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*:

$$M_\mathbb{P} = \{(0, 0), (0, 1), (1, 0), (1, 1)\}.$$

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

$$M_\mathbb{P}(\varphi) = M_\mathbb{P}(T) = \{(0, 1)\}.$$

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.

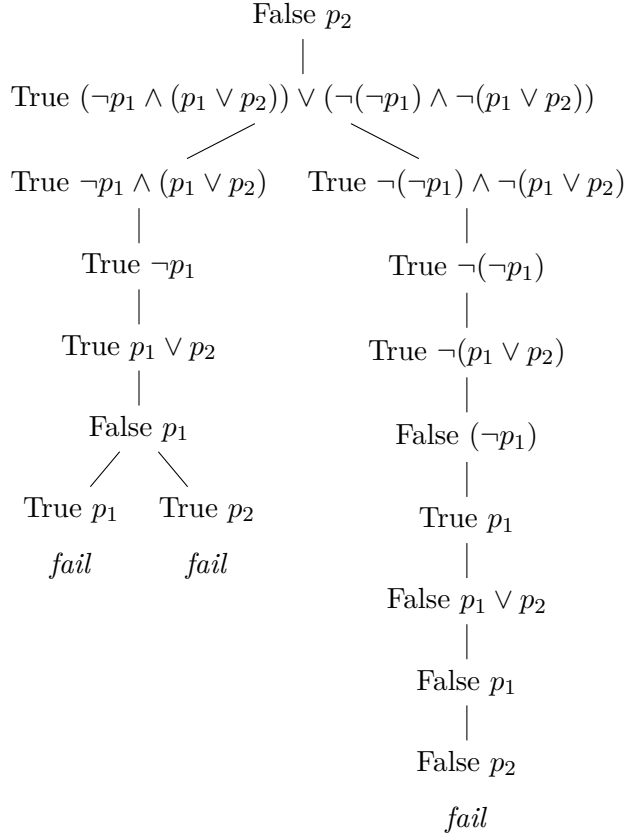[2]In practice, we commonly have thousands of variables.

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 **??**.

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 **??**).

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:

$$\{\neg p_1\} \quad \{p_1, p_2\}$$
$$\searrow \quad \swarrow$$
$$\{p_2\} \quad \{\neg p_2\}$$
$$\searrow \quad \swarrow$$
$$\square$$
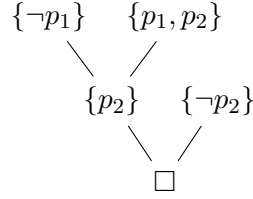
If the CNF formula had a model, the model would have to satisfy all of its clauses, and thus gradually all the resolvents in the sequence, and finally the empty clause. But no model can satisfy the empty clause (a disjunction of zero possibilities). We therefore have a proof by contradiction and thus know that we will find the treasure in the second corridor.

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

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.

---

[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.

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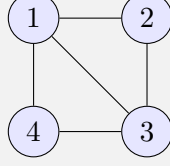
We represent the graph as a set of vertices and a set of edges, where each edge is a pair of vertices. It is easier to work with ordered pairs, so we choose an (arbitrary) orientation of the edges.

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

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 $|\,M_\mathbb{P}\,| = 2^{12} = 4096$ 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, $v = (1, 1, 0, 0, \ldots, 0)$ 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 $|\,M_\mathbb{P}(T_1)\,| = 4^4 = 2^8 = 256$ models. (Why?) If we want complete colorings, we add the condition that each vertex has at least one color.[4]

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

---

[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.

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 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.
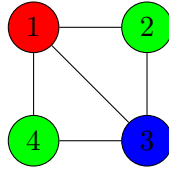
Once we have the theory T_3 formalizing 3-coloring of the graph $\mathcal{G}$, we can easily solve related questions, such as finding all colorings where vertex 1 is blue and vertex 2 is green: these correspond to the models of the theory $T_3 \cup \{p_1^B, p_2^G\}$. Or we can prove that vertices 2 and 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.*

---

[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.

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:
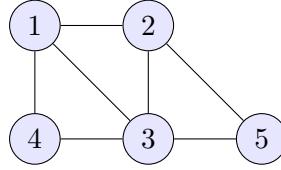
To formalize the new problem, we need to add three new propositional variables to our language: $\mathbb{P}' = \mathbb{P} \cup \{p_5^R, p_5^G, p_5^B\}$, and create new theories T_1', T_2', T_3' by adding axioms related to vertex 5 and edges $(2,5), (3,5)$. The issue here is that the structure of the 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:

---

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

- "There are no loops in the graph":

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

- "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]

$$(\forall S)((\exists x)S(x) \wedge (\exists x)(\forall y)(S(y) \to x \leq y) \to$$
$$(\exists x)((\forall y)(S(y) \to x \leq y) \wedge (\forall z)((\forall y)(S(y) \to z \leq y) \to z \leq x)))$$

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).

---

[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.

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:

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

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 $\iota \colon \{0, 1, \ldots, n-1\} \rightarrow \mathbb{P}$ (for a finite, n-element language) or $\iota \colon \mathbb{N} \rightarrow \mathbb{P}$ (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 $\land$, we write first the symbol '(', then the first proposition, the symbol '$\land$', 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 $\varphi, \psi \in \mathrm{PF}_{\mathbb{P}}$, $(\varphi \land \psi)$, $(\varphi \lor \psi)$, $(\varphi \to \psi)$, 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 $\square \in \{\land, \lor, \to, \leftrightarrow\}$, $(\varphi \square \psi)$ 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 $\varphi = ((p \lor (\neg q)) \leftrightarrow (r \to (p \land q)))$ has the following subpropositions: $p, q, (\neg q), (p \lor (\neg q)), r, (p \land q), (r \to (p \land q)), \varphi$.

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: $\top = (p \lor (\neg p))$ (*tautology*) and $\bot = (p \land (\neg p))$ (*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 $p \lor \neg q \leftrightarrow (r \to p \land q)$. We omit outer parentheses and use priority of operators: $\neg$ has the highest priority, followed by $\land, \lor$, and finally $\to, \leftrightarrow$ have the lowest priority. Furthermore, the notation $p \land q \land r \land s$ means the proposition $(p \land (q \land (r \land s)))$, and similarly for $\lor$.[56]

---

[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 $\land, \lor$, the placement of parentheses does not matter.

[6]Sometimes finer priorities are introduced, $\land$ often has higher priority than $\lor$, and $\to$ has higher priority than $\leftrightarrow$. Also, sometimes $p \to q \to r$ is written instead of $(p \to (r \to q))$, although $\to$ is not associative and so here the placement of parentheses does matter. We prefer to avoid both of those conventions.
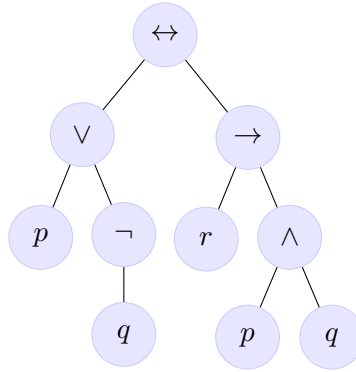
Figure 2.1: Tree of the proposition $\varphi = ((p \lor (\neg q)) \leftrightarrow (r \to (p \land q)))$

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 $\neg\varphi$, $\land\varphi\psi$, $\lor\varphi\psi$, $\to\varphi\psi$, and $\leftrightarrow\varphi\psi$ are also propositions.

The proposition $\varphi = ((p \lor (\neg q)) \leftrightarrow (r \to (p \land q)))$ would then be written as $\varphi = {\leftrightarrow}{\lor}p\neg q{\to}r{\land}pq$. We could also use *postfix* notation and write $\varphi = pq\neg \lor rpq\land{\to}\leftrightarrow$. 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 $\varphi = ((p \lor (\neg q)) \leftrightarrow (r \to (p \land q)))$ 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 $t = 0, 1, 2, \dots$. 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.
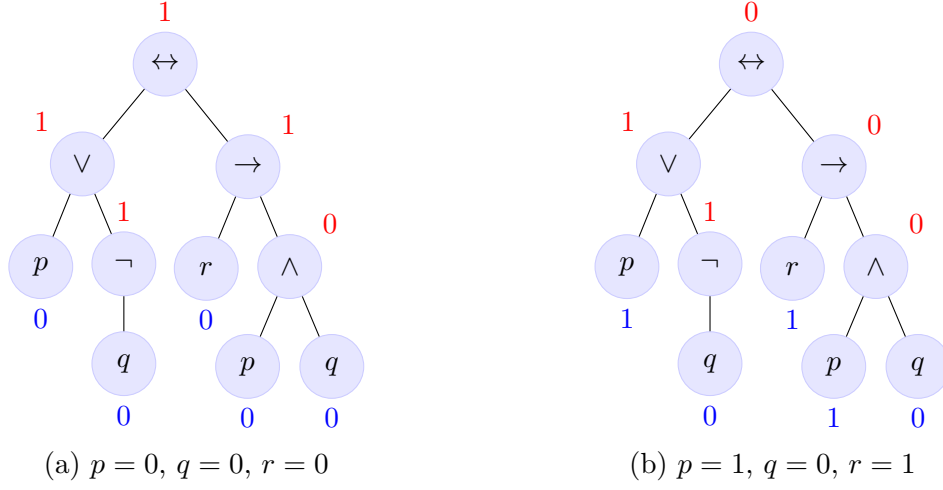
(a) $p = 0$, $q = 0$, $r = 0$ (b) $p = 1$, $q = 0$, $r = 1$

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 $f_{\varphi,\mathbb{P}}(x_0, \ldots, x_{n-1}) = x_i$,

- If $\varphi = (\neg \varphi')$, then

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

$$f_\wedge(x,y)\colon \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array} \quad f_\vee(x,y)\colon \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 1 \end{array} \quad f_\to(x,y)\colon \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 1 & 1 \\ 1 & 0 & 1 \end{array} \quad f_\leftrightarrow(x,y)\colon \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 1 & 0 \\ 1 & 0 & 1 \end{array}$$

Table 2.2: Boolean functions of logical connectives

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

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

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* $\mathbb{P} = \{0, 1, 2, \ldots\}$ 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:

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

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 $T = \{p \lor q \lor r, q \to r, \neg r\}$ (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.

- $(0,0,0) \models q \to r,$

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

- $(1,0,0) \models q \to r,$

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

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

$$M_{\mathbb{P}}(r, q \to r, p \vee q \vee r) = M_{\mathbb{P}}(T) = \{(1,0,0)\}.$$

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 $\mathbb{P}$), $M_{\mathbb{P}}(\varphi) = M_{\mathbb{P}}$,

- *false*, *contradictory*, if it has no model, $M_{\mathbb{P}}(\varphi) = \emptyset$,[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, $\emptyset \subsetneq M_{\mathbb{P}}(\varphi) \subsetneq M_{\mathbb{P}}$,

- *satisfiable*, if it has some model, i.e., it is not false, $M_{\mathbb{P}}(\varphi) \neq \emptyset$.

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.,

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

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:

    - $p \sim p \vee p \sim p \vee p \vee p,$
    - $p \to q \sim \neg p \vee q,$

---

[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 $\mathbb{P} = \{p_1, \ldots, p_n\}$. 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 $\{\text{NAND}\}$ and $\{\text{NOR}\}$ 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. $\square$

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 $\mathbb{P}' = \text{Var}(\varphi)$ and the set of models $M = \text{M}_{\mathbb{P}'}(\varphi)$. Since $M = \text{M}_{\mathbb{P}'}(\varphi_{\text{DNF}}) = \text{M}_{\mathbb{P}'}(\varphi_{\text{CNF}})$, we have $\varphi \sim \varphi_{\text{DNF}} \sim \varphi_{\text{CNF}}$. $\square$

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 $\mathbb{P}' = \text{Var}(T)$ (i.e., all propositional variables occurring in the axioms of T). If T in the language $\mathbb{P}'$ has finitely many models (i.e., $\text{M}_{\mathbb{P}'}(T)$ 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 $\varphi \rightarrow \psi \sim \neg \varphi \vee \psi$:

$$p \leftrightarrow (q \vee \neg r) \sim (p \rightarrow (q \vee \neg r)) \wedge ((q \vee \neg r) \rightarrow p)$$
$$\sim (\neg p \vee q \vee \neg r) \wedge (\neg(q \vee \neg r) \vee p)$$

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

28

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., $E = (p \wedge \neg r)$ and $E' = (p \wedge q \wedge \neg r)$. (Think about why, and formulate the dual simplification when converting to CNF.) The resulting proposition in DNF is:

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

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:

$$\varphi \to \psi \sim \neg \varphi \vee \psi$$
$$\varphi \leftrightarrow \psi \sim (\neg \varphi \vee \psi) \wedge (\neg \psi \vee \varphi)$$

- Negations:

$$\neg(\varphi \wedge \psi) \sim \neg \varphi \vee \neg \psi$$
$$\neg(\varphi \vee \psi) \sim \neg \varphi \wedge \neg \psi$$
$$\neg \neg \varphi \sim \varphi$$

- Conjunctions (conversion to DNF):

$$\varphi \wedge (\psi \vee \chi) \sim (\varphi \wedge \psi) \vee (\varphi \wedge \chi)$$
$$(\varphi \vee \psi) \wedge \chi \sim (\varphi \wedge \chi) \vee (\psi \wedge \chi)$$

- Disjunctions (conversion to CNF):

$$\varphi \vee (\psi \wedge \chi) \sim (\varphi \vee \psi) \wedge (\varphi \vee \chi)$$
$$(\varphi \wedge \psi) \vee \chi \sim (\varphi \vee \chi) \wedge (\psi \vee \chi)$$

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:

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

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 $T' = \{(\neg p \vee q) \wedge (\neg p \vee r) \wedge (p \vee \neg r)\}$.

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 $\varphi, \varphi_1, \ldots, \varphi_n$ 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) $\varphi \in \mathrm{Csq}_{\mathbb{P}}(\{\varphi_1, \ldots, \varphi_n\})$ if and only if the proposition $(\varphi_1 \wedge \cdots \wedge \varphi_n) \to \varphi$ 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.

- $\psi \rightarrow \varphi$ is a tautology if and only if $\mathrm{M}(\psi) \subseteq \mathrm{M}(\varphi)$,

- $\mathrm{M}(\varphi_1 \wedge \cdots \wedge \varphi_n) = \mathrm{M}(\varphi_1, \ldots, \varphi_n)$.

$\square$

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 $2^{|M_{\mathbb{P}}|} = 2^8 = 256$. 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 $|M_{\mathbb{P}}(T)| = k = 5$.

We count the sets $M = M_{\mathbb{P}}(\varphi)$ and require that $\emptyset \neq M \cap M_{\mathbb{P}}(T) \neq M_{\mathbb{P}}(T)$. Thus, we have a total of $2^k - 2 = 30$ possibilities for what the set $M \cap M_{\mathbb{P}}(T)$ 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 $h : \mathrm{PF}_{\mathbb{P}}/\sim \to \mathcal{P}(M_{\mathbb{P}})$ (where $\mathcal{P}(X)$ is the set of all subsets of X) defined by:

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

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:

$$\neg[\varphi]_{\sim} = [\neg\varphi]_{\sim}$$
$$[\varphi]_{\sim} \wedge [\psi]_{\sim} = [\varphi \wedge \psi]_{\sim}$$
$$[\varphi]_{\sim} \vee [\psi]_{\sim} = [\varphi \vee \psi]_{\sim}$$

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

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

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:

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

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 **??**, 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 $h_{i,j}, v_{i,j}$ ($1 \le i, j \le n$), 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 $1 \le i \le n$

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

$$h_{i,j-1} \vee h_{i,j} \vee v_{i-1,j} \vee v_{i,j} \text{ for } 1 < i, j \le n$$
$$h_{1,j-1} \vee h_{1,j} \vee v_{1,j} \text{ for } 1 < j \le n$$
$$h_{i,1} \vee v_{i-1,1} \vee v_{i,1} \text{ for } 1 < i \le n$$

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

$$(\neg h_{i,j-1} \vee \neg h_{i,j}) \wedge (\neg h_{i,j-1} \vee \neg v_{i-1,j}) \wedge (\neg h_{i,j-1} \vee \neg v_{i,j}) \wedge$$
$$(\neg h_{i,j} \vee \neg v_{i-1,j}) \wedge (\neg h_{i,j} \vee \neg v_{i,j}) \wedge (\neg v_{i-1,j} \vee \neg v_{i,j}) \text{ for } 1 < i, j \le n$$
$$(\neg h_{1,j-1} \vee \neg h_{1,j}) \wedge (\neg h_{1,j-1} \vee \neg v_{1,j}) \wedge (\neg h_{1,j} \vee \neg v_{1,j}) \text{ for } 1 < j \le n$$
$$(\neg h_{i,1} \vee \neg v_{i-1,1}) \wedge (\neg h_{i,1} \vee \neg v_{i,1}) \wedge (\neg v_{i-1,1} \vee \neg v_{i,1}) \text{ for } 1 < i \le n$$

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:

$$(\neg p_1 \lor p_2) \land (\neg p_2 \lor \neg p_3) \land (p_1 \lor p_3) \land (p_3 \lor \neg p_4) \land (\neg p_1 \lor p_5) \land (p_2 \lor p_5) \land p_1 \land \neg p_4$$

Implication Graph

The implication graph of a 2-CNF proposition φ is based on the idea that a 2-clause $\ell_1 \lor \ell_2$ (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 $\neg p_1 \lor p_2$, 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 $\mathrm{Var}(\varphi)$ and their negations), and edges are given by the implications described above:

- $V(\mathcal{G}_\varphi) = \{p, \neg p \mid p \in \mathrm{Var}(\varphi)\}$,

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

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:

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

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 $\neg p_1 \lor p_2$; 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.
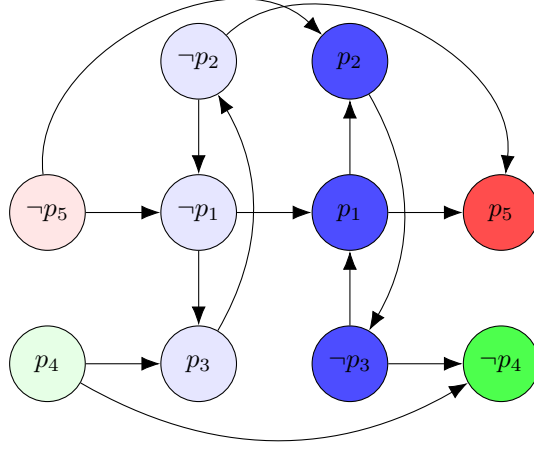
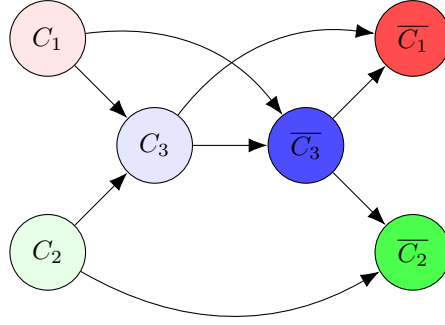Figure 3.1: Implication graph $\mathcal{G}_\varphi$. Strongly connected components are distinguished by colors.

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 $\ell, \bar{\ell}$.*

Proof. Every model, i.e., valid truth assignment, must assign all literals in the same compo-
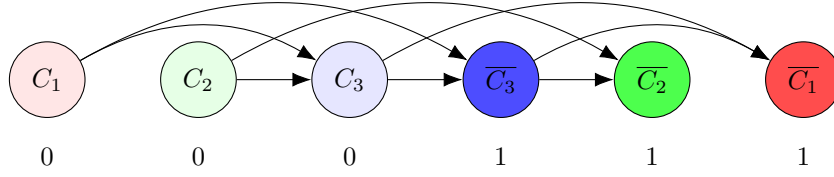
Figure 3.3: Implication graph $\mathcal{G}_\varphi$. Topological ordering of the graph $\mathcal{G}_\varphi^*$ and satisfying assignment of the components.

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 $\bar{\ell} \to \ell$ in the graph $\mathcal{G}_\varphi$. The same edge is also in the graph of components, so $\bar{\ell}$ precedes the component containing ℓ in the topological ordering. In constructing the model, we must have assigned $\bar{\ell} = 0$, 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 $\ell_2 = 1$. $\square$

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. $\square$

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 **??** and Section **??**.

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.

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.