is true even if both p and q are true, and that implication is purely logical, i.e., $p \rightarrow q$ is true whenever p is false.



(a) $p = 0$, $q = 0$, $r = 0$ (b) $p = 1$, $q = 0$, $r = 1$

Figure 2.2: Truth value of a proposition

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \to q$	$p \leftrightarrow q$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Table 2.1: Truth tables of logical connectives.

2.2.2 Propositions and Boolean Functions

To formalize the truth value of a proposition, we first look at the relationship between propositions and Boolean functions.

A *Boolean function* is a function $f\colon \{0,1\}^n \to \{0,1\}$, meaning that the input is an n-tuple of zeros and ones, and the output is 0 or 1. Each logical connective represents a Boolean function. In the case of negation, it is a unary function $f_\neg(x) = 1 - x$, while the other logical connectives correspond to binary functions described in Table 2.2.

Definition 2.2.2 (Truth Function)**.** The *truth function* of a proposition φ in a finite language $\mathbb{P}$ is the function $f_{\varphi,\mathbb{P}}\colon \{0,1\}^{|\mathbb{P}|} \to \{0,1\}$ defined inductively:

- If φ is the i-th atomic proposition from $\mathbb{P}$, then $f_{\varphi,\mathbb{P}}(x_0,\dots,x_{n-1}) = x_i$,
- If $\varphi = (\neg\varphi')$, then

$$f_{\varphi,\mathbb{P}}(x_0,\dots,x_{n-1}) = f_\neg(f_{\varphi',\mathbb{P}}(x_0,\dots,x_{n-1})),$$

$f_\wedge(x,y)$:

	0	1
0	0	0
1	0	1

$f_\vee(x,y)$:

	0	1
0	0	1
1	1	1

$f_\to(x,y)$:

	0	1
0	1	1
1	0	1

$f_\leftrightarrow(x,y)$:

	0	1
0	1	0
1	0	1

Table 2.2: Boolean functions of logical connectives

- If $(\varphi' \,\square\, \varphi'')$ where $\square \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, then

$$f_{\varphi,\mathbb{P}}(x_0, \dots, x_{n-1}) = f_\square(f_{\varphi',\mathbb{P}}(x_0, \dots, x_{n-1}), f_{\varphi'',\mathbb{P}}(x_0, \dots, x_{n-1})).$$

Example 2.2.3. Let us calculate the truth function of the proposition $\varphi = ((p \vee (\neg q)) \leftrightarrow (r \rightarrow (p \wedge q)))$ in the language $\mathbb{P}' = \{p, q, r, s\}$:

$$f_{\varphi,\mathbb{P}'}(x_0, x_1, x_2, x_3) = f_\leftrightarrow(f_\vee(x_0, f_\neg(x_1)), f_\rightarrow(x_2, f_\wedge(x_0, x_1)))$$

Truth value of the proposition φ for the truth assignment $p = 1$, $q = 0$, $r = 1$, $s = 1$ is calculated as follows (compare with Figure 2.2(b)):

$$\begin{aligned} f_{\varphi,\mathbb{P}'}(1,0,1,1) &= f_\leftrightarrow(f_\vee(1, f_\neg(0)), f_\rightarrow(1, f_\wedge(1,0))) \\ &= f_\leftrightarrow(f_\vee(1,1), f_\rightarrow(1,0)) \\ &= f_\leftrightarrow(1,0) \\ &= 0 \end{aligned}$$

Observation 2.2.4. *The truth function of a proposition φ over $\mathbb{P}$ depends only on the variables corresponding to the atomic propositions in* $\mathrm{Var}(\varphi) \subseteq \mathbb{P}$.

Thus, even if we have a proposition φ in an *infinite* language $\mathbb{P}$, we can restrict ourselves to the language $\mathrm{Var}(\varphi)$ (which is finite) and consider the truth function over this language.

2.2.3 Models

A given truth assignment of propositional variables is a representation of the 'real world' (system) in our chosen 'formal world,' hence it is also called a *model.*

Definition 2.2.5 (Model of a Language)**.** A *model* of a language $\mathbb{P}$ is any truth assignment $v\colon \mathbb{P} \rightarrow \{0,1\}$. The *set of (all) models of a language* $\mathbb{P}$ is denoted by $\mathrm{M}_\mathbb{P}$:

$$\mathrm{M}_\mathbb{P} = \{v \mid v\colon \mathbb{P} \rightarrow \{0,1\}\} = \{0,1\}^\mathbb{P}$$

Models will be denoted by letters v, u, w, etc. (v from the word 'valuation'). A model of a language is therefore a function, formally a set of pairs (input, output). For example, for the language $\mathbb{P} = \{p, q, r\}$ and the truth assignment where p is true, q false, and r true, we have the model

$$v = \{(p,1), (q,0), (r,1)\}.$$

For simplicity, we will write it as $v = (1,0,1)$. For the language $\mathbb{P} = \{p, q, r\}$, we have $2^3 = 8$ models:

$$\mathrm{M}_\mathbb{P} = \{(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)\}$$

Remark 2.2.6. Formally speaking, we identify the set $\{0,1\}^\mathbb{P}$ with the set $\{0,1\}^{|\mathbb{P}|}$ using the ordering ι of the language $\mathbb{P}$ (see Remark 2.1.1). Specifically, instead of the element $v = \{(p,1), (q,0), (r,1)\} \in \{0,1\}^\mathbb{P}$, we write $(1,0,1) = (v \circ \iota)(0,1,2) = (v(\iota(0)), v(\iota(1)), v(\iota(2))) \in \{0,1\}^{|\mathbb{P}|}$ (where we allow the functions v, ι to act 'component-wise').[11] If this seems confusing, imagine the model v as a set of atomic propositions that are true, i.e., $\{p, r\} \subseteq \mathbb{P}$, our notation $v = (1,0,1)$ is then the characteristic vector of this set. This identification will be used henceforth without further notice.

[11] Alternatively, we could require (at least for countable languages) that the language *be* $\mathbb{P} = \{0,1,2,\dots\}$ and use symbols p_0, p_1, p, q, r only for readability.

2.2.4 Validity

We are now ready to define the key concept of logic, *validity* of a proposition in a given model. Informally, a proposition is valid in a model (i.e., under a specific truth assignment of atomic propositions) if its truth value, as calculated in Example 2.2.1, equals 1. In the formal definition, we will use the truth function of the proposition (Definition 2.2.2).[12]

Definition 2.2.7 (Validity of a Proposition in a Model, Model of a Proposition)**.** Given a proposition φ in a language $\mathbb{P}$ and a model $v \in \mathrm{M}_{\mathbb{P}}$, if $f_{\varphi,\mathbb{P}}(v) = 1$, we say that the proposition φ is *valid* in the model v, v is a *model* of φ, and we write $v \models \varphi$. The set of all models of the proposition φ is denoted by $\mathrm{M}_{\mathbb{P}}(\varphi)$.

Models of a language that are not models of φ will sometimes be called *non-models* of φ. They form the complement of the set of models of φ. Using the standard notation for function inverse, we can write:

$$\begin{aligned} \mathrm{M}_{\mathbb{P}}(\varphi) &= \{v \in \mathrm{M}_{\mathbb{P}} \mid v \models \varphi\} = f^{-1}_{\varphi,\mathbb{P}}[1] \\ \overline{\mathrm{M}_{\mathbb{P}}(\varphi)} = \mathrm{M}_{\mathbb{P}} \setminus \mathrm{M}_{\mathbb{P}}(\varphi) &= \{v \in \mathrm{M}_{\mathbb{P}} \mid v \not\models \varphi\} = f^{-1}_{\varphi,\mathbb{P}}[0] \end{aligned}$$

If the language is clear from the context, we can simply write $\mathrm{M}(\varphi)$. We must be really sure, though: for example, in the language $\mathbb{P} = \{p, q\}$ we have

$$\mathrm{M}_{\{p,q\}}(p \to q) = \{(0,0),(0,1),(1,1)\},$$

while in the language $\mathbb{P}' = \{p, q, r\}$ we would have

$$\mathrm{M}_{\mathbb{P}'}(p \to q) = \{(0,0,0),(0,0,1),(0,1,0),(0,1,1),(1,1,0),(1,1,1)\}.$$

Definition 2.2.8 (Validity of a Theory, Model of a Theory)**.** Let T be a theory in a language $\mathbb{P}$. The theory T is *valid* in a model v if every axiom $\varphi \in T$ is valid in v. In this case, we also say that v is a *model* of T, and we write $v \models T$. The set of all models of the theory T in a language $\mathbb{P}$ is denoted by $\mathrm{M}_{\mathbb{P}}(T)$.

When dealing with a finite theory, or adding a finite number of new axioms to a theory, we will use the following simplified notation:

- $\mathrm{M}_{\mathbb{P}}(\varphi_1, \varphi_2, \ldots, \varphi_n)$ instead of $\mathrm{M}_{\mathbb{P}}(\{\varphi_1, \varphi_2, \ldots, \varphi_n\})$,
- $\mathrm{M}_{\mathbb{P}}(T, \varphi)$ instead of $\mathrm{M}_{\mathbb{P}}(T \cup \{\varphi\})$.

Note that $\mathrm{M}_{\mathbb{P}}(T, \varphi) = \mathrm{M}_{\mathbb{P}}(T) \cap \mathrm{M}_{\mathbb{P}}(\varphi)$, $\mathrm{M}_{\mathbb{P}}(T) = \bigcap_{\varphi \in T} \mathrm{M}_{\mathbb{P}}(\varphi)$, and for a finite theory (similarly for countable theories), we have

$$\mathrm{M}_{\mathbb{P}}(\varphi_1) \supseteq \mathrm{M}_{\mathbb{P}}(\varphi_1, \varphi_2) \supseteq \mathrm{M}_{\mathbb{P}}(\varphi_1, \varphi_2, \varphi_3) \supseteq \cdots \supseteq \mathrm{M}_{\mathbb{P}}(\varphi_1, \varphi_2, \ldots, \varphi_n).$$

We can use this when finding models by brute force.

Example 2.2.9. We can find models of the theory $T = \{p \vee q \vee r, q \to r, \neg r\}$ (in the language $\mathbb{P} = \{p, q, r\}$) as follows. First, we find the models of the proposition $\neg r$:

$$\mathrm{M}_{\mathbb{P}}(r) = \{(x, y, 0) \mid x, y \in \{0,1\}\} = \{(0,0,0),(0,1,0),(1,0,0),(1,1,0)\},$$

then we determine in which of these models the proposition $q \to r$ is valid:

[12]For *validity*, we use the symbol $\models$, which we read as 'satisfies' or 'models', in LaTeX as \models.

- $(0,0,0) \models q \rightarrow r$,
- $(0,1,0) \not\models q \rightarrow r$,
- $(1,0,0) \models q \rightarrow r$,
- $(1,1,0) \not\models q \rightarrow r$,

Thus $\mathrm{M}_{\mathbb{P}}(r, q \rightarrow r) = \{(0,0,0),(1,0,0)\}$. The proposition $p \vee q \vee r$ is valid only in the second of these models, so we get

$$\mathrm{M}_{\mathbb{P}}(r, q \rightarrow r, p \vee q \vee r) = \mathrm{M}_{\mathbb{P}}(T) = \{(1,0,0)\}.$$

This procedure is more efficient than determining the sets of models of the individual axioms and taking their intersection. (But much less efficient than using a formal proof system, such as the tableau method, which we will see later.)

2.2.5 Additional semantic notions

Following the notion of validity, we will use several other notions. For some properties, several different names are in use, depending on the context in which the property is discussed.

Definition 2.2.10 (Semantic notions)**.** We say that a proposition φ (in a language $\mathbb{P}$) is

- *true*, *tautology*, *valid (in logic/logically)*, and we write $\models \varphi$, if it is valid in every model (of the language $\mathbb{P}$), $\mathrm{M}_{\mathbb{P}}(\varphi) = \mathrm{M}_{\mathbb{P}}$,
- *false*, *contradictory*, if it has no model, $\mathrm{M}_{\mathbb{P}}(\varphi) = \emptyset$,[13]
- *independent*, if it is valid in some model, and it is not valid in some other model, i.e., it is neither true nor false, $\emptyset \subsetneq \mathrm{M}_{\mathbb{P}}(\varphi) \subsetneq \mathrm{M}_{\mathbb{P}}$,
- *satisfiable*, if it has some model, i.e., it is not false, $\mathrm{M}_{\mathbb{P}}(\varphi) \neq \emptyset$.

Furthermore, we say that propositions φ, ψ (in the same language $\mathbb{P}$) are *(logically) equivalent*, we write $\varphi \sim \psi$, if they have the same models, i.e.,

$$\varphi \sim \psi \text{ if and only if } \mathrm{M}_{\mathbb{P}}(\varphi) = \mathrm{M}_{\mathbb{P}}(\psi).$$

Example 2.2.11. For example, the following hold:

- propositions $\top$, $p \vee q \leftrightarrow q \vee p$ are true,
- propositions $\bot$, $(p \vee q) \wedge (p \vee \neg q) \wedge \neg p$ are false,
- propositions $p, p \wedge q$ are independent, and also satisfiable, and
- the following propositions are equivalent:
  - $p \sim p \vee p \sim p \vee p \vee p$,
  - $p \rightarrow q \sim \neg p \vee q$,

[13]Note that being *false* is not the same as not being *true*!

– $\neg p \to (p \to q) \sim \top$.

The notions from Definition 2.2.10 can also be relativized with respect to a given theory. This means that we restrict the individual definitions to the models of this theory:

Definition 2.2.12 (Semantic Notions Relative to a Theory)**.** Let T be a theory in a language $\mathbb{P}$. We say that a proposition φ in the language $\mathbb{P}$ is

- *true in* T, a *consequence of* T, *valid in* T, and we write $T \models \varphi$, if φ is valid in every model of the theory T, i.e., $\mathrm{M}_\mathbb{P}(T) \subseteq \mathrm{M}_\mathbb{P}(\varphi)$,
- *false in* T, *contradictory in* T, if it is not valid in any model of T, i.e., $\mathrm{M}_\mathbb{P}(\varphi) \cap \mathrm{M}_\mathbb{P}(T) = \mathrm{M}_\mathbb{P}(T, \varphi) = \emptyset$.
- *independent in* T, if it is valid in some model of T, and not valid in some other model of T, i.e., it is neither true in T nor false in T, $\emptyset \subsetneq \mathrm{M}_\mathbb{P}(T, \varphi) \subsetneq \mathrm{M}_\mathbb{P}(T)$,
- *satisfiable in* T, *consistent with* T, if it is valid in some model of T, i.e., it is not false in T, $\mathrm{M}_\mathbb{P}(T, \varphi) \neq \emptyset$.

And we say that propositions φ, ψ (in the same language $\mathbb{P}$) are *equivalent in* T, T*-equivalent*, we write $\varphi \sim_T \psi$ if they hold in the same models of T, i.e.,

$$\varphi \sim_T \psi \text{ if and only if } \mathrm{M}_\mathbb{P}(T, \varphi) = \mathrm{M}_\mathbb{P}(T, \psi).$$

Note that for the empty theory $T = \emptyset$, we have $\mathrm{M}_\mathbb{P}(T) = \mathrm{M}_\mathbb{P}$ and the above concepts for T coincide with the original ones. Again, we illustrate the concepts with several examples:

Example 2.2.13. Let $T = \{p \vee q, \neg r\}$. The following hold:

- propositions $q \vee p$, $\neg p \vee \neg q \vee \neg r$ are true in T,
- the proposition $(\neg p \wedge \neg q) \vee r$ is false in T,
- propositions $p \leftrightarrow q, p \wedge q$ are independent in T, and also satisfiable, and
- p and $p \vee r$ are T-equivalent, $p \sim_T p \vee r$ (but $p \not\sim p \vee r$).

2.2.6 Universality of Logical Connectives

In the language of propositional logic, we use the following logical connectives: $\neg, \wedge, \vee, \to, \leftrightarrow$. This is not the only possible choice; to build a fully-fledged logic, we could make do, for example, only with negation and implication,[14] or negation, conjunction, and disjunction.[15] And as we will see below, we could use other logical connectives as well. Our choice is the golden middle path between expressiveness on the one hand and succintness of syntactic definitions and proofs on the other.

What do we mean by saying that logic is fully-fledged? We say that a set of logical connectives S is *universal*[16] if any Boolean function f can be expressed as the truth function $f_{\varphi,\mathbb{P}}$ of some proposition φ built from the logical connectives in S (where $|\mathbb{P}| = n$ if f is

[14]Negation is needed to describe the state of a system, and implication to describe behavior over time.
[15]These are sufficient to build logical circuits.
[16]Some people would say *[functionally] complete.*

an n-ary function). Equivalently, for any finite language $\mathbb{P}$ (say n-element) and any set of models $M \subseteq \mathrm{M}_{\mathbb{P}}$, there must exist a proposition φ such that $\mathrm{M}_{\mathbb{P}}(\varphi) = M$. (The equivalence of these two statements follows from the fact that if we have a Boolean function f and choose $M = f^{-1}[1]$, then $f_{\varphi,\mathbb{P}} = f$ if and only if $\mathrm{M}_{\mathbb{P}}(\varphi) = M$.)

Proposition 2.2.14. *The sets of logical connectives $\{\neg, \wedge, \vee\}$ and $\{\neg, \rightarrow\}$ are universal.*

Proof. Let us have a function $f\colon \{0,1\}^n \to \{0,1\}$, or equivalently a set of models $M = f^{-1}[1] \subseteq \{0,1\}^n$. Our language will be $\mathbb{P} = \{p_1, \dots, p_n\}$. If the set M contained only one model, say $v = (1,0,1,0)$, we could represent it with the proposition $\varphi_v = p_1 \wedge \neg p_2 \wedge p_3 \wedge \neg p_4$, which says 'I must be the model v.' For a general model v, we would write the proposition φ_v as follows:

$$\varphi_v = p_1^{v_1} \wedge p_2^{v_2} \wedge \cdots \wedge p_n^{v_n} = \bigwedge_{i=1}^{n} p_i^{v(p_i)} = \bigwedge_{p \in \mathbb{P}} p^{v(p)},$$

where we introduce the following useful notation: $p^{v(p)}$ is the proposition p if $v(p) = 1$, and the proposition $\neg p$ if $v(p) = 0$.

If the set M contains more models, we say 'I must be at least one of the models from M':

$$\varphi_M = \bigvee_{v \in M} \varphi_v = \bigvee_{v \in M} \bigwedge_{p \in \mathbb{P}} p^{v(p)}$$

Clearly, $\mathrm{M}_{\mathbb{P}}(\varphi_M) = M$, or equivalently $f_{\varphi_M,\mathbb{P}} = f$. (If $M = \emptyset$, then by definition $\bigvee_{v \in M} \varphi_v = \bot$.)[17]

The universality of $\{\neg, \rightarrow\}$ follows from the universality of $\{\neg, \wedge, \vee\}$ and the fact that conjunction and disjunction can be expressed using negation and implication: $p \wedge q \sim \neg(p \rightarrow \neg q)$ and $p \vee q \sim \neg p \rightarrow q$. $\square$

Remark 2.2.15. Note that in the construction of the proposition φ_M, it is crucial that the set M is finite (it has at most 2^n elements). If it were infinite, the symbol '$\bigvee_{v \in M}$' would mean 'disjunction' of infinitely many propositions, and thus the result would not be a finite string, i.e., 'φ_M' would not be a proposition. (If we have a countably infinite language $\mathbb{P}'$, then not every subset $M \subseteq \mathrm{M}_{\mathbb{P}'}$ can be represented by a proposition—there are uncountably many such subsets, while propositions are only countably many.)

What other logical connectives could we use? Nullary Boolean functions,[18] i.e. constants 0, 1, could be introduced as symbols TRUE and FALSE; we will suffice with propositions $\top, \bot$. There are four unary Boolean functions ($4 = 2^{2^1}$), but negation is the only 'interesting' one: the others are $f(x) = x$, $f(x) = 0$, and $f(x) = 1$. There are more interesting binary logical connectives that occur naturally, for example:

- NAND or *Sheffer's stroke*, sometimes denoted as $p \uparrow q$, where $p \uparrow q \sim \neg(p \wedge q)$,
- NOR or *Peirce's arrow*, sometimes denoted as $p \downarrow q$, where $p \downarrow q \sim \neg(p \vee q)$,
- XOR, or *exclusive-OR*, sometimes denoted as $\oplus$, where $p \oplus q \sim (p \vee q) \wedge \neg(p \wedge q)$, i.e. the sum of truth values modulo 2.

[17] Similar to how the sum of an empty set of summands equals 0.

[18] In formalizing mathematics or computer science, a function of arity 0 means it has no inputs, so the output cannot depend on the input and is constant. Formally, these are functions $f\colon \emptyset \to \{0,1\}$. If this is confusing, assume that functions must have an arity of at least 1, and instead of 'nullary function', say 'constant.'

Exercise 2.2. Express $(p \oplus q) \oplus r$ using $\{\neg, \wedge, \vee\}$.

Exercise 2.3. Show that {NAND} and {NOR} are universal.

Exercise 2.4. Consider the ternary logical connective IFTE, where $IFTE(p, q, r)$ is satisfied if and only if 'if p then q else r'. Determine the truth table of this logical connective (i.e., the function f_{IFTE}) and show that $\{\text{TRUE}, \text{FALSE}, \text{IFTE}\}$ is universal.

2.3 Normal Forms

Let us recall that propositions are equivalent if they have the same set of models. For each proposition, there exist infinitely many equivalent propositions; it is often useful to express a proposition in a 'nice' (useful) 'shape', i.e., to find an equivalent proposition of that 'shape'. This concept of 'shape' in mathematics is called a *normal form.* We will introduce two most common normal forms: *conjunctive normal form (CNF)* and *disjunctive normal form (DNF).*

The following terminology and notation are needed:

- A *literal* ℓ is either a propositional variable p or the negation of a propositional variable $\neg p$. For a propositional variable p, denote $p^0 = \neg p$ and $p^1 = p$. If ℓ is a literal, then $\bar{\ell}$ denotes the *opposite literal* to ℓ. If $\ell = p$ (a *positive literal*), then $\bar{\ell} = \neg p$; if $\ell = \neg p$ (a *negative literal*), then $\bar{\ell} = p$.
- A *clause* is a disjunction of literals $C = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_n$. A *unit clause* is a single literal ($n = 1$) and the *empty clause* ($n = 0$) is interpreted as $\bot$.
- A proposition is in *conjunctive normal form (CNF)* if it is a conjunction of clauses. The *empty CNF proposition* is $\top$.
- An *elementary conjunction* is a conjunction of literals $E = \ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_n$. A *unit elementary conjunction* is a single literal ($n = 1$). The *empty elementary conjunction* ($n = 0$) is $\top$.
- A proposition is in *disjunctive normal form (DNF)* if it is a disjunction of elementary conjunctions. The *empty DNF proposition* is $\bot$.

Example 2.3.1. The proposition $p \vee q \vee \neg r$ is in CNF (it is a single clause) as well as in DNF (it is a disjunction of unit elementary conjunctions). The proposition $(p \vee q) \wedge (p \vee \neg q) \wedge \neg p$ is in CNF, while the proposition $\neg p \vee (p \wedge q)$ is in DNF.

Example 2.3.2. The proposition φ_v from the proof of Proposition 2.2.14 is in CNF (it is a conjunction of unit clauses, i.e., literals) and also in DNF (it is a single elementary conjunction). The proposition φ_M is in DNF.

Observation 2.3.3. *Note that a proposition in CNF is a tautology if and only if each of its clauses contains a pair of opposite literals. Similarly, a proposition in DNF is satisfiable if and only if not every elementary conjunction contains a pair of opposite literals.*

2.3.1 On Duality

Note that if we interchange the values for truth and falsehood in propositional logic, i.e., 0 and 1, the truth table for negation remains the same, while conjunction becomes disjunction, and vice versa. This concept is called *duality*; we will see many examples of this in logic.

We have $\neg(p \wedge q) \sim (\neg p \vee \neg q)$, and from *duality*, we also know that $\neg(\neg p \vee \neg q) \sim (\neg\neg p \wedge \neg\neg q)$, from which we can easily deduce $\neg(p \vee q) \sim (\neg p \wedge \neg q)$.[19] More generally, n-ary Boolean functions f, g are *dual* to each other if $f(\neg x) = \neg g(x)$. If we have a proposition φ built from $\{\neg, \wedge, \vee\}$ and we interchange $\wedge$ and $\vee$, and negate the propositional variables (i.e., interchange literals with their opposite literals), we obtain a proposition $\psi \sim \neg\varphi$ (i.e., the models of φ are the non-models of ψ and vice versa), and the functions $f_{\varphi,\mathbb{P}}$ and $f_{\psi,\mathbb{P}}$ are dual to each other.

The notion of DNF is dual to the notion of CNF; 'is a tautology' is dual to 'is not satisfiable', thus the previous observation can be understood as an example of duality. For every statement in propositional logic, we obtain a *dual* statement 'for free', resulting from the interchange of $\wedge$ and $\vee$, truth and falsehood.

2.3.2 Conversion to Normal Forms

We have already encountered the disjunctive normal form in the proof of Proposition 2.2.14. The key part of the proof can be formulated as follows: 'If the language is finite, any set of models can be *axiomatized* by a proposition in DNF'. From duality, we also obtain axiomatization in CNF since the complement of a set of models is also a set of models:

Proposition 2.3.4. *Given a finite language $\mathbb{P}$ and any set of models $M \subseteq \mathrm{M}_{\mathbb{P}}$, there exists a proposition φ_{DNF} in DNF and a proposition φ_{CNF} in CNF such that $M = \mathrm{M}_{\mathbb{P}}(\varphi_{\mathrm{DNF}}) = \mathrm{M}_{\mathbb{P}}(\varphi_{\mathrm{CNF}})$. Specifically:*

$$\varphi_{\mathrm{DNF}} = \bigvee_{v \in M} \bigwedge_{p \in \mathbb{P}} p^{v(p)}$$

$$\varphi_{\mathrm{CNF}} = \bigwedge_{v \in \overline{M}} \bigvee_{p \in \mathbb{P}} \overline{p^{v(p)}} = \bigwedge_{v \notin M} \bigvee_{p \in \mathbb{P}} p^{1-v(p)}$$

Proof. For the proposition φ_{DNF}, see the proof of Proposition 2.2.14, where each elementary conjunction describes one model. The proposition φ_{CNF} is dual to the proposition φ'_{DNF} constructed for the complement $M' = \overline{M}$. Alternatively, we can prove it directly: the models of the clause $C_v = \bigvee_{p \in \mathbb{P}} p^{1-v(p)}$ are all models except v, $\mathrm{M}_C = \mathrm{M}_{\mathbb{P}} \setminus \{v\}$, so each clause in the conjunction excludes one non-model. □

Proposition 2.3.4 provides a method for converting a proposition to disjunctive or conjunctive normal form:

Example 2.3.5. Consider the proposition $\varphi = p \leftrightarrow (q \vee \neg r)$. First, we find the set of models: $M = \mathrm{M}(\varphi) = \{(0,0,1), (1,0,0), (1,1,0), (1,1,1)\}$. Then, we find the propositions φ_{DNF} and φ_{CNF} according to Proposition 2.3.4. Those have the same models as φ and are therefore equivalent with it.

We find the proposition φ_{DNF} by constructing an elementary conjunction for each model that enforces precisely that model:

$$\varphi_{\mathrm{DNF}} = (\neg p \wedge \neg q \wedge r) \vee (p \wedge \neg q \wedge \neg r) \vee (p \wedge q \wedge \neg r) \vee (p \wedge q \wedge r)$$

For constructing φ_{CNF}, we need the *non-models* of φ, $\overline{M} = \{(0,0,0), (0,1,0), (0,1,1), (1,0,1)\}$. Each clause excludes one non-model:

$$\varphi_{\mathrm{CNF}} = (p \vee q \vee r) \wedge (p \vee \neg q \vee r) \wedge (p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee \neg r)$$

[19] Since p, q are propositional variables, they can be assigned both values 0 and 1, thus we can interchange them with their opposite literals.

Corollary 2.3.6. *Every proposition (in any, even infinite, language $\mathbb{P}$) is equivalent to some proposition in CNF and some proposition in DNF.*

Proof. Even if the language $\mathbb{P}$ is infinite, the proposition φ contains only finitely many propositional variables, so we can use Proposition 2.3.4 for the language $\mathbb{P}' = \mathrm{Var}(\varphi)$ and the set of models $M = \mathrm{M}_{\mathbb{P}'}(\varphi)$. Since $M = \mathrm{M}_{\mathbb{P}'}(\varphi_{\mathrm{DNF}}) = \mathrm{M}_{\mathbb{P}'}(\varphi_{\mathrm{CNF}})$, we have $\varphi \sim \varphi_{\mathrm{DNF}} \sim \varphi_{\mathrm{CNF}}$. $\square$

Exercise 2.5. Describe how one can easily generate models from a DNF proposition and non-models from a CNF proposition.

Remark 2.3.7. When can a *theory* be axiomatized by a proposition in DNF or CNF? Consider the language $\mathbb{P}' = \mathrm{Var}(T)$ (i.e., all propositional variables occurring in the axioms of T). If T in the language $\mathbb{P}'$ has finitely many models (i.e., $\mathrm{M}_{\mathbb{P}'}(T)$ is finite), we can construct a proposition in DNF, and if it has finitely many *non-models*, we can construct a proposition in CNF. Generally, however, not every theory can be axiomatized by a *single* proposition in CNF or DNF. We can always convert individual axioms to CNF (or DNF), and we can also axiomatize the theory using (potentially infinitely many) clauses.

This method of conversion to CNF or DNF requires knowledge of the set of models of the proposition, so it is quite inefficient. Additionally, the resulting normal form can be very long. We will now show another method.

Conversion Using Equivalent Transformations

We use the following observation: If we replace some subproposition ψ of a proposition φ with an equivalent proposition ψ', the resulting proposition φ' will also be equivalent to φ. We will first demonstrate the method on an example:

Example 2.3.8. Again, we will convert the proposition $\varphi = p \leftrightarrow (q \vee \neg r)$. First, we eliminate the equivalence, expressing it as a conjunction of two implications. In the next step, we remove the implications using the rule $\varphi \rightarrow \psi \sim \neg\varphi \vee \psi$:

$$\begin{aligned} p \leftrightarrow (q \vee \neg r) &\sim (p \rightarrow (q \vee \neg r)) \wedge ((q \vee \neg r) \rightarrow p) \\ &\sim (\neg p \vee q \vee \neg r) \wedge (\neg(q \vee \neg r) \vee p) \end{aligned}$$

Now, imagine the tree of the proposition; in the next step, we want to push the negations as low as possible in the tree, just above the leaves: we use $\neg(q \vee \neg r) \sim \neg q \wedge \neg\neg r$ and eliminate the double negation $\neg\neg r \sim r$. We obtain the proposition

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

At this point, we leave the literals untouched and apply the distributivity of $\wedge$ over $\vee$, or vice versa, depending on whether we want DNF or CNF. For conversion to CNF, we use the transformation $(\neg q \wedge r) \vee p \sim (\neg q \vee p) \wedge (r \vee p)$, which pushes the $\vee$ symbol lower in the tree. (Draw the tree!) This gives us a proposition in CNF; for better readability, we sort the literals in the clauses:

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

For conversion to DNF, we proceed similarly by repeatedly applying distributivity. Here, we start from the CNF form and combine each literal from the first clause with each literal from the second and each literal from the third clause. Note that the same literal does not need to be repeated twice in the elementary conjunction, and if the elementary conjunction contains a pair of opposite literals, it is contradictory and can be omitted in the DNF. We can also omit an elementary conjunction E if we have another elementary conjunction E' such that E' contains all the literals contained in E, e.g., $E = (p \land \neg r)$ and $E' = (p \land q \land \neg r)$. (Think about why, and formulate the dual simplification when converting to CNF.) The resulting proposition in DNF is:

$$(\neg p \land \neg q \land r) \lor (p \land q \land r) \lor (p \land \neg r)$$

We now list all equivalent transformations needed for the method. The proof that every proposition can be converted to DNF and CNF can be easily carried out by induction on the structure of the proposition (depth of the proposition tree).

- Implications and equivalences:

$$\varphi \to \psi \sim \neg\varphi \lor \psi$$
$$\varphi \leftrightarrow \psi \sim (\neg\varphi \lor \psi) \land (\neg\psi \lor \varphi)$$

- Negations:

$$\neg(\varphi \land \psi) \sim \neg\varphi \lor \neg\psi$$
$$\neg(\varphi \lor \psi) \sim \neg\varphi \land \neg\psi$$
$$\neg\neg\varphi \sim \varphi$$

- Conjunctions (conversion to DNF):

$$\varphi \land (\psi \lor \chi) \sim (\varphi \land \psi) \lor (\varphi \land \chi)$$
$$(\varphi \lor \psi) \land \chi \sim (\varphi \land \chi) \lor (\psi \land \chi)$$

- Disjunctions (conversion to CNF):

$$\varphi \lor (\psi \land \chi) \sim (\varphi \lor \psi) \land (\varphi \lor \chi)$$
$$(\varphi \land \psi) \lor \chi \sim (\varphi \lor \chi) \land (\psi \lor \chi)$$

As we will see in the next chapter, CNF is much more important in practice than DNF (even though they are dual concepts). For describing a real system, it is more natural to express it as a conjunction of many simpler properties than as a single very long disjunction. There are many other forms of representing Boolean functions. Similar to data structures, we choose the appropriate form of representation depending on the operations we need to perform on the function.[20]

2.4 Properties and Consequences of Theories

Let us delve deeper into properties of theories. Similar to propositions, we say that two theories T, T' in the language $\mathbb{P}$ are *equivalent* if they have the same set of models:

$$T \sim T' \text{ if and only if } \mathrm{M}_{\mathbb{P}}(T) = \mathrm{M}_{\mathbb{P}}(T')$$

These are theories expressing the same properties of models, just formulated (*axiomatized*) differently. In logic, we are mainly interested in properties of theories that are independent of the specific *axiomatization*.

Example 2.4.1. For example, the theory $T = \{p \to q, p \leftrightarrow r\}$ is equivalent to the theory $T' = \{(\neg p \lor q) \land (\neg p \lor r) \land (p \lor \neg r)\}$.

Definition 2.4.2 (Properties of Theories)**.** We say that a theory T in the language $\mathbb{P}$ is

- *inconsistent* (*unsatisfiable*), if it includes $\bot$ (a contradiction), equivalently, if it has no model, equivalently, if it includes all propositions,

[20]See, for example, the course NAIL031 Representations of Boolean Functions.

- *consistent* (*satisfiable*) if it is not inconsistent, i.e., it has some model,
- *complete* if it is not inconsistent and every proposition in it is either true or false (i.e., it has no independent propositions), equivalently, if it has exactly one model.

Let us verify that the equivalence of properties in the definition holds. Notice that in an inconsistent theory, all propositions are indeed valid! A proposition is valid in T if it is valid in every model of T, but there are none. Conversely, if the theory has at least one model, $\bot = p \wedge \neg p$ cannot be valid in this model.

If a theory is complete, it cannot have two different models $v \neq v'$. The proposition $\varphi_v = \bigwedge_{p\in\mathbb{P}} p^{v(p)}$ (which we encountered in the proof of Proposition 2.2.14) would be independent in T because it is valid in the model v but not in v'. Conversely, if T has a single model v, then every proposition is either valid in v and thus in T, or it is not valid in v and therefore false in T.

Example 2.4.3. An example of an inconsistent theory is $T_1 = \{p, p \to q, \neg q\}$. The theory $T_2 = \{p \vee q, r\}$ is consistent but not complete, for instance, the proposition $p \wedge q$ is neither true (it does not hold in the model $(1,0,1)$) nor false (it holds in the model $(1,1,1)$). The theory $T_2 \cup \{\neg p\}$ is complete, with its only model being $(0,1,1)$.

2.4.1 Consequences of Theories

Recall that a consequence of a theory T is any proposition that is true in T (i.e., valid in every model of T), and let us denote the *set of all consequences* of a theory T in the language $\mathbb{P}$ as

$$\mathrm{Csq}_{\mathbb{P}}(T) = \{\varphi \in \mathrm{PF}_{\mathbb{P}} \mid T \models \varphi\}$$

If the theory T is in the language $\mathbb{P}$, we can write:

$$\mathrm{Csq}_{\mathbb{P}}(T) = \{\varphi \in \mathrm{PF}_{\mathbb{P}} \mid \mathrm{M}_{\mathbb{P}}(T) \subseteq \mathrm{M}_{\mathbb{P}}(\varphi)\}$$

(However, it is also meaningful to talk about consequences of a theory that are in some smaller language, which is a subset of the language of T).

Let us demonstrate some simple properties of consequences:

Proposition 2.4.4. *Let T, T' be theories and $\varphi, \varphi_1, \dots, \varphi_n$ be propositions in a language $\mathbb{P}$. Then the following holds:*

(i) $T \subseteq \mathrm{Csq}_{\mathbb{P}}(T)$,

(ii) $\mathrm{Csq}_{\mathbb{P}}(T) = \mathrm{Csq}_{\mathbb{P}}(\mathrm{Csq}_{\mathbb{P}}(T))$,

(iii) if $T \subseteq T'$*, then* $\mathrm{Csq}_{\mathbb{P}}(T) \subseteq \mathrm{Csq}_{\mathbb{P}}(T')$,

(iv) $\varphi \in \mathrm{Csq}_{\mathbb{P}}(\{\varphi_1, \dots, \varphi_n\})$ *if and only if the proposition* $(\varphi_1 \wedge \dots \wedge \varphi_n) \to \varphi$ *is a tautology.*

Proof. The proof is straightforward, using the fact that φ is a consequence of T if and only if $\mathrm{M}_{\mathbb{P}}(T) \subseteq \mathrm{M}_{\mathbb{P}}(\varphi)$, and recognizing the following relationships:

- $\mathrm{M}(\mathrm{Csq}(T)) = \mathrm{M}(T)$,
- if $T \subseteq T'$, then $\mathrm{M}(T) \supseteq \mathrm{M}(T')$,[21]

[21]The more properties we prescribe, the fewer objects will satisfy them all.

- $\psi \to \varphi$ is a tautology if and only if $\mathrm{M}(\psi) \subseteq \mathrm{M}(\varphi)$,
- $\mathrm{M}(\varphi_1 \wedge \cdots \wedge \varphi_n) = \mathrm{M}(\varphi_1, \ldots, \varphi_n)$.

□

Exercise 2.6. Give a detailed proof of Proposition 2.4.4.

2.4.2 Extensions of Theories

Informally speaking, an *extension* of a theory T refers to any theory T' that satisfies everything that is true in T (and more, apart from the trivial case). If theory T models some system, it can be extended in two ways: by adding additional requirements about the system (which we will call a *simple extension*) or by expanding the system with some new parts. If in the second case there are no additional requirements on the original part of the system, i.e., exactly the same hold about the original part as before, we say that the extension is *conservative.*

Example 2.4.5. Let us return to the initial example of graph coloring, Example 1.1.2. The theory T_3 (complete colorings satisfying the edge condition) is a simple extension of the theory T_1 (partial colorings of vertices with no regard for edges). The theory T_3' from Section 1.2.1 (adding a new vertex to the graph) is a conservative, non-simple extension of T_3. And it is an extension of T_1 that is neither simple nor conservative.

Let us now finally present the formal definitions:

Definition 2.4.6 (Extension of a Theory)**.** Let T be a theory in the language $\mathbb{P}$.

- An *extension* of the theory T is any theory T' in a language $\mathbb{P}' \supseteq \mathbb{P}$ that satisfies $\mathrm{Csq}_{\mathbb{P}}(T) \subseteq \mathrm{Csq}_{\mathbb{P}'}(T')$,
- it is a *simple extension* if $\mathbb{P}' = \mathbb{P}$,
- it is a *conservative extension* if $\mathrm{Csq}_{\mathbb{P}}(T) = \mathrm{Csq}_{\mathbb{P}}(T') = \mathrm{Csq}_{\mathbb{P}'}(T') \cap \mathrm{PF}_{\mathbb{P}}$.

Thus, an extension means that it satisfies all the consequences of the original theory. An extension is simple if we do not add any new propositional variables to the language, and it is conservative if it does not change the validity of statements expressible in the original language, meaning that any new consequence must contain some of the newly added propositional variables.

What do these concepts mean *semantically*, in terms of models? Let us first formulate a general observation, which we will then illustrate with an example:

Observation 2.4.7. *Let T be a theory in the language $\mathbb{P}$, and T' a theory in a language $\mathbb{P}'$ containing the language $\mathbb{P}$. Then the following holds:*

- *T' is a simple extension of T if and only if $\mathbb{P}' = \mathbb{P}$ and $\mathrm{M}_{\mathbb{P}}(T') \subseteq \mathrm{M}_{\mathbb{P}}(T)$,*
- *T' is an extension of T if and only if $\mathrm{M}_{\mathbb{P}'}(T') \subseteq \mathrm{M}_{\mathbb{P}'}(T)$. We thus consider models of theory T in the extended language $\mathbb{P}'$.*[22] *In other words, the* restriction[23] *of any*

[22]Note that we cannot write $\mathrm{M}_{\mathbb{P}}(T')$ because models of T' must be truth assignments of the larger language $\mathbb{P}'$, and values only for the propositional variables in $\mathbb{P}$ are not sufficient to determine the truth value of axioms of T'. And we cannot write $\mathrm{M}_{\mathbb{P}'}(T') \subseteq \mathrm{M}_{\mathbb{P}}(T)$ either, as these are sets of vectors of different dimensions.

[23]*Restriction* means forgetting the values for the new propositional variables, i.e., deleting the corresponding coordinates in the vector representation of the model.

model $v \in \mathrm{M}_{\mathbb{P}'}(T')$ to the original language $\mathbb{P}$ must be a model of T. We could write $v{\restriction}_{\mathbb{P}} \in \mathrm{M}_{\mathbb{P}}(T)$ or:

$$\{v{\restriction}_{\mathbb{P}} \mid v \in \mathrm{M}_{\mathbb{P}'}(T')\} \subseteq \mathrm{M}_{\mathbb{P}}(T)$$

- *T' is a conservative extension of T if it is an extension and, moreover, every model of T (in the language $\mathbb{P}$) can be* expanded[24] *(in some way, not necessarily uniquely) to a model of T' (in the language $\mathbb{P}'$), in other words,* every *model of T (in the language $\mathbb{P}$) is obtained by restricting* some *model of T' to the language $\mathbb{P}$. We could write:*

$$\{v{\restriction}_{\mathbb{P}} \mid v \in \mathrm{M}_{\mathbb{P}'}(T')\} = \mathrm{M}_{\mathbb{P}}(T)$$

- *T' is an extension of T and T is an extension of T', if and only if $\mathbb{P}' = \mathbb{P}$ and $\mathrm{M}_{\mathbb{P}}(T') = \mathrm{M}_{\mathbb{P}}(T)$, in other words, $T' \sim T$.*

- *Complete simple extensions of T correspond to models of T, uniquely up to equivalence.*

Example 2.4.8. Consider the theory $T = \{p \to q\}$ in the language $\mathbb{P} = \{p, q\}$. The theory $T_1 = \{p \wedge q\}$ in the language $\mathbb{P}$ is a simple extension of T, we have $\mathrm{M}_{\mathbb{P}}(T_1) = \{(1,1)\} \subseteq \{(0,0),(0,1),(1,1)\} = \mathrm{M}_{\mathbb{P}}(T)$. It is a complete theory, and other complete simple extensions of theory T are, for example, $T_2 = \{\neg p, q\}$ and $T_3 = \{\neg p, \neg q\}$. Each complete simple extension of theory T is equivalent to T_1, T_2, or T_3.

Now consider the theory $T' = \{p \leftrightarrow (q \wedge r)\}$ in the language $\mathbb{P}' = \{p, q, r\}$. It is an extension of T because $\mathbb{P} = \{p, q\} \subseteq \{p, q, r\} = \mathbb{P}'$ and:

$$\begin{aligned}\mathrm{M}_{\mathbb{P}'}(T') &= \{(0,0,0),(0,0,1),(0,1,0),(1,1,1)\}\\ &\subseteq \{(0,0,0),(0,0,1),(0,1,0),(0,1,1),(1,1,0),(1,1,1)\} = \mathrm{M}_{\mathbb{P}'}(T)\end{aligned}$$

In other words, by restricting the models of T' to the language $\mathbb{P}$, we get $\{(0,0),(0,1),(1,1)\}$, which is a subset of $\mathrm{M}_{\mathbb{P}}(T)$.

Because $\{(0,0),(0,1),(1,1)\} = \mathrm{M}_{\mathbb{P}}(T)$, in other words, every model $v \in \mathrm{M}_{\mathbb{P}}(T)$ can be expanded to a model $v' \in \mathrm{M}_{\mathbb{P}'}(T')$ (e.g., $(0,1)$ can be expanded by defining $v'(r) = 0$ to the model $(0,1,0)$), T' is even a conservative extension of T. This means that every proposition in the language $\mathbb{P}$ is valid in T if and only if it is valid in T'. But the proposition $p \to r$ (which is in the language $\mathbb{P}'$, but not in the language $\mathbb{P}$) is a new consequence: it is valid in T' but not in T (see the model $(1,1,0)$).

The theory $T'' = \{\neg p \vee q, \neg q \vee r, \neg r \vee p\}$ in the language $\mathbb{P}'$ is an extension of T but not a conservative extension because $p \leftrightarrow q$ is valid in T'', but not in T. Or because the model $(0,1)$ of the theory T cannot be extended to a model of T'': neither $(0,1,0)$ nor $(0,1,1)$ satisfy the axioms of T''.

The theory T is a (simple) extension of the theory $\{\neg p \vee q\}$ in the language $\mathbb{P}$, and vice versa: $T \sim \{\neg p \vee q\}$. It is also, like any theory, a simple conservative extension of itself.

Exercise 2.7. Show (in detail) that if a theory T has a complete conservative extension, then it must itself be complete.

[24] By adding values for the new propositional variables, i.e., adding the corresponding coordinates in the vector representation.

2.5 Algebra of Propositions

In logic, we are usually[25] interested in propositions (or theories) *up to equivalence.*[26] The correct answer to the question 'How many different propositions are there in the language $\mathbb{P} = \{p, q, r\}$?' is 'Infinitely many'. However, we are probably interested in propositions *up to equivalence* (in other words, *mutually inequivalent*). There are as many of these as there are different subsets of models of the language, which is $2^{|\,\mathrm{M}_{\mathbb{P}}\,|} = 2^8 = 256$. Indeed, if two propositions have the same set of models, they are equivalent by definition. And for each set of models, we can find a corresponding proposition, e.g., in DNF (see 2.3.4). Let us consider slightly more complex reasoning:

Example 2.5.1. Let T be a theory in the language $\mathbb{P} = \{p, q, r\}$ having exactly five models. How many propositions over $\mathbb{P}$ (up to equivalence) are independent in the theory T? Let $|\mathbb{P}| = n = 3$ and $|\,\mathrm{M}_{\mathbb{P}}(T)| = k = 5$.

We count the sets $M = \mathrm{M}_{\mathbb{P}}(\varphi)$ and require that $\emptyset \neq M \cap \mathrm{M}_{\mathbb{P}}(T) \neq \mathrm{M}_{\mathbb{P}}(T)$. Thus, we have a total of $2^k - 2 = 30$ possibilities for what the set $M \cap \mathrm{M}_{\mathbb{P}}(T)$ may be. And for each model of the language that is not a model of T (there are $2^n - k = 3$ of these), we can choose arbitrarily whether or not it will be in M. Altogether, we get $(2^k - 2) \cdot 2^{2^n - k} = 30 \cdot 2^{8-5} = 240$ possible sets M. Hence that is the number of propositions independent in T, up to equivalence.

Let us explore the matter on a more abstract level. Formally, we consider the set of equivalence classes of $\sim$ on the set of all propositions $\mathrm{PF}_{\mathbb{P}}$, which we denote as ${}^{\mathrm{PF}_{\mathbb{P}}}/_{\sim}$. The elements of this set are sets of equivalent propositions, e.g., $[p \to q]_\sim = \{p \to q, \neg p \vee q, \neg(p \wedge \neg q), \neg p \vee q \vee q, \dots\}$. And we have a mapping $h : {}^{\mathrm{PF}_{\mathbb{P}}}/_{\sim} \to \mathcal{P}(\mathrm{M}_{\mathbb{P}})$ (where $\mathcal{P}(X)$ is the set of all subsets of X) defined by:

$$h([\varphi]_\sim) = \mathrm{M}(\varphi)$$

That is, to an equivalence class of propositions we assign the set of models of any member of that class. It is easy to verify that this mapping is well-defined (the definition does not depend on the choice of proposition from the class), injective, and if the language $\mathbb{P}$ is finite, then h is even bijective. (Do verify this!)

On the set ${}^{\mathrm{PF}_{\mathbb{P}}}/_{\sim}$, we can define operations $\neg, \wedge, \vee$ by:

$$\begin{aligned}\neg[\varphi]_\sim &= [\neg\varphi]_\sim\\ [\varphi]_\sim \wedge [\psi]_\sim &= [\varphi \wedge \psi]_\sim\\ [\varphi]_\sim \vee [\psi]_\sim &= [\varphi \vee \psi]_\sim\end{aligned}$$

In words, we select the representative or representatives and perform the operation with them, e.g., the 'conjunction' of the classes $[p \to q]_\sim$ and $[q \vee \neg r]_\sim$ is:

$$[p \to q]_\sim \wedge [q \vee \neg r]_\sim = [(p \to q) \wedge (q \vee \neg r)]_\sim$$

By adding *constants* $\bot = [\bot]_\sim$ and $\top = [\top]_\sim$, we obtain the *(mathematical) structure*[27]

$$\mathbf{AV}_{\mathbb{P}} = \langle {}^{\mathrm{PF}_{\mathbb{P}}}/_{\sim}; \neg, \wedge, \vee, \bot, \top \rangle$$

[25]Unless, for example, we have a syntactic-transformation-based method such as conversion to CNF.

[26]We can view them as a kind of abstract 'properties' of models, regardless of their specific description.

[27]A structure is a non-empty set together with relations, operations, and constants. For example, a (directed) graph, a group, a field, a vector space. Structures will play an important role in predicate logic.

which we call the *algebra of propositions* of the language $\mathbb{P}$. It is an example of a so-called *Boolean algebra*. This means that its operations 'behave' like the operations $^-$, $\cap$, $\cup$ on the set of all subsets $\mathcal{P}(X)$ of some non-empty set X, and the constants correspond to $\emptyset$, X (such a Boolean algebra is called a *power set algebra*).[28]

The mapping $h : {}^{\mathrm{PF}_{\mathbb{P}}}/_{\sim} \to \mathcal{P}(\mathrm{M}_{\mathbb{P}})$ is thus an injective mapping from the algebra of propositions $\mathbf{AV}_{\mathbb{P}}$ to the power set algebra

$$\mathcal{P}(\mathrm{M}_{\mathbb{P}}) = \langle \mathcal{P}(\mathrm{M}_{\mathbb{P}}); {}^-, \cap, \cup, \emptyset, \mathrm{M}_{\mathbb{P}} \rangle$$

and if the language is finite, it is a bijection. This mapping 'preserves' the operations and constants, i.e., it holds that $h(\bot) = \emptyset$, $h(\top) = \mathrm{M}_{\mathbb{P}}$, and

$$\begin{aligned} h(\neg[\varphi]_\sim) &= \overline{h([\varphi]_\sim)} = \overline{\mathrm{M}(\varphi)} = \mathrm{M}_{\mathbb{P}} \setminus \mathrm{M}(\varphi) \\ h([\varphi]_\sim \wedge [\psi]_\sim) &= h([\varphi]_\sim) \cap h([\psi]_\sim) = \mathrm{M}(\varphi) \cap \mathrm{M}(\psi) \\ h([\varphi]_\sim \vee [\psi]_\sim) &= h([\varphi]_\sim) \cup h([\psi]_\sim) = \mathrm{M}(\varphi) \cup \mathrm{M}(\psi) \end{aligned}$$

Such a mapping is called a *homomorphism* of Boolean algebras, and if it is a bijection, it is an *isomorphism*.

Remark 2.5.2. We can apply these relationships when searching for models: for the proposition $\varphi \to (\neg\psi \wedge \chi)$, it holds (using the fact that $\mathrm{M}(\varphi \to \varphi') = \mathrm{M}(\neg\varphi \vee \varphi')$):

$$\mathrm{M}(\varphi \to (\neg\psi \wedge \chi)) = \overline{\mathrm{M}(\varphi)} \cup (\overline{\mathrm{M}(\psi)} \cap \mathrm{M}(\chi))$$

We can also relativize all of the above reasoning with respect to a given theory T in the language $\mathbb{P}$ by replacing the equivalence $\sim$ with T-equivalence $\sim_T$ and the set of models of the language $\mathrm{M}_{\mathbb{P}}$ with the set of models of the theory $\mathrm{M}_{\mathbb{P}}(T)$. We get:

$$\begin{aligned} h(\bot) &= \emptyset, \\ h(\top) &= \mathrm{M}(T) \\ h(\neg[\varphi]_{\sim_T}) &= \mathrm{M}(T) \setminus \mathrm{M}(T, \varphi) \\ h([\varphi]_{\sim_T} \wedge [\psi]_{\sim_T}) &= \mathrm{M}(T, \varphi) \cap \mathrm{M}(T, \psi) \\ h([\varphi]_{\sim_T} \vee [\psi]_{\sim_T}) &= \mathrm{M}(T, \varphi) \cup \mathrm{M}(T, \psi) \end{aligned}$$

The resulting *algebra of propositions with respect to the theory* T is denoted as $\mathbf{AV}_{\mathbb{P}}(T)$. The algebra of propositions of the language is thus the same as the algebra of propositions with respect to the empty theory. For technical reasons, we need $\mathrm{M}(T)$ to be non-empty, i.e., T must be consistent. Let's summarize our considerations:

Corollary 2.5.3. *If T is a consistent theory over a* finite *language $\mathbb{P}$, then the algebra of propositions $\mathbf{AV}_{\mathbb{P}}(T)$ is isomorphic to the power set algebra $\mathcal{P}(\mathrm{M}_{\mathbb{P}}(\mathbf{T}))$ through the mapping $h([\varphi]_{\sim_T}) = \mathrm{M}(T, \varphi)$.*

Thus, we know that negation, conjunction, and disjunction correspond to complement, intersection, and union of sets of models, and that if we want to find the number of propositions up to equivalence or T-equivalence, we just need to determine the number of corresponding sets of models. Let us summarize a few such calculations in the form of a proposition, leaving its proof as an exercise.

[28]That is, they satisfy certain algebraic laws, such as the distributive law of $\wedge$ over $\vee$. Boolean algebras will be formally defined later, but let us mention another important example: the set of all n-bit vectors with operations $\sim$, &, | (coordinate-wise) and with constants $(0, 0, \ldots, 0)$ and $(1, 1, \ldots, 1)$.

Proposition 2.5.4. *Let $\mathbb{P}$ be a language of size n and T a consistent theory in $\mathbb{P}$ with exactly k models. Then in $\mathbb{P}$ there are,* up to equivalence*:*

- 2^{2^n} *propositions (or theories),*
- 2^{2^n-k} *propositions true (or false) in T,*
- $2^{2^n} - 2 \cdot 2^{2^n-k}$ *propositions independent in T,*
- 2^k *simple extensions of the theory T (of which 1 is inconsistent),*
- k *complete simple extensions of T.*

Furthermore, up to T-equivalence*, there are:*

- 2^k *propositions,*
- 1 *proposition true in T,* 1 *false in T,*
- $2^k - 2$ *propositions independent in T.*

Exercise 2.8. Choose a suitable theory T and use it as an example to demonstrate that Proposition 2.5.4 holds.

Exercise 2.9. Prove Proposition 2.5.4 in detail. (Draw a Venn diagram.)

Exercise 2.10. Prove in detail that the mapping h from Corollary 2.5.3 is well-defined, injective, and if the language is finite, also surjective.

Chapter 3

The Boolean Satisfiability Problem

The boolean satisfiability problem, also known as the *SAT problem*, is the following computational task: The input is a proposition φ in CNF (in some reasonable encoding[1]), and the goal is to decide whether φ is *satisfiable*.[2]

As we demonstrated in the previous chapter, we can convert every proposition, or even every propositional theory in a finite language, into a CNF formula. The SAT problem is thus, in a sense, universal; it answers the question of whether there exists a model.

The well-known Cook-Levin theorem states that the SAT problem is *NP-complete*, meaning it is in the NP class (if an oracle provides the right truth assignment, we can easily verify that all clauses are satisfied) and that every problem in the NP class can be reduced to it in polynomial time (specifically, the computation of a Turing machine can be described using a CNF formula).[3]

However, practical SAT solvers can handle instances containing many (up to tens of millions) propositional variables and clauses. In this chapter, we will first demonstrate the practical application of a SAT solver to a 'real-life' problem, then show two fragments of the SAT problem, namely *2-SAT* and *Horn-SAT*, for which there are polynomial algorithms, and finally, we will also present the DPLL algorithm, which is the basis of (almost) all SAT solvers. (Later, in Chapter 4, we will also see the connection with the *resolution method.*)

3.1 SAT Solvers

The first SAT solvers were developed in the 1960s. They are almost always based on the DPLL (Davis–Putnam–Logemann–Loveland) algorithm, which we will introduce in Section 3.4, or some of its improvements. After 2000, there has been a somewhat surprising, dramatic development of technologies for SAT solvers, leading to a rapid increase in their utility in various areas of applied computer science.

Modern SAT solvers use a range of technologies for efficiently solving typical instances from various application domains, strategies, and heuristics for exploring the solution space (including, for examle, the use of machine learning and neural networks), and other enhancements. These modern tools typically have several tens of thousands of lines of code. The

[1]For example, the DIMACS-CNF format.

[2]Beware, in some literature SAT means satisfiability of an *arbitrary* proposition, the restriction of inputs to CNF is done only for the problem k-SAT (see below).

[3]See the course NTIN090 Introduction to Complexity and Computability.

availability of efficient SAT solvers has significantly impacted developments in areas such as software verification, program analysis, optimization, and artificial intelligence. The best SAT solvers regularly compete in the SAT competition.

To try out SAT solving, we will use the solver `Glucose`. It accepts input in the simple DIMACS CNF format. Let's demonstrate the usage on the following puzzle called *boardomino*:

Example 3.1.1 (Boardomino). Can a chessboard with two opposite corners cut out be perfectly covered with domino tiles?

How to formalize this problem? Let us choose propositional variables $h_{i,j}, v_{i,j}$ $(1 \leq i, j \leq n)$, where $h_{i,j}$ means "the left half of a horizontally oriented domino lies at position (i, j)" and similarly $v_{i,j}$ for the top half of a vertically oriented domino. Here $n = 8$, but we can try it for other (even) dimensions of the chessboard. Now we axiomatize all required properties:

- the upper left and lower right corners are missing: $\neg h_{11}, \neg v_{11}, \neg h_{n,n-1}, \neg v_{n-1,n}$
- dominos do not extend beyond the chessboard (to the right or down): $\neg h_{i,n}, \neg v_{n,i}$ for $1 \leq i \leq n$
- each square is covered by at least one domino (the first row and column separately):

$$\begin{aligned} h_{i,j-1} \vee h_{i,j} \vee v_{i-1,j} \vee v_{i,j} &\text{ for } 1 < i, j \leq n \\ h_{1,j-1} \vee h_{1,j} \vee v_{1,j} &\text{ for } 1 < j \leq n \\ h_{i,1} \vee v_{i-1,1} \vee v_{i,1} &\text{ for } 1 < i \leq n \end{aligned}$$

- each square is covered by at most one domino (the first row and column separately):

$$\begin{aligned} (\neg h_{i,j-1} \vee \neg h_{i,j}) \wedge (\neg h_{i,j-1} \vee \neg v_{i-1,j}) \wedge (\neg h_{i,j-1} \vee \neg v_{i,j}) \wedge \\ (\neg h_{i,j} \vee \neg v_{i-1,j}) \wedge (\neg h_{i,j} \vee \neg v_{i,j}) \wedge (\neg v_{i-1,j} \vee \neg v_{i,j}) &\text{ for } 1 < i, j \leq n \\ (\neg h_{1,j-1} \vee \neg h_{1,j}) \wedge (\neg h_{1,j-1} \vee \neg v_{1,j}) \wedge (\neg h_{1,j} \vee \neg v_{1,j}) &\text{ for } 1 < j \leq n \\ (\neg h_{i,1} \vee \neg v_{i-1,1}) \wedge (\neg h_{i,1} \vee \neg v_{i,1}) \wedge (\neg v_{i-1,1} \vee \neg v_{i,1}) &\text{ for } 1 < i \leq n \end{aligned}$$

The resulting theory is already in CNF, and we can easily write it in the DIMACS CNF format and solve it using the Glucose solver. In practice, we might program this conversion or use one of the many high-level languages from the *constraint programming* area that allow translation to SAT.

We will see that such instances of the SAT problem will be difficult for the solver, and for relatively small dimensions of the chessboard, we will not get a solution. As mathematicians, we can easily see that there is no solution: Each domino covers one white and one black square, but we removed two white squares, so necessarily two black squares will remain. However, this view is not available in the CNF encoding. It is possible to find partial truth assignments of almost all variables without violating any condition. The solver will have to search almost the entire solution space before proving unsatisfiability.[4] The key insight into SAT solving is that such difficult instances almost never occur in practice.

[4]Similar properties also hold for the encoding of the *pigeonhole principle* into SAT.

3.2 2-SAT and Implication Graph

A proposition φ is in k-CNF if it is in CNF and each clause has at most k literals. The k-SAT problem asks whether a given k-CNF proposition is satisfiable. For $k \geq 3$, k-SAT remains NP-complete; any CNF proposition can be encoded into a 3-CNF proposition:

Exercise 3.1. Show that for any proposition φ in CNF, there exists an *equisatisfiable* proposition φ' in 3-CNF (i.e., φ is satisfiable if and only if φ' is satisfiable), which can be constructed in linear time.

For the 2-SAT problem, however, there is a polynomial (linear, even) algorithm, which we will now introduce. The algorithm utilizes the notion of *implication graph.* We will demonstrate the procedure with an example:

Example 3.2.1. Consider the following 2-CNF proposition φ:

$$(\neg p_1 \vee p_2) \wedge (\neg p_2 \vee \neg p_3) \wedge (p_1 \vee p_3) \wedge (p_3 \vee \neg p_4) \wedge (\neg p_1 \vee p_5) \wedge (p_2 \vee p_5) \wedge p_1 \wedge \neg p_4$$

Implication Graph

The implication graph of a 2-CNF proposition φ is based on the idea that a 2-clause $\ell_1 \vee \ell_2$ (where ℓ_1 and ℓ_2 are literals) can be viewed as a pair of implications: $\overline{\ell_1} \to \ell_2$ and $\overline{\ell_2} \to \ell_1$.[5] For example, from the clause $\neg p_1 \vee p_2$, we get the implications $p_1 \to p_2$ and $\neg p_2 \to \neg p_1$. Therefore, if p_1 holds in some model, p_2 must hold as well, and if p_2 does not hold, then p_1 must not hold either. A unit clause ℓ can also be expressed as an implication, namely $\overline{\ell} \to \ell$, e.g., from p_1 we get $\neg p_1 \to p_1$.

The implication graph $\mathcal{G}_\varphi$ is thus a directed graph, whose vertices are all the literals (variables from $\text{Var}(\varphi)$ and their negations), and edges are given by the implications described above:

- $V(\mathcal{G}_\varphi) = \{p, \neg p \mid p \in \text{Var}(\varphi)\}$,
- $E(\mathcal{G}_\varphi) = \{(\overline{\ell_1}, \ell_2), (\overline{\ell_2}, \ell_1) \mid \ell_1 \vee \ell_2 \text{ is a clause in } \varphi\} \cup \{(\overline{\ell}, \ell) \mid \ell \text{ is a unit clause in } \varphi\}$

In our example, the set of vertices is

$$V(\mathcal{G}_\varphi) = \{p_1, p_2, p_3, p_4, p_5, \neg p_1, \neg p_2, \neg p_3, \neg p_4, \neg p_5\}$$

and the edges are:

$$\begin{aligned} E(\mathcal{G}_\varphi) = \{&(p_1, p_2), (\neg p_2, \neg p_1), (p_2, \neg p_3), (p_3, \neg p_2), (\neg p_1, p_3), (\neg p_3, p_1), (\neg p_3, \neg p_4), \\ &(p_4, p_3), (p_1, p_5), (\neg p_5, \neg p_1), (\neg p_2, p_5), (\neg p_5, p_2), (\neg p_1, p_1), (p_4, \neg p_4)\} \end{aligned}$$

The resulting graph is shown in Figure 3.1.

3.2.1 Strongly Connected Components

We now need to find the strongly connected components[6] of this graph. In our example, we get the following components: $C_1 = \{p_4\}$, $C_2 = \{\neg p_5\}$, $C_3 = \{\neg p_1, \neg p_2, p_3\}$, $\overline{C_3} = \{p_1, p_2, \neg p_3\}$, $\overline{C_2} = \{p_5\}$, $\overline{C_1} = \{\neg p_4\}$.

[5] In the previous chapter, we expressed $p_1 \to p_2$ as $\neg p_1 \vee p_2$; here, we are performing the reverse procedure.

[6] *Strong connectivity* means that there is a directed path from u to v and from v to u, i.e., every two vertices in one component lie in a directed cycle. Conversely, every directed cycle lies within some component.

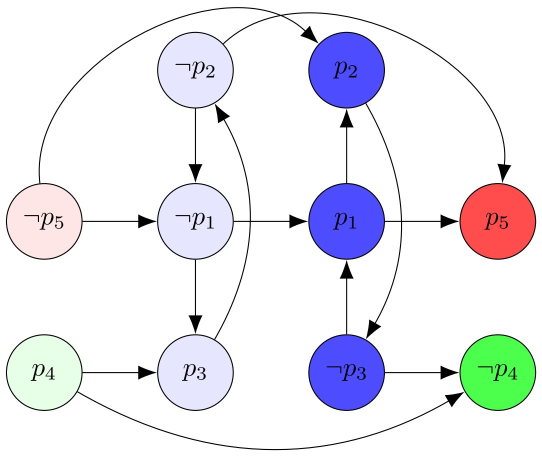


Figure 3.1: Implication graph $\mathcal{G}_\varphi$. Strongly connected components are distinguished by colors.

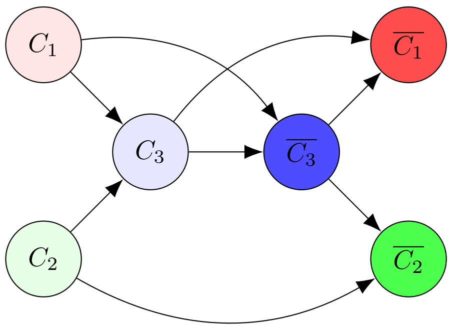


Figure 3.2: Implication graph $\mathcal{G}_\varphi$. Graph of strongly connected components $\mathcal{G}^*_\varphi$.

All literals in one component must be assigned the same value. Therefore, if we find a pair of opposite literals in one component, it means that the proposition is unsatisfiable. Otherwise, we can always find a satisfying assignment, as we will prove in Proposition 3.2.2. We need to ensure that no component assigned 1 has an edge leading to a component assigned 0. If we contract the components (and remove loops), the resulting graph $\mathcal{G}^*_\varphi$ is acyclic (every cycle was within some component), see Figure 3.2. Thus we can draw it in a *topological ordering* (i.e., an ordering on a line where edges only go to the right), see Figure 3.3 below.

When searching for a satisfying assignment (if we are not content with the information that the proposition is satisfiable), we proceed by taking the leftmost unassigned component, assigning it 0, assigning the opposite component 1, and repeating this process until no unassigned components remain. For example, the topological ordering in Figure 3.3 corresponds to the model $v = (1, 1, 0, 0, 1)$.

Finally, we summarize our reasoning in the following proposition:

Proposition 3.2.2. *A proposition φ is satisfiable if and only if no strongly connected component in $\mathcal{G}_\varphi$ contains a pair of opposite literals $\ell, \overline{\ell}$.*

Proof. Every model, i.e., valid truth assignment, must assign all literals in the same compo-

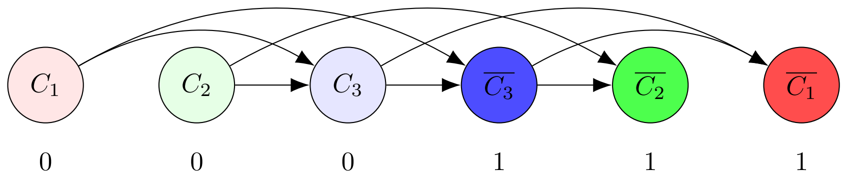


Figure 3.3: Implication graph $\mathcal{G}_\varphi$. Topological ordering of the graph $\mathcal{G}^*_\varphi$ and satisfying assignment of the components.

nent the same truth value. (Otherwise, there would necessarily be an implication $\ell_1 \to \ell_2$ where ℓ_1 is valid in the model but ℓ_2 is not.) Therefore, there cannot be opposite literals in one component.

Conversely, assume that no component contains a pair of opposite literals, and let us show that there exists a model. Let $\mathcal{G}^*_\varphi$ be the graph obtained from $\mathcal{G}_\varphi$ by contracting the strongly connected components. This graph is acyclic; choose a topological ordering. We construct a model by choosing the first unassigned component in our topological ordering, assigning all literals in it the value 0, and assigning the opposite literals the value 1. We continue this process until all components are assigned.

Why does the proposition φ hold in the model thus obtained? If it did not, some clause would not hold. A unit clause ℓ must hold because there is an edge $\overline{\ell} \to \ell$ in the graph $\mathcal{G}_\varphi$. The same edge is also in the graph of components, so $\overline{\ell}$ precedes the component containing ℓ in the topological ordering. In constructing the model, we must have assigned $\overline{\ell} = 0$, so $\ell = 1$. Similarly, a 2-clause $\ell_1 \vee \ell_2$ must also hold: there are edges $\overline{\ell_1} \to \ell_2$ and $\overline{\ell_2} \to \ell_1$. If we assigned ℓ_1 first, we must have assigned $\overline{\ell_1} = 0$ due to the edge $\overline{\ell_1} \to \ell_2$, so ℓ_1 holds. Similarly, if we assigned ℓ_2 first, $\overline{\ell_2} = 0$ and $\ell_2 = 1$. □

Corollary 3.2.3. *The 2-SAT problem is solvable in linear time. We can also construct a model in linear time, if one exists.*

Proof. The strongly connected components can be found in $\mathcal{O}(|V| + |E|)$ time, and the topological ordering can also be constructed in $\mathcal{O}(|V| + |E|)$ time. □

Exercise 3.2. Find an unsatisfiable 2-CNF proposition, construct its implication graph, and verify that there is a pair of opposite literals in the same strongly connected component.

Exercise 3.3. Find all topological orderings of the graph $\mathcal{G}^*_\varphi$ from the example above and the corresponding models. Think about why we obtain exactly all models of the proposition φ in this way.

Exercise 3.4. Explain how to find the components and topological ordering in $\mathcal{O}(|V| + |E|)$ time.

3.3 Horn-SAT and Unit Propagation

Next, we will introduce another fragment of SAT that can be solved in polynomial time, the so-called *Horn-SAT*, the *Horn satisfiability problem*. A propositional formula is *Horn* (*in Horn form*)[7] if it is a conjunction of *Horn clauses*, i.e., clauses containing *at most one *positive* literal.* The significance of Horn clauses is evident from their equivalent expression in the form of an implication:

$$\neg p_1 \vee \neg p_2 \vee \cdots \vee \neg p_n \vee q \ \sim\ (p_1 \wedge p_2 \wedge \cdots \wedge p_n) \to q$$

Horn formulas are thus well suited to model systems where the satisfaction of certain conditions guarantees the satisfaction of another condition. Note that a unit clause ℓ is also Horn. In the context of logic programming, it is called a *fact* if the literal is positive, and a *goal* if it is negative.[8] Horn formulas with at least one positive and at least one negative literal are *rules.*

Example 3.3.1. An example of a proposition that is in CNF but not Horn is, for instance, $(p_1 \vee p_2 \vee \neg p_3) \wedge (\neg p_1 \vee p_3)$. As an example to illustrate the algorithm, we will use the following Horn formula:

$$\varphi = (\neg p_1 \vee p_2) \wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_2 \vee \neg p_3) \wedge (\neg p_5 \vee \neg p_4) \wedge p_4$$

The polynomial algorithm for solving the Horn-SAT problem is based on the simple idea of *unit propagation*: If our proposition contains a *unit* clause, we know how the propositional variable in this clause must be valued. And this knowledge can be *propagated*—exploited to simplify the proposition.

Our proposition φ contains the unit clause p_4. We know, therefore, that every model $v \in \mathrm{M}(\varphi)$ must satisfy $v(p_4) = 1$. This means that in any model of the proposition φ:

- Every clause containing the positive literal p_4 is satisfied, so we can remove it from the proposition,
- The negative literal $\neg p_4$ cannot be satisfied, so we can remove it from all clauses that contain it.

This step is called *unit propagation.* The result is the following simplified proposition, denoted as φ^{p_4} (in general φ^{ℓ} if we have a unit clause ℓ):

$$\varphi^{p_4} = (\neg p_1 \vee p_2) \wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_2 \vee \neg p_3) \wedge \neg p_5$$

Observation 3.3.2. *Note that φ^{ℓ} no longer contains the literal ℓ nor $\overline{\ell}$, and it is obvious that the models of φ are exactly the models of $\{\varphi^{\ell}, \ell\}$, i.e., the models of φ^{ℓ} in the original language $\mathbb{P}$, in which ℓ is valid.*

By unit propagation, we obtained a new unit clause $\neg p_5$ in the proposition φ^{p_4}, so we can continue by setting $v(p_5) = 0$ and performing another unit propagation:

$$(\varphi^{p_4})^{\neg p_5} = (\neg p_1 \vee p_2) \wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_2 \vee \neg p_3)$$

The resulting proposition no longer contains a unit clause. This means that each clause contains at least two literals, and at most one of them can be positive! (Here we need the 'Horn property' of the proposition.) Because each clause contains a negative literal, it is sufficient to evaluate all remaining variables as 0, and the proposition will be satisfied: $v(p_1) = v(p_2) = v(p_3) = 0$. Thus, we get the model $v = (0, 0, 0, 1, 0)$.

[7]Mathematician Alfred Horn discovered the significance of this form of logical formulas (thus laying the foundation for logic programming) in 1951.

[8]Because we are proving by contradiction, more on this in Chapter 4 and Section 8.7.2.

Example 3.3.3. What would happen if the proposition was not satisfiable? Let us look at the proposition

$$\psi = p \wedge (\neg p \vee q) \wedge (\neg q \vee r) \wedge \neg r$$

and perform unit propagation as in the previous example: we have $v(p) = 1$ and $\psi^p = q \wedge (\neg q \vee r) \wedge \neg r$, further $v(q) = 1$ and $(\psi^p)^q = r \wedge \neg r$. This proposition is unsatisfiable because it contains a pair of opposite unit clauses. [9]

Let us summarize the algorithm for solving the Horn-SAT problem:

Algorithm (Horn-SAT)**. input**: a proposition φ in Horn form, **output**: a model of φ or information that φ is unsatisfiable

1. If φ contains a pair of opposite unit clauses $\ell, \overline{\ell}$, it is unsatisfiable.
2. If φ does not contain any unit clause, it is satisfiable, evaluate remaining variables to 0.
3. If φ contains a unit clause ℓ, evaluate the literal ℓ to 1, perform unit propagation, replace φ with the proposition φ^ℓ, and return to the beginning.

Proposition 3.3.4. *The algorithm is correct.*

Proof. Correctness follows from the Observation and the preceding discussion. □

Corollary 3.3.5. *Horn-SAT can be solved in linear time.*

Proof. In each step, it is sufficient to traverse the proposition once, and unit propagation always shortens the proposition. This easily gives a quadratic upper bound, but with a suitable implementation, we can achieve linear time with respect to the length of φ. □

Exercise 3.5. Propose an implementation of the algorithm for Horn-SAT in linear time.

Exercise 3.6. Propose a modification of the Horn-SAT algorithm that finds all models.

3.4 The DPLL Algorithm for Solving the SAT Problem

To conclude the chapter on the satisfiability problem, we will introduce the most widely used algorithm, by far, for solving the general SAT problem: the DPLL algorithm.[10] Although it has exponential complexity in the worst case, it performs very efficiently in practice.

The algorithm uses unit propagation along with the following observation: We say that a literal ℓ has *pure occurrence* in φ if it occurs in φ but the opposite literal $\overline{\ell}$ does not occur in φ. If we have a literal with pure occurrence, we can set its value to 1, and thus satisfy (and remove) all clauses containing it. If the proposition cannot be simplified neither by unit propagation nor by pure literal, we branch the computation by assigning both possible values to a selected propositional variable.

Algorithm (DPLL)**. input**: a proposition φ in CNF, **output**: a model of φ or information that φ is unsatisfiable

[9] In other words, in the next step, we would perform unit propagation on r, remove the unit clause r, and from the remaining unit clause $\neg r$, we would remove the literal $\neg r$, resulting in an *empty clause*, which is unsatisfiable.

[10] Named after its creators, Davis-Putnam-Logemann-Loveland, it was invented in 1961.

1. While φ contains a unit clause ℓ, set the literal ℓ to 1, perform unit propagation, and replace φ with the proposition φ^ℓ.
2. While there is a literal ℓ with pure occurrence in φ, set ℓ to 1, and remove clauses containing ℓ.
3. If φ contains no clauses, it is satisfiable.
4. If φ contains an empty clause, it is unsatisfiable.
5. Otherwise, choose an unassigned propositional variable p, and recursively call the algorithm on $\varphi \wedge p$ and on $\varphi \wedge \neg p$.

The algorithm runs in exponential time: the number of branching points in the computation cannot exceed the number of variables. It can be shown that in the worst case, exponential time is indeed needed. The correctness of the algorithm is not difficult to verify.

Proposition 3.4.1. *The DPLL algorithm solves the SAT problem.*

Example 3.4.2. We will demonstrate the algorithm on the following example:

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

The proposition does not contain any unit clauses. The literal $\neg r$ has pure occurrence, so we set $v(r) = 0$ and remove the clauses containing $\neg r$:

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

No other literal has pure occurrence. We therefore recursively run the algorithm:

(p=1) Add the unit clause p:

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

Set $v(p) = 1$ and perform unit propagation: $(\neg q \vee \neg s) \wedge (q \vee s)$. Now we branch on the variable q:

(q=1) $(\neg q \vee \neg s) \wedge (q \vee s) \wedge q$. After setting $v(q) = 1$ and unit propagation, we get $\neg s$, and after setting $v(s) = 0$ and unit propagation, we get a proposition containing no clauses, which is thus satisfiable by the model $(1, 1, 0, 0)$. We already have the answer to the satisfiability problem, so we do not need to complete the other branches of the computation. For illustration, we will do so.

(q=0) $(\neg q \vee \neg s) \wedge (q \vee s) \wedge \neg q$. By unit propagation with $v(q) = 0$, we get s, and after setting $v(s) = 1$ and unit propagation, we get an empty set of clauses. We obtain the model $(1, 0, 0, 1)$.

(p=0) Add the unit clause $\neg p$:

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

After performing unit propagation on $\neg p$, we have $s \wedge \neg s \wedge (q \vee s)$. After unit propagation on s, we have $\square \wedge q$, where $\square$ is an empty clause. The proposition is thus unsatisfiable, and we do not get any models in this branch.

We found that the original proposition is satisfiable. We found 2 models: $(1, 1, 0, 0)$ and $(1, 0, 0, 1)$. However, there may be other models, the valuation $v(r) = 0$ for the literal $\neg r$ with pure occurrence may not be necessary to satisfy all clauses; this step does not preserve the set of models, only satisfiability.

What's next? The basic DPLL algorithm, which systematically explores the solution space, was enhanced and extended in various ways at the end of 1990s. Let us mention the algorithm called *Conflict-driven clause learning (CDCL).* It is based on the idea that we can *learn* a new clause from the failure of a branch in the search tree, which prevents future repetitions of the same failure ("conflict"). Additionally, we can backtrack in the tree by more than one level at once (called *back-jumping*) to the point where we started evaluating variables in this new clause. Thus we prevent repeatedly discovering the "same" conflict. You can learn much more about SAT solvers in the course NAIL094 Decision procedures and SAT/SMT solvers.

Chapter 4

The Method of Analytic Tableaux

In this chapter, we introduce the *Method of Analytic Tableaux.* It is a syntactic procedure that can be used to determine whether a given proposition is true in a given theory without having to deal with semantics (e.g., searching for all models, which is impractical). We will prove its *soundness* ('it gives correct answers') and *completeness* ('it always works'). Moreover, we will use it to prove the so-called *Compactness Theorem* ('properties of an infinite object can be established by proving them only for all its finite parts').

4.1 Formal Proof Systems

A *formal proof system* formalizes 'proof' (e.g., in mathematics) as a precisely (algorithmically) given syntactic procedure. A *proof* of the fact that a proposition φ holds in a theory T (i.e., $T \models \varphi$) is a finite syntactic object derived from the axioms of T and the proposition φ. If a proof exists, it can be found 'algorithmically'.[1] Moreover, one must be able to algorithmically (and reasonably efficiently) verify that a given object is indeed a valid proof.

If a proof exists, we say that φ is *provable* [in the given proof system] from T, and we write $T \vdash \varphi$. We require the following two properties from a proof system:

- *soundness*: if a proposition is provable from a theory, then it is true in that theory ($T \vdash \varphi \Rightarrow T \models \varphi$)
- *completeness*: if a proposition is true in a theory, then it is provable from that theory ($T \models \varphi \Rightarrow T \vdash \varphi$)

(While soundness is always required, an effective proof system can be practical even if it is not complete, especially if it is complete for some interesting class of propositions or theories.)

In this chapter, besides the *tableaux method*, we will also introduce the *Hilbert calculus*, and in the next chapter, we will present another proof system, the so-called *resolution method.*

4.2 Introduction to the Tableaux Method

For the rest of this chapter, we will always assume that we have a *countable* language $\mathbb{P}$. This implies that any theory over $\mathbb{P}$ is also countable. We will first focus on the case where $T = \emptyset$, i.e., we are proving that a proposition φ is *logically* valid (it is a *tautology*).

[1]Here we must be cautious in the case of an infinite theory T: how is it given? The algorithm must have effective access to all axioms.

A *tableau* is a labeled tree representing the search for a counterexample, i.e., a model in which φ does not hold. The labels on the nodes, which we will call *entries*, consist of the symbol T or F ('True'/'False') followed by a proposition ψ and represent the assumption (requirement) that the proposition ψ is or is not valid in a model, respectively. At the root of the tableau, we place the entry $\mathrm{F}\varphi$, i.e., we are looking for a model in which φ *is not valid.* We will then develop the tableau using rules for *reducing* the entries. These rules ensure the following invariant:

> Any model that *agrees* the entry at the root (i.e., in which φ is not valid) must *agree* with some branch of the tableau (i.e., satisfy all the requirements expressed by the entries on that branch).

If a branch contains entries of the form $\mathrm{T}\psi$ and $\mathrm{F}\psi$ for the same ψ, we say that the branch *fails* (is *contradictory*) and we know that no model can agree with it. Thus, if all branches fail, we can conclude that there is no model in which φ is invalid, and thus we have a *proof* that φ is a tautology. (Note that this is a *proof by contradiction.*)

If a branch does not fail and is *finished*, i.e., all entries are reduced, we know that φ is not a tautology, and we will be able to construct a specific model from this branch in which φ is not valid.

Example 4.2.1. Let us illustrate the entire procedure with two examples, see Figure 4.2.1.

(a) First, let us construct a tableau proof of the proposition $\varphi = ((p \to q) \to p) \to p$. We start with the root entry $\mathrm{F}\varphi$. This entry is of the form $\mathrm{F}\varphi_1 \to \varphi_2$ ('implication does not hold'), so if any model agrees with it, it must satisfy $\mathrm{T}(p \to q) \to p$ and $\mathrm{F}p$. We thus append these two entries. (Technically, we append the *atomic tableau* for this case, see Table 4.1, but we omit the root of this atomic tableau to avoid repeating the same entry.) This reduces the entry at the root.

    We continue with the entry $\mathrm{T}(p \to q) \to p$, which is of the form 'implication holds'. We split into two branches: the model agrees with $\mathrm{F}(p \to q)$ or with $\mathrm{T}p$ (or both). The right branch *fails* (is *contradictory*) because it contains the entries $\mathrm{T}p$ and $\mathrm{F}p$, so no model agrees with it; we mark it with the symbol $\otimes$. In the left branch, we further reduce the entry $\mathrm{F}p \to q$ and also get a contradictory branch. All branches are contradictory, so no counterexample exists, and we have a proof of the proposition φ. We write $\vdash \varphi$.

(b) Now let us construct a tableau with the entry $\mathrm{F}(\neg q \vee p) \to p$ in the root. We are trying to find a counterexample: a model in which $(\neg q \vee p) \to p$ does not hold. We first used the atomic tableau for 'implication does not hold' and then reduced the entry $\mathrm{T}\neg q \vee p$ by adding the atomic tableau for 'disjunction holds'. The right branch failed. In the left branch, we further reduced $\mathrm{T}\neg q$ to $\mathrm{F}q$ (the atomic tableau for 'negation holds'), obtaining a finished branch since all entries have been reduced. This finished branch is non-contradictory (we mark it with the symbol ✓). This means that a counterexample exists: we have the entries $\mathrm{F}p$ and $\mathrm{F}q$, which correspond to the model $(0, 0)$, where $(\neg q \vee p) \to p$ does not hold.

In the next section, we will formalize the entire procedure and explain how to proceed when we want to prove propositions not in logic, but rather in some theory T (spoiler alert: we append the entries $\mathrm{T}\alpha$ for the axioms $\alpha \in T$ during the construction). We will also see an example with an infinite theory where a *finished* branch sometimes needs to be infinite.

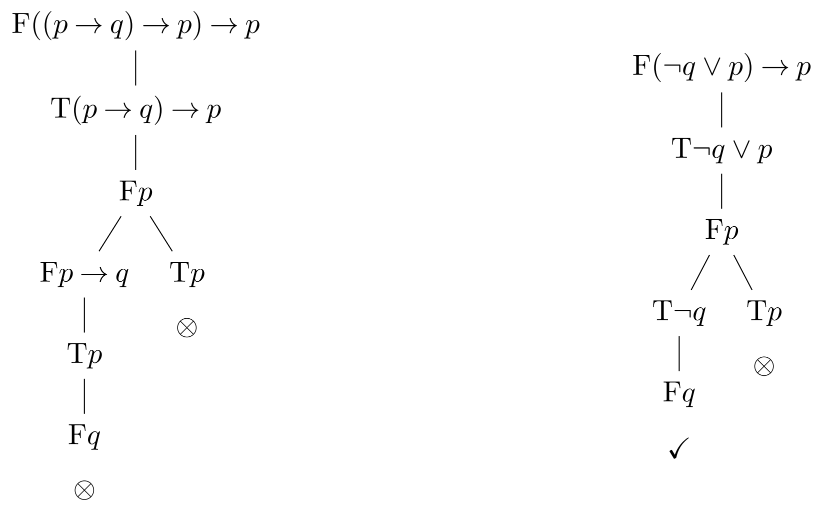


Figure 4.1: Examples of tableaux. (a) Tableau proof of the proposition $((p \to q) \to p) \to p$. (b) Tableau for the proposition $(\neg q \vee p) \to p$. The left branch gives a counterexample, the model $(0, 0)$ where the proposition is not valid.

In the rest of this section, we will introduce all the *atomic tableaux* needed for construction, and formalize the concept of a *tree*.

4.2.1 Atomic Tableaux

Atomic tableaux represent the rules by which we reduce the entries. For each logical connective and each of the two signs T/ F, we have one atomic tableau, shown in Table 4.1.

	$\neg$	$\wedge$	$\vee$	$\to$	$\leftrightarrow$
True	$\mathrm{T}\neg\varphi$ $\mathrm{F}\varphi$	$\mathrm{T}\varphi \wedge \psi$ $\mathrm{T}\varphi$ $\mathrm{T}\psi$	$\mathrm{T}\varphi \vee \psi$ $\mathrm{T}\varphi \quad \mathrm{T}\psi$	$\mathrm{T}\varphi \to \psi$ $\mathrm{F}\varphi \quad \mathrm{T}\psi$	$\mathrm{T}\varphi \leftrightarrow \psi$ $\mathrm{T}\varphi \quad \mathrm{F}\varphi$ $\mathrm{T}\psi \quad \mathrm{F}\psi$
False	$\mathrm{F}\neg\varphi$ $\mathrm{T}\varphi$	$\mathrm{F}\varphi \wedge \psi$ $\mathrm{F}\varphi \quad \mathrm{F}\psi$	$\mathrm{F}\varphi \vee \psi$ $\mathrm{F}\varphi$ $\mathrm{F}\psi$	$\mathrm{F}\varphi \to \psi$ $\mathrm{T}\varphi$ $\mathrm{F}\psi$	$\mathrm{F}\varphi \leftrightarrow \psi$ $\mathrm{T}\varphi \quad \mathrm{F}\varphi$ $\mathrm{F}\psi \quad \mathrm{T}\psi$

Table 4.1: Atomic tableaux

The tableaux from Example 4.2.1 are constructed by sequentially adding atomic tableaux, see Figure 4.2.1. The roots of the atomic tableaux are marked in blue; we will adopt the convention of not drawing them.

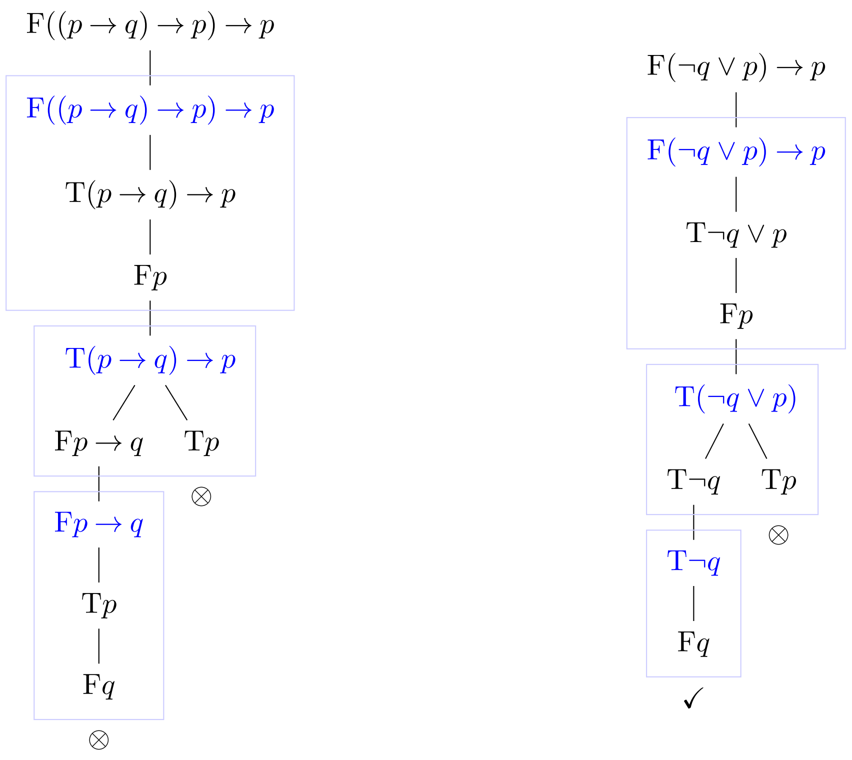


Figure 4.2: Construction of tableaux from Example 4.2.1.

Exercise 4.1. Try to construct a tableau with the entry F$((\neg p \wedge \neg q) \vee p) \to (\neg p \wedge \neg q)$ at the root and also a tableau with the entry T$(p \to q) \leftrightarrow (p \wedge \neg q)$. When constructing, use only atomic tableaux (check if your construction matches the definition of a tableau from the following section). Think about what these tableaux say about the propositions at their roots.

Exercise 4.2. Verify that all atomic tableaux satisfy the invariant: if a model agrees with the entry at the root, it agrees with some branch.

Exercise 4.3. Propose atomic tableaux for the logical connectives NAND, NOR, XOR, IFTE.

4.2.2 On Trees

Before we proceed to the formal definition and proofs, let us specify what we mean by a 'tree'. In graph theory, a tree would mean a connected graph without cycles, but our trees are rooted, ordered (i.e., with left-right order among the children of each node), and labeled. They can, and often will, be infinite. Formally:

Definition 4.2.2 (Tree).

- A *tree* is a non-empty set T with a partial order $<_T$ that has a (single) minimal element (*root*) and in which the set of ancestors of any node is *well-ordered*.[2]
- A *branch* of a tree T is a maximal[3] linearly ordered subset of T.

[2]That is, every non-empty subset has a least element. (This prevents infinite descending chains of ancestors.)
[3]That is, it cannot be extended by adding more nodes from the tree.

- An *ordered tree* is a tree T along with a linear order $<_L$ on the set of children of each node. We call the order of children *left-right* while the order $<_T$ is *tree order.*
- A *labeled tree* is a tree along with a labeling function label: $V(T) \to \text{Labels}$.

We will use standard terminology about trees, for example, we will talk about the *n-th level of a tree* or the *depth* of a tree (which is infinite, if and only if we have an infinite branch). In one of the theorems that we will prove below, we will need the following famous result, which is a consequence of the axiom of choice.

Lemma 4.2.3 (König's Lemma)**.** *An infinite, finitely branching tree has an infinite branch.*

(A tree is *finitely branching* if each node has finitely many children.)

4.3 Tableau Proof

We now present the formal definition of a tableau. In the definition, we also include a theory T: its axioms can be added during construction with the sign T ("true"). Recall that an *entry* is an expression $\mathrm{T}\varphi$ or $\mathrm{F}\varphi$, where φ is some proposition.

Definition 4.3.1 (Tableau)**.** A *finite tableau from a theory* T is an ordered, labeled tree constructed by applying finitely many of the following rules:

- A single-node tree labeled with any entry is a tableau from the theory T.
- For any entry E on any branch B, we can append the atomic tableau for the entry E at the end of the branch B.
- We can append the entry $\mathrm{T}\alpha$ for any axiom $\alpha \in T$ at the end of any branch.

A *tableau from the theory* T can be either finite or *infinite*: in the latter case, it is constructed in countably many steps. Formally, it can be expressed as the union $\tau = \bigcup_{i\geq 0} \tau_i$, where τ_i are finite tableaux from T, τ_0 is a single-node tableau, and τ_{i+1} is obtained from τ_i in one step.[4]

A tableau *for an entry* E is a tableau that has the entry E at its root.

Recall the convention that we do not write the root of an atomic tableau (since the node labeled by the entry E is already in the tableau). The definition does not specify the order in which to perform the steps; however, later we will describe a concrete construction procedure (algorithm) that we will call *systematic tableau.*

To obtain a proof system, it remains to define the concept of a *tableau proof* (and related terms). Recall once again that it is a proof by contradiction, i.e., we assume the proposition is not valid in T, and find a contradiction (a contradictory tableau):

Definition 4.3.2 (Tableau Proof)**.** A *tableau proof* of the proposition φ from the theory T is a *contradictory* tableau from the theory T with the entry $\mathrm{F}\varphi$ at the root. If it exists, then φ is *(tableau) provable* from T, written as $T \vdash \varphi$. (Also, we define *tableau refutation* as a contradictory tableau with the entry $\mathrm{T}\varphi$ at the root. If it exists, then φ is *(tableau) refutable* from T, i.e., $T \vdash \neg\varphi$ holds.)

- A tableau is *contradictory* if every branch is contradictory.

[4]We take the union because, in each step, we add new nodes to the tableau, so τ_i is a subtree of τ_{i+1}.

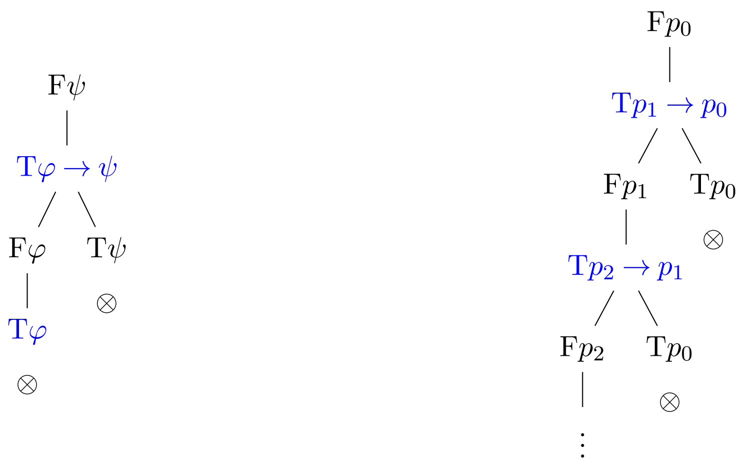


Figure 4.3: Tableaux from Example 4.3.3. Entries coming from axioms are marked in blue.

- A branch is *contradictory* if it contains entries $\mathrm{T}\psi$ and $\mathrm{F}\psi$ for some proposition ψ; otherwise, it is *non-contradictory.*
- A tableau is *finished* if every branch is finished.
- A branch is *finished* if
  - it is contradictory, or
  - every entry on this branch is *reduced* and it contains the entry $\mathrm{T}\alpha$ for every axiom $\alpha \in T$.
- An entry E is *reduced* on a branch B passing through this entry if
  - it is of the form $\mathrm{T}p$ or $\mathrm{F}p$ for some propositional variable $p \in \mathbb{P}$, or
  - it appears on B as the root of an atomic tableau[5] (i.e., typically, during the construction of the tableau, it has already been developed on B).

Example 4.3.3. Let us look at two examples. The tableaux are shown in Figure 4.3.

(a) Tableau proof of the proposition ψ from the theory $T = \{\varphi, \varphi \to \psi\}$, i.e., $T \vdash \psi$ (where φ, ψ are some fixed propositions). This fact is called the *Deduction Theorem.*

(b) Finished tableau for the proposition p_0 from the theory $T = \{p_{n+1} \to p_n \mid n \in \mathbb{N}\}$. The leftmost branch is non-contradictory and finished. It contains the entries $\mathrm{T}p_{i+1} \to p_i$ and $\mathrm{F}p_i$ for all $i \in \mathbb{N}$. Thus, it agrees with the model $v = (0, 0, \dots)$, i.e., $v : \mathbb{P} \to \{0, 1\}$ where $v(p_i) = 0$ for all i.

Exercise 4.4. Let us return to the tableaux from Exercise 4.1. Are they tableau proofs or refutations (from the theory $T = \emptyset$)? Which entries on which branches are reduced? Which branches are contradictory, and which are finished?

[5]Although, by convention, we do not write this root.

4.4 Finiteness and Systematicity of Proofs

In this section, we will prove that if a tableau proof exists, there is always also a *finite* tableau proof. Additionally, we will present an algorithm by which a tableau proof can always be found; however, we will need the Soundness Theorem and the Completeness Theorem from the next section to prove this fact. For now, we will show that this algorithm will always allow us to construct a finished tableau.

Note that when reducing an entry, we only add entries containing shorter propositions to the tableau. Therefore, if we have a finite theory and do not make unnecessary steps (e.g., repeatedly adding the same axiom or the same atomic tableau), it is easy to construct a finished tableau that will be finite.

If the theory T is infinite, we must be more careful. We could be constructing the tableau forever, without ever reducing a particular entry or using one of the axioms. Therefore, we define a specific algorithm for constructing a tableau, and the resulting tableau will be called a *systematic tableau.* The idea of the construction is simple: we alternate between reducing an entry (simultaneously on all non-contradictory branches passing through it) and using an axiom. We traverse the entries by levels, and within a level in left-right order. We go through the axioms of the theory one by one in a fixed ordering.

Definition 4.4.1. Let R be an entry and $T = \{\alpha_1, \alpha_2, \dots\}$ a theory (finite or infinite[6]). A *systematic tableau* from the theory T for the entry R is the tableau $\tau = \bigcup_{i\geq 0} \tau_i$, where τ_0 is a single-node tableau labeled by R, and for each $i \geq 0$:

- Let E be the leftmost entry at the smallest level that is not reduced on some non-contradictory branch passing through E. We first define the tableau τ_i' as the tableau obtained from τ_i by appending the atomic tableau for E to each non-contradictory branch passing through E. (If such an entry does not exist, then $\tau_i' = \tau_i$.)
- Subsequently, τ_{i+1} is the tableau obtained from τ_i' by appending $\mathrm{T}\alpha_{i+1}$ to each non-contradictory branch of τ_i'. That is while $i < |T|$, otherwise (if T is finite and we have already used all the axioms) skip this step and define $\tau_{i+1} = \tau_i'$.

Lemma 4.4.2. *The systematic tableau is finished.*

Proof. We will show that each branch is finished. Contradictory branches are finished. Non-contradictory branches contain entries $\mathrm{T}\alpha_i$ (which we added in the i-th step), and every entry on them is reduced. Indeed, if E were not reduced on a non-contradictory branch B, it would be processed in some step since there are only finitely many entries above E and to the left of E. (We use the obvious fact that any prefix of a non-contradictory branch is also a non-contradictory branch, so during the construction, B is never contradictory.) □

Now, let us return to the question of finiteness of proofs:

Theorem 4.4.3 (Finiteness of Contradiction)**.** *If $\tau = \bigcup_{i\geq 0} \tau_i$ is a contradictory tableau, then there exists $n \in \mathbb{N}$ such that τ_n is a finite contradictory tableau.*

Proof. Consider the set S of all nodes of the tree τ that do not contain a contradiction above them (in the tree order), i.e., a pair of entries $\mathrm{T}\psi$, $\mathrm{F}\psi$.

[6]Recall that T is countable since the language is (in the entire chapter) countable.

If the set S were infinite, by König's lemma applied to the subtree of τ on the set S, we would have an infinite, non-contradictory branch in S. However, this would mean that we have a non-contradictory branch in τ, which contradicts the fact that τ is contradictory. (In detail: The branch on S would be a sub-branch of some branch B in τ, which is contradictory, i.e., it contains some (specific) contradictory pair of entries, which exists already in some finite prefix of B.)

Therefore, the set S is finite. This means that there exists $d \in \mathbb{N}$ such that the entire S lies at a depth of at most d. Thus, every node at level $d+1$ has a contradiction above it. Choose n such that τ_n already contains all the nodes of τ from the first $d+1$ levels: each branch of τ_n is therefore contradictory. □

Corollary 4.4.4. *If we never extend contradictory branches during the construction of a tableau (e.g., for systematic tableau), then a contradictory tableau is finite.*

Proof. We use Theorem 4.4.3, and we have $\tau = \tau_n$ since we do not further extend the tableau once it is contradictory. □

Corollary 4.4.5 (Finiteness of Proofs)**.** *If $T \vdash \varphi$, then there exists a* finite *tableau proof of φ from T.*

Proof. It easily follows from Corollary 4.4.4: during the construction of τ, we ignore steps that would extend a contradictory branch. □

We also state the following corollary. We will prove it in the next section.

Corollary 4.4.6 (Systematicity of Proofs)**.** *If $T \vdash \varphi$, then the systematic tableau is a (finite) tableau proof of φ from T.*

To prove this, we will need two facts: if φ is provable from T, then it holds in T (Soundness Theorem), i.e., no counterexample can exist. And if the systematic tableau had a non-contradictory branch, it would mean that a counterexample exists (which is the key to the Completeness Theorem).

4.5 Soundness and Completeness

In this section, we will prove that the tableau method is a *sound* and *complete* proof system, i.e., that $T \vdash \varphi$ holds if and only if $T \models \varphi$.

4.5.1 Soundness Theorem

We say that a model v *agrees* with an entry E if $E = \mathrm{T}\varphi$ and $v \models \varphi$, or $E = \mathrm{F}\varphi$ and $v \not\models \varphi$. Furthermore, v agrees with a branch B if it agrees with every entry on that branch.

As already mentioned, the design of atomic tableaux ensures that if a model agrees with the entry at the root of the tableau, it agrees with some branch. It is not difficult to show the following lemma by induction on the construction of the tableau:

Lemma 4.5.1. *If a model of a theory T agrees with the root entry of a tableau from T, then it agrees with some branch.*

Proof. Consider a tableau $\tau = \bigcup_{i\geq 0} \tau_i$ from the theory T and a model $v \in \mathrm{M}(T)$ that agrees with the root of τ, i.e., with the (single-element) branch B_0 in the (single-element) tableau τ_0.

By induction on i (steps in the construction of the tableau), we find a sequence $B_0 \subseteq B_1 \subseteq \dots$ such that B_i is a branch in the tableau τ_i agreeing with the model v, and B_{i+1} extends B_i. The branch of the tableau τ we are looking for is then $B = \bigcup_{i\geq 0} B_i$.

- If τ_{i+1} is obtained from τ_i without extending the branch B_i, we define $B_{i+1} = B_i$.
- If τ_{i+1} is obtained from τ_i by adding the entry $\mathrm{T}\alpha$ (for some axiom $\alpha \in T$) to the end of the branch B_i, we define B_{i+1} as this extended branch. Since v is a model of T, it satisfies the axiom α, so it agrees with the new entry $\mathrm{T}\alpha$.
- Let τ_{i+1} be obtained from τ_i by adding the atomic tableau for some entry E to the end of the branch B_i. Since the model v agrees with the entry E (which is already on the branch B_i), it agrees with the root of the appended atomic tableau and thus with some branch of the atomic tableau. (This property is easily verified for all atomic tableaux.) We define B_{i+1} as the extension of B_i by this branch of the atomic tableau.[7]

□

We are now ready to prove the Soundness Theorem. In short, if both a proof and a counterexample existed, the counterexample would have to agree with some branch of the proof, but all branches are contradictory.

Theorem 4.5.2 (Soundness)**.** *If a proposition φ is tableau provable from a theory T, then φ is true in T, i.e., $T \vdash \varphi \Rightarrow T \models \varphi$.*

Proof. The proof is by contradiction. Assume that φ is not true in T, i.e., there is a counterexample: a model $v \in \mathrm{M}(T)$ in which φ does not hold.

Since φ is provable from T, there exists a tableau proof of φ from T, which is a contradictory tableau from T with root entry $\mathrm{F}\varphi$. The model v agrees with the entry $\mathrm{F}\varphi$, so by Lemma 4.5.1, it agrees with some branch B. However, all branches are contradictory, including B. Therefore, B contains the entries $\mathrm{T}\psi$ and $\mathrm{F}\psi$ (for some proposition ψ), and the model v agrees with these entries. Thus, we have $v \models \psi$ and simultaneously $v \not\models \psi$, which is a contradiction. □

4.5.2 Completeness Theorem

We will show that if proving fails, i.e., if we obtain a *non-contradictory* branch in a *finished* tableau from the theory T with root entry $\mathrm{F}\varphi$, then this branch provides a counterexample: a model of the theory T that agrees with the entry $\mathrm{F}\varphi$ at the root of the tableau, i.e., it does not satisfy φ. There may be multiple such models; we will define a specific one:

Definition 4.5.3 (Canonical Model)**.** If B is a non-contradictory branch of a finished tableau, then the *canonical model* for B is a model defined by the rule (for $p \in \mathbb{P}$):

$$v(p) = \begin{cases} 1 \text{ if the entry } \mathrm{T}p \text{ appears on } B, \\ 0 \text{ otherwise.} \end{cases}$$

[7]Or by any such branch: the model v may agree with more than one branch of the atomic tableau.

Lemma 4.5.4. *The canonical model for a (non-contradictory finished) branch B agrees with B.*

Proof. We will show that the canonical model v agrees with all entries E on the branch B, by induction on the structure of the proposition in the entry.[8] First, the base case of the induction:

- If $E = \mathrm{T}p$ for some atomic proposition $p \in \mathbb{P}$, we have $v(p) = 1$ by definition; v agrees with E.
- If $E = \mathrm{F}p$, then the entry $\mathrm{T}p$ cannot appear on the branch B, otherwise B would be contradictory. By definition, we have $v(p) = 0$, and v again agrees with E.

Now, the induction step. We will cover two cases; the other cases are proved similarly.

- Let $E = \mathrm{T}\varphi \wedge \psi$. Since B is a finished branch, the entry E is reduced on it. This means that the entries $\mathrm{T}\varphi$ and $\mathrm{T}\psi$ appear on B. By the induction hypothesis, the model v agrees with these entries, so $v \models \varphi$ and $v \models \psi$. Thus, $v \models \varphi \wedge \psi$ holds, and v agrees with E.
- Let $E = \mathrm{F}\varphi \wedge \psi$. Since E is reduced on B, the entry $\mathrm{F}\varphi$ or the entry $\mathrm{F}\psi$ appears on B. Therefore, $v \not\models \varphi$ or $v \not\models \psi$, from which it follows that $v \not\models \varphi \wedge \psi$, and v agrees with E.

□

Theorem 4.5.5 (Completeness). *If a proposition φ is true in a theory T, then it is tableau provable from T, i.e., $T \models \varphi \Rightarrow T \vdash \varphi$.*

Proof. We will show that any *finished* (thus, for example, *systematic*) tableau from T with root entry $\mathrm{F}\varphi$ must be contradictory. The proof is by contradiction: If such a tableau were not contradictory, it would contain a non-contradictory (finished) branch B. Consider the canonical model v for this branch. Since B is finished, it contains the entry $\mathrm{T}\alpha$ for all axioms $\alpha \in T$. By Lemma 4.5.4, the model v agrees with all entries on B, thus satisfying all the axioms and we have $v \models T$. However, since v also agrees with the root entry $\mathrm{F}\varphi$, we have $v \not\models \varphi$, which means $T \not\models \varphi$, a contradiction. Thus, the tableau must have been contradictory, i.e., a tableau proof of φ from T. □

Proof of Corollary 4.4.6. From the previous proof, we also obtain the 'systematicity of proofs,' i.e., that a proof can always be found by constructing a systematic tableau: If $T \models \varphi$, then the systematic tableau for the entry $\mathrm{F}\varphi$ must be contradictory, and thus it is a tableau proof of φ from T. □

Exercise 4.5. Verify the remaining cases in the proof of Lemma 4.5.4.

Exercise 4.6. Describe what *all* models agreeing with a given non-contradictory finished branch look like.

Exercise 4.7. Suggest a procedure to use the tableau method to find all models of a given theory T.

[8] Recall that this means induction on the depth of the proposition tree.

4.6 Consequences of Soundness and Completeness

The Soundness and Completeness Theorems together state that *provability* is the same as *validity*. This allows us to formulate syntactic analogues of semantic concepts and properties.

The analogue of *consequences* are *theorems* of the theory T:

$$\mathrm{Thm}_{\mathbb{P}}(T) = \{\varphi \in \mathrm{PF}_{\mathbb{P}} \mid T \vdash \varphi\}$$

Corollary 4.6.1 (Provability = Validity)**.** *For any theory T and propositions φ, ψ, the following hold:*

- *$T \vdash \varphi$ if and only if $T \models \varphi$*
- $\mathrm{Thm}_{\mathbb{P}}(T) = \mathrm{Csq}_{\mathbb{P}}(T)$

Proof. Follows immediately from the Soundness and Completeness Theorems. □

In all definitions and theorems, we can therefore replace the notion of '*validity*' with the notion of '*provability*' (i.e., the symbol '$\models$' with the symbol '$\vdash$') and the notion of '*consequence*' with the notion of '*theorem*'. For example:

- A theory is *contradictory* if it proves a contradiction (i.e., $T \vdash \bot$).
- A theory is *complete* if for every proposition φ, either $T \vdash \varphi$ or $T \vdash \neg\varphi$ (but not both, otherwise it would be contradictory).

Let us state one more easy consequence:

Theorem 4.6.2 (Deduction Theorem)**.** *For a theory T and propositions φ, ψ, we have that: $T, \varphi \vdash \psi$ if and only if $T \vdash \varphi \to \psi$.*

Proof. It suffices to prove that $T, \varphi \models \psi \Leftrightarrow T \models \varphi \to \psi$, which is straightforward. □

Exercise 4.8. Prove the Deduction Theorem directly by a transformation of tableau proofs.

4.7 Compactness Theorem

An important consequence of the Soundness and Completeness Theorems is the so-called *Compactness Theorem.*[9] This principle allows for converting statements about infinite objects or processes into statements about (all) their finite parts.

Theorem 4.7.1 (Compactness Theorem)**.** *A theory has a model if and only if every finite subset of it has a model.*

Proof. Every model of a theory T is evidently a model of each of its subsets. For the other implication, we will prove its contrapositive: Assume that T has no model, i.e., it is contradictory, and find a finite part $T' \subseteq T$ that is also contradictory.

Since T is contradictory, we have $T \vdash \bot$ (here we need the Completeness Theorem). According to Corollary 4.4.5, there exists a *finite* tableau proof τ of the contradiction $\bot$

[9]The word *compactness* comes from compact (i.e., bounded and closed) sets in Euclidean spaces, where one can select a convergent subsequence from every sequence. You can imagine a sequence of increasing finite parts 'converging' to the infinite whole.

from T. The construction of this proof involves only finitely many steps, so we used only finitely many axioms from T. If we define $T' = \{\alpha \in T \mid \mathrm{T}\alpha \text{ is an entry in the tableau } \tau\}$, then τ is also a tableau proof of the contradiction from the theory T'. Therefore, T' is a finite contradictory part of T. □

4.7.1 Applications of Compactness

The following simple application of the Compactness Theorem can be seen as a template followed by many other, more complex applications of this theorem.

Corollary 4.7.2. *A countably infinite graph is bipartite if and only if every finite subgraph of it is bipartite.*

Proof. Every subgraph of a bipartite graph is evidently also bipartite. We will prove the other implication. A graph is bipartite if and only if it is 2-colorable. Let the colors be 0 and 1.

We will construct a propositional theory T in the language $\mathbb{P} = \{p_v \mid v \in V(G)\}$, where the value of the propositional variable p_v represents the color of the vertex v.

$$T = \{p_u \leftrightarrow \neg p_v \mid \{u, v\} \in E(G)\}$$

Clearly, G is bipartite if and only if T has a model. According to the Compactness Theorem, it suffices to show that every finite part of T has a model. Consider a finite $T' \subseteq T$. Let G' be the subgraph of G induced by the set of vertices mentioned in the theory T', i.e., $V(G') = \{v \in V(G) \mid p_v \in \mathrm{Var}(T')\}$. Since T' is finite, G' is also finite, and by assumption, it is 2-colorable. Any 2-coloring of $V(G')$ determines a model of the theory T'. □

The essence of this technique is to describe the desired property of an infinite object using an (infinite) propositional theory. Additionally, note how we construct a finite subobject having the given property from a finite part of the theory (in our case, a finite subgraph that is bipartite).

Exercise 4.9. Generalize Corollary 4.7.2 for more colors, i.e., show that a countably infinite graph is k-colorable if and only if every finite subgraph of it is k-colorable. (See Section 1.1.7.)

Exercise 4.10. Show that every partial order on a countable set can be extended to a linear order.

Exercise 4.11. State and prove the 'countably infinite' analog of Hall's theorem.

4.8 Hilbert Calculus

At the end of the chapter on the tableau method, we will for comparison describe another proof system, the so-called *Hilbert deductive system* or *Hilbert calculus*. This is the oldest proof system, modeled after mathematical proofs. As we will see from the example, proving in it is quite laborious, so it is more suitable for theoretical purposes. It is also a sound and complete proof system. (We will show soundness, but leave completeness without proof.)

The Hilbert calculus uses only two basic logical connectives: negation and implication. (Recall that other logical connectives can be derived from these.) The system consists of logical axioms given by the following *schemata* and a single *inference rule*, the so-called *modus ponens*:

Definition 4.8.1 (Axiom Schemata in Hilbert Calculus)**.** For any propositions φ, ψ, χ, the following propositions are logical axioms:

(i) $\varphi \to (\psi \to \varphi)$

(ii) $(\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))$

(iii) $(\neg\varphi \to \neg\psi) \to (\psi \to \varphi)$

Note that all logical axioms are indeed tautologies. It should be noted that other systems of logical axioms can be chosen; there are many, see the article List of Hilbert systems on Wikipedia.

Definition 4.8.2 (Modus Ponens)**.** The *modus ponens* inference rule states that if we have already proved the proposition φ and also the proposition $\varphi \to \psi$, we can infer the proposition ψ. We write it as follows:

$$\frac{\varphi, \varphi \to \psi}{\psi}$$

Note that modus ponens is *sound*, i.e., if $T \models \varphi$ and $T \models \varphi \to \psi$ hold in some theory, we also have $T \models \psi$.

We are now ready to define a *proof.* It will be a finite sequence of propositions, in which each newly written proposition is either an axiom (logical or from the theory in which we are proving), or can be inferred from some previous ones using modus ponens:

Definition 4.8.3 (Hilbert Proof)**.** A *Hilbert proof* of a proposition φ from a theory T is a *finite* sequence of propositions $\varphi_0, \dots, \varphi_n = \varphi$, in which for each $i \leq n$, the following holds:

- φ_i is a logical axiom, or
- φ_i is an axiom of the theory ($\varphi_i \in T$), or
- φ_i can be derived from some previous propositions φ_j, φ_k (where $j, k < i$) using modus ponens.

If a Hilbert proof exists, we say that φ is *(Hilbert) provable*, and we write $T \vdash_H \varphi$.

We will illustrate the concept of a Hilbert proof with a simple example:

Example 4.8.4. Let us show that for the theory $T = \{\neg\varphi\}$ and for any proposition ψ, we have $T \vdash_H \varphi \to \psi$. The Hilbert proof is the following sequence of propositions:

1.	$\neg\varphi$	*axiom of the theory*
2.	$\neg\varphi \to (\neg\psi \to \neg\varphi)$	*logical axiom by (i)*
3.	$\neg\psi \to \neg\varphi$	*modus ponens on 1 and 2*
4.	$(\neg\psi \to \neg\varphi) \to (\varphi \to \psi)$	*logical axiom by (iii)*
5.	$\varphi \to \psi$	*modus ponens on 3 and 4*

As mentioned earlier, the Hilbert calculus is a sound and complete proof system.

Theorem 4.8.5 (Soundness of Hilbert Calculus)**.** *For any theory T and proposition φ:*

$$T \vdash_H \varphi \Rightarrow T \models \varphi$$

Proof. By induction on the index i, we will show that every proposition φ_i in the proof (thus also $\varphi_n = \varphi$) holds in T.

If φ_i is a logical axiom, $T \models \varphi_i$ holds because logical axioms are tautologies. If $\varphi_i \in T$, then clearly $T \models \varphi_i$ also holds. If φ_i is obtained using modus ponens from φ_j and $\varphi_k = \varphi_j \to \varphi_i$ (for some $j, k < i$), we know from the induction hypothesis that $T \models \varphi_j$ and $T \models \varphi_j \to \varphi_i$. Then, by the soundness of modus ponens, $T \models \varphi_i$ also holds. □

For completeness, we will state the Completeness theorem but leave it without proof.

Theorem 4.8.6 (Completeness of Hilbert Calculus)**.** *For any theory T and proposition φ:*

$$T \models \varphi \Rightarrow T \vdash_H \varphi$$

Chapter 5

Resolution Method

In this chapter, we introduce another proof system more suitable for practical applications, called the *resolution method.* This method is the foundation of, among others, *logical programming*, *automated theorem proving*, or *software verification.* In this chapter, we limit ourselves to the resolution method in propositional logic, but later, in Chapter **??**, we will introduce the concept of *unification*, which allows us to search for resolution proofs in predicate logic.

The resolution method works with propositions in *conjunctive normal form (CNF).* Recall that every proposition can be converted to CNF. This conversion is, in the worst case, of exponential time copmlexity (there are even propositions whose shortest CNF equivalent is exponentially longer), but in practice, this is not an issue.

Similar to the tableau method, the resolution method is based on proof by contradiction, i.e., we add the *negation* of the proposition we want to prove to the theory in which we are proving (both converted to CNF), and show that this leads to a contradiction.

To find the contradiction, the resolution method uses a single inference rule called the *resolution rule.* This is a special case of the *cut rule*, which states: *"From the propositions $\varphi \vee \psi$ and $\neg\varphi \vee \chi$, we can infer the proposition $\psi \vee \chi$,"* written as:

$$\frac{\varphi \vee \psi, \neg\varphi \vee \chi}{\psi \vee \chi}$$

In the *resolution rule*, which we will demonstrate shortly, φ is a *literal*, and ψ, χ are *clauses.*

Exercise 5.1. Show that the cut rule is *sound.* (What does it mean?)

5.1 Set Representation

First, we introduce a more compact notation for CNF propositions, called *set notation.* For it would be impractical to represent CNF propositions as strings including brackets and logical symbols.

- Recall that a *literal* ℓ is a propositional variable or its negation, and that $\overline{\ell}$ denotes the *opposite literal* to ℓ.
- A *clause* C is a finite set of literals. The *empty clause*, which is never satisfied,[1] is denoted by $\square$.

[1]It represents the disjunction of an empty set of literals, meaning none of the disjuncts are satisfied.

- A *(CNF) formula* S is a (finite, or even infinite) set of clauses. The *empty formula* $\emptyset$ is always satisfied.[2]

Remark 5.1.1. Note that a *CNF formula* can also be an *infinite* set of clauses. If we convert an infinite propositional theory to CNF, we take (in set representation) all infinitely many clauses as elements of a single formula (set). In practical applications, the formula is (almost always) finite.

In set notation, models correspond to sets of literals that contain exactly one of the literals $p, \neg p$ for each propositional variable p:

- A *(partial) assignment* $\mathcal{V}$ is any set of literals that is *consistent*, i.e., it does not contain both a literal and its negation.
- An assignment is *complete* if it contains either the positive or negative literal for each propositional variable.
- An assignment $\mathcal{V}$ *satisfies* a formula S, written $\mathcal{V} \models S$, if $\mathcal{V}$ contains some literal from each clause in S, i.e.,
$$\mathcal{V} \cap C \neq \emptyset \text{ for each } C \in S.$$

Example 5.1.2. The proposition $\varphi = (\neg p_1 \vee p_2) \wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_3 \vee \neg p_4) \wedge (\neg p_4 \vee \neg p_5) \wedge p_4$ is written in set notation as:

$$S = \{\{\neg p_1, p_2\}, \{\neg p_1, \neg p_2, p_3\}, \{\neg p_3, \neg p_4\}, \{\neg p_4, \neg p_5\}, \{p_4\}\}$$

The assignment $\mathcal{V} = \{\neg p_1, \neg p_3, p_4, \neg p_5\}$ satisfies S, written $\mathcal{V} \models S$. It is not complete, but we can extend it with any literal for p_2: both $\mathcal{V} \cup \{p_2\} \models S$ and $\mathcal{V} \cup \{\neg p_2\} \models S$ hold. These two complete assignments correspond to the models $(0, 1, 0, 1, 0)$ and $(0, 0, 0, 1, 0)$.

5.2 Resolution Proof

First, we define one inference step in the resolution proof, the *resolution rule*, which we apply to a pair of clauses; the result is a clause called the *resolvent*, which is a logical consequence of the original pair of clauses:

Definition 5.2.1 (Resolution Rule). Given clauses C_1 and C_2 and a literal ℓ such that $\ell \in C_1$ and $\bar{\ell} \in C_2$, the *resolvent* of the clauses C_1 and C_2 *over the literal* ℓ is the clause

$$C = (C_1 \setminus \{\ell\}) \cup (C_2 \setminus \{\bar{\ell}\}).$$

So, we remove the literal ℓ from the first clause and the literal $\bar{\ell}$ from the second clause (both had to be there!) and take the union of all remaining literals to be the resulting resolvent. Using the symbol $\dot\sqcup$ for disjoint union, we could also write:

$$C_1' \cup C_2' \text{ is the resolvent of the clauses } C_1' \dot\sqcup \{\ell\} \text{ and } C_2' \dot\sqcup \{\bar{\ell}\}$$

Example 5.2.2. From the clauses $C_1 = \{\neg q, r\}$ and $C_2 = \{\neg p, \neg q, \neg r\}$, we can derive the resolvent $\{\neg p, \neg q\}$ over the literal r. From the clauses $\{p, q\}$ and $\{\neg p, \neg q\}$, we can derive $\{p, \neg p\}$ over the literal q or $\{q, \neg q\}$ over the literal p (both of which are tautologies).[3]

[2]It represents the conjunction of an empty set of clauses, meaning all clauses in S are satisfied.

[3]We cannot, however, derive $\square$ 'by resolving over p and q simultaneously' (a common mistake). Note that $\{\{p, q\}, \{\neg p, \neg q\}\}$ is not unsatisfiable, e.g., $(1, 0)$ is a model.

Observation 5.2.3 (Soundness of the Resolution Rule)**.** *The resolution rule is sound, i.e., for any assignment $\mathcal{V}$, the following holds:*

$$\text{If } \mathcal{V} \models C_1 \text{ and } \mathcal{V} \models C_2, \text{ then } \mathcal{V} \models C.$$

A resolution proof is defined similarly to a Hilbert proof as a finite sequence of clauses, ensuring the validity of each clause in the sequence: at each step, we can either write an 'axiom' (a clause from S) or a resolvent of some two previously written clauses.

Definition 5.2.4 (Resolution Proof)**.** A *resolution proof* of a clause C from a CNF formula S is a *finite* sequence of clauses $C_0, C_1, \ldots, C_n = C$ such that for each i, either $C_i \in S$ or C_i is the resolvent of some C_j, C_k where $j < i$ and $k < i$.

If a resolution proof exists, we say that C is *resolution provable* from S, written $S \vdash_R C$. A *(resolution) refutation* of the CNF formula S is a resolution proof of $\square$ from S, in which case S is *(resolution) refutable.*

Example 5.2.5. The CNF formula $S = \{\{p, \neg q, r\}, \{p, \neg r\}, \{\neg p, r\}, \{\neg p, \neg r\}, \{q, r\}\}$ is (resolution) refutable, one possible refutation is:

$$\{p, \neg q, r\}, \{q, r\}, \{p, r\}, \{\neg p, r\}, \{r\}, \{p, \neg r\}, \{\neg p, \neg r\}, \{\neg r\}, \square$$

A resolution proof has a natural tree structure: the leaves are labeled by axioms and the inner nodes represent individual resolution steps.

Definition 5.2.6 (Resolution Tree)**.** A resolution tree of a clause C from a CNF formula S is a *finite* binary tree with nodes labeled by clauses, where:

- The root is labeled C,
- The leaves are labeled by clauses from S,
- Each inner node is labeled by a resolvent of its child nodes.

Example 5.2.7. The resolution tree of the empty clause $\square$ from the CNF formula S from Example 5.2.5 is:

$$\begin{tikzcd}
\{p,\neg q,r\} \arrow[dr, no head] & & \{q,r\} \arrow[dl, no head] & & & & & \\
& \{p,r\} \arrow[dr, no head] & & \{\neg p,r\} \arrow[dl, no head] & & \{p,\neg r\} \arrow[dr, no head] & & \{\neg p,\neg r\} \arrow[dl, no head] \\
& & \{r\} \arrow[drr, no head] & & & & \{\neg r\} \arrow[dll, no head] & \\
& & & & \square & & &
\end{tikzcd}$$

It is easy to prove the following observation, by induction on the depth of the tree and the length of the resolution proof:

Observation 5.2.8. *A clause C has a resolution tree from a CNF formula S if and only if $S \vdash_R C$.*

Each resolution proof corresponds to a unique resolution tree. Conversely, from a single resolution tree, we can derive multiple resolution proofs: they are given by any traversal of the tree nodes where an inner node is visited only after both of its children have been visited.

We also introduce another concept, called the *resolution closure*, which contains all clauses that can be 'learned' by resolution from a given formula. It is a useful theoretical perspective on resolution; in applications, constructing the entire resolution closure would be impractical.

Definition 5.2.9 (Resolution Closure)**.** The *resolution closure* $\mathcal{R}(S)$ of the formula S is defined inductively as the smallest set of clauses satisfying:

- $C \in \mathcal{R}(S)$ for all $C \in S$,
- If $C_1, C_2 \in \mathcal{R}(S)$ and C is the resolvent of C_1, C_2, then $C \in \mathcal{R}(S)$.

Example 5.2.10. Let us compute the resolution closure of the formula $S = \{\{p, \neg q, r\}, \{p, \neg r\}, \{\neg p, r\}, \{\neg p, \neg r\}$
The clauses from S are in blue; additional clauses are obtained by resolving them in pairs (first with the first, second with the first, second with the second, etc., over all possible literals):

$$\begin{aligned}\mathcal{R}(S) = \{&\{p, \neg q, r\}, \{p, \neg r\}, \{\neg p, r\}, \{\neg p, \neg r\}, \{q, r\},\\ &\{p, \neg q\}, \{\neg q, r\}, \{r, \neg r\}, \{p, \neg p\}, \{r, \neg s\}, \{p, r\}, \{p, q\}, \{r\}, \{p\}\}\end{aligned}$$

5.3 Soundness and Completeness of the Resolution Method

The resolution method is also both sound and complete.

5.3.1 Soundness of Resolution

Soundness can be easily proved by induction on the length of the resolution proof.

Theorem 5.3.1 (Soundness of Resolution)**.** *If a formula S is resolution refutable, then S is unsatisfiable.*

Proof. Suppose $S \vdash_R \square$ and consider a resolution proof $C_0, C_1, \ldots, C_n = \square$. Suppose, for a contradiction, that S is satisfiable, i.e., $\mathcal{V} \models S$ for some assignment $\mathcal{V}$. By induction on i, we prove that $\mathcal{V} \models C_i$. For $i = 0$, this holds because $C_0 \in S$. For $i > 0$, there are two cases:

- $C_i \in S$, in which case $\mathcal{V} \models C_i$ follows from the assumption that $\mathcal{V} \models S$,
- C_i is the resolvent of C_j, C_k, where $j, k < i$: by the induction hypothesis, $\mathcal{V} \models C_j$ and $\mathcal{V} \models C_k$. $\mathcal{V} \models C_i$ follows from the soundness of the resolution rule.

(Alternatively, we could proceed by induction on the depth of the resolution tree.) $\square$

5.3.2 Substitution Tree

In the completeness proof, we need to construct a resolution tree, whose construction is based on the so-called *substitution tree*. By *substituting* a literal into a formula, we mean simplifying the formula under the assumption that the given literal is true. Substitution was already encountered in Section 3.3 during *unit propagation*: we remove clauses containing this literal and remove the opposite literal from the remaining clauses.

Definition 5.3.2 (Literal Substitution)**.** Given a formula S and a literal ℓ, the *substitution* of ℓ into S is the formula:

$$S^{\ell} = \{C \setminus \{\bar{\ell}\} \mid \ell \notin C \in S\}$$

Observation 5.3.3. *Here we summarize several simple facts about substitution:*

- S^{ℓ} *is the result of* unit propagation *applied to* $S \cup \{\{\ell\}\}$.
- S^{ℓ} *does not contain the literal* ℓ *or its opposite* $\bar{\ell}$ *(it does not contain the propositional variable from* ℓ *at all).*
- *If* S *did not contain the literal* ℓ *or its opposite* $\bar{\ell}$*, then* $S^{\ell} = S$*.*
- *If* S *contained the unit clause* $\{\bar{\ell}\}$*, then* $\square \in S^{\ell}$*, meaning* S^{ℓ} *is unsatisfiable.*

The key property of substitution is expressed by the following lemma:

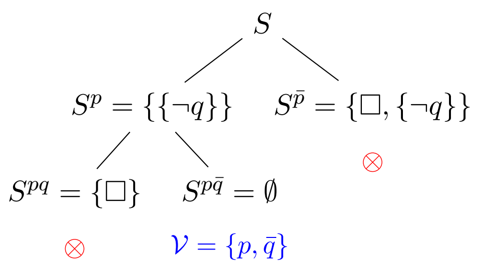
Lemma 5.3.4. *S is satisfiable if and only if S^{ℓ} or $S^{\bar{\ell}}$ is satisfiable.*

Proof. Let $\mathcal{V} \models S$, it cannot contain both ℓ and $\bar{\ell}$ (it must be consistent); without loss of generality, assume $\bar{\ell} \notin \mathcal{V}$, and show that $\mathcal{V} \models S^{\ell}$. Consider any clause in S^{ℓ}. It is of the form $C \setminus \{\bar{\ell}\}$ for a clause $C \in S$ (not containing the literal ℓ). We know that $\mathcal{V} \models C$, but since $\mathcal{V}$ does not contain $\bar{\ell}$, the assignment $\mathcal{V}$ must satisfy some other literal of C, so $\mathcal{V} \models C \setminus \{\bar{\ell}\}$.

Conversely, suppose there exists an assignment $\mathcal{V}$ satisfying S^{ℓ} (again, without loss of generality). Since $\bar{\ell}$ (or ℓ) does not appear in S^{ℓ}, it holds that $\mathcal{V} \setminus \{\bar{\ell}\} \models S^{\ell}$. The assignment $\mathcal{V}' = (\mathcal{V} \setminus \{\bar{\ell}\}) \cup \{\ell\}$ then satisfies each clause $C \in S$: if $\ell \in C$, then $\ell \in C \cap \mathcal{V}'$ and $C \cap \mathcal{V}' \neq \emptyset$, otherwise $C \cap \mathcal{V}' = (C \setminus \{\bar{\ell}\}) \cap \mathcal{V}' \neq \emptyset$ because $\mathcal{V} \setminus \{\bar{\ell}\} \models C \setminus \{\bar{\ell}\} \in S^{\ell}$. We have verified that $\mathcal{V}' \models S$, so S is satisfiable. $\square$

Whether a given *finite* formula is satisfiable can thus be determined recursively (using the *divide and conquer* method) by substituting both possible literals for (some, say the first) propositional variable appearing in the formula and branching the computation. Essentially, this is a similar principle to the DPLL algorithm (see Section 3.4). The resulting tree is called a *substitution tree.*

Example 5.3.5. Let's illustrate this concept with an example by constructing a substitution tree for the formula $S = \{\{p\}, \{\neg q\}, \{\neg p, \neg q\}\}$:


S
$S^{p} = \{\{\neg q\}\}$
$S^{\bar{p}} = \{\square, \{\neg q\}\}$
$\otimes$
$S^{pq} = \{\square\}$
$S^{p\bar{q}} = \emptyset$
$\otimes$
$\mathcal{V} = \{p, \bar{q}\}$

Once a branch contains the empty clause $\square$, it is unsatisfiable, and we need not continue in that branch. In the leaves, there are either unsatisfiable theories or empty theories: in the latter case, the sequence of substitutions gives us a satisfying assignment.

From the construction, it is evident how to proceed in the case of a finite formula. However, the substitution tree makes sense, and the following corollary holds, even for infinite formulas:

Corollary 5.3.6. *A formula S (over a countable language) is unsatisfiable if and only if every branch of the substitution tree contains the empty clause $\square$.*

Proof. For a finite formula S, this follows from the above discussion; it can be easily proved by induction on the size of $\mathrm{Var}(S)$:

- If $|\mathrm{Var}(S)| = 0$, we have $S = \emptyset$ or $S = \{\square\}$, in both cases, the substitution tree is one node, and the statement holds.
- In the inductive step, we choose any literal $\ell \in \mathrm{Var}(S)$ and apply Lemma 5.3.4.

If S is infinite and satisfiable, it has a satisfying assignment, which 'matches' the corresponding (infinite) branch in the substitution tree. If S is infinite and unsatisfiable, then by the Compactness Theorem, there is a finite part $S' \subseteq S$ that is also unsatisfiable. Substitution for all variables from $\mathrm{Var}(S')$ will lead to $\square$ in each branch, which will happen after finitely many steps. $\square$

5.3.3 Completeness of Resolution

Theorem 5.3.7 (Completeness of Resolution)**.** *If S is unsatisfiable, it is resolution refutable (i.e., $S \vdash_R \square$).*

Proof. If S is infinite, it has a finite unsatisfiable part S' by the Compactness Theorem. A resolution refutation of S' is also a resolution refutation of S. Suppose S is finite.

The proof is by induction on the number of variables in S. If $|\mathrm{Var}(S)| = 0$, the only possible unsatisfiable formula without variables is $S = \{\square\}$, and we have a one-step proof $S \vdash_R \square$. Otherwise, choose $p \in \mathrm{Var}(S)$. By Lemma 5.3.4, both S^p and $S^{\bar{p}}$ are unsatisfiable. They have one fewer variable, so by the induction hypothesis, there are resolution trees T for $S^p \vdash_R \square$ and T' for $S^{\bar{p}} \vdash_R \square$.

We show how to construct the resolution tree $\widehat{T}$ for $S \vdash_R \neg p$. Similarly, we construct $\widehat{T'}$ for $S \vdash_R p$, and then easily construct the resolution tree for $S \vdash_R \square$: we attach the roots of the trees $\widehat{T}$ and $\widehat{T'}$ as the left and right children of the root $\square$ (i.e., in the last step of the resolution proof, we obtain $\square$ by resolving $\{\neg p\}$ and $\{p\}$).

It remains to show the construction of the tree $\widehat{T}$: the set of nodes and the order remain the same; we only change some clauses in the nodes by adding the literal $\neg p$. At each leaf of the tree T is some clause $C \in S^p$, and either $C \in S$ or it is not, but $C \cup \{\neg p\} \in S$. In the first case, we leave the label unchanged. In the second case, we add the literal $\neg p$ to C *and to all clauses above this leaf.* In the leaves, there are now only clauses from S, in the root, we have changed $\square$ to $\neg p$. And each inner node remains the resolvent of its children. $\square$

Exercise 5.2. The proof of the Completeness Theorem for resolution gives a method for recursively 'growing' a resolution refutation. Think about how to do this and apply it to an example of an unsatisfiable formula.

5.4 LI-Resolution and Horn-SAT

We begin with a different view of the resolution proof, called the *linear proof.*

5.4.1 Linear Proof

In addition to the resolution tree, a resolution proof can also be organized as a *linear proof*, where in each step we have one *central* clause, which we resolve with a *side* clause, which is either one of the previous central clauses or an axiom from S. The resolvent then becomes the new central clause.[4]

Definition 5.4.1 (Linear Proof)**.** A *linear proof* (by resolution) of a clause C from a formula S is a finite sequence

$$\begin{bmatrix} C_0 \\ B_0 \end{bmatrix}, \begin{bmatrix} C_1 \\ B_1 \end{bmatrix}, \ldots, \begin{bmatrix} C_n \\ B_n \end{bmatrix}, C_{n+1}$$

where C_i are called *central* clauses, C_0 is the *initial* clause, $C_{n+1} = C$ is the *final* clause, B_i are *side* clauses, and it holds that:

- $C_0 \in S$, for $i \leq n$, C_{i+1} is the resolvent of C_i and B_i,
- $B_0 \in S$, for $i \leq n$, $B_i \in S$ or $B_i = C_j$ for some $j < i$.

A *linear refutation* of S is a linear proof of $\square$ from S. A linear proof can be illustrated as follows:

$$\begin{tikzcd}
C_0 \arrow[r, dash] & C_1 \arrow[r, dash] & C_2 \arrow[r, dash] & \cdots\cdots\cdots \arrow[r, dash] & C_n \arrow[r, dash] & C_{n+1} \\
B_0 \arrow[ur, dash] & B_1 \arrow[ur, dash] & & B_{n-1} \arrow[ur, dash] & B_n \arrow[ur, dash] &
\end{tikzcd}$$

Example 5.4.2. Let's construct a linear refutation of the formula $S = \{\{p, q\}, \{p, \neg q\}, \{\neg p, q\}, \{\neg p, \neg q\}\}$ (i.e., a linear proof of $\square$ from S). A linear proof might look like this:

$$\begin{tikzcd}
\{p, q\} \arrow[r, dash] & \{p\} \arrow[r, dash] & \{q\} \arrow[r, dash] & \{\neg p\} \arrow[r, dash] & \square \\
\{p, \neg q\} \arrow[ur, dash] & \{\neg p, q\} \arrow[ur, dash] & \{\neg p, \neg q\} \arrow[ur, dash] & \{p\} \arrow[ur, dash] &
\end{tikzcd}$$

The last side clause $\{p\}$ (in red) is not from S, but is equal to the previous central clause (in blue).

Exercise 5.3. Convert the linear proof from Example 5.4.2 into a resolution tree.

Remark 5.4.3. C has a linear proof from S if and only if $S \vdash_R C$.

A resolution tree can easily be produced from a linear proof by induction on the length of the proof: the base case is obvious, and if there is a side clause B_i that is not an axiom from S, then $B_i = C_j$ for some $j < i$ and we only need to attach the resolution tree for proving C_j from S instead of B_i. Notice that this also implies the *soundness* of linear resolution.

We will not present the proof of the reverse implication. It follows from the *completeness* of linear resolution, whose proof can be found in the textbook *A. Nerode, R. Shore: Logic for Applications* [3].

[4]While the construction of the resolution tree can be easily described recursively, the linear proof better corresponds to a procedural computation. It is only about finding a suitable side clause.

5.4.2 LI-Resolution

In a general linear proof, each subsequent side clause can either be an axiom from S or one of the previous central clauses. If we forbid the latter option and require that all side clauses must be from S, we get the so-called *LI (linear-input) resolution*:

Definition 5.4.4 (LI-Proof)**.** An *LI-proof* (by resolution) of a clause C from a formula S is a linear proof

$$\begin{bmatrix} C_0 \\ B_0 \end{bmatrix}, \begin{bmatrix} C_1 \\ B_1 \end{bmatrix}, \ldots, \begin{bmatrix} C_n \\ B_n \end{bmatrix}, C$$

in which each side clause B_i is an axiom from S. If an LI-proof exists, we say that C is *LI-provable* from S, and we write $S \vdash_{LI} C$. If $S \vdash_{LI} \square$, then S is *LI-refutable.*

Remark 5.4.5*.* An LI-proof directly gives a resolution tree (all leaves are axioms) in a special form that we might call a 'hairy path'. Conversely, from a resolution tree in the form of a hairy path, we immediately obtain an LI-proof: the vertices on the path are central clauses, the hairs are side clauses.

While *linear resolution*[5] is just another view of the general resolution proof, *LI-resolution* introduces a significant restriction: we lose *completeness* (not every unsatisfiable formula has an LI-refutation). On the other hand, LI-proofs are simpler to construct.[6]

5.4.3 Completeness of LI-Resolution for Horn Formulas

As we will now show, LI-resolution is *complete for Horn formulas.* As we will see in the next section, it is the basis of Prolog interpreters, which work with Horn formulas. First, let us recall the terminology related to hornness and also programs, in set representation:

- A *Horn clause* is a clause containing at most one positive literal.
- A *Horn formula* is (finite or even infinite) a set of Horn clauses.
- A *fact* is a positive unit (Horn) clause, i.e., $\{p\}$, where p' is a propositional variable.
- A *rule* is a (Horn) clause with exactly one positive and at least one negative literal.
- Rules and facts are called *program clauses.*
- A *goal* is a non-empty (Horn) clause without a positive literal.[7]

We will find the following simple observation useful:

Observation 5.4.6. *If a Horn formula S is unsatisfiable and $\square \notin S$, then it contains a fact and a goal.*

Proof. If it does not contain a fact, we can evaluate all variables to 0; if it does not contain a goal, we evaluate to 1. $\square$

[5]That is, a proof system based on finding *linear* proofs or refutations.

[6]In each step, we only have to choose from clauses in S, not from previously proven central clauses.

[7]Recall that we prove by *contradiction*, so the *goal* is the negation of what we want to prove.

Now we will state and prove the Completeness Theorem for LI-Resolution for Horn formulas. The proof also provides a guide on how to construct an LI-refutation, based on the process of unit propagation. This procedure is illustrated in the example below, which you can follow along with the reading of the proof.

Theorem 5.4.7 (Completeness of LI-Resolution for Horn Formulas)**.** *If a Horn formula T is satisfiable, and $T \cup \{G\}$ is unsatisfiable for a goal G, then $T \cup \{G\} \vdash_{LI} \square$, by an LI-refutation that starts with the goal G.*

Proof. Similarly to the Completeness Theorem for Resolution, we can assume by the Compactness Theorem that T is finite. The proof (construction of the LI-refutation) will be done by induction on the number of variables in T.

By Observation 5.4.6, T contains a fact $\{p\}$ for some propositional variable p. Since $T \cup \{G\}$ is unsatisfiable, according to Lemma 5.3.4, $(T \cup \{G\})^p = T^p \cup \{G^p\}$ is also unsatisfiable, where $G^p = G \setminus \{\neg p\}$.

If $G^p = \square$, then $G = \{\neg p\}$, $\square$ is the resolvent of G and $\{p\} \in T$, and we have a one-step LI-refutation of $T \cup \{G\}$ (this is the base case of the induction).

Otherwise, we use the induction hypothesis. Note that the formula T^p is satisfiable (by the same assignment as T, because it must contain p due to the fact $\{p\}$, so it does not contain $\neg p$) and has fewer variables than T. Thus, by the induction hypothesis, there exists an LI-derivation of $\square$ from $T^p \cup \{G^p\}$ starting with $G^p = G \setminus \{\neg p\}$.

We construct the required LI-refutation of $T \cup \{G\}$ starting with G (similarly to the proof of the Completeness Theorem for Resolution) by adding the literal $\neg p$ to all leaves that are not already in $T \cup \{G\}$ (i.e., they were created by removing $\neg p$), and to all vertices above them. This gives us $T \cup \{G\} \vdash_{LI} \neg p$, and finally we add the side clause $\{p\}$ and derive $\square$. $\square$

Example 5.4.8. Consider a (satisfiable, Horn) theory T, written in set representation as the formula $T = \{\{p, \neg r, \neg s\}, \{\neg q, r\}, \{q, \neg s\}, \{s\}\}$. Suppose we want to prove that $p \wedge q$ holds in the theory T.[8] In the resolution method, we consider the goal $G = \{\neg p, \neg q\}$ and show that $T \cup \{G\} \vdash_{LI} \square$.

Following the guide from the proof, we find a fact in the formula T, and perform unit propagation in both T and the goal G. We repeat the process until the formula is empty:

- $T = \{\{p, \neg r, \neg s\}, \{\neg q, r\}, \{q, \neg s\}, \{s\}\}$, $G = \{\neg p, \neg q\}$
- $T^s = \{\{p, \neg r\}, \{\neg q, r\}, \{q\}\}$, $G^s = \{\neg p, \neg q\}$
- $T^{sq} = \{\{p, \neg r\}, \{r\}\}$, $G^{sq} = \{\neg p\}$
- $T^{sqr} = \{\{p\}\}$, $G^{sqr} = \{\neg p\}$
- $T^{sqrp} = \emptyset$, $G^{sqrp} = \square$

Now, we construct the resolution refutation in reverse:

- $T^{sqrp}, G^{sqrp} \vdash_{LI} \square$:

$$\square$$

[8] In Prolog, we would pose the 'query': `?-p,q.`

- $T^{sqr}, G^{sqr} \vdash_{LI} \square$:

$$\begin{tikzcd}
\{\neg p\} \arrow[r, dash] & \square \\
\{p\} \arrow[ur, dash] &
\end{tikzcd}$$

- $T^{sq}, G^{sq} \vdash_{LI} \square$:

$$\begin{tikzcd}
\{\neg p\} \arrow[r, dash] & \{\neg r\} \arrow[r, dash] & \square \\
\{p, \neg r\} \arrow[ur, dash] & \{r\} \arrow[ur, dash] &
\end{tikzcd}$$

- $T^{s}, G^{s} \vdash_{LI} \square$:

$$\begin{tikzcd}
\{\neg p, \neg q\} \arrow[r, dash] & \{\neg q, \neg r\} \arrow[r, dash] & \{\neg q\} \arrow[r, dash] & \square \\
\{p, \neg r\} \arrow[ur, dash] & \{\neg q, r\} \arrow[ur, dash] & \{q\} \arrow[ur, dash] &
\end{tikzcd}$$

- $T, G \vdash_{LI} \square$

$$\begin{tikzcd}
\{\neg p, \neg q\} \arrow[r, dash] & \{\neg q, \neg r, \neg s\} \arrow[r, dash] & \{\neg q, \neg s\} \arrow[r, dash] & \{\neg s\} \arrow[r, dash] & \square \\
\{p, \neg r, \neg s\} \arrow[ur, dash] & \{\neg q, r\} \arrow[ur, dash] & \{q, \neg s\} \arrow[ur, dash] & \{s\} \arrow[ur, dash] &
\end{tikzcd}$$

5.4.4 Prolog Program

Although the real power of Prolog comes from *unification* and resolution in predicate logic, we will show how Prolog uses the resolution method with a *propositional* program. Adaptation to predicates will be straightforward later.

A *program* in Prolog is a Horn formula containing only *program clauses*, i.e., *facts* or *rules.* A *query* is a conjunction of facts, the negation of the query is the *goal.*

Example 5.4.9. As an example of a Prolog program, we will use the theory (formula) T and the query $p \wedge q$ from Example 5.4.8. For instance, the clause $\{p, \neg r, \neg s\}$, which is equivalent to $r \wedge s \to p$, is written in Prolog as: `p:-r,s.`

```
p:-r,s.
r:-q.
q:-s.
s.
```

And we pose the query to the program:

```
?-p,q.
```

Corollary 5.4.10. *Let P be a program and $Q = p_1 \wedge \cdots \wedge p_n$, and denote $G = \{\neg p_1, \ldots, \neg p_n\}$ (i.e., $G \sim \neg Q$). The following conditions are equivalent:*

- $P \models Q$,
- $P \cup \{G\}$ *is unsatisfiable,*
- $P \cup \{G\} \vdash_{LI} \square$, *and there is an LI-refutation starting with the goal G.*

Proof. The equivalence of the first two conditions is by contradiction, and the equivalence of the second and third is by the Completeness Theorem for LI-Resolution for Horn formulas. (Note that the Program is always satisfiable.) $\square$

Part II

Predicate logic

Chapter 6

Syntax and Semantics of Predicate Logic

Courses on logic generally start with propositional logic, which is more suitable for initial exposition due to its simplicity. However, the full power of logic in computer science only manifests itself with the use of predicate logic. Let us begin with an informal introduction that illustrates the basic aspects of predicate logic. We will return to a formal exposition in the following sections.

6.1 Introduction

Recall that in propositional logic, we described the world using *propositions* composed of *atomic propositions*—answers to yes/no questions about the world. In (first-order[1]) predicate logic the basic building blocks are *variables* representing *individuals*—indivisible objects from some set: e.g., natural numbers, vertices of a graph, or states of a microprocessor.

These individuals can have certain properties and relationships, which we call *predicates*, e.g., ‘$\mathrm{Leaf}(x)$’ or ‘$\mathrm{Edge}(x, y)$’ when talking about a graph, or ‘$x \leq y$’ in natural numbers. In addition, individuals can be passed into functions, e.g., ‘$\mathrm{lowest\_common\_ancestor}(x, y)$’ in a rooted tree, ‘$\mathrm{succ}(x)$’ or ‘$x + y$’ in natural numbers, and they can be *constants* with special meaning, e.g., ‘root’ in a rooted tree, ‘0’ in natural numbers.

Atomic formulas describe a predicate (including *equality* predicate $=$) about variables or *terms* (‘expressions’ composed[2] of functions or constants). More complex statements (*formulas*) are built from atomic formulas using logical connectives and two *quantifiers*:

- $\forall x$ “for all individuals (represented by variable x),” and
- $\exists x$ “there exists an individual (represented by variable x)”.

Let us look at an example: the statement *“Everyone who has a child is a parent.”* could be formalized by the following formula:

$$(\forall x)((\exists y)\mathrm{child\_of}(y, x) \rightarrow \mathrm{is\_parent}(x))$$

[1]In second-order logic, we also have variables representing sets of individuals or even sets of n-tuples, i.e., relations on the set of individuals.

[2]Similarly to how we create arithmetic expressions.

where child_of(y,x) is a binary predicate expressing that the individual represented by variable y is a child of the individual represented by variable x, and is_parent(x) is a unary predicate (i.e., a 'property') expressing that the individual represented by x is a parent.

What about the validity of this formula? That depends on the specific *model* of the world/system we are interested in. A model is a (non-empty) set of objects together with a unary relation (i.e., a subset) *interpreting* the *unary relation symbol* is_parent and a binary relation interpreting the *binary relation symbol* child_of. These relations can generally be arbitrary, and it is easy to construct a model where the formula is not valid.[3] However, if we model, for example, all people in the world, and the relations have their natural meaning, then the formula will be valid.[4]

Let us look at another example, this time with function symbols and a constant symbol: "If $x_1 \leq y_1$ and $x_2 \leq y_2$, then $(y_1 \cdot y_2) - (x_1 \cdot x_2)$ is nonnegative." The resulting formula could look like this:

$$\varphi = (x_1 \leq y_1) \wedge (x_2 \leq y_2) \rightarrow ((y_1 \cdot y_2) + (-(x_1 \cdot x_2)) \geq 0)$$

We see two binary relation symbols $(\leq, \geq)$, a binary function symbol $+$, a unary function symbol $-$, and a constant symbol 0.

An example of a model in which the formula is valid is the set of natural numbers $\mathbb{N}$ with binary relations $\leq^{\mathbb{N}}, \geq^{\mathbb{N}}$, binary functions $+^{\mathbb{N}}, \cdot^{\mathbb{N}}$, unary function $-^{\mathbb{N}}$, and the constant $0^{\mathbb{N}} = 0$. However, if we similarly take the set of integers, the formula will no longer be valid.

Remark 6.1.1. We could understand the symbol $-$ as a binary operation, but it is usually introduced as unary. For the constant symbol 0, we use (as is customary) the same symbol as for the natural number 0. But note that in our model, this constant symbol could be interpreted as a different number, or our model might not consist of numbers at all!

There are no quantifiers in the formula (such formulas are called *open*), the variables x_1, x_2, y_1, y_2 are *free variables* of this formula (they are not *bound* by any quantifier), we write $\varphi(x_1, x_2, y_1, y_2)$. We understand the semantics of this formula in the same way as the formula

$$(\forall x_1)(\forall x_2)(\forall y_1)(\forall y_2)\varphi(x_1, x_2, y_1, y_2)$$

The expression $(y_1 \cdot y_2) + (-(x_1 \cdot x_2))$ is an example of a *term*, and the expressions $(x_1 \leq y_1)$, $(x_2 \leq y_2)$ and $((y_1 \cdot y_2) + (-(x_1 \cdot x_2)) \geq 0)$ are *atomic (sub)formulas*. What is the difference? Given a specific model and a specific *variable assignment* by individuals (elements) of this model, atomic formulas can be assigned a truth value. Therefore, they can be combined with logical connectives into more complex 'logical expressions', i.e., formulas. On the other hand, the 'result' of a term (under a given variable assignment) is some specific individual from the model.

We also note that in the formula φ, we used infix notation for the function symbols $+, \cdot$ and for the relations $\leq, \geq$, and similar conventions for parentheses as in propositional logic. Otherwise, we would write the formula φ as follows:

$$((\leq (x_1, y_1) \wedge \leq (x_2, y_2)) \rightarrow \leq (+(\cdot(y_1, y_2), -(\cdot(x_1, x_2))), 0))$$

[3]For example, take a single-element set $A = \{a\}$, and the relations child_of$^A = \{(a, a)\}$, parent$^A = \emptyset$; here the only object is its own child, but it is not a parent.

[4]When formalizing, we must be very careful not to add additional assumptions that may not hold in the modeled system. Here, for example, there is an implicit assumption that if someone has a child, they must be the child's parent.

Exercise 6.1. Find a suitable definition for the notion of a *tree of a formula* (generalizing the *tree of a proposition* from propositional logic). Draw the tree for the following formula from the above example: $(\forall x_1)(\forall x_2)(\forall y_1)(\forall y_2)\varphi(x_1, x_2, y_1, y_2)$.

Now, let us start by formalizing this notion of a "*model*", a so-called *structure*. The rest of the chapter follows the outline of the exposition on propositional logic: we will introduce the syntax, then the semantics, and finally the more advanced properties of formulas, theories, and structures. At the end, we will show a simple but very useful application of predicate logic, called *definability* of subsets and relations, which is the basis of *relational databases* (e.g., SQL), and once again look at the relationship between propositional and predicate logic.

6.2 Structures

First, we specify what *type* the given structure will be, i.e., what relations, functions (of which arities), and constants it will have, and what symbols we will use for them. This formal specification is sometimes called a *type*, but we will call it a *signature*.[5] Recall that we can consider *constants* as functions of arity 0 (i.e., functions without inputs).

Definition 6.2.1. A *signature* is a pair $\langle \mathcal{R}, \mathcal{F} \rangle$, where $\mathcal{R}, \mathcal{F}$ are disjoint sets of symbols (*relation* and *function*, the latter include *constant* symbols) with given arities (i.e., given by a function $\mathrm{ar}\colon \mathcal{R} \cup \mathcal{F} \to \mathbb{N}$) and not containing the symbol '=' (which is reserved for *equality*).

However, we will often write a signature just by listing the symbols, where their arity and whether they are relation or function symbols will be clear from the context. Here are a few examples of signatures:

- $\langle E \rangle$ the signature of *graphs*: E is a binary relation symbol (structures are directed graphs),
- $\langle \leq \rangle$ the signature of *partial orders*: the same as the signature of graphs, just a different symbol,[6]
- $\langle +, -, 0 \rangle$ the signature of *groups*: $+$ is a binary function, $-$ a unary function, 0 a constant symbol
- $\langle +, -, 0, \cdot, 1 \rangle$ the signature of *fields*: $\cdot$ is a binary function, 1 a constant symbol
- $\langle +, -, 0, \cdot, 1, \leq \rangle$ the signature of *ordered fields*: $\leq$ is a binary relation symbol,
- $\langle -, \wedge, \vee, \perp, \top \rangle$ the signature of *Boolean algebras*: $\wedge, \vee$ are binary function symbols, $\perp, \top$ are constant symbols,
- $\langle S, +, \cdot, 0, \leq \rangle$ the signature of *arithmetic*: S is a unary function symbol ('successor').

[5]You can think of a signature as similar to the definition of a *class* in OOP; structures then correspond to *objects* of this class (in the 'programming language' of set theory).

[6]Not every structure in this signature is a partial order; for that, it needs to satisfy the corresponding *axioms*.

In addition to common symbols for relations, functions, and constants (familiar e.g. from arithmetic), we typically use $P, Q, R, \dots$ for relation symbols, $f, g, h, \dots$ for function symbols, and $c, d, a, b, \dots$ for constant symbols.

A *structure* of a given signature is obtained by taking some non-empty *domain* and choosing *interpretations* (also called *realizations*) of all relation and function symbols (and constants) on that domain, i.e., specific relations or functions of the appropriate arities. (In the case of a constant symbol, its interpretation is a chosen element from the domain.)[7]

Example 6.2.2. The formal definition of a *structure* is given below; first, let us show a few examples:

- A structure in the empty signature $\langle\ \rangle$ is any non-empty set.[8] (It does not have to be finite, not even countable!)
- A structure in the signature of graphs is $\mathcal{G} = \langle V, E\rangle$, where $V \neq \emptyset$ and $E \subseteq V^2$, called a *directed graph.*
  - If E is irreflexive and symmetric, it is a *simple* graph (i.e., undirected, without loops).
  - If E is reflexive, transitive, and antisymmetric, it is a *partial order.*
  - If E is reflexive, transitive, and symmetric, it is an *equivalence relation.*
- Structures in the signature of partial orders are the same as in the signature of graphs, differing only by the symbol used. (Thus, not every structure in the signature of partial orders is a partial order!)
- Structures in the signature of groups are, for example, the following *groups*:
  - $\underline{\mathbb{Z}_n} = \langle \mathbb{Z}_n, +, -, 0\rangle$, the *additive group of integers modulo n* (operations are modulo n).[9]
  - $\mathcal{S}_n = \langle \mathrm{Sym}_n, \circ, {}^{-1}, \mathrm{id}\rangle$ is the *symmetric group* (the group of all permutations) on n elements.
  - $\underline{\mathbb{Q}}^* = \langle \mathbb{Q} \setminus \{0\}, \cdot, {}^{-1}, 1\rangle$ is the *multiplicative group of (non-zero) rational numbers.* Note that the interpretation of the *symbol* 0 is the *number* 1.

  All these structures *satisfy the axioms of group theory*, but we can easily find other structures that do not satisfy these axioms and are therefore not groups. For example, if we change the interpretation of the symbol $+$ in the structure $\mathbb{Z}_n$ to the function $\cdot$ (modulo n).
- Structures $\underline{\mathbb{Q}} = \langle \mathbb{Q}, +, -, 0, \cdot, 1, \leq\rangle$ and $\underline{\mathbb{Z}} = \langle \mathbb{Z}, +, -, 0, \cdot, 1, \leq\rangle$, with the standard operations and the standard order relation, are in the signature of ordered fields (but only the first one is an ordered field).

[7]It does not matter what specific symbols we use in the signature; we can interpret them arbitrarily. For example, having the symbol $+$ does not mean that its interpretation must have anything to do with addition (other than being a binary function).

[8]As we will see in the definition below, formally, it is the triple $\langle A, \emptyset, \emptyset\rangle$, but we will ignore this distinction.

[9]Here, $\underline{\mathbb{Z}_n}$ denotes the structure, while $\mathbb{Z}_n = \{0, 1, \dots, n-1\}$ denotes only its domain. Often, this distinction is not made, and the symbol $\mathbb{Z}_n$ is used for both the whole structure and its domain. Similarly, $+, -, 0$ are both symbols and their interpretations. This is a common abuse of notation; it is crucial to always be aware of the meaning of the symbol in the given context.

- $\underline{\mathcal{P}(X)} = \langle \mathcal{P}(X), \bar{}, \cap, \cup, \emptyset, X \rangle$, the so-called *power set algebra* over a set X, is a structure in the signature of Boolean algebras. (It is a *Boolean algebra*, as long as $X \neq \emptyset$.)
- $\underline{\mathbb{N}} = \langle \mathbb{N}, S, +, \cdot, 0, \leq \rangle$, where $S(x) = x + 1$, and other symbols are interpreted in the standard way, is the *standard model of arithmetic.*

Definition 6.2.3 (Structure)**.** A *structure in the signature* $\langle \mathcal{R}, \mathcal{F} \rangle$ is a triple $\mathcal{A} = \langle A, \mathcal{R}^{\mathcal{A}}, \mathcal{F}^{\mathcal{A}} \rangle$, where

- A is a non-empty set, called the *domain* (also *universe*),
- $\mathcal{R}^{\mathcal{A}} = \{R^{\mathcal{A}} \mid R \in \mathcal{R}\}$ where $R^{\mathcal{A}} \subseteq A^{\mathrm{ar}(R)}$ is the *interpretation* of the relation symbol R,
- $\mathcal{F}^{\mathcal{A}} = \{f^{\mathcal{A}} \mid f \in \mathcal{F}\}$ where $f^{\mathcal{A}} \colon A^{\mathrm{ar}(f)} \to A$ is the *interpretation* of the function symbol f (in particular, for a constant symbol $c \in \mathcal{F}$, we have $c^{\mathcal{A}} \in A$).

Exercise 6.2. Consider the signature of *n constants* $\langle c_1, c_2, \ldots, c_n \rangle$. What do structures in this signature look like? Describe, for example, all structures with at most five elements in the signature of three constants. (The interpretations of constants do not have to be different!) And what about the case of the signature of *countably many constants* $\langle c_1, c_2, \ldots \rangle = \langle c_i \mid i \in \mathbb{N} \rangle$?

6.3 Syntax

In this section, we introduce the syntax of predicate (first-order) logic. Compare what the syntax has in common with, and how it differs from, the syntax of propositional logic.

6.3.1 Language

When specifying a language, we first determine what type of structures we want to describe, i.e., we specify the *signature.* Additionally, we include the information on whether the language is *with equality* or not, i.e., whether we can use the symbol '=' in formulas to express the equality (identity) of elements in the domain of structures.[10] The language includes the following:

- countably many *variables* $x_0, x_1, x_2, \ldots$ (but we also write $x, y, z, \ldots$; the set of all variables is denoted Var),
- *relation*, *function*, and *constant symbols* from the signature, and the symbol $=$ if the language is with equality,
- *universal* and *existential quantifiers* $(\forall x), (\exists x)$ for each variable $x \in \mathrm{Var}$,[11]
- symbols for logical connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ and parentheses $(,)$.

[10]In most applications, we will use languages with equality. However, in some special areas, it is useful to not have equality in the language. For example, if we deal with very fast models of computation: finding out which variables are equal requires finding the transitive closure of equality predicates given by formulas, which is already a relatively computationally demanding problem.

[11]A quantifier is understood as a single symbol, so $(\forall x)$ *does not include* the variable x. Sometimes the symbols $\forall_x, \exists_x$ are used instead.

Similarly to the symbol $\square$ representing any binary logical connective, we will sometimes write (Qx) for the quantifier $(\forall x)$ or $(\exists x)$.

Symbols from the signature, and $=$, are called *non-logical*, and the other symbols are *logical.* The language must contain at least one relation symbol (either equality, or in the signature).[12]

Thus, we specify the language by giving a signature and the information 'with equality' (or 'without equality'). For example:

- The language $L = \langle\rangle$ with equality is the language of *pure equality*,
- the language $L = \langle c_0, c_1, c_2, \dots\rangle$ with equality is the language of *countably many constants*,
- the language of *order* is $\langle\leq\rangle$ with equality,
- the language of *graph theory* is $\langle E\rangle$ with equality,
- the languages of *group theory, field theory, ordered field theory, Boolean algebras, arithmetic* are languages with equality corresponding to the signatures from Example 6.2.2.

6.3.2 Terms

Terms are syntactic 'expressions' composed of variables, constant symbols, and function symbols.

Definition 6.3.1 (Terms)**.** *Terms* of the language L are finite strings defined inductively:

- each variable and each constant symbol from L is a term,
- if f is a function symbol from L of arity n and $t_1, \dots, t_n$ are terms, then the string $f(t_1, t_2, \dots, t_n)$ is also a term.

The set of all *terms* of the language L is denoted Term_L.

When writing terms containing a binary function symbol, we can use *infix* notation, e.g., $(t_1 + t_2)$ means $+(t_1, t_2)$. Parentheses are sometimes omitted if the structure of the term ('operator precedence') is clear.

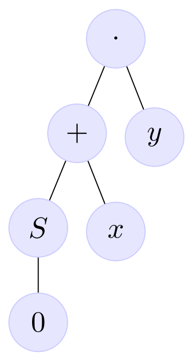
A *subterm* is a substring of a term that is itself a term (it is either the whole term or it appeared as some t_i in the construction of the term).

If a term does not contain a variable, we call it *ground* (also *constant*), for example, $((S(0) + S(0)) \cdot S(S(0)))$ is a ground term in the language of arithmetic.[13]

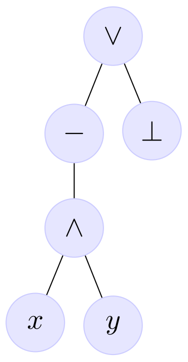
The *tree of the term t*, denoted $\mathrm{Tree}(t)$, is defined similarly to a tree of a proposition: leaves are labeled by variables or constant symbols, inner nodes by function symbols whose arity matches the number of children.

[12]Otherwise, we would not be able to build any 'statements' (*formulas*) in the language, see below.

[13]Note that terms are purely syntactic; we can only use symbols from the language, not elements of the structure, so $(1 + 1) \cdot 2$ is *not* a term in the language of arithmetic! (However, we could *define* new constant symbols $1, 2$ as abbreviations for $S(0)$ and $S(S(0))$ and *extend* our language, see Section 6.7.1.)



(a) $(S(0) + x) \cdot y$ in the language of arithmetic



(b) $-(x \wedge y) \vee \bot$ in the language of Boolean algebras

Figure 6.1: Term tree

Example 6.3.2. Let us draw the trees of the terms (a) $(S(0)+x)\cdot y$ in the language of arithmetic, (b) $-(x \wedge y) \vee \bot$ in the language of Boolean algebras. Here, $\wedge, \vee$ are not logical connectives from the language but non-logical symbols from the signature of Boolean algebras (although we use the same symbols)! Terms in this language can be understood as propositional formulas (with constants for falsity and truth), see Section 6.9. Figure 6.1 shows the trees of these terms.

It is not hard to guess what the *semantics* of terms will be. Given a specific structure, a term corresponds to a function on its domain: the input is an assignment of the variables to elements of the domain, the constant and function symbols are replaced by their interpretations, and the output is the value (element of the domain) at the root. More formally, this will be covered in Section 6.4.

6.3.3 Formulas

Terms cannot be assigned a truth value in any sense; for that, we need a *predicate* (a relation symbol or equality) that talks about the 'relationship' between terms: in a specific structure with a specific variable assignment to elements of the domain, this relationship is either satisfied or not.

The simplest *formulas* are *atomic formulas*. We then build all formulas from them using logical connectives and quantifiers.

Definition 6.3.3 (Atomic Formula)**.** An *atomic formula* of the language L is a string $R(t_1, \ldots, t_n)$, where R is an n-ary relation symbol from L (including $=$ if it is a language with equality) and $t_i \in \mathrm{Term}_L$.

For binary relation symbols, we often use infix notation, e.g., the atomic formula $\leq (x, y)$ is written as $x \leq y$, and (if the language has equality) instead of $= (t_1, t_2)$, we will write $t_1 = t_2$.

Example 6.3.4. Here are some examples of atomic formulas:

- $R(f(f(x)), c, f(d))$ where R is a ternary relation symbol, f a unary function symbol, c, d are constant symbols,
- $(x \cdot x) + (y \cdot y) \leq (x + y) \cdot (x + y)$ in the language of ordered fields,

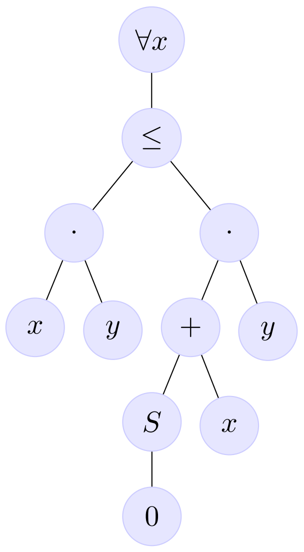


Figure 6.2: Tree of the formula $(\forall x)(x \cdot y \leq (S(0) + x) \cdot y)$

- $x \cdot y \leq (S(0) + x) \cdot y$ in the language of arithmetic,
- $-(x \wedge y) \vee \bot = \bot$ in the language of Boolean algebras

Definition 6.3.5 (Formula)**.** *Formulas* of the language L are finite strings defined inductively:

- each atomic formula of the language L is a formula,
- if φ is a formula, then $(\neg\varphi)$ is also a formula,
- if φ, ψ are formulas, then $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$, and $(\varphi \leftrightarrow \psi)$ are also formulas,
- if φ is a formula and x a variable, then $((\forall x)\varphi)$ and $((\exists x)\varphi)$ are also formulas.

A *subformula* is a substring that is itself a formula. The *tree of a formula*, denoted $\mathrm{Tree}(\varphi)$, is defined as follows: the tree of an atomic formula $\varphi = R(t_1, \ldots, t_n)$ has the relation symbol R at the root, to which we append the trees $\mathrm{Tree}(t_i)$. If φ is not atomic, the tree is constructed similarly to a propositional formula tree.[14] When writing formulas, we use similar conventions as in propositional logic, with quantifiers having the same precedence as $\neg$ (higher than other logical connectives). Therefore, instead of $((\forall x)\varphi)$ we can write $(\forall x)\varphi$.[15]

Example 6.3.6. An example of a formula in the language of arithmetic is $(\forall x)(x \cdot y \leq (S(0) + x) \cdot y)$. Its tree is shown in Figure 6.2.

Free and Bound Variables

The meaning of a formula[16] may or may not depend on the variables that appear in it: compare $x \leq 0$ and $(\exists x)(x \leq 0)$ (and how about $x \leq 0 \vee (\exists x)(x \leq 0)$?). We now clarify this concept and introduce the necessary terminology.

[14]Quantifiers, like negations, have a single child.

[15]Sometimes parentheses are not written in quantifiers, e.g., $\forall x\varphi$, but we will always include them, for better readability.

[16]More precisely, its *truth value*, formally defined below in Section 6.4.3.

An *occurrence* of a variable x in a formula φ means a leaf in Tree(φ) labeled x. [17] An occurrence is *bound* if it is a part of some subformula (subtree) starting with (Qx). If an occurrence is not bound, it is *free*. A variable is *free* in φ if it has a free occurrence in φ, and *bound* in φ if it has a bound occurrence in φ. The notation $\varphi(x_1, \ldots, x_n)$ means that $x_1, \ldots, x_n$ are all the free variables in the formula φ.

Example 6.3.7. A variable can be both free and bound, e.g., in the formula $\varphi = (\forall x)(\exists y)(x \leq y) \vee x \leq z$, the first occurrence of x is bound and the second occurrence is free. (Draw the formula tree!) The variable y is bound (its only occurrence is bound) and z is free. Thus, we can write $\varphi(x, z)$.

Remark 6.3.8. As we will see below, the meaning (*truth value*) of a formula depends only on the assignment of free variables. Variables in quantifiers, along with their corresponding bound occurrences, can be renamed (we have to be careful though, see below).

Open and Closed Formulas

We often talk about the following two important properties of formulas:

Definition 6.3.9 (Open and Closed Formula)**.** A formula is *open* if it contains no quantifier, and *closed* (or a *sentence*) if it has no free variable.

Example 6.3.10. Here are a few examples:

- the formula $x + y \leq 0$ is open,
- the formula $(\forall x)(\forall y)(x + y \leq 0)$ is closed (i.e., it is a sentence),
- the formula $(\forall x)(x + y \leq 0)$ is neither open nor closed,
- the formula $(0 + 1 = 1) \wedge (1 + 1 = 0)$ is both open and closed.

Every atomic formula is open, and open formulas are just combinations of atomic formulas using logical connectives. A formula can be both open and closed if all its terms are constant. A formula is closed if and only if it has no free variable.[18]

Remark 6.3.11. As we will see later, the *truth value* of a formula depends only on the assignment of its free variables. In particular, a sentence has a truth value of 0 or 1 in a given structure (independently of variable assignments). This is why sentences play an important role in logic.

6.3.4 Instances and Variants

As we have seen, one variable can appear in different 'roles' in a formula. This is a very similar principle to programming, where one identifier can mean different variables in a program (either local or global). The term *instance* can be understood as 'substituting' (a term) into a (global) variable (or better, 'replacing' a variable with some expression that computes it), and the term *variant* as 'renaming' a (local) variable. Consider, for example, the formula $\varphi(x)$:

$$P(x) \wedge (\forall x)(Q(x) \wedge (\exists x)R(x))$$

[17] Thus, the variable x does *not occur* in the symbol for the quantifier (Qx).

[18] It is not true that a formula is open if it has no bound variable, see the formula $(\forall x)0 = 1$.

The first occurrence of the variable x is free, the second is bound by the quantifier $(\forall x)$, and the third is bound by $(\exists x)$. If we 'substitute' the term $t = 1+1$ for the variable x, we get an *instance* of the formula φ, denoted $\varphi(x/t)$:

$$P(1+1) \wedge (\forall x)(Q(x) \wedge (\exists x)R(x))$$

We can also rename the quantifiers in the formula, thus obtaining a *variant* of the formula φ, e.g.:

$$P(x) \wedge (\forall y)(Q(y) \wedge (\exists z)R(z))$$

How do we know when and how we can do this to preserve the meaning, i.e., so that the instance is a *consequence* of φ, and the variant is *equivalent* to φ? This is what we now want to formalize.

Instances

If we *substitute* a term t for a free variable x in a formula φ, we require that the resulting formula 'says' about t 'the same' as φ says about x.

Example 6.3.12. For example, the formula $\varphi(x) = (\exists y)(x + y = 1)$ says about x that 'there exists $1 - x$'. The term $t = 1$ can be substituted because $\varphi(x/t) = (\exists y)(1 + y = 1)$ says 'there exists 1-1'. But the term $t = y$ cannot be substituted because $(\exists y)(y + y = 1)$ says '1 is divisible by 2'. The problem is that the term $t = y$ contains the variable y, which will now be bound by the quantifier $(\exists y)$. We must avoid such situations.

Definition 6.3.13 (Substitutability and Instance)**.** A term t is *substitutable* for a variable x in a formula φ if, after simultaneously replacing all free occurrences of x in φ with t, no new bound occurrence of a variable from t arises in φ. In that case, the resulting formula is called an *instance* of φ obtained by substituting t for x, denoted $\varphi(x/t)$.

Remark 6.3.14. Note that a term t is *not* substitutable for x in φ if and only if x has a free occurrence in some subformula φ of the form $(Qy)\psi$ and the variable y appears in t. In particular, ground terms are always substitutable.

Variants

If we need to substitute a term t into a formula φ, we can always do so if we first rename all quantified variables to entirely new ones (i.e., ones that do not appear in φ or t), and then substitute t into the resulting *variant* of the formula φ.

Definition 6.3.15 (Variant)**.** If a formula φ has a subformula of the form $(Qx)\psi$ and y is a variable such that

- y is substitutable for x in ψ, and
- y has no free occurrence in ψ,

then replacing the subformula $(Qx)\psi$ with $(Qy)\psi(x/y)$ results in a *variant* of the formula φ in the subformula $(Qx)\psi$. The result of successive variations in multiple subformulas is also called a *variant*.

Note that the requirement for the variable y in the definition of a variant is always satisfied if y does not appear in the formula φ.

Example 6.3.16. Consider the formula $\varphi = (\exists x)(\forall y)(x \leq y)$. Then:

- $(\exists y)(\forall y)(y \leq y)$ is not a variant of φ because y is not substitutable for x in $\psi = (\forall y)(x \leq y)$,
- $(\exists x)(\forall x)(x \leq x)$ is not a variant of φ because x has a free occurrence in the subformula $\psi = (x \leq y)$,
- $(\exists u)(\forall v)(u \leq v)$ is a variant of φ.

This concludes the exposition on syntax; next is semantics.

6.4 Semantics

Before we delve into a more formal exposition, let us briefly summarize the semantics as we have already hinted in previous sections:

- Models are structures of the given signature,
- A formula is valid in a structure if it holds under every assignment of free variables to elements from the domain,
- The values of terms are evaluated according to their trees, where symbols are replaced by their interpretations (relations, functions, and constants from the domain),
- From the values of terms, we obtain the truth values of atomic formulas: is the resulting n-tuple in the relation?
- The values of complex formulas are also evaluated according to their tree, where $(\forall x)$ acts as 'conjunction over all elements' and $(\exists y)$ acts as 'disjunction over all elements' of the structure's domain.

Now more formally:

6.4.1 Models of the Language

Definition 6.4.1 (Model of the Language)**.** A *model of the language* L, or an L*-structure*, is any structure in the signature of the language L. The *class of all models* of the language is denoted M_L.

Remark 6.4.2. The definition does not care whether the language is with or without equality. And why can't we talk about the *set* of all models M_L, why do we have to say *class*? Because the domain of a structure can be any non-empty set, and the 'set of all sets' does not exist; it is a classic example of a so-called proper class. A class is the *'collection'* of all sets satisfying a given property (describable in the *language of set theory*).

Example 6.4.3. Among the models of the language of order $L = \langle \leq \rangle$ are the following structures: $\langle \mathbb{N}, \leq \rangle$, $\langle \mathbb{Q}, > \rangle$, any directed graph $G = \langle V, E \rangle$, $\langle \mathcal{P}(X), \subseteq \rangle$. But also, for instance, $\langle \mathbb{C}, R^{\mathbb{C}} \rangle$ where $(z_1, z_2) \in R^{\mathbb{C}}$ if and only if $|z_1| = |z_2|$ or $\langle \{0, 1\}, \emptyset \rangle$, which are *not* partial orders.

6.4.2 Value of a term

Consider a term t in the language $L = \langle \mathcal{R}, \mathcal{F} \rangle$ (with or without equality), and an L-structure $\mathcal{A} = \langle A, \mathcal{R}^{\mathcal{A}}, \mathcal{F}^{\mathcal{A}} \rangle$. A *variable assignment* in the set A is any function $e : \mathrm{Var} \to A$.

Definition 6.4.4 (Value of a term)**.** The *value of the term t in the structure $\mathcal{A}$ under the assignment e*, denoted $t^{\mathcal{A}}[e]$, is defined inductively:

- $x^{\mathcal{A}}[e] = e(x)$ for a variable $x \in \mathrm{Var}$,
- $c^{\mathcal{A}}[e] = c^{\mathcal{A}}$ for a constant symbol $c \in \mathcal{F}$, and
- if $t = f(t_1, \ldots, t_n)$ is a compound term where $f \in \mathcal{F}$, then:

$$t^{\mathcal{A}}[e] = f^{\mathcal{A}}(t_1^{\mathcal{A}}[e], \ldots, t_n^{\mathcal{A}}[e])$$

Remark 6.4.5. Note that the value of a term depends only on the assignment of the variables appearing in it. In particular, if t is a ground term, its value does not depend on the assignment. In general, each term t represents a *term function* $f_t^{\mathcal{A}} \colon A^k \to A$, where k is the number of variables in t, and ground terms correspond to constant functions.

Example 6.4.6. Here are two examples:

- The value of the term $-(x \vee \bot) \wedge y$ in the Boolean algebra $\underline{\mathcal{P}(\{0, 1, 2\})}$ under the assignment e where $e(x) = \{0, 1\}$ and $e(y) = \{1, 2\}$ is $\{2\}$.
- The value of the term $x + 1$ in the structure $\mathcal{N} = \langle \mathbb{N}, \cdot, 3 \rangle$ of the language $L = \langle +, 1 \rangle$ under the assignment e where $e(x) = 2$ is $(x + 1)^{\mathcal{N}}[e] = 6$.

6.4.3 Truth Value of a Formula

We are now ready to define the *truth value*. Locally, we introduce the notation TVal for it.

Definition 6.4.7 (Truth Value)**.** Given a formula φ in the language L, a structure $\mathcal{A} \in \mathrm{M}_L$, and a variable assignment $e : \mathrm{Var} \to A$. The *truth value of φ in $\mathcal{A}$ under the assignment e*, $\mathrm{TVal}^{\mathcal{A}}(\varphi)[e]$, is defined inductively according to the structure of the formula:

For an atomic formula $\varphi = R(t_1, \ldots, t_n)$, we have

$$\mathrm{TVal}^{\mathcal{A}}(\varphi)[e] = \begin{cases} 1 & \text{if } (t_1^{\mathcal{A}}[e], \ldots, t_n^{\mathcal{A}}[e]) \in R^{\mathcal{A}}, \\ 0 & \text{otherwise.} \end{cases}$$

In particular, if φ is of the form $t_1 = t_2$, then $\mathrm{TVal}^{\mathcal{A}}(\varphi)[e] = 1$ if and only if $(t_1^{\mathcal{A}}[e], t_2^{\mathcal{A}}[e]) \in =^{\mathcal{A}}$, where $=^{\mathcal{A}}$ is the identity on A, i.e., if and only if $t_1^{\mathcal{A}}[e] = t_2^{\mathcal{A}}[e]$ (both sides of the equality are the same element $a \in A$).

The truth value of a negation is defined as follows:

$$\mathrm{TVal}^{\mathcal{A}}(\neg\varphi)[e] = f_{\neg}(\mathrm{TVal}^{\mathcal{A}}(\varphi)[e]) = 1 - \mathrm{TVal}^{\mathcal{A}}(\varphi)[e]$$

Similarly, for binary logical connectives, if φ, ψ are formulas and $\square \in \{\wedge, \vee, \to, \leftrightarrow\}$, then:

$$\mathrm{TVal}^{\mathcal{A}}(\varphi \,\square\, \psi)[e] = f_{\square}(\mathrm{TVal}^{\mathcal{A}}(\varphi)[e], \mathrm{TVal}^{\mathcal{A}}(\psi)[e])$$

It remains to define the truth value for quantifiers, i.e., formulas of the form $(Qx)\varphi$. We will need the following notation: If in the assignment $e : \mathrm{Var} \to A$ we change the value for the variable x to a, the resulting assignment is written as $e(x/a)$. Thus, $e(x/a)(x) = a$. The truth value for $(Qx)\varphi$ is defined as follows:

$$\mathrm{TVal}^{\mathcal{A}}((\forall x)\varphi)[e] = \min_{a\in A}(\mathrm{TVal}^{\mathcal{A}}(\varphi)[e(x/a)])$$
$$\mathrm{TVal}^{\mathcal{A}}((\exists x)\varphi)[e] = \max_{a\in A}(\mathrm{TVal}^{\mathcal{A}}(\varphi)[e(x/a)])$$

Thus, under the assignment e, we set the value of the variable x successively to all elements $a \in A$ and require that the truth value is 1 always (in the case of $\forall$) or at least once (in the case of $\exists$).[19]

Remark 6.4.8. The truth value depends only on the assignment of free variables. In particular, if φ is a sentence, then its truth value does not depend on the assignment.

Example 6.4.9. Consider the ordered field $\underline{\mathbb{Q}}$. Then:

- $\mathrm{TVal}^{\underline{\mathbb{Q}}}(x \leq 1 \wedge \neg(x \leq 0))[e] = 1$ if and only if $e(x) \in (0, 1]$,
- $\mathrm{TVal}^{\underline{\mathbb{Q}}}((\forall x)(x \cdot y = y))[e] = 1$ if and only if $e(y) = 0$,
- $\mathrm{TVal}^{\underline{\mathbb{Q}}}((\exists x)(x \leq 0 \wedge \neg x = 0))[e] = 1$ for any assignment e (it is a sentence), but
- $\mathrm{TVal}^{\mathcal{A}}((\exists x)(x \leq 0 \wedge \neg x = 0))[e] = 0$ (for any e), if $\mathcal{A} = \langle \mathbb{N}, +, -, 0, \cdot, 1, \leq\rangle$ with the standard operations and order.

6.4.4 Validity

Based on the truth value, we can now define the key notion of semantics, *validity.*

Definition 6.4.10 (Validity in a Structure)**.** Given a formula φ and a structure $\mathcal{A}$ (in the same language).

- If e is an assignment and $\mathrm{TVal}^{\mathcal{A}}(\varphi)[e] = 1$, we say that φ *is valid in* $\mathcal{A}$ *under the assignment* e, and write $\mathcal{A} \models \varphi[e]$. (Otherwise, we say that φ *is not valid in* $\mathcal{A}$ *under the assignment* e, and write $\mathcal{A} \not\models \varphi[e]$.)
- If φ is valid in $\mathcal{A}$ under every assignment $e : \mathrm{Var} \to A$, we say that φ *is valid (true) in* $\mathcal{A}$, and write $\mathcal{A} \models \varphi$.
- If $\mathcal{A} \models \neg\varphi$, i.e., φ is not valid in $\mathcal{A}$ under any assignment (for every e we have $\mathcal{A} \not\models \varphi[e]$), then φ *is contradictory in* $\mathcal{A}$.[20]

Let us summarize some simple properties, first concerning validity under an assignment. Let $\mathcal{A}$ be a structure, φ, ψ formulas, and e a variable assignment.

- $\mathcal{A} \models \neg\varphi[e]$ if and only if $\mathcal{A} \not\models \varphi[e]$,
- $\mathcal{A} \models (\varphi \wedge \psi)[e]$ if and only if $\mathcal{A} \models \varphi[e]$ and $\mathcal{A} \models \psi[e]$,

[19]Recall that $f_\wedge(x, y) = \min(x, y)$ and $f_\vee(x, y) = \max(x, y)$. Thus, quantifiers play the role of 'conjunction' ($\forall$) or 'disjunction' ($\exists$) over all elements of the structure.

[20]Note that *contradictory* is not the same as *not valid*! This only holds for sentences.

- $\mathcal{A} \models (\varphi \vee \psi)[e]$ if and only if $\mathcal{A} \models \varphi[e]$ or $\mathcal{A} \models \psi[e]$,
- $\mathcal{A} \models (\varphi \to \psi)[e]$ if and only if: if $\mathcal{A} \models \varphi[e]$ then $\mathcal{A} \models \psi[e]$,
- $\mathcal{A} \models (\varphi \leftrightarrow \psi)[e]$ if and only if: $\mathcal{A} \models \varphi[e]$ if and only if $\mathcal{A} \models \psi[e]$,
- $\mathcal{A} \models (\forall x)\varphi[e]$ if and only if $\mathcal{A} \models \varphi[e(x/a)]$ for all $a \in A$,
- $\mathcal{A} \models (\exists x)\varphi[e]$ if and only if $\mathcal{A} \models \varphi[e(x/a)]$ for some $a \in A$.
- If a term t is substitutable for the variable x in the formula φ, then

$$\mathcal{A} \models \varphi(x/t)[e] \text{ if and only if } \mathcal{A} \models \varphi[e(x/a)] \text{ for } a = t^{\mathcal{A}}[e].$$

- If ψ is a variant of φ, then $\mathcal{A} \models \varphi[e]$ if and only if $\mathcal{A} \models \psi[e]$.

Exercise 6.3. Prove all the listed properties of validity under an assignment in detail.

And what about the notion of truth (validity) in a structure?

- If $\mathcal{A} \models \varphi$, then $\mathcal{A} \not\models \neg\varphi$. If φ is a sentence, then the converse implication also holds (i.e., it is 'if and only if').
- $\mathcal{A} \models \varphi \wedge \psi$ if and only if $\mathcal{A} \models \varphi$ and $\mathcal{A} \models \psi$,
- If $\mathcal{A} \models \varphi$ or $\mathcal{A} \models \psi$, then $\mathcal{A} \models \varphi \vee \psi$. If φ is a sentence, then the converse implication also holds (i.e., it is 'if and only if').
- $\mathcal{A} \models \varphi$ if and only if $\mathcal{A} \models (\forall x)\varphi$.

The *general closure* of a formula $\varphi(x_1, \dots, x_n)$ (i.e., $x_1, \dots, x_n$ are all the free variables of the formula φ) is the sentence $(\forall x_1) \cdots (\forall x_n)\varphi$. From the last point, it follows that a formula is valid in a structure if and only if its general closure is valid in it.

Exercise 6.4. Prove all the listed properties of validity in a structure in detail.

Exercise 6.5. Give an example of a structure $\mathcal{A}$ and a formula φ such that $\mathcal{A} \not\models \varphi$ and yet $\mathcal{A} \not\models \neg\varphi$.

Exercise 6.6. Give an example of a structure $\mathcal{A}$ and formulas φ, ψ such that $\mathcal{A} \models \varphi \vee \psi$ but $\mathcal{A} \not\models \varphi$ and $\mathcal{A} \not\models \psi$.

6.5 Properties of Theories

Based on the notion of *validity*, we will build semantic terminology similar to that in propositional logic. A *theory* of a language L is any set T of L-formulas, whose elements are called *axioms*. A *model* of the theory T is an L-structure in which all axioms of the theory T are valid, i.e., $\mathcal{A} \models \varphi$ for all $\varphi \in T$, which we denote by $\mathcal{A} \models T$. The *class of models*[21] of the theory T is:

$$\mathrm{M}_L(T) = \{\mathcal{A} \in \mathrm{M}_L \mid \mathcal{A} \models T\}$$

As in propositional logic, we will often omit the language L when it is clear from the context and write $M(\varphi_1, \dots, \varphi_n)$ instead of $M(\{\varphi_1, \dots, \varphi_n\})$ and $M(T, \varphi)$ instead of $M(T \cup \{\varphi\})$.

[21]Recall that we cannot say 'set'.

6.5.1 Validity in a Theory

If T is a theory in the language L and φ is an L-formula, then we say that φ is:

- *valid (true) in T*, denoted $T \models \varphi$, if $\mathcal{A} \models \varphi$ for all $\mathcal{A} \in \mathrm{M}(T)$ (in other words: $\mathrm{M}(T) \subseteq \mathrm{M}(\varphi)$),
- *contradictory in T*, if $T \models \neg\varphi$, i.e., if it is contradictory in every model of T (in other words: $\mathrm{M}(T) \cap \mathrm{M}(\varphi) = \emptyset$),
- *independent in T*, if it is neither valid in T nor contradictory in T.

If we have an empty theory $T = \emptyset$ (i.e., $\mathrm{M}(T) = \mathrm{M}_L$), then we omit the theory T, write $\models \varphi$, and say that φ *is (universally) valid, (logically) valid, is a tautology*; similarly for other notions.

A theory is *inconsistent* if the *contradiction* $\bot$ is valid in it; $\bot$ in predicate logic can be defined as $R(x_1, \ldots, x_n) \wedge \neg R(x_1, \ldots, x_n)$, where R is any (say, the first) relation symbol from the language or the equality symbol (recall: if the language has no relation symbol, it must be with equality). A theory is *inconsistent* if and only if every formula is valid in it, or equivalently, if and only if it has no model. Otherwise, we say that the theory is *consistent* (if the contradiction is not valid in it, equivalently, if it has at least one model).

Sentences valid in T are called *consequences* of T; the *set of all consequences* of T in the language L is:

$$\mathrm{Csq}_L(T) = \{\varphi \mid \varphi \text{ is a sentence and } T \models \varphi\}$$

Completeness in Predicate Logic

How about the notion of *completeness* of a theory?[22]

Definition 6.5.1. A theory T is *complete* if it is consistent and every *sentence* is either valid in T or contradictory in T.

However, we cannot say that a theory is complete if and only if it has a single model. If we have one model, we derive infinitely many different, but *isomorphic* models, i.e., differing only by the naming of the elements of the universe.[23] Even considering a single model 'up to isomorphism' would not be sufficient. The correct notion is called *elementary equivalence*:

Definition 6.5.2. Structures $\mathcal{A}, \mathcal{B}$ (in the same language) are *elementarily equivalent* if the same sentences are valid in both of them. We denote this by $\mathcal{A} \equiv \mathcal{B}$.

Example 6.5.3. An example of structures that are elementarily equivalent but not isomorphic are the ordered sets $\mathcal{A} = \langle \mathbb{Q}, \leq \rangle$ and $\mathcal{B} = \langle \mathbb{R}, \leq \rangle$. They are not isomorphic because $\mathbb{Q}$ is countable while $\mathbb{R}$ is uncountable, so there is not even a *bijection* between their universes. It is not difficult to show that for any sentence φ, $\mathcal{A} \models \varphi \Leftrightarrow \mathcal{B} \models \varphi$ holds: by induction on the structure of the formula φ, the only non-trivial case is the existential quantifier, and the key property is the *density* of both orders, i.e., the following property:

$$(x \leq y \wedge \neg x = y) \rightarrow (\exists z)(x \leq z \wedge z \leq y \wedge \neg x = z \wedge \neg y = z)$$

[22] Recall that a *propositional* theory is complete if it is consistent and every proposition is either valid in it or its negation is valid in it. Equivalently, it has exactly one model.

[23] Formally, the notion of *isomorphism* is defined later in the part on *model theory*, in Section **??**, but it is a generalization of the isomorphism you know from graph theory.

Observation 6.5.4. *A theory is complete if and only if it has exactly one model up to elementary equivalence.*

Validity by Unsatisfiability

The question of truth (validity) in a given theory can be reduced to the problem of the existence of a model:

Proposition 6.5.5 (On Unsatisfiability and Validity)**.** *If T is a theory and φ is a* sentence *(in the same language), then: $T \cup \{\neg\varphi\}$ has no model if and only if $T \models \varphi$.*

Proof. The following equivalences hold: $T \cup \{\neg\varphi\}$ has no model if and only if $\neg\varphi$ is not valid in any model of T, if and only if (since it is a sentence) φ is valid in every model of T. □

The assumption that φ is a sentence is necessary: consider the theory $T = \{P(c)\}$ and the formula $\varphi = P(x)$ (which is not a sentence). Then $\{P(c), \neg P(x)\}$ has no model, but $P(c) \not\models P(x)$. (Here, P is a unary relation symbol and c is a constant symbol.)

6.5.2 Examples of Theories

Here are some examples of important theories.

Theory of Graphs

The *theory of graphs* is a theory in the language $L = \langle E \rangle$ with equality, satisfying the axioms of *irreflexivity* and *symmetry*:

$$T_{\text{graph}} = \{\neg E(x,x), E(x,y) \to E(y,x)\}$$

The models of T_{graph} are structures $\mathcal{G} = \langle G, E^{\mathcal{G}} \rangle$, where $E^{\mathcal{G}}$ is a symmetric irreflexive relation, i.e., so-called *simple* graphs, where the edge $\{x,y\}$ is represented by the ordered pairs $(x,y), (y,x)$.

- The formula $\neg x = y \to E(x,y)$ holds in a graph if and only if the graph is a *clique* (a *complete* graph). The formula is thus independent in T_{graph}.
- The formula $(\exists y_1)(\exists y_2)(\neg y_1 = y_2 \wedge E(x,y_1) \wedge E(x,y_2) \wedge (\forall z)(E(x,z) \to z = y_1 \vee z = y_2))$ expresses that each vertex has degree exactly 2. It thus holds precisely in graphs that are disjoint unions of cycles and is independent in the theory T_{graph}.

Theory of Order

The *theory of order* is a theory in the language of order $L = \langle \leq \rangle$ with equality, whose axioms are:

$$\begin{aligned} T = \{&x \leq x, \\ &x \leq y \wedge y \leq x \to x = y, \\ &x \leq y \wedge y \leq z \to x \leq z\} \end{aligned}$$

These axioms are called *reflexivity*, *antisymmetry*, and *transitivity*. The models of T are L-structures $\langle S, \leq^S \rangle$, in which the axioms T are valid, so-called *(partially) ordered sets*. For example: $\mathcal{A} = \langle \mathbb{N}, \leq \rangle$, $\mathcal{B} = \langle \mathcal{P}(X), \subseteq \rangle$ for $X = \{0, 1, 2\}$.

- The formula $x \leq y \vee y \leq x$ (*linearity*) is valid in $\mathcal{A}$ but not in $\mathcal{B}$, because it is not valid, for example, under the assignment where $e(x) = \{0\}$, $e(y) = \{1\}$ (we write $\mathcal{B} \not\models \varphi[e]$). It is thus independent in T.
- The sentence $(\exists x)(\forall y)(y \leq x)$ (denote it by ψ) is valid in $\mathcal{B}$ and contradictory in $\mathcal{A}$, we write $\mathcal{B} \models \psi$, $\mathcal{A} \models \neg\psi$. It is thus also independent in T.
- The formula $(x \leq y \wedge y \leq z \wedge z \leq x) \rightarrow (x = y \wedge y = z)$ (denote it by χ) is valid in T, we write $T \models \chi$. The same applies to its *general closure* $(\forall x)(\forall y)(\forall z)\chi$.

Algebraic Theories

- The *theory of groups* is a theory in the language $L = \langle +, -, 0 \rangle$ with equality, whose axioms are:

$$\begin{aligned} T_1 = \{ & x + (y + z) = (x + y) + z, \\ & 0 + x = x,\ x + 0 = 0, \\ & x + (-x) = 0,\ (-x) + x = 0 \} \end{aligned}$$

  These properties are called *associativity of* $+$, *neutrality of* 0 *with respect to* $+$, and $-x$ *is the inverse element of* x *(with respect to* $+$ *and* 0*)*.

- The *theory of commutative groups* additionally contains the axiom $x + y = y + x$ (*commutativity of* $+$), thus:

$$T_2 = T_1 \cup \{x + y = y + x\}$$

- The *theory of rings* is in the language $L = \langle +, -, 0, \cdot, 1 \rangle$ with equality and adds the following axioms:

$$\begin{aligned} T_3 = T_2 \cup \{ & 1 \cdot x = x \cdot 1, \\ & x \cdot (y \cdot z) = (x \cdot y) \cdot z, \\ & x \cdot (y + z) = x \cdot y + x \cdot z, \\ & (x + y) \cdot z = x \cdot z + y \cdot z \} \end{aligned}$$

  These properties are called *neutrality of* 1 *with respect to* $\cdot$, *associativity of* $\cdot$, and *(left and right) distributivity of* $\cdot$ *with respect to* $+$.

- The *theory of commutative rings* additionally has the axiom of *commutativity of* $\cdot$, thus:

$$T_4 = T_3 \cup \{x \cdot y = y \cdot x\}$$

- The *theory of fields* is in the same language but additionally has the axioms of *existence of an inverse element with respect to* $\cdot$ and *non-triviality*:

$$T_5 = T_4 \cup \{\neg\, x = 0 \rightarrow (\exists y)(x \cdot y = 1), \neg\, 0 = 1\}$$

- The *theory of ordered fields* is in the language $\langle +, -, 0, \cdot, 1, \leq \rangle$ with equality, consisting of the axioms of the theory of fields, the theory of order along with the axiom of linearity, and the following axioms of *compatibility of the order*: $x \leq y \to (x + z \leq y + z)$ and $(0 \leq x \wedge 0 \leq y) \to 0 \leq x \cdot y$. (The models are thus fields with *linear (total)* order compatible with the field operations in this sense.)

6.6 Substructure, Expansion, Reduct

In this section, we will look at ways to create new structures from existing ones.

Substructure

The notion of *substructure* generalizes subgroups, subspaces of a vector space, and induced subgraphs of a graph: we select a subset B of the universe of the structure $\mathcal{A}$ and create on it a structure $\mathcal{B}$ of the same signature, which 'inherits' the relations, functions, and constants. To do this, we need the set B to be *closed* under all functions and contain all constants.[24]

Definition 6.6.1 (Substructure)**.** Let $\mathcal{A} = \langle A, \mathcal{R}^{\mathcal{A}}, \mathcal{F}^{\mathcal{A}} \rangle$ be a structure in the signature $\langle \mathcal{R}, \mathcal{F} \rangle$. A structure $\mathcal{B} = \langle B, \mathcal{R}^{\mathcal{B}}, \mathcal{F}^{\mathcal{B}} \rangle$ is an *(induced) substructure of* $\mathcal{A}$, denoted $\mathcal{B} \subseteq \mathcal{A}$, if

- $\emptyset \neq B \subseteq A$,
- $R^{\mathcal{B}} = R^{\mathcal{A}} \cap B^{\mathrm{ar(R)}}$ for each relation symbol $R \in \mathcal{R}$,
- $f^{\mathcal{B}} = f^{\mathcal{A}} \cap (B^{\mathrm{ar(f)}} \times B)$ for each function symbol $f \in \mathcal{F}$ (i.e., the function $f^{\mathcal{B}}$ is the restriction of $f^{\mathcal{A}}$ to the set B, and also its outputs are all from B),
- in particular, for each constant symbol $c \in \mathcal{F}$ we have $c^{\mathcal{B}} = c^{\mathcal{A}} \in B$.

A set $C \subseteq A$ is *closed* under a function $f : A^n \to A$ if $f(x_1, \ldots, x_n) \in C$ for all $x_i \in C$. We have:

Observation 6.6.2. *A set $\emptyset \neq C \subseteq A$ is the universe of a substructure of the structure $\mathcal{A}$ if and only if C is closed under all functions of the structure $\mathcal{A}$ (including constants).*

In that case, we call this substructure the *restriction* of $\mathcal{A}$ to the set C, and denote it by $\mathcal{A} \upharpoonright C$.

Example 6.6.3. $\underline{\mathbb{Z}} = \langle \mathbb{Z}, +, \cdot, 0 \rangle$ is a substructure of $\underline{\mathbb{Q}} = \langle \mathbb{Q}, +, \cdot, 0 \rangle$, we can write $\underline{\mathbb{Z}} = \underline{\mathbb{Q}} \upharpoonright \mathbb{Z}$. The structure $\underline{\mathbb{N}} = \langle \mathbb{N}, +, \cdot, 0 \rangle$ is a substructure of both these structures, $\underline{\mathbb{N}} = \underline{\mathbb{Q}} \upharpoonright \mathbb{N} = \underline{\mathbb{Z}} \upharpoonright \mathbb{N}$.

Validity in Substructure

How is it with validity of formulas in a substructure? Here are some simple observations about *open* formulas.

Observation 6.6.4. *If $\mathcal{B} \subseteq \mathcal{A}$, then for any* open *formula φ and variable assignment $e\colon \mathrm{Var} \to B$, we have that: $\mathcal{B} \models \varphi[e]$ if and only if $\mathcal{A} \models \varphi[e]$.*

[24] Just as not every set of vectors is a subspace, it must contain the zero vector, contain all scalar multiples of each vector, and for any pair of vectors, contain their sum. In other words, only (non-empty) sets closed under *linear combinations* of vectors form subspaces.

Proof. For atomic formulas, this is obvious; further, it can be easily proved by induction on the structure of the formula. □

Corollary 6.6.5. *An* open *formula is valid in the structure $\mathcal{A}$ if and only if it is valid in every substructure $\mathcal{B} \subseteq \mathcal{A}$.*

We say that a theory T is *open* if all its axioms are open formulas.

Corollary 6.6.6. *The models of an open theory are closed under substructures, i.e., every substructure of a model of an open theory is also a model of this theory.*

Example 6.6.7. The theory of graphs is open. Every substructure of a graph (a model of the theory of graphs) is also a graph, called an (induced) *subgraph.*[25] Similarly, for subgroups or Boolean subalgebras.

Example 6.6.8. The theory of fields is not open. As we will show later, it is not even *openly axiomatizable*, i.e., there is no equivalent open theory—there is no way to get rid of the quantifier in the axiom of the existence of an inverse element. The substructure of the field of real numbers $\mathbb{Q}$ on the set of all integers $\mathbb{Q} \restriction \mathbb{Z}$ is not a field. (It is a so-called *ring*, but non-zero elements other than $1, -1$ do not have a multiplicative inverse, e.g., the equation $2 \cdot x = 1$ has no solution in $\mathbb{Z}$).

Generated Substructure

What to do if we have a subset of the universe that is *not* closed under the functions of the structure? In that case, we consider the *closure* of this set under the functions.[26]

Definition 6.6.9. Let $\mathcal{A} = \langle A, \mathcal{R}^{\mathcal{A}}, \mathcal{F}^{\mathcal{A}} \rangle$ be a structure and a non-empty subset $X \subseteq A$. Let B be the smallest subset of A that contains the set X and is closed under all functions of the structure $\mathcal{A}$ (i.e., it also contains all constants). Then the substructure $\mathcal{A} \restriction B$ is said to be *generated* by the set X, and denoted by $\mathcal{A}\langle X \rangle$.

Example 6.6.10. Consider the structures $\underline{\mathbb{Q}} = \langle \mathbb{Q}, +, \cdot, 0 \rangle$, $\underline{\mathbb{Z}} = \langle \mathbb{Z}, +, \cdot, 0 \rangle$, and $\underline{\mathbb{N}} = \langle \mathbb{N}, +, \cdot, 0 \rangle$. Then $\underline{\mathbb{Q}}\langle \{1\} \rangle = \underline{\mathbb{N}}$, $\underline{\mathbb{Q}}\langle \{-1\} \rangle = \underline{\mathbb{Z}}$, and $\underline{\mathbb{Q}}\langle \{2\} \rangle$ is a substructure of $\underline{\mathbb{N}}$ on the set of all even numbers.

Example 6.6.11. If $\mathcal{A}$ has no functions (not even constants), e.g., if it is a graph or an order, then there is nothing to generate, and $\mathcal{A}\langle X \rangle = \mathcal{A} \restriction X$.

Expansion and Reduct

So far, we have constructed new structures by changing the universe. However, we can also keep the universe the same and add or remove relations, functions, and constants. The result of such an operation is called an *expansion* or a *reduct.* Note that this is a structure in a different signature.

[25] The notion of a *subgraph* in graph theory often means just $E^{\mathcal{B}} \subseteq E^{\mathcal{A}} \cap (B \times B)$, not $E^{\mathcal{B}} = E^{\mathcal{A}} \cap (B \times B)$. However, we will use the term *subgraph* in the stricter sense, as an induced subgraph.

[26] See the notion of *linear span* of a set of vectors.

Definition 6.6.12 (Expansion and Reduct)**.** Let $L \subseteq L'$ be languages, $\mathcal{A}$ an L-structure, and $\mathcal{A}'$ an L'-structure on the same domain $A = A'$. If the interpretation of each [relation, function, constant] symbol from L is the same [relation, function, constant] in both $\mathcal{A}$ and $\mathcal{A}'$, then we say that the structure $\mathcal{A}'$ is an *expansion* of the structure $\mathcal{A}$ to the language L' (also called an L'*-expansion*) and that the structure $\mathcal{A}$ is a *reduct* of the structure $\mathcal{A}'$ to the language L (also called an L*-reduct*).

Example 6.6.13. Consider the group of integers $\langle \mathbb{Z}, +, -, 0\rangle$. Then the structure $\langle \mathbb{Z}, +\rangle$ is its reduct, while the structure $\langle \mathbb{Z}, +, -, 0, \cdot, 1\rangle$ (the *ring* of integers) is its expansion.

Example 6.6.14. Consider a graph $\mathcal{G} = \langle G, E^{\mathcal{G}}\rangle$. Then the structure $\langle G, E^G, c_v^{\mathcal{G}}\rangle_{v\in G}$ in the language $\langle E, c_v\rangle_{v\in G}$, where $c_v^{\mathcal{G}} = v$ for all vertices $v \in G$, is an *expansion of* $\mathcal{G}$ *with names of elements (from the set* G*).*

6.6.1 Theorem on Constants

The *theorem on constants* states (informally) that satisfying a formula with a single free variable is equivalent to satisfying a sentence in which this free variable is replaced (substituted) by a *new* constant symbol (which is not bound by any axioms). The key fact is that this new symbol can be interpreted as any (hence each) element in each of the models. We will later use this trick in the tableau method.

Theorem 6.6.15 (Theorem on Constants)**.** *Let* φ *be a formula in the language* L *with free variables* $x_1, \ldots, x_n$*. Let* L' *be the extension of the language with new constant symbols* $c_1, \ldots, c_n$ *and let* T' *be the same theory as* T *but in the language* L'*. Then:*

$$T \models \varphi \text{ if and only if } T' \models \varphi(x_1/c_1, \ldots, x_n/c_n)$$

Proof. It is sufficient to prove the statement for one free variable x and one constant c; by induction, it can be easily extended to n constants.

First, suppose that φ holds in every model of the theory T. We want to show that $\varphi(x/c)$ holds in every model $\mathcal{A}'$ of the theory T'. So take such a model $\mathcal{A}'$ and any assignment $e\colon \mathrm{Var} \to A'$ and show that $\mathcal{A}' \models \varphi(x/c)[e]$.

Let $\mathcal{A}$ be the reduct of $\mathcal{A}'$ to the language L ('forget' the constant $c^{\mathcal{A}'}$). Note that $\mathcal{A}$ is a model of the theory T (the axioms of T are the same as T', they do not contain the symbol c) and hence φ holds in it. Since, by assumption, $\mathcal{A} \models \varphi[e']$ for *any* assignment e', it also holds for the assignment $e(x/c^{\mathcal{A}'})$ in which we evaluate the variable x as the interpretation of the constant symbol c in the structure $\mathcal{A}'$, so we have $\mathcal{A} \models \varphi[e(x/c^{\mathcal{A}'})]$. But this means that $\mathcal{A}' \models \varphi(x/c)[e]$, which is what we wanted to prove.

Conversely, suppose that $\varphi(x/c)$ holds in every model of the theory T' and show that φ holds in every model $\mathcal{A}$ of the theory T. So take such a model $\mathcal{A}$ and some assignment $e\colon \mathrm{Var} \to A$ and show that $\mathcal{A} \models \varphi[e]$.

Let $\mathcal{A}'$ be the expansion of $\mathcal{A}$ to the language L', where the constant symbol c is interpreted as the element $c^{\mathcal{A}'} = e(x)$. Since, by assumption, $\mathcal{A}' \models \varphi(x/c)[e']$ for all assignments e', it also holds that $\mathcal{A}' \models \varphi(x/c)[e]$, which means that $\mathcal{A}' \models \varphi[e]$. (Since $e = e(x/c^{\mathcal{A}'})$ and $\mathcal{A}' \models \varphi(x/c)[e]$ if and only if $\mathcal{A}' \models \varphi[e(x/c^{\mathcal{A}'})]$, which is $\mathcal{A}' \models \varphi[e]$.) But the formula φ does not contain c (here we use that c is *new*), so we also have $\mathcal{A} \models \varphi[e]$. □

6.7 Extension of Theories

The notion of *extension* of a theory is defined similarly as in propositional logic:

Definition 6.7.1 (Extension of a Theory)**.** Let T be a theory in the language L.

- An *extension* of the theory T is any theory T' in the language $L' \supseteq L$ satisfying $\mathrm{Csq}_L(T) \subseteq \mathrm{Csq}_{L'}(T')$,
- it is a *simple extension* if $L' = L$,
- it is a *conservative extension* if $\mathrm{Csq}_L(T) = \mathrm{Csq}_L(T') = \mathrm{Csq}_{L'}(T') \cap \mathrm{Fm}_L$, where Fm_L denotes the set of all formulas in the language L.
- The theory T' (in the language L) is *equivalent* to the theory T if T' is an extension of T and T is an extension of T'.

Similar to propositional logic, for theories in the same language, the following semantic description of these notions holds:

Observation 6.7.2. *Let T, T' be theories in the language L. Then:*

- *T' is an extension of T if and only if $\mathrm{M}_L(T') \subseteq \mathrm{M}_L(T)$.*
- *T' is equivalent to T if and only if $\mathrm{M}_L(T') = \mathrm{M}_L(T)$.*

What about the case when the theory T' is in a larger language than T? Recall the situation in propositional logic, described in Observation 2.4.7. We will formulate and prove an analogous statement: While in propositional logic we were adding values for new atomic propositions or forgetting them, in predicate logic we will expand or reduce structures, i.e., add or forget interpretations of relation, function, and constant symbols. The principle behind the two statements (and their proofs) is the same.

Proposition 6.7.3. *Let $L \subseteq L'$ be languages, T a theory in the language L, and T' a theory in the language L'.*

(i) T' is an extension of the theory T if and only if the reduct of every model of T' to the language L is a model of T.

(ii) If T' is an extension of the theory T, and every model of T can be expanded to the language L' into some model of the theory T', then T' is a conservative extension of the theory T.

Remark 6.7.4. The reverse implication in part (ii) also holds, but the proof is not as simple as in propositional logic, so we will not present it. (The technical issue is how to obtain, from a model of T that cannot be expanded to a model of T', an L-sentence that holds in T but not in T'.)

Proof. First, let us prove (i): Let $\mathcal{A}'$ be a model of the theory T' and denote by $\mathcal{A}$ its reduct to the language L. Since T' is an extension of the theory T, every axiom $\varphi \in T$ holds in T', and thus in $\mathcal{A}'$. But then $\mathcal{A} \models \varphi$ (since φ contains only symbols from the language L), hence $\mathcal{A}$ is a model of T.

Conversely, let φ be an L-sentence such that $T \models \varphi$. We want to show that $T' \models \varphi$. For any model $\mathcal{A}' \in \mathrm{M}_{L'}(T')$ we know that its L-reduct $\mathcal{A}$ is a model of T, hence $\mathcal{A} \models \varphi$. Therefore, $\mathcal{A}' \models \varphi$ (again, since φ is in the language L).

Now (ii): Take any L-sentence φ that holds in the theory T', and show that it also holds in T. Every model $\mathcal{A}$ of the theory T can be expanded to some model $\mathcal{A}'$ of the theory T'. We know that $\mathcal{A}' \models \varphi$, so $\mathcal{A} \models \varphi$. Thus, we have shown that $T \models \varphi$, i.e., it is a conservative extension. □

6.7.1 Extension by Definitions

Now we will introduce a special kind of conservative extension, namely the *extension by definitions* of new (relation, function, constant) symbols.

Definition of a Relation Symbol

The simplest case is defining a new relation symbol $R(x_1, \ldots, x_n)$. Any formula with n free variables $\psi(x_1, \ldots, x_n)$ can serve as the definition.

Example 6.7.5. First, let us provide some examples:

- Any theory in a language with equality can be extended with the binary relation symbol $\neq$, which is *defined* by the formula $\neg x_1 = x_2$. This means that we require $x_1 \neq x_2 \leftrightarrow \neg x_1 = x_2$.
- The theory of order can be extended with the symbol $<$ for strict order, which is *defined* by the formula $x_1 \leq x_2 \wedge \neg x_1 = x_2$. This means that we require $x_1 < x_2 \leftrightarrow x_1 \leq x_2 \wedge \neg x_1 = x_2$.
- In arithmetic, we can introduce the symbol $\leq$ using $x_1 \leq x_2 \leftrightarrow (\exists y)(x_1 + y = x_2)$.

Now, we give the definition:

Definition 6.7.6 (Definition of a Relation Symbol)**.** Let T be a theory and $\psi(x_1, \ldots, x_n)$ a formula in the language L. Let L' be the extension of the language L with a new n-ary relation symbol R. The *extension of the theory T by the definition of R by the formula ψ* is the L'-theory:

$$T' = T \cup \{R(x_1, \ldots, x_n) \ \leftrightarrow \ \psi(x_1, \ldots, x_n)\}$$

Note that every model of T can be *uniquely* expanded to a model of T'. From Proposition 6.7.3, it follows immediately:

Corollary 6.7.7. *T' is a conservative extension of T.*

We will also show that the new symbol can be replaced in formulas by its definition, thus obtaining a (T'-equivalent) formula in the original language:

Proposition 6.7.8. *For every L'-formula φ', there exists an L-formula φ such that $T' \models \varphi' \leftrightarrow \varphi$.*

Proof. We need to replace atomic subformulas that use the new symbol R, i.e., of the form $R(t_1, \ldots, t_n)$. Such a subformula is replaced by the formula $\psi'(x_1/t_1, \ldots, x_n/t_n)$, where ψ' is a variant of ψ ensuring the substitutability of all terms, i.e., for example, we rename all bound variables of ψ to entirely new ones (not appearing in the formula φ'). □

Definition of a Function Symbol

We define a new function symbol similarly, but we must ensure that the definition provides a unique way to interpret the new symbol as a function.

Example 6.7.9. Again, we start with examples:

- In the theory of groups, we can introduce a *binary* function symbol $-_b$ using $+$ and unary $-$ as follows:
$$x_1 -_b x_2 = y \leftrightarrow x_1 + (-x_2) = y$$
It is clear that for every x, y, there *exists* a *unique* z satisfying the definition.

- Consider the theory of *linear orders*, i.e., the theory of orders together with the linearity axiom $x \leq y \vee y \leq x$. Define the binary function symbol min as follows:
$$\min(x_1, x_2) = y \leftrightarrow y \leq x_1 \wedge y \leq x_2 \wedge (\forall z)(z \leq x_1 \wedge z \leq x_2 \rightarrow z \leq y)$$
Existence and uniqueness hold thanks to linearity. However, if we had only the theory of orders, such a formula would not be a good definition: in some models, $\min(x_1, x_2)$ would not exist for some elements, thus failing the required *existence.*

Definition 6.7.10 (Definition of a Function Symbol)**.** Let T be a theory and $\psi(x_1, \ldots, x_n, y)$ a formula in the language L. Let L' be the extension of the language L with a new n-ary function symbol f. Let the following hold in the theory T:

- *existence axiom* $(\exists y)\psi(x_1, \ldots, x_n, y)$,
- *uniqueness axiom* $\psi(x_1, \ldots, x_n, y) \wedge \psi(x_1, \ldots, x_n, z) \rightarrow y = z$.

Then the *extension of the theory T by the definition of f by the formula ψ* is the L'-theory:

$$T' = T \cup \{f(x_1, \ldots, x_n) = y \leftrightarrow \psi(x_1, \ldots, x_n, y)\}$$

The formula ψ thus defines in each model an $(n+1)$-ary relation, and we require that this relation be a function, i.e., that for each n-tuple of elements, there exists a unique way to extend it into an $(n+1)$-tuple that is an element of this relation. Note that if the defining formula ψ is of the form $t(x_1, \ldots, x_n) = y$, where $x_1, \ldots, x_n$ are variables of the L-term t, then the existence and uniqueness axioms always hold.

Again, we have that every model of T can be *uniquely* expanded to a model of T', hence:

Corollary 6.7.11. *T' is a conservative extension of T.*

And the same statement about unwinding definitions holds:

Proposition 6.7.12. *For every L'-formula φ', there exists an L-formula φ such that $T' \models \varphi' \leftrightarrow \varphi$.*

Proof. It suffices to prove for a formula φ' with a single occurrence of the symbol f; if there are multiple occurrences, we apply the procedure inductively, in the case of nested occurrences in one term $f(\ldots f(\ldots)\ldots)$, we proceed from inner to outer.

Denote by φ^* the formula obtained from φ' by replacing the term $f(t_1, \ldots, t_n)$ with a *new* variable z. Construct the formula φ as follows:

$$(\exists z)(\varphi^* \wedge \psi'(x_1/t_1, \ldots, x_n/t_n, y/z))$$

where ψ' is a variant of ψ ensuring the substitutability of all terms.

Let $\mathcal{A}$ be a model of the theory T' and e a variable assignment. Denote $a = (f(t_1, \ldots, t_n))^{\mathcal{A}}[e]$. Due to existence and uniqueness, it holds:

$$\mathcal{A} \models \psi'(x_1/t_1, \ldots, x_n/t_n, y/z)[e] \text{ if and only if } e(z) = a$$

Thus, $\mathcal{A} \models \varphi[e]$ if and only if $\mathcal{A} \models \varphi^*[e(z/a)]$, if and only if $\mathcal{A} \models \varphi'[e]$. This holds for any variable assignment e, hence $\mathcal{A} \models \varphi' \leftrightarrow \varphi$ for every model T', thus $T' \models \varphi' \leftrightarrow \varphi$. $\square$

Definition of a Constant Symbol

A constant symbol is a special case of a function symbol of arity 0. Thus, the same statements hold. The existence and uniqueness axioms are: $(\exists y)\psi(y)$ and $\psi(y) \wedge \psi(z) \rightarrow y = z$. The extension by definition of a constant symbol c by the formula $\psi(y)$ is the theory $T' = T \cup \{c = y \leftrightarrow \psi(y)\}$.

Example 6.7.13. Let us show two examples:

- Any theory in the language of arithmetic can be extended by defining the constant symbol 1 with the formula $\psi(y)$ of the form $y = S(0)$, thus adding the axiom $1 = y \leftrightarrow y = S(0)$.
- Consider the theory of fields and a new symbol $\frac{1}{2}$, defined by the formula $y \cdot (1 + 1) = 1$, i.e., adding the axiom:
  $$\frac{1}{2} = y \leftrightarrow y \cdot (1 + 1) = 1$$
  This is *not* a correct extension by definition because the existence axiom does not hold. In the two-element field $\mathbb{Z}_2$ (and in every field *of characteristic 2*), the equation $y \cdot (1 + 1) = 1$ has no solution because $1 + 1 = 0$.

  However, if we take the theory of fields with characteristic not equal to 2, i.e., adding the axiom $\neg(1 + 1 = 0)$ to the theory of fields, it becomes a correct extension by definition. For example, in the field $\mathbb{Z}_3$, we have $\frac{1}{2}^{\mathbb{Z}_3} = 2$.

Extension by Definitions

If we have an L-theory T and an L'-theory T', we say that T' is an *extension* of T *by definitions* if it is obtained from T by a successive extension by definitions of relation and function (or possibly constant) symbols. The properties we proved about extensions by one symbol (whether relation or function) easily extend inductively to multiple symbols:

Corollary 6.7.14. *If T' is an extension of the theory T by definitions, then:*

- *Every model of the theory T can be uniquely expanded to a model of T'.*
- *T' is a conservative extension of T.*
- *For every L'-formula φ', there exists an L-formula φ such that $T' \models \varphi' \leftrightarrow \varphi$.*

Finally, let us show one more example, illustrating also the unwinding of definitions:

Example 6.7.15. In the theory $T = \{(\exists y)(x + y = 0), (x + y = 0) \wedge (x + z = 0) \rightarrow y = z\}$ of the language $L = \langle +, 0, \leq \rangle$ with equality, we can introduce $<$ and a unary function symbol $-$ by adding the following axioms:

$$\begin{aligned} -x = y \ &\leftrightarrow\ x + y = 0 \\ x < y \ &\leftrightarrow\ x \leq y \wedge \neg(x = y) \end{aligned}$$

The formula $-x < y$ (in the language $L' = \langle +, -, 0, \leq, < \rangle$ with equality) is in this extension by definitions equivalent to the following formula:

$$(\exists z)((z \leq y \wedge \neg(z = y)) \wedge x + z = 0)$$

6.8 Definability in Structures

A formula with one free variable x can be understood as a *property* of elements. In a given structure, such a formula *defines* the set of elements that satisfy this property, i.e., those for which the formula is valid under an assignment e where $e(x) = a$. If we have a formula with two free variables, it defines a binary relation, etc. We now formalize this concept. Recall that the notation $\varphi(x_1, \ldots, x_n)$ means that $x_1, \ldots, x_n$ are precisely all the free variables of the formula φ.

Definition 6.8.1 (Definable Sets)**.** Let $\varphi(x_1, \ldots, x_n)$ be a formula and $\mathcal{A}$ a structure in the same language. The *set defined by the formula* $\varphi(x_1, \ldots, x_n)$ *in the structure* $\mathcal{A}$, denoted by $\varphi^{\mathcal{A}}(x_1, \ldots, x_n)$, is:

$$\varphi^{\mathcal{A}}(x_1, \ldots, x_n) = \{(a_1, \ldots, a_n) \in A^n \mid \mathcal{A} \models \varphi[e(x_1/a_1, \ldots, x_n/a_n)]\}$$

For brevity, we can also write $\varphi^{\mathcal{A}}(\bar{x}) = \{\bar{a} \in A^n \mid \mathcal{A} \models \varphi[e(\bar{x}/\bar{a})]\}$.

Example 6.8.2. Here are some examples:

- The formula $\neg(\exists y)E(x, y)$ defines the set of all *isolated* vertices in a given simple graph.
- Consider the field of real numbers $\underline{\mathbb{R}}$. The formula $(\exists y)(y \cdot y = x) \wedge \neg x = 0$ defines the set of all positive real numbers.
- The formula $x \leq y \wedge \neg x = y$ defines the relation of *strict ordering* $<^S$ in a given ordered set $\langle S, \leq^S \rangle$.

It is often useful to talk about properties of elements relative to other elements of a given structure. This cannot be expressed purely syntactically, but we can substitute elements of the structure as *parameters* for some of the free variables. The notation $\varphi(\bar{x}, \bar{y})$ means that the formula φ has free variables $x_1, \ldots, x_n, y_1, \ldots, y_k$ (for some n, k).

Definition 6.8.3. Let $\varphi(\bar{x}, \bar{y})$ be a formula, where $|\bar{x}| = n$ and $|\bar{y}| = k$, $\mathcal{A}$ a structure in the same language, and $\bar{b} \in A^k$. The *set defined by the formula* $\varphi(\bar{x}, \bar{y})$ *with parameters* $\bar{b}$ *in the structure* $\mathcal{A}$, denoted by $\varphi^{\mathcal{A},\bar{b}}(\bar{x}, \bar{y})$, is:

$$\varphi^{\mathcal{A},\bar{b}}(\bar{x}, \bar{y}) = \{\bar{a} \in A^n \mid \mathcal{A} \models \varphi[e(\bar{x}/\bar{a}, \bar{y}/\bar{b})]\}$$

For a structure $\mathcal{A}$ and a subset $B \subseteq A$, let $\mathrm{Df}^n(\mathcal{A}, B)$ denote the set of all sets definable in the structure $\mathcal{A}$ with parameters from B.

Example 6.8.4. For $\varphi(x, y) = E(x, y)$, $\varphi^{\mathcal{G},v}(x, y)$ is the set of all neighbors of vertex v.

Observation 6.8.5. *The set* $\mathrm{Df}^n(\mathcal{A}, B)$ *is closed under complement, intersection, and union, and contains* $\emptyset$ *and* A^n*. Therefore, it is a subalgebra of the power set algebra* $\mathcal{P}(A^n)$*.*

6.8.1 Database Queries

Definability finds natural application in relational databases, such as in the well-known query language SQL. A *relational database* consists of one or more *tables*, sometimes called *relations*, with rows of a table being *records* or *tuples.* Essentially, it is a structure in a purely relational language. Consider a database containing two tables, Program and Movies, as illustrated in Figure 6.3.

cinema	title	time
Atlas	Forrest Gump	20:00
Lucerna	Forrest Gump	21:00
Lucerna	Philadelphia	18:30
⋮	⋮	⋮

title	director	actor
Forrest Gump	R. Zemeckis	T. Hanks
Philadelphia	J. Demme	T. Hanks
Batman Returns	T. Burton	M. Keaton
⋮	⋮	⋮

Figure 6.3: Tables Program and Movies

An SQL query in its simplest form (ignoring, for example, *aggregate functions*) is essentially a formula, and the result of the query is the set defined by this formula (with parameters). For instance, when and where can we see a movie starring Tom Hanks?

select Program.cinema, Program.time **from** Program, Movies **where**
Program.title = Movies.title **and** Movies.actor = 'T. Hanks'

The result will be the set $\varphi^{\text{Database,'T. Hanks'}}(x_{\text{cinema}}, x_{\text{time}}, y_{\text{actor}})$ defined in the structure Database $= \langle D, \text{Program}, \text{Movies}\rangle$, where $D = \{\text{'Atlas'}, \text{'Lucerna'}, \dots, \text{'M. Keaton'}\}$, with the parameter 'T. Hanks' by the following formula $\varphi(x_{\text{cinema}}, x_{\text{time}}, y_{\text{actor}})$:

$$(\exists y_{\text{title}})(\exists y_{\text{director}})(\text{Program}(x_{\text{cinema}}, y_{\text{title}}, x_{\text{time}}) \land \text{Movies}(y_{\text{title}}, y_{\text{director}}, y_{\text{actor}}))$$

6.9 Relationship Between Propositional and Predicate Logic

We will now look at how propositional logic can be 'simulated' in predicate logic, in particular in the theory of Boolean algebras. First, we introduce the axioms of this theory:

Definition 6.9.1 (Boolean Algebras)**.** The *theory of Boolean algebras* is the theory of the language $L = \langle -, \land, \lor, \bot, \top\rangle$ with equality consisting of the following axioms:[27]

- *associativity* of $\land$ and $\lor$:

$$x \land (y \land z) = (x \land y) \land z$$
$$x \lor (y \lor z) = (x \lor y) \lor z$$

- *commutativity* of $\land$ and $\lor$:

$$x \land y = y \land x$$
$$x \lor y = y \lor x$$

[27]Note the *duality*: by swapping $\land$ with $\lor$ and $\bot$ with $\top$, we obtain the same axioms.

- *distributivity* of $\wedge$ over $\vee$ and $\vee$ over $\wedge$:

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$
$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$$

- *absorption*:

$$x \wedge (x \vee y) = x$$
$$x \vee (x \wedge y) = x$$

- *complementation*:

$$x \wedge (-x) = \bot$$
$$x \vee (-x) = \top$$

- *non-triviality*:

$$\neg(\bot = \top)$$

The smallest model is the *2-element Boolean algebra* $\langle \{0,1\}, f_\neg, f_\wedge, f_\vee, 0, 1\rangle$. Finite Boolean algebras are (up to *isomorphism*) precisely $\langle \{0,1\}^n, f_\neg^n, f_\wedge^n, f_\vee^n, (0,\dots,0), (1,\dots,1)\rangle$, where f^n means that the function f is applied component-wise.[28]

Propositions can thus be understood as *Boolean terms* (and the constants $\bot, \top$ represent falsity and truth). The truth value of a proposition (under a given assignment of propositional variables) is given by the value of the corresponding term in the 2-element Boolean algebra. Moreover, the *algebra of propositions* of a given propositional language or theory is a Boolean algebra (this holds even for infinite languages).

On the other hand, if we have an *open* formula φ (without equality), we can represent atomic propositions using propositional variables and obtain a proposition that holds if and only if φ holds. We will learn more about this direction in Chapter 8 (on resolution in predicate logic), where we will first eliminate quantifiers using the so-called *Skolemization.*

Propositional logic could also be introduced as a fragment of predicate logic if we allowed *nullary relations* (and nullary relation symbols in the language): $A^0 = \{\emptyset\}$, thus on any set there are precisely two nullary relations $R^A \subseteq A^0$: $R^A = \emptyset = 0$ and $R^A = \{\emptyset\} = \{0\} = 1$. However, we will not do that.

[28]These Boolean algebras are isomorphic to the *power set algebras* $\mathcal{P}(\{1,\dots,n\})$; the isomorphism is the bijection between subsets and their characteristic vectors.

Chapter 7

Tableau Method in Predicate Logic

In this chapter, we will demonstrate how to generalize the *method of analytic tableau* from propositional logic to predicate logic.[1] The method operates similarly but must be able to handle *quantifiers*.

7.1 Informal Introduction

In this section, we will informally introduce the tableau method in predicate logic. Formal definitions will follow later. We will start with two examples illustrating how the tableau method works in predicate logic and how it deals with quantifiers.

Example 7.1.1. Figure 7.1.1 shows two tableaux. These are tableau proofs (in logic, i.e., from the empty theory) of the *sentences* $(\exists x)\neg P(x) \rightarrow \neg(\forall x)P(x)$ (right) and $\neg(\forall x)P(x) \rightarrow (\exists x)\neg P(x)$ (left) in the language $L = \langle P\rangle$ (without equality), where P is a unary relation symbol. The symbol c_0 is an *auxiliary constant symbol* that we add to the language during the construction of the tableau.

Entries

Formulas in entries must always be *sentences* because we need them to have a *truth value* in a given model (independently of a variable assignment). This is not a substantial restriction; if we want to prove that a formula φ holds in a theory T, we can first replace the formula φ and all the axioms of T with their *general closures* (i.e., universally quantify all free variables). This yields a *closed* theory T' and a sentence φ', and it holds that $T' \models \varphi'$ if and only if $T \models \varphi$.

Quantifiers

The reduction of entries works the same way, using the same atomic tableaux for logical connectives (see Table 4.1, where instead of propositions, φ, ψ are sentences). However, we must add four new atomic tableaux for T/F and universal/existential quantifiers. These entries can be divided into two types:

- Type "*witness*": entries of the form $\mathrm{T}(\exists x)\varphi(x)$ and $\mathrm{F}(\forall x)\varphi(x)$

[1] At this point, it is helpful to revisit the tableau method in propositional logic, see Chapter 4.

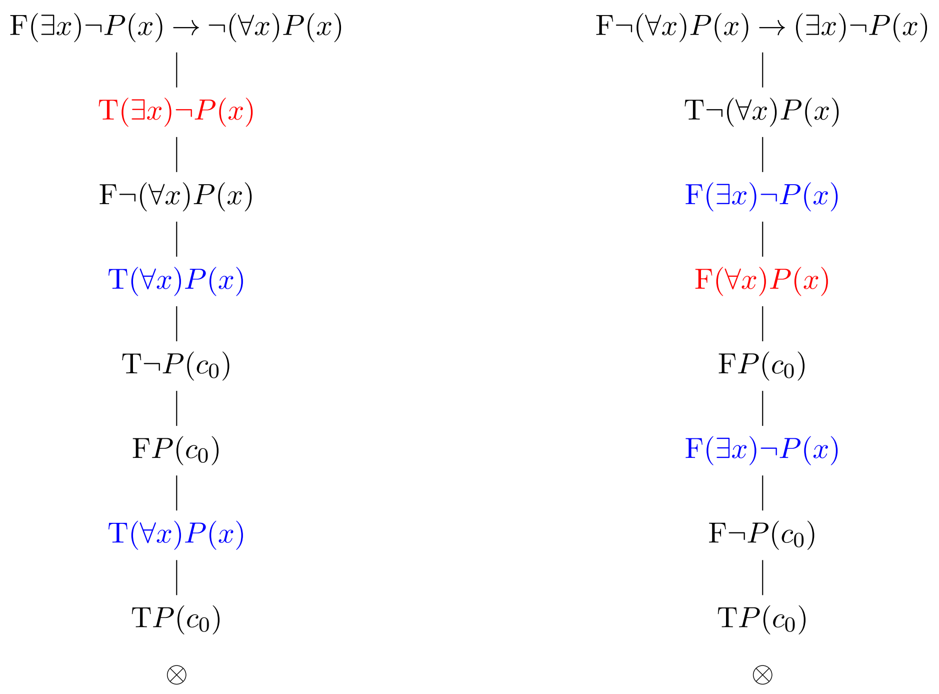

$\mathrm{F}(\exists x)\neg P(x) \to \neg(\forall x)P(x)$
$\mathrm{T}(\exists x)\neg P(x)$
$\mathrm{F}\neg(\forall x)P(x)$
$\mathrm{T}(\forall x)P(x)$
$\mathrm{T}\neg P(c_0)$
$\mathrm{F}P(c_0)$
$\mathrm{T}(\forall x)P(x)$
$\mathrm{T}P(c_0)$
$\otimes$
$\mathrm{F}\neg(\forall x)P(x) \to (\exists x)\neg P(x)$
$\mathrm{T}\neg(\forall x)P(x)$
$\mathrm{F}(\exists x)\neg P(x)$
$\mathrm{F}(\forall x)P(x)$
$\mathrm{F}P(c_0)$
$\mathrm{F}(\exists x)\neg P(x)$
$\mathrm{F}\neg P(c_0)$
$\mathrm{T}P(c_0)$
$\otimes$

Figure 7.1: Example tableaux. Entries of the 'witness' type are shown in red, entries of the 'everyone' type in blue.

- Type "*everyone*": entries of the form $\mathrm{T}(\forall x)\varphi(x)$ and $\mathrm{F}(\exists x)\varphi(x)$

Examples can be seen in the tableaux in Figure 7.1.1 ('witnesses' are in red, 'everyone' in blue).

We cannot simply remove the quantifier because the resulting formula $\varphi(x)$ would not be a sentence. Instead, simultaneously with removing the quantifier, we *substitute* a *ground term* for x, resulting in the new entry being the *sentence* $\varphi(x/t)$. The ground term t we substitute depends on whether it is a "witness" or "everyone" type entry.

Auxiliary Constant Symbols

The language L of the theory T in which we are proving is extended with a countable number of *new (auxiliary) constant symbols* $C = \{c_0, c_1, c_2, \dots\}$ (but we will also write $c, d, \dots$), and the resulting extended language is denoted L_C. Thus, ground terms in the language L_C exist even if the original language L has no constants. When constructing the tableau, we always have some *new*, previously *unused* (neither in the theory nor in the constructed tableau) auxiliary constant symbol $c \in C$ available.

Witnesses

When reducing a "witness" type entry, we substitute for the variable one of these new, auxiliary symbols that *has not yet been used on the given branch*. For an entry $\mathrm{T}(\exists x)\varphi(x)$, we thus get $\mathrm{T}\varphi(x/c)$. This constant symbol c will represent (some) element that satisfies the formula (or refutes it, in the case of an entry of the form $\mathrm{F}(\forall x)\varphi(x)$). Compare this with the Theorem on Constants (Theorem 6.6.15). It is important that the symbol c has not yet been used on the branch or in the theory. Typically, we then use "everyone" type entries to learn what must *hold about this witness.*

In Figure 7.1.1, we see an example: the entry $\mathrm{T}(\exists x)\neg P(x)$ in the left tableau is reduced, with its reduction resulting in the entry $\mathrm{T}\neg P(c_0)$; $c_0 \in C$ is an auxiliary symbol that has not yet appeared on the branch (and is the first such). Similarly for the entry $\mathrm{F}(\forall x)P(x)$ and $\mathrm{F}P(c_0)$ in the right tableau.

Everyone

When reducing an "everyone" type entry, we substitute for the variable x any *ground term* t of the extended language L_C. From an entry of the form $\mathrm{T}(\forall x)\varphi(x)$, we thus get the entry $\mathrm{T}\varphi(x/t)$.

However, for a branch to be *finished*, it must contain entries $\mathrm{T}\varphi(x/t)$ for *all* ground L_C-terms t. (We must 'use' everything the entry $\mathrm{T}(\forall x)\varphi(x)$ 'says'.) The same applies for an entry of the form $\mathrm{F}(\exists x)\varphi(x)$.

In propositional logic, we used the convention of omitting the roots of atomic tableaux when appending them (otherwise, we would repeat the same entry twice on the branch). In predicate logic, we will use the same convention but *with the exception of 'everyone' type entries.* For type 'everyone', we will write the root of the appended atomic tableau as well. Why do we do this? To remind ourselves that we are not yet done with this entry and that we must append atomic tableaux with other ground terms.

In Figure 7.1.1, the entry $\mathrm{T}(\forall x)P(x)$ in the left tableau is *not reduced.* Its *first occurrence* (4th node from the top) has been reduced, substituting the term $t = c_0$, resulting in $\varphi(x/t) = P(c_0)$. We have appended an atomic tableau consisting of the same entry at the root $\mathrm{T}(\forall x)P(x)$, which we *do write* in the tableau, and the entry $\mathrm{T}P(c_0)$ below it. While the *first occurrence* of the entry $\mathrm{T}(\forall x)P(x)$ is thus reduced, the *second occurrence* (7th node from the top) is not. Similarly for the entry $\mathrm{F}(\exists x)\neg P(x)$ in the right tableau.

This somewhat technical approach to defining the *reducedness* of (occurrences of) 'everyone' type entries will be useful in defining a *systematic tableau.*

Language

We will assume that the language L is *countable.*[2] As a consequence, any L-theory T has only countably many axioms, and there are also only countably many ground terms in the language L_C. This restriction is necessary because every tableau, even an infinite one, has only countably many entries, and we must be able to use all the axioms of the given theory and substitute all the ground terms of the language L_C.

Initially, we will also assume that the language is *without equality*, which is the simpler case. The issue with equality is that *the tableau* is a purely syntactic object, but *equality* has a special semantic meaning: it must be interpreted as the identity relation in every model. How to adapt the method for languages with equality will be discussed later.

7.2 Formal Definitions

In this section, we define all the notions needed for the tableau method for languages without equality. We will return to languages with equality in Section 7.3.

[2]This is not a significant limitation from the perspective of computational logic.

Let L be a *countable* language without equality. Denote by L_C the extension of the language L with countably many new *auxiliary* constant symbols $C = \{c_i \mid i \in \mathbb{N}\}$. Let us choose some numbering of the ground terms of the language L_C, denoted by $\{t_i \mid i \in \mathbb{N}\}$.

Let T be some L-theory and φ an L-sentence.

7.2.1 Atomic Tableaux

An *entry* is a label $\mathrm{T}\varphi$ or $\mathrm{F}\varphi$, where φ is some L_C-sentence. Entries of the form $\mathrm{T}(\exists x)\varphi(x)$ and $\mathrm{F}(\forall x)\varphi(x)$ are of the *'witness' type*, while entries of the form $\mathrm{T}(\forall x)\varphi(x)$ and $\mathrm{F}(\exists x)\varphi(x)$ are of the *'everyone' type*.

Atomic tableaux are labeled trees shown in Tables 7.1 and 7.2.

	$\neg$	$\wedge$	$\vee$	$\rightarrow$	$\leftrightarrow$
True	$\begin{array}{c} \mathrm{T}\neg\varphi \\ \vert \\ \mathrm{F}\varphi \end{array}$	$\begin{array}{c} \mathrm{T}\varphi\wedge\psi \\ \vert \\ \mathrm{T}\varphi \\ \vert \\ \mathrm{T}\psi \end{array}$	$\begin{array}{cc} \multicolumn{2}{c}{\mathrm{T}\varphi\vee\psi} \\ / & \backslash \\ \mathrm{T}\varphi & \mathrm{T}\psi \end{array}$	$\begin{array}{cc} \multicolumn{2}{c}{\mathrm{T}\varphi\rightarrow\psi} \\ / & \backslash \\ \mathrm{F}\varphi & \mathrm{T}\psi \end{array}$	$\begin{array}{cc} \multicolumn{2}{c}{\mathrm{T}\varphi\leftrightarrow\psi} \\ / & \backslash \\ \mathrm{T}\varphi & \mathrm{F}\varphi \\ \vert & \vert \\ \mathrm{T}\psi & \mathrm{F}\psi \end{array}$
False	$\begin{array}{c} \mathrm{F}\neg\varphi \\ \vert \\ \mathrm{T}\varphi \end{array}$	$\begin{array}{cc} \multicolumn{2}{c}{\mathrm{F}\varphi\wedge\psi} \\ / & \backslash \\ \mathrm{F}\varphi & \mathrm{F}\psi \end{array}$	$\begin{array}{c} \mathrm{F}\varphi\vee\psi \\ \vert \\ \mathrm{F}\varphi \\ \vert \\ \mathrm{F}\psi \end{array}$	$\begin{array}{c} \mathrm{F}\varphi\rightarrow\psi \\ \vert \\ \mathrm{T}\varphi \\ \vert \\ \mathrm{F}\psi \end{array}$	$\begin{array}{cc} \multicolumn{2}{c}{\mathrm{F}\varphi\leftrightarrow\psi} \\ / & \backslash \\ \mathrm{T}\varphi & \mathrm{F}\varphi \\ \vert & \vert \\ \mathrm{F}\psi & \mathrm{T}\psi \end{array}$

Table 7.1: Atomic tableaux for logical connectives; φ and ψ are any L_C-sentences.

	$\forall$	$\exists$
True	$\begin{array}{c} \mathrm{T}(\forall x)\varphi(x) \\ \vert \\ \mathrm{T}\varphi(x/t_i) \end{array}$	$\begin{array}{c} \mathrm{T}(\exists x)\varphi(x) \\ \vert \\ \mathrm{T}\varphi(x/c_i) \end{array}$
False	$\begin{array}{c} \mathrm{F}(\forall x)\varphi(x) \\ \vert \\ \mathrm{F}\varphi(x/c_i) \end{array}$	$\begin{array}{c} \mathrm{F}(\exists x)\varphi(x) \\ \vert \\ \mathrm{F}\varphi(x/t_i) \end{array}$

Table 7.2: Atomic tableaux for quantifiers; φ is an L_C-sentence, x a variable, t_i any ground L_C-term, $c_i \in C$ is a new auxiliary constant symbol (not yet occurring on the given branch of the constructed tableau).

7.2.2 Tableau Proof

The definitions in this section are almost identical to the corresponding definitions from propositional logic. The main technical issue is how to define the reducedness of 'everyone' type entries on a branch of the tableau: we want *all* ground L_C-terms t_i to be substituted for the variable.

Definition 7.2.1 (Tableau)**.** A *finite tableau from theory T* is an ordered, labeled tree constructed by applying finitely many of the following rules:

- A single-node tree labeled with any entry is a tableau from theory T,
- For any entry E on any branch B, we can append an atomic tableau for the entry E to the end of branch B, provided that if E is of 'witness' type, we use only an auxiliary constant symbol $c_i \in C$ not yet appearing on branch B (for 'everyone' type entries, we can use any ground L_C-term t_i),
- We can append the entry $\mathrm{T}\alpha$ for any axiom $\alpha \in T$ to the end of any branch.

A *tableau from theory T* can be either finite or *infinite*: in that case it is constructed in countably many steps. It can be formally expressed as the union $\tau = \bigcup_{i\geq 0} \tau_i$, where τ_i are finite tableaux from T, τ_0 is a single-node tableau, and τ_{i+1} is constructed from τ_i in one step.[3]

A tableau *for entry E* is a tableau with the entry E at its root.

Recall the convention that if E is *not* of the 'everyone' type, we omit the root of the atomic tableau (since the node with entry E is already in the tableau).

Exercise 7.1. Show step-by-step how the tableaux from Figure 7.1.1 were constructed.

Definition 7.2.2 (Tableau Proof)**.** A *tableau proof* of a sentence φ from a theory T is a *contradictory* tableau from theory T with the entry $\mathrm{F}\varphi$ at the root. If such a proof exists, then φ is *(tableau) provable* from T, denoted by $T \vdash \varphi$. (We also define a *tableau refutation* as a contradictory tableau with $\mathrm{T}\varphi$ at the root. If such a proof exists, then φ is *(tableau) refutable* from T, i.e., $T \vdash \neg\varphi$ holds.)

- A tableau is *contradictory* if each of its branches is contradictory.
- A branch is *contradictory* if it contains entries $\mathrm{T}\psi$ and $\mathrm{F}\psi$ for some sentence ψ, otherwise it is *non-contradictory.*
- A tableau is *finished* if each of its branches is finished.
- A branch is *finished* if
  - it is contradictory, or
  - each entry on this branch is *reduced* and the branch contains the entry $\mathrm{T}\alpha$ for each axiom $\alpha \in T$.
- An entry E is *reduced* on a branch B passing through this entry if

[3]Union because in each step, we add new nodes to the tableau, so τ_i is a subtree of τ_{i+1}.

  - it is of the form $\mathrm{T}\psi$ or $\mathrm{F}\psi$ for an *atomic sentence* ψ (i.e., $R(t_1, \ldots, t_n)$, where t_i are *ground* L_C-terms), or
  - it is not of the 'everyone' type and appears on B as the root of an atomic tableau[4] (i.e., typically, the entry has already been 'developed' on B during the tableau construction), or
  - it is of the 'everyone' type and all its *occurrences* on B are reduced on the branch B.

- An occurrence of an 'everyone' type entry E on branch B is *i-th* if it has exactly $i-1$ predecessors labeled with this entry on B, and the i-th occurrence is *reduced* on B if
  - the entry E has the $(i+1)$-th occurrence on B, and at the same time
  - the branch B contains the entry $\mathrm{T}\varphi(x/t_i)$ (if $E = \mathrm{T}(\forall x)\varphi(x)$) or $\mathrm{F}\varphi(x/t_i)$ (if $E = \mathrm{F}(\exists x)\varphi(x)$), where t_i is the i-th ground L_C-term.[5]

Note that if an 'everyone' type entry is reduced on some branch, it must have infinitely many occurrences on that branch, and we must have used all possibilities, i.e., all ground L_C-terms, in the substitutions.

Example 7.2.3. As an example, let us construct tableau proofs *in logic* (from the empty theory) of the following sentences:

(a) $(\forall x)(P(x) \to Q(x)) \to ((\forall x)P(x) \to (\forall x)Q(x))$, where P, Q are unary relation symbols.

(b) $(\forall x)(\varphi(x) \land \psi(x)) \leftrightarrow ((\forall x)\varphi(x) \land (\forall x)\psi(x))$, where $\varphi(x), \psi(x)$ are arbitrary formulas with a single free variable x.

The resulting tableaux are in Figures 7.2 and 7.3. The pairs of contradictory entries are shown in red. Consider how the tableaux were constructed step-by-step.

7.2.3 Systematic Tableau and Finiteness of Proofs

In Section 4.4, we showed that if we do not extend contradictory branches (and there is no reason to do that), then a contradictory tableau, in particular a tableau proof, will always be finite. The same argument applies to predicate logic.

Corollary 7.2.4 (Finiteness of Proofs). *If $T \vdash \varphi$, then there also exists a* finite *tableau proof of φ from T.*

Proof. The same as in propositional logic, see the proof of Corollary 4.4.5. □

In the same section, we also showed the construction of the *systematic tableau.* This too can be easily adapted to predicate logic. We must ensure that we eventually reduce each entry, use every axiom, and now, in predicate logic, also substitute every L_C-term t_i for the variable in entries of type 'everyone'.

[4]Even though by convention we do not write this root.

[5]I.e., (typically) we have already substituted the term t_i for x.

$$\begin{tikzcd}[column sep=small]
 & \mathrm{F}(\forall x)(P(x) \to Q(x)) \to ((\forall x)P(x) \to (\forall x)Q(x)) \arrow[d, dash] & \\
 & \mathrm{T}(\forall x)(P(x) \to Q(x)) \arrow[d, dash] & \\
 & \mathrm{F}(\forall x)P(x) \to (\forall x)Q(x) \arrow[d, dash] & \\
 & \mathrm{T}(\forall x)P(x) \arrow[d, dash] & \\
 & \mathrm{F}(\forall x)Q(x) \arrow[d, dash] & \\
 & \mathrm{F}Q(c_0) \arrow[d, dash] & \\
 & \mathrm{T}(\forall x)P(x) \arrow[d, dash] & \\
 & \mathrm{T}P(c_0) \arrow[d, dash] & \\
 & \mathrm{T}(\forall x)(P(x) \to Q(x)) \arrow[d, dash] & \\
 & \mathrm{T}P(c_0) \to Q(c_0) \arrow[dl, dash] \arrow[dr, dash] & \\
\mathrm{F}P(c_0) & & \mathrm{T}Q(c_0) \\
\otimes & & \otimes
\end{tikzcd}$$

Figure 7.2: Tableau proof from Example 7.2.3 (a).

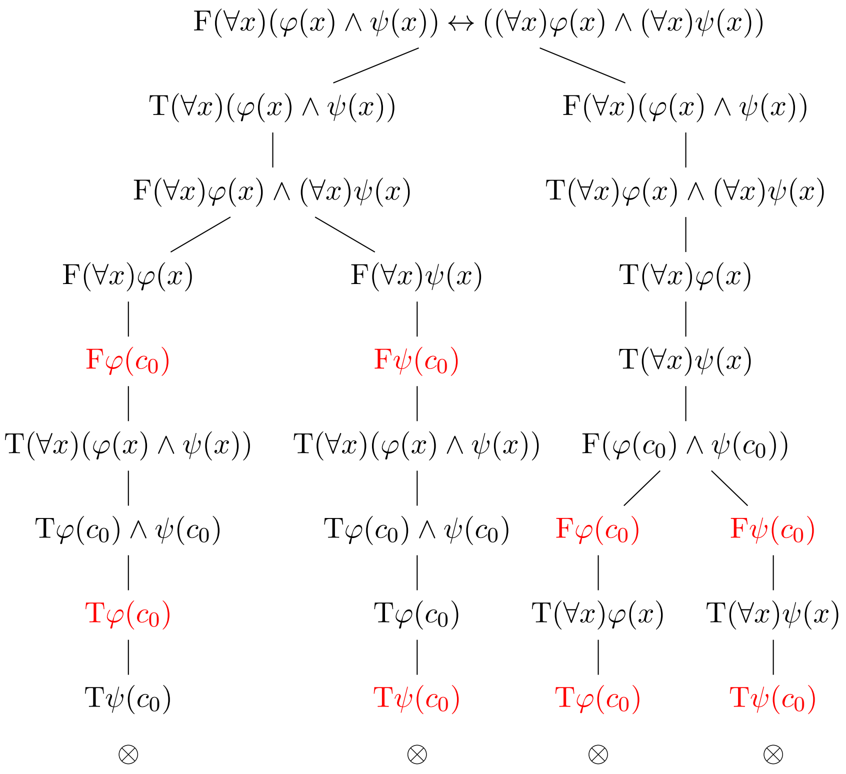

$\mathrm{F}(\forall x)(\varphi(x) \wedge \psi(x)) \leftrightarrow ((\forall x)\varphi(x) \wedge (\forall x)\psi(x))$
$\mathrm{T}(\forall x)(\varphi(x) \wedge \psi(x))$
$\mathrm{F}(\forall x)(\varphi(x) \wedge \psi(x))$
$\mathrm{F}(\forall x)\varphi(x) \wedge (\forall x)\psi(x)$
$\mathrm{T}(\forall x)\varphi(x) \wedge (\forall x)\psi(x)$
$\mathrm{F}(\forall x)\varphi(x)$
$\mathrm{F}(\forall x)\psi(x)$
$\mathrm{T}(\forall x)\varphi(x)$
$\mathrm{F}\varphi(c_0)$
$\mathrm{F}\psi(c_0)$
$\mathrm{T}(\forall x)\psi(x)$
$\mathrm{T}(\forall x)(\varphi(x) \wedge \psi(x))$
$\mathrm{T}(\forall x)(\varphi(x) \wedge \psi(x))$
$\mathrm{F}(\varphi(c_0) \wedge \psi(c_0))$
$\mathrm{T}\varphi(c_0) \wedge \psi(c_0)$
$\mathrm{T}\varphi(c_0) \wedge \psi(c_0)$
$\mathrm{F}\varphi(c_0)$
$\mathrm{F}\psi(c_0)$
$\mathrm{T}\varphi(c_0)$
$\mathrm{T}\varphi(c_0)$
$\mathrm{T}(\forall x)\varphi(x)$
$\mathrm{T}(\forall x)\psi(x)$
$\mathrm{T}\psi(c_0)$
$\mathrm{T}\psi(c_0)$
$\mathrm{T}\varphi(c_0)$
$\mathrm{T}\psi(c_0)$
$\otimes$
$\otimes$
$\otimes$
$\otimes$

Figure 7.3: Tableau proof from Example 7.2.3 (b). The constant c_0 can be used as *new* in all three cases. It suffices that it does not yet occur *on the given branch.*

Definition 7.2.5. Let R be an entry and $T = \{\alpha_0, \alpha_1, \alpha_2, \dots\}$ a theory. The *systematic tableau* from theory T for entry R is the tableau $\tau = \bigcup_{i\geq 0} \tau_i$, where τ_0 is a single-node tableau with entry R, and for each $i \geq 0$:

Let E be an entry at the leftmost node v at the smallest level of the tableau τ_i that is not reduced on some non-contradictory branch passing through E (or, if the entry is of the 'everyone' type, its *occurrence* in this node is not reduced). Then τ_i' is the tableau constructed from τ_i by appending an atomic tableau for E to each non-contradictory branch passing through v, where

- If E is of the type 'everyone' and has its k-th occurrence in the node v, then we substitute the k-th L_C-term t_k for the variable.
- If E is of the 'witness' type, then on the given branch B, we substitute $c_i \in C$ with the smallest possible i (such that c_i does not yet occur on B).

Otherwise, if such an entry E and node v do not exist, i.e., all entries are reduced, we define $\tau_i' = \tau_i$.

The tableau τ_{i+1} is then the tableau constructed from τ_i' by appending $\mathrm{T}\alpha_i$ to each non-contradictory branch of τ_i', if $i \leq |T|$. Otherwise (if T is finite and we have used all the axioms), we skip this step and define $\tau_{i+1} = \tau_i'$.

Just as in propositional logic, it holds that the systematic tableau is always finished and provides a finite proof:

Lemma 7.2.6. *The systematic tableau is finished.*

Proof. Similar to the proof in propositional logic (Lemma 4.4.2). For 'everyone' type entries, note that we reduce the k-th occurrence when we encounter it in the construction: by appending a node with the $(k+1)$-th occurrence and substituting the k-th L_C-term t_k. □

Corollary 7.2.7 (Systematicity of Proofs)**.** *If $T \vdash \varphi$, then the systematic tableau is (a finite) tableau proof of φ from T.*

Proof. The same as the proof in propositional logic (Corollary 4.4.6). □

7.3 Languages with Equality

Now we will show how to apply the tableau method to languages with equality. What is equality? In mathematics, it can mean different relations in different contexts. Does $1 + 0 = 0 + 1$ hold? If we are talking about integers, then yes, but if we mean arithmetic expressions (or e.g., terms in the language of fields), then the left and right sides are not equal: they are different expressions. [6]

Imagine we have a theory T in a language with equality containing constant symbols c_1, c_2, a unary function symbol f, and a unary relation symbol P. Let us have some finished tableau from this theory, and in it a non-contradictory branch, on which we find the entry $\mathrm{T}c_1 = c_2$. We aim to construct a *canonical model* $\mathcal{A}$ for this branch, similar to propositional logic. The entry will mean that in the canonical model we have $c_1^{\mathcal{A}} =^{\mathcal{A}} c_2^{\mathcal{A}}$, i.e., $(c_1^{\mathcal{A}}, c_2^{\mathcal{A}}) \in =^{\mathcal{A}}$. But this is not enough, we also want, for example:

[6] Similarly, $t_1 = t_2$ in Prolog does not mean they are the same term but that the terms t_1 and t_2 are *unifiable*, see the next chapter, Section 8.4.

- $c_2^{\mathcal{A}} =^{\mathcal{A}} c_1^{\mathcal{A}}$,
- $f^{\mathcal{A}}(c_1^{\mathcal{A}}) =^{\mathcal{A}} f^{\mathcal{A}}(c_2^{\mathcal{A}})$,
- $c_1^{\mathcal{A}} \in P^{\mathcal{A}}$ if and only if $c_2^{\mathcal{A}} \in P^{\mathcal{A}}$.

In general, we want the relation $=^{\mathcal{A}}$ to be a so-called *congruence*,[7] i.e., an equivalence that behaves 'well' with respect to the functions and relations of the structure $\mathcal{A}$. We achieve this by adding so-called *axioms of equality* to the theory T, which enforce these properties, and construct the tableau from the resulting theory T^*.

In the model $\mathcal{A}$, the relation $=^{\mathcal{A}}$ will then be a congruence. But this is not enough for us; we want equality to be *identity*, i.e., $(a, b) \in =^{\mathcal{A}}$ only if a and b are the same element of the domain. We achieve this by identifying all $=^{\mathcal{A}}$-equivalent elements into a single element. This construction is called a *quotient (structure)* by the congruence $=^{\mathcal{A}}$.[8] Now we formalize these concepts.

Definition 7.3.1 (Congruence)**.** Let $\sim$ be an equivalence on the set A, $f\colon A^n \to A$ be a function, and $R \subseteq A^n$ be a relation. We say that $\sim$ is

- *a congruence for the function* f if for all $a_i, b_i \in A$ such that $a_i \sim b_i$ $(1 \le i \le n)$ it holds that $f(a_1, \dots, a_n) \sim f(b_1, \dots, b_n)$,
- *a congruence for the relation* R if for all $a_i, b_i \in A$ such that $a_i \sim b_i$ $(1 \le i \le n)$ it holds that $(a_1, \dots, a_n) \in R$ if and only if $(b_1, \dots, b_n) \in R$.

A congruence of the structure $\mathcal{A}$ is an equivalence $\sim$ on the set A that is a congruence for all functions and relations of $\mathcal{A}$.

Definition 7.3.2 (Quotient Structure)**.** Let $\mathcal{A}$ be a structure and $\sim$ be its congruence. The *quotient structure* of $\mathcal{A}$ by $\sim$ is the structure $\mathcal{A}/_\sim$ in the same language, whose universe $A/_\sim$ is the set of all equivalence classes of A by $\sim$, and whose functions and relations are defined *using representatives*, i.e.:

- $f^{\mathcal{A}/\sim}([a_1]_\sim, \dots, [a_n]_\sim) = [f^{\mathcal{A}}(a_1, \dots, a_n)]_\sim$, for each ($n$-ary) function symbol f, and
- $R^{\mathcal{A}/\sim}([a_1]_\sim, \dots, [a_n]_\sim)$ if and only if $R^{\mathcal{A}}(a_1, \dots, a_n)$, for each ($n$-ary) relation symbol R.

Definition 7.3.3 (Axioms of Equality)**.** The *axioms of equality* for a language L with equality are as follows:

(i) $x = x$,

(ii) $x_1 = y_1 \wedge \dots \wedge x_n = y_n \to f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$ for every n-ary function symbol f of the language L,

(iii) $x_1 = y_1 \wedge \dots \wedge x_n = y_n \to (R(x_1, \dots, x_n) \to R(y_1, \dots, y_n))$ for every n-ary relation symbol R of the language L *including the equality symbol.*

[7] The name comes from *congruence modulo n*, which is a congruence in this sense on the set of all integers, e.g., it satisfies: $a + b \equiv c + d \pmod{n}$ whenever $a \equiv c \pmod{n}$ and $b \equiv d \pmod{n}$.

[8] Just as the group $\mathbb{Z}_n$ is a quotient structure of the group $\mathbb{Z}$ over $\equiv \pmod{n}$; for example, the element $2 \in \mathbb{Z}_n$ represents the set of all integers whose remainder after division by n is 2.

Exercise 7.2. The first of the axioms of equality expresses the reflexivity of the relation $=^{\mathcal{A}}$. What about symmetry and transitivity? Show that they follow from axiom (iii) for the equality symbol $=$.

Thus, from axioms (*i*) and (*iii*) it follows that the relation $=^{\mathcal{A}}$ is an equivalence on A, and axioms (*ii*) and (*iii*) express that $=^{\mathcal{A}}$ is a congruence of $\mathcal{A}$. In the tableau method for a language with equality, we implicitly add all the axioms of equality:

Definition 7.3.4 (Tableau Proof with Equality)**.** If T is a theory in a language L with equality, then denote as T^* the extension of the theory T by the general closures[9] of the axioms of equality for the language L. A *tableau proof* from the theory T is a *tableau proof* from T^*, similarly for tableau refutation (and generally any tableau).

The following simple observation holds:

Observation 7.3.5. *If $\mathcal{A} \models T^*$, then it holds that $\mathcal{A}/_{=^{\mathcal{A}}} \models T^*$, and in the structure $\mathcal{A}/_{=^{\mathcal{A}}}$ the equality symbol is interpreted as identity. On the other hand, in any model where the equality symbol is interpreted as identity, the axioms of equality hold.*

We will use this observation in the construction of the *canonical model*, which we will need to prove the Completeness Theorem. But first, we will prove the Soundness Theorem.

7.4 Soundness and Completeness

In this section, we will prove that the tableau method is sound and complete in predicate logic as well. The proofs of both theorems have the same structure as in propositional logic, differing only in the implementation details.

7.4.1 Soundness Theorem

A model (structure) $\mathcal{A}$ *agrees* with an entry E if $E = \mathrm{T}\varphi$ and $\mathcal{A} \models \varphi$, or $E = \mathrm{F}\varphi$ and $\mathcal{A} \not\models \varphi$. Further, $\mathcal{A}$ agrees with a branch B if it agrees with every entry on that branch.

First, we show an auxiliary lemma analogous to Lemma 4.5.1:

Lemma 7.4.1. *If a model $\mathcal{A}$ of the theory T agrees with the entry at the root of a tableau from the theory T (in the language L), then $\mathcal{A}$ can be expanded to the language L_C such that it agrees with some branch in the tableau.*

Note that it suffices to expand $\mathcal{A}$ by those new constants $c^{\mathcal{A}}$ where c appears on the branch B. Other constant symbols can be interpreted arbitrarily.

Proof. Let $\tau = \bigcup_{i\geq 0} \tau_i$ be a tableau from the theory T and $\mathcal{A} \in \mathrm{M}_L(T)$ a model agreeing with the root of τ, i.e., with the (single-element) branch B_0 in the (single-element) τ_0.

By induction on i, we find a sequence of branches B_i and expansions $\mathcal{A}_i$ of the model $\mathcal{A}$ by constants $c^{\mathcal{A}} \in C$ appearing on B_i such that B_i is a branch in the tableau τ_i agreeing with the model $\mathcal{A}_i$, B_{i+1} is an extension of B_i, and $\mathcal{A}_{i+1}$ is an expansion of $\mathcal{A}_i$ (they might be the same structure). The desired branch of the tableau τ is then $B = \bigcup_{i\geq 0} B_i$. The expansion of the model $\mathcal{A}$ to the language L_C is obtained as the 'limit' of the expansions $\mathcal{A}_i$, i.e., if a symbol $c \in C$ appears on B, it appears on some branch B_i and we interpret it the same as in $\mathcal{A}_i$ (other auxiliary symbols are interpreted arbitrarily).

[9]Because for the tableau method we need to have *sentences*.

- If τ_{i+1} was created from τ_i without extending the branch B_i, we define $B_{i+1} = B_i$ and $\mathcal{A}_{i+1} = \mathcal{A}_i$.
- If τ_{i+1} was created from τ_i by adding the entry $\mathrm{T}\alpha$ (for some axiom $\alpha \in T$) to the end of the branch B_i, we define B_{i+1} as this extended branch and $\mathcal{A}_{i+1} = \mathcal{A}_i$ (we did not add any new auxiliary constant symbol). Since $\mathcal{A}_{i+1}$ is a model of T, the axiom α holds in it, so it agrees with the new entry $\mathrm{T}\alpha$.
- Suppose τ_{i+1} was created from τ_i by appending an atomic tableau for some entry E to the end of the branch B_i. Since the model $\mathcal{A}_i$ agrees with the entry E (which lies on the branch B_i), it also agrees with the root of the appended atomic tableau.
  - If we appended an atomic tableau for a logical connective, we set $\mathcal{A}_{i+1} = \mathcal{A}_i$ (we did not add a new auxiliary symbol). Since $\mathcal{A}_{i+1}$ agrees with the root of the atomic tableau, it also agrees with one of its branches (just as in propositional logic); we define B_{i+1} as the extension of B_i by this branch.
  - If the entry E is of the type 'witness': If $E = \mathrm{T}(\exists x)\varphi(x)$, then $\mathcal{A}_i \models (\exists x)\varphi(x)$, so there exists $a \in A$ such that $\mathcal{A}_i \models \varphi(x)[e(x/a)]$. We define the branch B_{i+1} as the extension of B_i by the newly added entry $\mathrm{T}\varphi(x/c)$ and the model $\mathcal{A}_{i+1}$ as the expansion of $\mathcal{A}_i$ by the constant $c^A = a$. The case $E = \mathrm{F}(\forall x)\varphi(x)$ is analogous.
  - If the entry E is of the type 'everyone', we define the branch B_{i+1} as the extension of B_i by the atomic tableau. The newly added entry is $\mathrm{T}\varphi(x/t)$ or $\mathrm{F}\varphi(x/t)$ for some L_C-term t. Suppose it is the first of these two possibilities, for the second the proof is analogous. We define the model $\mathcal{A}_{i+1}$ as *any* expansion of $\mathcal{A}_i$ by the new constants appearing in t. Since $\mathcal{A}_i \models (\forall x)\varphi(x)$, it also holds $\mathcal{A}_{i+1} \models (\forall x)\varphi(x)$ and thus $\mathcal{A}_{i+1} \models \varphi(x/t)$; the model $\mathcal{A}_{i+1}$ thus agrees with the branch B_i.

□

Let us briefly recall the idea of the proof of the Soundness Theorem: If there existed a proof and at the same time a counterexample, the counterexample would have to agree with some branch of the proof, but they are all contradictory. The proof is thus almost the same as in propositional logic.

Theorem 7.4.2 (On Soundness)**.** *If a sentence φ is tableau provable from a theory T, then φ is valid in T, i.e., $T \vdash \varphi \;\Rightarrow\; T \models \varphi$.*

Proof. Assume for contradiction that $T \not\models \varphi$, i.e., there exists $\mathcal{A} \in \mathrm{M}(T)$ such that $\mathcal{A} \not\models \varphi$. Since $T \vdash \varphi$, there exists a contradictory tableau from T with $\mathrm{F}\varphi$ at the root. The model $\mathcal{A}$ agrees with $\mathrm{F}\varphi$, so by Lemma 7.4.1 it can be expanded to the language L_C such that the expansion agrees with some branch B. But all branches are contradictory. □

7.4.2 Completeness Theorem

Same as in propositional logic, we will show that a *non-contradictory* branch in a *finished* tableau proof provides a counterexample: a model of the theory T that agrees with the entry $\mathrm{F}\varphi$ at the root of the tableau, i.e., φ does not hold in it. There can be more such models, so we again define one specific, *canonical* model.

The model must have some domain. How do we obtain it from the tableau, which is a purely syntactic object? We use a standard trick in mathematics: we turn syntactic objects into semantic ones. In particular, we choose the set of all *ground terms* of the language L_C as the domain.[10] We understand these as finite strings. In the following exposition, we will sometimes (informally) write "t" instead of the term t to emphasize that at that place we view t as a string of characters, not e.g., as a term function to be evaluated.[11]

Definition 7.4.3 (Canonical Model)**.** Let T be a theory in a language $L = \langle \mathcal{F}, \mathcal{R} \rangle$ and let B be a non-contradictory branch of some finished tableau from the theory T. Then the *canonical model* for B is the L_C-structure $\mathcal{A} = \langle A, \mathcal{F}^{\mathcal{A}} \cup C^{\mathcal{A}}, \mathcal{R}^{\mathcal{A}} \rangle$ defined as follows:

If the language L is without equality, then:

- The domain A is the set of all ground L_C-terms.
- For each n-ary relation symbol $R \in \mathcal{R}$ and "s_1", ..., "s_n" from A:

$$(\text{“}s_1\text{”}, \ldots, \text{“}s_n\text{”}) \in R^{\mathcal{A}} \text{ if and only if the branch } B \text{ contains the entry } \mathrm{T}R(s_1, \ldots, s_n)$$

- For each n-ary function symbol $f \in \mathcal{F}$ and "s_1", ..., "s_n" from A:

$$f^{\mathcal{A}}(\text{“}s_1\text{”}, \ldots, \text{“}s_n\text{”}) = \text{“}f(s_1, \ldots, s_n)\text{”}$$

  In particular, for a constant symbol c, we have $c^{\mathcal{A}} = \text{“}c\text{”}$.

Thus, we interpret the function $f^{\mathcal{A}}$ as the 'creation' of a new term from the symbol f and the input terms (strings).

Let L be a language with equality. Recall that our tableau is now from the theory T^*, i.e., from the extension of T by the axioms of equality for L. First, we create the canonical model $\mathcal{D}$ for the branch B as if L were without equality (its domain D is thus the set of all ground L_C-terms). Then we define the relation $=^D$ just like for other relation symbols:

$$\text{“}s_1\text{”} =^D \text{“}s_2\text{”} \text{ if and only if the branch } B \text{ contains the entry } \mathrm{T}s_1 = s_2$$

The *canonical model* for B is then defined as the quotient structure $\mathcal{A} = \mathcal{D}/_{=^D}$.

As follows from the discussion in Section 7.3, the relation $=^D$ is indeed a congruence of the structure $\mathcal{D}$, so the definition is sound, and the relation $=^{\mathcal{A}}$ is the identity on A. The following simple observation holds:

Observation 7.4.4. *For any formula φ we have $\mathcal{D} \models \varphi$ (where the symbol $=$ is interpreted as the binary relation $=^D$) if and only if $\mathcal{A} = \mathcal{D}/_{=^D} \models \varphi$ (where $=$ is interpreted as identity).*

Note that in a language without equality, the canonical model is always countably infinite. In a language with equality, it can be finite, as we will see in the following examples.

[10] That is, terms constructed by applying the function symbols of the language L to the constant symbols of the language L (if it has any) and auxiliary constant symbols from C.

[11] Compare the arithmetic expression "1+1" and 1+1=2.

Example 7.4.5. First, let us show an example of a canonical model in a language without equality. Let $T = \{(\forall x)R(f(x))\}$ be a theory in the language $L = \langle R, f, d\rangle$ without equality, where R is a unary relation, f a unary function, and d a constant symbol. Let us find a counterexample showing that $T \not\models \neg R(d)$.

The systematic tableau from T with the entry F$\neg R(d)$ at the root is not contradictory; it contains a single branch B, which is non-contradictory. (Construct the tableau yourself!) The canonical model for B is the L_C-structure $\mathcal{A} = \langle A, R^{\mathcal{A}}, f^{\mathcal{A}}, d^{\mathcal{A}}, c_0^{\mathcal{A}}, c_1^{\mathcal{A}}, c_2^{\mathcal{A}}, \dots\rangle$, its domain is

$$A = \{\text{“}d\text{”}, \text{“}f(d)\text{”}, \text{“}f(f(d))\text{”}, \dots, \text{“}c_0\text{”}, \text{“}f(c_0)\text{”}, \text{“}f(f(c_0))\text{”}, \dots, \text{“}c_1\text{”}, \text{“}f(c_1)\text{”}, \text{“}f(f(c_1))\text{”}, \dots\}$$

and the interpretations of the symbols are as follows:

- $d^{\mathcal{A}} = \text{“}d\text{”}$,
- $c_i^{\mathcal{A}} = \text{“}c_i\text{”}$ for all $i \in \mathbb{N}$,
- $f^{\mathcal{A}}(\text{“}d\text{”}) = \text{“}f(d)\text{”}$, $f^{\mathcal{A}}(\text{“}f(d)\text{”}) = \text{“}f(f(d))\text{”}$, ...
- $R^{\mathcal{A}} = A \setminus C = \{\text{“}d\text{”}, \text{“}f(d)\text{”}, \text{“}f(f(d))\text{”}, \dots, \text{“}f(c_0)\text{”}, \text{“}f(f(c_0))\text{”}, \dots, \text{“}f(c_1)\text{”}, \text{“}f(f(c_1))\text{”}, \dots\}$.

The reduct of the canonical model $\mathcal{A}$ to the original language L is then $\mathcal{A}' = \langle A, R^{\mathcal{A}}, f^{\mathcal{A}}, d^{\mathcal{A}}\rangle$.

Example 7.4.6. Now an example in a language with equality: Let $T = \{(\forall x)R(f(x)), (\forall x)(x = f(f(x)))\}$ be in the language $L = \langle R, f, d\rangle$ with equality. Again, let us find a counterexample showing that $T \not\models \neg R(d)$.

The systematic tableau from the theory T^* (i.e., from T extended by the axioms of equality for L) with the entry F$\neg R(d)$ at the root contains a non-contradictory branch B. (Construct the tableau yourself!) First, we construct the canonical model $\mathcal{D}$ for this branch as if the language were without equality:

$$\mathcal{D} = \langle D, R^{\mathcal{D}}, f^{\mathcal{D}}, d^{\mathcal{D}}, c_0^{\mathcal{D}}, c_1^{\mathcal{D}}, c_2^{\mathcal{D}}, \dots\rangle$$

where D is the set of all ground L_C-terms. The relation $=^D$ is defined as if the symbol ‘=’ were an ‘ordinary’ relation symbol in L. It is a congruence of the structure $\mathcal{D}$, and it holds that $s_1 =^D s_2$ if and only if $s_1 = f(\cdots(f(s_2))\cdots)$ or $s_2 = f(\cdots(f(s_1))\cdots)$ for an even number of applications of f. Thus, we can choose terms with zero or one occurrence of the symbol f as representatives of individual equivalence classes:

$$D/_{=^D} = \{[\text{“}d\text{”}]_{=^D}, [\text{“}f(d)\text{”}]_{=^D}, [\text{“}c_0\text{”}]_{=^D}, [\text{“}f(c_0)\text{”}]_{=^D}, [\text{“}c_1\text{”}]_{=^D}, [\text{“}f(c_1)\text{”}]_{=^D}, \dots\}$$

The canonical model for the branch B is then the L_C-structure

$$\mathcal{A} = \mathcal{D}/_{=^D} = \langle A, R^{\mathcal{A}}, f^{\mathcal{A}}, d^{\mathcal{A}}, c_0^{\mathcal{A}}, c_1^{\mathcal{A}}, c_2^{\mathcal{A}}, \dots\rangle$$

where $A = D/_{=^D}$ and the interpretations of the symbols are as follows:

- $d^{\mathcal{A}} = [\text{“}d\text{”}]_{=^D}$,
- $c_i^{\mathcal{A}} = [\text{“}c_i\text{”}]_{=^D}$ for all $i \in \mathbb{N}$,
- $f^{\mathcal{A}}([\text{“}d\text{”}]_{=^D}) = [\text{“}f(d)\text{”}]_{=^D}$, $f^{\mathcal{A}}([\text{“}f(d)\text{”}]_{=^D}) = [\text{“}f(f(d))\text{”}]_{=^D} = [\text{“}d\text{”}]_{=^D}$, ...
- $R^{\mathcal{A}} = A = D/_{=^D}$.

The reduct of the canonical model $\mathcal{A}$ to the original language L is again $\mathcal{A}' = \langle A, R^{\mathcal{A}}, f^{\mathcal{A}}, d^{\mathcal{A}} \rangle$.

Exercise 7.3. (a) Construct a finished tableau with the entry $\mathrm{T}(\forall x)(\forall y)(x = y)$ at the root. Construct the canonical model for the (only, non-contradictory) branch of this tableau.

(b) Construct a finished tableau with the entry $\mathrm{T}(\forall x)(\forall y)(\forall z)(x = y \vee x = z \vee y = z)$ at the root. Construct canonical models for several of its non-contradictory branches, and compare them.

We are now ready to prove the Completeness Theorem. We again use the following auxiliary lemma, whose wording is exactly the same as Lemma 4.5.4 and whose proof differs only in technical details.

Lemma 7.4.7. *The canonical model for a (non-contradictory, finished) branch B agrees with B.*

Proof. First, consider languages without equality. We will show by induction on the structure of sentences in the entries that the canonical model $\mathcal{A}$ agrees with all entries E on the branch B.

The base case, i.e., when $\varphi = R(s_1, \ldots, s_n)$ is an atomic sentence, is simple: If there is an entry $\mathrm{T}\varphi$ on B, then $(s_1, \ldots, s_n) \in R^{\mathcal{A}}$ directly follows from the definition of the canonical model, so we have $\mathcal{A} \models \varphi$. If there is an entry $\mathrm{F}\varphi$ on B, then there is no entry $\mathrm{T}\varphi$ on B (since B is non-contradictory), $(s_1, \ldots, s_n) \notin R^{\mathcal{A}}$, and $\mathcal{A} \not\models \varphi$.

Now the induction step. We will discuss only some of the cases, the remaining cases are proved similarly.

For logical connectives, the proof is exactly the same as in propositional logic, for example, if $E = \mathrm{F}\varphi \wedge \psi$, then since E on B is reduced, there is an entry $\mathrm{F}\varphi$ or an entry $\mathrm{F}\psi$ on B. Thus, $\mathcal{A} \not\models \varphi$ or $\mathcal{A} \not\models \psi$, from which it follows that $\mathcal{A} \not\models \varphi \wedge \psi$ and $\mathcal{A}$ agrees with E.

If we have an entry of type 'everyone', for example, $E = \mathrm{T}(\forall x)\varphi(x)$ (the case $E = \mathrm{F}(\exists x)\varphi(x)$ is analogous), then there are entries $\mathrm{T}\varphi(x/t)$ for each ground L_C-term, i.e., for each element "t" $\in A$ on B. By the induction hypothesis, $\mathcal{A} \models \varphi(x/t)$ for each "t" $\in A$, so $\mathcal{A} \models (\forall x)\varphi(x)$.

If we have an entry of type 'witness', for example, $E = \mathrm{T}(\exists x)\varphi(x)$ (the case $E = \mathrm{F}(\forall x)\varphi(x)$ is agan analogous), then there is an entry $\mathrm{T}\varphi(x/c)$ for some "c" $\in A$ on B. By the induction hypothesis, $\mathcal{A} \models \varphi(x/c)$, so $\mathcal{A} \models (\exists x)\varphi(x)$.

If the language is with equality, we have the canonical model $\mathcal{A} = \mathcal{D}/_{=^D}$; the proof above applies to $\mathcal{D}$, and the rest follows from Observation 7.4.4. □

Exercise 7.4. Verify the remaining cases in the proof of Lemma 7.4.7.

The proof of the Completeness Theorem is also analogous to its version for propositional logic:

Theorem 7.4.8 (On Completeness)**.** *If a sentence φ is valid in a theory T, then it is tableau provable from T, i.e., $T \models \varphi \;\Rightarrow\; T \vdash \varphi$.*

Proof. We will show that any *finished* tableau from T with the entry $\mathrm{F}\varphi$ at the root must be contradictory. We prove this by contradiction: if such a tableau were not contradictory, there would be an non-contradictory (finished) branch B in it. Consider the canonical model $\mathcal{A}$ for this branch and denote its reduct to the language L as $\mathcal{A}'$. Since B is finished, it contains $\mathrm{T}\alpha$ for all axioms $\alpha \in T$. The model $\mathcal{A}$ agrees with all entries on B by Lemma 7.4.7, so it satisfies all axioms, and we have $\mathcal{A}' \models T$. But since $\mathcal{A}$ also agrees with the entry $\mathrm{F}\varphi$ at the root, it holds that $\mathcal{A}' \not\models \varphi$, which means that $\mathcal{A}' \in \mathrm{M}_L(T) \setminus \mathrm{M}_L(\varphi)$, thus $T \not\models \varphi$, and this is a contradiction. Therefore, the tableau must have been contradictory, i.e., it must have been a tableau proof of φ from T. □

7.5 Consequences of Soundness and Completeness

Same as in propositional logic, the Soundness and Completeness Theorems together state that *provability* is the same as *validity.* This allows us to similarly formulate syntactic analogs of semantic concepts and properties.

The analogs of *consequences* are *theorems* of the theory T:

$$\mathrm{Thm}_L(T) = \{\varphi \mid \varphi \text{ is an } L\text{-sentence and } T \vdash \varphi\}$$

Corollary 7.5.1 (Provability = Validity). *For any theory T and sentences φ, ψ, the following holds:*

- *$T \vdash \varphi$ if and only if $T \models \varphi$*
- $\mathrm{Thm}_L(T) = \mathrm{Csq}_L(T)$

For example, it holds that:

- A theory is *inconsistent* if the contradiction is provable in it (i.e., $T \vdash \bot$).
- A theory is *complete* if for every sentence φ either $T \vdash \varphi$ or $T \vdash \neg\varphi$ (but not both, otherwise it would be inconsistent).
- Deduction Theorem: For a theory T and sentences φ, ψ, it holds that $T, \varphi \vdash \psi$ if and only if $T \vdash \varphi \to \psi$.

At the end of this section, we will look at several applications of the Soundness and Completeness Theorems.

7.5.1 Löwenheim-Skolem Theorem

Theorem 7.5.2 (Löwenheim-Skolem). *If L is a countable language without equality, then every consistent L-theory has a countably infinite model.*

Proof. Take any finished (e.g., systematic) tableau from the theory T with the entry $\mathrm{F}\bot$ at the root. Since T is consistent, a contradiction is not provable in it, so the tableau must contain a non-contradictory branch. The desired countably infinite model is the L-reduct of the canonical model for this branch. □

We will return to this theorem in Chapter **??**, where we will show a stronger version including languages with equality (for those, the canonical model is countable but it can also be finite).

7.5.2 Compactness Theorem

Just as in propositional logic, the Compactness Theorem holds, and its proof is the same:

Theorem 7.5.3 (On Compactness)**.** *A theory has a model if and only if every finite part of it has a model.*

Proof. A model of a theory is obviously a model of each of its parts. Conversely, if T has no model, it is contradictory, so $T \vdash \bot$. Take any *finite* tableau proof of $\bot$ from T. To construct it, finitely many axioms of T suffice, and they form a finite subtheory $T' \subseteq T$, which has no model. $\square$

7.5.3 Nonstandard Model of Natural Numbers

At the very end of this section, we will show that there exists a so-called *nonstandard model* of natural numbers. The key is the Compactness Theorem.

Let $\underline{\mathbb{N}} = \langle \mathbb{N}, S, +, \cdot, 0, \leq \rangle$ be the standard model of natural numbers. Denote by $\mathrm{Th}(\underline{\mathbb{N}})$ the set of all sentences *valid* in the structure $\underline{\mathbb{N}}$ (the so-called *theory of the structure* $\underline{\mathbb{N}}$). For $n \in \mathbb{N}$, define the *n-th numeral* as the term $\underline{n} = S(S(\cdots(S(0)\cdots))$, where S is applied n times.

Let us take a new constant symbol c and express that it is strictly greater than each n-th numeral:

$$T = \mathrm{Th}(\underline{\mathbb{N}}) \cup \{\underline{n} < c \mid n \in \mathbb{N}\}$$

Note that every finite part of the theory T has a model. Thus, it follows from the Compactness Theorem that the theory T also has a model. We call it a *nonstandard model* (denote it by $\mathcal{A}$). It satisfies the same sentences as the standard model but also contains an element $c^{\mathcal{A}}$ that is greater than every $n \in \mathbb{N}$ (by which we mean the value of the term $\underline{n}$ in the nonstandard model $\mathcal{A}$).

7.6 Hilbert Calculus in Predicate Logic

At the end of the chapter, we will show how Hilbert's calculus, introduced in Section 4.8, can be adapted for use in predicate logic. This is not difficult; to deal with quantifiers, it is sufficient to add two new schemes of logical axioms and one new inference rule. We will again show the soundness of this proof system and only state without proof that it is also complete.

Proofs will consist of arbitrary formulas, not just sentences. Recall that Hilbert's calculus uses only the connectives $\neg$ and $\rightarrow$. We will have similar logical axioms as in propositional logic; in the case of a language with equality, we will additionally include the *axioms of equality.*

Definition 7.6.1 (Axiom Schemes in Hilbert Calculus in Predicate Logic)**.** For any formulas φ, ψ, χ, term t, and variable x, the following formulas are logical axioms:

(i) $\varphi \rightarrow (\psi \rightarrow \varphi)$

(ii) $(\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi))$

(iii) $(\neg\varphi \rightarrow \neg\psi) \rightarrow (\psi \rightarrow \varphi)$

(iv) $(\forall x)\varphi \rightarrow \varphi(x/t)$, if t is substitutable for x in φ

(v) $(\forall x)(\varphi \to \psi) \to (\varphi \to (\forall x)\psi)$, if x is not free in φ

If the language is with equality, then the logical axioms also include the *axioms of equality* for the given language.

Note that all logical axioms are indeed tautologies. The inference rules are *modus ponens* and *generalization*:

Definition 7.6.2 (Modus Ponens)**.** *Modus ponens* states that if we have already proved φ and also $\varphi \to \psi$, we can derive the formula ψ:

$$\frac{\varphi, \varphi \to \psi}{\psi}$$

Definition 7.6.3 (Generalization Rule)**.** The *generalization rule* states that if we have proved φ, we can derive the formula $(\forall x)\varphi$ (for any variable x):

$$\frac{\varphi}{(\forall x)\varphi}$$

Note that both inference rules are *sound*, i.e., if $T \models \varphi$ and $T \models \varphi \to \psi$ in some theory, we also have $T \models \psi$, and similarly if $T \models \varphi$, then $T \models (\forall x)\varphi$.

Just as in propositional logic, a *proof* will be a finite sequence of formulas, in which each newly written formula is either an axiom (logical, including the axiom of equality, or from the theory we are proving in), or can be derived from previous ones using one of the inference rules:

Definition 7.6.4 (Hilbert-style Proof)**.** A *Hilbert-style proof* of the formula φ from the theory T is a *finite* sequence of formulas $\varphi_0, \ldots, \varphi_n = \varphi$, in which for each $i \leq n$:

- φ_i is a logical axiom (including axioms of equality, if the language is with equality), or
- φ_i is an axiom of the theory ($\varphi_i \in T$), or
- φ_i can be derived from some previous formulas φ_j, φ_k (where $j, k < i$) using modus ponens, or
- φ_i can be derived from some previous formula φ_j (where $j < i$) using the generalization rule.

If there is a Hilbert-style proof, φ is *(Hilbert-style) provable*, and we write $T \vdash_H \varphi$.

In predicate logic, Hilbert's calculus is again a sound and complete proof system.

Theorem 7.6.5 (On the Soundness of Hilbert Calculus)**.** *For any theory T and formula φ, the following holds:*

$$T \vdash_H \varphi \;\Rightarrow\; T \models \varphi$$

Proof. By induction on the index i, we show that each formula φ_i in the proof (and thus also $\varphi_n = \varphi$) is true in T.

If φ_i is a logical axiom (including the axioms of equality), $T \models \varphi_i$ holds because logical axioms are tautologies. If $\varphi_i \in T$, certainly also $T \models \varphi_i$. The rest follows from the soundness of the inference rules. □

For the sake of completeness, we state completeness as well, but we will omit a proof.

Theorem 7.6.6 (On the Completeness of Hilbert Calculus)**.** *For any theory T and formula φ, the following holds:*

$$T \models \varphi \;\Rightarrow\; T \vdash_H \varphi$$

Chapter 8

Resolution in Predicate Logic

In this chapter, we will discuss how the resolution method introduced in Chapter 5 can be adapted for predicate logic. This chapter, the last in the part of the lecture about predicate logic, is quite extensive, so let us provide an overview of its structure:

- We begin with an informal introduction (Section 8.1).

In the following three sections, we introduce the tools that allow us to deal with the specifics of predicate logic: quantifiers, variables, and terms.

- In Section 8.2, we show how to remove quantifiers using *Skolemization*, to obtain open formulas that can be converted into CNF.
- In Section 8.3, we explain that we could seek a resolution refutation 'at the propositional logic level' (so-called *grounding*), if we first substituted 'suitable' ground terms for the variables.
- In Section 8.4, we show how to find such 'suitable' substitutions using the *unification algorithm.*

This will give us all the necessary tools to present the resolution method itself. The rest of the chapter follows a similar structure to Chapter 5.

- The resolution rule, the resolution proof, and related concepts are described in Section 8.5.
- Section 8.6 is dedicated to the proof of soundness and completeness.
- Finally, in Section 8.7, we describe LI-resolution and its application in Prolog.

8.1 Introduction

Just as in propositional logic, the resolution method in predicate logic is based on proof by contradiction. To prove that a sentence φ holds in a theory T (i.e., $T \models \varphi$), we start with the theory $T \cup \{\neg\varphi\}$. We 'convert' this theory to CNF, and then reject the resulting set of clauses S by resolution (i.e., show that $S \vdash_R \square$), thereby showing that it is unsatisfiable.

What do we mean by *conjunctive normal form*? The role of a *literal* is played by an *atomic formula*[1] or its negation. A *clause* (in set representation) is a finite set of literals, and a *formula* is a set of clauses.[2] Otherwise, we use the same terminology, e.g., we talk about *positive*, *negative*, *opposite* literals, $\square$ denotes the empty clause (which is unsatisfiable), etc.

First, let us informally demonstrate the specifics of resolution in predicate logic with a few simple examples.

Notice first that if the theory T and the sentence φ are *open* (do not contain quantifiers), we can easily construct a CNF formula S *equivalent* to the theory $T \cup \{\neg\varphi\}$ (i.e., having the same set of models). Even universal quantifiers at the beginning of the formula are not problematic; we can remove them without changing the meaning.[3]

Example 8.1.1. Let $T = \{(\forall x)P(x), (\forall x)(P(x) \to Q(x))\}$ and $\varphi = (\exists x)Q(x)$. It is easily seen that

$$T \sim \{P(x), P(x) \to Q(x)\} \sim \{P(x), \neg P(x) \vee Q(x)\}$$

and also:

$$\neg\varphi = \neg(\exists x)Q(x) \sim (\forall x)\neg Q(x) \sim \neg Q(x)$$

Therefore, we can convert the theory $T \cup \{\neg\varphi\}$ to an *equivalent* CNF formula

$$S = \{\{P(x)\}, \{\neg P(x), Q(x)\}, \{\neg Q(x)\}\}$$

which we can easily refute by resolution in two steps. (Imagine $P(x)$ as a propositional variable p and $Q(x)$ as a propositional variable q.)

Generally, this won't be possible, particularly with existential quantifiers. Unlike in propositional logic, *not* every theory is equivalent to a CNF formula. However, we can always find an *equisatisfiable* CNF formula, i.e., one that is unsatisfiable *if and only if* $T \cup \{\neg\varphi\}$ is unsatisfiable, which is sufficient for proof by contradiction. This construction is called *Skolemization* and involves replacing existentially quantified variables with newly added constant or function symbols.

For example, we replace the formula $(\exists x)\psi(x)$ with the formula $\psi(x/c)$, where c is a new constant symbol representing a *witness*, i.e., an element that satisfies the existential quantifier. Since there may be many such elements, we lose *equivalence* of theories, but it holds that if the original formula is satisfiable, then the new formula is also satisfiable, and vice versa.

Example 8.1.2. If $T = \{(\exists x)P(x), P(x) \leftrightarrow Q(x)\}$ and $\varphi = (\exists x)Q(x)$, then

$$\neg\varphi \sim (\forall x)\neg Q(x) \sim \neg Q(x)$$

and equivalence can be converted to CNF as usual, giving:

$$T \cup \{\neg\varphi\} \sim \{(\exists x)P(x), \neg P(x) \vee Q(x), \neg Q(x) \vee P(x), \neg Q(x)\}$$

Now we replace the formula $(\exists x)P(x)$ with $P(c)$, where c is a new constant symbol. This gives the CNF formula:

$$S = \{\{P(c)\}, \{\neg P(x), Q(x)\}, \{\neg Q(x), P(x)\}, \{\neg Q(x)\}\}$$

It is not equivalent to the theory $T \cup \{\neg\varphi\}$, but it is *equisatisfiable* (in this case, both are unsatisfiable).

[1] I.e., $R(t_1, \ldots, t_n)$ or $t_1 = t_2$, where t_i are L-terms and R is an n-ary relation symbol from L.

[2] As in propositional logic, we also allow infinite sets of clauses.

[3] Any formula is equivalent to its *general closure*, and the equivalence holds in both directions.

Skolemization can be more complex; sometimes a constant symbol is not enough. If we have a formula of the form $(\forall x)(\exists y)\psi(x,y)$, the chosen witness for y depends on the chosen value for x, so 'y is a function of x'. In this case, we must replace y with $f(x)$, where f is a new unary function symbol. This gives the formula $(\forall x)\psi(x, y/f(x))$, and we can now remove the universal quantifier and write only $\psi(x, y/f(x))$, which is now an open formula, albeit in a different language (extended by the symbol f). Skolemization is formally described, and the necessary properties are proved, in Section 8.2.

Now let us look at the *resolution rule*. In predicate logic, it is more complex. Again, we will show just a few examples; the formal definition will come later (Section 8.5).

Example 8.1.3. In the previous example, we arrived at the following CNF formula S, which is unsatisfiable, and we want to refute it by resolution:

$$S = \{\{P(c)\}, \{\neg P(x), Q(x)\}, \{\neg Q(x), P(x)\}, \{\neg Q(x)\}\}$$

If we look at it 'at the propositional logic level' ('ground level') and replace each atomic formula with a new propositional variable, we get $\{\{r\}, \{\neg p, q\}, \{\neg q, p\}, \{\neg q\}\}$, which is not unsatisfiable. We need to use the fact that $P(c)$ and $P(x)$ have a 'similar structure' (they are *unifiable*).

Since the clause $\{\neg P(x), Q(x)\}$ is valid in S (it is an axiom), it is also valid after performing *any substitution*, i.e., the clause $\{\neg P(x/t), Q(x/t)\}$ is a consequence of S for any term t. We might imagine 'adding' all such clauses to S.[4] The resulting CNF formula, when converted to the 'propositional logic level', would be unsatisfiable.

The *unification algorithm* directly tells us that the correct substitution is x/c, and we include this already in the *resolution rule*, i.e., the *resolvent* of the clauses $\{P(c)\}$ and $\{\neg P(x), Q(x)\}$ will be the clause $\{Q(c)\}$.

Unification can be more complex, and we should also note another difference from propositional logic: we allow resolution over multiple literals at once if they are all *unifiable* together:

Example 8.1.4. From the clauses $\{R(x, f(x)), R(g(y), z)\}$ and $\{\neg R(g(c), u), P(u)\}$ (where R is binary relational, f and g are unary functional, and c is a constant symbol), it will be possible to derive the resolvent $\{P(f(g(c))\}$ using the following *substitution* (*unification*): $\{x/g(c), y/c, z/f(g(c)), u/f(g(c))\}$; here we select *both* literals at once from the first clause.

Remark 8.1.5. The fact that variables have a 'local meaning' in individual clauses (i.e., we can substitute them in one clause without affecting other clauses) is due to the following simple tautology, which holds for any formulas ψ, χ (even if x is free in both):

$$\models (\forall x)(\psi \land \chi) \leftrightarrow (\forall x)\psi \land (\forall x)\chi$$

As we have seen in the previous example, we will also require that clauses in the resolution rule have disjoint sets of variables; this can be achieved by renaming variables, which is a special case of substitution.

8.2 Skolemization

In this section, we will describe a procedure to reduce the question of the satisfiability of a given theory T to the question of the satisfiability of an *open* theory T'. Recall that T and T' will generally not be equivalent but will be *equisatisfiable*:

[4]There are infinitely many such clauses; even just the *variants* of one clause, i.e., clauses obtained by simply renaming variables, are infinite in number. We are OK with that; a CNF formula can, by definition, be infinite.

Definition 8.2.1 (Equisatisfiability)**.** Given a theory T in language L and a theory T' in not necessarily the same language L', we say that T and T' are *equisatisfiable* if:

$$T \text{ has a model } \Leftrightarrow T' \text{ has a model}$$

The entire construction consists of the following steps, which we will explain below:

1. Conversion to *prenex normal form* (pulling quantifiers to the front).
2. Replacing formulas with their general closures (to obtain sentences).
3. Removing existential quantifiers (replacing sentences with their *Skolem variants*).
4. Removing remaining universal quantifiers (resulting in open formulas).

8.2.1 Prenex Normal Form

First, we will show the process by which any formula can have its quantifiers 'pulled to the front', i.e., converted to the so-called *prenex normal form*, which starts with a sequence of quantifiers and the rest is a free formula.

Definition 8.2.2 (PNF)**.** A formula φ is in *prenex normal form (PNF)* if it is of the form

$$(Q_1x_1)\dots(Q_nx_n)\varphi'$$

where Q_i is a quantifier (either $\forall$ or $\exists$), and the formula φ' is open. The formula φ' is called the *matrix* of φ, and $(Q_1x_1)\dots(Q_nx_n)$ is the *quantifier prefix.*

If φ is a formula in PNF and all the quantifiers are universal, then we say that φ is a *universal* formula.

The goal of this subsection is to show the following proposition:

Proposition 8.2.3 (Conversion to PNF)**.** *For each formula φ, there exists an equivalent formula in prenex normal form.*

The algorithm, like the conversion to CNF, will be based on replacing subformulas with *equivalent* subformulas, aiming to move quantifiers closer to the root of the formula tree. What do we mean by the equivalence of formulas $\varphi \sim \varphi'$? That they have the same meaning, i.e., they have the same truth value in every model and for every variable assignment. Equivalently, that $\models \varphi \leftrightarrow \varphi'$ holds. We will need the following simple observation:

Observation 8.2.4. *If we replace a subformula ψ of a formula φ with an equivalent formula ψ', then the resulting formula φ' is also equivalent to the formula φ.*

The conversion is based on repeated application of the following syntactic rules:

Lemma 8.2.5. *Let $\overline{Q}$ denote the quantifier opposite to Q. Let φ and ψ be formulas, and let x be a variable that is* not *free in the formula ψ. Then the following hold:*

$$\begin{aligned}
\neg(Qx)\varphi &\sim (\overline{Q}x)\neg\varphi \\
(Qx)\varphi \wedge \psi &\sim (Qx)(\varphi \wedge \psi) \\
(Qx)\varphi \vee \psi &\sim (Qx)(\varphi \vee \psi) \\
(Qx)\varphi \rightarrow \psi &\sim (\overline{Q}x)(\varphi \rightarrow \psi) \\
\psi \rightarrow (Qx)\varphi &\sim (Qx)(\psi \rightarrow \varphi)
\end{aligned}$$

Proof. The rules can be easily verified semantically, or proved using the tableau method (in that case, if the formulas are not sentences, we must replace them with their general closures). □

Note that in the rule $(Qx)\varphi \to \psi \sim (\overline{Q}x)(\varphi \to \psi)$ for pulling a quantifier out of the *antecedent* of an implication, we must change the quantifier (from $\forall$ to $\exists$ and vice versa), whereas when pulling it out of the *consequent*, the quantifier remains the same. Why is it so? This is best seen if we rewrite the implication using disjunction and negation:

$$(Qx)\varphi \to \psi \sim \neg(Qx)\varphi \lor \psi \sim (\overline{Q}x)(\neg\varphi) \lor \psi \sim (\overline{Q}x)(\neg\varphi \lor \psi) \sim (\overline{Q}x)(\varphi \to \psi)$$

Also note the assumption that x is not free in ψ. Without it, the rules would not work, e.g.:

$$(\exists x)P(x) \land Q(x) \ \not\sim\ (\exists x)(P(x) \land Q(x))$$

In such a situation, we replace the formula with a variant in which we rename the bound variable x to a new variable:

$$(\exists x)P(x) \land Q(x) \ \sim\ (\exists y)P(y) \land Q(x) \sim (\exists y)(P(y) \land Q(x))$$

Exercise 8.1. Prove Observation 8.2.4 and all the rules in Lemma 8.2.5.

Let us demonstrate the process with an example:

Example 8.2.6. Convert the formula $((\forall z)P(x,z) \land P(y,z)) \to \neg(\exists x)P(x,y)$ to PNF. We will only write out the intermediate steps. Note what rule was applied to which subformula (and also the renaming of the variable in the first step), and follow the process on the formula tree.

$$\begin{aligned}
&(\forall z)P(x,z) \land P(y,z) \ \to\ \neg(\exists x)P(x,y)\\
\sim\ &(\forall u)P(x,u) \land P(y,z) \ \to\ (\forall x)\neg P(x,y)\\
\sim\ &(\forall u)(P(x,u) \land P(y,z)) \ \to\ (\forall v)\neg P(v,y)\\
\sim\ &(\exists u)(P(x,u) \land P(y,z) \ \to\ (\forall v)\neg P(v,y))\\
\sim\ &(\exists u)(\forall v)(P(x,u) \land P(y,z) \ \to\ \neg P(v,y))
\end{aligned}$$

Now we are ready to prove Proposition 8.2.3:

Proof of Proposition 8.2.3. By induction on the structure of the formula φ, using Lemma 8.2.5 and Observation 8.2.4. □

Since every formula $\varphi(x_1,\dots,x_n)$ is equivalent to its *general closure*

$$(\forall x_1)\dots(\forall x_n)\varphi(x_1,\dots,x_n)$$

we can state Proposition 8.2.3 as follows:

Corollary 8.2.7. *For every formula φ, there exists an equivalent* sentence *in PNF.*

E.g., in Example 8.2.6, the resulting sentence is $(\forall x)(\forall y)(\forall z)(\exists u)(\forall v)(P(x,u) \land P(y,z) \to \neg P(v,y))$.

Remark 8.2.8. The prenex form is not unique; the rules for conversion can be applied in different order. As we will see in the next subsection, it is advantageous to first pull out quantifiers that become existential: if we have a choice between $(\forall x)(\exists y)\varphi(x,y)$ and $(\exists y)(\forall x)\varphi(x,y)$, we choose the second option because, in the first case, 'y depends on x'.

8.2.2 Skolem Variant

Now we have converted our axioms to equivalent sentences in prenex form. If any sentence contained only universal quantifiers, i.e., if it was of the form

$$(\forall x_1)\dots(\forall x_n)\varphi(x_1,\dots,x_n)$$

where φ is open, we could simply replace it with its matrix φ, which is equivalent in this case. But how do we deal with existential quantifiers, e.g., $(\exists x)\varphi(x)$, $(\forall x)(\exists y)\varphi(x,y)$, etc.? We first replace them with their *Skolem variant.*

Definition 8.2.9 (Skolem Variant)**.** Let φ be an L-sentence in PNF, and let all its bound variables be different. Let the existential quantifiers from the prefix of φ be $(\exists y_1),\dots,(\exists y_n)$ (in this order), and let for each i, $(\forall x_1),\dots,(\forall x_{n_i})$ be precisely all the universal quantifiers preceding the quantifier $(\exists y_i)$ in the prefix of φ.

Let L' be the extension of L with *new* n_i-ary function symbols $f_1,\dots,f_n$, where the symbol f_i has arity n_i for each i. The *Skolem variant* of the sentence φ is the L'-sentence φ_S obtained from φ by, for each $i=1,\dots,n$:

- removing the quantifier $(\exists y_i)$ from the prefix, and
- substituting the variable y_i with the term $f_i(x_1,\dots,x_{n_i})$.

This process is also called *skolemization.*

Example 8.2.10. The Skolem variant of the sentence

$$\varphi = (\exists y_1)(\forall x_1)(\forall x_2)(\exists y_2)(\forall x_3)R(y_1,x_1,x_2,y_2,x_3)$$

is the sentence

$$\varphi_S = (\forall x_1)(\forall x_2)(\forall x_3)R(f_1,x_1,x_2,f_2(x_1,x_2),x_3)$$

where f_1 is a new constant symbol, and f_2 is a new binary function symbol.

Remark 8.2.11. Note that when skolemizing, we must start from a sentence! For example, given the formula $(\exists y)E(x,y)$, $E(x,c)$ is *not* its Skolem variant. We must first take the general closure $(\forall x)(\exists y)E(x,y)$, and then correctly skolemize it as $(\forall x)E(x,f(x))$, which is equivalent to the open formula $E(x,f(x))$ (which says something much weaker than $E(x,c)$).

It is also important that each symbol used in skolemization is genuinely new; its only 'role' in the whole theory must be to represent the 'existing' elements in this formula.

In the following lemma, we show the key property of the Skolem variant:

Lemma 8.2.12. *Let $\varphi = (\forall x_1)\dots(\forall x_n)(\exists y)\psi$ be an L-sentence, and let φ' be the sentence*

$$(\forall x_1)\dots(\forall x_n)\psi(y/f(x_1,\dots,x_n))$$

where f is a new function symbol. Then:

(i) The L-reduct of every model of φ' is a model of φ, and

(ii) Every model of φ can be expanded to a model of φ'.

Proof. First, prove part (i): Let $\mathcal{A}' \models \varphi'$, and let $\mathcal{A}$ be its reduct to the language L. For each variable assignment e, $\mathcal{A} \models \psi[e(y/a)]$ holds for $a = (f(x_1, \dots, x_n))^{\mathcal{A}'}[e]$, so $\mathcal{A} \models \varphi$.

Now, part (ii): Since $\mathcal{A} \models \varphi$, there exists a function $f^A : A^n \to A$ such that for every variable assignment e, $\mathcal{A} \models \psi[e(y/a)]$, where $a = f^A(e(x_1), \dots, e(x_n))$. Thus, the expansion of the structure $\mathcal{A}$ obtained by adding the function f^A is a model of φ'. $\square$

Remark 8.2.13. The expansion of the model in the second part of the lemma need not be (and typically is not) unique, unlike the extension by definition of a new function symbol.

Applying the previous lemma repeatedly (for all existential quantifiers, in sequence) gives us the following corollary:

Corollary 8.2.14. *A sentence φ and its Skolem variant φ_S are equisatisfiable.*

8.2.3 Skolem's Theorem

In this subsection, we summarize the entire process described in the previous subsections. The key is the following theorem by Norwegian logician Thoralf Skolem:

Theorem 8.2.15 (Skolem's Theorem)**.** *Every theory has an open conservative extension.*

Proof. Let T be an L-theory. Replace each axiom with its general closure (if it is not already a sentence) and convert it to PNF, obtaining an equivalent theory T'. Now replace each axiom of the theory T' with its Skolem variant. This gives us a theory T'' in an expanded language L'. From Lemma 8.2.12, it follows that the L-reduct of each model of T'' is a model of T', so T'' is an extension of T', and that each model of T' can be expanded to the language L' to be a model of T'', so it is a conservative extension. The theory T'' is axiomatized by universal sentences; removing the quantifier prefixes (i.e., taking the matrices of the axioms) gives us the open theory T''', which is equivalent to T'' and therefore also a conservative extension of T. $\square$

From the semantic characterization of conservative extension, the following corollary easily follows:

Corollary 8.2.16. *For any theory, using skolemization we can find an equisatisfiable open theory.*

We can now easily convert an open theory to CNF (expressed as a *formula S* in set representation) using equivalent syntactic transformations, just as in propositional logic (see Section 2.3.2).

8.3 Grounding

In this section, we show that if we have an open theory that is unsatisfiable, we can demonstrate its unsatisfiability 'on specific elements'. What do we mean by that? There exists a finite number of *ground instances* of the axioms (instances where we substitute ground terms for the variables) such that their conjunction (which contains no variables) is unsatisfiable.

Definition 8.3.1 (Ground Instance)**.** Let φ be an open formula in free variables $x_1, \dots, x_n$. We say that the instance $\varphi(x_1/t_1, \dots, x_n/t_n)$ is a *ground instance* if all terms $t_1, \dots, t_n$ are ground (constant).

Example 8.3.2. The theory $T = \{P(x,y) \vee R(x,y), \neg P(c,y), \neg R(x, f(x))\}$ in the language $L = \langle P, R, f, c\rangle$ has no model. We can demonstrate this with the following conjunction of ground instances of the axioms, where we substitute the constant c for the variable x and the ground term $f(c)$ for y:

$$(P(c,f(c)) \vee R(c,f(c))) \ \wedge \ \neg P(c,f(c)) \ \wedge \ \neg R(c,f(c))$$

This sentence is clearly unsatisfiable. Moreover, the ground atomic sentences $(P(c,f(c))$ and $R(c,f(c)))$ can be understood (because they contain no variables) as propositional variables p_1, p_2, where p_1 means '$P(c,f(c))$ is valid' and p_2 means '$R(c,f(c))$ is valid'. We then get the following proposition, which can be easily refuted by resolution:

$$(p_1 \vee p_2) \wedge \neg p_1 \wedge \neg p_2$$

This process of converting to ground instances (and thus to propositional logic) is called 'grounding'. We will formalize it shortly, and prove *Herbrand's theorem*,[5] which states that such an unsatisfiable conjunction of ground instances of the axioms exists for every unsatisfiable theory.

8.3.1 Direct Reduction to Propositional Logic

Let us now realize that thanks to Herbrand's theorem, grounding allows for the following (although inefficient) procedure for refuting formulas by resolution 'at the propositional logic level': In the input formula S, we replace each clause with the set of all its ground instances (if there are none, i.e., if the language does not contain a constant symbol, we add one constant symbol to the language). In the resulting set of clauses S', we treat atomic sentences as propositional variables, and we refute S' using propositional resolution (which we know is sound and complete).

The problem with this approach is that the number of clauses in S' (ground instances of the clauses from S) can be large, even infinite, e.g., whenever the language contains at least one functional (non-constant) symbol.

Example 8.3.3. If we have a CNF formula $S = \{\{P(x,y), R(x,y)\}, \{\neg P(c,y)\}, \{\neg R(x, f(x))\}\}$ in the language $L = \langle f, c\rangle$, we replace it with the following infinite formula S':

$$\begin{aligned} S' = \{&\{P(c,c), R(c,c)\}, \{P(c,f(c)), R(c,f(c))\}, \{P(f(c),c), R(f(c),c)\}, \ldots, \\ &\{\neg P(c,c)\}, \{\neg P(c,f(c))\}, \{\neg P(c,f(f(c)))\}, \{\neg P(c,f(f(f(c))))\}, \ldots, \\ &\{\neg R(c,f(c))\}, \{\neg R(f(c),f(f(c)))\}, \{\neg R(f(f(c)),f(f(f(c))))\}, \ldots\} \end{aligned}$$

It is unsatisfiable because it contains the following finite subset, which is unsatisfiable, as we can easily show by propositional resolution:

$$\{\{P(c,f(c)), R(c,f(c))\}, \{\neg P(c,f(c))\}, \{\neg R(c,f(c))\}\} \vdash_R \square$$

In Section 8.4, we will show an efficient procedure for finding suitable ground instances of clauses using so-called *unification*.

[5]French mathematician Jacques Herbrand worked at the end of the 1920s. During his short career (he tragically died at the age of 23), he discovered several other important results, and among other things, formalized the concept of a recursive function.

8.3.2 Herbrand's Theorem

In this subsection, we state and prove Herbrand's theorem. We assume that the language contains some constant symbol: if the language does not contain any, we add one. We need a constant symbol so that there are ground terms and we can create the so-called *Herbrand model.* This is the construction of a semantic object (model) from syntactic objects (ground terms) very similar to the *canonical model* (Definition 7.4.3).[6]

Definition 8.3.4 (Herbrand Model)**.** Let $L = \langle \mathcal{R}, \mathcal{F} \rangle$ be a language with at least one constant symbol. An L-structure $\mathcal{A} = \langle A, \mathcal{R}^{\mathcal{A}}, \mathcal{F}^{\mathcal{A}} \rangle$ is a *Herbrand model* if:

- A is the set of all ground L-terms (the so-called *Herbrand universe*), and
- for each n-ary function symbol $f \in \mathcal{F}$ and ground terms "t_1", ..., "t_n" $\in A$,

$$f^{\mathcal{A}}(\text{“}t_1\text{”}, \ldots, \text{“}t_n\text{”}) = \text{“}f(t_1, \ldots, t_n)\text{”}$$

- Specifically, for each constant symbol $c \in \mathcal{F}$, $c^{\mathcal{A}} = $ "c".

We do not impose any conditions on the interpretations of relation symbols.

Recall that we use quotes around terms informally to clearly distinguish terms as syntactic objects (strings of symbols) from their interpretations (functions).

Example 8.3.5. Let $L = \langle P, f, c \rangle$ be a language where P is a unary relation symbol, f is a binary function symbol, and c is a constant symbol. The Herbrand universe for this language is the set

$$A = \{\text{“}c\text{”}, \text{“}f(c,c)\text{”}, \text{“}f(c, f(c,c))\text{”}, \text{“}f(f(c,c),c)\text{”} \ldots\}$$

The structure $\mathcal{A} = \langle A, P^{\mathcal{A}}, f^{\mathcal{A}}, c^{\mathcal{A}} \rangle$ is a Herbrand model if $c^{\mathcal{A}} = $ "c" and the function $f^{\mathcal{A}}$ satisfies:

- $f^{\mathcal{A}}(\text{“}c\text{”}, \text{“}c\text{”}) = \text{“}f(c,c)\text{”}$,
- $f^{\mathcal{A}}(\text{“}c\text{”}, \text{“}f(c,c)\text{”}) = \text{“}f(c, f(c,c))\text{”}$,
- $f^{\mathcal{A}}(\text{“}f(c,c)\text{”}, \text{“}c\text{”}) = \text{“}f(f(c,c),c)\text{”}$, etc.

The relation $P^{\mathcal{A}}$ can be any subset of A.

We are now ready to state Herbrand's theorem. Informally, it states that if a theory is satisfiable, i.e., has a model, then it even has a Herbrand model, and otherwise, we can find an unsatisfiable conjunction of ground instances of the axioms, which can be used for resolution refutation 'at the propositional logic level'.

Theorem 8.3.6 (Herbrand's Theorem)**.** *Let T be an open theory in a language L without equality and with at least one constant symbol. Then either T has a* Herbrand model*, or there exist finitely many ground instances of axioms of T whose conjunction is unsatisfiable.*

[6]The difference is that we do not add countably many new constant symbols (we only use the constant symbols already in the language), and we do not prescribe how the relations of the model should look.

Proof. Let T_{ground} denote the set of all ground instances of the axioms of the theory T. We construct a systematic[7] tableau from the theory T_{ground} with the entry F$\bot$ at the root, but in the language L, without expanding it by auxiliary constant symbols to the language L_C.[8]

If the tableau contains a non-contradictory branch, then the canonical model for this branch (again without adding auxiliary constant symbols) is a Herbrand model of T. Otherwise, we have a tableau proof of contradiction, so the theory T_{ground}, and therefore T, is unsatisfiable. Since the tableau proof is finite, we used only finitely many ground instances of the axioms $\alpha_{\text{ground}} \in T_{\text{ground}}$. Their conjunction is therefore unsatisfiable. $\square$

Remark 8.3.7. If the language is with equality, we first extend the theory T with the axioms of equality to obtain the theory T^*, and if T^* has a Herbrand model $\mathcal{A}$, we factorize it by the congruence $=^A$, just as in the case of the canonical model.

To conclude this section, we state two corollaries of Herbrand's theorem.

Corollary 8.3.8. *Let $\varphi(x_1, \ldots, x_n)$ be an open formula in a language L with at least one constant symbol. Then there exist ground L-terms t_{ij} ($1 \leq i \leq m, 1 \leq j \leq n$) such that the sentence*

$$(\exists x_1) \ldots (\exists x_n)\varphi(x_1, \ldots, x_n)$$

is true if and only if the following formula (a propositional tautology) is true:

$$\varphi(x_1/t_{11}, \ldots, x_n/t_{1n}) \vee \cdots \vee \varphi(x_1/t_{m1}, \ldots, x_n/t_{mn})$$

Proof. The sentence $(\exists x_1) \ldots (\exists x_n)\varphi(x_1, \ldots, x_n)$ is true if and only if $(\forall x_1) \ldots (\forall x_n)\neg\varphi$ is unsatisfiable, i.e., if $\neg\varphi$ is unsatisfiable. The claim follows from Herbrand's theorem applied to the theory $T = \{\neg\varphi\}$. $\square$

Corollary 8.3.9. *Let T be an open theory in a language with at least one constant symbol. The theory T has a model if and only if the theory T_{ground}, consisting of all ground instances of the axioms of T, has a model.*

Proof. In a model of the theory T, all the axioms hold, and so do all their ground instances. Therefore, it is also a model of T_{ground}. If T has no model, then by Herbrand's theorem, some finite subset of the theory T_{ground} is unsatisfiable. $\square$

8.4 Unification

Instead of substituting *all* ground terms and working with this new, huge, and typically infinite set of clauses, it is better to find a substitution 'suitable' for the specific resolution step, and work only with it. In this section, we will explain what 'suitable' means (so-called *unification*) and how to find it (using the *unification algorithm*).

[7]Or any finished tableau, but in such a way that we do not extend contradictory branches.

[8]Since there are no quantifiers in T_{ground}, auxiliary symbols are not used anywhere in the tableau.

8.4.1 Substitution

First, let us provide a few examples of 'suitable' substitutions:

Example 8.4.1.
- From the clauses $\{P(x), Q(x,a)\}$ and $\{\neg P(y), \neg Q(b,y)\}$, we obtain, using the substitution $\{x/b, y/a\}$, the clauses $\{P(b), Q(b,a)\}$ and $\{\neg P(a), \neg Q(b,a)\}$, and from them, we derive the clause $\{P(b), \neg P(a)\}$ by resolution. We could also use the substitution $\{x/y\}$ and derive the resolvent $\{Q(y,a), \neg Q(b,y)\}$ by resolving over $P(y)$.
- Given the clauses $\{P(x), Q(x,a), Q(b,y)\}$ and $\{\neg P(v), \neg Q(u,v)\}$, a suitable substitution is $\{x/b, y/a, u/b, v/a\}$; we get $\{P(b), Q(b,a)\}$ and $\{\neg P(a), \neg Q(b,a)\}$, whose resolvent is $\{P(b), \neg P(a)\}$.
- Consider the clauses $\{P(x), Q(x,z)\}$ and $\{\neg P(y), \neg Q(f(y),y)\}$. We could use the substitution $\{x/f(a), y/a, z/a\}$ to obtain the pair of clauses $\{P(f(a)), Q(f(a),a)\}$ and $\{\neg P(a), \neg Q(f(a),a)\}$, resolving to $\{P(f(a)), \neg P(a)\}$.

  However, it is better to use the substitution $\{x/f(z), y/z\}$, after which we have the clauses $\{P(f(z)), Q(f(z),z)\}$ and $\{\neg P(z), \neg Q(f(z),z)\}$. Their resolvent is then the clause $\{P(f(z)), \neg P(z)\}$. This substitution is *more general*; and the resulting resolvent 'says more' than $\{P(f(a)), \neg P(a)\}$ (the latter is its consequence, but not vice versa).

We now introduce the necessary terminology related to substitutions. Substitutions will be applied to terms or literals (atomic formulas or their negations), collectively referred to as *expressions.*

Definition 8.4.2 (Substitution)**.** A *substitution* is a finite set $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$, where x_i are distinct variables and t_i are terms, with the requirement that term t_i is not equal to the variable x_i. The substitution σ is

- *ground* if all the terms t_i are ground,
- a *renaming of variables* if all terms t_i are distinct variables.

An *instance* of an expression (term or literal) E *under the substitution* $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$ is the expression obtained from E by simultaneously replacing all occurrences of variables x_i with terms t_i, denoted $E\sigma$. If S is a set of expressions, we denote $S\sigma = \{E\sigma \mid E \in S\}$.

Since variables are replaced *simultaneously* for all variables at once, any occurrence of variable x_i in term t_j will not lead to a chain of substitutions.

Example 8.4.3. For $S = \{P(x), R(y,z)\}$ and substitution $\sigma = \{x/f(y,z), y/x, z/c\}$, we have:

$$S\sigma = \{P(f(y,z)), R(x,c)\}$$

Substitutions can be naturally *composed.* The composition of substitutions σ and τ, where σ is applied first, and then τ is applied to the result, will be denoted as $\sigma\tau$. Thus, $E(\sigma\tau) = (E\sigma)\tau$ holds for any expression E.

Example 8.4.4. Let us start with an example. Given the expression $E = P(x,w,u)$ and the substitutions

$$\sigma = \{x/f(y), w/v\}$$
$$\tau = \{x/a, y/g(x), v/w, u/c\}$$

we have $E\sigma = P(f(y), v, u)$ and $(E\sigma)\tau = P(f(g(x)), w, c)$. Therefore, it must hold:

$$\sigma\tau = \{x/f(g(x)), y/g(x), v/w, u/c\}$$

Now the formal definition:

Definition 8.4.5 (Composition of Substitutions)**.** Let $\sigma = \{x_1/t_1, \dots, x_n/t_n\}$ and $\tau = \{y_1/s_1, \dots, y_m/s_m\}$ be substitutions. The *composition of substitutions σ and τ* is the substitution

$$\sigma\tau = \{x_i/t_i\tau \mid x_i \in X, x_i \neq t_i\tau\} \cup \{y_j/s_j \mid y_j \in Y \setminus X\}$$

where $X = \{x_1, \dots, x_n\}$ and $Y = \{y_1, \dots, y_m\}$.

Note that the composition of substitutions is not commutative; $\sigma\tau$ is typically a completely different substitution from $\tau\sigma$.

Example 8.4.6. If σ and τ are as in Example 8.4.4, then:

$$\tau\sigma = \{x/a, y/g(f(y)), u/c, w/v\} \neq \sigma\tau$$

We now show that thus defined composition of substitutions satisfies the required properties and, moreover, that it is *associative*. From associativity, it follows that we do not need to (and will not) write parentheses in the composition $\sigma\tau\varrho$, $\sigma_1\sigma_2\cdots\sigma_n$, etc.

Proposition 8.4.7. *Let σ, τ, and ϱ be substitutions, and let E be any expression. Then the following holds:*

(i) $(E\sigma)\tau = E(\sigma\tau)$

(ii) $(\sigma\tau)\varrho = \sigma(\tau\varrho)$

Proof. Let $\sigma = \{x_1/t_1, \dots, x_n/t_n\}$ and $\tau = \{y_1/s_1, \dots, y_m/s_m\}$. It is sufficient to prove the case when the expression E is a single variable, the rest easily follows by structural induction. (Substitutions do not change other symbols.) We split the argument into three cases:

- If $E = x_i$ for some i, then $E\sigma = t_i$ and $(E\sigma)\tau = t_i\tau = E(\sigma\tau)$, where the second equality is by definition of $\sigma\tau$.
- If $E = y_j$ for some j, where $y_j \notin \{x_1, \dots, x_n\}$, then $E\sigma = E$ and $(E\sigma)\tau = E\tau = s_j = E(\sigma\tau)$ again by definition of $\sigma\tau$.
- If E is another variable, then $(E\sigma)\tau = E = E(\sigma\tau)$.

This proves (i). Associativity (ii) can be easily proved by repeated use of (i). The following holds for any expression E, hence also for any variable:

$$E((\sigma\tau)\varrho) = (E(\sigma\tau))\varrho = ((E\sigma)\tau)\varrho = (E\sigma)(\tau\varrho) = E(\sigma(\tau\varrho)).$$

It follows that $(\sigma\tau)\varrho$ and $\sigma(\tau\varrho)$ are the same substitution.[9] □

[9]In more detail: we use the obvious property that for a substitution π it holds $\pi = \{z_1/v_1, \dots, z_k/v_k\}$ if and only if $E\pi = v_i$ for $E = z_i$ and $E\pi = E$ if E is a variable different from all z_i.

8.4.2 Unification Algorithm

Which substitutions are 'suitable'? Those that, after being applied, cause the given expressions to 'become the same', i.e., *unified* (see Example 8.4.1).

Definition 8.4.8 (Unification)**.** Let $S = \{E_1, \ldots, E_n\}$ be a finite set of expressions. A substitution σ is a *unification of S* if $E_1\sigma = E_2\sigma = \cdots = E_n\sigma$, i.e., $S\sigma$ contains a single expression. If it exists, we also say that S is *unifiable.*

A unification of S is *most general* if for every unification τ of S there exists a substitution λ such that $\tau = \sigma\lambda$. Note that there may be multiple most general unifications of S, but they differ only by renaming of variables.

Example 8.4.9. Consider the set of expressions $S = \{P(f(x), y), P(f(a), w)\}$. The most general unification of S is $\sigma = \{x/a, y/w\}$. Another unification is, for example, $\tau = \{x/a, y/b, w/b\}$, but it is not most general because it cannot yield, for example, the unification $\varrho = \{x/a, y/c, w/c\}$. The unification τ can be obtained from the most general unification σ using the substitution $\lambda = \{w/b\}$: $\tau = \sigma\lambda$.

We now introduce the *unification algorithm.* Its input is a non-empty, finite set of expressions S, and its output is either the most general unification of S or information that S is not unifiable. The algorithm proceeds from the start of the expressions, successively applying substitutions to make the expressions more similar. We need the following definition:

Let p be the first (leftmost) position where some two expressions from S differ. Then the *difference in S*, denoted $D(S)$, is the set of all subexpressions starting at position p of expressions in S.

Example 8.4.10. For $S = \{P(x, y), P(f(x), z), P(z, f(x))\}$, $p = 3$ and $D(S) = \{x, f(x), z\}$.

Algorithm (Unification Algorithm)**.**

- **Input**: a finite set of expressions $S \neq \emptyset$,
- **Output:** the most general unification σ of S or information that S is not unifiable

```
(0) set $S_0 := S$, $\sigma_0 := \emptyset$, $k := 0$
(1) if $|S_k| = 1$, return $\sigma = \sigma_0\sigma_1 \cdots \sigma_k$
(2) determine if $D(S_k)$ contains a variable $x$ and a term $t$ not containing $x$
(3) if yes, set $\sigma_{k+1} := \{x/t\}$, $S_{k+1} := S_k\sigma_{k+1}$, $k := k + 1$, and go to (1)
(4) if no, report that $S$ is not unifiable
```

Remark 8.4.11. Finding a variable x and a term t in step (2) can be computationally expensive.

Before proving correctness, let us demonstrate the algorithm with an example.

Example 8.4.12. Let us apply the unification algorithm to the following set:

$$S = \{P(f(y, g(z)), h(b)),\ P(f(h(w), g(a)), t),\ P(f(h(b), g(z)), y)\}$$

($k = 0$) The set $S_0 = S$ is not a singleton, $D(S_0) = \{y, h(w), h(b)\}$ contains the term $h(w)$ and the variable y not occurring in $h(w)$. Set $\sigma_1 = \{y/h(w)\}$ and $S_1 = S_0\sigma_1$, thus we have:

$$S_1 = \{P(f(h(w), g(z)), h(b)),\ P(f(h(w), g(a)), t),\ P(f(h(b), g(z)), h(w))\}$$

$(k=1)$ $D(S_1) = \{w, b\}$, $\sigma_2 = \{w/b\}$, $S_2 = S_1\sigma_2$, thus:

$$S_2 = \{P(f(h(b), g(z)), h(b)),\ P(f(h(b), g(a)), t)\}$$

$(k=2)$ $D(S_2) = \{z, a\}$, $\sigma_3 = \{z/a\}$, $S_3 = S_2\sigma_3$, thus:

$$S_3 = \{P(f(h(b), g(a)), h(b)),\ P(f(h(b), g(a)), t)\}$$

$(k=3)$ $D(S_3) = \{h(b), t\}$, $\sigma_4 = \{t/h(b)\}$, $S_4 = S_3\sigma_4$, thus:

$$S_4 = \{P(f(h(b), g(a)), h(b))\}$$

$(k=4)$ S_4 is one-element, the most general unification of S is the following:

$$\sigma = \sigma_1\sigma_2\sigma_3\sigma_4 = \{y/h(w)\}\{w/b\}\{z/a\}\{t/h(b)\} = \{y/h(b), w/b, z/a, t/h(b)\}$$

Proposition 8.4.13. *The unification algorithm is correct. For any input S, it terminates in a finite number of steps, and if S is unifiable, it returns the most general unification σ; otherwise, it reports that S is not unifiable.*

Moreover, if S is unifiable, then for the constructed most general unification σ, it additionally holds that if τ is any unification, then $\tau = \sigma\tau$.

Proof. In each step k, we eliminate some variable, so the algorithm must terminate. If the algorithm terminates unsuccessfully in step k, then it is not possible to unify the set S_k. It is easy to see that in that case, it is not possible to unify S either.

If the algorithm returns $\sigma = \sigma_0\sigma_1 \cdots \sigma_k$, it is evidently a unification. It remains to prove that it is most general, for which it suffices to prove the stronger property ('moreover') described in the proposition.

Let τ be any unification of S. We will show by induction that for every $0 \leq i \leq k$:

$$\tau = \sigma_0\sigma_1 \cdots \sigma_i\tau$$

For $i = 0$, $\sigma_0 = \emptyset$ and $\tau = \sigma_0\tau$ holds trivially. Assume it holds for some i, and prove it for $i + 1$. Let $\sigma_{i+1} = \{x/t\}$. It suffices to prove that for any variable u, we have that

$$u\sigma_{i+1}\tau = u\tau$$

From this, $\tau = \sigma_0\sigma_1 \cdots \sigma_i\sigma_{i+1}\tau$ immediately follows.

If $u \neq x$, then $u\sigma_{i+1} = u$, so $u\sigma_{i+1}\tau = u\tau$. In the case $u = x$, we have $u\sigma_{i+1} = x\sigma_{i+1} = t$. Since τ unifies the set $S_i = S\sigma_0\sigma_1 \cdots \sigma_i$, and the variable x and the term t are in the difference $D(S_i)$, τ must unify x and t. In other words, $t\tau = x\tau$, i.e., $u\sigma_{i+1}\tau = u\tau$, which is what we wanted to prove. □

8.5 Resolution Method

If we want to prove that $T \models \varphi$, we find (using skolemization) a CNF formula S that is unsatisfiable if and only if the theory $T \cup \{\neg\varphi\}$ is unsatisfiable, i.e., if and only if $T \models \varphi$. Then, all we need to do is to find a resolution refutation of S.

In this section, we describe the resolution method itself. Most notions and theorems will be very similar to their counterparts from propositional logic. The only significant difference will be the *resolution rule.*

8.5.1 Resolution Rule

The resolvent of a pair of clauses will be a clause derived from them by applying a *(most general) unification.* First, an example:

Example 8.5.1. Let $C_1 = \{P(x), Q(x, y), Q(x, f(z))\}$ and $C_2 = \{\neg P(u), \neg Q(f(u), u)\}$. Select *both* positive literals starting with Q from the first clause, and the negative literal starting with $\neg Q$ from the second clause. The set of expressions $S = \{Q(x, y), Q(x, f(z)), Q(f(u), u)\}$ can be unified using the most general unification $\sigma = \{x/f(f(z)), y/f(z), u/f(z)\}$. After applying this unification, we get the clauses $C_1\sigma = \{P(f(f(z))), Q(f(f(z)), f(z))\}$ and $C_2\sigma = \{\neg P(f(z)), \neg Q(f(f(z)), f(z))\}$, from which we derive the clause $C = \{P(f(f(z))), \neg P(f(z))\}$. This clause will be called the *resolvent* of the original clauses C_1 and C_2.

Definition 8.5.2 (Resolution Rule)**.** Let C_1 and C_2 be clauses with disjoint sets of variables and let them be of the form

$$C_1 = C_1' \sqcup \{A_1, \dots, A_n\}, \quad C_2 = C_2' \sqcup \{\neg B_1, \dots, \neg B_m\}$$

where $n, m \geq 1$ and the set of expressions $S = \{A_1, \dots, A_n, B_1, \dots, B_m\}$ is unifiable.[10] Let σ be the most general unification of S.[11] The *resolvent* of the clauses C_1 and C_2 is the following clause:

$$C = C_1'\sigma \cup C_2'\sigma$$

Remark 8.5.3. The condition about disjoint sets of variables can always be satisfied if we rename the variables in one of the clauses. Why is this necessary? For example, from the clauses $\{\{P(x)\}, \{\neg P(f(x))\}\}$ we can obtain the empty clause $\square$ if we replace the clause $\{P(x)\}$ with the clause $\{P(y)\}$. However, the set of expressions $\{P(x), P(f(x))\}$ is not unifiable, so this would not work without renaming variables.

8.5.2 Resolution Proof

Once we have defined the resolution rule, we can introduce the *resolution proof* and related notions. The definitions will be the same as in propositional logic, with one difference: we allow renaming of variables in the clauses, see Remark 8.5.3.

Definition 8.5.4 (Resolution Proof)**.** A *resolution proof (derivation, deduction)* of a clause C from a formula S is a *finite* sequence of clauses $C_0, C_1, \dots, C_n = C$ such that for each i,

- either $C_i = C_i'\sigma$ for some clause $C_i' \in S$ and renaming of variables σ, or

[10]The symbol $\sqcup$ denotes *disjoint union.*

[11]Recall that unification means that $A_1\sigma = A_2\sigma = \cdots = B_1\sigma = \cdots = B_m\sigma$.

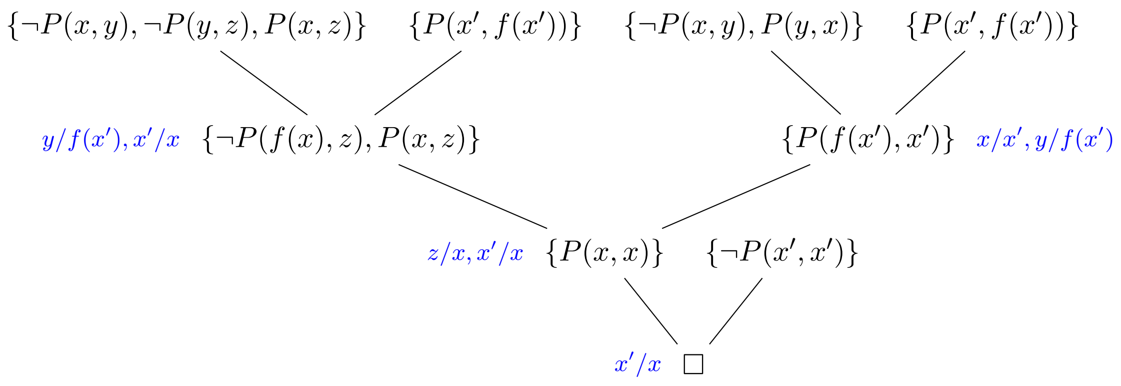


Figure 8.1: Resolution refutation of the formula S from Example 8.5.6. The unification used in each resolution step is noted.

- C_i is the resolvent of some C_j, C_k where $j < i$ and $k < i$.

If a resolution proof exists, we say that C is *provable by resolution* from S, and we write $S \vdash_R C$. A *resolution refutation* of the formula S is a resolution proof of $\square$ from S; in that case, S is *resolution refutable*.

Remark 8.5.5. Why do we need to eliminate multiple literals at once from one clause in the resolution step? Consider the formula $S = \{\{P(x), P(y)\}, \{\neg P(x), \neg P(y)\}\}$. It is resolution refutable, but there is no refutation that eliminates only one literal in each step.

Now we will show an example of application of the resolution method to prove the validity of a sentence.

Example 8.5.6. Let $T = \{\neg P(x,x), P(x,y) \to P(y,x), P(x,y) \wedge P(y,z) \to P(x,z)\}$ and let φ be the sentence $(\exists x)\neg P(x, f(x))$. We want to show that $T \models \varphi$. The theory $T \cup \{\neg\varphi\}$ is equisatisfiable (in this case, even equivalent) to the following CNF formula:

$$S = \{\{\neg P(x,x)\}, \{\neg P(x,y), P(y,x)\}, \{\neg P(x,y), \neg P(y,z), P(x,z)\}, \{P(x, f(x))\}\}$$

We show that $S \vdash_R \square$. The resolution proof is, for example, the following sequence:

$$\{\neg P(x,y), \neg P(y,z), P(x,z)\}, \{P(x', f(x'))\}, \{\neg P(f(x), z), P(x,z)\}, \{\neg P(x,y), P(y,x)\},$$
$$\{P(x', f(x'))\}, \{P(f(x'), x')\}, \{P(x,x)\}, \{P(x', x')\}, \square$$

However, a resolution tree, as shown in Figure 8.5.2, is more illustrative.

8.6 Soundness and Completeness

In this section, we will prove that the resolution method is both sound and complete in predicate logic, too.

8.6.1 Soundness Theorem

We begin with the proof of soundness of the resolution rule. The principle is the same as the analogous observation in propositional logic. The proof is a bit more technical:

Proposition 8.6.1 (Soundness of the Resolution Step). *Let C_1, C_2 be clauses, and C their resolvent. If the clauses C_1 and C_2 are valid in some structure $\mathcal{A}$, then C is also valid in it.*

Proof. From the definition of the resolution rule, we know that the clauses and their resolvent can be expressed as $C_1 = C_1' \sqcup \{A_1, \ldots, A_n\}$, $C_2 = C_2' \sqcup \{\neg B_1, \ldots, \neg B_m\}$, and $C = C_1'\sigma \cup C_2'\sigma$, where σ is the most general unification of the set of expressions $S = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$, i.e., $S\sigma = \{A_1\sigma\}$.

Since clauses C_1 and C_2 are open formulas valid in $\mathcal{A}$, their instances after substitution σ are also valid in $\mathcal{A}$, i.e., we have $\mathcal{A} \models C_1\sigma$ and $\mathcal{A} \models C_2\sigma$. We also know that $C_1\sigma = C_1'\sigma \cup \{A_1\sigma\}$ and similarly $C_2\sigma = C_2'\sigma \cup \{\neg A_1\sigma\}$.

Our goal is to show that $\mathcal{A} \models C[e]$ for any variable assignment e. If $\mathcal{A} \models A_1\sigma[e]$, then $\mathcal{A} \not\models \neg A_1\sigma[e]$ and it must be that $\mathcal{A} \models C_2'\sigma[e]$. Therefore, $\mathcal{A} \models C[e]$. Conversely, if $\mathcal{A} \not\models A_1\sigma[e]$, then $\mathcal{A} \models C_1'\sigma[e]$, and again $\mathcal{A} \models C[e]$. $\square$

The statement and proof of the Soundness Theorem are now the same as in propositional logic:

Theorem 8.6.2 (Soundness of Resolution). *If a CNF formula S is resolution refutable, then it is unsatisfiable.*

Proof. We know that $S \vdash_R \square$, so let us take some resolution proof of $\square$ from S. If there existed a model $\mathcal{A} \models S$, due to the soundness of the resolution rule, we could prove by induction on the length of the proof that $\mathcal{A} \models \square$, which is impossible. $\square$

8.6.2 Completeness Theorem

The completeness theorem for resolution in predicate logic, i.e., that unsatisfiable formulas are resolution refutable, will be proved by reducing to propositional logic. We will show that a resolution proof 'at the propositional logic level' can be 'lifted' to the predicate logic level.

The key is the following lemma, which provides such 'lifting' of one resolution step. Its proof is somewhat technical.

Lemma 8.6.3 (Lifting Lemma). *Let C_1 and C_2 be clauses with disjoint sets of variables. If C_1^* and C_2^* are ground instances of C_1 and C_2, respectively, and C^* is a resolvent of C_1^* and C_2^*, then there exists a resolvent C of C_1 and C_2 such that C^* is a ground instance of C.*

Proof. Let $C_1^* = C_1\tau_1$ and $C_2^* = C_2\tau_2$, where τ_1 and τ_2 are ground substitutions that share no variables. We find a resolvent C such that $C^* = C\tau_1\tau_2$.

Let C^* be a resolvent of C_1^* and C_2^* over the literal $P(t_1, \ldots, t_k)$. We know that the clauses C_1 and C_2 can be expressed as $C_1 = C_1' \sqcup \{A_1, \ldots, A_n\}$ and $C_2 = C_2' \sqcup \{\neg B_1, \ldots, \neg B_m\}$, where $\{A_1, \ldots, A_n\}\tau_1 = \{P(t_1, \ldots, t_k)\}$ and $\{\neg B_1, \ldots, \neg B_m\}\tau_2 = \{\neg P(t_1, \ldots, t_k)\}$.

This means that $(\tau_1\tau_2)$ unifies the set of expressions $S = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$. Now, take the most general unification σ of S obtained from the Unification Algorithm. Define C to be the resolvent $C = C_1'\sigma \cup C_2'\sigma$.

It remains to show that $C^* = C\tau_1\tau_2$. Using the 'moreover' property from Proposition 8.4.13 on the correctness of the Unification Algorithm, we know that $(\tau_1\tau_2) = \sigma(\tau_1\tau_2)$, which we use in the third equality of the following calculation. In the fourth equality, we use the fact that $C_1'\tau_1\tau_2 = C_1'\tau_1$, and $C_2'\tau_1 = C_2'$, which follows from the fact that these are ground substitutions that share no variables, and that $C_1'\tau_1$ and $C_2'\tau_2$ are ground instances:

$$\begin{aligned}
C\tau_1\tau_2 &= (C_1'\sigma \cup C_2'\sigma)\tau_1\tau_2 \\
&= C_1'\sigma\tau_1\tau_2 \cup C_2'\sigma\tau_1\tau_2 \\
&= C_1'\tau_1\tau_2 \cup C_2'\tau_1\tau_2 \\
&= C_1'\tau_1 \cup C_2'\tau_2 \\
&= (C_1 \setminus \{A_1, \dots, A_n\})\tau_1 \cup (C_2 \setminus \{\neg B_1, \dots, B_m\})\tau_2 \\
&= (C_1^* \setminus \{P(t_1, \dots, t_k)\}) \cup (C_2^* \setminus \{\neg P(t_1, \dots, t_k)\}) = C^*
\end{aligned}$$

$\square$

By induction on the length of the resolution proof, we easily obtain the following corollary:

Corollary 8.6.4. *Given a CNF formula S, let S^* denote the set of all its ground instances. If $S^* \vdash_R C^*$ ('at the propositional logic level') for some ground clause C^*, then there exists a clause C and a ground substitution σ such that $C^* = C\sigma$ and $S \vdash_R C$ ('at the predicate logic level').*

Now it is easy to prove completeness:

Theorem 8.6.5 (Completeness of Resolution). *If a CNF formula S is unsatisfiable, then it is resolution refutable.*

Proof. Let S^* denote the set of all ground instances of clauses from S. Since S is unsatisfiable, by Herbrand's theorem (specifically Corollary 8.3.9), S^* is also unsatisfiable. From the completeness theorem for *propositional* resolution, we know that $S^* \vdash_R \square$ ('at the propositional logic level'). From the Lifting Lemma (or rather from Corollary 8.6.4), we get a clause C and a ground substitution σ such that $C\sigma = \square$ and $S \vdash_R C$ ('at the predicate logic level'). But since the empty clause $\square$ is an instance of C, C must be the empty clause $\square$ as well. Thus, we have found a resolution refutation $S \vdash_R \square$. $\square$

8.7 LI-resolution

In this section, we recall the concepts of *linear and linear-input proofs*, *LI-resolution*, and its completeness for Horn formulas. The definitions and theorems are the same as in propositional logic (the only difference is that in proofs we can use *variants* of clauses from S), and the proof can be done by reduction to propositional logic again using Herbrand's theorem and the Lifting Lemma.

Definition 8.7.1 (Linear and LI Proof). A *linear proof* (by resolution) of a clause C from a formula S is a finite sequence

$$\begin{bmatrix} C_0 \\ B_0 \end{bmatrix}, \begin{bmatrix} C_1 \\ B_1 \end{bmatrix}, \dots, \begin{bmatrix} C_n \\ B_n \end{bmatrix}, C_{n+1}$$

where C_i are called *central* clauses, C_0 is the *initial* clause, $C_{n+1} = C$ is the *final* clause, B_i are *side* clauses, and the following hold:

- C_0 is a variant of a clause from S, for $i \leq n$ C_{i+1} is the resolvent of C_i and B_i,
- B_0 is a variant of a clause from S, for $i \leq n$ B_i is a variant of a clause from S or $B_i = C_j$ for some $j < i$.

A *linear refutation* of S is a linear proof of $\square$ from S.

An *LI-proof* is a linear proof in which each side clause B_i is a variant of a clause from S. If there is an LI-proof, we say that C is *LI-provable* from S, and write $S \vdash_{LI} C$. If $S \vdash_{LI} \square$, then S is *LI-refutable.*

In Remark 5.4.3, we noted that 'linear' resolution (based on linear proofs) is complete.[12] The same theorem holds for predicate resolution:

Theorem 8.7.2 (Completeness of Linear Resolution)**.** *A clause C has a linear proof from a CNF formula S if and only if it has a resolution proof from S (i.e., $S \vdash_R C$).*

Proof. From a linear proof, we can easily construct a resolution tree. The converse implication follows from Remark 5.4.3 and the Lifting Lemma (whose application preserves the linearity of the resolution proof). $\square$

8.7.1 Completeness of LI-resolution for Horn Formulas

Recall the terminology related to Hornness and programs: A *Horn clause* is a clause containing at most one positive literal. A *Horn formula* is a (finite or infinite) set of Horn clauses. A *fact* is a positive unit (Horn) clause, a *rule* is a (Horn) clause with exactly one positive and at least one negative literal, and a *goal* is a non-empty (Horn) clause with no positive literals. Rules and facts are called *program clauses.*

As in propositional logic, LI-resolution is complete for Horn formulas:

Theorem 8.7.3 (Completeness of LI-resolution for Horn Formulas)**.** *If a Horn formula T is satisfiable, and $T \cup \{G\}$ is unsatisfiable for some goal G, then $T \cup \{G\} \vdash_{LI} \square$, by an LI-refutation that starts with the goal G.*

Proof. It follows from the analogous theorem in propositional logic, from Herbrand's theorem, and from the Lifting Lemma. $\square$

8.7.2 Resolution in Prolog

Finally, we show the application of LI-resolution in the Prolog programming language. A *program* in Prolog is a Horn formula containing only *program clauses*, i.e., *rules* and *facts.*

Example 8.7.4. As an example, consider a simple program describing the family relationships of three individuals, described in Table 8.7.4. On the left side, we see the Prolog syntax, and on the right is the set notation of the corresponding clauses; the corresponding CNF formula will be denoted P.

The last line in the table is not a part of the program; it is an *existential query.* We are interested to know whether it holds in the program: $P \models (\exists X)son(charlie, X)$? Note that by negating the query, we obtain the *goal* $G = \{\neg son(charlie, X)\}$. Therefore, we want to *refute* the CNF formula $P \cup \{G\}$.

[12] The proof of the harder implication was omitted; it can be found in *A. Nerode, R. Shore: Logic for Applications* [3].

`son(X,Y):-father(Y,X),man(X).`	$\{son(X,Y), \neg father(Y,X), \neg man(X)\}$
`son(X,Y):-mother(Y,X),man(X).`	$\{son(X,Y), \neg mother(Y,X), \neg man(X)\}$
`man(charlie).`	$\{man(charlie)\}$
`father(bob,charlie).`	$\{father(bob, charlie)\}$
`mother(alice,charlie).`	$\{mother(alice, charlie)\}$
`?-son(charlie,X).`	$\{\neg son(charlie, X)\}$

Table 8.1: Example Prolog Program

As in propositional logic (Corollary 5.4.10), the completeness of LI-resolution for Horn formulas has the following simple corollary.

Corollary 8.7.5. *For a program P and a goal $G = \{\neg A_1, \dots, \neg A_k\}$ in variables $X_1, \dots, X_n$, the following conditions are equivalent:*

- $P \models (\exists X_1) \dots (\exists X_n)(A_1 \wedge \dots \wedge A_k)$
- $P \cup \{G\}$ *has an LI-refutation starting with the goal* G.

Proof. It is not hard to see that the program P is always a satisfiable Horn formula. The first condition is equivalent to the unsatisfiability of $P \cup \{G\}$. The equivalence then follows from the completeness of LI-resolution for Horn formulas (Theorem 8.7.3). $\square$

If the answer to the query is positive, we want to know the *output substitution* σ, i.e., the composition of unifications from individual resolution steps, restricted to the variables in G. We have that:

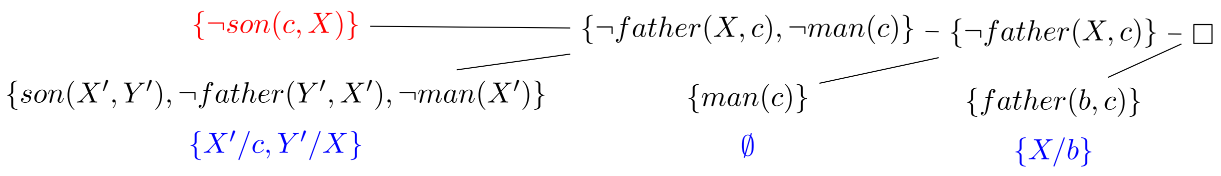
$$P \models (A_1 \wedge \dots \wedge A_k)\sigma$$

Example 8.7.6. Continuing Example 8.7.4, we find all output substitutions for our query:

```
?-son(charlie,X).
X = bob ;
X = alice ;
No
```

Which substitution we get depends on which of the two rules we apply to the goal. The respective refutations are shown below. The output substitution is obtained by composing the substitutions from individual steps and restricting it to the variable X. (For lack of space, we have abbreviated the constant symbols to a, b, c.)

(a) Output substitution $\sigma = \{X/b\}$:



(b) Output substitution $\sigma = \{X/a\}$:

$$\begin{tikzcd}
\{\neg son(c,X)\} \arrow[r, dash] & \{\neg mother(X,c), \neg man(c)\} \arrow[r, dash] & \{\neg mother(X,c)\} \arrow[r, dash] & \Box \\
\{son(X',Y'), \neg mother(Y',X'), \neg man(X')\} \arrow[ur, dash] & \{man(c)\} \arrow[ur, dash] & \{mother(a,c)\} \arrow[ur, dash] & \\
\{X'/c, Y'/X\} & \emptyset & \{X/a\} &
\end{tikzcd}$$

Bibliography

[1] Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. Springer London, June 2012. Google-Books-ID: hxOpugAACAAJ.

[2] Petr Gregor. Výroková a predikátová logika.

[3] Anil Nerode and Richard A. Shore. *Logic for Applications*. Springer Science & Business Media, December 2012. Google-Books-ID: 90HhBwAAQBAJ.

[4] Martin Pilát. Lecture Notes on Propositional and Predicate Logic. original-date: 2017-10-05T20:42:26Z.