Chapter 1

Introduction to Logic

The word *logic* is used in two senses:

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

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

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

1.1 Propositional Logic

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

1.1.1 Example: Treasure Hunting

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

1.1.2 Formalization in Propositional Logic

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

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

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

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

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

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

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

1.1.3 Models and Consequences

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

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

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

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

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

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

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

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

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

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

1.1.4 Proof Systems

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

We want two properties from a proof system:

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

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

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

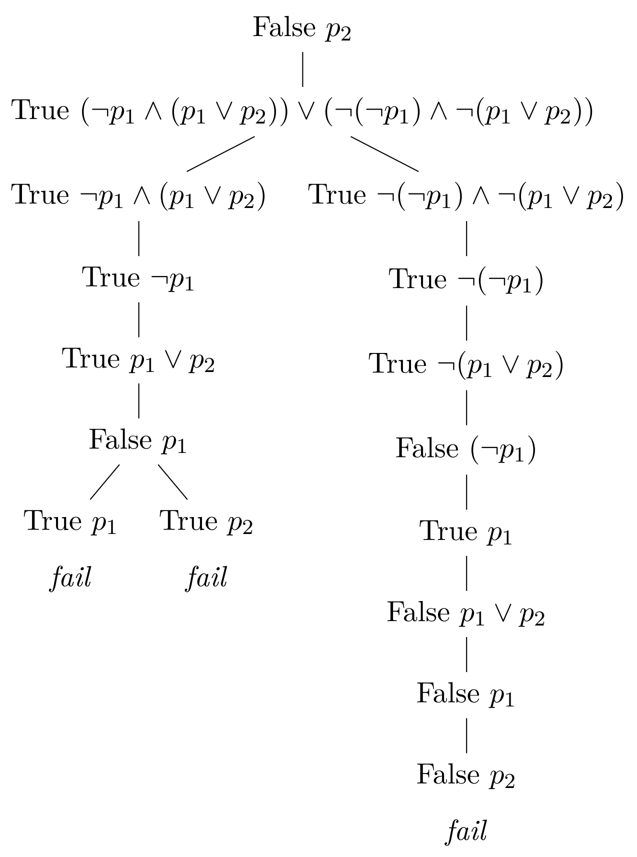

False p_2
True $(\neg p_1 \wedge (p_1 \vee p_2)) \vee (\neg(\neg p_1) \wedge \neg(p_1 \vee p_2))$
True $\neg p_1 \wedge (p_1 \vee p_2)$
True $\neg(\neg p_1) \wedge \neg(p_1 \vee p_2)$
True $\neg p_1$
True $p_1 \vee p_2$
False p_1
True p_1
fail
True p_2
fail
True $\neg(\neg p_1)$
True $\neg(p_1 \vee p_2)$
False $(\neg p_1)$
True p_1
False $p_1 \vee p_2$
False p_1
False p_2
fail

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

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

1.1.5 Tableau Method

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

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

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

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

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

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

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:

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

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

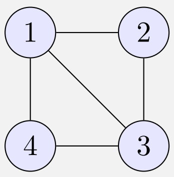
The details of the resolution method will be presented in Chapter **??**.

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

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

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

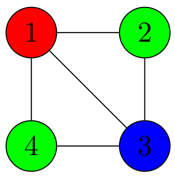
$$\begin{aligned}
T_2 &= T_1 \cup \{p_1^R \vee p_1^G \vee p_1^B, p_2^R \vee p_2^G \vee p_2^B, p_3^R \vee p_3^G \vee p_3^B, p_4^R \vee p_4^G \vee p_4^B\}\\
&= T_1 \cup \{p_v^R \vee p_v^G \vee p_v^B \mid v \in V\}\\
&= T_1 \cup \{\bigvee_{c\in C} p_v^c \mid v \in V\}
\end{aligned}$$

[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:

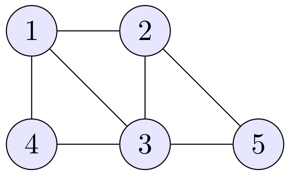
All humans are mortal.
Socrates is a human.
Socrates is mortal.

[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":

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

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:

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

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":

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

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

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

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.