SAPIENZA
UNIVERSITÀ DI ROMA

"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

June 11, 2025

Contents

Information and Contacts **1**

1 Propositional logic **2**
1.1 Introduction to logical constructs 2
1.2 Mathematical formalization 4
1.3 Disjunctive and Conjunctive Normal Forms 10
1.4 The compactness theorem 12
1.5 Decidability in logic 18

2 Predicate logic **21**
2.1 Syntax and semantics 21
2.2 Decision problems for predicate logic 25
2.3 Expressibility and structural isomorphisms 27
2.4 The back-and-forth method 31
2.5 Bounded equivalence 35
2.6 Ehrenfeucht-Fraïssé games 39
2.6.1 Non-expressibility results 41
2.6.2 Infinite rounds games 44
2.7 Inexpressibility through logical reductions 45
2.8 Quantifier elimination 46

3 True Arithmetic **54**
3.1 Non-standard models of arithmetic 54
3.2 Computable functions and arithmetic 56
3.2.1 Undecidability of arithmetic 58
3.2.2 Undefinability of arithmetic 59
3.3 Algorithmically inseparable sets 61
3.3.1 Incompleteness of arithmetic 64

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	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	$A \vee B$
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 $A \to (B \wedge C)$. A truth table for this expression requires evaluating the truth values for A, B and C, then determining the truth value of the conjunction $B \wedge C$, and finally computing the implication $A \to (B \wedge C)$.

A	B	C	$B \wedge C$	$A \to (B \wedge C)$
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 $A \to (B \wedge C)$

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 $\mathcal{L} = \{p_1, \ldots, p_n\}$, 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 finite expressions over $\mathcal{L} \cup \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ 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 $(A \wedge B), (A \vee B), (A \rightarrow B), (A \leftrightarrow B) \in \mathsf{PROP}_\mathcal{L}$

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

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 $\alpha : \mathcal{L} \rightarrow \{0, 1\}$, 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 PROP. These assignments are functions $\widehat{\alpha} : \mathsf{PROP} \rightarrow \{0, 1\}$ inductively defined as:

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

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

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

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

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

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

To make notation easier, we'll directly refer to $\widehat{\alpha}$ as α. 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 α 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*: $A \vee A \equiv A$ and $A \wedge A \equiv A$
- *Associativity*: $(A \vee B) \vee C \equiv A \vee (B \vee C)$ and $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$
- *Commutativity*: $A \vee B \equiv B \vee A$ and $A \wedge B \equiv B \wedge A$
- *Distributivity*: $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$ and $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
- *De Morgan's law*: $\neg(A \vee B) \equiv \neg A \wedge \neg B$ and $\neg(A \wedge B) \equiv \neg A \vee \neg B$

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 $\wedge$ (or $\neg$ and $\vee$). This property is very useful in circuit logic.

- $A \wedge B \equiv \neg(\neg A \vee \neg B)$
- $A \vee B \equiv \neg(\neg A \wedge \neg B)$
- $A \to B \equiv \neg A \vee B$
- $A \leftrightarrow B \equiv (A \to B) \wedge (B \to A)$

If for an assignment α and a formula A it holds that $\alpha(A) = 1$, we say that α *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 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 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*: $A \vee 0 \equiv A$ and $A \wedge 1 \equiv A$
- *Cancellation law*: $A \vee 1 \equiv 1$ and $A \wedge 0 \equiv 0$
- *Tertium non datur*: $A \vee \neg A \equiv 1$ and $A \wedge \neg A \equiv 0$

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:

$$\begin{aligned} A \vee B \rightarrow B \vee \neg C &\equiv \neg(A \vee B) \vee B \vee \neg C \\ &\equiv (\neg A \wedge \neg B) \vee B \vee \neg C \\ &\equiv ((\neg A \vee B) \wedge (\neg B \vee B)) \vee \neg C \\ &\equiv ((\neg A \vee B) \wedge 1) \vee \neg C \\ &\equiv \neg A \vee B \vee \neg C \end{aligned}$$

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

Through truth tables, evaluating if we can conclude A through $A_1, \ldots, A_n$ is pretty easy: $A_1, \ldots, A_n \models A$ if and only if for every row where $A_1, \ldots, A_n$ 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. $P \rightarrow (D \rightarrow T) \equiv \neg P \vee (\neg D \vee T) \equiv \neg P \neg D \vee T$
2. $P \rightarrow D \equiv \neg P \vee D$
3. $P \rightarrow T \equiv \neg P \vee D$

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 α 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 ϕ, ψ:

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

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

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

$$\psi = (\neg A \vee \neg B \vee A) \wedge \neg(\neg A \wedge \neg B) \wedge \neg(\neg A \wedge A) \wedge \neg(\neg B \wedge A)$$

We observe that ϕ cannot be reduced too much:

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

However, ψ can be reduced completely:

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

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

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

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 $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, 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:

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

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

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

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^{DNF} and a CNF formula F^{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 α_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 α 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 α it holds that $\alpha(A) = 1$ if and only if for some $j \in [2^n]$ it holds that α corresponds to α_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^{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 α 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 α it holds that $\alpha(A) = 0$ if and only if for some $j \in [2^n]$ it holds that α corresponds to α_j over $p_1, \ldots, p_n$. Hence, we conclude that $F \equiv F^{\mathsf{CNF}}$, where:

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

□

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 α 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 α 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 α_i, α_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 α 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 $T = \{\neg A \vee B, \neg A \vee C\}$, 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. 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: $T^* = \bigcup_{i \in \mathbb{N}} T_i$ 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 α^* 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 $p_{i_1}, \ldots, p_{i_k}$ 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 $A^* = \{A, p^*_{i_1}, \ldots, p^*_{i_k}\}$. Since $A^* \subseteq T^*$ and T^* is finitely satisfiable, there must be an assignment β_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 $\beta(p_{i_1}) = \alpha^*(p_{i_1}), \ldots, \beta(p_{i_k}) = \alpha^*(p_{i_k})$ and $p_{i_1}, \ldots, p_{i_k}$ 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. □

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:

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

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

- We add $(p_{1,\ell}, \dots, p_{2^\ell,\ell}) \in S_T$
- For all $i, j \in [2^\ell]$ we add $\neg(p_{i,\ell} \wedge p_{j,\ell}) \in S_T$
- 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 α 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_ℓ 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_ℓ is always satisfiable. □

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_{ℓ^*} 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 $\mathcal{L} = \{p_1, p_2, \ldots\}$ has infinite variables. Suppose that S is finitely satisfiable. For each $\ell \in \mathbb{N}$, let S_ℓ be the sub-theory containing all the propositions defined on the first ℓ variables.

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

Let T_S be the tree defined as follows:

- For each $\ell \in \mathbb{N}$, the i-th node of level ℓ is mapped to the i-th assignment $\alpha_{i,\ell} : \{p_1, \ldots, p_\ell\} \to \{0, 1\}$ that satisfies S_ℓ, 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_ℓ may be finite or infinite. If S_ℓ 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 ℓ. Suppose now that S_ℓ is infinite. Given an enumeration $S_\ell = \{A_1, A_2, \ldots\}$, for each $k \in \mathbb{N}$ we know that $S_\ell^k = \{A_1, \ldots, A_k\}$ is a finite subset of S, hence satisfiable by hypothesis. Thus, for each $k \in \mathbb{N}$ there is an assignment $\alpha_k : \{p_1, \ldots, p_\ell\}$ that satisfies S_ℓ^k.

However, since there are at most 2^ℓ satisfying assignments for S_ℓ, by the *Infinite Pigeonhole Principle* we know that there must be an assignment $\alpha^* : \{p_1, \ldots, p_\ell\}$ 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_ℓ and thus that there is at least one satisfying assignment in level ℓ. □

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. □

1.5 Decidability in logic

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, Σ^* denotes the set of all strings defined on an set of symbols Σ, 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 Σ^* in parallel and runs $\mathcal{B}$ 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

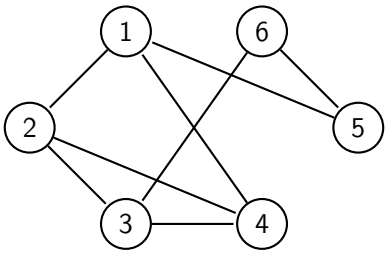
A set $S \subseteq \Sigma^*$ is decidable if and only if both S and $\overline{S}$ are semi-decidable.

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:

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

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

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

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 $\mathcal{A} = (A, R_1, \ldots, R_n, c_1, \ldots, c_k)$ where:

- A is a non-empty set called *domain*
- $R = \{R_1, \ldots, R_n\}$ are *relations* on A of any arity, even different ones.
- $c_1, \ldots, c_k$ 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 $\mathcal{N} = (\mathbb{N}, \leq, +, \times, 0, 1)$ is a relational structure on the natural numbers where 0 and 1 have been explicitly stated. In other words, we have that $\mathbb{N} = \{n_0, n_1, n_2, \ldots\}$ 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:

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

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 $x_1, \ldots, x_n$, logical connectives $\neg, \wedge, \vee, \ldots$, 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 $v_1, v_2, \ldots$ 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 $v_1, \ldots, v_2$, 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 $x_1, \ldots, x_n$ are distinct variables, we write $F(x_1, \ldots, x_n)$ to indicate that the free variables of F are contained in the set $x_1, \ldots, x_n$. This notation is also useful to deal with **substitution** of terms for variables: f $F(x_1, \ldots, x_n)$ is a formula and $t_1, \ldots, t_n$ are terms, we denote with $F(t_1, \ldots, t_n)$ (or $F(x_1/t_1, \ldots, x_n/t_n)$) 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 $\mathfrak{A}$ be an $\mathcal{L}$-structure. An **assignment** for $\mathfrak{A}$ is a function $\alpha : \mathsf{Vars}_{\mathcal{L}} \to \mathrm{Dom}(\mathfrak{A})$.

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

- F is atomic
- F is equal to $R(t_1, \ldots, t_n)$ and $(\alpha(t_1), \ldots, \alpha(t_n)) \in R^{\mathfrak{A}}$, where $t_1, \ldots, t_n$ 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 $\mathfrak{A} \overset{\alpha}{\not\models} H$

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

We observe that if F is a sentence then $\mathfrak{A} \overset{\alpha}{\models} F$ doesn't depend on α: 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 $\mathfrak{A}$, we say that it is **true** in $\mathfrak{A}$, written as $\mathfrak{A} \models F$. 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. $\mathfrak{A} \overset{\alpha}{\models} F$ means that F is true in $\mathfrak{A}$ over α
2. $\mathfrak{A} \models F$ means that F is true in $\mathfrak{A}$ for all α, i.e. $\forall \alpha$ it holds that $\mathfrak{A} \overset{\alpha}{\models} F$
3. $\models F$ means that F is true in $\mathcal{L}$, i.e. $\forall \mathfrak{A}$ it holds that $\mathfrak{A}\alpha \models F$

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

- $\mathfrak{A} \models T$ when $\mathfrak{A} \models F$ for all $F \in T$
- $\models T$ when $\models F$ for all $F \in T$

We observe that if F is finite then $\mathfrak{A} \overset{\alpha}{\models} F$ can always be decided through the inductive definition. If $\mathrm{Dom}(\mathfrak{A})$ is also finite then $\mathfrak{A} \models F$ 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.

2.2 Decision problems for predicate logic

After formalizing the three concepts of truthfulness in predicate logic, it's natural to ask if there concepts are treatable under a computational aspect, i.e. if they are decidable.

We start by considering the **validity problem**. Given a language $\mathcal{L}$, we say that a sentence S is *valid* (or a *tautology*) in $\mathcal{L}$ if $\models S$. The set of all valid sentences for a language $\mathcal{L}$ is denoted as $\mathsf{Val}_{\mathcal{L}}$. This problem has no algorithmic solutions (not even semi-decidable ones) for almost all non-trivial first-order languages, while it has algorithmic solutions for some restricted languages. We will be also interested in the version of the validity problem restricted to finite models, that is deciding the set of sentences that are true in all finite relational structures. We denote this set by $\mathsf{FinVal}_{\mathcal{L}}$.

An easier version of this problem is the **satisfiability problem**. Here, we're interested in deciding if there is an $\mathcal{L}$-structure $\mathfrak{A}$ that satisfies a fixed sentence S, i.e. $\mathfrak{A} \models S$. The

set of satisfiable sentences in a language $\mathcal{L}$ is denoted with $\mathsf{Sat}_{\mathcal{L}}$. Even though it is clearly simpler than $\mathsf{Val}_{\mathcal{L}}$, this set is also not semi-decidable.

What about the easiest type of truth evaluation? The **model-checking** problem asks to decide if for a structure $\mathfrak{A}$, a formula F and an assignment α it holds that $\mathfrak{A} \models F[\alpha]$. For infinite structures, the problem is semi-decidable. For finite structures, instead, the problem is decidable. In fact, we observe that structures "preserve completeness" in predicate logic. In particular, given any assignment α, a formula F and a structure $\mathfrak{A}$ over $\mathcal{L}$ it holds that:

$$\mathfrak{A} \not\models F[\alpha] \iff \mathfrak{A} \models \neg F[\alpha]$$

Hence, we can build a recursive algorithm over the inductive definition of $\mathfrak{A} \models F[\alpha]$ to decide it. The time complexity of such algorithm is polynomial in the size of the structure and exponential in the size of the formula. The space complexity, instead, is logarithmic in the size of the structure and polynomial in the size of the formula. In fact, the model-checking for first-order can be proven to be PSPACE-complete by reduction from TQBF, i.e. the Totally Quantified Boolean Formulas problem.

It is natural to consider formulas with free variables as defining subsets of a structure, by looking at the set of assignments that satisfy the formula. From the perspective of Database Theory, this means reading a formula as a query; from a more mathematical perspective this means investigating which subsets of a structure can be defined by formulas in the language. The **query evaluation** problem formalizes this question: given a structure $\mathfrak{A}$ and a formula $F(x_1, \ldots, x_n)$ in $\mathcal{L}$, the problem asks to decide the set $F^{\mathfrak{A}}$, defined as:

$$F^{\mathfrak{A}} = \{(a_1, \ldots, a_n) \in A^n \mid \mathfrak{A} \models F(a_1, \ldots, a_n)\}$$

Some structures/databases are more interesting than others. It is therefore natural to fix one such structure and investigate the set of sentences that are true in it. This set of sentences is called **theory of the structure**.

Definition 2.5: Theory of a structure

Given an $\mathcal{L}$-structure $\mathfrak{A}$, the **theory of** $\mathfrak{A}$ is the set $\mathrm{Th}(\mathfrak{A})$, defined as:

$$\mathrm{Th}(\mathfrak{A}) = \{S \text{ sentence} \mid \mathfrak{A} \models S\}$$

The decision problem for theories of particular structures is of fundamental interest in logic, database theory and from a more general mathematical perspective. Depending on the structure $\mathfrak{A}$ and the language considered, the problem can have one between the following answers:

- The problem is undecidable
- The problem is decidable but unfeasible
- The problem is decidable and feasible

For the particular case of finite structures, the problem is always decidable. The level of computational complexity for this problem is referred to as **expression complexity** (or *query complexity*). For first-order logic, this complexity is PSPACE.

In some cases, it's interesting to ask the inverse question, i.e. asking which structures are models for a given sentence, equivalent to deciding the set of **models of a sentence**.

Definition 2.6: Models of a sentence

Given a sentence S and a language $\mathcal{L}$, the **models of S** is the set $\mathrm{Mod}(S)$, defined as:

$$\mathrm{Mod}(S) = \{\mathfrak{A} \ \mathcal{L}\text{-structure} \mid \mathfrak{A} \models S\}$$

For possibly infinite structures, it might not even make sense to ask for an algorithm to decide membership in $\mathrm{Mod}(S)$, since there are infinitely many of them. For finite structures, instead, such an algorithm always exists and. The level of computational complexity for this problem is referred to as **structure complexity** (or *data complexity*). For first-order logic, this complexity is LOGSPACE.

2.3 Expressibility and structural isomorphisms

Consider a class $\mathcal{C}$ of structures adequate for some language $\mathcal{L}$ (e.g. the class of all graphs, the class of all finite graphs, ...). Consider now any property P of structures in the class $\mathcal{C}$. This property induces a bipartition on $\mathcal{C}$ through the set of all structures with property P and all those for which P doesn't hold (e.g. P can be the property of being a connected and $\mathcal{C}$ is the class of all simple undirected graphs). We can then ask whether there is a theory T in the language $\mathcal{L}$ which precisely defines the property P over the class $\mathcal{C}$ in the sense described above, i.e. such that for all $\mathfrak{A}$ in $\mathcal{C}$ it holds that $\mathfrak{A} \models T$ if and only if $P(\mathfrak{A})$ holds. If such sentence exists, we say that the property P is **expressible** in the language $\mathcal{L}$ over the class $\mathcal{C}$.

Definition 2.7: Expressibility

Let $\mathcal{L}$ be a language and let $\mathcal{C}$ be a class of $\mathcal{L}$-structures. Given a property P, we say that P is **expressible** (or *definable*, *axiomatizable*) in $\mathcal{L}$ over $\mathcal{C}$ if there is a theory T (a set of sentences) in $\mathcal{L}$ such that:

$$\forall \mathfrak{A} \in \mathcal{C} \ \ \mathfrak{A} \models T \iff P(\mathfrak{A}) \text{ holds}$$

If the theory T is finite, we say that P is **finitely expressible**.

The concept of expressibility plays a main role in first-order logic: it corresponds to the fundamental question "what can be said in first order logic?".

For instance, consider the property $P =$ "$\mathfrak{A}$ has exactly p elements". This property can

be expressed through the following sentence S^P:

$$S^P \equiv \exists x_1 \exists x_2 \dots \exists x_p \left(\bigwedge_{1 \leq i,j \leq p} \neg(x_i = x_j) \right) \wedge \left(\forall y \left(\bigvee_{1 \leq j \leq p} y = x_j \right) \right)$$

The first series of ANDs expresses that the domain contain at least p distinct elements, while the second series expresses that every additional element must be equal to one of the first p distinct elements.

The use of infinite theories allows us to express concepts that would otherwise be impossible to finitely express in first-order logic. For instance, consider the property $I =$ "$\mathfrak{A}$ has an infinite number of elements". For any $n \in \mathbb{N}$, let A_n be the property "$\mathfrak{A}$ has at least n elements". This last property can be expressed with a generalization of the first part of the previous sentence:

$$S^{A_n} \equiv \exists x_1 \exists x_2 \dots \exists x_n \bigwedge_{1 \leq i,j \leq n} \neg(x_i = x_j)$$

Consider now the following theory T^I:

$$T^I = \{S^{A_n} \mid n \in \mathbb{N}\}$$

where S^{A_n} is the sentence expressing A_n. It's easy to see that:

$$\mathfrak{A} \models T^I \iff I(\mathfrak{A}) \text{ holds}$$

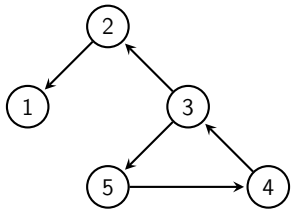
What about the property $\neg I =$ "$\mathfrak{A}$ has a finite number of elements"? Surprisingly, this simple property is **inexpressible** in first-order logic! To prove this, we can use the first-order logic version of the **compactness theorem**, in particular it's second formulation.

Theorem 2.1: Compactness theorem in FOL

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

We recall that satisfiability in FOL is defined through the existence of a satisfying structure for the theory. By way of contradiction, suppose that there is a theory $T^{\neg I}$ that expresses $\neg I$. Then, considering a new theory $T = T^I \cup T^{\neg I}$. For each finite subset of T we can build an ad-hoc structure that satisfies it, meaning that T is finitely satisfiable. Formally, given a finite subset $S \underset{fin}{\subseteq} T$, consider $S - T^{\neg I} = \{S^{A_{i_1}}, \dots, S^{A_{i_k}}\}$. Let $M = \max(i_1, \dots, i_k)$. Then, the structure $\mathfrak{A}$ with exactly M elements satisfies all the sentences in $S - T^{\neg I}$ and $T^{\neg I}$, concluding that S is satisfiable. By Compactness, T can be finitely satisfiable if and only if it is satisfiable. However, we know that this is absurd since T is clearly unsatisfiable since no structure can be finite and infinite at the same time. Hence, the only possibility is that $T^{\neg I}$ cannot exist, concluding that $\neg I$ is inexpressible in first-order logic.

Consider now the following directed graph:



Let $\mathcal{G} = (V, E)$ be the structure corresponding to the above graph. Given the following sentence S :

$$S \equiv \exists x_1 \exists x_2 \exists x_3 \exists x_4 \exists x_5 \, (E(x_2, x_1) \wedge E(x_3, x_2) \wedge E(x_3, x_5) \\ \wedge E(x_4, x_3) \wedge E(x_5, x_4)) \wedge \bigwedge_{1 \leq i,j \leq 5} \neg(v_i = v_j)$$

it's easy to see that $\mathcal{G}$ satisfies it. However, the sentence doesn't *exactly describe* $\mathcal{G}$. For instance, this sentence can be also satisfied by any graph extension of $\mathcal{G}$, that being graphs obtained by adding other nodes or edges to $\mathcal{G}$. In expressibility terms, the sentence S defines the property "$\mathfrak{A}$ is a graph with the edges $(v_2, v_1), \ldots, (x_5, x_4)$ for some $v_1, \ldots, v_5$".

To perfectly express $\mathcal{G}$, i.e. build a sentence that defined the property "$\mathfrak{A}$ is the graph $\mathcal{G}$" we have to specify that there are exactly six vertices and which edges do or do not exist between them:

$$S^{\mathcal{G}} \equiv \left(\bigwedge_{1 \leq i,j \leq 5} \neg(x_i = x_j) \right) \wedge \left(\forall y \, \left(\bigvee_{1 \leq j \leq 5} y = x_j \right) \right) \\ \wedge (\neg E(x_1, x_2) \wedge \neg E(x_1, x_3) \wedge \neg E(x_1, x_4) \wedge \neg E(x_1, x_5)) \\ \wedge (E(x_2, x_1) \wedge \neg E(x_2, x_3) \wedge \neg E(x_2, x_4) \wedge \neg E(x_2, x_5)) \\ \wedge (\neg E(x_3, x_1) \wedge E(x_3, x_2) \wedge \neg E(x_3, x_4) \wedge E(x_3, x_5)) \\ \wedge (\neg E(x_4, x_1) \wedge \neg E(x_4, x_2) \wedge E(x_4, x_3) \wedge \neg E(x_4, x_5)) \\ \wedge (\neg E(x_5, x_1) \wedge \neg E(x_5, x_2) \wedge \neg E(x_5, x_3) \wedge E(x_5, x_4))$$

However, we observe that $\mathcal{G}$ actually isn't the unique graph satisfying $S^{\mathcal{G}}$. In fact, any graph $\mathcal{G}'$ obtained by permuting the labels of the vertices of $\mathcal{G}$ also satisfies such sentence. From a mathematical prospective, the two graphs aren't equivalent but they are indistinguishable from each other under a logical level. When this happens, we say that the two structures are **isomorphic**.

Definition 2.8: Structural isomorphism

Let $\mathfrak{A}, \mathfrak{B}$ be two $\mathcal{L}$-structures. We say that $\mathfrak{A}$ and $\mathfrak{B}$ are **stucturally isomorphic**, written as $\mathfrak{A} \cong \mathfrak{B}$, when there is a bijective function $h : \text{Dom}(\mathfrak{A}) \to \text{Dom}(\mathfrak{B})$, called **isomorphism**, that *preserves truthfulness*, meaning that:

- For each relation symbol R of arity k in $\mathcal{L}$ and for all $(a_1, \ldots, a_n) \in A^n$ it holds that:
$$(a_1, \ldots, a_n) \in R^{\mathfrak{A}} \iff (h(a_1), \ldots, h(a_n)) \in R^{\mathfrak{B}}$$
- For each constant symbol c in $\mathcal{L}$ it holds that:
$$c^{\mathfrak{A}} = h(c^{\mathfrak{B}})$$

Since by definition structural isomorphism preserve truthfulness, it's easy to see that two isomorphic structures always satisfy the same sentences – it can be formally proven using the inductive definition of satisfiability. This concept is known as **elementary equivalence** between structures.

Definition 2.9: Elementary equivalence

Let $\mathfrak{A}, \mathfrak{B}$ be two $\mathcal{L}$-structures. We say that $\mathfrak{A}$ and $\mathfrak{B}$ are **elementarily equivalent** in $\mathcal{L}$, written as $\mathfrak{A} \equiv_{\mathcal{L}} \mathfrak{B}$, when $\text{Th}(\mathfrak{A}) = \text{Th}(\mathfrak{B})$.

Proposition 2.1

Let $\mathfrak{A}, \mathfrak{B}$ be two $\mathcal{L}$-structures. If $\mathfrak{A} \cong \mathfrak{B}$ then $\mathfrak{A} \equiv_{\mathcal{L}} \mathfrak{B}$

The latter proposition implies that any graph that is isomorphic to our example graph $\mathcal{G}$ satisfies $S^{\mathcal{G}}$. However, it's easy to see that this sentence can be satisfied only by structures that are isomorphic to $\mathcal{G}$. In other words, the sentence $S^{\mathcal{G}}$ **characterizes** the graph $\mathcal{G}$ up to isomorphism.

Theorem 2.2: Characterizing sentence

Let $\mathfrak{A}$ be a finite $\mathcal{L}$-structure. Then, there is a sentence $S^{\mathfrak{A}} \in \text{Th}(\mathfrak{A})$ called **characterizing sentence** such that for all $\mathcal{L}$-structures $\mathfrak{B}$ it holds that:

$$\mathfrak{A} \cong \mathfrak{B} \iff \mathfrak{B} \models S^{\mathfrak{A}}$$

Proof. Since $\mathfrak{A}$ is finite, we can build a sentence $S^{\mathfrak{A}}$ that exactly describes the domain and the relations of $\mathfrak{A}$. If $\mathfrak{A} \cong \mathfrak{B}$, we know that $\mathfrak{B} \models S^{\mathfrak{A}}$. Vice versa, if $\mathfrak{B} \models S^{\mathfrak{A}}$ then we know that this can happen only if $\mathfrak{A}$ and $\mathfrak{B}$ have the same structure. Thus, we can build an isomorphism $h : \text{Dom}(\mathfrak{A}) \to \text{Dom}(\mathfrak{B})$ by matching the quantifiers of the sentence $S^{\mathfrak{A}}$, concluding that $\mathfrak{B} \models S^{\mathfrak{A}}$. □

We observe that the characterizing sentence can be used as a medium to strengthen the previous proposition in the case of finite structures: if the two structures are elementarily equivalent then they will both satisfy the characterizing sentence, which automatically implies that they are isomorphic.

Corollary 2.1

Let $\mathfrak{A}, \mathfrak{B}$ be two finite $\mathcal{L}$-structures. Then, $\mathfrak{A} \cong \mathfrak{B}$ if and only if $\mathfrak{A} \equiv_{\mathcal{L}} \mathfrak{B}$

For general infinite structures, however, this corollary doesn't always hold, not even for countably infinite ones. In fact, we'l see that there exist equivalent structures that are not isomorphic. In some cases, however, it is possibile to characterize up to isomorphism a countable structure using sentences.

2.4 The back-and-forth method

In the previous section, we discussed how finite structures have a characterizing sentence. For countably infinite theories, we cannot guarantee the existence of such sentence since the trivial sentence describing exactly the structure cannot exist due to the structure being infinite (recall that formulas are finite).

To fix this issue, we can try to extend the concept to theories. The trivial candidate for this task is the set of all the sentences that are satisfiable by a structure, i.e. its theory. In fact, we already stated that if $\mathfrak{A} \cong \mathfrak{B}$ then $\mathfrak{A} \equiv_{\mathcal{L}} \mathfrak{B}$, which means that $\mathrm{Th}(\mathfrak{A}) = \mathrm{Th}(\mathfrak{B})$. Indeed, there are some countably infinite theories for which the contrary implication also holds, even thought not all of them do. We'll see some examples of the latter case in following sections. For now, we'll focus on showing a general approach called the **back-and-forth method** for proving that elementary equivalence can (sometimes) imply structural isomorphism.

Consider the structure $\mathcal{Q} = (\mathbb{Q}, <)$ describing the set of rational numbers $\mathbb{Q}$ and the less than equal relation $<$ over $\mathbb{Q}$. Let $\mathfrak{B} = (B, <^{\mathfrak{B}})$ be a countably infinite structure such that $\mathrm{Th}(\mathcal{Q}) = \mathrm{Th}(\mathfrak{B})$.

Since both $\mathcal{Q}$ and $\mathfrak{B}$ are countably infinite, $\mathbb{Q}$ and B can be enumerated. Let $a_1, a_2, \ldots$ be an enumeration of $\mathbb{Q}$ and let $b_1, b_2, \ldots$ be an enumeration of B. The idea is to inductively define two new enumerations $p_1, p_2, \ldots$ and $q_1, q_2, \ldots$ of $\mathbb{Q}$ and B such that the map $h : \mathbb{Q} \to B : p_i \mapsto q_i$ is an isomorphism.

We start by setting $p_0 = a_0$ and $q_0 = b_0$. For the inductive step, consider a generic n and assume that $p_0, p_1, \ldots, p_n$ and $q_0, q_1, \ldots, q_n$ have been defined. The back-and-forth method splits in two cases: when n is even, we pick a new element from $\mathbb{Q}$ and map it to a new element of B, otherwise we pick a new element from B and map it to a new element of $\mathbb{Q}$. The first case is called the **forth-step**, while the second case is called the **back-step**.

First, we observe that since $\mathrm{Th}(\mathcal{Q}) = \mathrm{Th}(\mathfrak{B})$, any expressible property that lies in $\mathrm{Th}(\mathcal{Q})$ must be also satisfied by $\mathfrak{B}$.

Forth-step. Suppose that n is even. Let p_{n+1} be the element in $\mathbb{Q} - \{p_0, p_1, \ldots, p_n\}$ with the smallest possible index in the enumeration $a_0, a_1, \ldots$. By comparing this new element with the previous ones, we have three cases:

1. For all $i \leq n$ we have that $p_{n+1} < p_i$. In this case, we set q_{n+1} as any element in B such that $q_{n+1} <^{\mathfrak{B}} q_i$ for all $i \leq n$. The existence of such element q_{n+1} is guaranteed since $\mathrm{Th}(\mathcal{Q})$ contains the sentence expressing *non-minimality*, i.e. $\forall x \exists y\ y < x$
2. For all $i \leq n$ we have that $p_i < p_{n+1}$. In this case, we set q_{n+1} as any element in B such that $q_i <^{\mathfrak{B}} q_{n+1}$ for all $i \leq n$. The existence of such element q_{n+1} is guaranteed since $\mathrm{Th}(\mathcal{Q})$ contains the sentence expressing *non-maximality*, i.e. $\forall x \exists y\ x < y$
3. The first two cases do not apply. In this case, there must be two indices $i, j \leq n$ with $i \neq j$ such that $p_i < p_{n+1} < p_j$ and $\nexists k \leq n$ such that $p_i < p_k < p_j$. We set q_{n+1} as any element in B such that $q_i < q_{n+1} < q_j$. The existence of such element q_{n+1} is guaranteed since $\mathrm{Th}(\mathcal{Q})$ contains the sentence expressing *density*, i.e. $\forall x \forall y\ (x < y \to \exists z\ (x < z) \wedge (z < y))$.

In all of the three cases, q_{n+1} is guaranteed to be a new element since $\mathrm{Th}(\mathcal{Q})$ contains the sentence expressing *irreflexivity*, i.e. $\forall x\ \neg(x < x)$. Moreover, the existence of each relation involving q_{n+1} used in the three cases is guaranteed by the sentences expressing *transitivity* and *totality*, i.e. $\forall x \forall y \forall z((x < y) \wedge (y < z) \to x < y)$ and $\forall x \forall y((x < y) \vee (y < x) \vee (x = y))$.

Back-step. Suppose that n is odd. Let q_{n+1} be the element in $B - \{q_0, q_1, \ldots, q_n\}$ with the smallest possible index in the enumeration $b_0, b_1, \ldots$. The element p_{n+1} is chosen through an argument similar to the three cases of the forth-step.

Since the back and forth step alternate between each other over each iteration of the inductive process, $\mathbb{Q}$ and B are guaranteed to be fully covered. Moreover, by choice of each next element is easy to see that $h : \mathbb{Q} \to B : p_i \mapsto q_i$ is indeed an isomorphism. This concludes that $\mathcal{Q} \cong \mathfrak{B}$. We also notice that the application of the back-and-forth method described above uses only a *restricted theory* of $\mathrm{Th}(\mathcal{Q})$, called **DLO (Dense Linear Order)**. In fact, the proof shown above works for structure that satisfies this finite theory, concluding that it suffices as a **characterizing theory** of $\mathcal{Q}$.

Definition 2.10: Dense Linear Order (DLO)

We define the **Dense Linear Order** as the finite theory DLO containing:

1. *Irreflexivity*: $\forall x\ \neg(x < x)$
2. *Transitivity*: $\forall x \forall y \forall z((x < y) \wedge (y < z) \to x < y)$
3. *Totality*: $\forall x \forall y((x < y) \vee (y < x) \vee (x = y))$.
4. *Non-minimality*: $\forall x \exists y\ y < x$.
5. *Non-maximality*: $\forall x \exists y\ x < y$.
6. *Density*: $\forall x \forall y\ (x < y \to \exists z\ (x < z) \wedge (z < y))$

Theorem 2.3: Characterizing theory of $(\mathbb{Q}, <)$

Let $\mathcal{Q} = (\mathbb{Q}, <)$. For any countably infinite structure $\mathfrak{B}$ it holds that:

$$\mathcal{Q} \cong \mathfrak{B} \iff \mathfrak{B} \models \mathsf{DLO}$$

In some cases, the characterizing sentence (or theory) of a class of structures may correspond to a particular property. This is the case of the **random graph model**, introduced by Erdős and Rényi [ER59]. Given a vertex set V, a random graph on V is a graph whose edges have a $\frac{1}{2}$ probability of being added to the graph.

$$\forall \{u, v\} \in \binom{V}{2} \quad \Pr[\{u, v\} \in E(G)] = \frac{1}{2}$$

We define the *Random Graph Property* (also known as *Extension Property* or *Alice's Restaurant Property*) as the property stating "for any pair of disjoint finite subsets $A, B \subseteq V$, there is a vertex outsite of A and B that is connected to all the vertices in A and disconnected from all the vertices in B". This property can be easily expresses in first-order logic through an infinite theory $T = \{S_{n,m} \mid n, m \in \mathbb{N}\}$, where:

$$S_{n,m} \equiv \forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_m \left(\bigwedge_{1 \le i,j \le n} \neg(x_i = y_j) \rightarrow \exists z \left(\bigwedge_{1 \le i \le n} \neg(z = x_i) \wedge E(z, x_i) \right) \wedge \left(\bigwedge_{1 \le j \le n} \neg(z = y_i) \wedge \neg E(z, y_i) \right) \right)$$

Erdős and Rényi were able to prove every sufficiently large random graph satisfies the random graph property $S_{n,m}$ for each $n, m \in \mathbb{N}$ and that every pair of structures satisfying the theory T are isomorphic to each other.

Theorem 2.4: Characterizing theory of random graphs

Any countable model satisfying the *Random Graph Property* is almost certainly isomorphic to a random graph.

Proof. First, we prove that any sufficiently large random graph almost certainly satisfies the property. Fix a pair of disjoint finite subsets $A, B \subseteq V$ and fix a vertex $v \in V - (A \cup B)$. Let $|V| = k\,|A| = n$ and $|B| = m$. Let Q_v be the event "$N(v) \subseteq A \wedge N(v) \cap B = \varnothing$". We observe that:

$$\Pr[\neg Q_v] = 1 - \Pr[Q_v] = 1 - \left(\frac{1}{2}\right)^n \left(\frac{1}{2}\right)^m = 1 - \frac{1}{2^{n+m}}$$

Hence, we have that:

$$\Pr[\forall v \in V - (A \cup B)\ \neg Q_v] = \left(1 - \frac{1}{2^{n+m}}\right)^{k-n-m}$$

Since not all finite subsets A of n elements and finite subsets B of m elements are disjoint and there are at most k^{n+m} pairs of subsets, the probability of $S_{n,m}$ not being satisfied is lower bounded by:

$$\Pr[S_{n,m}] = 1 - \Pr[\neg S_{n,m}] = 1 - \Pr[\exists A, B\ \forall v \in V - (A \cup B) \neg Q_v] \geq 1 - k^{n+m}\left(1 - \frac{1}{2^{n+m}}\right)^{k-n-m}$$

For sufficiently large random graphs, we conclude that:

$$\lim_{k \to +\infty} \Pr[S_{n,m}] \geq \lim_{k \to +\infty} 1 - k^{n+m}\left(1 - \frac{1}{2^{n+m}}\right)^{k-n-m} = 1$$

For the second part, let $\mathfrak{A} = (A, E^{\mathfrak{A}})$ and $\mathfrak{B} = (B, E^{\mathfrak{B}})$ be two structures satisfying the random graph property. Let $n = |A| = |B|$. We'll prove that these two structures are isomorphic through a back-and-forth argument. Let $a_1, a_2, \ldots$ be an enumeration of A and let $b_1, b_2, \ldots$ be an enumeration of B. We start by setting $p_0 = a_0$ and $q_0 = b_0$. For the inductive step, consider a generic n and assume that $p_0, p_1, \ldots, p_n$ and $q_0, q_1, \ldots, q_n$ have been defined.

Forth-step. Suppose that n is even. Let p_{n+1} be the element in $A - \{p_0, p_1, \ldots, p_n\}$ with the smallest possible index in the enumeration $a_0, a_1, \ldots$. We split $\{p_0, p_1, \ldots, p_n\}$ into two two disjoint subsets according to the connectedness on p_{n+1}:

$$X = \{i \mid i \leq n, (p_i, p_{n+1} \in E^{\mathfrak{A}})\}$$

$$Y = \{j \mid j \leq n, (p_j, p_{n+1} \notin E^{\mathfrak{A}})\}$$

Consider now the subsets X', Y' of H induced by X, Y:

$$X' = \{q_i \mid p_i \in X\}$$

$$Y' = \{q_j \mid p_j \in Y\}$$

By construction of the inductive process, since X, Y are disjoint X', Y' must also be disjoint. Hence, since $\mathfrak{B}$ satisfies the random graph property and the premise of the axiom $S_{|X'|,|Y'|}$, its consequence must also be satisfied, meaning that there is a vertex $v \in B - (X' \cup Y')$ that is adjacent to every vertex in X' and disconnected from each vertex in Y'. We fix $q_{n+1} = v$.

Back-step. Suppose that n is odd. Let q_{n+1} be the element in $B - \{q_0, q_1, \ldots, q_n\}$ with the smallest possible index in the enumeration $b_0, b_1, \ldots$. The element p_{n+1} is chosen through an argument similar to the forth-step.

Since the back and forth step alternate between each other over each iteration of the inductive process, $\mathbb{Q}$ and A are guaranteed to be fully covered. Moreover, by choice of each next element is easy to see that $h : A \to B : p_i \mapsto q_i$ is indeed an isomorphism. This concludes that $\mathfrak{A} \cong \mathfrak{B}$. □

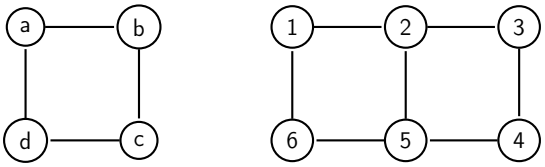
2.5 Bounded equivalence

Elementary equivalence can be used to prove non-expressibility results: if two structures $\mathfrak{A}$ and $\mathfrak{B}$ are elementarily equivalent but only one of them has the property P then the only possibility is that the property P cannot be expressed in first-order logic, since otherwise it would lie inside both $\text{Th}(\mathfrak{A})$ and $\text{Th}(\mathfrak{B})$.

Proposition 2.2: First method for non-expressibility in FOL

Let P be a property for a class of structures $\mathcal{C}$. Then, P is not expressible in FOL if there are two structures $\mathfrak{A}, \mathfrak{B}$ such that $\mathfrak{A} \equiv \mathfrak{B}$ but they don't share the property P.

However, we observe that this method for proving non-expressibility fails for finite models: we proved that in this case elementary equivalence implies that the two structures are isomorphic, meaning that the two models cannot differ by any sensible property P (as long as P is preserved under isomorphisms, a standard assumption in both *Mathematics* and *Computer Science*). To overcome this problem, we will consider a *graded* version of elementary equivalence, i.e. **bounded elementary equivalence**, which expresses the idea that two structures cannot be distinguished by sentences of a specific syntactic complexity. For instance, consider the following two graphs:



Many sentences are able to distinguish the two graphs, such as:

1. The sentence expressing "there is a node with at least three neighbors"

$$\exists x_1 \exists x_2 \exists x_3 \exists x_4 \bigwedge_{i,j} \neg(x_i = x_j) \wedge E(x_1, x_2) \wedge E(x_1, x_3) \wedge E(x_1, x_4)$$

   is satisfied only by the second graph.

2. The sentence expressing "there are three nodes that are independent"

$$\exists x_1 \exists x_2 \exists x_3 \bigwedge_{i,j} \neg(x_i = x_j) \wedge \neg E(x_i, x_j)$$

   is satisfied only by the second graph.

After some trial and error, we can convince ourselves that there seem to be no sentence with less than three quantifiers that is able to distinguish the two graphs: after fixing two nodes through two quantifiers, we require at least another quantifier to express any property. This fact can be formally proven, meaning that we strictly require at least

three quantifiers to distinguish the two graphs. This gives us some notion of *syntactic complexity* for predicate formulas: the **quantifier rank** of a formula.

Definition 2.11: Quantifier rank

Let F be a formula. The **quantifier rank** (or *degree*) of F, written as $\mathrm{rk}(F)$, is inductively defined as:

- If F is an atomic formula then $\mathrm{rk}(F) = 0$
- If $F = G * H$ for some boolean connective $*$ then $\mathrm{rk}(F) = \max(\mathrm{rk}(G), \mathrm{rk}(H))$
- If $F = \neg H$ then $\mathrm{rk}(F) = \mathrm{rk}(H)$
- If $F = Qx_1 \neg H$ for some quantifier Q then $\mathrm{rk}(F) = \mathrm{rk}(H) + 1$

We denote with $\mathrm{Th}_k(\mathfrak{A})$ the set of all sentences of rank at most k that are satisfied by $\mathfrak{A}$. As we discussed, the two previous example graphs satisfy the same sentences up to two quantifiers. This is known as 2-elementary equivalence.

Definition 2.12: k-elementary equivalence

Let $\mathfrak{A}, \mathfrak{B}$ be two $\mathcal{L}$-structures. We say that $\mathfrak{A}$ and $\mathfrak{B}$ are **k-elementarily equivalent** in $\mathcal{L}$, written as $\mathfrak{A} \equiv_k \mathfrak{B}$, when $\mathrm{Th}_k(\mathfrak{A}) = \mathrm{Th}_k(\mathfrak{B})$.

By definition, it holds that every structure for the same language is 0-equivalent and that $\mathfrak{A} \equiv_k \mathfrak{B}$ for all $k \in \mathbb{N}$ if and only if $\mathfrak{A} \equiv \mathfrak{B}$. The notion of k-elementary equivalence gives rise to a method for proving non-expressibility of queries over finite models. Suppose that a property P is *finitely axiomatizable*, meaning that there is a sentence expressing it. Then, the axioms have some maximal quantifier rank k. If we show that for any k there are two structures that don't share P but satisfy the same sentences up to that complexity then P is not finitely axiomatizable.

Proposition 2.3: Second method for non-expressibility in FOL

Let P be a property for a class of structures $\mathcal{C}$. Then, P is not finitely expressible in FOL if for all $k \in \mathbb{N}$ there are two structures $\mathfrak{A}, \mathfrak{B}$ such that $\mathfrak{A} \equiv_k \mathfrak{B}$ but they don't share the property P.

This method clearly works for both finite and non-finite structures, making it of interest to isolate methods for proving that $\mathfrak{A} \equiv_k \mathfrak{B}$. One such method was given by Fraïssé [Fra54], who used the back-and-forth method on **structure expansions**.

Definition 2.13: Structure expansion

Let $\mathfrak{A}$ be an $\mathcal{L}$-structure and let $c_1, \ldots, c_n$ be new constant symbols. Let $\mathcal{L}^c = \mathcal{L} \cup \{c_1, \ldots, c_n\}$. Given $a_1, \ldots, a_n \in A$, the **expansion** of $\mathfrak{A}$ over $a_1, \ldots, a_n$, written as $(\mathfrak{A}, a_1, \ldots, a_n)$, is the $\mathcal{L}^c$-structure that coincides with $\mathfrak{A}$ and interprets c_i as a_i.

We observe that the language $\mathcal{L}^c$ for structure $(\mathfrak{A}, a_1, \ldots, a_n)$ contains more sentences (in particular more atomic ones) than language $\mathcal{L}$. In general, if $F(x_1, \ldots, x_n)$ is a formula of $\mathcal{L}$, we denote with F^c the $\mathcal{L}^c$ formula obtained by substituting each x_i with c_i. By definition, we have that:

$$(\mathfrak{A}, a_1, \ldots, a_n) \models F^c \iff \mathfrak{A} \models F(x_1, \ldots, x_n)[a_1, \ldots, a_n]$$

To make things clearer, we illustrate an example. Consider the language $\mathcal{L} = \{U(x), M(x, y), P(x, y)\}$. Let $\mathfrak{A} = (A, U^{\mathfrak{A}}, M^{\mathfrak{A}}, P^{\mathfrak{A}})$ be a structure representing a database with a finite domain of individuals, i.e. $A = \{\text{George, Mary, } \ldots, \text{ Lisa}\}$, where $U^{\mathfrak{A}}$ is the relation of *male individuals*, $M^{\mathfrak{A}}$ is the relation of *married individuals* and $P^{\mathfrak{A}}$ is the relation of *parent individuals*. We expand the language $\mathcal{L}$ with the new constants c_1, c_2. While there are no atomic sentences in the original language, the following are atomic sentences in the expanded language $\mathcal{L}^c$: $U(c_1), U(c_2), M(c_1, c_2), M(c_2, c_1), S(c_2, c_1), \ldots$

Consider now the following formula in $\mathcal{L}$:

$$F(x_1, x_2) \equiv \exists x_3 \; P(x_1, x_3) \land P(x_2, x_3) \land M(x_1, x_2) \land U(x_3)$$

In the new language, this formula becomes:

$$F^c \equiv \exists x_3 \; P(c_1, x_3) \land P(c_2, x_3) \land M(c_1, c_2) \land U(x_3)$$

Fix two individuals in A, say George and Lisa (G, L for short). The formula F^c is true in the expansion of $(\mathfrak{A}, G, L)$ if and only if G and L are married and they have a common son, which is true if and only if $F(x_1, x_2)[G, L]$ is true in $\mathfrak{A}$.

Theorem 2.5: Fraïssé's theorem

Let $\mathfrak{A}, \mathfrak{B}$ be two $\mathcal{L}$-structures. For all $k \geq 0$, it holds that $\mathfrak{A} \equiv_{k+1} \mathfrak{B}$ if and only if the two following conditions hold:

1. *Forth condition*: for all $a \in A$ there is a $b \in B$ such that $(\mathfrak{A}, a) \equiv_k (\mathfrak{B}, b)$
2. *Back condition*: for all $b' \in B$ there is a $a' \in A$ such that $(\mathfrak{A}, a') \equiv_k (\mathfrak{B}, b')$

Proof. First, we observe that any formula of the form $\forall x G(x)$ can be rewritten as $\neg(\exists x \neg G(x))$. We also observe that any formula can be written in *prefix form*, meaning that all the quantifiers appear at the start of the formula. Hence, we can restrict our interest of formulas of the form $\exists x F(x)$ of rank at most k and with one free variable x.

Assume that the forth and back conditions hold. Suppose that $\mathfrak{A} \models \exists x F(x)$. Then, for some $a \in A$ it must hold that $\mathfrak{A} \models F(x)[a]$, which can happen if and only if $(\mathfrak{A}, a) \models F^c$.

By the forth condition, there must be some $b \in B$ such that $(\mathfrak{B}, b) \models F^c$, which can happen if and only if $\mathfrak{B} \models F(x)[b]$, concluding that $\mathfrak{B} \models \exists x F(x)$. Using the back condition and the same argument, we get that $\mathfrak{B} \models \exists x F(x)$ implies $\mathfrak{A} \models \exists x F(x)$. Thus, the two structures satisfy the same formulas of rank at most $k+1$, concluding that $\mathfrak{A} \equiv_{k+1} \mathfrak{B}$.

Vice versa, assume that $\mathfrak{A} \equiv_{k+1} \mathfrak{B}$. We prove that the forth condition holds (the back condition follows the same argument). Fix $a \in A$. We notice that the sentences of rank at most k in the language $\mathcal{L}^c$ either contain the new constant c – meaning that they are of the form F^c for some F in $\mathcal{L}$ with rank at most k – or they don't contain it – meaning that they are also in $\mathcal{L}$. In the second case, there is nothing to prove so we assume to be working with the first case.

For a generic structure $\mathfrak{D}$ and a value $d \in D$, we define the *k-type of d in $\mathfrak{D}$*, written as $Q_{\mathfrak{D},d}$, set of formulas with one free variable, rank at most k and that are satisfied by $\mathfrak{D}$ with assignment $x \mapsto d$.

$$Q_{\mathfrak{D},d} = \{F(x) \mid \mathrm{rk}(F(x)) \leq k, \mathfrak{D} \models F(x)[d]\}$$

Claim 1: There is a formula $H_{\mathfrak{D},d}(x)$ of rank at most k and with one free variable x that exactly describes $Q_{\mathfrak{D},d}$.

Proof of Claim 1. Let Q be the set of formulas with one free variable and rank at most k.

$$Q = \{F(x) \mid \mathrm{rk}(F(x)) \leq k\}$$

We observe that Q is finite modulo logical equivalence, meaning that we can consider an unique representative formula for every pair of formulas satisfied by the same assignments. Let $X = \{G_1(x), \ldots, G_m(x)\}$ be the set of such representative formulas. Since $Q_{\mathfrak{D},d} \subseteq Q$, it can be represented by a subset $X_{\mathfrak{D},d} = \{G_{i_1}(x), \ldots, G_{i_t}(x)\}$ of X. Let $I = \{i_1, \ldots, i_t\}$. Since I and $\overline{I}$ are finite, we can express through a single formula $H_{\mathfrak{D},d}(x)$ the fact that x satisfies all and only the formulas in $X_{\mathfrak{D},d}$.

$$H_{\mathfrak{D},d}(x) \equiv \bigwedge_{i \in I} G_i \wedge \bigwedge_{j \in \overline{I}} \neg G_j$$

Since each formula of X has rank at most k, $H_{\mathfrak{D},d}(x)$ is a formula of degree at most k with one free variable x. □

Through the first claim, we're able to encode a whole k-type into a single formula: proving that another structure satisfies such formula is equivalent to proving that the k-type of the initial structure.

Claim 2: There is a $b \in B$ such that $Q_{\mathfrak{A},a} = Q_{\mathfrak{B},b}$.

Proof of the Claim 2. By construction, we clearly have that $\mathfrak{A} \models H_{\mathfrak{A},a}(x)[a]$, which can happen if and only if $\mathfrak{A} \models \exists x H_{\mathfrak{A},a}(x)$. Since $\mathfrak{A} \equiv_{k+1} \mathfrak{B}$ by hypothesis, we get that $\mathfrak{B} \models \exists x H_{\mathfrak{A},a}(x)$, which can happen if and only if for some $b \in B$ we have that $\mathfrak{B} \models H_{\mathfrak{A},a}(x)[b]$, which can happen if and only if $Q_{\mathfrak{A},a} = Q_{\mathfrak{B},b}$. □

Consider now any sentence F^c in $\mathcal{L}^c$ with rank at most k and suppose that $(\mathfrak{A}, a) \models F^c$. Then, we know that this happens if and only if $\mathfrak{A} \models F(x)[a]$, meaning that $F(x) \in Q_{\mathfrak{A},a} = Q_{\mathfrak{B},b}$, which happens if and only if $(\mathfrak{B}, b) \models F^c$. □

2.6 Ehrenfeucht-Fraïssé games

The combinatorial characterization established by Fraïssé's theorem gives us a tool to reason on k-elementary equivalence. However, this tool is often impractical and very inconvenient since it basically requires a back-and-forth argument. Building on Fraïssé's result, Ehrenfeucht [Ehr60] was able to establish a more convient characterization in terms of a two player game, the now called **Ehrenfeucht-Fraïssé game** (or *Duplicator-Spoiler game*). Given two structures $\mathfrak{A}$ and $\mathfrak{B}$, on every round the first player – the *Spoiler* – picks an element from either $\mathfrak{A}$ or $\mathfrak{B}$ and the second player – the *Duplicator* – has to answer with an element in the other structure so that the sequence of choices defines a *partial isomorphism*.

Definition 2.14: Partial isomorphism

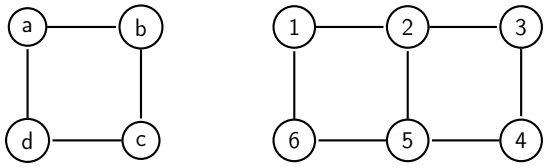
Let $\mathfrak{A}, \mathfrak{B}$ be two structures and let $a_1, \ldots, a_n \in A$ and $b_1, \ldots, b_n \in B$. The partial map $a_1 \mapsto b_1, \ldots, a_n \mapsto b_n$ is said to be a **partial isomorphism** between $\mathfrak{A}$ and $\mathfrak{B}$ when:

1. For all $i, j \leq n$ it holds that $a_i = a_j$ if and only if $b_i = b_j$
2. For all $i \leq n$ it holds that $a_i = c$ if and only if $b_i = c$
3. For every relation symbol R, for all $i_1, \ldots, i_k \in [n]$ it holds that $(a_{i_1}, \ldots, a_{i_k}) \in R^{\mathfrak{A}}$ if and only if $(b_{i_1}, \ldots, b_{i_k}) \in R^{\mathfrak{B}}$

After k rounds of the game, the two players will have built two sequences $a_1, \ldots, a_k$ of elements of A and $b_1, \ldots, b_k$ of B. The Duplicator wins the if the map $a_i \mapsto b_i$ preserves equality and relations, otherwise the Spoiler wins. When the Duplicator has a **strategy** to win all games of at most k rounds, we say that they win the k-round game, otherwise we say that the Spoiler wins the k-round game. We denote the k-round game on $\mathfrak{A}, \mathfrak{B}$ with $G_k(\mathfrak{A}, \mathfrak{B})$.

We observe that if $\mathfrak{A} \cong \mathfrak{B}$ are isomorphic then the Duplicator wins the k-game for any possible $k \in \mathbb{N}$ since they always have an adequate answer due to the two structures being indistinguishable from each other. Vice versa, if the Duplicator is able to win the n-game, with $|A| = |B| = n$, then $\mathfrak{A} \cong \mathfrak{B}$ since the partial isomorphism defined by the game covers the whole set.

To give a concrete example, consider again the following two example graphs.



Suppose that the Spoiler starts the first round by playing the node 2. To preserve the partial isomorphism, the Duplicator answers with the node a, trivially winning the first round since there are no unary relations to be satisfied and no equality constraints to check. For the second round, suppose that the Spoiler plays the node c. If the Duplicator answers with the node 6, they also win the second round since $a \neq c$, $2 \neq 6$ and $(b, d) \notin E^{\mathcal{G}_1}, (2, 6) \notin E^{\mathcal{G}_2}$. For the third round, suppose that the Spoiler plays the node 4. Then, the Duplicator has no way to answer while also preserving the partial isomorphism: if he answers with b then we have $(a, b), (b, c) \in E^{\mathcal{G}_1}$ but $(2, 4), (4, 6) \notin E^{\mathcal{G}_2}$, while if he answers with d then we have $(a, d), (d, c) \in E^{\mathcal{G}_1}$ but $(2, 4), (4, 6) \notin E^{\mathcal{G}_2}$.

Thus, there is a strategy for the Spoiler to win the 3-round game. Moreover, it can be easily proven that the Duplicator always has a strategy to win every single 2-round game. This has an interesting correspondence with the quantifier complexity of sentences distinguishing $\mathcal{G}_1$ from $\mathcal{G}_2$: we already notices that they can be distinguished by a sentence of rank at least 3, meaning that $\mathcal{G}_1 \not\equiv_3 \mathcal{G}_2$, but the two structures cannot be distinguished by a sentence of smaller quantifier rank, meaning that $\mathcal{G}_1 \equiv_2 \mathcal{G}_2$.

Lemma 2.1

Let $\mathfrak{A}, \mathfrak{B}$ be two structures and let $a_1, \dots, a_n \in A$ and $b_1, \dots, b_n \in B$. Then, for any $k \geq 1$, the Duplicator wins the k-game on $(\mathfrak{A}, a_1, \dots, a_n)$ and $(\mathfrak{B}, b_1, \dots, b_n)$ if and only if the following two conditions hold:

1. For all $a \in A$ there is a $b \in B$ such that the Duplicator wins the $(k-1)$-game on $(\mathfrak{A}, a_1, \dots, a_n, a)$ and $(\mathfrak{B}, b_1, \dots, b_n, b)$
2. For all $b' \in B$ there is a $a' \in B$ such that the Duplicator wins the $(k-1)$-game on $(\mathfrak{A}, a_1, \dots, a_n, a')$ and $(\mathfrak{B}, b_1, \dots, b_n, b')$

Proof. Suppose that the Duplicator wins the k-game on $(\mathfrak{A}, a_1, \dots, a_n)$ and $(\mathfrak{B}, b_1, \dots, b_n)$. On the first round, the Spoiler can play any $a \in A$ (or any $b \in B$). Since $k \geq 1$, the Duplicator has always an answer $b \in B$ (or $a \in A$) that maintains the partial isomorphism. For any pair (a, b) chosen on the first round, the sequences $a_1, \dots, a_n, a \mapsto b_1, \dots, b_n, b$ form a partial isomorphism. Thus, the Duplicator has a strategy to win the $(k-1)$ game on $(\mathfrak{A}, a_1, \dots, a_n, a)$ and $(\mathfrak{B}, b_1, \dots, b_n, b)$, hence the first condition holds. The second condition holds through an analogous argument.

Vice versa, suppose that the two conditions holds. Then, for any choice $a \in A$ (or $b \in B$) of the Spoiler, the Duplicator can pick an element $b \in B$ (or $a \in A$) that allows him to win the $k-1$ game on $(\mathfrak{A}, a_1, \dots, a_n, a)$ and $(\mathfrak{B}, b_1, \dots, b_n, b)$. The first round is one through the pair (a, b), while the other $k-1$ rounds are won through the same strategy that allows him to win the $k-1$ game. □

It's easy to see that the above lemma is very similar to Fraïssé's theorem. In fact, Ehrenfeucht was able to extend the lemma in order to prove that his game characterization is equivalent to the combinatorial charaterization.

Theorem 2.6: Ehrenfeucht's theorem

Let $\mathfrak{A}, \mathfrak{B}$ be two $\mathcal{L}$-structures. For all $k \geq 0$, it holds that $\mathfrak{A} \equiv_k \mathfrak{B}$ if and only if the Duplicator wins $G_k(\mathfrak{A}, \mathfrak{B})$.

Proof. We proceed by induction. Suppose that the Duplicator wins the 0-game. Then, this happens if and only if there is a partial isomorphism $h : c^{\mathfrak{A}} \mapsto c^{\mathfrak{B}}$ on the constant symbols. By definition of partial isomorphism, for each set of constants $c_{i_1}, \ldots, c_{i_k}$ we have that $(c^{\mathfrak{A}}_{i_1}, \ldots, c^{\mathfrak{A}}_{i_k}) \in R^{\mathfrak{A}}$ iff $(c^{\mathfrak{B}}_{i_1}, \ldots, c^{\mathfrak{B}}_{i_k}) \in R^{\mathfrak{B}}$ and $c^{\mathfrak{A}}_i = c^{\mathfrak{A}}_j$ iff $c^{\mathfrak{B}}_i = c^{\mathfrak{B}}_j$. Hence, $\mathfrak{A}$ and $\mathfrak{B}$ satisfy the same atomic sentences. Since all rank 0 sentences are atomic, we conclude that $\mathfrak{A} \equiv_0 \mathfrak{B}$. Vice versa, if $\mathfrak{A} \equiv_0 \mathfrak{B}$, we can create a partial isomorphism on the constant using the equivalences induced by the atomic sentences, thus the Duplicator always wins the 0-game.

Assume that the theorem holds for all $m \leq k$ and consider the case $k+1$. By the previous lemma, the Duplicator wins $G_{k+1}(\mathfrak{A}, \mathfrak{B})$ if and only if the following two conditions hold:

- For all $a \in A$ there is a $b \in B$ such that the Duplicator wins the k-game on $(\mathfrak{A}, a)$ and $(\mathfrak{B}, b)$
- For all $b' \in B$ there is a $a' \in A$ such that the Duplicator wins the k-game on $(\mathfrak{A}, a')$ and $(\mathfrak{B}, b')$

By inductive hypothesis, these two conditions are equivalent to:

- For all $a \in A$ there is a $b \in B$ such that the Duplicator wins the k-game on $(\mathfrak{A}, a) \equiv_k (\mathfrak{B}, b)$
- For all $b' \in B$ there is a $a' \in A$ such that the Duplicator wins the k-game on $(\mathfrak{A}, a') \equiv_k (\mathfrak{B}, b')$

Finally, by Theorem 2.5, this can happen if and only if $\mathfrak{A} \equiv_{k+1} \mathfrak{B}$. □

2.6.1 Non-expressibility results

After proving that the Ehrenfeucht-Fraïssé games can be used to decide the k-equivalence between two structures, we'll show some examples on how to apply the game in order to prove that a property is non-expressible in first-order logic.

Proposition 2.4

The Even Cardinality query is not first-order definable over the language of Graph Theory.

Proof. By way of contradiction, suppose that the query is expressible by the sentence S. Then, since the expression is finite, it must have a quantifier rank. Let $\mathrm{rk}(S) = k$. Let $\mathfrak{A}$ and $\mathfrak{B}$ respectively be the to totally disconnected graphs with $k+1$ and $k+2$ nodes. It's easy to see that the Delayer can win $G_k(\mathfrak{A}, \mathfrak{B})$ since the only relation that has to be preserved is the total disconnection. By Theorem 2.6, this concludes that $\mathfrak{A} \equiv_k \mathfrak{B}$.

However, if k is even then $k+1$ is odd and $k+2$ is even, otherwise $k+1$ is even and $k+2$ is odd. Hence, only one between $\mathfrak{A}$ and $\mathfrak{B}$ can satisfy S, raising a contradiction. □

We observe that the fact that the *EC* query is not expressible over finite graphs does not imply that it cannot be defined over another class of structures. In particular it does not imply that the query is not definable over a smaller class of structures: restricting the class could give more chances of expressing a property. For example, the above proof does not rule out that the query is not definable over the class of graphs that are not totally disconnected, since the structures used in the proof would not belong to that smaller class of structures. We'll now focus on a more complex example, showing that the Even Cardinality query also doesn't hold in the class of finite linear orders.

Proposition 2.5

Given $k > 0$, if $\mathfrak{A}$ and $\mathfrak{B}$ are two linear orders with at least 2^k elements then $\mathfrak{A} \equiv_k \mathfrak{B}$.

Proof. To prove this, we show that the Duplicator has a winning strategy for the game with k moves on $\mathfrak{A}$ and $\mathfrak{B}$. To make the proof easier to read, we extend the language with two constant symbols that denote the minimum and the maximum of the linear order, i.e. the first and last element of the order. This extension of the language is equivalent to assuming that the Duplicator plays the minimum in one structure if the Spoiler plays the minimum in the other structure, and similarly for the maximum. It's easy to see that this assumption is solid: in order for the Duplicator to survive, the minimum must always be mapped to the minimum (and similarly for the maximum) since otherwise the Spoiler immediately wins the next round by asking for a predecessor of the minimum (or a successor of the maximum). Hence, we assume that these two moves are played in the first two rounds.

Let a_{-1}, a_0 and b_{-1}, b_0 be the interpretations in $\mathfrak{A}$ and in $[B]$ of min and max. Let d be a distance function on the order, i.e. $d(x, y) = |x - y|$. If $d(x, y) < n$ we say that they are *n-close*. Let $B_n(x) = \{y \mid d(x, y) < n\}$ be the set of elements that are n-close from y.

Fix a round $i \in \mathbb{N}$. Let $a_1, \ldots, a_i$ be the elements played in $\mathfrak{A}$ after move i and let $b_1, \ldots, b_i$ be the corresponding elements player in $\mathfrak{B}$.

Claim: the Duplicator has a strategy such that, after every i-th round of the game, for every $-1 \leq p, q \leq i$ the following three points are satisfied:

1. If a_q is 2^{k-i}-close a_p then $d(b_p, b_q) = d(a_p, a_q)$
2. If a_q is not 2^{k-i}-close a_p then $d(b_p, b_q) \geq 2^{k-1}$
3. $a_p < a_q$ iff $b_p < b_q$ and $a_p = a_q$ iff $b_p = b_q$

Proof of the claim. We proceed by induction on i. If $i = 0$ then $-1 \leq p, q \leq 0$. Since $\mathfrak{A}$ and $\mathfrak{B}$ have more than 2^k elements, we know that $d(a_{-1}, a_0), d(b_{-1}, b_0) \geq 2^k$. The first condition is vacously satisfied, while the other two are true.

Assume that the claim holds for all $j \leq i$. Consider the round case $i+1$. We observe that the elements $a_{-1}, a_0, a_1, \ldots, a_i$ that have already been played partition A into $i+1$ intervals. Suppose that the spoiler picks an element $a_{i+1} \in A$. Then, $a_j < a_{i+1} < a_\ell$ for some $j, \ell \leq i$ such that the interval does not contain other alrady played elements (this is where we need the assumption that a_{-1}, a_0 have been already played). We refer to such open interval $]a_j, a_\ell[$ as *fresh interval*.

Since $a_{-1}, a_0, a_1, \ldots, a_i$ and $b_{-1}, b_0, b_1, \ldots, b_i$ is an order isomorphism by the third point of the inductive hypothesis, the open interval $]b_j, b_\ell[$ is also a fresh interval, meaning that there are no elements of B already played in the interval $]b_j, b_\ell[$. We distinguish two cases depending on the size of the interval $[a_j, a_\ell]$.

1. $d(a_j, a_\ell) < 2^{k-i}$. By the first point of the inductive hypothesis, $d(a_j, a_\ell) = d(b_j, b_\ell)$. Hence, we can choose b_{i+1} as a value in the interval $]b_j, b_\ell[$ with $d(a_j, a_{i+1}) = d(b_j, b_{i+1})$ and $d(a_{i+1}, \ell) = d(b_{i+1}, b_\ell)$.
2. $d(a_j, a_\ell) \geq 2^{k-i}$. By the second point of the inductive hypothesis, $d(b_j, b_\ell) \geq 2^{k-i}$. Hence, the intervals $[a_j, a_\ell]$ and $[b_j, b_\ell]$ are split into two intervals with at least 2^{k-i-1} elements. We distinguish between three subcases depending on whether a_{i+1} is closer to the left-end point or to the right end-point:
   (a) If $d(a_j, a_{i+1}) < 2^{k-i-1}$ then $d(a_{i+1}, a_\ell) \geq 2^{k-1-i}$. Thus, we can chose b_{i+1} in $[b_j, b_\ell]$ with $d(a_j, a_{i+1}) = d(b_j, b_{i+1})$ in order to satisfy $d(b_{i+1}, b\ell) \geq 2^{k-i-1}$.
   (b) If $d(a_{i+1}, a_\ell) < 2^{k-i-1}$ then $d(a_j, a_{i+1}) \geq 2^{k-1-i}$. Thus, we can chose b_{i+1} in $[b_j, b_\ell]$ with $d(a_{i+1}, a_\ell) = d(b_{i+1}, b_\ell)$ in order to satisfy $d(b_j, b_{i+1}) \geq 2^{k-i-1}$.
   (c)
   (d) If $d(a_j, a_{i+1}) \geq 2^{k-i-1}$ and $d(a_{i+1}, a_\ell) \geq 2^{k-i-1}$ then we can chose b_{i+1} from the middle of $[b_j, b_\ell]$, i.e. with $d(b_j, b_{i+1}) \geq 2^{k-i-1}$ and $d(b_{i+1}, b_\ell) \geq 2^{k-i-1}$. One such element always exists since $d(b_j, b_\ell) \geq 2^{k-i}$

The choices made in the preceding cases are sufficient to prove the validity of the three points of the inductive hypothesis for $i+1$: in every case the three points are met for the pair of points a_{i+1} and a_h with $-1 \leq h \leq i$. □

We notice that only the third point is necessary in order for the Duplicator to win the k-game since it defines a partial order, while the other two conditions are only needed for the induction to hold. □

Corollary 2.2

The Even Cardinality query is not first-order definable over the class of Finite Linear Orders.

Proof. By way of contradiction, suppose that the query is expressible by the sentence S. Given $\mathrm{rk}(S) = k$, let $\mathfrak{A}$ and $\mathfrak{B}$ respectively be two finite linear orders with 2^k and $2^k + 1$ elements. Through the previous proposition, we know that $\mathfrak{A} \equiv_k \mathfrak{B}$. However, only $\mathfrak{A}$ satisfies S, raising a contradiction. □

2.6.2 Infinite rounds games

The method of Ehrenfeucht-Fraïssé (EF) games extends naturally beyond finite structures. Specifically, the fact that the Duplicator wins the $G_k(\mathfrak{A}, \mathfrak{B})$ for any $k \in \mathbb{N}$ if and only if $A \equiv B$ also holds for infinite structures as well.

Theorem 2.7

Given two structures $\mathfrak{A}, \mathfrak{B}$, the Duplicator wins $G_k(\mathfrak{A}, \mathfrak{B})$ for all $k \in \mathbb{N}$ if and only if $A \equiv B$

However, we have discussed how some infinite strucures may be elementarily equivalent but not isomorphic to each other. For instance, consider the structure $\mathfrak{A} = (\mathbb{Z}, <)$ describing the common integers with the less than relation and the structure $\mathfrak{B} = (\{0, 1\} \times \mathbb{Z}, <^{\mathfrak{B}})$ describing the disjoint union of two copies of $(\mathbb{Z}, <)$, ordered such that every element in the first copy is less than every element in the second copy, i.e. $(m_1, n_1) <^{\mathfrak{B}} (m_2, n_2)$ if and only if $(m_1 < m_2)$ or $((m_1 = m_2) \wedge (n_1 < n_2))$. It is straightforward to verify that for every $k \in \mathbb{N}$, the Duplicator can win the k-round EF-game on $\mathfrak{A}$ and $\mathfrak{B}$, concluding that $\mathfrak{A} \equiv \mathfrak{B}$. However, $\mathfrak{A}$ and $\mathfrak{B}$ are clearly not isomorphic: the structure $\mathfrak{B}$ has pairs of elements (such as the zeros in each copy of $\mathbb{Z}$) that are infinitely far apart in the order, a property not present in $\mathfrak{A}$.

To distinguish more precisely between elementary equivalence and isomorphism in infinite structures, we consider EF-games on **infinite rounds**, written as $G_\infty(\mathfrak{A}, \mathfrak{B})$. In fact, in the previous example we're able to prove that the two structures are elementarily equivalent only because the strategy used by the Duplicator for the $(k+1)$-game is not necessarily an extension of the strategy used for the k-game. This allows the Duplicator to fool the Spoiler for any fixed amount of rounds, but not for infinitely many rounds.

When the Duplicator has an infinite strategy, by the very definition of EF-game we're able to construct an isomorphism between two infinite structures. Vice versa, if two structures are isomorphic, the Duplicator can win the ∞-game through the isomorphism itself.

Theorem 2.8

Given two infinite structures $\mathfrak{A}, \mathfrak{B}$, the Duplicator wins $G_\infty(\mathfrak{A}, \mathfrak{B})$ if and only if $\mathfrak{A} \cong \mathfrak{B}$

Consider the structures $\mathfrak{A} = (\mathbb{Q}, <)$ and $\mathfrak{B} = (\mathbb{R}, <)$. We can easily show that the Duplicator wins the ∞-game on $\mathfrak{A}$ and $\mathfrak{B}$. At any round i, suppose the sequences $a_1, \ldots, a_i$ in A and $b_1, \ldots, b_i$ in B have been chosen. If the Spoiler picks an element in A, we distinguish three cases:

1. The new element is smaller than all $a_1, \ldots, a_i$.
2. The new element is larger than all $a_1, \ldots, a_i$.
3. The new element lies in an open interval (a_n, a_m) for some $n, m \in [i]$ and in no smaller such interval defined by earlier moves.

In all cases, the Duplicator can select a corresponding element in B to preserve the partial isomorphism. The argument is symmetric if the Spoiler plays in B. This yields a general result about dense linear orders.

Proposition 2.6

All models (countable and uncountable) of DLO are isomorphic.

An important corollary is the **completeness** of the theory DLO: for any sentence φ, either DLO $\models \varphi$, meaning that all models of DLO satisfy φ, or DLO $\models \neg\varphi$. This completeness property will be of interest later from a computational standpoint.

2.7 Inexpressibility through logical reductions

Just as in Computability Theory and Computational Complexity, one can derive inexpressibility results in logic through the method of **reduction**. Generally, this involves showing that a property or query Q is not definable by arguing that if Q were definable, then another known-to-be-undefinable property P would also be definable. Since P is not definable, we conclude that Q is likewise not definable.

The nature of the reduction depends on the context. In complexity theory, reductions are typically required to be polynomial-time computable; in computability theory, they must be computable functions. In our logical setting, however, reductions must be **expressible within a given logical language**. As an application of this method, we can derive the following result from the known inexpressibility of the even cardinality query over finite linear orders.

Proposition 2.7

Connectivity is not first-order definable over finite graphs.

Proof. We reduce the even cardinality query on finite linear orders to connectivity in graphs.Given a finite linear order L, we construct a directed graph $G_L = (V_L, E_L)$ such that G_L is connected if and only if L has an *odd* number of elements. The construction is as follows:

- The vertex set V_L is the domain of L.
- The edge relation E_L is defined by:
  1. For each $x \in L$, place an edge from x to its *second successor* in L (if it exists).
  2. Add an edge from the last element to the second element of L.
  3. Add an edge from the penultimate element to the first element.

This construction is uniform in L and defines E_L from the order structure alone: L has even cardinality if and only if G_L is disconnected. This yields a logical reduction: we aim

to express this graph construction using a first-order formula in the language of linear orders. Let $F(x, y)$ be a first-order formula in the language $\{<\}$ such that for any linear order L and any $a, b \in L$ it holds that:

$$L \models F(a, b) \iff (a, b) \in E_L$$

To define $F(x, y)$, we translate the construction of E_L using first-order formulas. For instance, the successor relation can be defined as:

$$S(x, y) := (x < y) \wedge \forall z \neg(x < z \wedge z < y)$$

which holds if and only if y is the immediate successor of x. Similarly, one can define the second successor, first, last, and penultimate elements using first-order logic (see Exercise).

Now suppose, for contradiction, that connectivity is first-order definable in the language of graphs. Let C be such a sentence. We replace every occurrence of $E(x, y)$ in C with $F(x, y)$ to obtain a sentence C^* in the language of linear orders such that for any finite linear order L it holds that:

$$L \models C^* \iff G_L \models C$$

Claim: C^* expresses the *odd cardinality* property over finite linear orders

Proof of the claim. If $L \models C^*$, then $G_L \models C$ and G_L is connected, hence L has odd cardinality. Vice versa, if L has odd cardinality, then G_L is connected, so $G_L \models C$ and $L \models C^*$. □

Therefore, $\neg C^*$ would define the even cardinality query over finite linear orders, contradicting the known result that this query is not first-order definable. □

Several other graph properties can be shown to be non-expressible in first-order logic using similar reductions, such as *planarity*, *acyclicity*, *k-colorability*. In fact, this reduction method is **complete** for proving inexpressibility in first-order logic over finite models. i.e. if a query is not definable in first-order logic, then this can be established via a reduction of the kind described above.

Theorem 2.9: Completeness of FOL under inexpressibility

Any property that isn't expressible in FOL can be reduced to all other inexpressible properties in FOL.

Proof. Omitted. □

2.8 Quantifier elimination

As previously mentioned, the Compactness Theorem allows us to show that if T is a computable set of sentences, then the set $\{S \mid T \models S\}$ is semi-decidable. Moreover, if

T is a complete theory,i.e. for every sentence S, either $T \models S$ or $T \models \neg S$, this semi-decidability implies that the set of logical consequences of T is actually decidable.

This general result highlights the significance of complete theories. For instance, it sheds light on the early 20th-century interest among logicians in determining whether the **first-order Peano Axioms** form a complete theory. However, the decision procedure derived from the above general argument is not customized for any specific theory T, and thus, it is generally not efficient. This naturally leads to the question of whether certain theories admit more efficient, ad hoc decision algorithms. In what follows, we will explore some examples and introduce a general approach for constructing such algorithms.

One notable technique is **Quantifier Elimination**, originally introduced by Tarski [Tar98] to establish the algorithmic decidability of sentence truth in certain (infinite) mathematical structures. The core idea of quantifier elimination is to show that every formula in the language is logically equivalent—relative to truth in the structure under consideration—to a *quantifier-free formula* (also known as an *open formula*). If there exists an algorithm that can decide the truth of open formulas in the structure, and if the process of transforming formulas into their quantifier-free equivalents is effective, then this yields a decision procedure for the theory of the structure.

To build some intuition, we give some examples. Consider the formula $\exists y\ x < y \wedge y < z$ and the structure $\mathcal{Q} = (\mathbb{Q}, <)$. For any $a, b \in \mathcal{Q}$, we observe that:

$$\mathcal{Q} \models \exists y\ x < y \wedge y < z \begin{bmatrix} x & z \\ a & b \end{bmatrix} \iff \mathcal{Q} \models x < z \begin{bmatrix} x & z \\ a & b \end{bmatrix}$$

In other words, we have that:

$$\mathcal{Q} \models (\exists y\ x < y \wedge y < z) \leftrightarrow (x < z)$$

This shows that, within $\mathcal{Q}$, the quantified formula $\exists y\ x < y \wedge y < z$ is equivalent to the quantifier-free formula $x < z$. In mathematical practice, many properties that are typically defined using quantifiers can be shown – at least relative to specific structures – to be equivalent to properties expressible using open formulas. Consider the formula $\exists x\ ax^2 + bx + c = 0$, where a, b, c are constants of the language. In the field of the reals, we know that this formula is satisfied if and only if the following open formula is satisfied.

$$(a \neq 0 \wedge b^2 - 4ac \geq 0) \vee (a = 0 \wedge (b \neq 0 \vee c = 0))$$

In the complex field, instead, we have an equivalence with an even simpler open formula:

$$(a \neq 0 \vee b \neq 0 \vee c = 0)$$

Theorem 2.10: Tarski's theorem

For every formula $F(x_1, \ldots, x_n)$ with n free variables in the language of orders $\mathcal{L} = \{<\}$ there exists a quantifier-free formula $F'(x_1, \ldots, x_n)$ such that:

$$\mathsf{DLO} \models ((\forall x_1 \ldots \forall x_n \; F(x_1, \ldots, x_n)) \leftrightarrow F'(x_1, \ldots, x_n))$$

We say that F' is **DLO-equivalent** (or *equivalent modulo* DLO) to F. Moreover, F' can be found algorithmically from F.

Proof. Let $\mathcal{Q} = (\mathbb{Q}, <)$. Through Theorem 2.3 we know that $\mathrm{Th}(\mathsf{DLO}) = \mathrm{Th}(\mathcal{Q})$, hence we can restrict our interest to $\mathcal{Q}$. The proof itself defines the algorithmic procedure that outputs the resulting open formula. Let F be a formula in the language of orders. We can assume that F is in *prenex normal form*, i.e. $F = Q_1 x_1 \ldots Q_n x_n \; G$ where G is an open formula and each Q_i in chosen from $\{\forall, \exists\}$.

We show how to transform it into an open formula consisting of a combination of atomic formulas. At each step we preserve equivalence modulo $\mathcal{Q}$, in the sense that if a formula A is transformed into a formula B then we want to ensure that $\mathcal{Q} \models \forall \overrightarrow{x} \; (A \leftrightarrow B)$, where $\overrightarrow{x}$ contains all free variables occurring in A and B. We aim at an inductive procedure working from the innermost quantifier, that being $\exists x_n$, to the outermost one.

We make some assumptions on the formulas that we're working with. First of all, we can assume that the formula G – restricted to the innermost quantifier – has the form $\exists x_n \; G$ since $\forall$ quantifiers can be reduced to existential ones by double negation. Moreover, we can also assume that G is written in DNF form, meaning that:

$$G \equiv \bigvee_i \bigwedge_j L_{i,j}$$

where each $L_{i,j}$ is a *first-order literal*, that being atomic formulas or negations of atomic formulas. Finally, we also observe that:

$$\mathcal{Q} \models \left((\exists x_n \; G) \leftrightarrow \left(\exists x_n \; \bigvee_i \bigwedge_j L_{i,j} \right) \leftrightarrow \left(\bigvee_i \exists x_n \; \bigwedge_j L_{i,j} \right) \right)$$

Hence, we can actually restrict our interest exclusively to formulas of the form $\exists x_n \; \bigwedge_j L_{i,j}$. We now use properties of the structure $\mathcal{Q}$ to manipulate this formulas. First of all, we replace all negated atomic formulas by atomic formulas using the fact that:

$$\mathcal{Q} \models \forall x \forall y \; (\neg(x = y) \leftrightarrow (x < y \vee y < x))$$

and:

$$\mathcal{Q} \models \forall x \forall y \; (\neg(x < y) \leftrightarrow (x = y \vee y < x))$$

Thus, we can assume that the open formula $\bigwedge_j L_{i,j}$ after the existential quantifier contains no negation sign, meaning that it is a conjunction of atomic formulas and disjunctions

of atomic formulas. By distributing the conjunctions into the disjunction, we can then assume that the open formula is a disjunction of conjunctions of atomic formulas, meaning that it has the form $\exists x_n \bigvee_k F_k$ where each F_k is an atomic formula. Again, we know that:

$$\mathcal{Q} \models \left(\left(\exists x_n \bigvee_k F_k \right) \leftrightarrow \left(\bigvee_k \exists x_n \, F_k \right) \right)$$

allowing us to restrict to the case of formulas of the form:

$$(\exists x_n \, F_1(x, x_1, \ldots, x_n)) \vee \ldots \vee (\exists x_n \, F_k(x, x_1, \ldots, x_n))$$

where each F_i is a conjunction of atomic formulas. This concludes all of our assumptions on the formulas.

Fix an F_i inside the formula and let $F_i \equiv \bigwedge_p H_p$, where each H_p is an atomic formula. The algorithm splits into two cases, where in turn the second case splits into 8 cases.

1. x_n doesn't appear inside F_i. Then, we can delete $\exists x_n$ from the outside formula, preserving equivalence since it isn't a free variable in F_i.

2. x_n appears inside F_i. Then, we collect all the atoms containing x_n by rewriting F_i as follows:

$$\mathcal{Q} \models \left(F_i \leftrightarrow \bigwedge_i (x_n < u_i) \wedge \bigwedge_j (v_j < x_n) \wedge \bigwedge_k (w_k = x_n) \wedge J \right)$$

   where J is a conjunction of atomic formulas that don't contain x_n. Since J doesn't contain x_n, the latter is not free inside it, meaning that we can remove it from the quantifier while preserving equivalence:

$$\mathcal{Q} \models \exists x_n \, \left(\bigwedge_i (x_n < u_i) \wedge \bigwedge_j (v_j < x_n) \wedge \bigwedge_k (w_j = x_n) \right) \wedge J$$

   Thus, we restrict our interest of formulas of the type $\exists x_n \, B$ where B is a conjunction of atomic formulas all containing x_n. We can remove all conjuncts of type $(x_n = x_n)$. If these are the only conjuncts in the formula we replace the entire formula $\exists x_n \, B$ with $\top$, the symbol representing a tautology. If a conjunct is of the form $(x_n < x_n)$ then we replace the entire formula with $\bot$, the symbol representing an unsatisfiable formula.

   We can from now on assume that there are no atomic formulas of type $(x_n = x_n)$ or $(x_n < x_n)$ in the formula. We're left with 7 cases based on the types of conjucts that remain in the formula:

   (a) B contains all three types of axiom, meaning that:

$$B \equiv \bigwedge_i (x_n < u_i) \wedge \bigwedge_j (v_j < x_n) \wedge \bigwedge_k (w_j = x_n)$$

Then, given the first equality atom $(w_0 = x_n)$, we consider the formula B' obtained by replacing x_n with w_0 in B:

$$B \equiv \bigwedge_i (w_0 < u_i) \wedge \bigwedge_j (v_j < w_0) \wedge \bigwedge_k (w_j = w_0)$$

It's easy to see that $\exists x_n\ B$ and B' share the same satisfying assignments.

(b) B has no less than axiom, meaning that:

$$B \equiv \bigwedge_i (x_n < u_i) \wedge \bigwedge_k (w_k = x_n)$$

We consider the formula B' defined as:

$$B' \equiv \bigwedge_i (w_0 < u_i) \wedge \bigwedge_k (w_k = w_0)$$

Again, $\exists x_n\ B$ and B' share the same satisfying assignments.

(c) B has no greater than axiom, meaning that:

$$B \equiv \bigwedge_j (v_j < x_n) \wedge \bigwedge_k (w_k = x_n)$$

We consider the formula B' defined as:

$$B' \equiv \bigwedge_j (v_j < w_0) \wedge \bigwedge_k (w_k = w_0)$$

Again, $\exists x_n\ B$ and B' share the same satisfying assignments.

(d) B has no equality axiom, meaning that:

$$B \equiv \bigwedge_i (x_n < u_i) \wedge \bigwedge_j (v_j < x_n)$$

We consider the formula B' defined as:

$$B' \equiv \bigwedge_{i,j} v_j < u_i$$

Clearly, any assignment satisfying $\exists x_n\ B$ also satisfies B'. Moreover, since $\mathcal{Q}$ satisfies the density property, any assignment satisfying B' also satisfies $\exists x_n\ B$.

(e) B has only less than axioms, meaning that:

$$B \equiv \bigwedge_i (x_n < u_i)$$

Since $\mathcal{Q}$ satisfies the non-minimality property, B is always true. Hence, it satisfies the same assignments as $B' \equiv \top$.

(f) B has only greater than axioms, meaning that:

$$B \equiv \bigwedge_j (v_j < x_n)$$

Since $\mathcal{Q}$ satisfies the non-maximality property, B is always true. Hence, it satisfies the same assignments as $B' \equiv \top$.

(g) B has only equality axioms, meaning that:

$$B \equiv \bigwedge_k (w_k = x_n)$$

We consider the formula B' defined as:

$$B \equiv \bigwedge_k (w_k = w_0)$$

It's easy to see that $\exists x_n\ B$ and B' share the same satisfying assignments.

In any of the seven sub-cases, we have that:

$$\mathcal{Q} \models ((\exists B) \leftrightarrow B')$$

allowing us to replace the entire formula $\exists x_n\ B$ with B'.

After inductively applying this replacement process, we're able to construct a quantifier-free formula that is $\mathcal{Q}$-equivalent to the original one. Moreover, this process can be computer through a procedure. □

Corollary 2.3: Decidabilily of DLO

Th(DLO) is decidable.

Proof. Let $\mathcal{Q} = (\mathbb{Q}, <)$. Through the previous theorem, for any formula F we can algorithmically find a quantifier-free formula F such that $\mathsf{DLO} \models F$ if and only if $\mathsf{DLO} \models F$. However, we observe that the only quantifier-free formulas without free variables in the language of orders are $\top$ and $\bot$. Therefore any sentence is $\mathcal{Q}$-equivalent either to $\top$ or to $\bot$. □

An analysis of the above algorithm shows that each step of elimination of an existential quantifier results in a quadratic blow-up, so that the overall procedure has time complexity $O(n^{2m})$ where n is the size of the formula G and m is the number of quantifiers in its prenex normal form.

The fact that a theory admits quantifier-elimination can be characterized in semantical terms. Below we show one such characterization in terms of an **Extension Property** that is reminiscent of the Back-and-Forth conditions. To make things easier, we consider a

simple language with finitely many predicate symbols and no constant symbols. Consider the possible atomic formulas in some fixed set of variables $x_1, \ldots, x_n$. Let $\mathfrak{A}$ be an $\mathcal{L}$-structure and let $\overrightarrow{a} = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix}$ and $\overrightarrow{b} = \begin{bmatrix} b1 & \cdots & b_n \end{bmatrix}$ in A^n such that the two vectors satisfy the same atomic formulas in $\mathfrak{A}$, written as $\overrightarrow{a} \equiv_{\text{atoms}} \overrightarrow{b}$. It is easy to see that the two vectors also satisfy the same quantifier-free formulas, written as $\overrightarrow{a} \equiv_{\text{open}} \overrightarrow{b}$.

Given $\overrightarrow{a} \in A^n$, let $H_{\overrightarrow{a}}(x_1, \ldots, x_n)$ the conjunction of all first-order literals, that being atomic formulas or their negations, satisfied by $\overrightarrow{a}$ in $\mathfrak{A}$. Then, for any vector $\overrightarrow{b} \in A^n$, we have that:

$$\overrightarrow{a} \equiv_{\text{atom}} \overrightarrow{b} \iff \mathfrak{A} \models H_{\overrightarrow{a}}(x_1, \ldots, x_n)[\overrightarrow{b}]$$

Definition 2.15: Extension property

We say that a theory T has the **Extension property** when for every $\mathfrak{A}$ such that $\mathfrak{A} \models T$ and for all $\overrightarrow{a}, \overrightarrow{b} \in A^n$ if $\overrightarrow{a} \equiv_{\text{atom}} \overrightarrow{b}$ then for any $a_{n+1} \in A$ there is a $b_{n+1} \in A$ such that $\begin{bmatrix} \overrightarrow{a} & a_{n+1} \end{bmatrix} \equiv_{\text{atoms}} \begin{bmatrix} \overrightarrow{b} & b_{n+1} \end{bmatrix}$.

The below theorem gives a characterization for the fact that a theory T in $\mathcal{L}$ admits quantifier elimination in the sense that each formula is T-equivalent to a quantifier-free formula over the same free variables. We observe that it's not guaranteed that the latter formula can be found algorithmically.

Theorem 2.11: Characterization of Quantifier Elimination

Let $\mathcal{L}$ be a finite language without constants. Then, the theory T admits Quantifier Elimination if and only if the Extension Property holds in T.

Proof. Suppose that T admits Quantifier Elimination. Fix a structure $\mathfrak{A}$ such that $\mathfrak{A} \models T$ and suppose that $\overrightarrow{a}, \overrightarrow{b} \in A^n$ are such that $\overrightarrow{a} \equiv_{\text{atoms}} \overrightarrow{b}$. Then, given any element $a_{n+1} \in A$ by construction of $H_{\overrightarrow{a},a_{n+1}}$ we have that:

$$\mathfrak{A} \models H_{\overrightarrow{a},a_{n+1}}(x_1, \ldots, x_n)[\overrightarrow{a}, a_{n+1}] \implies \mathfrak{A} \models \exists x\, H_{\overrightarrow{a}}(x_1, \ldots, x_n)[\overrightarrow{a}]$$

Since T admits Quantifier Elimination, we know that there is a quantifier-free formula $F(x_1, \ldots, x_n)$ that is T-equivalent to $H_{\overrightarrow{a},a_{n+1}}(x_1, \ldots, x_n)$, hence:

$$H_{\overrightarrow{a},a_{n+1}}(x_1, \ldots, x_n)[\overrightarrow{a}] \iff \mathfrak{A} \models F(x_1, \ldots, x_n)[\overrightarrow{a}]$$

Furthermore, since $\overrightarrow{a} \equiv_{\text{atoms}} \overrightarrow{b}$ we get that:

$$\mathfrak{A} \models F(x_1, \ldots, x_n)[\overrightarrow{a}] \iff \mathfrak{A} \models F(x_1, \ldots, x_n)[\overrightarrow{b}] \iff \mathfrak{A} \models \exists x\, H_{\overrightarrow{a},a_{n+1}}(x_1, \ldots, x_n)[\overrightarrow{b}]$$

Thus, there must be an element b_{n+1} such that:

$$A \models \exists x\, H_{\overrightarrow{a},a_{n+1}}(x_1, \ldots, x_n)[\overrightarrow{b}] \implies A \models H_{\overrightarrow{a},a_{n+1}}(x_1, \ldots, x_n)[\overrightarrow{b}, b_{n+1}]$$

By construction of $H_{\overrightarrow{a},a_{n+1}}$, this can happen if and only if $\left[\overrightarrow{a} \quad a_{n+1}\right] \equiv_{\text{atoms}} \left[\overrightarrow{b} \quad b_{n+1}\right]$, concluding that the Extension Property holds.

Vice versa, suppose that the Extension Property holds in T. As usual, we just need to show that a single existential can be eliminated. Fix a model $\mathfrak{A}$ such that $\mathfrak{A} \models T$. Consider the formula $\exists x_{n+1}\, F(x_1, \ldots, x_n)$ in T, where F is an open formula.

Claim: the satisfiability of $\exists x_{n+1} F$ over $\mathfrak{A}$ only depends on the atomic formulas satisfied by $\overrightarrow{a}$ (and thus also by $\overrightarrow{b}$)

Proof of the claim. Since $F \in \text{Th}(\mathfrak{A})$, there must be a $\overrightarrow{a} \in A^n$ such that $\mathfrak{A} \models \exists x_{n+1} F[\overrightarrow{a}]$. Then, there is a value $a_{n+1} \in A$ such that $\mathfrak{A} \models F[\overrightarrow{a}, a_{n+1}]$.

By the Extension Property, there is a $b_{n+1} \in A$ such that $\left[\overrightarrow{a} \quad a_{n+1}\right] \equiv_{\text{atoms}} \left[\overrightarrow{b} \quad b_{n+1}\right]$. Therefore, we have that:

$$\mathfrak{A} \models F[\overrightarrow{a}, a_{n+1}] \implies \mathfrak{A} \models F[\overrightarrow{b}, b_{n+1}] \implies \mathfrak{A} \models \exists \exists x_{n+1} F[\overrightarrow{b}]$$

Since we have shown that:

$$\mathfrak{A} \models \exists x_{n+1} F[\overrightarrow{a}] \implies \mathfrak{A} \models \exists x_{n+1} F[\overrightarrow{b}]$$

the satisfiability of $\exists x_{n+1} F$ over $\mathfrak{A}$ only depends on the atomic formulas satisfied by $\overrightarrow{a}$ (and thus also by $\overrightarrow{b}$). □

Let X be the set of all the formulas describing assignments that satisfy $\exists x_{n+1} F$:

$$X = \{H_{\overrightarrow{c}} \mid \overrightarrow{c} \in A^n, \mathfrak{A} \models \exists x_{n+1} F[\overrightarrow{c}]\}$$

Since the number of atomic formulas over $x_1, \ldots, x_n$ is finite, the set X is also finite. Hence the following quantifier-free formula G is also finite:

$$G(x_1, \ldots, x_n) \equiv \bigvee_{H_{\overrightarrow{c}} \in X} H_{\overrightarrow{c}}(x_1, \ldots, x_n)$$

Since for each $\overrightarrow{c} \in A^n$ the satisfiability of $\exists x_{n+1} F$ over $\mathfrak{A}$ only depends on the atomic formulas satisfied by $\overrightarrow{c}$, we conclude that:

$$\mathfrak{A} \models \forall x_1 \ldots \forall x_n (\exists x_{n+1} F \leftrightarrow G)$$

Therefore $T \models (\exists x_{n+1} F \leftrightarrow G)$. □

3

True Arithmetic

3.1 Non-standard models of arithmetic

We proved that there is an algorithm to decide membership in $\mathrm{Th}(\mathbb{Q}, <)$. In a similar fashion, it can be proven that the following theories are also decidable:

- *Real addition*, i.e. $\mathrm{Th}(\mathbb{R}, 0, 1, +, <)$.
- *Real arithmetic*, i.e. $\mathrm{Th}(\mathbb{R}, 0, 1, +, \times, <)$.
- *Petersburg arithmetic* (or *natural addition*), i.e. $\mathrm{Th}(\mathbb{N}, 0, 1, +, <)$.
- *Skolem arithmetic* (or *natural multiplication*), i.e. $\mathrm{Th}(\mathbb{N}, 0, 1, \times, <)$.

We observe that proving the decidability of the above theories is no trivial result. Moreover, these results are independent from each other. For instance, the decidability of real addition doesn't imply the decidability of natural addition. In fact, the latter is way harder to prove than the former since natural numbers satisfy less properties than the real numbers – think about the proof of Theorem 2.10. The next natural step – pun intended – is asking if **True Arithmetic (TA)** is decidable, that being the arithmetic over natural numbers. Surprisingly, the answer to this question is negative. Before proving this result, we discuss easier ones.

Definition 3.1: True Arithmetic

We define **True Arithmetic (TA)** as the tuple $\mathcal{N} = (\mathbb{N}, 0, 1, +, \times, <)$.

The Compactness Theorem gives us an easy tool to answer the following question: "is it possible to axiomatize the structure of the natural numbers up to isomorphism?" That is, can we write a first-order theory T such that all models of T are isomorphic copies of the natural numbers? The answer to this question is also negative.

Consider the language of arithmetic $\mathcal{L} = \{0, 1, +, \times, <\}$. For any natural $n \in \mathbb{N}$, we denote

with $\overline{n}$ the closed term of $\mathcal{L}$ in the form $(1 + (1 + (\ldots(1 + 1)\ldots)))$ with n occurrences of 1. We call these terms *numerals*. The structure $\mathcal{N} = (\mathbb{N}, 0, 1, +, \times)$ is adequate for $\mathcal{L}$, where $\overline{n}^{\mathcal{N}}$ corresponds to n. We call this structure the **standard natural model**.

Theorem 3.1: Non-uniqueness of TA in FOL

True Arithmetic doesn't admit a characterizing theory in first-order logic.

Proof. Suppose now that there is a theory T such that all its models are isomorphic to $\mathcal{N}$. In particular, we assume that T is the strongest possible theory, meaning that it contains all sentences that are true in the standard model, i.e. $\text{Th}(\mathcal{N}) \subseteq T$. Now, we'll artificially construct a stronger theory that contains T. Models that satisfy this theory will be referred to as **non-standard natural models**.

Let c be a new constant and consider the following theory:

$$T^* = T \cup \{\overline{n} < c \mid n \in \mathbb{N}\}$$

By Compactness, every non-standard model $\mathfrak{A}$ that satisfies T^* must also satisfy T, all the sentences that are true in $\mathcal{N}$ are also true in $\mathfrak{A}$. Moreover, the element $c^{\mathfrak{A}}$ that is mapped to the constant c is larger over $<^{\mathfrak{A}}$ than any other element in A – think of $c^{\mathfrak{A}}$ as a special value that acts like "an infinite value number". We refer to the interpretations $\overline{n}^{\mathfrak{A}}$ of the numerals $\overline{n}$ in $\mathfrak{A}$ as *standard numbers of* $\mathfrak{A}$. Since $\mathfrak{A}$ satisfies T, we know that:

- $<^{\mathfrak{A}}$ is a linear order
- There is an element $0^{\mathfrak{A}}$ that is minimal with respect to $<^{\mathfrak{A}}$.
- The successor of any standard number is a standard number
- There are no elements between a standard number and its successor

In particular, this implies that the partial function $n \mapsto \overline{n}^{\mathfrak{A}}$ is a partial isomorphism for standard numbers. However, $\mathfrak{A}$ also satisfies additional properties. In particular, if $a \in A$ is a non-standard number then its predecessor is also a non-standard number (otherwise a would be a standard number). Since, $c^{\mathfrak{A}}$ is clearly a non-standard number, there is a subset of non-standard numbers in $\mathfrak{A}$ that forms an infinitely decreasing chain $c^{\mathfrak{A}} >^{\mathfrak{A}} a_1 >^{\mathfrak{A}} a_2 >^{\mathfrak{A}} \ldots$, meaning that there is no minimal element for this subset. This concludes that $<^{\mathfrak{A}}$ is not a well-ordering of A. However, we know that $\mathcal{N}$ satisfies the Least Element Principle, raising a contradiction to the fact that $\mathfrak{A}$ satisfies $\text{Th}(\mathcal{N})$. □

Expanding on the above reasoning we can show that any model of the theory T^* used in the proof contains, besides the initial copy of $\mathbb{N}$, infinitely many copies of $\mathbb{Z}$ and that the relative ordering between these copies is a dense linear order. In other words, $\mathfrak{A}$ is isomorphic to a model of $\mathbb{Q}$ where each element is a copy of $\mathbb{Z}$.

3.2 Computable functions and arithmetic

To discuss some advanced topics about the underlying logical structures behind computability, we give a new definition of computable function, which was actually the first ever definition of computability given by Kurt Gödel, in the 1930s.

To make things easier to read, we'll assume that functions denoted by greek letters $\phi, \psi, \dots$ represent partial functions, while $f, g, \dots$ denote total functions. We refer to the following functions as **basic functions** of arithmetic:

- *Addition*, i.e. $+(x, y) = z$
- *Multiplication*, i.e. $\times(x, y) = z$
- *Projection*, i.e. $\pi_i^n(x_1, \dots, x_n) = x_i$
- *Characteristic function of numerical equality*, i.e. $\text{eq}(x, y) = 1$ if $x = y$ and $\text{eq}(x, y) = 0$ otherwise.

Definition 3.2: Computable functions

We define the class of **computable functions** $\mathcal{C}$ as the smallest class of (partial) functions $\mathbb{N}^k \to \mathbb{N}$ that contains the basic functions and that is closed under composition and minimalization.

In this context, the **composition operator** referres to the following generalized form of composition. Let $\theta_1, \dots, \theta_k : \mathbb{N}^n \to \mathbb{N}$ and let $\varphi : \mathbb{N}^k \to \mathbb{N}$. The composition of $\varphi, \theta_1, \dots, \theta_k$ is the function $\psi : \mathbb{N}^n \to \mathbb{N}$ defined as:

$$\psi(a_1, \dots, a_n) = \varphi(\theta_1(a_1, \dots, a_n), \dots, \theta_k(a_1, \dots, a_n))$$

The **minimalization operator**, instead, is defined by cases. If $\varphi(\overrightarrow{x}, y)$ is a function then the function $\psi(\overrightarrow{x})$ is defined as the minimum z such that the values $\varphi(\overrightarrow{x}, 0), \dots \varphi(\overrightarrow{x}, z-1)$ are defined and different from 0, while $\varphi(\overrightarrow{x}, z)$ is equal to 0. If no element z exists, the function is *undefined*.

$$\psi(\overrightarrow{x}) = z^* \text{ s.t. } z^* \in \arg\min_z (\varphi(\overrightarrow{x}, z) = 0)$$

The operator of minimalization corresponds to an unlimited and possibly not ending search. Therefore, the class of computable functions must contains partial functions. In fact, it can be proven that no list of total functions $(f_i)_{i \in \mathbb{N}}$ such that the operation $g : i \mapsto f_i(i)$ is algorithmic can contain the class of algorithmically computable functions: if this were the case, then the function $h : i \mapsto f_i(i)+1$ would be algorithmically computable, which is known to be false.

The class $\mathcal{C}$ is to be interpreted as nothing more than a mathematical definition of computability: there is no proof of computability being equivalent to this class, we're only imposing this equivalence. What can be proved, instead, is that $\mathcal{C}$ concides with the class

of functions computable by Turing machines [Tur37] and all other well-defined models of computation.

Proposition 3.1

The class $\mathcal{C}$ of computable functions corresponds to the class of functions computable by a Turing machine.

Definition 3.3: Structural definability of a function

Let $\mathfrak{A}$ be a structure and let $\varphi : A^k \to A$ be a function. We say that φ is **definable in** $\mathfrak{A}$ when there is a formula $F(x_1, \ldots, x_n, y)$ such that for all $a_1, \ldots, a_k, b \in A$ it holds that:

$$\varphi(a_1, \ldots, a_k) = b \iff \mathfrak{A} \models F[a_1, \ldots, a_k, b]$$

For our purposes, we'll focus in definability over the standard model of natural arithmetic $\mathcal{N} = (\mathbb{N}, 0, 1, +, \times, <)$. Note that in this model the following holds: for any formula $F(x_1, \ldots, x_n, y)$ and any choice of $n_1, \ldots, n_k, m \in \mathbb{N}$ it holds that

$$\mathbb{N} \models F(x_1, \ldots, x_k, y)[n_1, \ldots, n_k, m] \iff \mathbb{N} \models F(\overline{n_1}, \ldots, \overline{n_k}, \overline{m})$$

The following result, due to to Gödel, is the key to connect the realm of computability to the realm of formal first-order arithmetic.

Theorem 3.2: Representability theorem

All computable functions are definable by a formula in True Arithmetic.

Proof. We prove that each computable function is definable by a formula and that this class of functions contain the vasic functions, is closed under composition and closed under minimalization.

Claim 1: basic functions are definable in $\mathcal{N}$

Proof. Proof of Claim 1. Addition is easily defined by the formula $+(x, y) = z$, multiplication by the formula $\times(x, y) = z$, projection by the formula

$$x_1 = x_1 \wedge \ldots \wedge x_i = z \wedge \ldots \wedge x_n = x_n$$

and the characteristic function of numerical equality is definable by the formula

$$(x = y \wedge z = 1) \vee (x \neq y \wedge z = 0)$$

□

Claim 2: functions definable in $\mathcal{N}$ are closed under composition and minimalization

Proof. Proof of Claim 2. Consider any set of functions $\theta_1, \ldots, \theta_k : \mathbb{N}^n \to \mathbb{N}$ and any function $\varphi : \mathbb{N}^k \to \mathbb{N}$. For each $i \in [k]$, let $G_i(\overrightarrow{x}, y_i)$ be the formula that defines θ_i and let $H(y_1, \ldots, y_k, z)$ be the formula that defines φ. Then, the following formula:

$$\exists y_1 \ldots \exists y_k \; G_1(\overrightarrow{x}, y_1) \wedge \ldots \wedge G_k(\overrightarrow{x}, y_k) \wedge H(y_1, \ldots, y_k, z)$$

defines the composition $\psi : \mathbb{N}^n \to \mathbb{N}$ of $\theta_1, \ldots, \theta_k, \varphi$ since $\psi(\overrightarrow{a}) = p$ if and only if there are $q_1, \ldots, q_k$ such that $\theta_i(\overrightarrow{a}) = q_i$ and $\varphi(q_1, \ldots, q_k) = p$.

Let $\phi(\overrightarrow{x}) \arg\min_z \tau(\overrightarrow{x}, z) = 0$. Given the formula $F_\tau(\overrightarrow{x}, z, y)$ that defines τ, the following formula defines ϕ:

$$\forall w \; (w \leq z \to (\exists y \; F_\tau(\overrightarrow{x}, w, y) \wedge (y = 0 \leftrightarrow w = z)))$$

□

□

3.2.1 Undecidability of arithmetic

The concepts of decidability and computable enumerability are translated in a natural way in terms of computable functions. A set is **algorithmically decidable** if its characteristic function is a computable function. A set is **computably enumerable** if it is the domain of a computable function.

Proposition 3.2: Representability of comp. enum. sets

If $S \subseteq \mathbb{N}$ is computably enumerable then there is a formula $F(x)$ defined in $\mathcal{N} = (\mathbb{N}, 0, 1, +, \times, <)$ such that for every $n \in \mathbb{N}$ it holds that:

$$n \in S \iff \mathcal{N} \models F(x)[n]$$

The previous corollary already gives an idea behind why the Representability Theorem is useful to solve issues of decidability and undecidability. Let $(\varphi_n)_{n \in \mathbb{N}}$ be a model of computation, for example the model of unary functions. The *Diagonal Halting problem* is the set K defined as:

$$K = \{n \in \mathbb{N} \mid \varphi_i(i) \downarrow\}$$

Since it is a variant of the Halting problem, we know that K is computably enumerable but not decidable. By the above corollary, there must be a formula $F(x)$ that defines K in $\mathcal{N}$. However, this implies that True Arithmetic is *undecidable*: if it were to be decidable, we could decide membership in K, which we know to be false.

Corollary 3.1: True Arithmetic is undecidable

The theory of True Arithmetic is undecidable.

The above important result is strictly connected to Gödel's famous *Incompleteness Theorems*, which were actually proven without using the the existence of an algorithmically undecidable problems such as K. Consider now the complementary problem of K, the Diagonal Looping problem:

$$\overline{K} = \{n \in \mathbb{N} \mid \varphi_i(i) \uparrow\}$$

Again, since it is a variant of the Looping problem, we know that $\overline{K}$ is not computably enumerable. Hence, we conclude that True Arithmetic is also *not computably enumerable*: if it were to be computably enumerable, we could computably enumerate $\overline{K}$ by printing each value n for which $\neg F(n) \in \mathrm{Th}(\mathcal{N})$.

Corollary 3.2: True Arithmetic not computably enumerable

The theory of True Arithmetic is not computably enumerable.

We also observe that the Representability Theorem gives a necessary condition for a function to be computable. Thus, to show that a function is not computable it is sufficient to prove that it is not representable in $\mathcal{N}$ by a formula. If we are interested in the computability/decidability of sets rather than functions it is enough to call a set $S \subseteq \mathbb{N}$ computable if its characteristic function χ_S is in $\mathcal{C}$. Then it is easy to see that from the Representability Theorem we have the following corollary.

Proposition 3.3: Representability of computable sets

If $S \subseteq \mathbb{N}$ is decidable (or computable) then there is a formula $F(x)$ defined in $\mathcal{N} = (\mathbb{N}, 0, 1, +, \times, <)$ such that for every $n \in \mathbb{N}$ it holds that:

- $n \in S$ then $\mathcal{N} \models F(n)$
- $n \notin S$ then $\mathcal{N} \models \neg F(n)$

We observe that the above proposition is equivalent to the previous one only if $\mathcal{N}$ is complete. However, as we'll show in the following sections, True Arithmetic is incomplete!

3.2.2 Undefinability of arithmetic

The representability results shown in this section hold for True Arithmetic, but what about sets of objects other than natural numbers? To consider this question, we need to resort to encoding the objects into elements $\mathbb{N}$ itself. This process is, technically, already implicitly assumed when we speak about computability of a set of sentences since the class $\mathcal{C}$ only deals with functions on natural numbers. In other words, we need to replace $\mathrm{Th}(\mathcal{N}) = \{S \mid \mathbb{N} \models S\}$ with the set $\{\mathrm{code}(S) \mid \mathbb{N} \models S\}$, where *code* is a computable coding function that assigns a numerical code to each sentence.

This can be done in many (efficiently) computable ways through machines that can compute the code of a given sentence and, vice versa, can decode a number and write down

the sentence it codes (if any). Furthermore, syntactic properties of formulas and sentences (e.g. checking whether a formula has one free variable, whether a formula is a sentence, ...) can be easily decidabed through a machine. Similarly, syntactic manipulations can be performed by algorithms. All these properties and operations are computable also when performed on the numerical codes of formulas. For instance, given a number $n \in \mathbb{N}$, a machine can decode the formula F it encodes (if any), replace free occurrences of a variable x in F by a term t (if any) and compute the numerical code of the resulting formula.

Through this encoding, we're able to prove **Tarski's Undefinability theorem**[Tar33]. Informally, the theorem states that "arithmetical truth cannot be defined through arithmetic". This is one the most important limitative results in mathematical logic, the foundations of mathematics and in formal semantics. From the logical point of view, the theorem says that no formula in the language of arithmetic, however complex, can discriminate between sentences that are true and sentences that are false in the natural numbers. From the point of view of computability, insteadn, the theorem implies that the truth in $\mathbb{N}$ has a degree of unsolvability that is even higher than non-computability and non-computable enumerability.

Theorem 3.3: Undefinability theorem

The theory of True Arithmetic is not expressible by a formula of first-order arithmetic over $\mathcal{N}$, meaning that there is no formula $T(x)$ with one free variable such that:

$$\{n \in \mathbb{N} \mid n \text{ encodes a } \mathsf{TA} \text{ sentence}\} = \{n \in \mathbb{N} \mid \mathcal{N} \models T(n)\}$$

Proof. Let D be a function such that if u is the code of a formula $A(x)$ with free variable x then $D(u)$ is the code of $A(u)$. It is easy to see that D is a computable function. Then, by Proposition 3.3, we know that D is representable in $\mathbb{N}$ by a formula $D(x, y)$ such that:

$$D(k) = j \iff \mathcal{N} \models D(\overline{k}, \overline{j})$$

Then, for every $k \in \mathbb{N}$, we have that:

$$\mathcal{N} \models \forall y \forall z \left((D(k, y) \wedge D(k, z)) \rightarrow y = z\right)$$

By way of contradiction, suppose that $\mathrm{Th}(\mathcal{N})$ is expressible in $\mathcal{N}$ through a formula $T(y)$. Then, for each sentence S it holds that:

$$\mathcal{N} \models S \implies \mathcal{N} \models T(\overline{\mathrm{code}(F)})$$

and that:

$$\mathcal{N} \not\models S \implies \mathcal{N} \models \neg T(\overline{\mathrm{code}(F)})$$

To make notation lighter, we'll shorten $T(\overline{\mathrm{code}}(S))$ with $T(S)$ and we'll omit the bar over numerals. Consider the following formula $A(x)$:

$$A(x) \equiv \forall y (D(x, y) \rightarrow \neg T(y))$$

and let $\text{code}(A(x)) = p$. Consider now the formula $A(p)$:

$$A(p) \equiv \forall y(D(p, y) \rightarrow \neg T(y))$$

and let $\text{code}(A(p)) = q$. By definition of D, we have that $D(p) = q$.

Claim: $\mathcal{N} \models \neg T(q)$.

Proof of the claim. We proceed by cases:

1. $\mathcal{N} \not\models A(p)$. Then, $A(p)$ is not a true sentence in $\mathcal{N}$, implying that $q \notin \text{Th}(\mathcal{N})$. Therefore, we have that $\mathcal{N} \models \neg T(q)$.
2. $\mathcal{N} \models A(p)$. Then, we also have that $\mathcal{N} \models D(p, q) \rightarrow \neg T(q)$ implying that $\mathcal{N} \models \neg T(q)$.

□

Since $D(p) = q$, we know that $\mathcal{N} \models D(p, q)$. Moreover, since D is a function, we know that $\mathcal{N} \models D(p, y) \rightarrow y = q$. Through the claim, we also know that $\mathcal{N} \models y = q \rightarrow \neg T(y)$. Therefore, it follows that:

$$\mathcal{N} \models D(p, y) \rightarrow \neg T(y)$$

Since y is a free variable, we also get that:

$$\mathcal{N} \models \forall y(D(p, q) \rightarrow \neg T(y))$$

which equals to $\mathcal{N} \models A(p)$. Then, since $T(y)$ represents arithmetical truth, we get that:

$$\mathcal{N} \models T(\text{code}(A(p)))$$

which equals to $\mathcal{N} \models T(q)$, contradicting the claim. □

3.3 Algorithmically inseparable sets

After defining the concept of computable functions through functions over natural numbers, we dive more deeply into relation between such formulation and problems. In particular, we'll focus on tuples of the form $(A, \overline{A})$ and their **inseparability**, i.e. the idea of these two sets not being completely algorithmically discriminable from each other.

Definition 3.4: Algorithmical inseparability

Let $A, B \subseteq \mathbb{N}$. We say that the pair (A, B) is **algorithmically inseparable** if there is no computable total function $f : \mathbb{N} \rightarrow \{0, 1\}$ such that:

- For every $a \in A$ it holds that $f(a) = 1$
- For every $b \in B$ it holds that $f(b) = 0$

Equivalently, there is no decidable set C such that $A \subseteq C$ and $B \subseteq N - C$.

We observe that if $A \cup B \neq \varnothing$ then the pair (A, B) is trivially inseparable. Hence, we're interested in pairs where $A \cap B = \varnothing$. From the very definition of inseparability, it's easy to see that if a set and its complement are inseparable then they are obth undecidable. The converse also holds: if a set is undecidable then there is at least one element for which no computable function is able to decide if it belongs in the set or in its complement.

Proposition 3.4

Let $A \subseteq \mathbb{N}$. The pair $(A, \overline{A})$ is inseparable if and only if A is undecidable.

A more interesting property of inseparability is the transferability of inseparability through reductions. define a good concept of reduction between problems: if (A, B) is inseparable, $A \subseteq C$ and $B \subseteq D$ with $C \cap D = \varnothing$ then (C, D) must also be inseparable, since otherwise (A, B) would be separable.

Proposition 3.5

Let $A, B, C, D \subseteq \mathbb{N}$ be such that $A \subseteq C$ and $B \subseteq D$. If (A, B) is inseparable and $C \cap D = \varnothing$ then (C, D) is inseparable.

Suppose now that we have fixed a reasonable model of computation for the computable functions $(P_i)_{i \in \mathbb{N}}$, meaning that each computable function has at least one program that computes it. We denote with $P_i(n)$ the output (possibly undefined) of the procedure P_i on input n. In all reasonable models of computation, the **universal function** $u : (i, x) \mapsto P_i(x)$ is computable, as proved by Turing [Tur37]. We can derive the following proposition.

Proposition 3.6

The pair (S_0, S_1), where$S_0 = \{i \in \mathbb{N} \mid P_i(i) = 0\}$ and $S_1 = \{i \in \mathbb{N} \mid P_i(i) = 1\}$, is algorithmically inseparable.

Proof. By way of contradiction, suppose that there is a computable total function $f : \mathbb{N} \to \{0, 1\}$ such that:

- For every $a \in S_0$ it holds that $f(a) = 1$
- For every $b \in S_1$ it holds that $f(b) = 0$

Then, there must be an index $c \in \mathbb{N}$ such that $P_c = f$. However, since $c \in \mathbb{N}$ we have two possibilities:

1. If $c \in S_0$ then $1 = f(c) = P_c(c)$, thus $c \in S_1$
2. If $c \in S_1$ then $0 = f(c) = P_c(c)$, thus $c \in S_0$

Since S_0 and S_1 are disjoint, in both cases we get a contradiction. □

We now show how the computable inseparability of (S_0, S_1) can be strenghtened to what is called **effective computable inseparability**. The assertion that S_0 and S_1 are inseparable implies the following weaker claim: if A and B are two computably enumerable sets such that A and B are disjoint, $S_0 \subseteq A$ and $S_1 \subseteq B$, then there is at least an element in $\mathbb{N} - (A \cup B)$. Otherwise, we would have $B = \overline{A}$ and therefore A would be a *computable separator* for S_0 and S_1.

We observe that each statement of the form "For all ... there exists ..." always implicitly defines a function, taking from the universally quantified parameters (the input) to one of the existing witnesses of the existential (the output). In this cases we can always ask whether the statement holds *effectively*: "is there an algorithm that takes from the universally quantified parameters to the witness of the existential?". In the case at hand the statement is "for all sets A and B such that .... there exists a number m neither in A nor in B". Note that the sets in questions are computably enumerable since they are the domain of an algorithm: there exists an $a \in \mathbb{N}$ such that program $\mathrm{dom}(P_a) = A$ and $\mathrm{dom}(P_b) = B$. Thus, we can quantify over computably enumerable As and Bs by quantifying over their numerical codes. The statement becomes the following: "for all $a, b \in \mathbb{N}$ such that $\mathrm{dom}(P_a) \cap \mathrm{dom}(P_b) = \varnothing$ with $S_0 \subseteq \mathrm{dom}(P_a)$ and $S_1 \subseteq \mathrm{dom}(P_b)$ there is an $m \in \mathbb{N} - (\mathrm{dom}(P_a) \cup \mathrm{dom}(P_b))$", which implicitely defines a function of the type $\alpha : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

Theorem 3.4: Effective computable inseparability

There is a computable function $\alpha : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that for all $a, b \in \mathbb{N}$ if all of the followings hold:

1. $\mathrm{dom}(P_a) \cap \mathrm{dom}(P_b) = \varnothing$
2. $S_0 \subseteq \mathrm{dom}(P_a)$
3. $S_1 \subseteq \mathrm{dom}(P_b)$

then $\alpha(a, b) \in \mathbb{N} - (\mathrm{dom}(P_a) \cup \mathrm{dom}(P_b))$.

Proof. Fix $a, b \in \mathbb{N}$. We define $\psi_{a,b}(y)$ as the function computed by the following procedure: while simultaneously enumerating $\mathrm{dom}(P_a)$ and $\mathrm{dom}(P_b)$, if y appears in $\mathrm{dom}(P_a)$ then output 1, otherwise if it appears in $\mathrm{dom}(P_b)$ then output 0.

We observe that the dependence of $\psi_{a,b}$ in the values a, b is *uniform and algorithmic*, meaning that a program for $\psi_{a,b}$ can be algorithmically obtained from programs that compute P_a and P_b. This implies that there is a total computable function f such that $P_{f(a,b)}$ computes ϕ.

Claim: $f(a, b) \in \mathbb{N} - (\mathrm{dom}(P_a) \cup \mathrm{dom}(P_b))$

Proof of the claim. By way of contradiction, suppose that $f(a, b) \in \mathrm{dom}(P_a) \cup \mathrm{dom}(P_b)$. Then, we have that:

1. $f(a, b) \in \mathrm{dom}(P_a)$. Then, $1 = \psi(f(a, b)) = P_{f(a,b)}(f(a, b))$, which implies that $f(a, b) \in S_1 \subseteq \mathrm{dom}(P_b)$

2. $f(a,b) \in \text{dom}(P_b)$. Then, $0 = \psi(f(a,b)) = P_{f(a,b)}(f(a,b))$, which implies that $f(a,b) \in S_0 \subseteq \text{dom}(P_1)$

In both cases, we contradict the fact that $\text{dom}(P_a) \cap \text{dom}(P_b) \neq \varnothing$. □

□

3.3.1 Incompleteness of arithmetic

The undecidability of the theory of True Arithmetic has some more refined consequences related to the possibility of mechanizing number theory. For every sentence S, in True Arithmetic it holds that either $\mathcal{N} \models S$ or $\mathcal{N} \models \neg S$. Thus, since there is no algorithm that decides if a given sentence S is true in $\mathcal{N}$ or not, no algorithm can decide which of $\mathcal{N} \models S$ and $\mathcal{N} \models \neg S$ holds. This idea can be formally restated as follows.

Proposition 3.7

Let $T_0 \subseteq \{S \mid \mathcal{N} \models S\}$ and $T_1 \subseteq \{S \mid \mathcal{N} \models \neg S\}$. If both T_0 and T_1 are computably enumerable, there is a sentence G such that G holds in True Arithmetic but $G \notin T_0 \cup T_1$.

Proof. Suppose that both T_0 and T_1 are computably enumerable. First, we observe that at least one sentence in the language of arithmetic must lie outside of both of them.

Claim: $T_0 \cup T_1$ is not the set of all sentences.

Proof. By way of contradiction, suppose that all the possible sentences of the language of arithmetic are in $T_0 \cup T_1$. Then, since T_0 and T_1 are computably enumerable and for every sentence either $\mathcal{N} \models S$ or $\mathcal{N} \models \neg S$, we get that $T_0 = \text{Th}(\mathcal{N})$ and $T_1 = \overline{\text{Th}(\mathcal{N})}$. By Proposition 1.4, we get that $\text{Th}(\mathcal{N})$ is decidable, contradicting Corollary 3.1. □

Through the above claim, we know that there is at least one sentence G that is neither in T_0 nor T_1. This sentence can be either true or false in True Arithmetic. Hence, either G or $\neg G$ is true in TA but not in $T_0 \cup T_1$. □

The above proposition is close to Gödel original formulation of his **First Incompleteness Theorem**. The latter was expressed in terms of provability in formal axiomatic systems of arithmetic and not in terms of satisfaction in the structure $\mathcal{N}$. We give a notion of **proof system** as originally defined by Gödel. Suppose that we have an initial set of *axioms*, such as first-order sentences, and fix some *rules of deduction* for moving from some premises to some conclusions, such as concluding B derives from A and $A \to B$). Then, a **proof** is just a finite sequence of formulas starting from axioms and proceeding by application of the rules of deduction. The final formula is a **theorem** (or *conclusion*).

If the set of axioms and the correct application of the rules of deduction are such that they can be identified (e.g. computed by an algorithm) then we can also algorithmically decide whether a string is a proof or not. Moreover, the set of formulas that admit a proof from the given axioms (i.e. the set of theorems of the theory) would be computably

enumerable by generating, in some ordered way, all the possible proofs. Therefore, each computable set of axioms for arithmetic (e.g. True Arithmetic) determines a *computably enumerable set of theorems.*

These sets act as the counterparts of the sets T_0 and T_1 of our formulation: if we fix a set A of axioms for arithmetic, the sets $A_0 = \{S \mid S \text{ admits a proof from } A\}$ and $A_1 = \{S \mid \neg S \text{ admits a proof from } A\}$ are both computably enumerable. We would like these axioms to be **consistent**, meaning that they are non-contradictory (which implies that A_0 and A_1 are disjoint), and we would also clearly want them to be **correct** (true in $\mathcal{N}$). To conclude, we would like to design a set of axioms A such that for all sentences S either S or $\neg S$ is provable from A, i.e. either $S \in A_0$ or $S \in A_1$. We refer to such property as **completeness**. The above propositon, however, shows that this is impossible since our axioms system produces two sets like T_0 and T_1 that are computably enumerable, concluding that the formula $G \notin T_0 \cup T_1$ is true but cannot be proved, raising a contradiction. Hence, no proof system based on True Arithmetic can be both consistent and complete.

A good eye may also have notices that the proof of the above proposition is very similar to the concept of algorithmic inseparability. In fact, the latter can be used to give an *effective version* of Gödel's First Incompleteness Theorem.

Proposition 3.8

There is a formula $U(i, x, y)$ such that, for all set of sentences T_0, T_1, if the following conditions hold:

1. $T_0 \subseteq \{S \mid \mathcal{N} \models S\}$ and T_0 is computably enumerable
2. $T_1 \subseteq \{S \mid \mathcal{N} \models \neg S\}$ and T_0 is computably enumerable
3. If $i \in S_0$ then $U(i, i, 0) \in T_0$
4. If $i \in S_1$ then $U(i, i, 0) \in T_1$

then there is a number $e \in \mathbb{N}$ such that $U(e, e, 0) \notin (T_0 \cup T_1)$.

Proof. Since the universal function $u : (i, x) \mapsto P_i(x)$ is computable, by Theorem 3.2 we know that there is a formula $U(i, x, y)$ such that:

$$u(i, x) = P_i(x) \iff \mathcal{N} \models U(i, x, P_i(x))$$

Hence, we have that:

$$i \in S_0 \implies \mathcal{N} \models U(i, i, 0)$$

and that:

$$i \in S_1 \implies \mathcal{N} \models U(i, i, 1)$$

Since $\mathcal{N} \models \neg(0 = 1)$ and $U(i, x, y)$ is functional, we have that:

$$i \in S_1 \implies \mathcal{N} \models \neg U(i, i, 0)$$

Since T_0, T_1 are computably enumerable, the sets $I_0 = \{i \in \mathbb{N} \mid U(i,i,0) \in T_0\}$ and $I_1 = \{i \in \mathbb{N} \mid U(i,i,0) \in T_1\}$ also are.

Suppose now that T_0 and T_1 are strong enough to respectively contain all sentences of the form $U(i,i,0)$ with $i \in S_0$ and all those with $i \in S_1$. Then, by the above observations, $S_0 \subseteq I_0$ and $S_1 \subseteq I_1$. Then, by Theorem 3.4, we can compute a number $e \in \mathbb{N}$ such that $U(e,e,0) \notin (T_0 \cup T_1)$ □

The condition relating S_0 to T_0 and S_1 to T_1 imply that T_0 and T_1 do not miss some specific truths of arithmetic, meaning that they are strong enough to recognize at least some very specific arithmetical truths. In Gödel's original formulation, he only required that the axioms contain a *minimal amount of arithmetic axioms*. We refer to this set of axioms as **Minimal Arithmetic (MA)**.

Definition 3.5: Minimal Arithmetic (MA)

We define the **Minimal Arithmetic** as the finite theory MA containing:

1. $0 + 1 = 1$
2. $\forall x \, \neg(x + 1 = 0)$
3. $\forall x \, (\neg(x = 0) \to \exists z \, (z + 1 = x))$
4. $\forall x \forall y \, (x + 1 = y + 1 \to x = y)$
5. $\forall x \, (x + 0 = x)$
6. $\forall x \forall y \, (x + (y + 1) = (x + y) + 1)$
7. $\forall x \, (x \times 0 = 0)$
8. $\forall x \forall y \, (x \times (y + 1) = (x \times y) + x)$
9. $\forall x \forall y \, (x < y \vee x = y \vee y < x)$

This minimal set is such that it is enough to carry over a formal version of the proof that all computable functions are representable in true arithmetic: all computable functions are representable (in a naturally defined meaning) in any theory of arithmetic that contains MA. This allows us to carry all the arguments we did above in the new setting of proofs rather than satisfaction in a model.

The main ingredient for obtaining Gödel's result for any computable theory extending MA is the fact that the Representation Theorem holds relative to MA in the following sense: for every computable function $\varphi : \mathbb{N}^k \to \mathbb{N}$, there is an arithmetic formula $F(\overrightarrow{x}, y)$ that represents φ in all models of MA, meaning that if $\varphi(\overrightarrow{a}) = b$ then $\mathsf{MA} \models F(\overrightarrow{a}, b)$ and F is functional, i.e.:

$$\mathsf{MA} \models \forall x \forall y \forall z \, (F(x,y) \wedge F(x,z) \to y = z)$$

In particular, this also holds true for the universal function, implying that there is a formula $U(i,x,y)$ that represents it in MA. Let T be any computable extension of MA.

It's easy to see that the map $\alpha : i \mapsto U(i, i, 0) \wedge \mathsf{MA}$ is a computable reduction with the following properties:

1. If $i \in S_0$ then
$$T \models U(i, i, 0) \wedge \mathsf{MA}$$
2. If $i \in S_1$ then the formula $U(i, i, 0) \wedge \mathsf{MA}$ is unsatisfiable

Since (S_0, S_1) is inseparable, it follows that the pair $(\{S \mid T \models S\}, \text{UNSAT})$ is also inseparable. Moreover, if T has a model then no unsatisfiable formula can be a consequence of T. Therefore, we also get that $(\{S \mid T \models S\}, \{S \mid T \models \neg S\})$ is inseparable, concluding that $\{S \mid T \models S\}$ is undecidable. This establishes the fact that every computable satisfiable theory T that contains MA has an undecidable set of logical consequences. Finally, this also implies that the theory is incomplete, since the set of consequences of a computable theory is computably enumerable.

Theorem 3.5: First Incompleteness Theorem

Let T be a computable set of axioms such that $\mathsf{MA} \subseteq T$. If T is consistent, then T is incomplete.

Bibliography

[Ehr60] A. Ehrenfeucht. “An application of games to the completeness problem for formalized theories”. In: *Fundamenta mathematicae* (1960). DOI: 10.2307/2271711.

[ER59] P. Erdős and A. Rényi. “"On Random Graphs. I”. In: *Publicationes Mathematicae* (1959). DOI: 10.5486/PMD.1959.6.3-4.12.

[Fra54] Roland Fraïssé. “Sur quelques classifications des systèmes de relations”. 1954.

[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/.

[Tar33] Alfred Tarski. “Pojęcie Prawdy w Językach Nauk Dedukcyjnych”. In: *Nakładem Towarzystwa Naukowego Warszawskiego* (1933).

[Tar98] Alfred Tarski. “A Decision Method for Elementary Algebra and Geometry”. In: Springer Vienna, 1998. DOI: 10.1007/978-3-7091-9459-1_3.

[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.