"SAPIENZA" UNIVERSITY OF ROME
FACULTY OF INFORMATION ENGINEERING, INFORMATICS AND STATISTICS
DEPARTMENT OF COMPUTER SCIENCE

# Mathematical Logic for Computer Science

Lecture notes integrated with the book "Introduction to Mathematical Logic", E. Mendelson

*Author*
Simone Bianco

March 31, 2025

# Contents

# Information and Contacts

Personal notes and summaries collected as part of the *Mathematical Logic for Computer Science* course offered by the degree in Computer Science of the University of Rome "La Sapienza".

Further information and notes can be found at the following link:
**https://github.com/Exyss/university-notes**. Anyone can feel free to report inaccuracies, improvements or requests through the Issue system provided by GitHub itself or by contacting the author privately:

- Email: **bianco.simone@outlook.it**
- LinkedIn: **Simone Bianco**

The notes are constantly being updated, so please check if the changes have already been made in the most recent version.

**Suggested prerequisites:**

Sufficient knowledge of logic and theoretical computer science

**Licence:**

# 1 Propositional logic

## 1.1 Introduction to logical constructs

Logic is concerned with the validity of reasoning independently of the meaning of its components. In particular,**propositional logic** deals with studying the properties of certain logical constructs used in natural language as well as in scientific and mathematical practice. In particular, we're interested in the following constructs:

- *Negation* ($\neg$): This operation inverts the truth value of a statement. If a statement is true, its negation is false, and vice versa.
- *Conjunction* ($\wedge$): This operation represents the logical "and". The result is true if and only if both components are true.
- *Disjunction* ($\vee$): This operation represents the logical "or". The result is true if at least one of the components is true.
- *Implication* ($\rightarrow$): This operation represents the logical "if ... then" statement. The result is false only if the first component is true and the second component is false. In other words, this operator forces that when the premise holds, the conclusion must always follow (if the premise is false, we don't care about the truthfulness of the consequence).
- *Equivalence* ($\leftrightarrow$): This operation represents the logical "if and only if" statement. The result is true when both components have the same truth value, i.e. they're either both true or both false.

For instance, consider the following simple arithmetic argument:

1. If $a = 0$ or $b = 0$, then $a \cdot b = 0$
2. $a \cdot b \neq 0$
3. $a \neq 0$ and $b \neq 0$

Intuitively, the third proposition "follows" from the other two propositions, i.e. it is the conclusion of an argument with the first two as premises. To formalize this argument, we first identify those parts that cannot be further analyzed in terms of logical constructs and that can be true or false (called **atomic parts**). In our case, these atomic parts are $a = 0$, $b = 0$, and $a \cdot b = 0$.

We assign a distinct letter to each of these atomic parts: for $a = 0$, we associate $A$; for $b = 0$, we associate $B$; and for $a \cdot b = 0$, we associate $C$. Then, we replace the logical constructs in natural language (if ... then, or, and, not) with formal symbols, called Boolean connectives: $\neg$ for negation, $\vee$ for disjunction, $\wedge$ for conjunction, $\rightarrow$ for implication (if ... then), and $\leftrightarrow$ for equivalence (if and only if).

We obtain the following formalization:

1. $(A \vee B) \rightarrow C$
2. $\neg C$
3. $\neg A \wedge \neg B$

where we read $A \vee B$ as "A or B", $\neg C$ as "not C" and so on. Consider now the following verbal argument:

1. If the father is tall or the mother is tall, then the child is tall
2. The child is short
3. The father is short and the mother is short

If we attempt a formalization of the argument, we quickly realize that we obtain the same formalization as the previous argument, where we write $A$ for "the father is tall", $B$ for "the mother is tall", and $C$ for "the child is tall". Therefore, the two arguments are identical. Formal logic allows us to identify the common logical structure in arguments that deal with different objects and structures, as is the case in the examples above. However, in this case the first premise is obviously known to be empirically false. Nevertheless, if we assume it as premise for an argument, we intuitively recognize that the reasoning is **valid** (or correct). When we say the argument is valid (or correct), we mean that if the premises are true, then the conclusion is also true; not that the premises are true.

To evaluate the truthfulness of a propositional statement, we use **truth tables**. In logic, a truth table is a table used to determine whether a logical expression is true or false based on the **truth values** of its components. Truth tables are primarily used to evaluate the validity of logical propositions and determine the relationships between various logical statements. Each row of a truth table represents a unique combination of truth values for the variables involved in the logical expression. The columns show the intermediate truth values of the components of the expression, ultimately leading to the evaluation of the entire logical formula.

| $A$ | $\neg A$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

| $A$ | $B$ | $A \land B$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| $A$ | $B$ | $A \lor B$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| $A$ | $B$ | $A \to B$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| $A$ | $B$ | $A \leftrightarrow B$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Figure 1.1: Truth tables of the five logical operators previously introduces. The value 0 represents the falseness of the propositions, while the value 1 represents their truthfulness.

Consider now a more complex logical expression $A \to (B \land C)$. A truth table for this expression requires evaluating the truth values for $A$, $B$ and $C$, then determining the truth value of the conjunction $B \land C$, and finally computing the implication $A \to (B \land C)$.

| $A$ | $B$ | $C$ | $B \land C$ | $A \to (B \land C)$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Figure 1.2: Truth table for the logical expression $A \to (B \land C)$

## 1.2 Mathematical formalization

The first step is to define a completely rigorous formal language that can be subjected to mathematical study. For propositional logic, the procedure is simple: propositions are formalized by formulas obtained by applying Boolean connectives to some atomic building blocks, called *propositional variables*. These variables intuitively represent propositions that cannot be further analyzed in terms of logical operators such as negation, conjunction, disjunction, implication, or equivalence.

**Definition 1.1: Language and propositions**

A propositional language is a set $\mathcal{L} = \{p_1, \dots, p_n\}$, where each $p_i$ is a symbol called propositional variable. Given a propositional language $\mathcal{L}$, the set of propositions (or valid formulas) over $\mathcal{L}$, denoted with $\mathsf{PROP}_{\mathcal{L}}$ (or $\mathsf{FML}_{\mathcal{L}}$), is the minimal set of <u>finite</u> expressions over $\mathcal{L} \cup \{\neg, \land, \lor, \to, \leftrightarrow\}$ such that:

- Each propositional variable in $\mathcal{L}$ is a proposition, i.e. $\mathcal{L} \subseteq \mathsf{PROP}_{\mathcal{L}}$
- If $A \in \mathsf{PROP}_{\mathcal{L}}$ then $\neg A \in \mathsf{PROP}_{\mathcal{L}}$
- If $A, B \in \mathsf{PROP}_{\mathcal{L}}$ then $(A \land B), (A \lor B), (A \to B), (A \leftrightarrow B) \in \mathsf{PROP}_{\mathcal{L}}$

When $\mathcal{L}$ is non-ambiguous in the context, we denote $\mathsf{PROP}_{\mathcal{L}}$ directly as $\mathsf{PROP}$. Moreover, to make expressions easier to read, we use the following precedence rules: $\neg$ has precedence over $\land, \lor$, while $\land, \lor$ have precedence over $\to, \leftrightarrow$. In other words, we have that $(\neg((\neg A) \lor B)) \to C$ can be written as $\neg(\neg A \lor B) \to C$.

After formalizing the concept of propositional language and propositions, we're ready to formalize the concept of truth tables. Each row of a truth table can be described through **assignments**. Here, an assignment is any function $\alpha : \mathcal{L} \to \{0, 1\}$, i.e. a map that assigns a truth value to each propositional variable of the language. By extension, any assignment over $\mathcal{L}$ induces an assignment over $\mathsf{PROP}$. These assignments are functions $\widehat{\alpha} : \mathsf{PROP} \to \{0, 1\}$ inductively defined as:

$$\widehat{\alpha}(A) = \alpha(A) \text{ when } A \in \mathcal{L}$$

$$\widehat{\alpha}(\neg A) = \begin{cases} 1 & \text{if } \widehat{\alpha}(A) = 0 \\ 0 & \text{if } \widehat{\alpha}(A) = 1 \end{cases}$$

$$\widehat{\alpha}(A \land B) = \begin{cases} 1 & \text{if } \widehat{\alpha}(A) = \widehat{\alpha}(B) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\widehat{\alpha}(A \lor B) = \begin{cases} 0 & \text{if } \widehat{\alpha}(A) = \widehat{\alpha}(B) = 0 \\ 1 & \text{otherwise} \end{cases}$$

$$\widehat{\alpha}(A \to B) = \begin{cases} 0 & \text{if } \widehat{\alpha}(A) = 1 \text{ and } \widehat{\alpha}(B) = 0 \\ 1 & \text{otherwise} \end{cases}$$

$$\widehat{\alpha}(A \leftrightarrow B) = \begin{cases} 1 & \text{if } \widehat{\alpha}(A) = \widehat{\alpha}(B) \\ 0 & \text{otherwise} \end{cases}$$

To make notation easier, we'll directly refer to $\widehat{\alpha}$ as $\alpha$. Using the concept of assignments, we can also define the concept of **logical equivalence**. Informally, two formulas are said to be logically equivalent when they share the same exact truth table.

**Definition 1.2: Logical equivalence**

We say that two formulas $A, B \in \mathsf{PROP}$ are logically equivalent, denoted with $A \equiv B$, when for every assignment $\alpha$ it holds that $\alpha(A) = \alpha B$.

By definition, it clearly holds that $\equiv$ is an equivalence relation since it is reflexive, symmetric and transitive. Logical equivalences allow us to treat logical components as if they were algebraic operators. For instance, common algebraic properties include:

- *Involution*: $\neg\neg A \equiv A$
- *Idempotency*: $A \lor A \equiv A$ and $A \land A \equiv A$
- *Associativity*: $(A \lor B) \lor C \equiv A \lor (B \lor C)$ and $(A \land B) \land C \equiv A \land (B \land C)$
- *Commutativity*: $A \lor B \equiv B \lor A$ and $A \land B \equiv B \land A$
- *Distributivity*: $A \lor (B \land C) \equiv (A \lor B) \land (A \lor C)$ and $A \land (B \lor C) \equiv (A \land B) \lor (A \land C)$
- *De Morgan's law*: $\neg(A \lor B) \equiv \neg A \land \neg B$ and $\neg(A \land B) \equiv \neg A \lor \neg B$

Through these basic properties, we notice that each connective can be rewritten in terms of the other connectives. In fact, we could define propositional logic using only $\neg$ and $\land$ (or $\neg$ and $\lor$). This property is very useful in circuit logic.

- $A \land B \equiv \neg(\neg A \lor \neg B)$
- $A \lor B \equiv \neg(\neg A \land \neg B)$
- $A \to B \equiv \neg A \lor B$
- $A \leftrightarrow B \equiv (A \to B) \land (B \to A)$

If for an assignment $\alpha$ and a formula $A$ it holds that $\alpha(A) = 1$, we say that $\alpha$ *satisfies* $A$. If $A$ has at least one satisfying assignment, we say that $A$ is **satisfiable**. When no satisfying assignment exists, we say that the formula $A$ is **unsatisfiable**. On the contrary, when every possible assignment satisfies $A$ we say that $A$ is a **tautology** (or *logical truth*, *logically valid*). By definition, tautologies represent statements that are always true, no matter the situation. We denote with $\mathsf{SAT}, \mathsf{UNSAT}$ and $\mathsf{TAUT}$ the sets of satisfiable, unsatisfiable and tautological formulas. By definition, we have that $F \in \mathsf{TAUT}$ if and only if $\neg F \in \mathsf{UNSAT}$. If a formula is not a tautology, we cannot ensure the truthfulness of the statement. Hence, we consider *true statements* only as formulas that are *always true* (the set $\mathsf{TAUT}$), while we consider *false statement* as formulas that are *not always true* (the set $\mathsf{UNSAT} \cup (\mathsf{SAT} - \mathsf{TAUT})$).

With a slight abuse of notation, we define 1 and 0 as the formulas such that for every assignment it holds that $\alpha(1) = 1$ and $\alpha(0) = 0$. This allows us to further simplify logical equivalences algebraically:

- *Absorption law*: $A \lor 0 \equiv A$ and $A \land 1 \equiv A$
- *Cancellation law*: $A \lor 1 \equiv 1$ and $A \land 0 \equiv 0$
- *Tertium non datur*: $A \lor \neg A \equiv 1$ and $A \land \neg A \equiv 0$

Using all the algebraic rules that we have shown, we can easily evaluate propositional formulas by first reducing them to a smaller equivalent formula and then (eventually) by evaluating the truth table of the reduced form:

$$\begin{aligned} A \lor B \to B \lor \neg C &\equiv \neg(A \lor B) \lor B \lor \neg C \\ &\equiv (\neg A \land \neg B) \lor B \lor \neg C \\ &\equiv ((\neg A \lor B) \land (\neg B \lor B)) \lor \neg C \\ &\equiv ((\neg A \lor B) \land 1) \lor \neg C \\ &\equiv \neg A \lor B \lor \neg C \end{aligned}$$

The concepts of tautology and unsatisfiability are strictly related with the concept of **logical consequence** that we have discussed in the previous section.

**Definition 1.3: Logical consequence**

Given the formulas $A_1, \dots, A_n, A$, we say that $A$ is a logical consequence of $A_1, \dots, A_n$, written as $A_1, \dots, A_n \models A$ if whenever $A_1, \dots, A_n$ are true $A$ is also true.

Through truth tables, evaluating if we can conclude $A$ through $A_1, \dots, A_n$ is pretty easy: $A_1, \dots, A_n \models A$ if and only if for every row where $A_1, \dots, A_n$ are evaluated as 1 we also have that $A$ is evaluated as 1. For instance consider the *Logician party* problem, made of the following propositions:

1. If the Logician party wins the elections then the taxes will increase if the deficit remains high
2. If the Logician party wins the elections, the deficit remains high
3. If the Logician party wins the elections, the taxes will increase

We want to know if the last proposition is a logical consequence of the other two propositions. First, we define the following propositional variables:

- Let $P$ be the variable such that $P = 1$ if and only if the Logician party wins the elections
- Let $T$ be the variable such that $T = 1$ if and only if the taxes will increase
- Let $D$ be the variable such that $D = 1$ if and only if the deficit remains high

Using the three variables, we formalize the three propositions as follows:

1. $P \to (D \to T) \equiv \neg P \lor (\neg D \lor T) \equiv \neg P \neg D \lor T$
2. $P \to D \equiv \neg P \lor D$
3. $P \to T \equiv \neg P \lor D$

Then, we compute the truth table for the problem:

| $P$ | $T$ | $D$ | $\neg P \neg D \vee T$ | $\neg P \vee D$ | $\neg P \vee T$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

Figure 1.3: Truth table for the Logician party problem.

We observe that for every assignment $\alpha$ such that $\alpha(P \to (D \to T)) = 1$ and $\alpha(P \to D) = 1$, it also holds that $\alpha(P \to T) = 1$. Hence, we can indeed conclude that $P \to (D \to T), P \to D \models P \to T$. Since the number of rows in a truth table is exponential with respect to the number of propositional variables, enumerating the whole truth table may be too expensive. For instance, with only 4 variables we would have to enumerate $2^4 = 16$ rows. To help with this issue, the following theorem comes in handy.

**Theorem 1.1**

Given the formulas $A_1, \ldots, A_n, A$, the three following statements are equivalent:

1. $A_1, \ldots, A_n \models A$
2. $(A_1 \wedge \ldots \wedge A_n \to A) \in \mathsf{TAUT}$
3. $(A_1 \wedge \ldots \wedge A_n \wedge \neg A) \in \mathsf{UNSAT}$

*Proof.* Omitted (trivial) □

Hence, in order to evaluate if $A_1, \ldots, A_n \models A$ holds, we can show that $(A_1 \wedge \ldots \wedge A_n \to A) \equiv 1$ or that $(A_1 \wedge \ldots \wedge A_n \wedge \neg A) \equiv 0$ through the algebraic properties shown before.

For instance, consider the *self-destruction button* problem. We must travel far away into the galaxy and Adam lends us his spaceship. Adams tells us that the spaceship has three buttons and that one and only one of them triggers the self-destruction protocol, while the others do nothing. Moreover, each of the three buttons has a post-it attached to it. The first and second button say "I'm not the self-destruction button", while the third button says "The first button is the self-destruction button". Adam tells us that one and only one of the buttons says the truth. Using pure logic, we have to find out which button is the self-destruction one in order to avoid blowing up ourselves.

First, we define three propositional variables:

- Let $A$ be the variable such that $A = 1$ if and only if the first button triggers the self-destruction protocol

- Let $B$ be the variable such that $B = 1$ if and only if the second button triggers the self-destruction protocol
- Let $C$ be the variable such that $C = 1$ if and only if the third button triggers the self-destruction protocol

Then, we formalize the entire problem through the following two propositions $\phi, \psi$:

1. One and only one of the buttons triggers the self-destruction protocol, i.e.

$$\phi = (A \lor B \lor C) \land \neg(A \land B) \land \neg(A \land C) \land \neg(B \land C)$$

2. One and only one of the buttons says the truth, i.e.

$$\psi = (\neg A \lor \neg B \lor A) \land \neg(\neg A \land \neg B) \land \neg(\neg A \land A) \land \neg(\neg B \land A)$$

We observe that $\phi$ cannot be reduced too much:

$$\begin{aligned}\phi &= (A \lor B \lor C) \land \neg(A \land B) \land \neg(A \land C) \land \neg(B \land C)\\ &\equiv (A \lor B \lor C) \land (\neg A \lor \neg B) \land (\neg A \lor \neg C) \land (\neg B \lor \neg C)\end{aligned}$$

However, $\psi$ can be reduced completely:

$$\begin{aligned}\psi &= (\neg A \lor \neg B \lor A) \land \neg(\neg A \land \neg B) \land \neg(\neg A \land A) \land \neg(\neg B \land A)\\ &\equiv 1 \land \neg(\neg A \land \neg B) \land \neg 0 \land \neg(\neg B \land A)\\ &\equiv (A \lor B) \land (B \lor \neg A)\\ &\equiv B\end{aligned}$$

Thus, it's very easy to see that $\phi \land \psi \to B$ is indeed a tautology:

$$\begin{aligned}\phi \land \psi \to B &\equiv \neg(\phi \land \psi) \lor B\\ &\equiv \neg\phi \lor \neg\psi \lor B\\ &\equiv \neg\phi \lor \neg B \lor B\\ &\equiv 1\end{aligned}$$

Hence, this concludes that $\phi, \psi \models B$ and thus that the second button is the self-destruction one. However, we still observe that this alternative procedure may be as tedious as enumerating the whole truth table, even though it is usually faster. Currently, there are no efficient procedures that can answer questions such as "is $F$ satisfiable?", "is $F$ unsatisfiable?" and "is $F$ a tautology?". Finding such algorithm or proving that such algorithm cannot exist is equivalent to solving the Millennium Problem $\mathsf{P} \overset{?}{=} \mathsf{NP}$, which asks the question "can every efficiently verifiable problem also be efficiently solved?". For this problem, the *Clay Mathematical Institute* offers a prize of one million dollars. In many cases, it is possible to decide whether a certain proposition is a tautology or not, or whether a certain conclusion is a logical consequence of other propositions without constructing the truth table, but only by reasoning rigorously at a higher level.

## 1.3 Disjunctive and Conjunctive Normal Forms

Every propositional formula can be rewritten in a standard way. In particular, we have two standard ways to write a formula: the **Disjuntive Normal Form (DNF)** and the **Conjunctive Normal Form (CNF)**. A formula is said to be in DNF if it is a disjuction of AND clauses. An AND *clause* is a conjunction of literals. A *literal* is a propositional variable or a negation of a propositional variable. In other words, we say that $F$ is in DNF if:

$$F = \bigvee_{j=1}^{m} D_j = \bigvee_{j=1}^{m} (\ell_{1,j} \land \ell_{2,j} \land \ldots \land \ell_{k_j,j})$$

where each $D_j$ is an AND clause of the formula and each $\ell_{i_h}$ is a literal. Similarly, a formula is said to be in CNF if it is a conjunction of OR clauses.

$$F = \bigwedge_{j=1}^{m} C_j = \bigwedge_{j=1}^{m} (\ell_{1,j} \lor \ell_{2,j} \lor \ldots \lor \ell_{k_j,j})$$

Using the logical equivalence properties, it's clear that every formula has an equivalent DNF and CNF formula. In a more direct way, we can use truth tables (hence assignments) to yield an equivalent DNF or CNF formula.

**Theorem 1.2: Equivalent DNF and CNF formulas**

For every formula $F$ there is a DNF formula $F^{\mathsf{DNF}}$ and a CNF formula $F^{\mathsf{CNF}}$ such that $F \equiv F^{\mathsf{DNF}} \equiv F^{\mathsf{CNF}}$.

*Proof.* Given $\mathcal{L} = \{p_1, \ldots, p_n\}$, consider the truth table of $F$.

| $p_1$ | $p_2$ | $\ldots$ | $p_n$ | $F$ |
|---|---|---|---|---|
| $0$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $\vdots$ | $\ddots$ | $\ddots$ | $\vdots$ | $\vdots$ |
| $0$ | $\ddots$ | $\ddots$ | $\vdots$ | $\vdots$ |
| $1$ | $\ddots$ | $\ddots$ | $\vdots$ | $\vdots$ |
| $\vdots$ | $\ddots$ | $\ddots$ | $\vdots$ | $\vdots$ |
| $1$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |

This truth table has $2^n$ rows. Each $j$-th row defines an assignment $\alpha_j$ such that if $p_1$ assumes value $\alpha_j(p_1)$, $p_2$ assumes value $\alpha_j(p_2)$ and so on then $A$ assumes value $\alpha_j(A)$. Hence, the cases where $A$ is true are completely described by the rows $i$ where $\alpha_i(A) = 1$.

For each $j \in [2^n]$ and each $i \in [n]$, let $\ell_{i,j}$ be defined as:

$$\ell_{i,j} = \begin{cases} p_1 & \text{if } \alpha_j(p_i) = 1 \\ \neg p_1 & \text{if } \alpha_j(p_i) = 0 \end{cases}$$

We notice that for every assignment $\alpha$ it holds that:

$$\alpha(\ell_{1,j} \wedge \ldots \wedge \ell_{n,j}) = 1 \implies \alpha(\ell_{1,j}) = \alpha_j(p_i), \ldots, \alpha(\ell_{n,j}) = \alpha_j(p_n)$$

This implies that for every assignment $\alpha$ it holds that $\alpha(A) = 1$ if and only if for some $j \in [2^n]$ it holds that $\alpha$ corresponds to $\alpha_j$ over $p_1, \ldots, p_n$. Hence, we conclude that $F \equiv F^{\mathsf{DNF}}$, where:

$$F^{\mathsf{DNF}} = \bigvee_{j=1}^{2^n} \bigwedge_{i=1}^{n} \ell_{i,j}$$

Equivalently, we can obtain $F^{\mathsf{CNF}}$ proceeding in the same way by defining each $\ell'_{i,j}$ as:

$$\ell'_{i,j} = \begin{cases} \neg p_1 & \text{if } \alpha_j(p_i) = 1 \\ p_1 & \text{if } \alpha_j(p_i) = 0 \end{cases}$$

In this case, for every assignment $\alpha$ it holds that:

$$\alpha(\ell'_{1,j} \vee \ldots \vee \ell'_{n,j}) = 0 \implies \alpha(\ell'_{1,j}) = \alpha_j(p_i), \ldots, \alpha(\ell'_{n,j}) = \alpha_j(p_n)$$

This implies that for every assignment $\alpha$ it holds that $\alpha(A) = 0$ if and only if for some $j \in [2^n]$ it holds that $\alpha$ corresponds to $\alpha_j$ over $p_1, \ldots, p_n$. Hence, we conclude that $F \equiv F^{\mathsf{CNF}}$, where:

$$F^{\mathsf{CNF}} = \bigwedge_{j=1}^{2^n} \bigvee_{i=1}^{n} \ell'_{i,j}$$

$\square$

We observe that the proof of the above theorem is *constructive*, meaning that is based on a procedure that can be used to obtain the DNF and CNF formulas equivalent to an original one. For instance, given the truth table of $A \to B \wedge C$ shown in Section 1.2, we get the following equivalent DNF formula:

$$(\neg A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge B \wedge C) \vee (A \wedge B \wedge C)$$

while the equivalent CNF formula corresponds to:

$$(\neg A \vee B \vee C) \wedge (\neg A \vee B \vee \neg C) \wedge (\neg A \vee \not B \vee C)$$

We also observe that the second part of the last proof could've also been achieved through a different argument. Consider the fact that $F \equiv \neg\neg F$. Since we know that each formula has a DNF equivalent, we know that there is a formula $(\neg F)^{\mathsf{DNF}}$ such that $\neg F \equiv (\neg F)^{\mathsf{DNF}}$. By iteratively applying De Morgan's law, we can transform $\neg(\neg F)^{\mathsf{DNF}}$ into an equivalent CNF formula. Hence, we conclude that $F^{\mathsf{CNF}} = \neg(\neg F)^{\mathsf{DNF}}$ is the CNF formula such that $F \equiv F^{\mathsf{CNF}}$.

## 1.4 The compactness theorem

The **compactness theorem** is a fundamental result in mathematical logic first proved by Kurt Gödel. It asserts that if every finite subset of a set of propositions is satisfiable, then the entire set is satisfiable. When the set of propositions is finite, the theorem is trivial. When the set of proposition is countably infinite, i.e. is in bijection with $\mathbb{N}$, this theorem becomes of particular interest, acting as a **bridge** between finiteness and infiniteness.

**Definition 1.4: Theory**

Given a language $\mathcal{L}$, a theory $T$ over $\mathcal{L}$ is a set of propositions defined on $\mathcal{L}$, i.e. $T \subseteq \mathsf{PROP}_{\mathcal{L}}$.

A theory is said to be satisfiable when there is an assignment $\alpha$ such that for all $A \in T$ it holds that $\alpha(A) = 1$, otherwise the theory is said to be unsatisfiable – in other words, a theory can be viewed as a conjunction of propositions. With a slight abuse, we extend the notation $\alpha(T)$ to theories for their evaluation.

We observe that while a theory *can* have infinite cardinality every proposition that lies inside it must have a *finite* number of variables since by definition a proposition is a *finite* expression. This fact plays a crucial role for the compactness theorem, which can be stated as follows.

**Theorem 1.3: Compactness theorem**

Let $T$ be a theory of finite or countably infinite cardinality and let $A$ be a proposition. Then, $A$ follows from $T$ if and only if there is a finite sub-theory $T^{fin}$ such that $A$ follows from $T^{fin}$.

$$T \models A \iff \exists T^{fin} \underset{fin}{\subseteq} T \text{ such that } T^{fin} \models A$$

As we already mentioned, when the theory $T$ is finite the theorem is trivial. For the countably infinite case, instead, we observe that only one of the two directions is trivial: if $A$ follows from a finite sub-theory $T^{fin} \underset{fin}{\subseteq} T$ then $A$ clearly also follows from $T$ since it contains $T^{fin}$. The other direction appears to be intuitive: in order for $A$ to follow from $T$, the derivation process must stop “somewhere” in order to prevent an infinite evaluation. All the propositions used in this evaluation process form the finite sub-theory from which $A$ follows. However, such argument is actually a *consequence* of the theorem since we're guaranteed to stop the evaluation only if the theorem is valid. To prove the other direction, we first define the concept of **finite satisfiability**.

**Definition 1.5: Finite satisfiability**

Given a theory $T$, we say that $T$ is finitely satisfiable when all of its finite subsets are satisfiable.

We observe that the definition of finite satisfiability is different from the definition of satisfiability of a theory. In the former, we require that for every finite sub-theory there is a satisfying assignment, while in the latter we require that there is an assignment that satisfies every finite sub-theory. This inversion of quantifiers is the trick behind the compactness theorem. Before proceeding, we observe that the compactness theorem can actually be restated in an equivalent way.

**Proposition 1.1**

Let $T$ be a theory of finite or countably infinite cardinality. Then, the two following points are equivalent:

- $T \models A$ if and only if there is a finite sub-theory $T^{fin}$ such that $T^{fin} \models A$.
- $T$ is satisfiable if and only if it is finitely satisfiable

*Proof.* Assume that $T \models A$ if and only $\exists T^{fin} \underset{fin}{\subseteq} T$ such that $T^{fin} \models A$. Suppose that $T$ is unsatisfiable. Then, this can happen if and only if $T \models 0$. Using the assumption, we get that:

$$T \models 0 \iff \exists T_0 \underset{fin}{\subseteq} T \; T_0 \models 0$$

Again, $T_0 \models 0$ can happen if and only if $T_0$ is unsatisfiable, concluding that $T$ is unsatisfiable if and only if $\exists T_0 \underset{fin}{\subseteq} T$ it holds that $T_0$ is also unsatisfiable. By contrapositive, we conclude that $T$ is satisfiable if and only if it is finitely satisfiable.

Vice versa, assume now that $T$ is satisfiable if and only if it is finitely satisfiable. If there is a sub-theory $T_0 \underset{fin}{\subseteq}$ such that $T_0 \models A$ then $T \models A$ is trivially true. By way of contradiction, suppose that $T \models A$ but $\forall T_0 \underset{fin}{\subseteq} T$ it holds that $T_0 \not\models A$. Then, we know that $\forall T_0 \underset{fin}{\subseteq} T$ there is an assignment $\alpha$ such that $\alpha(T_0) = 1$ and $\alpha(A) = 0$, which implies that $\alpha(\neg A) = 1$. Using the assumption, we get that:

$$\forall T_0 \underset{fin}{\subseteq} T \; \exists \alpha \;\; \alpha(T_0) = 1, \alpha(\neg A) = 1 \iff \exists \beta \;\; \beta(T \cup \{\neg A\}) = 1$$

Since $\beta(T) = 1$ and $\beta(\neg A) = 1$, we conclude that $T \not\models A$. □

Using the above proposition, we can prove the first version of the compactness theorem by proving its second version. Again, one of the two directions of the equivalent statement is trivial: if $T$ is satisfiable then every sub-theory of $T$ must be satisfiable. As in the previous case, the other direction also seems to be intuitive: if $T$ is finitely satisfiable then there should be a way of combining the assignments $\alpha_1, \alpha_2, \ldots$, that satisfy the finite sub-theories $T_1, T_2, \ldots$. However, such combination cannot be easily achieved: two assignments $\alpha_i, \alpha_j$ that satisfy the finite sub-theories $T_i, T_j$ may conflict on a variable, meaning that $\alpha_i(x) \neq \alpha_j(x)$, making their combination not so easy. To prove this direction, we use the following lemma.

**Lemma 1.1: Extendibility of a finitely satisfiable theory**

If $T$ is a finitely satisfiable theory then at least one between $T \cup \{A\}$ and $T \cup \{\neg A\}$ is finitely satisfiable.

*Proof.* By way of contradiction, suppose that $T$ is a finitely satisfiable but both $T \cup \{A\}$ and $T \cup \{\neg A\}$ are not finitely satisfiable. Hence, there are two finite sub-theories $T_0 \underset{fin}{\subseteq} T \cup \{\neg A\}$ and $T_1 \underset{fin}{\subseteq} T \cup \{A\}$ that are unsatisfiable. Let $\widehat{T_0} = T_0 - \{\neg A\}$ and $\widehat{T_1} = T_1 - \{A\}$. Since $\widehat{T_0} \cup \widehat{T_1} \subseteq T$ and $T$ is finitely satisfiable, there must be an assignment $\alpha$ such that $\alpha(\widehat{T_0} \cup \widehat{T_1}) = 1$. Now, we have two cases: if $\alpha(A) = 1$ then $\alpha(T_1) = 1$, while if $\alpha(A) = 0$ then $\alpha(T_0) = 1$. Both cases are a contradiction, concluding the proof. □

We observe that the lemma is not mutually exclusive. For instance, given the finitely satisfiable theory $T = \{\neg A \lor B, \neg A \lor C\}$, both $T \cup \{\neg A\}$ and $T \cup \{\neg A\}$ are finitely satisfiable. We can now use the lemma to prove the second version of the compactness theorem.

**Theorem 1.4: Compactness theorem (2nd version)**

Let $T$ be a theory of finite or countably infinite cardinality and let $A$ be a proposition. Then, $T$ is satisfiable if and only if it is finitely satisfiable.

*Proof.* The finite case is trivial, hence we focus on the countably infinite case. If $T$ is satisfiable then every sub-theory of $T$ must be satisfiable, including finite ones. Vice versa, suppose that $T$ is finitely satisfiable. Assume that there is an enumeration, i.e. a total order, of $\mathcal{L} = \{p_1, p_2, \ldots\}$. Let $T_0 = T$ and for each $i \in \mathbb{N}$ let:

$$T_{i+1} = \begin{cases} T_i \cup \{p_i\} & \text{if } T_i \cup p_i \text{ finitely satisfiable} \\ T_i \cup \{\neg p_i\} & \text{otherwise} \end{cases}$$

By the previous lemma, we know that each $T_{i+1}$ is finitely satisfiable since at least one between $T_i \cup \{p_i\}$ and $T_i \cup \{p_i\}$ must be finitely satisfiable (by construction, when both are finitely satisfiable we give precedence to $T_i \cup \{p_i\}$). Hence, the sequence $T_0 \subseteq T_1 \subseteq \ldots$ is well-defined and ensures that they don't conflict with each other.

**Claim 1:** $T^* = \bigcup\limits_{i \in \mathbb{N}} T_i$ is finitely satisfiable.

*Proof of Claim 1.* Let $X$ be a finite sub-theory of $T^*$. By construction, there must be $T_i$ such that $X \subseteq T_i$. Since $T_i$ is finitely satisfiable, $X$ must also be satisfiable. □

Let $\alpha^*$ be an assignment defined as:

$$\alpha^*(p_i) = \begin{cases} 1 & \text{if } p_i \in T^* \\ 0 & \text{if } \neg p_i \in T^* \end{cases}$$

**Claim 2**: $\alpha^*(T) = 1$

*Proof of Claim 2.* Fix $A \in T$. Let $p_{i_1}, \dots, p_{i_k}$ be the variables that appear in $A$ – recall that a proposition always has a finite number of variables. For each $j \in [k]$, let:

$$p^*_{i_j} = \begin{cases} p_{i_j} & \text{if } p_{i_j} \in T^* \\ \neg p_{i_j} & \text{if } \neg p_{i_j} \in T^* \end{cases}$$

Let $A^* = \{A, p^*_{i_1}, \dots, p^*_{i_k}\}$. Since $A^* \subseteq T^*$ and $T^*$ is finitely satisfiable, there must be an assignment $\beta_A$ such that $\beta_A(A^*) = 1$, which can happen if and only if $\beta_A(A) = 1$ and $\beta_A(p^*_{i_j}) = 1$ for each $j \in [k]$. We notice that by construction it holds that $\beta_A(p^*_{i_j}) = \alpha^*(p_{i_j})$ for any $j \in [k]$. Moreover, we notice that:

- If $p_{i_j} \in T^*$ then $\alpha^*(p_{i_j}) = 1$ and $p^*_{i_j} = p_{i_j}$, implying that $\beta_A(p_{i_j}) = \beta_A(p^*_{i_j}) = 1$ and thus that $\alpha^*(p_{i_j}) = \beta_A(p_{i_j})$
- If $\neg p_{i_j} \in T^*$ then $\alpha^*(p_{i_j}) = 0$ and $p^*_{i_j} = \neg p_{i_j}$, implying that $\beta_A(p_{i_j}) = \neg\beta_A(\neg p_{i_j}) = \neg\beta_A(p^*_{i_j}) = 0$ and thus that $\alpha^*(p_{i_j}) = \beta_A(p_{i_j})$

Since in both cases we have that $\beta(p_{i_1}) = \alpha^*(p_{i_1}), \dots, \beta(p_{i_k}) = \alpha^*(p_{i_k})$ and $p_{i_1}, \dots, p_{i_k}$ are the variables inside $A$, it must also hold that $\alpha^*(A) = \beta_A(A) = 1$. By applying the same argument on each $A \in T$, we conclude that $\alpha^*(T) = 1$. □

Since $\alpha^*(T) = 1$, we conclude that $T$ is satisfiable. □

### 1.4.1 König's lemma

The Compactness theorem has major implications in mathematics. In particular, it can be applied to prove results in areas outside of logic itself, such as:

1. Every finitely-branching infinite tree has an infinite path
2. A (infinite) graph is $k$-colorable if and only if every finite subgraph is $k$-colorable
3. Every partial order can be extended to a total order

The idea behind the proofs of the three results is the same. The infine structure and the property that has to be proven are converted into an infinite theory, which we know to be satisfiable if and only if it is finitely satisfiable, meaning that the statement must be true also for every finite restriction of the infinite case.

In particular, we'll focus on the first result, which is known as **König's lemma** [Kö27]. A tree is said to be finitely-branching if each node has a finite number of children. On first sight, the statement of the lemma may seem trivial: if the tree is infinite and every branch has a finite number of children then the only way for it to have infinite number of nodes is to have an infinite path. However, we observe that for infinite quantities nothing is considerable trivial, as many intuitive statements can be proven to be false. Moreover, even if the lemma were to be trivial, proving it is no easy task.

**Lemma 1.2: König's lemma**

Every finitely-branching infinite tree has an infinite path

To prove the lemma, we'll show that it is actually equivalent to the Compactness theorem. This equivalence shows that Compactness is a core concept for mathematics. In fact, König's lemma (and thus also Compactness) are equivalent to the *axiom of dependent choice*, a weak form of the *axiom of choice* that is sufficient to develop most of mathematics. In our context, trees are described by a pair $(T, \prec)$ where $T$ is a set of nodes and $\prec$ is a partial order describing the tree, i.e. $t \prec s$ if and only if $t$ is the parent of $s$.

**Proposition 1.2**

The Compactness theorem and König's lemma are equivalent

*Proof.* To prove that Compactness implies König's lemma, we'll define an infinite theory that models the tree and the existence of an infinite path inside it. Let $(T, \prec)$ be a finitely-branching infinite tree. For each level $\ell \in \mathbb{N}$ of the tree, let $\mathcal{L}$ be the language defined as:

$$\mathcal{L} = \bigcup_{\ell \in \mathbb{N}} \{p_{1,\ell}, \ldots, p_{2^\ell,\ell}\}$$

We'll use each variable $p_{i,\ell}$ to represent the possibility of the $i$-th node of level $\ell$ to be inside of $T$. For each level $\ell \in \mathbb{N}$, let $S_{T_\ell}$ be the finite theory containing the following propositions:

- We add $(p_{1,\ell}, \ldots, p_{2^\ell,\ell}) \in S_T$
- For all $i, j \in [2^\ell]$ we add $\neg(p_{i,\ell} \land p_{j,\ell}) \in S_T$
- If the nodes $t_{i,\ell-1}$ and $t_{j,\ell}$ exist in $T$ and $t_{i,\ell-1} \prec t_{j,\ell}$ then we add $(p_{i,\ell-1} \to p_{j,\ell}) \in S_T$

Then, let $S_T$ be the unions of all such finite theories, i.e. $S_T = \bigcup_{\ell \in \mathbb{N}} S_{T_\ell}$

**Claim 1:**: if $S_T$ is satisfiable then $(T, \prec)$ has an infinite branch

*Proof of Claim 1.* Given an assignment $\alpha$ that satisfies $S_T$, let $B = \{(t, \ell) \mid \alpha(p_{t,\ell}) = 1\}$. By construction of $S_T$, if $\alpha(S_T) = 1$ then the second type of constraints impose that $B$ describes a path, the third type imposes that such path respects $\prec$ and the first type imposes that $B$ contains at least a variable for each level. Hence, $B$ describes a branch $T$ with infinite levels. □

We'll now use Compactness to show that $S_T$ is indeed satisfiable. Fix $k \in \mathbb{N}$ and consider the sub-theory $S_\ell = \bigcup_{\ell=1}^{k} S_{t_\ell}$.

**Claim 2:** for any $k \in \mathbb{N}$, $S_\ell$ is satisfiable

*Proof of Claim 2.* It's easy to see that $S$ describes the existence of a finite branch on $(T, \prec)$ that has $k$ levels. Hence, since any tree with at least $k$ levels has a finite branch of $k$ levels, the sub-theory $S_\ell$ is always satisfiable. $\square$

For any finite sub-theory $S^{fin} \underset{fin}{\subseteq} S_T$, let $\ell^* \in \mathbb{N}$ be the smallest level such that $S^{fin} \subset S_{\ell^*}$. By Claim 2, $S^{fin}$ must be satisfiable since otherwise $S_{\ell^*}$ would be unsatisfiable. Since any finite subset is satisfiable, by Compactness we know that $S_T$ is also satisfiable. By Claim 1, this concludes that $(T, \prec)$ contains an infinite branch.

Similarly, to prove Compactness using König's lemma we'll construct a finitely-branching infinite tree that describes possible assignments for the theory. Without loss of generality, let $S$ be a countably infinite theory whose language $\mathcal{L} = \{p_1, p_2, \dots\}$ has infinite variables. Suppose that $S$ is finitely satisfiable. For each $\ell \in \mathbb{N}$, let $S_\ell$ be the sub-theory containing all the propositions defined on the first $\ell$ variables.

$$S_\ell = \{A \in S \mid \mathrm{Var}(A) \subseteq \{p_1, \dots, p_\ell\}\}$$

Let $T_S$ be the tree defined as follows:

- For each $\ell \in \mathbb{N}$, the $i$-th node of level $\ell$ is mapped to the $i$-th assignment $\alpha_{i,\ell} : \{p_1, \dots, p_\ell\} \to \{0, 1\}$ that satisfies $S_\ell$, where $i \leq 2^\ell$.
- For each node $t, s$ of $T_S$, we let $t \prec s$ if and only if the assignment of $t$ is a restriction of the assignment of $s$

**Claim 3:** $(T_S, \prec)$ has an infinite number of levels

*Proof.* Proof of Claim 3. Fix $\ell \in \mathbb{N}$. We observe that each $S_\ell$ may be finite or infinite. If $S_\ell$ is finite then we know that it is also satisfiable since $S$ is finitely satisfiable hypothesis. Hence, in this care there must be at least one satisfying assignment in level $\ell$. Suppose now that $S_\ell$ is infinite. Given an enumeration $S_\ell = \{A_1, A_2, \dots\}$, for each $k \in \mathbb{N}$ we know that $S_\ell^k = \{A_1, \dots, A_k\}$ is a finite subset of $S$, hence satisfiable by hypothesis. Thus, for each $k \in \mathbb{N}$ there is an assignment $\alpha_k : \{p_1, \dots, p_\ell\}$ that satisfies $S_\ell^k$.

However, since there are at most $2^\ell$ satisfying assignments for $S_\ell$, by the *Infinite Pigeonhole Principle* we know that there must be an assignment $\alpha^* : \{p_1, \dots, p_\ell\}$ such that for each $k \in \mathbb{N}$ it holds that $\alpha^*(S_\ell^k) = \alpha_k(S_\ell^k) = 1$. meaning that it must also satisfy $S_\ell$ and thus that there is at least one satisfying assignment in level $\ell$. $\square$

Since $(T_S, \prec)$ has an infinite number of levels, know that it must be infinite. Moreover, since each node is finitely-branching, by König's lemma we know that there must be an infinite branch $B$ inside of it. Finally, the assignment $\beta^* = \bigcup_{\beta \in B}$ satisfies $S$. $\square$

### 1.4.2 Compactness and decidability

Given the notion of problem formalization through propositional logic, the relations between properties and the notion of satisfiability and logic consequence, it's natural to ask if it is possible to **automate** questions such as "$T \models A$?", i.e. creating an algorithmic procedure that is able to answer the question. In particular, we're not interested in the amount of resources needed by a machine to achieve such task.

If $T$ is a finite theory then the answer is trivially "yes": we know that answering $T \models A$ is equivalent to answering $(T \to A) \in \text{TAUT}$, hence we can build a machine that tries every possible assignment, answering positively if every one of them satisfies the formula and negatively otherwise. In computability theory, we say that such problem is *decidable*.

**Definition 1.6: Decidability**

A subset $S \subseteq \Sigma^*$ is said to be **decidable** (or *computable*) when there is an algorithm $\mathcal{A} : \Sigma^* \to \{0, 1\}$ such that $\forall x \in \Sigma^*$ it holds that $\mathcal{A}(x) \downarrow= 1$ for all $x \in S$ and $\mathcal{A}(x') \downarrow= 0$ for all $x' \notin S$.

Here, $\Sigma^*$ denotes the set of all strings defined on an set of symbols $\Sigma$, while $\mathcal{A}(x) \downarrow= 1$ denotes that the procedure $\mathcal{A}$ terminates on input $x$ and return 1 (likewise for $\mathcal{A}(x) \downarrow= 0$).

What about infinitely countable theories? We know that such theories may have an infinite number of variables, hence an infinite number of assignments, breaking the trivial procedure. Through Compactness, we know that $T \models A$ if and only if there is a finite subset $T^{fin} \underset{fin}{\subseteq} T$ such that $T^{fin} \models A$. Let $\text{Fin}(T)$ be the set of all finite subsets of $T$. We observe that since $T$ is countably infinite, $\text{Fin}(T)$ also is. Thus, we know that $\text{Fin}(T)$ has an enumeration. If such enumeration can be yield by an algorithm then we say that $\text{Fin}(T)$ is *computably enumerable*.

**Definition 1.7: Computably enumerable**

A subset $S \subseteq \Sigma^*$ is said to be **computably enumerable** (or *recursively enumerable*) when there is an algorithmic procedure $\mathcal{A} : \mathbb{N} \to \Sigma^*$ that produces a list of all and only the elements inside it, meaning that $S = \{\mathcal{A}(0), \mathcal{A}(1), \ldots\}$

Suppose $\text{Fin}(T)$ is computably enumerable and let $\mathcal{A}$ be the enumerating procedure. We can "modify" such enumerating procedure to make it also test if $A$ is a logical consequence of the $i$-th finite subset. If $T \models A$ then Compactness guarantees the existence of an index $i \in \mathbb{N}$ such that $\mathcal{A}(i) \models A$, meaning that we can decide if the answer is positive. If $T \not\models A$, instead, no such index can existing, making the procedure never halt, denoted as meaning that we cannot decide if the answer is negative. When this happens, we say that the problem is *semi-decidable* (or *recognizable*).

**Definition 1.8: Semi-decidability**

A subset $S \subseteq \Sigma^*$ is said to be **semi-decidable** (or *recognizable*) when there is an algorithm $\mathcal{A} : \Sigma^* \to \{0, 1\}$ such that $\forall x \in \Sigma^*$ it holds that $\mathcal{A}(x) \downarrow= 1$ for every $x \in S$ and for every $x' \notin S$ $\mathcal{A}(x') \downarrow= 0$ or $\mathcal{A}(x') \uparrow$, i.e. it never halts.

**Lemma 1.3**

Let $T$ be a countably infinite theory. If $\mathrm{Fin}(T)$ is computably enumerable then the set $\{A \mid T \models A\}$ is semi-decidable.

By definition, every decidable problem is also semi-decidable, but the opposite doesn't always hold. The typical example is the *Halting problem*, which was proven by Turing [Tur37] to be semi-decidable but undecidable.

**Proposition 1.3**

A subset $S \subseteq \Sigma^*$ is semi-decidable if and only if $S$ is computably enumerable.

*Proof.* Suppose that $S$ is semi-decidable by a procedure $\mathcal{B}$. Then, we can build an algorithm $\mathcal{A}$ that enumerates every possible element of $\Sigma^*$ in parallel and runs $\mathbf{B}$ on every element, returning only those elements $x$ such that $\mathcal{B}(0) \downarrow= 1$.

Vice versa, suppose that $S$ is computably enumerable. Then, we can build an algorithm $\mathcal{A}$ that enumerates each element of $S$ in parallel and checks if the input $x$ is equal to one of them. If $x \in S$, the procedure will eventually halt and accept. If $x \notin S$, the procedure will go on forever. □

We observe that in order to guarantee that $\mathrm{Fin}(T)$ is computably enumerable it suffices to prove that $T$ is computably enumerable. In fact, if the latter condition holds, we can use $T$'s enumerating procedure to yield an enumerating procedure for $\mathrm{Fin}(T)$: if we can mechanically produce the enumeration $F_1, F_2, F_3, \ldots$ of the predicates of $T$ then we can mechanically produce the enumeration $\{F_1\}, \{F_1, F_2\}, \{F_1, F_2, F_3\}$.

**Theorem 1.5**

Let $T$ be a countably infinite theory. If $T$ is computably enumerable then the set $\{A \mid T \models A\}$ is semi-decidable.

Counterintuitively, there are indeed some countably infinite theories, hence enumerable ones, that cannot be *computably* enumerable. This comes from a simple counting argument. Since each procedure is nothing more than a finite sequence of instructions in a formal language, the set of all procedures is countably infinite. Each computaby enumerable theory can be mapped to an algorithmic procedure that enumerates it. Thus, the set of all computably enumerable theories is countably infinite. However, the set of all enumerable theories is not countably infinite, since it corresponds to the set of all possible numerable subsets of the set of propositions – this is the same argument as to why $\mathbb{N}$ is countably infinite but $\mathcal{P}(\mathbb{N})$ isn't. Hence, there must be at least one theory that isn't computably enumerable.

But what about decidability? Is there a way to guarantee that the set $\{A \mid T \models A\}$ is also decidable and not only semi-decidable? The answer is yes: if the theory is *semantically complete* then the set becomes semi-decidable.

**Definition 1.9: Semantically complete**

A theory $T$ is said to be **semantically complete** if for all formulas $A$ either $T \models A$ or $T \models \neg A$.

We observe that, by definition, if a theory is semantically complete then $T \not\models A$ if and only if $T \models \neg A$. This property can be used to construct a procedure that always terminates: the idea is to use two parallel copies of the semi-decidable procedure that we previously defined. The first copy checks if $T \models A$, while the second checks if $T \models \neg A$. Since the theory is complete, we know that exactly one of the two copies has to eventually halt, while the other will *diverge*, i.e. never halt. If the first copy halts then we're sure that $T \models A$, while if the other copy halts then we're sure that $T \not\models A$.

**Theorem 1.6**

Let $T$ be a countably infinite theory. If $T$ is computably enumerable and semantically complete then the set $\{A \mid T \models A\}$ is decidable.

We observe that previous parallel procedure implicitly uses the fact that, if $T$ is computably enumerable and semantically complete, the complementary set $\{A \mid T \not\models A\}$ is also semi-decidable. In fact, the same approach can be generalized for any semi-decidable problem whose complement is also semi-decidable.

**Proposition 1.4**

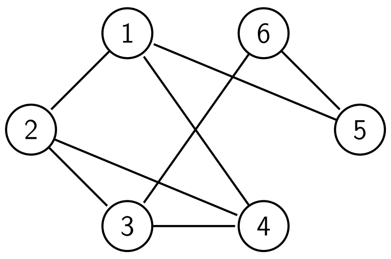
A set $S \subseteq \Sigma^*$ is decidable if and only if both $S$ and $\overline{S}$ are semi-decidable.

2

# Predicate logic

## 2.1 Syntax and semantics

We discussed how problems can be easily modeled through propositional logic. However, sometimes this *zero-th order* logical language isn't powerful enough. For instance, consider the following graph $G = (V, E)$:



Suppose that we want to describe the property "every vertex of the graph $G$ has at most 3 neighbors". Predicate logic allows us to express such property through the use of **quantifiers** in the following way:

$$\forall x_t, x_i, x_j, x_k(\neg E(x_t, x_i) \lor \neg E(x_t, x_j) \lor \neg E(x_t, x_k))$$

When the domain is finite, such as this case, the quantifier $\forall$ can be "simulated" through a conjunction:

$$\bigwedge_{1\leq t\leq 6} \bigwedge_{1\leq i,j,k\leq 6} (\neg p_{t,i} \lor \neg p_{t,j} \lor p_{t,k})$$

obtaining a proposition where we intuitively have that $p_{a,b} = 1$ if and only if $(a, b) \in E$. When the domain is infinite, instead, such simulation cannot be achieved since, by definition, a proposition cannot be infinite. Nonetheless, the above formula can indeed be simulated in propositional logic, but it would require the construction of an infinite theory, which we amply discussed to be not so easy to work with.

In *predicate logic*, concepts described in terms of relational structures.

**Definition 2.1: Relational structure**

A **relational structure** (or *predicate structure*) is a tuple $\mathcal{A} = (A, R_1, \dots, R_n, c_1, \dots, c_k)$ where:

- $A$ is a non-empty set called *domain*
- $R = \{R_1, \dots, R_n\}$ are *relations* on $A$ of any arity, even different ones.
- $c_1, \dots, c_k$ are *constants*, i.e. special elements of $A$ that have been "explicitely named"

For instance, the previous graph can be described as $\mathcal{G} = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6\}$ is the domain and $E$ is the edge relation.

$$E = \{(1, 2), (1, 4), (1, 5), (2, 3), (3, 4), (3, 6), (5, 6)\}$$

We observe that even concepts such as mathematical operations can be described as a relational structure. For instance $\mathcal{N} = (\mathbb{N}, \leq, +, \cdot, 0, 1)$ is a relational structure on the natural numbers where 0 and 1 have been explicitly stated. In other words, we have that $\mathbb{N} = \{n_0, n_1, n_2, \dots\}$ and we have denoted $n_0$ as 0 and $n_1$ as 1. The operation $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ can be described through its *operation graph*, i.e. the ternary relation defined as:

$$+ = \{(n_0, n_1, n_1), (n_0, n_2, n_2), \dots, (n_1, n_2, n_3), (n_1, n_3, n_4), \dots\}$$

where $+(a, b, c)$ if and only if $a + b = c$. Generally, we want to use predicate logic express natural global properties of structures and local properties of elements of structures. For example, for a graph $\mathcal{G} = (V, E)$ we might want to express global properties such as: $\mathcal{G}$ is undirected, $\mathcal{G}$ is complete, .... The concept of predicate logic is designed in order to reflect the complexity of what we called *structures*. Basically we have one linguistic (syntactical) component corresponding to each (semantic) component of a structure. Besides, we have properly logical syntactical elements such as variables $x_1, \dots, x_n$, logical connectives $\neg, \wedge, \vee, \dots$, quantifiers $\forall, \exists$ and the identity symbol $=$.

However, in predicate logic we can quantify only on elements of the structure and not on structures themselves. This is the reason predicate logic is also called **First-order logic**. Quantification over structures, instead, is allowed in **Second-order logic**, while no quantification is allowed in **Zeroth-order logic**, i.e. propositional logic.

**Definition 2.2: Relational language**

A **relational language** (or *predicate language*, *first-order language*) is a tuple $\mathcal{L}$ of symbols of the following types:

1. A countably infinite set of *variables*
2. A finite or countably infinite set of *relation symbols*, each with its arity
3. A finite or countably infinite set of *constant symbols*

To properly distinguish between structures and language, we observe that structures are **instances of a language** – in programming terms, this can be viewed as the difference between an object and a class. For instance, a possible language for Graph Theory is $\mathcal{L}_G = (E)$, where the binary relation symbol $E$ denotes that any structure $\mathcal{G}$ defined on $\mathcal{L}_\mathcal{G}$ has to have a binary relation $E^\mathcal{G}$. This relation varies from instance to instance.

From now on, we'll always assume that any relational language has the equality symbol $=$. This symbol can be described in two ways:

1. $=$ is a binary relation symbol and any structure over the language instances it as $\{(a_1, a_1), (a_2, a_2), \dots, \}$
2. $=$ is a special symbol and any formula over the language always implicitly contains clauses that describe equality

The two definitions are equivalently solid, so we'll use both of them – mostly the first one. In any language, objects of the domain are referred to as **terms**. By definition, such objects can be variables or constants.

Terms with no variables, i.e. constants, are sometimes referred to as *closed terms*. Similarly to predicate logic, formulas are also inductively defined in terms of connectives (and quantifiers in this case).

**Definition 2.3: Formula**

Given a relational language $\mathcal{L}$, a **formula** is a finite string $F$ such that one of the following holds:

- $F$ is the string $R(t_1, \dots, t_n)$ where $t_1, \dots, t_n$ are terms of $\mathcal{L}$
- $F$ is the string $t_i = t_j$ where $t_i, t_j$ are terms of $\mathcal{L}$
- $F$ is the string $\neg H$, where $H$ is a formula
- $F$ is the string $G * H$, where $G, H$ are formulas and $*$ is a boolean connective
- $F$ is the string $Qx\ H$ where $H$ is a formula, $x$ is a variable of $\mathcal{L}$ and $Q$ is a quantifier $(\forall, \exists)$.

The first two types of formulas are called **atomic formulas**, while the others are called *non-atomic*. In the last type of formula, $x$ is a "variable on the variables of $\mathcal{L}$", referred to as **metavariable**, while $H$ is referred to as the **scope** of the quantifier. In other words, if $v_1, v_2, \dots$ are the variables of $\mathcal{L}$, the formula $\forall x\ H$ expresses that what $H$ says about $x$ has to hold for any variable among $v_1, \dots, v_2$, while $\exists x\ H$ expresses that it has to hold for at least one variable.

When the scope $H$ contains a variable $v$ that hasn't been quantified, the variable is said to be **free**, otherwise we say that it is **bound**. We observe that he same variable can have in the same formula free and bound occurrences. For instance, in the formula $(\forall x\ R(x,x)) \to R(x,x)$ the variable $x$ is bound in the first subformula and free in the second one. A formula without free variables is referred to as **sentence**.

If $F$ is a formula and $x_1, \dots, x_n$ are distinct variables, we write $F(x_1, \dots, x_n)$ to indicate that the free variables of $F$ are contained in the set $x_1, \dots, x_n$. This notation is also useful to deal with **substitution** of terms for variables: f $F(x_1, \dots, x_n)$ is a formula and $t_1, \dots, t_n$ are terms, we denote with $F(t_1, \dots, t_n)$ (or $F(x_1/t_1, \dots, x_n/t_n)$) the formula obtained by simultaneously substituting each free occurrence of variable $x_i$ in $F$ with the term $t_i$. For instance, if $F(x_1, x_2, x_3) = R(x_1, x_2) \wedge R(x_3, x_1)$ and $a, b, c$ are terms of the language then $F(a, b, c) = R(a, b) \wedge R(a, c)$.

After defining the syntax of first-order logic, we're ready to define its semantics, i.e. the evaluation of formulas.

**Definition 2.4: $\mathcal{L}$-structure**

Let $\mathcal{L}$ be a language. An $\mathcal{L} -$ **structure** is a structure $\mathcal{A}$ such that:

- $A$ is the domain of $\mathcal{A}$
- For each relation symbol $R_i$ in $\mathcal{L}$ there is a relation $R_i^{\mathcal{A}}$ over $A$ in $\mathcal{A}$
- For each constant symbol $c_i$ in $\mathcal{L}$ there is a constant $c_i^{\mathcal{A}}$ taken from $A$ in $\mathcal{A}$

For the Graph Theory language $\mathcal{L}_{\mathcal{G}} = \{E\}$, any structure $\mathcal{G} = (V, E^{\mathcal{G}})$ is an $\mathcal{L}_{\mathcal{G}}$-structure. If $E^{\mathcal{G}}$ is symmetric, the structure $\mathcal{G}$ represents an undirected graph, otherwise it represents a directed graph.

In this type of logic, the concept of assignment has to be slightly redefined, along with the concept of satisfiability. Let $\mathcal{S}$ be an $\mathcal{L}$-structure. An **assignment** for $\mathcal{S}$ is a function $\alpha : \mathsf{Vars}_{\mathcal{L}} \to \mathrm{Dom}(\mathcal{S})$.

We say that a formula $F$ is **satisfied** by $\alpha$ on $\mathcal{S}$, written as $\mathcal{S} \overset{\alpha}{\models} F$ or $\mathcal{S} \models F[\alpha]$, if one of the following inductive cases holds:

- $F$ is atomic
- $F$ is equal to $R(t_1, \dots, t_n)$ and $(\alpha(t_1), \dots, \alpha(t_n)) \in R^{\mathcal{S}}$, where $t_1, \dots, t_n$ are terms
- $F$ is equal to $t_i = t_j$ and $\alpha(t_i) = \alpha(t_j)$ in $\mathcal{L}$, where $t_i, t_j$ are terms
- $F$ is equal to $\neg H$ and $\mathcal{S} \overset{\alpha}{\not\models} H$

- $F$ is equal to $G * H$ and $(\mathcal{S} \overset{\alpha}{\models} G) * (\mathcal{S} \overset{\alpha}{\models} H)$, where $*$ is a boolean connective
- $F$ is equal to $\forall x\ H$ and for all $s \in \mathrm{Dom}(\mathcal{S})$ it holds that $\mathcal{S} \models G\left[\alpha\begin{pmatrix}x\\s\end{pmatrix}\right]$, where $\alpha\begin{pmatrix}x\\s\end{pmatrix}$ is the assignment obtained by mapping $x \mapsto s$ in $\alpha$
- $F$ is equal to $\exists x\ H$ and $\mathcal{S} \overset{\alpha}{\not\models} \forall x\ \neg G$

We observe that if $F$ is a sentence then $\mathcal{S} \overset{\alpha}{\models} F$ doesn't depend on $\alpha$: since there are no free variables, either every assignment satisfies the formula or none does. When a formula is satisfied by every assignment in a structure $\mathcal{S}$, we say that it is **true** in $\mathcal{S}$, written as $\mathcal{S} \models F$. When a formula is true in every structure, we say that it is a **tautology** for the language, written as $\models F$. To recap, we have defined four concepts of truthfulness:

1. $\mathcal{S} \overset{\alpha}{\models} F$ means that $F$ is true in $\mathcal{S}$ over $\alpha$
2. $\mathcal{S} \models F$ means that $F$ is true in $\mathcal{S}$ for all $\alpha$, i.e. $\forall\alpha$ it holds that $\mathcal{S} \overset{\alpha}{\models} F$
3. $\models F$ means that $F$ is true in $\mathcal{L}$, i.e. $\forall\mathcal{S}$ it holds that $\mathcal{S}\alpha{\models}F$

This definitions of truthfulness easily extend to theories, i.e. set of formulas:

- $\mathcal{S} \overset{\alpha}{\models} T$ when $\mathcal{S} \overset{\alpha}{\models} F$ for all $F \in T$
- $\mathcal{S} \models T$ when $\mathcal{S} \models F$ for all $F \in T$
- $\models T$ when $\models F$ for all $F \in T$

We observe that if $F$ is finite then $\mathcal{S} \overset{\alpha}{\models} F$ can always be decided through the inductive definition. If $\mathrm{Dom}(\mathcal{S})$ is also finite then $\mathcal{S} \models F$ can also be decided since there are a finite amount of assignments. However, even if $\mathcal{L}$ has a finite number of relation and constant symbols, $\models F$ is only semi-decidable: Even if every structure is finite, there are infinitely possible structures.

# Bibliography

[Kö27] Dénes König. “Über eine Schlussweise aus dem Endlichen ins Unendliche”. In: *Acta litterarum ac scientiarum Regiae Universitatis Hungaricae Francisco Josephinae: Sectio scientiarum mathematicarum* (1927). URL: https://acta.bibl.u-szeged.hu/13338/.

[Tur37] A. M. Turing. “On Computable Numbers, with an Application to the Entscheidungsproblem”. In: *Proceedings of the London Mathematical Society* (1937). DOI: 10.1112/plms/s2-42.1.230.