Lecture Notes on Propositional and Predicate Logic

Martin Pilát

Based on lectures by Petr Gregor

MARCH 9, 2020

Introduction

Generally, logic is the study of arguments and inferences. While it started as a philosophical discipline in ancient times, it is now widely studied in mathematics and computer science. As such, logic provides the basic language and tools for most of mathematics. It studies different proof systems and discusses whether they are sound (everything they prove is valid) and complete (everything that is valid can be proven).

While logic provides rather low-level tools for mathematics and computer science, it still has rather wide applications. For example, in some areas of artificial intelligence, logic is used to represent the knowledge of intelligent agents and reason about it. The agents then use logical reasoning (theorem proving) to decide what to do next, or prove that a certain action is safe in a given environment. Another important area is formal software verification, where logic (and, again, theorem proving) can be used to formally verify that a program indeed does what it should according to a specification. This is essential while implementing e.g. cryptographic protocols. Formal verification is also used while designing digital circuits.

There are also attempts to formalize all of mathematics in logic and use computers to check that all the proofs are correct. For example, Mizar[1] is a system that aims to re-create most of mathematics with formal and verified proofs. The verification starts from the basic mathematical axioms – in the case of the Mizar system, authors of so called Mizar articles can use only axioms of set-theory and theorems from previously verified articles. Therefore, everything published in the Mizar mathematical library is verified to be a correct consequence of the base axioms.

[1] http://www.mizar.org/library/

Logic serves as the formal language of mathematics, and therefore logic also needs to formally specify the syntax of the language. The syntax defines what is a valid logical formula and what is not, however the meaning and validity of a formula is given by the semantics of the language. Logic itself prescribes the meaning of only a handful of symbols – namely the logical connectives ($\wedge, \vee, \rightarrow, \leftrightarrow, \neg$) and the quantifiers ($\forall, \exists$). Additionally, in languages with equality, the meaning of "$=$" is also given. All other symbols used in logical formulas can have arbitrary meanings, which are given by the semantics. So, for example, if we write a formula $(\forall x)(\forall y)(x + y = y + x)$, we cannot discuss its validity before defining the meaning of "$+$". The formula is valid if we are talking about the real numbers and "$+$" denotes

their addition. However, the symbol "$+$" can also represent (quite unusually) the multiplication of square matrices, and in such a case, the formula is not valid.

There are different levels of the language of logic. In propositional logic, only propositional variables (those that are either true or false) and the logical connectives can be used. In first-order logic, we can additionally use functions, relations and quantifiers for variables that range over objects from some universe. In second-order logic, there are additionally quantifiers for sets of objects in the universe (and, more specifically, for functions and relations). In higher-order logic, we also have variables for sets of sets of objects. For example, a formula in propositional logic

$$(d \wedge c) \rightarrow s$$

can express that if it is dark and clear outside, the stars are visible. In a first-order language, we can have a more complex formula

$$(\forall x)(\forall y)(S(x) \wedge E(y) \rightarrow (L(x,y) \rightarrow P(x,y)))$$

that expresses that if x is a student ($S(x)$) and y is an exam ($E(y)$), if x learns the material for y ($L(x,y)$) then x passes y ($P(x,y)$). As an example of a second-order language, we can write the axiom of induction:

$$(\forall P)(P(0) \wedge (\forall n)(P(n) \rightarrow P(n+1)) \rightarrow (\forall n)P(n))\,.$$

In the lecture, we will deal mostly with propositional and first-order logic, however there are also other extensions of logic. For example, in multi-agent systems, so-called modal logic is often used to represent knowledge. In modal logic, there are special modalities that can further qualify a statement. For example, there is a modality that says that a statement may be true, or must be true. Other types of modal logic contain modalities that express knowledge of other agents (e.g. "agent A knows that statement S is true" can be written as $\mathrm{K}_A.S$, or even "agent A knows that agent B knows that statement S is true" ($\mathrm{K}_A.\mathrm{K}_B.S$)). Another interesting type of modal logic is temporal logic, which contain modalities about time and can express e.g. "statement S will be true at some time in the future".

About these lecture notes

In these lecture notes, logic is presented for students of computer science. Therefore, focus is given to areas most needed for computer scientists. For example, we use the more intuitive tableau method instead of Hilbert-style proof systems. We also explain the resolution method in logic as a background to Prolog and logic programming. The more advanced topics on decidability and incompleteness are explained in a more informal way.

You are currently reading the first version of the lecture notes, which can, and most probably will, contain some errors. If you find an error, or if something is not clear, do not hesitate to contact the author by

e-mail[2], or, alternatively, create an issue in the GitHub repository of the book[3].

There are also other resources you may want to check. One of them are the presentations created by Petr Gregor for his version of the lecture[4], which serve as a base for these lecture notes.

Informal! The author of these notes sometimes likes to explain things in a more intuitive way with some not-so-formal examples and metaphors. While he believes these can help to give a better understanding of the given concept, they sometimes (read "often") are rather informal and have some limitations. Therefore, in these notes, they will be set in boxes like the one you are reading just now with a bold "**Informal!**" warning. The information contained in these boxes is always non-essential to the rest of the text and can be (some would even argue should be) skipped.

[2] Martin.Pilat@mff.cuni.cz

[3] https://github.com/martinpilat/logic-book

[4] http://ktiml.mff.cuni.cz/~gregor/logics/index.html

Preliminaries

Like many mathematical texts, these lecture notes also assume that the reader has some basic knowledge. The most important concepts (many of which should sound familiar) are briefly introduced in this short section, both to provide a single place where these can be found and to introduce the notation used in these lecture notes.

We will start with the basic set-theoretic notions. The most basic of these is the *class*. Each property of sets $\varphi(x)$ defines a class $\{x|\varphi(x)\}$. Some classes are also sets; those that are not are called *proper classes*. The distinction between sets and classes is probably new for most readers. Why would we need any other collections of objects than sets? How is it possible that there is a collection of objects which is not a set? The reason to distinguish between these two is that if everything were considered a set, we could find paradoxes in set theory. For example, if we had a set of all sets that do not contain themselves, does this set contain itself, or not? Let us assume, it does, but then, by definition, it does not. If we instead assume it does not contain itself, then, again, by definition, it does. This so-called Russell's paradox can be avoided by using a notion of classes that cannot contain other classes.

Informal! A class can be understood as any collection of sets that can be described by the language of set theory. However, some of these collections do not make much sense and can lead to paradoxes. Therefore, any collection that would lead to a paradox is denoted as a proper class instead of a set and the paradoxes can thus be avoided.

The other set-theoretic notions should be much more familiar. We use $x \in y$ to denote that x is a member of set y. $x \notin y$ and $x \neq y$ are shortcuts for $\neg(x \in y)$ and $\neg(x = y)$. A set containing exactly elements $x_0, x_1, \ldots, x_n$ is denoted as $\{x_0, x_1, \ldots, x_n\}$. A set with only one element $\{x\}$ is called a *singleton* and a set with two elements $\{x_0, x_1\}$ is called an *unordered pair*. We will also use the common notation for set operations: $\emptyset$ denotes an *empty set*, $\cup$ and $\cap$ denote the *union* and *intersection* of sets. $\setminus$ is the *set difference* operator and $\triangle$

is the *symetric set difference* operator

$$x \triangle y = (x \setminus y) \cup (y \setminus x).$$

Two sets are *disjoint* if their intersection is an empty set, and $x \subseteq y$ denotes that x is a subset of y (all elements of x are also elements of y). The set of all subsets of a set x – the *power set of* x – is denoted as $\mathcal{P}(x)$. The *union of set* x, $\bigcup x$, is the union of all sets contained in x. A *cover of a set* x is a set $y \subseteq \mathcal{P}(x) \setminus \emptyset$ such that $\bigcup y = x$. If all the sets in the cover y are mutually disjoint, then y is a *partition of* x.

The definition of an unordered pair can be used to define the *ordered pair* $(a,b) = \{a,\{a,b\}\}$ and an *ordered n-tuple* $(x_0,\ldots,x_{n-1}) = ((x_0,\ldots,x_{n-2}),x_{n-1})$ for $n > 2$. A Cartesian product of two sets a and b is $a \times b = \{(x,y)|x \in a, y \in b\}$ and the Cartesian power of a set x is defined as $x^0 = \{\emptyset\}$, $x^n = x^{n-1} \times x$. A *binary relation* R is a set of ordered pairs. The *domain of* R is defined as $\mathrm{dom}(R) = \{x|(\exists y)(x,y) \in R\}$; the *range of* R is similarly $\mathrm{rng}(R) = \{y|(\exists x)(x,y) \in R\}$. The *extension of* x *in* R is the set $R[x] = \{y|(x,y) \in R\}$. The symbol R^{-1} denotes the *inverse relation* $R^{-1} = \{(y,x)|(x,y) \in R\}$. The *restriction of* R *to a set* z is defined as $R \restriction z = \{(x,y) \in R|x \in z\}$. Two relations can also be *composed* into one, $R \circ S = \{(x,z)|(\exists y)((x,y) \in R \wedge (y,z) \in S\}$. The *identity relation* on set z, $\mathrm{Id}_z = \{(x,x)|x \in z\}$.

A binary function f is a special type of binary relation where for every $x \in dom(f)$ there is exactly one y such that $(x,y) \in f$; then y is the value of f in x denoted as $f(x)$. $f : X \to Y$ denotes a function f with $\mathrm{dom}(f) = X$ and $\mathrm{rng}(f) \subseteq Y$. The set of all such functions is ${}^{Y}X$. A function $f : X \to Y$ is a *surjection* (onto) if $\mathrm{rng}(f) = Y$, and it is an *injection* (one-to-one) if for any $x,y \in dom(f)$, $x \neq y \to f(x) \neq f(y)$. A function that is both a surjection and injection is called a *bijection*. Similarly to relations, we can define the inverse function f^{-1}, and the composition of functions $f : X \to Y$ and $g : Y \to Z$ as the function $f \circ g$ with $(f \circ g)(x) = g(f(x))$. The image of a set A, denoted as $f[A]$ is the set of function values for all elements of A, $f[A] = \{y|(x,y) \in f, x \in A\}$.

There are also two special types of relations which will be important later: equivalences and orders. An equivalence on a set X is a relation that is reflexive ($R(x,x)$ for all $x \in X$), symmetric ($R(x,y) \to R(y,x)$ for $x,y \in X$) and transitive ($(R(x,y) \wedge R(y,z)) \to R(x,z)$) for all $x,y,z \in X$). The extension of x in R is called the equivalence class of x and is also denoted as $[x]_R$. $X/R = \{R[x]|x \in X\}$ is the *quotient set of* X *by* R. The quotient set is always a partition of X and every partition of X also defines an equivalence on X (two elements are equivalent if they are in the same set in the partition).

The other important types of relations are the orders; usually an order is denoted as $\leq$. A relation is a *partial order of a set* X if it is reflexive ($x \leq x$ for $x \in X$), antisymmetric ($x \leq y \wedge y \leq x \to x = y$ for $x,y \in X$) and transitive ($x \leq y \wedge y \leq z \to x \leq z$ for $x,y,z \in X$). If, additionally, for every two elements $x,y \in X$ it holds that $x \leq y$ or $y \leq x$ (dichotomy) than $\leq$ is a total (linear) order. It is a well-order if additionally every non-empty subset of X has a least element.

Finally, an order of X is *dense* if X is not a singleton and for every two elements $x, y \in X$, there is another element $z \in X$ between these two $(x < y \rightarrow (\exists z)(x < z \wedge z < y))$, where $a < b$ means that $a \leq b \wedge a \neq b$.

For example, the common ordering of natural numbers ($\leq$ on $\mathbb{N}$) is a linear well-order (as every two natural numbers are comparable and every subset of natural numbers has a least element under this order), however, it is not a dense order, as for example there is no natural number between 0 and 1. On the other hand, the common ordering of rational numbers is a dense linear order (there is a rational number between any pair of distinct rational numbers), however it is not a well-order, as e.g. the set $\{x \in \mathbb{Q} \,|x \leq 0\}$ has no least element.

The natural numbers can be defined using the empty set in an inductive way – $0 = \emptyset, 1 = \{0\} = \{\emptyset\}, 2 = \{0,1\} = \{\emptyset, \{\emptyset\}\}, \ldots, n = \{0, \ldots, n-1\}, \ldots$. The set of all natural numbers $\mathbb{N}$ is the smallest set containing $\emptyset$ and closed under the operation of successor $S(x) = x \cup \{x\}$. The other common sets of numbers are the integers, which can be defined as $\mathbb{Z} = (\mathbb{N} \times \mathbb{N})/\sim$, with $(a,b) \sim (c,d)$ if and only if $a + d = b + c$. Similarly, the set of rational numbers $\mathbb{Q}$ can be defined as $\mathbb{Q} = (\mathbb{Z} \times (\mathbb{Z} \setminus \{0\}))/\sim$, with $(a,b) \sim (c,d)$ if and only if $ad = bc$. The definition of real numbers $\mathbb{R}$ is more complex. These are usually defined as cuts of the rational numbers $\mathbb{Q}$, where a cut is a partition of $\mathbb{Q}$ into two sets A and B, where all numbers in B are greater than all numbers of A, and A has no greatest element. For example, the cut corresponding to the irrational number $\sqrt{2}$ is $A = \{a \in \mathbb{Q} | a^2 < 2 \vee a < 0\}, B = \{b \in \mathbb{Q} | b^2 > 2 \wedge b > 0\}$.

Another important notion for the rest of the lecture deals with the cardinality ("size") of sets. A set X has a cardinality smaller or equal to the cardinality of a set Y ($X \preceq Y$) if there is an injective function $f : X \rightarrow Y$. If there is a bijection $f : X \rightarrow Y$ then we say that X and Y have the same cardinality ($X \approx Y$), finally X has strictly smaller cardinality than Y ($X \prec Y$) if ($X \preceq Y \wedge \neg(X \approx Y)$). For each set x, there is a cardinal number $\kappa \approx x$, denoted as $|x| = \kappa$. A set X is *finite* if $|X| = n$ for some $n \in \mathbb{N}$. It is *countable*, if it is finite or if $|x| = |\mathbb{N}| = \omega$. Otherwise, it is *uncountable*. The cardinality of $\mathcal{P}(\mathbb{N})$ is called the continuum.

It is interesting to know the cardinality of the common sets of numbers. Obviously, the set of natural numbers $\mathbb{N}$ is countable. A less obvious fact is that the sets of integers and rational numbers also have the same cardinality and are therefore also countable. For the integers, we can create an infinite sequence of integers $s = \langle 0, 1, -1, 2, -2, 3, -3, \ldots \rangle$, then the function $f(i) = s_i$ is an injective function $\mathbb{N} \rightarrow \mathbb{Z}$, therefore $\mathbb{Z} \preceq \mathbb{N}$. The other inequality ($\mathbb{N} \preceq \mathbb{Z}$) is obvious (use identity as the injective function). In order to show that the set of rational numbers $\mathbb{Q}$ is also countable, we can create a function $f(\frac{p}{q}) = 2^{|p|}3^q 5^{\text{sign}(p)}$ (we consider only cases where $p \in \mathbb{Z}, q \in \mathbb{N} \setminus \{0\}$, which clearly covers all the rationals). This is again an injective mapping $\mathbb{Q} \rightarrow \mathbb{N}$ and therefore $\mathbb{Q} \preceq \mathbb{N}$. As before, the other inequality is trivial. Finally, we can show that the set of real numbers $\mathbb{R}$ has a larger cardinality than the set of natural numbers $\mathbb{N}$.

Obviously $\mathbb{N} \preceq \mathbb{R}$ as $\mathbb{N} \subseteq \mathbb{R}$. Let us assume that both the sets have the same cardinality; in such a case there is a bijection $f : \mathbb{N} \to \mathbb{R}$. We will now define a new real number r in the following way. The integer part of the number is 0, the first digit after the decimal point is different from the first digit after decimal point in $f(0)$, the second digit is different from the second digit in $f(1)$, and so on[5]. This real number is different from all the numbers in $\{f(0), f(1), \dots\}$ as it differs from the number $f(i)$ in the i-th digit after the decimal point. This is a contradiction with the assumption that f is a bijection between $\mathbb{N}$ and $\mathbb{R}$ and therefore $\mathbb{N} \prec \mathbb{R}$.

[5] If we write the decimal value of the number r as $r = 0.r_0r_1r_2\dots$, where r_i is the i-th decimal digit, we can define $r_i = (f(i)_i + 1) \bmod 10$, where $f(i)_i$ is the i-th decimal digit of $f(i)$.

We will conclude our discussion of cardinalities by presenting Cantor's theorem.

Theorem 1 (Cantor)**.** *For every set x, $x \prec \mathcal{P}(x)$.*

Proof. First, $f(x) = \{x\}$ is an injection $X \to \mathcal{P}(x)$ and therefore $x \preceq \mathcal{P}(x)$. Suppose there is also an injective $g : \mathcal{P}(x) \to x$. We can define a set $y = \{g(z) | z \subseteq x \wedge g(z) \notin z\}$. Now, similarly to Russell's paradox, $g(y) \in y$ if and only if $g(y) \notin y$, which is a contradiction, and therefore there cannot be any such injective g and so $x \prec \mathcal{P}(x)$. Note that because g is injective, the element $g(y)$ could belong to the set y only because it fulfilled the condition (and not because it is the same as some $g(x)$ for $x \neq y$). □

As the tableau method used in this lecture relies on trees, we will conclude this preliminary section with a brief discussion about trees. Most of the readers are probably familiar with finite trees, however, we will sometimes need to work with infinite trees and therefore we define a *tree* as a set T with a partial order $<_T$ (called the tree order) with a unique least element (*the root*) and in which the set of predecessors of any element is well-ordered by $<_T$. In this definition a branch is a maximal linearly ordered subset of T. Apart from this difference in definition, we will use the common terminology on trees from graph theory. For simplicity, we will only consider finitely branching trees, where each node except the root has an immediate predecessor[6]. In such trees we can define the *levels of the tree*. The root is on level 0, the children of the nodes on the $(n-1)$-th level are on level n. The depth of a tree is the maximal $n \in \mathbb{N}$ of a non-empty level. In case the tree has an infinite branch it has an infinite depth ω. In an n-ary tree, each node has at most n children and a tree is finitely branching if each node has a finite number of children.

[6] This means, we will not deal, for example, with trees where the nodes would be set of rational numbers $\mathbb{Q}$ and the tree order $<_T$ would be the common order on $\mathbb{Q}$.

Lemma 1 (König)**.** *Every infinite, finitely branching tree contains an infinite branch.*

Proof. The root of the tree has only finitely many children, therefore there is a child of the root that is infinite. We choose this child and continue in the same way with its children, thus constructing an infinite branch. □

Apart from the tree order $<_T$ we sometimes need to work with *ordered trees* where the children of each node are additionally ordered

from left to right with a *left-to-right order* $<_L$. In a *labeled tree* each node also contains an additional piece of information. For example, the formula

$$(p \wedge q) \to q$$

can be represented as a labeled ordered tree on the right.

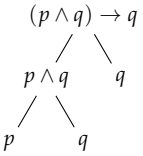

Figure 1: The labeled ordered tree representing the formula $(p \wedge q) \to q$.

Part I

Propositional Logic

Propositional Formulas and Models

In this chapter, we begin the discussion of propositional logic. We will define how propositional formulas look, what is a model in propositional logic and we will also discuss some special forms of formulas.

Propositional logic is the more basic type of logic (and predicate logic is an extension of propositional logic in a sense). Propositional formulas (propositions) are created from so-called *propositional variables* that represent an atomic fact which can either be true or false. These propositional variables can only be connected by common logic connectives ($\rightarrow$, $\leftrightarrow$, $\wedge$, $\vee$, $\neg$). Logical formulas can additionally use parentheses to indicate the order of application of connectives. While the propositional formulas are simple compared to formulas in other types of logic, they are still useful. One of the most important problems in propositional logic and in computer science in general is the satisfiability of propositional formulas (SAT). Many other NP-complete problems are often solved by transformation to the SAT problem and using one of the existing SAT solvers.

Syntax of Propositional Logic

The set of propositional variables is often called $\mathbb{P}$ and the variables themselves are usually named p, q, r, s or $p_0, p_1, \ldots, q_0, q_1$, or similarly. Now we can formally define a propositional formula (over $\mathbb{P}$).

Definition 1. Let $\mathbb{P}$ be the set of propositional variables, then

1. Every propositional variable from $\mathbb{P}$ is a propositional formula.
2. If φ and ψ are propositional formulas, then $(\varphi \rightarrow \psi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \leftrightarrow \psi)$, and $(\neg\varphi)$ are propositional formulas.
3. Every propositional formula is created by finite application of the two rules above.

The last part of the definition ensures that every formula is finite; this also means that each formula can contain only a finite number of distinct variables. The set of propositional variables used in a formula φ will be denoted as $\mathrm{var}(\varphi)$. On the other hand, the set of all propositional formulas using only variables from a set $\mathbb{P}$ will be denoted as $\mathrm{VF}_{\mathbb{P}}$.

Formulas are thus strings created from propositional variables, logical connectives, and parentheses, that fulfill the conditions in the

definition above. A substring of such a string that also fulfills the conditions is called a *sub-formula*.

The formal definition of formula dictates the use of parentheses around every sub-formula, which can be rather cumbersome. Therefore, we define priorities of the logical connectives and can thus omit some of the parentheses. The standard priorities are such that negation ($\neg$) has the highest priority (therefore parentheses around $(\neg\varphi)$ can always be omitted), conjunction and disjunction ($\vee, \wedge$) have "middle" priority, and implication and equivalence ($\rightarrow, \leftrightarrow$) have the lowest priority. Therefore, we can write $\varphi \wedge \psi \rightarrow \neg\varphi \vee \xi$ instead of $((\varphi \wedge \psi) \rightarrow ((\neg\varphi) \vee \xi))$.

Each formula can be also represented by a so-called *formation tree*, which is a finite ordered tree, whose nodes are labeled with propositions. The leaves are labeled with propositional variables. If a node has label $(\neg\varphi)$, it has a single child labeled with φ, and if a node has label $(\varphi \rightarrow \psi), (\varphi \wedge \psi), (\varphi \vee \psi)$, or $(\varphi \leftrightarrow \psi)$, it has two children, the left one with label φ, and the right one with label ψ. For example, the formula $p \wedge q \rightarrow \neg(p \vee s)$ is represented by the formation tree on the right.

It is simple to show (by induction on the number of nested parentheses) that each formula is associated with a unique formation tree.

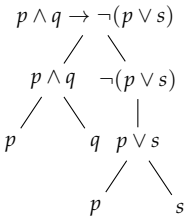

Figure 2: The formation tree representing the formula $p \wedge q \rightarrow \neg(p \vee s)$.

Semantics of Propositional Logic

Once we have a formal definition of formulas (the syntax of propositional logic), we can define their semantics (what formulas mean). The propositional variables represent atomic statements that can have one of two truth values – either 0 (false) or 1 (true). The truth value of a whole proposition is then given by the truth values of the variables and by the semantics of the logical connectives, which are given in Table 1 bellow.

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

Table 1: The semantics of logical connectives

We can also consider the table above as a definition of Boolean functions $\vee_1, \wedge_1, \rightarrow_1, \leftrightarrow_1$, and $-_1$, which implement the logical connectives. We will use these functions in cases where it is needed (e.g. while talking about truth values of propositions). More generally, any propositional formula with n variables defines a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ (later, we will also see that any Boolean function can be expressed using a propositional formula).

We also define two special logical formulas: the formula $\top \equiv p \vee \neg p$, which is always true, and the formula $\bot \equiv p \wedge \neg p$ which is always false.

We can now define truth assignments and the truth value of a

formula more formally.

Definition 2. A *truth assignment* is a function $v : \mathbb{P} \to \{0, 1\}$, that is $v \in {}^{\mathbb{P}}2$.

A *truth value* $\bar{v}(\varphi)$ of a propositional formula φ for a truth assignment v is defined inductively as:

- $\bar{v}(p) = v(p)$ if $p \in \mathbb{P}$
- $\bar{v}(\neg\varphi) = -_1(\bar{v}(\varphi))$
- $\bar{v}(\varphi \vee \psi) = \vee_1(\bar{v}(\varphi), \bar{v}(\psi))$
- $\bar{v}(\varphi \wedge \psi) = \wedge_1(\bar{v}(\varphi), \bar{v}(\psi))$
- $\bar{v}(\varphi \to \psi) = \to_1 (\bar{v}(\varphi), \bar{v}(\psi))$
- $\bar{v}(\varphi \leftrightarrow \psi) = \leftrightarrow_1 (\bar{v}(\varphi), \bar{v}(\psi))$

We can easily show (by induction on the structure of the formula) that the truth value of a formula φ depends only on the truth assignment of variables from $\mathrm{var}(\varphi)$.

A proposition φ over $\mathbb{P}$ is *true in (satisfied by) an assignment* $v \in {}^{\mathbb{P}}2$, if $\bar{v}(\varphi) = 1$. In such a case, v is called a *satisfying assignment* for φ; we denote this fact $v \vDash \varphi$. If the formula is true for all assignments $v \in {}^{\mathbb{P}}2$, we say that it is *valid (a tautology)* and denote the fact as $\vDash \varphi$. On the other hand, if there is no assignment for which the formula is true, it is called *unsatisfiable (a contradiction)*. A formula φ is *independent (a contingency)* if it is neither a tautology nor a contradiction, i.e. there are two assignments $v_1, v_2 \in {}^{\mathbb{P}}2$, such that $\bar{v}_1(\varphi) = 1$ and $\bar{v}_2(\varphi) = 0$. Finally, a formula is *satisfiable* if there is a truth assignment in which it is true.

A truth assignment of $\mathbb{P}$ is also called a model of the language $\mathbb{P}$. The set of all models of $\mathbb{P}$ is denoted as $M(\mathbb{P})$, and, obviously $M(\mathbb{P}) = {}^{\mathbb{P}}2$. A proposition φ over $\mathbb{P}$ is valid in a model $v \in M(\mathbb{P})$ if $\bar{v}(\varphi) = 1$. Then we also say that v is a model of φ, denoted as $v \vDash \varphi$. $M^{\mathbb{P}}(\varphi) = \{v \in M(\mathbb{P}) | v \vDash \varphi\}$ is the *class of all models* of φ. A formula is valid if it is true in every model of the language, it is unsatisfiable if it does not have a model, and is satisfiable if it has a model. It is independent if it is true in some model of the language and false in some other one. Formulas φ and ψ are logically equivalent ($\varphi \sim \psi$) if $M^{\mathbb{P}}(\varphi) = M^{\mathbb{P}}(\psi)$.

The last two paragraphs say basically the same thing; the difference is that in the latter one we use the notion of a model, which is central to logic. The notion of models and sets of models will be important later, and "model" is one of the key terms in logic.

In the definition of propositions, we used 5 different logical connectives. However, if we take a look at the table with their semantics, we may notice that, for example, $p \to q$ is equivalent to $\neg p \vee q$. Therefore, even without using implication ($\to$) we can still express everything we could with it. More formally, for every formula $\varphi \in \mathrm{VF}_{\mathbb{P}}$, there is an equivalent formula φ' that does not use implication. Moreover, we can notice that $p \leftrightarrow q$ is equivalent to $(p \to q) \wedge (q \to p)$, therefore we even do not need equivalence, and every formula can be written using only negation, conjunction, and disjunction ($\neg, \wedge, \vee$). This feature of a set of connectives can be defined more formally.

Definition 3. A set of connectives is *adequate* if they can express any Boolean function by some proposition formed from them.

We have already discussed that the set $\{\neg, \wedge, \vee\}$ is adequate. We can also show that the set $\{\rightarrow, \neg\}$ is adequate; the easiest way to do that is to realize that $(p \wedge q) \sim \neg(p \rightarrow \neg q)$ and $(p \vee q) \sim (\neg p \rightarrow q)$.

Generally we can also define custom connectives. For example, the so-called Shaffer stroke (NAND) is defined as $p \uparrow q \sim \neg(p \wedge q)$, and the Pierce arrow (NOR) is defined as $p \downarrow q \sim \neg(p \vee q)$. Interestingly, both $\{\uparrow\}$ and $\{\downarrow\}$ are adequate sets. This is an important fact for the construction of logical circuits as we can use a logical gate of only one kind (either NAND or NOR) to express any Boolean function.

Normal Forms

There are also special forms of formulas, which are often used. Among the most common ones are so-called conjunctive and disjunctive normal forms. In order to define these two forms, we first need to define a literal. A *literal* is a propositional variable or its negation. For example, if $\mathbb{P} = \{p, q\}$ then all the literals we can construct over $\mathbb{P}$ are $\{p, \neg p, q, \neg q\}$. A formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals. Disjunctions of literals are also called *clauses*, therefore we can also say that a CNF formula is a conjunction of clauses. On the other hand, a formula is in disjunctive normal form (DNF) if it is a disjunction of conjunctions of literals. So, for example, $(p \vee \neg q \vee r) \wedge (p \vee q) \wedge (\neg p \vee q \vee r)$ is a formula in CNF and $(\neg p \wedge q \wedge \neg r) \vee (\neg p \wedge \neg q) \vee (p \wedge \neg q \wedge \neg r)$ is a formula in DNF (and, moreover a negation of the previous one in CNF).

Now we would like to show that for every formula there is an equivalent formula in CNF and another equivalent formula in DNF. To this end, we will need the following set of rules, which can be proven by checking the truth table of the propositional connectives:

1. $(\varphi \rightarrow \psi) \sim (\neg\varphi \vee \psi), (\varphi \leftrightarrow \psi) \sim ((\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi))$
2. $\neg\neg\varphi \sim \varphi, \neg(\varphi \wedge \psi) \sim (\neg\varphi \vee \neg\psi), \neg(\varphi \vee \psi) \sim (\neg\varphi \wedge \neg\psi)$
3. $(\varphi \vee (\psi \wedge \chi)) \sim ((\psi \wedge \chi) \vee \varphi) \sim ((\varphi \vee \psi) \wedge (\varphi \vee \chi))$
4. $(\varphi \wedge (\psi \vee \chi)) \sim ((\psi \vee \chi) \wedge \varphi) \sim ((\varphi \wedge \psi) \vee (\varphi \wedge \chi))$

We can also easily show (again by induction on the structure of the formula) that if we have a formula φ' which is obtained from φ by replacing some occurrences of its sub-formula ψ with an equivalent sub-formula ψ', then $\varphi \sim \varphi'$.

And finally, we can show the following theorem.

Theorem 2. *For every formula φ over $\mathbb{P}$, there are formulas φ_C and φ_D such that φ_C is in CNF, φ_D is in DNF and $\varphi \sim \varphi_C$ and $\varphi \sim \varphi_D$.*

Proof. The propositions φ_C and φ_D can be obtained from φ by applying the rules 1 to 4 mentioned above. □

The discussion above shows one of the ways to obtain formulas in CNF and DNF equivalent to a given formula. We can in fact apply

the rules in the order in which they are presented. First, we remove all the implications and equivalences by using rule no. 1. Then we move all negations to the literals (i.e. there are no negations outside of parentheses), using rule no. 2 and, finally, we repeatedly apply rules no. 3 and 4 to obtain the CNF and DNF.

This syntactic approach is not the only one to obtain CNF/DNF from a given formula. We can also construct the truth table of the formula and then read the CNF/DNF almost directly from the table. We will show a more general approach here: we will construct CNF and DNF formulas φ_C and φ_D such that $M^{\mathbb{P}}(\varphi_C) = M^{\mathbb{P}}(\varphi_D) = K \subseteq M(\mathbb{P})$, for a given finite set of truth assignments K over a finite $\mathbb{P}$.

Before we show the construction, we will define the notion of p^t for a variable p and a truth value t as

$$p^t = \begin{cases} p & \text{if } t = 1 \\ \neg p & \text{if } t = 0 \end{cases} .$$

Now we can easily see that for a single assignment $v \in K$, the set of models of the formula $\bigwedge_{p \in \mathbb{P}} p^{v(p)}$ contains only v. For a set of assignments K, we can just make a disjunction over all assignments in K (remember K is a finite set). Therefore,

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

Thus we have constructed a formula in DNF whose models are exactly the set K.

Constructing a formula φ in CNF such that $M(\varphi) = K$ for some given finite K is slightly more complex. However, we can use the fact that the negation of a formula in DNF is a formula in CNF. Negating a formula in CNF/DNF means changing all the conjunctions to disjunctions and vice versa and changing all literals to the complementary ones (i.e. changing p to $\neg p$ and vice versa). So, we start by creating a formula $\neg\varphi$ in DNF for the set ${}^{\mathbb{P}}2 \setminus K$ according to the approach above. Then we negate the formula, thus obtaining φ in CNF such that $M(\varphi) = K$. Following these two steps we obtain the CNF formula

$$\varphi = \bigwedge_{v \in {}^{\mathbb{P}}2 \setminus K} \bigvee p^{-1 v(p)}$$

such that $M(\varphi) = K$.

If we want to use this approach to create a formula in CNF or DNF equivalent to a formula φ, we simply choose $K = M(\varphi)$. This description also shows that any Boolean function f (i.e. function $f : \{0,1\}^n \to \{0,1\}$) can be expressed as a proposition. We can choose $K = \{v | f(v) = 1\}$.

Both the techniques described above lead to an equivalent formula in CNF/DNF. The table-based method is typically used only for formulas with lower number of variables, as the size of the table for a formula with n variables is 2^n.

Logical theories

In mathematics, we often need to work in theories – we assume that some facts are true (the axioms of the theory) and are interested in which other facts are true. Therefore, in logic, we define a *propositional theory over the language* $\mathbb{P}$ as a set of propositions from $\mathrm{VF}_{\mathbb{P}}$. These propositions are called *axioms*. An assignments $v \in M(\mathbb{P})$ is a *model of theory* T ($v \vDash T$), if all axioms of T are true in v. Similarly to formulas, we define the *class of models of* T as $M(T) = \{v \in M(\mathbb{P}) | v \vDash \varphi \text{ for all } \varphi \in T\}$. A finite theory is equivalent to a conjunction of its axioms. We will also write $M(T, \varphi)$ as a shortcut for $M(T \cup \{\varphi\})$.

We can now re-define semantic concepts with respect to a theory. Let T be a theory over $\mathbb{P}$ and φ a proposition over $\mathbb{P}$. We say that φ *is true (valid) in* T ($T \vDash \varphi$) it if is true in every model of T. In such a case, we also say that φ is a (semantic) consequence of T. A formula φ is *unsatisfiable (contradictory) in* T *(inconsistent with* T*)*, if it is false in every model of T. It is *independent (a contingency) in* T, if it is true is some model of T and false in some other model of T, and *satisfiable in* T if it is true in some model of T. Two propositions φ and ψ are *equivalent in* T *(*T*-equivalent)* ($\varphi \sim_T \psi$), if for every model $v \in M(T)$, $v \vDash \varphi$ if and only if $v \vDash \psi$. For an empty theory ($T = \emptyset$), or for a theory where all axioms are tautologies, the re-definitions in this paragraph are equivalent to the definitions mentioned earlier.

The concepts defined above can also be expressed using the sets of models. For example $T \vDash \varphi$ is the same as $M(T) \subseteq M(\varphi)$, and $\varphi \sim_T \psi$ is equivalent to $M(T, \varphi) = M(T, \psi)$.

For each theory, we define its consequences as the set of all propositions that are true in the theory – $\theta^{\mathbb{P}}(T) = \{\varphi \,| \varphi \in \mathrm{VF}_{\mathbb{P}}, T \vDash \varphi\}$. Now, if we have two theories T and T', such that $T \subseteq T'$ over $\mathbb{P}$, we can prove that $T \subseteq \theta^{\mathbb{P}}(T) = \theta^{\mathbb{P}}(\theta^{\mathbb{P}}(T)) \subseteq \theta^{\mathbb{P}}(T')$. The first part says, that each axiom of T is always a consequence of T. This makes sense, as an axiom of T is by definition true in all models of T. The next part says, that the consequences of consequences of T are still the original consequences. However, this is also simple to show. Obviously, $M^{\mathbb{P}}(T) = M^{\mathbb{P}}(\theta(T))$ and therefore also $\theta^{\mathbb{P}}(T) = \theta^{\mathbb{P}}(\theta^{\mathbb{P}}(T))$ by the definition of a consequence. Finally, if we have a formula φ which is valid in all models of T, then φ is also valid in all models of T' ($T \subseteq T'$) as each model of T' also must be a model of T. Therefore $\theta^{\mathbb{P}}(T) \subseteq \theta^{\mathbb{P}}(T')$.

Similarly, if we have propositions $\varphi, \varphi_1, \varphi_2, \ldots \varphi_n$ over $\mathbb{P}$, we can show that $\varphi \in \theta^{\mathbb{P}}(\{\varphi_1, \ldots, \varphi_n\})$ if and only if $\vDash (\varphi_1 \wedge \cdots \wedge \varphi_n) \rightarrow \varphi$.

A theory T over $\mathbb{P}$ is *inconsistent (unsatisfiable)* if $T \vDash \bot$, otherwise T is *consistent (satisfiable)*. A theory is consistent if and only if it has a model. A theory is *complete* if it is consistent and $T \vDash \varphi$ or $T \vDash \neg\varphi$ for every $\varphi \in \mathrm{VF}_{\mathbb{P}}$, i.e. there are no independent propositions in T. This is also equivalent to the fact that T has exactly one model (if T had two models v_1 and v_2, then there would be a propositional variable p, such that $v_1(p) \neq v_2(p)$ and therefore the formula p is true in one of the models and false in the other one, thus p is independent). In

mathematics, we very often create new theories by adding axioms to other theories. Such new theories are called extensions of the original theories. More formally, a theory T over $\mathbb{P}$ is an *extension* of T' over $\mathbb{P}'$, if $\mathbb{P}' \subseteq \mathbb{P}$ and $\theta^{\mathbb{P}'}(T') \subseteq \theta^{\mathbb{P}}(T)$. An extension is *simple* if $\mathbb{P} = \mathbb{P}'$, and it is *conservative* if $\theta^{\mathbb{P}'}(T') = \theta^{\mathbb{P}}(T) \cap \mathrm{VF}_{\mathbb{P}'}$. Two theories T and T' are equivalent, if T is an extension of T' and vice versa.

Although we motivated the notion of extension by adding new axioms, it is defined more generally using the sets of consequences of the theory. This abstracts from the particular axioms and considers all equivalent theories the same. The notion of extension can also be expressed with the sets of models, if both theories T and T' are over the same language $\mathbb{P}$. In such a case T is an extension of T' if and only if $M^{\mathbb{P}}(T) \subseteq M^{\mathbb{P}}(T')$, and the two theories are equivalent if $M^{\mathbb{P}}(T) = M^{\mathbb{P}}(T')$.

We will conclude this section with the discussion about the number of nonequivalent propositions and theories over a finite language $\mathbb{P}$. We defined two formulas or theories to be equivalent if they have the same sets of models. Therefore, if we want to compute the number of non-equivalent theories/formulas, we can instead compute the number of sets of models. So, if $|\mathbb{P}| = n$, then there are 2^{2^n} non-equivalent formulas (theories) over $\mathbb{P}$, as there are 2^n different assignments, and every set of assignments represents a formula (remember, we know how to write that formula in CNF/DNF) or a theory.

Using similar reasoning, we can show how many nonequivalent valid (or contradictory – the number is the same) propositions are there in a theory. A valid proposition is true in all models of T, therefore there are $2^n - |M(T)|$ assignments where a valid proposition can be (but does not have to be) true. This means there are $2^{2^n - |M(T)|}$ valid (and contradictory) propositions in T. Every proposition is either valid, contradictory or independent, therefore there are $2^{2^n} - 2 \times 2^{2^n - |M(T)|}$ nonequivalent independent propositions in T. A theory has $2^{|M(T)|}$ simple extensions. One of these is contradictory (the set of models of an extension is a subset of the models of the original theory), and the same theory has $|M(T)|$ simple complete extensions (those correspond to single-element subsets of $M(T)$).

Instead of talking about nonequivalent propositions, we can also discuss T-nonequivalent propositions. There are $2^{|M(T)|}$ T-nonequivalent propositions (we now consider only subsets of $M(T)$ as the possible sets of models for the proposition). One of them is valid and one is contradictory in T, thus the number of T-nonequivalent propositions independent in T is $2^{|M(T)|} - 2$.

The fact that we can use the number of subsets of models while computing the number of nonequivalent theories or formulas is more formally explained by so-called Lindenbaum-Tarski algebra. For a consistent theory T over $\mathbb{P}$, we can define operations $\neg, \wedge, \vee, \bot, \top$ on the quotient set $\mathrm{VF}_{\mathbb{P}} / \sim_T$ by using representatives, e.g. $[\varphi]_{\sim_T} \wedge [\psi]_{\sim_T} = [\varphi \wedge \psi]_{\sim_T}$. Then $AV^{\mathbb{P}}(T) = \langle \mathrm{VF}_{\mathbb{P}} / \sim_T, \neg, \wedge, \vee, \bot, \top \rangle$ is the *Lindenbaum-Tarski algebra for* T. Since $\varphi \sim_T \psi$ if and only if $M(T, \varphi) = M(T, \psi)$ then $h([h]_{\sim_T}) = M(T, \varphi)$ is an injective function $h : \mathrm{VF}_{\mathbb{P}} \to \mathcal{P}(M(T))$.

If $M(T)$ is finite then h is additionally surjective, and therefore AV is isomorphic to the algebra of sets $\mathcal{P}(M(T))$.

Satisfiability of Propositional Formulas

The problem of satisfiability of logical formulas is one of the central problems in computer science. The general question posed by the problem is whether a given formula in CNF is satisfiable. In general, this problem is NP-complete[7], which means that we do not know any polynomial-time algorithm to solve it. However, there are some special types of formulas for which SAT can be solved in polynomial time. In this section, we will discuss such formulas and show the algorithms that solve SAT for these. We will also briefly discuss local search algorithms for SAT, and describe the complete (but generally exponential) DPLL [8] procedure.

The first class of formulas are so called 2-CNF. A formula is in *k-CNF* if it is a conjunction of clauses and each of the clauses contains at most k literals. The SAT problem for k-CNF formulas is called k-SAT. The k-SAT problem is NP-complete for $k > 2$, however for $k = 2$ it can be solved in polynomial time using the so-called implication graph of the formula.

Definition 4. Let φ is a formula over $\mathbb{P}$ in 2-CNF, $\varphi \equiv \bigwedge_{i=1}^{j}(l_{i1} \vee l_{i2}) \wedge \bigwedge_{i=j+1}^{k} l_i$. The *implication graph* of φ is an oriented graph $G_\varphi = (V, E)$, where the set of vertices is

$$V = \{p | p \in \mathbb{P}\} \cup \{\neg p | p \in \mathbb{P}\}$$

and the set of edges is

$$E = \{(\bar{l}_{i1}, l_{i2}) | 1 \leq i \leq j\} \cup \{(\bar{l}_{i2}, l_{i1}) | 1 \leq i \leq j\} \cup \{(\bar{l}_i, l_i) | j+1 \leq i \leq k\}\,.$$

In the implication graph, the set of vertices corresponds to all literals from variables in $\mathrm{var}(\varphi)$ and each clause in the formula is represented as one or two edges. For a clause $l_1 \vee l_2$, we include two edges (implications) $\bar{l}_1 \to l_2$ and $\bar{l}_2 \to l_1$. These two implications are logically equivalent to the clause. For a *unit clause* l (a clause with only a single literal), we include a single edge $\bar{l} \to l$; this is also equivalent to l. The implication graph thus contains the 2-CNF formula written as implications between its literals. The implication graph of a formula can be constructed in a time linear in the length of the formula.

Let us now assume that a truth assignment $v \in {}^{\mathbb{P}}2$ satisfies a formula φ. In such a case, in every strongly connected component[9] in G_φ, all the literals have the same truth value. Otherwise there would be an implication which is not true in the assignment, which contradicts the fact that the whole formula is true in the assignment[10]. This also means that if we have a satisfying assignment for φ, none of the strongly connected components contain both a literal and its negation.

Can we use the implication graph of a formula to obtain a satisfying truth assignment? Indeed we can, but only if none of the strongly connected components contain a pair of complementary literals. In

[7] A problem c is NP-complete, if it is NP and if any other NP problem is reducible to c in polynomial time. A problem is NP if given a candidate solution, we can check in polynomial time that it is indeed a solution. A problem p is reducible to c in polynomial time, if we can transform each instance of c into an instance of p in polynomial time.

[8] The name of the procedure is derived from the names of its authors – it was introduced in 1962 by Martin Davis, George Logemann and Donald W. Loveland as an extension of an earlier procedure by Martin Davis and Hilary Putnam.

[9] In a strongly connected component, there is an oriented path between every pair of vertices.

[10] Assume that literals l_1 and l_n are in the same strongly connected component and that $v(l_1) = 1$ and $v(l_n) = 0$. There is a chain of implications $l_1 \to l_2, l_2 \to l_3, \ldots, l_{n-1} \to l_n$, at least one of these must be $1 \to 0$ and therefore false.

such a case, we can contract each of the strongly connected components into a single vertex (thus obtaining a graph G^*_φ). Such a graph would be acyclic and therefore has a topological ordering $<$. We create an assignment v in a few steps: for every unassigned component in increasing order of $<$, we assign 0 to all its literals and 1 to all the complementary literals in the graph (they would in fact also form an strongly connected component). Such an assignment is satisfying for φ. If not, G^*_φ would contain edges $p \to q$ and $\bar{q} \to \bar{p}$ with $v(p) = 1$ and $v(q) = 0$, but that contradicts the order of assignments as $p < q$ and $\bar{q} < \bar{p}$.

The discussion above can be summarized in the theorem bellow.

Theorem 3. *Proposition φ in 2-CNF is satisfiable if and only if no strongly connected component of its implication graph G_φ contains a pair of complementary literals.*

As the implication graph can be constructed in linear time and the strongly connected components can also be found in linear time, the 2-SAT problem can be solved in linear time.

Another class of formulas where SAT can be solved in polynomial time are conjunctions of clauses with at most one positive literal. Such clauses are called Horn clauses, and such formulas are called Horn formulas. The problem of satisfiability of Horn formulas is called Horn-SAT.

The Horn clauses can also be interpreted as implications. The Horn clause $(\neg p_1 \vee \neg p_2 \vee \cdots \vee \neg p_n \vee q)$ is equivalent to the implication $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \to q$.

Deciding whether a Horn formula φ is satisfiable or not is simple, and can be done using the following algorithm:

1. If φ contains a pair of unit clauses l and $\bar{l}$ (a pair of complementary literals) it is not satisfiable.

2. If φ contains a unit clause l, assign 1 to l, remove all clauses containing l, remove $\bar{l}$ from all clauses, and continue from the start.

3. If φ does not contain a unit clause, it is satisfied by assigning 0 to all remaining propositional variables.

The first step of the algorithm is obviously correct, as $p \wedge \neg p$ is a contradiction, and the last step follows from the form of Horn formulas, as each of the remaining clauses contains at least one negative literal. It remains to show that the second step (also called *unit propagation*) is also correct. The formula φ can be satisfied only if each of its clauses is true, and therefore the unit clause l must also be true. Once we assign 1 to l, we can remove all the clauses that contain l (these are already satisfied) and we can also remove $\bar{l}$ from all the remaining clauses as $\bar{l}$ is 0, and therefore the clauses need to be satisfied by other literals.

This shows that Horn-SAT can be solved in polynomial time. The direct implementation of the algorithm described above is quadratic, however there are even linear implementations.

We already mentioned that there are no polynomial algorithms for the SAT problem in general, but we can use some local search algorithms to attempt to solve the problem. For example, the GSAT algorithm starts with a random truth assignment. If this assignment satisfies the formula, the algorithm ends. Otherwise it flips the truth value for one of the variables – it chooses the variable whose change leads to the smallest number of unsatisfied clauses in the new assignment. There is a small chance to change a random variable (this allows the algorithm to escape local minima in the number of unsatisfied clauses). The WalkSAT algorithm works in a similar way, but instead of picking a variable from the whole formula, it first selects a random clause and picks a variable from it which minimizes the number of previously satisfied clauses that become unsatisfied by the change. It also has a small chance to pick a variable at random.

While none of these algorithms can guarantee that they will find a satisfying assignment if one exists, they are very fast, and very often can indeed find a satisfying assignment.

A complete algorithm (which will always find an assignment, if one exists) can be implemented using backtracking and testing all possible assignments, however, such an algorithm is generally exponential.

The DPLL procedure implements this kind of backtracking with some improvements. It first removes all clauses that are tautologies, then if a clause becomes empty during the run of the algorithm, it indicates that the current partial assignment cannot satisfy the formula and the DPLL procedure fails. After these simple steps, the DPPL procedure simplifies the formula using unit propagation and so-called *pure literal elimination*[11]. If none of the previous steps can be applied, the DPLL procedure uses a splitting rule: it selects a literal and tries to call the DPLL procedure twice, once for each possible truth assignment of that literal. If at least one of these calls succeeds, the formula is satisfiable.

[11] if a literal l is only positive or only negative in the formula, it can be assigned such value $v(l) = 1$ and all the clauses containing it can be removed

Formal Proof Systems

Up to now, we have mostly discussed the semantics of propositional logic, and have also defined many different terms semantically using the notions of truth assignments and models. We have defined a consequence of a theory as a formula that is true in all models of the theory. However, in mathematics, we usually do not check all possible models of a theory in order to tell whether a given formula is a consequence or not. Instead, we prove the formula from the axioms of the theory.

In this chapter, we will formalize the notion of proof as a syntactical method that can be used to prove formulas in propositional logic. The formalization will be called a proof system, and, informally, a proof system is a collection of syntactical rules that provide a proof of a given formula in a given theory. The proof is then a finite object that can be built from the axioms of the theory, and if a formula has a proof, it can be found algorithmically.

There are many different proof systems, among them are the tableau method, Hilbert systems and Gentzen systems. We will discuss the tableau method in detail later and we will also briefly mention Hilbert systems. However, a proof system can only be useful, if any formula proved by the system is also valid, and vice versa, if every valid formula can be proven. These two features of a proof system are called soundness and completeness.

Tableau Method

The tableau method is a proof system, where the proof (tableau) of a formula φ in a theory T is a binary labeled tree representing a search for a model of T where φ is not true (a counterexample). If the search fails, the formula is proved and in such a case the tableau is finite. In case there is a counterexample of φ, the tableau can be infinite and there is a branch in the tree that provides the counterexample.

In tableau methods, we assume a fixed and countable language $\mathbb{P}$; in this case, also every theory over $\mathbb{P}$ is countable.

We already mentioned that every tableau is a labeled binary tree. The nodes in the tree are labeled by *entries*, which are formulas with a *sign* T/F that represent the assumption that the formula is true (T) or false (F). The tree will be constructed using *atomic tableaux* and a set of rules. For a propositional variable p and propositions φ, ψ, the atomic tableaux are given in the figure bellow.

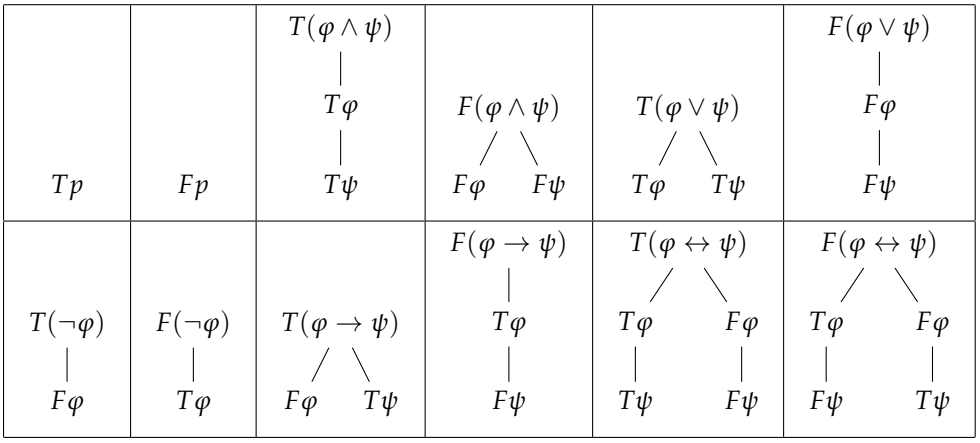

Figure 3: The atomic tableaux

Informal! The labels in each of the tableaux show whether a formula φ should be true ($T\varphi$) or false ($F\varphi$). The tableaux themselves then "rewrite" the requirement in their root into more simple requirements. For example, the atomic tableau for $T(\varphi \to \psi)$ expresses that a formula $(\varphi \to \psi)$ is true (T), if φ is false ($F\varphi$) or ψ is true $T\psi$. The "or" is expressed by the two children. On the other hand, the atomic tableau for $F(\varphi \to \psi)$ says that $\varphi \to \psi$ is false, if φ is true and ψ is false. The "and" is presented by the fact, that both these facts are on a single branch.
While the atomic tableaux are based on the semantics of propositional logic, the tableau method itself is purely syntactic – it only says how to manipulate tableaux in order to obtain the proof of a formula.

Using the atomic tableaux, we can define tableaux in general.

Definition 5. A *finite tableau* is a binary tree labeled with entries defined inductively as follows:

1. Every atomic tableau is a finite tableau.
2. If P is an entry on a branch V in a finite tableau τ and τ' is obtained by adjoining the atomic tableau for P at the end of branch V, then τ' is also a finite tableau.
3. Every finite tableau is formed by finite number of steps above.

A *tableau* τ is a (potentially infinite) sequence $\tau_0, \tau_1, \ldots, \tau_n, \ldots$ of finite tableaux such that τ_{n+1} is formed from τ_n by an application of step 2 above. Formally, $\tau = \bigcup \tau_n$.

An example of a tableau is shown below. In propositional logic, we do not need to repeat the entries that we expand, therefore, we will generally use only the version on the right, where these repeated entries are removed.[12]

The definition above does not specify how to choose an entry P on branch V for expansion. Later, we will define systematic tableaux, where this is specified.

Before we can define the formal notion of proof using the tableau method, we first need to discuss some of the terms related to tableaux.

[12] In predicate logic, some of the repeated entries need to be included in the tableau again.

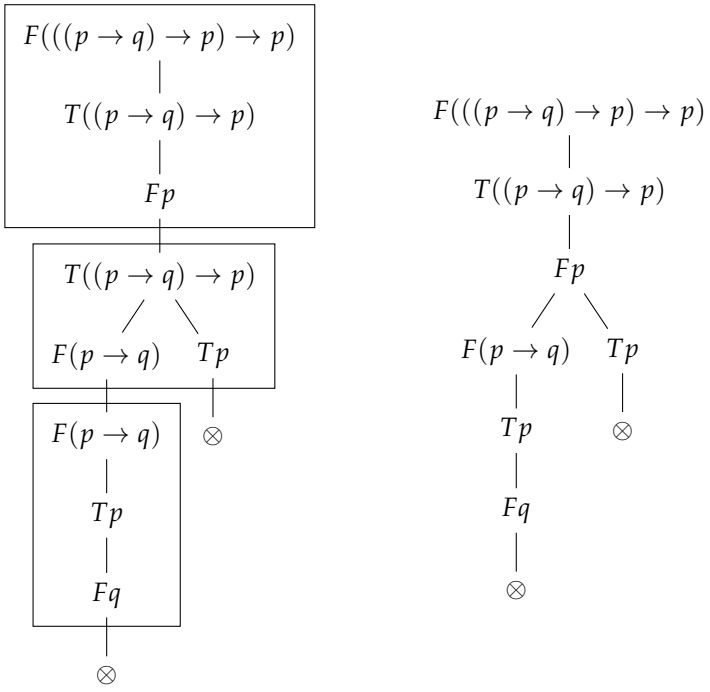

Figure 4: Example tableau. The rectangles on the left show the atomic tableaux used. The version on the right removes the repeated entries. The symbol $\otimes$ denotes a contradictory branch.

For an entry P on a branch V in a tableau τ, we say that *P is reduced on V* if it occurs on V as a root of an atomic tableau. A *branch V is contradictory* if it contains entries $T\varphi$ and $F\varphi$ for some proposition φ, otherwise it is noncontradictory. A *branch V is finished* if it is contradictory, or every entry on V is reduced on V. Finally, a *tableau τ is finished* if every branch in τ is finished, and τ is contradictory if every branch in τ is contradictory.

A *tableau proof of φ* is a contradictory tableau with the root entry $F\varphi$. A formula *φ is tableau provable* ($\vdash \varphi$) if it has a tableau proof. On the other hand, a *refutation of φ by tableau* is a contradictory tableau with the root entry $T\varphi$, and *φ is tableau refutable* it it has a tableau refutation; in this case we write $\vdash \neg\varphi$.

Informal! Why does a tableau proof of φ start with $F\varphi$? Tableaux in fact represent systematic searches for assignments that fulfill the condition expressed by the entry in the root. Therefore, if we cannot find a truth assignment in which φ is false (the tableau for $F\varphi$ is contradictory), then φ must be true in all assignments, and therefore valid. The formal proof of the soundness and completeness of the tableau methods will be discussed shortly.

Figure 5 shows a tableau with the root entry $F(((\neg p \land \neg q) \lor p) \to (\neg p \land \neg q))$. The tableau has three branches. The leftmost one is contradictory, as it contains both $F(\neg p \land \neg q)$ and $T(\neg p \land \neg q)$, the middle one is finished and noncontradictory, as every entry on that branch is expanded on it, and the rightmost one is unfinished, as the entry $F(\neg q)$ is not expanded on that branch.

On the other hand, the tableau in Figure 4 is a tableau proof of the proposition $(((p \to q) \to p) \to p)$, as it starts with the entry $F(((p \to q) \to p) \to p)$ and all its branches are contradictory.

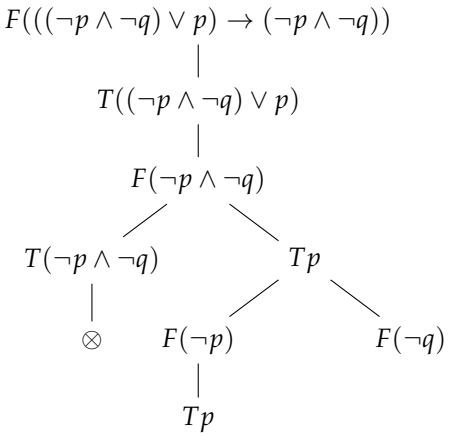

Figure 5: Example tableau. Both left and middle branches are finished. The left one is also contradictory, while the middle one is noncontradictory. The right branch is not finished.

We often need to work with theories, and also prove propositions in a theory. Therefore, the notion of tableau needs to be generalized to the notion of a tableau from a theory. Theories provide axioms, these are assumed to be true, and therefore the tableau from a theory T can additionally contain entries of the from $T\varphi$ for an axiom $\varphi \in T$. More formally, a *finite tableau from a theory* T is a generalized tableau with an additional rule – if V is a branch of a finite tableau (from T) and $\varphi \in T$, then by adjoining $T\varphi$ at the end of V we obtain a finite tableau from T. The rest of the definitions related to tableaux can be generalized in the same way. *A tableau from* T is a sequence $\tau_0, \tau_1, \ldots, \tau_n, \ldots$ of finite tableaux from T such that τ_{n+1} is formed from τ_n applying rule 2 (from the definition of tableaux), or the additional rule above; formally, $\tau = \bigcup \tau_n$. A *tableau proof of* φ *from* T is a contradictory tableau from T with $F\varphi$ in the root. $T \vdash \varphi$ denotes that φ is tableau provable from T. A *refutation of* φ *by a tableau from* T is a contradictory tableau from T with $T\varphi$ in the root. A branch V of a tableau from T is finished if it is contradictory, or every entry on V is already reduced on V and, additionally, V contains $T\varphi$ for each $\varphi \in T$.

While the current definition of tableaux is enough for proving propositions in theories, here we provide a stricter definition of so-called systematic tableau. We will see later that a systematic tableau is always finished and, in case the tableau is a proof of a proposition, it is also finite. The definition prescribes the precise order of steps to use while constructing tableaux from theories – it specifies which entry in the tableau should be expanded next and also which axiom from the theory should be added next.

Definition 6. Let R be an entry and $T = \{\varphi_0, \varphi_1, \ldots\}$ a theory. Then the *systematic tableau* τ from T for the entry R is the result of the following construction, i.e. $\tau = \bigcup \tau_n$

1. t_0 is the atomic tableau for R. Then proceed with the following steps until possible:

2. Let P be the leftmost entry in the smallest possible level of the tableau τ_n such that P is not reduced on some noncontradictory branch through P.

3. Let τ'_n be the tableau obtained from τ_n by adjoining the atomic tableau for P to every noncontradictory branch through P ($\tau'_n = \tau_n$ if no such P exists).

4. Let τ_{n+1} be the tableau obtained from τ'_n by adjoining $T\varphi_n$ to every noncontradictory branch that does not contain $T\varphi_n$ (if φ_n does not exist, $\tau_{n+1} = \tau'_n$).

The first thing to notice is that every systematic tableau is finished. Assume we have a tableau $\tau = \bigcup \tau_n$. If there is a noncontradictory branch in τ, the prefix of this branch is noncontradictory in each τ_n. Therefore, the branch must contain $T\varphi_n$ for each φ_n in T. Let us now assume that there is an entry R, such that R is not reduced on a branch. However, there are only finitely many levels above R in τ and therefore only finitely many entries above R, thus R will be eventually selected in step 2 and reduced in step 3, which is a contradiction with R not being reduced. So, every noncontradictory branch in the tableau is finished (it contains $T\varphi_n$ for each $\varphi_n \in T$, and every entry on the branch is reduced).

Interestingly, if tableau is used as a proof, it is not only finished, it is also finite. More specifically – for every contradictory tableau $\tau = \bigcup t_n$, there is some n such that τ_n is a contradictory finite tableau. Why? Let S be the set of nodes in τ that have no pair of contradictory entries $T\varphi$, $F\varphi$ amongst their predecessors. We can imagine this set as a "top part" of the tableau – the root is definitely in this set. If there is a node in the set, all of its predecessors are also there. Such a set S must be finite, because otherwise, by König's lemma, the subtree of τ induced by the set S would have an infinite branch (it is a finitely branching infinite tree), and therefore the tableau τ would not be contradictory. Now, since S is finite, all of the nodes in S belong to levels up to m for some m. Thus every node in level $m+1$ has a pair of contradictory entries among its predecessors. We can now choose n such that the top $m+1$ levels of τ are a subtree of τ_n. Every branch in τ_n now contains a pair of contradictory entries and is thus contradictory.

In the construction of systematic tableaux, we extend only noncontradictory branches, therefore if a systematic tableau (from a theory) is a proof, it is finite (remember that a proof is a contradictory tableau with $F\varphi$ in its root). This is an important result; it shows that if a formula has a proof, we have an algorithm (the construction of systematic tableau) that can find the proof in a finite amount of time. It also shows that any proof from a theory depends only on a finite number of axioms from the theory.

Soundness and Completeness

Now we want to show the soundness and completeness of the tableau method. We start with soundness and show that if a formula has a tableau proof from a theory, the formula is also valid in the theory. However, before we get to the proof, we need a definition and a lemma. We say that an *entry P agrees with an assignment v*, if P is $T\varphi$

and $\bar{v}(\varphi) = 1$, or if P is $F\varphi$ and $\bar{v}(\varphi) = 0$. A *branch V agrees with v* if every entry on V agrees with v.

Lemma 2. *Let v be a model of a theory T that agrees with the root entry of a tableau $\tau = \bigcup \tau_n$. Then τ contains a branch that agrees with v.*

Proof. We will find a sequence $V_0, V_1, \ldots$ for every n, such that V_n is a branch in τ_n, $V_n \subseteq V_{n+1}$ and V_n agrees with n. We start by verifying the lemma for all atomic tableaux, thus verifying the base of the induction. For example, if we have $v(p) = 1, v(q) = 0$ and an atomic tableau with root entry $T(p \vee q)$, then v agrees with the root entry, and the branch of the tableau containing Tp also agrees with v. We can check the other atomic tableaux similarly. Now, if τ_{n+1} is obtained from τ_n without extending V_n, we take $V_{n+1} = V_n$. If τ_{n+1} is obtained from τ_n by adjoining $T\varphi$ to V_n for some $\varphi \in T$, let V_{n+1} be this branch; then v agrees with V_{n+1} as v is a model of T (and therefore all axioms of T are true in v). Finally, if τ_{n+1} is obtained from τ_n by adjoining the atomic tableau for some entry P on V_n to the end of V_n, we can extend V_n to V_{n+1} as required as P agrees with v and all atomic tableaux are verified. (For example, if $P = T(p \vee q)$, and v is as in the example on atomic tableaux above, we obtain V_{n+1} by adding $T(p \vee q)$ and Tp to the end of V_n). □

Using the lemma, we can now easily prove the soundness of the tableau method in propositional logic.

Theorem 4 (Soundness of tableau method in propositional logic)**.** *For every theory T and proposition φ, if φ is tableau provable from T, then φ is valid in T, i.e. $T \vdash \varphi \Rightarrow T \vDash \varphi$.*

Proof. If the proposition φ is tableau provable from T, there is a contradictory tableau τ from T with root entry $F\varphi$. Suppose φ is not valid in T. In such a case, there is a model v of the theory T in which φ is false. Therefore, the root entry of the proof ($F\varphi$) agrees with v and by the previous lemma, there is a branch in τ that agrees with v. However, that leads to a contradiction as τ is the proof of φ from T, and therefore every branch of τ is contradictory and cannot agree with v. □

The soundness theorem says that whenever we have a tableau proof of a formula in a theory, the formula is valid. However, can we also prove any valid formula using the tableau method? We indeed can, as the completeness theorem states. Again, before we get to the proof of the completeness theorem, we prove a helper lemma, that formally shows that a noncontradictory branch in a finished tableau provides a counterexample.

Lemma 3. *Let V be a noncontradictory branch of a finished tableau τ. Then V agrees with the following assignment v:*

$$v(p) = \begin{cases} 1 & \text{if } Tp \text{ occurs on } V \\ 0 & \text{otherwise} \end{cases}$$

Proof. We prove the lemma by induction on the structure of formulas in entries on V.

- For an entry $T p$ on V, where p is a propositional variable, we have $\bar{v}(p) = 1$ by definition.
- For an entry $F p$ on V, the entry $T p$ is not on V as V is noncontradictory, and thus we have $\bar{v}(p) = 0$ by definition.
- For an entry $T(\varphi \wedge \psi)$, we have both $T\varphi$ and $T\psi$ on V as τ is finished, and by induction, we know that $\bar{v}(\varphi) = \bar{v}(\psi) = 1$, therefore $\bar{v}(\varphi \wedge \psi) = 1$ and v agrees with $T(\varphi \wedge \psi)$.
- For an entry $F(\varphi \wedge \psi)$, we have $F\varphi$ or $F\psi$ on V as τ is finished, therefore we have $\bar{v}(\varphi) = 0$, or $\bar{v}(\psi) = 0$, which leads to $\bar{v}(\varphi \wedge \psi) = 0$ and thus v agrees with $F(\varphi \wedge \psi)$.

The lemma can be proven for the other possible types of entries (with $\vee, \rightarrow, \leftrightarrow, \neg$) similarly to the last two steps for entries with $\wedge$. □

Using this lemma, it is simple to prove the completeness theorem.

Theorem 5. *For every theory T and proposition φ, if φ is valid in T, then φ is tableau provable from T, i.e. $T \vDash \varphi \Rightarrow T \vdash \varphi$.*

Proof. We will show that an arbitrary finished tableau τ from theory T with root entry $F\varphi$ is contradictory if φ is valid in T.

Assume (for contradiction) that there is a noncontradictory branch V in τ. The previous lemma provides an assignment v such that V agrees with v, therefore also the root entry $F\varphi$ agrees with v and thus $\bar{v}(\varphi) = 0$. Since V is finished, it contains $T\psi$ for every $\psi \in T$, but that means that v is a model of T (V agrees with v, therefore $\bar{v}(\psi) = 1$ for all $\psi \in T$). However, this contradicts the assumption that φ is valid in T, therefore every branch in τ is contradictory and τ is a proof of φ from T. □

We can now introduce a syntactic definition of the semantic terms defined earlier and discuss the relation between the syntactic and semantic notions. First of all, we define the *set of propositions provable from T*

$$\mathrm{Thm}^{\mathbb{P}}(T) = \{\varphi | \varphi \in \mathrm{VF}_{\mathbb{P}}, T \vdash \varphi\}.$$

We say that a theory T *is inconsistent*, if $T \vdash \bot$, otherwise T *is consistent*. A theory T *is complete* if it is consistent and every proposition is provable or refutable from T, i.e. if $T \vdash \neg\varphi$ or $T \vdash \varphi$ for every $\varphi \in \mathrm{VF}_{\mathbb{P}}$. A theory T over $\mathbb{P}$ is *an extension* of T' over $\mathbb{P}'$, if $\mathbb{P}' \subseteq \mathbb{P}$ and $\mathrm{Thm}^{\mathbb{P}'}(T') \subseteq \mathrm{Thm}^{\mathbb{P}}(T)$. The extension is *simple* if $\mathbb{P} = \mathbb{P}'$, and it is *conservative* if $\mathrm{Thm}^{\mathbb{P}'}(T') = \mathrm{Thm}^{\mathbb{P}}(T) \cap \mathrm{VF}_{\mathbb{P}'}$. Two *theories T and T' are equivalent* if T is an extension of T' and vice versa.

There are strong relations between the syntactic terms introduced above and the semantic terms introduced in the previous chapter. Most of these are corollaries of the soundness and completeness of the tableau method. For each theory T and propositions φ, ψ over $\mathbb{P}$

1. $T \vdash \varphi$ if and only if $T \vDash \varphi$,
2. $\mathrm{Thm}^{\mathbb{P}}(T) = \theta^{\mathbb{P}}(T)$,

3. T is inconsistent if and only if T is unsatisfiable, i.e. it has no model,
4. T is complete if and only if T is semantically complete, i.e. it has a single model,
5. (deduction theorem) $T \cup \{\varphi\} \vdash \psi$ if and only if $T \vdash \varphi \to \psi$.

Another important corollary of the theorems is the compactness theorem.

Theorem 6. *A theory T has a model if and only if every finite subset of T has a model.*

Proof. The implication to the right (if a theory has a model, every finite subset has a model) is trivial. In order to prove the other implication, we first realize that if T has no model, it is inconsistent, thus $T \vdash \bot$ and $\bot$ is provable by a systematic tableau τ from T. The tableau is finite, therefore τ is also provable from a finite subset of $T' \subseteq T$ (T' contains the axioms from T that were used in the proof), so T' is inconsistent and, therefore, has no model. □

While the compactness theorem is interesting in itself, it is also a very strong theorem that can be used to prove other theorems in different parts of mathematics. Consider for example the theorem on infinite k-colorable graphs[13]: a countably infinite graph $G = (V, E)$ is k-colorable if and only if each finite subgraph of G is k-colorable. Again, if the infinite graph is colorable, every finite subgraph is obviously also colorable. The other implication is more interesting. Consider a set of propositional variables $\mathbb{P} = \{p_{u,i} | u \in V, i \in k\}$, where $p_{u,i}$ means that vertex u has color i. We can create a theory T with axioms $p_{u,0} \vee p_{u,1} \vee \cdots \vee p_{u,k-1}$ for each $u \in V$ (every vertex has a color), $\neg(p_{u,i} \wedge p_{u,j})$ for every $u \in V, i < j < k$ (every vertex has only one color), and $\neg(p_{u,i} \wedge p_{v,i})$ for each $\{u, v\} \in E, i < k$ (two vertices connected with an edge do not have the same color). Obviously, G is colorable if and only if T has a model. We only need to show that every finite $T' \subseteq T$ has a model (and use the compactness theorem). Let G' be a subgraph of G induced by vertices u such that $p_{u,i}$ appears in T' for some i. By assumption, G' is k-colorable, and therefore T' has a model.

[13] A graph is k-colorable if there is a function $c : V \to k$, such that $c(u) \neq c(v)$ for every edge $\{u, v\} \in E$.

Hilbert systems

A (more traditional) alternative to the tableau method is the Hilbert calculus. In this proof system, formulas are defined using only implication ($\to$) and negation ($\neg$), and all other logical connectives are defined using these two (we already know that the set $\{\to, \neg\}$ is adequate, so this can be done). The Hilbert proof system then defines the following set of schemas of axioms (for two propositions $\varphi, \psi \in \mathrm{VF}_{\mathbb{P}}$):

1. $\varphi \to (\psi \to \varphi)$
2. $(\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))$
3. $(\neg\varphi \to \neg\psi) \to (\psi \to \varphi)$

Apart from the axioms, there is also a single inference rule: *modus ponens*, which can be expressed as

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

That means that if φ and $\varphi \to \psi$ are true we can infer that also ψ is true.

A proof of formula φ from a theory T in the Hilbert style is defined as a finite sequence of formulas $\varphi_0, \varphi_1, \ldots \varphi_n = \varphi$ such that for every $i \leq n$, φ_i is a logical axiom, or an axiom from the theory ($\varphi_n \in T$), or φ_i is inferred from φ_j and φ_k ($j, k < i$) using the modus ponens rule. As with the tableau method, a formula φ is provable from T ($T \vdash_H \varphi$), if it has a proof.

For example, we can show that $\varphi \to \psi$ is provable from $T = \{\neg\varphi\}$ for every ψ.

1. $\neg\varphi$
2. $\neg\varphi \to (\neg\psi \to \neg\varphi)$
3. $\neg\psi \to \neg\varphi$
4. $(\neg\psi \to \neg\varphi) \to (\varphi \to \psi)$
5. $\varphi \to \psi$

The first two steps are an axiom of a theory and a logical axiom (the schema number 2). The third formula is obtained from the previous two by modus ponens, the fourth one is again an axiom (by schema number 3), and the last one is obtained from formulas number 3 and 4 using modus ponens.

It is easy to prove the soundness of the Hilbert calculus ($T \vdash_H \varphi \Rightarrow T \vDash \varphi$). Logical axioms are tautologies, and axioms from T hold in all models of T, therefore soundness holds for axioms of any kind, and the modus ponens rule is sound (as can be easily checked using the truth tables of φ, $\varphi \to \psi$, and ψ). Thus, soundness is proved. The Hilbert calculus is also complete, but we will not show the proof here.

Resolution method

The resolution method is the base of many automated systems – SAT solvers, automated deduction or verification systems and Prolog[14] interpreters. The method assumes that the input formulas are given in CNF and it works with a set representation of the formulas (a CNF formula is represented as a set of sets of literals). The method has no explicit axioms, but some of the axioms are implicitly included. It uses a single inference rule (the resolution rule). Similarly to the tableau method, the resolution method is also a refutation procedure, i.e. it tries to show that a given formula or theory is unsatisfiable. There are several variants of the resolution method that give more specific rules on when the resolution rule can be applied (e.g. LI-resolution, or SLD-resolution).

[14] Prolog is a programming language based on the specification of programs as sets of Horn formulas.

Before we describe the resolution method formally, we must define the set representation of CNF formulas. Similarly to our discussion of CNF formulas, a literal is either a propositional variable or its negation. The complementary literal to l is still denoted as $\bar{l}$. A *clause* C is a finite set of literals, and an *empty clause*, denoted as $\square$, is never satisfied. A *formula* S is then a (possibly infinite) set of clauses. An empty formula $\emptyset$ is always satisfied. Infinite formulas represent infinite theories. A *(partial) assignment* $\mathcal{V}$ is a consistent set of literals (i.e. the set does not contain a complementary pair of literals). An assignment is *total* if it contains a positive or negative literal for each propositional variable. An assignment $\mathcal{V}$ satisfies a formula S (denoted as $\mathcal{V} \vDash S$) if $C \cap \mathcal{V} \neq \emptyset$ for each clause $C \in S$.

For example, the CNF-formula $((\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge (\neg r \vee \neg s) \wedge s)$ is represented as $S = \{\{\neg p, q\}, \{\neg p, \neg q, r\}, \{\neg r, \neg s\}, \{s\}\}$ and $\mathcal{V} = \{s, \neg r, \neg p\}$ is a satisfying assignment for S.

Informal! While the definitions above are different from those we used previously, they are in fact equivalent. The only reason why they are worded differently is the set representation of the CNF formulas. We know that a formula in CNF is a conjunction of clauses, therefore, we can only represent them as a set of clauses. A clause is a disjunction of literals, and therefore it is again natural to represent each clause as a set of literals. The definition of assignment may seem strange, but the set of literals only says which literals are true and which are false.

There is only one inference rule in resolution – the resolution rule: let C_1 and C_2 are clauses such that $l \in C_1$ and $\bar{l} \in C_2$, then infer a clause C (called a resolvent) such that $C = (C_1 \setminus \{l\}) \cup (C_2 \setminus \{\bar{l}\})$. The resolution rule is a special case of the cut rule:

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

for any formulas φ, ψ, χ.

It is easy to realize that the resolution rule is sound, i.e. if $\mathcal{V} \vDash C_1$ and $\mathcal{V} \vDash C_2$, then $\mathcal{V} \vDash C$ – the assignment $\mathcal{V}$ cannot contain a pair of complementary literals (by definition), therefore at least one of the intersections $\mathcal{V} \cap (C_1 \setminus \{l\})$ or $\mathcal{V} \cap (C_2 \setminus \{\bar{l}\})$ must be non-empty, therefore $\mathcal{V} \cap C$ is also non-empty.

A *resolution proof (deduction) of a clause C from formula S* is a finite sequence of clauses $C_0, C_1, \ldots, C_n = C$ such that for each $i \leq n$, $C_i \in S$, or C_i is a resolvent of some previous clauses. As usual, a *clause C is provable from formula S* ($S \vdash_R C$), if it has a resolution proof from S. We already mentioned that resolution is used as a refutation procedure. A *resolution refutation of formula S* is a resolution proof $S \vdash_R \square$, and a *formula is resolution refutable*, if there is such a proof.

Let us now show that resolution is also a sound and complete method. Soundness is simple, and follows from the soundness of the resolution rule.

Theorem 7 (Soundness of resolution)**.** *If a formula S is resolution refutable, it is unsatisfiable.*

Proof. Let $S \vdash_R \square$ and assume (for contradiction) there is an assignment $\mathcal{V}$ such that $\mathcal{V} \vDash S$. Because the resolution rule is sound, also $\mathcal{V} \vDash \square$, but that is not possible ($\square$ is never satisfied). $\square$

The proof of completeness is a bit more involved. To this end, we first define resolution trees, which in fact show how we obtained a proof of a clause. A *resolution tree* of clause C from formula S is a finite binary tree with nodes labeled by clauses such that the root is labeled by C, the leaves are labeled by clauses from S, and every inner node is labeled by the resolvent of its sons. Obviously, there is a resolution tree for C from S if and only if $S \vdash_R C$.

Another important notion is the *resolution closure* of a formula S, denoted as $\mathcal{R}(S)$ and defined as the smallest set containing all clauses of S and closed under the resolution rule, i.e. if $C_1, C_2 \in \mathcal{R}(S)$ and C is the resolvent of C_1 and C_2, then also $C \in \mathcal{R}(S)$. Obviously, $C \in \mathcal{R}(S)$ if and only if $S \vdash_R C$, and all the notions on resolution proofs can be also defined using the resolution trees and closures.

As a simple example of the resolution method, we can show that the formula $S = \{\{p,r\},\{q,\neg r\},\{\neg q\},\{\neg p,t\},\{\neg s\},\{s,\neg t\}\}$ is unsatisfiable as $S \vdash_R \square$.

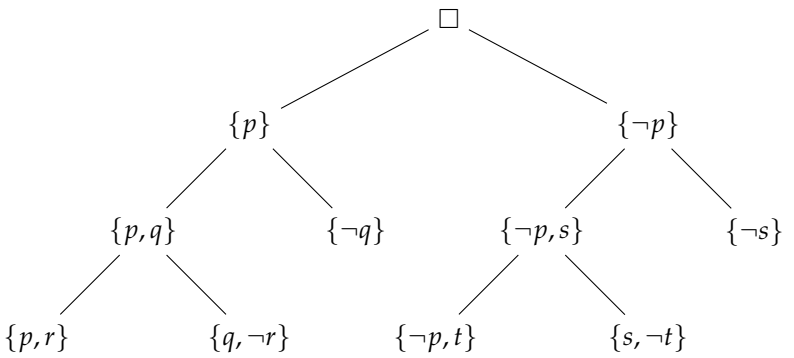

Figure 6: The resolution proof of $S \vdash_R \square$.

We can also compute the resolution closure

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

and as $\square \in \mathcal{R}(S)$, we also know that S in unsatisfiable.

In the proof of completeness, we will use the notion of reduction by substitution. Let S be a formula and l a literal; we define

$$S^l = \{C \setminus \{\bar{l}\} | l \notin C \in S\}.$$

The new formula S^l is in fact equivalent to a formula where the literal l was assigned a true value ($\top$) and $\bar{l}$ was assigned a false value ($\bot$). In such a case, any clause containing l can be removed (as it is satisfied), and $\bar{l}$ is removed from all other clauses. The formula S^l does not contain any of the literals l and $\bar{l}$, and if S contained a clause $\{\bar{l}\}$, then S^l contains $\square$.

In the proof of completeness of the resolution method, we will need the following lemma.

Lemma 4. *A formula S is satisfiable if and only if S^l or $S^{\bar{l}}$ is satisfiable.*

Proof. Let $\mathcal{V} \vDash S$ and (without loss of generality) $\bar{l} \in \mathcal{V}$. Then $\mathcal{V} \vDash S^l$, since for clauses C such that $l \notin C \in S$, $\mathcal{V} \vDash C \setminus \{\bar{l}\}$, as $\mathcal{V}$ does not contain $\{\bar{l}\}$ and it is satisfying for each clause $C \in S$.

On the other hand, assume (without loss of generality) $\mathcal{V} \vDash S^l$ for some $\mathcal{V}$. Since neither l nor $\bar{l}$ occur in S^l, $\mathcal{V}' = (\mathcal{V} \setminus \{\bar{l}) \cup \{l\}\} \vDash S^l$. Then $\mathcal{V}' \vDash S$, since for $C \in S$ such that $l \in C$, we have $l \in \mathcal{V}$, and for $C \in S$ not containing l, we have $\mathcal{V}' \vDash (C \setminus \{\bar{l}\}) \in S^l$. □

The reductions of literals can be represented in a binary tree – a so-called reduction tree. The root of the tree is the formula S and each node N has two children – N^l and $N^{\bar{l}}$. In a reduction tree, the formula S is unsatisfiable if and only if every branch contains □.

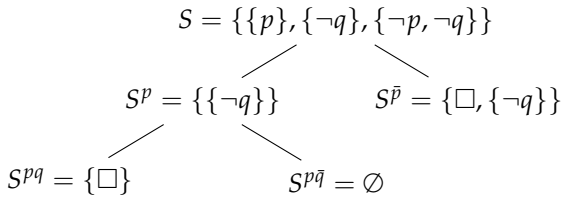

Figure 7: An example of a reduction tree.

Interestingly, since S can be infinite over a countable language, the tree can also be infinite. However, if S is unsatisfiable, according to the compactness theorem, there is a finite $S' \subseteq S$ such that S' is unsatisfiable. Therefore, after the reduction of all literals from S', there will be □ on every branch after finitely many steps.

Finally, we can prove the completeness of resolution. The theorem below shows completeness for finite formulas; the general version is obtained from that theorem by using compactness, similarly to the discussion of the reduction trees above.

Theorem 8 (completeness of resolution)**.** *If a finite S is unsatisfiable, it is resolution refutable, i.e. $S \vdash_R \square$.*

Proof. We will prove the theorem by induction on the number of variables in S. There is only one unsatisfiable S without variables – $\{\square\}$ and therefore $S \vdash_R \square$ (the proof is the single step □).

Let us now assume that S is unsatisfiable and contains a literal l. Then by the previous lemma S^l and $S^{\bar{l}}$ are unsatisfiable. These contain fewer literals than S and therefore by induction there are resolution trees T^l and $T^{\bar{l}}$ deriving □ from S^l and $S^{\bar{l}}$ respectively. Now, if every leaf of T^l is in S, then T^l is a resolution tree of □ from S and therefore $S \vdash_R \square$. Otherwise, we can append the literal l to each leaf of T^l which is not in S and to all of its predecessors, thus obtaining a resolution tree for $\{l\}$ from S (if the original leaf was not in S, the one with added l will be, as the only difference between S^l and S is the removal of l). Similarly, by appending $\{\bar{l}\}$ to leaves in $T^{\bar{l}}$ we obtain a resolution tree for $\{\bar{l}\}$ from S. Resolving the roots of these trees yields a resolution tree of □ from S. □

We have already mentioned that resolution is widely used in different automated systems – SAT solvers, formal verification systems

etc. One of the important examples is a Prolog interpreter. Prolog is a programming language where programs are sets of Horn clauses. The program can answer queries (goals). As Prolog programs are limited to Horn formulas, the resolution method can be improved. The Prolog interpreter uses so-called SLD-resolution which is based on LD-resolution and that is in turn based on LI-resolution, which is a special case of linear resolution. Therefore, we will now define linear resolution, show that LI-resolution is complete for Horn formulas and finally define LD- and SLD-resolution as simple improvements of LI-resolution.

Linear resolution

The general resolution procedure can be further simplified without losing the completeness. We define a linear proof of a clause C from a formula S as a finite sequence of pairs $(C_0, B_0), \ldots, (C_n, B_n)$, such that $C_0 \in S$ and for every $i \leq n$, $B_i \in S$ or $B_i = C_j$ for some $j < i$, and C_{i+1} is a resolvent of C_i and B_i, where $C_{n+1} = C$. In the linear proof C_0 is called the *starting clause*, C_i a *central clause*, and B_i a *side clause*. Again, we say that C is *linearly provable* from S ($S \vdash_L C$), if it has a linear proof from S. A linear proof of $\square$ from S is a *linear refutation* of S and S is *linearly refutable* if $S \vdash \square$.

Obviously, every linear proof can be transformed into a general resolution proof and therefore the linear resolution is also sound. Moreover, is is also complete (we omit the proof of completeness here).

LI-resolution

If we deal with Horn formulas, we can use an even more refined resolution procedure called linear input (LI) resolution. A LI-resolution from a formula S is a linear resolution from S where each side clause B_i is from the input formula S (i.e. B_i cannot be a previously resolved central clause). We write $S \vdash_{LI} C$ to denote that C is provable by LI-resolution from S.

We already defined Horn formulas while discussing the satisfiability problem. The definition from the resolution point of view is similar; the only difference is that we again use the set representation instead of the general one (and also formulas in this case can be infinite, as there is no distinction between theories and formulas in resolution). So a *Horn clause* is a finite set of literals containing at most one positive literal. A Horn formula is then a (potentially infinite) set of Horn clauses. A clause {p}, where p is a positive literal is called a *fact*, and a clause with exactly one positive literal is called a *rule*. Rules and facts are also called *program clauses*. A non-empty Horn clause without any positive literal is called a *goal*.

We can easily see that if a Horn formula S is unsatisfiable and it does not contain $\square$, it must contain some fact and some goal. Why? If S does not contain any fact, it is satisfied by setting all the propositional variables to 0. if it does not contain any goal, it is satisfied by setting all variables to 1.

LI-resolution is complete for Horn formulas, as the following theorem says. The proof of the theorem is similar to the proof of completeness of general resolution.

Theorem 9 (completeness of LI-resolution)**.** *If T is a satisfiable Horn formula but $T \cup \{G\}$ is unsatisfiable for some goal G, then $\square$ has an LI-resolution of from $T \cup \{G\}$ with starting clause G.*

Proof. As in the proof of completeness of general resolution, we use induction on the number of variables, this time in T. By the observation above, T must contain a fact $\{p\}$ for some variable p ($T \cup \{G\}$ is unsatisfiable, therefore it must contain a goal and a fact; G is a goal, so the fact must be in T). By the lemma we used in the proof of completeness of general resolution, $T' = (T \cup \{G\})^p = T^p \cup \{G^p\}$ is unsatisfiable and $G^p = G \setminus \{\neg p\}$. Now, if $G^p = \square$, we have $G = \{\neg p\}$ and thus $\square$ is a resolvent of G and $\{p\} \in T$. Otherwise, since T^p is satisfiable (by the satisfying assignment for T) and has fewer variables, by the induction assumption, there is an LI-resolution of $\square$ form T' starting with G^p. If we now append the literal $\neg p$ to all leaves that are not in $T \cup \{G\}$ (and their predecessors), we have an LI-resolution proof of $\{\neg p\}$ from T; we can resolve it with $\{p\}$ from T to obtain the LI-resolution proof of $\square$ from T. $\square$

Resolution in Prolog

A program in Prolog is a set of program clauses (i.e. rules or facts). An example program is shown below (the program contains seven clauses; the numbers indicate line numbers and are not part of the program):

```
1: p:-q,r.
2: p:-s.
3: q.
4: q:-s.
5: r.
6: s:-t.
7: s.
```

The formulas on lines 3, 5, and 7 are facts, the rest are rules. The symbol `:-` can be interpreted as an implication from right to left ($\leftarrow$). So the meaning of the clauses is as given below:

1. $q \wedge r \rightarrow p$
2. $s \rightarrow p$
3. q
4. $s \rightarrow q$
5. r
6. $t \rightarrow s$
7. s

In Prolog, we want to know whether a query is a consequence of a given program. A query is a conjuction of goals (positive literals), e.g. $p \wedge q$. That means that the question is whether for a program P and query $(q_1 \wedge \cdots \wedge q_n)$ it holds that $P \models (q_1 \wedge \cdots \wedge q_n)$. However, such a question is equivalent to the fact that $P \cup \{\neg q_1, \ldots, \neg q_n\}$ is unsatisfiable, which is equivalent to $\square$ having an LI-resolution from $P \cup \{G\}$ starting with the goal $G = \{\neg q_1, \ldots, \neg q_n\}$.

In the Prolog interpreter, the clauses are represented as sequences of literals (as opposed to sets of literals as in LI-resolution). Therefore, Prolog uses a slightly different version of LI-resolution called LD-resolution (linear definite). In LD-resolution, the resolvent of a goal $(\neg p_1, \ldots, \neg p_{i-1}, \neg p_i, \neg p_{i+1}, \ldots, \neg p_n)$ and a rule $(p_i, \neg q_1, \ldots, \neg q_m)$ is a new goal $(\neg p_1, \ldots, \neg p_{i-1}, \neg q_1, \ldots, \neg q_m, \neg p_{i+1}, \ldots, \neg p_n)$, i.e. one of the negative literals in the current goal is replaced by the negative literals from the rule.

LD-resolution does not specify which of the negative literals in the goal should be resolved next. This would make programming in Prolog hard, therefore it extends the LD resolution with a selection rule $\mathcal{R}$. Typically, the rule is "select the first literal". More formally, an SLD-resolution via $\mathcal{R}$ is an LD-resolution in which we resolve each step (C_i, B_i) through $\mathcal{R}(C_i)$.

Obviously, any LI-resolution can be expressed as an LD-resolution (just use the sequences of literals instead of the sets of literals), and any LD-resolution can be expressed as an SLD-resolution with the correct selection rule (select the literal that was selected in the LD resolution). Therefore, we can see that SLD resolution is complete for Prolog programs.

Further discussion of Prolog is out of the scope of this lecture, so we will omit it here. The important message was to show an application of logic in computer science. Prolog is quite often used in certain areas of artificial intelligence.

This concludes the part of the lecture dedicated to propositional logic. We started with a discussion about the syntax of propositions, explained their semantics and showed some formal proof systems. In the next part of the lecture, we will build on our understanding of concepts from propositional logic and extend them to predicate logic (more precisely to first-order logic).

Part II

First-Order Logic

Basic Syntax and Semantics

We are now ready to discuss predicate logic, mostly first-order logic. The language of predicate logic is more expressive than that of propositional logic and allows us to express complex formulas in a much more concise way. In propositional logic, we often needed to use many variables and created long formulas. In predicate logic, some of these can be written more elegantly, as we can now use functions, relations, and logical quantifiers.

This whole part will follow the structure of the previous one – we will again start with the basic syntax and semantics of predicate logic, then we will discuss logical theories and their models, the tableau method in predicate logic and also the resolution method. While the basic ideas remain the same, there are also important differences. For example, a model of a theory will now be defined as a mathematical structure in which all the axioms of the theory are true, instead of the simpler definition as a truth assignment.

First-order formulas and theories

The symbols used in first-order language can be divided into two groups – symbols of logic and non-logical symbols. The *symbols of logic* consist of *variables* ($x, y, z, \ldots, x_1, x_2, \cdots \in \mathrm{Var}$), logical connectives ($\rightarrow, \wedge, \vee, \leftrightarrow, \neg$), the quantifiers $(\forall x), (\exists x)$ for each variable $x \in \mathrm{Var}$, and parenthesis.

The *non-logical symbols* consist of function symbols $(f, g, \ldots)$, including constant symbols $(c, d, \ldots)$, which are nullary function symbols, and relation (predicate) symbols (P, Q, R). Each function and relation symbol S, has an associated arity $\mathrm{ar}(S) \in \mathbb{N}$ that expresses the number of arguments the symbol takes.

Equality $(=)$ is a special relation symbol that is often considered separately, as it is central to many parts of mathematics and there are even special axioms regarding equality. Equality is also not considered a non-logical symbol.

A language in first-order logic is determined by the sets of function and relation symbols – these are coupled in the so-called *signature*, which is a pair $\langle \mathcal{R}, \mathcal{F} \rangle$ of relation and function symbols with their arities. None of the symbols is the equality symbol. The *language* is then given by a signature $L = \langle \mathcal{R}, \mathcal{F} \rangle$ and by specifying whether the language is with equality or not. A language must always contain at least one relation symbol (either equality or a non-logical one),

otherwise it would not be possible to write formulas in the language.

The meaning of the symbols in the language is not given by logic, i.e. even common symbols such as $+$ or $\leq$ do not need to represent addition or ordering.

There are many languages that are commonly used in mathematics, for example (all the languages are with equality):

1. $L = \langle\rangle$ is the language of pure equality,
2. $L = \langle c_i \rangle_{i \in \mathbb{N}}$ is the language of countably many constants,
3. $L = \langle \leq \rangle$ is the language of orderings,
4. $L = \langle E \rangle$ is the language of graph theory,
5. $L = \langle +, -, 0 \rangle$ it the language of group theory,
6. $L = \langle +, -, \cdot, 0, 1 \rangle$ it the language of field theory,
7. $L = \langle -, \wedge, \vee, 0, 1 \rangle$ is the language of Boolean algebras, and
8. $L = \langle S, +, \cdot, 0, \leq \rangle$ is the language of arithmetic.

In the examples, 0, 1, and c_i are constant symbols, $-$ and S are unary function symbols, $+, \cdot, \wedge, \vee$ are binary function symbols, and E and $\leq$ are binary relation symbols.

The structure of formulas in first-order language is more complex that the structure of propositional formulas. Before we formally define formulas, we first need to define terms and atomic formulas. Informally, terms are expressions created from variables and functions, while atomic formulas are relations applied to terms.

More formally, a *term of a language* L is defined inductively as follows:

1. Every variable $x \in \text{Var}$ or a constant symbol in L is a term.
2. If f is a function symbol in L with arity $n > 0$ and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.
3. Every term is obtained by a finite number of applications of steps 1 and 2 above.

A term without variables is called a *ground term*. The set of all terms of a language L is denoted as Term_L. A term that is a part of another term t is called a *subterm* of t. Terms can also be expressed using formation trees. For binary functions, we often use infix notation, so we write $x + y$ instead of $+(x, y)$.

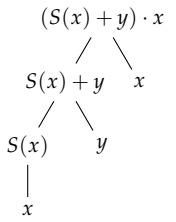

Figure 8: A formation tree of the term $(S(x) + y) \cdot x$.

The simplest type of formulas are the atomic formulas. These are only relations applied to terms. More formally, an *atomic formula of a language* L is an expression $R(t_1, \ldots, t_n)$, where R is a relation symbol in L and $t_1, \ldots, t_n$ are terms of L. The set of all atomic formulas of a language L is denoted as AFm_L. Similarly to terms, an atomic formula can also be represented using a formation tree from the formation trees of its terms, and for binary relations, we use infix notation, e.g. $\leq (x, y)$ can be written as $(x \leq y)$. For example $(x + y) = 0$, or

$R(f(x), g(y,z), x)$ are atomic formulas (f is a unary function, g and $+$ are binary functions, and R is a ternary relation).

We can finally define formulas in a first-order language. The definition is similar to the one in a propositional language, but this time the propositional variables are represented by atomic formulas and we additionally have quantifiers. Formally, a *formula of a language L* is defined inductively as follows:

1. Every atomic formula is a formula.
2. If φ and ψ are formulas, then $(\varphi \to \psi), (\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \leftrightarrow \psi), (\neg\varphi)$ are also formulas.
3. If φ is a formula and $x \in \mathrm{Var}$ is a variable, then $((\forall x)\varphi)$ and $((\exists x)\varphi)$ are formulas.
4. Every formula is obtained by a finite number of applications of the steps above.

The set of all formulas of a language L is denoted by Fm_L. A formula that is a part of another formula φ is a subformula of φ. Of course, formulas can also be expressed using a formation tree. An example is on the right.

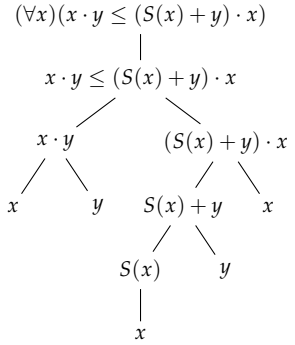

Figure 9: A formation tree of the formula $(\forall x)(x \cdot y \leq (S(x) + y) \cdot x)$. Moreover, $x \cdot y$ and $(S(x) + y) \cdot x$ are roots of formation trees of the terms included in the formula.

As before, we can define some conventions to simplify writing formulas. After introducing priorities of binary function symbols $(+, \cdot, \dots)$ we can omit parentheses in the infix notation of terms that are around a subterm formed by a symbol of higher priority. We also introduce priorities of logical connectives, similar to the priorities in propositional logic. Negation and quantifiers $(\neg, (\forall x), (\exists x))$ have the highest priorities, then we have conjunction and disjunction $(\wedge, \vee)$ and, finally, implication and equivalence $(\to, \leftrightarrow)$ have the lowest priority. Now we can omit some of the parentheses in the formulas.

In predicate logic, there is an important difference between so-called free and bound (occurrences of) variables, as we deal with each type differently in the semantics. For a formula φ and variable x, an *occurrence of x in φ* is a leaf labeled by x in the formation tree of φ. The occurrence of x in φ is *bound* if it is in some subformula ψ that starts with $(\forall x)$ or $(\exists x)$. Otherwise the occurrence is *free*. A *variable is free* in a formula, if it has at least one free occurrence in the formula and it is *bound* if it has at least one bound occurrence. A variable can be both free and bound at the same time. For example, in the formula $x > 0 \vee (\forall x)(\exists y)(x > y)$ the variable x is both free and bound, as its first occurrence is free and the second one is bound. We will use the notation $\varphi(x_1, \dots, x_n)$ to denote that $x_1, \dots, x_n$ are all the free variables in φ.

A formula is *open* if it contains no quantifiers. The set of all open formulas in a language L will be denoted as OFm_L. Obviously, $\mathrm{AFm}_L \subsetneq \mathrm{OFm}_L \subsetneq \mathrm{Fm}_L$. On the other hand, a formula is *closed (a sentence)* if it has no free variables. A formula can be both closed and open at the same time; all terms of such formulas are ground terms.

In mathematics, we very often have general theorems and we later use them with more specific substitutions. This can more formally be

expressed by substituting terms for free variables in formulas. However, we need to be careful, as in some cases the substitution can change the meaning of the formula. For example, if we substituted the term y for x in $(\exists y)(x + y = 1)$, we would change the original meaning of the formula "there is a y such that $y = 1 - x$" to a new meaning that says "y is divisible by 2". We want to avoid such a situation while performing a substitution. Therefore, we define that a term t is *substitutable* for a variable x in a formula φ if, after the substitution of t for all free occurrences of x, none of the variables of t become bound in φ. The new formula is denoted as $\varphi(x/t)$ and we call it an *instance of the formula* φ after a substitution of term t for variable x. Alternatively, we can also define that t is not substitutable for x in φ if x has a free occurrence in a subformula of the form $(\exists y)\psi$ or $(\forall y)\psi$ for some variable y in t.

We can also rename the quantified variables, but we again need to be careful. In this case, we would like to obtain an equivalent formula. Let $(Qx)\psi$ be a subformula of φ where Q is either $\forall$ or $\exists$ and y is a variable. Then, if y is substitutable for x in ψ and y is not free in ψ, we can replace the subformula $(Qx)\psi$ with $(Qy)\psi(x/y)$ to obtain a *variant* of φ in subformula $(Qx)\psi$. A variant of φ is obtained by variation of one or more subformulas of φ.

Informal! Creating variants in predicate logic serves some important purposes. One of them is, that we can easily transform any formula with a variable that is both free and bound into its variant, where each variable is "pure", i.e. only free or only bound. Moreover, we will very often create variants from formulas in order to fulfill assumptions such as "variable x is not free in φ".

Semantics of first-order logic

In propositional logic, models were defined as truth assignments. A truth assignment was enough to tell whether a proposition is true or false. In predicate logic, the situation is a bit more complex. First of all, the values of variables can be taken from a larger set than only $\{0, 1\}$. Moreover, we need to define what all the functions and relations mean. A natural representation of models in first-order logic is a mathematical structure. A structure is a set and a definition of functions and relations on this set.

More formally, if we have a signature of a language $L = \langle \mathcal{R}, \mathcal{F} \rangle$ and a non-empty set A, a *realization (interpretation) of a relation symbol* $R \in \mathcal{R}$ on the set A is any relation $R^A \subseteq A^{\mathrm{ar}(R)}$. A realization of $=$ is the identity relation on A, i.e. $\mathrm{Id}_A = \{(x, x) | x \in A\}$. *A realization (interpretation) of a function symbol* $f \in \mathcal{F}$ is any function $f^A : A^{\mathrm{ar}(f)} \to A$. Specifically, a realization of a constant symbol is some element of A. A *structure for the language* L (L-structure) is a triple $\mathcal{A} = \langle A, \mathcal{R}^A, \mathcal{F}^A \rangle$, where A is a non-empty set called the domain of the structure $\mathcal{A}$, $\mathcal{R}^A$ is a collection of realizations of the relation symbols on A, and $\mathcal{F}^A$ is a collection of realizations of function symbols on A. A structure of

the language is also called a model of the language, and the class of all models of a language L will be denoted as $M(L)$.

You probably already know different mathematical structures from other parts of mathematics, for example:

1. $S = \langle S, \leq \rangle$ is an ordered set, where $\leq$ is reflexive, antisymmetric, and transitive binary relation,
2. $G = \langle V, E \rangle$ is a graph,
3. $\mathbb{Z}_p = \langle \mathbb{Z}_p, +, -, 0 \rangle$ is the additive group of integers modulo p,
4. $\mathbb{Q} = \langle \mathbb{Q}, +, -, 0, 1 \rangle$ is the field of rational numbers,
5. $\mathcal{P}(X) = \langle \mathcal{P}(X), \setminus, \cap, \cup, \emptyset, X \rangle$ is the set algebra over X, and
6. $\mathbb{N} = \langle \mathbb{N}, S, +, \cdot, 0, \leq \rangle$ is the standard model of arithmetic.

But also many other objects can be defined as structures, e.g. finite automata or even databases.

We now aim to define the truth value of formulas in first-order logic. We already know, that a formula is constructed from atomic formulas, which are in turn constructed from terms. Therefore, in order to define the truth value of a formula, we need to start with the definition of the value of a term. Let t be a term of $L = \langle \mathcal{R}, \mathcal{F} \rangle$ and $\mathcal{A} = \langle A, \mathcal{R}^A, \mathcal{F}^A \rangle$ an L-structure. A *variable assignment* over the domain A is a function $e : \mathrm{Var} \to A$. The *value* $t^A[e]$ *of the term* t in the structure $\mathcal{A}$ with respect to the assignment e is defined inductively by

1. $x^A[e] = e(x)$, for $x \in \mathrm{Var}$,
2. $(f(t_1, \ldots, t_n))^A[e] = f^A(t_1^A[e], \ldots, t_n^A[e])$ for $f \in \mathcal{F}$.

For a constant symbol $c^A[e] = c^A$, i.e. the values of constants do not depend on the assignment e, and therefore also the value of ground terms does not depend on the assignment. Obviously, the value of a term t depends only on the assignment of variables in t.

We now know, how to compute the values of individual terms, therefore, we can define the value of atomic formulas. Contrary to the values of terms, values of formulas are always from the set $\{0, 1\}$. Let φ be an atomic formula of $L = \langle \mathcal{R}, \mathcal{F} \rangle$ in the form $R(t_1, \ldots, t_n)$, $\mathcal{A} = \langle A, \mathcal{R}^A, \mathcal{F}^A \rangle$ is an L-structure and e a variable assignment over A. The *value* $H_{at}^A(\varphi)[e]$ *of the atomic formula* φ in the structure $\mathcal{A}$ with respect to e is

$$H_{at}^A(\varphi)[e] = \begin{cases} 1 & \text{if } (t_1^A[e], \ldots t_n^A[e]) \in R^A \\ 0 & \text{otherwise} \end{cases}$$

Specifically, for the equality relation $=$, the only possible realization is Id_A and therefore $H_{at}^A(t_1 = t_2)[e] = 1$, if $t_1^A[e] = t_2^A[e]$ and 0 otherwise. We can again see that the value of a formula depends only on the assignment of variables in the formula and that the value of a ground formula does not depend on the assignment at all.

We can finally define the value of a general formula. The definition is quite long, but also very similar to the one in propositional logic. In

fact, the only difference is in the last two cases. The atomic formulas in this case play the role of the propositional variables.

The value $H^A(\varphi)[e]$ of formula φ in the structure A with respect to e is

$$\begin{aligned}
H^A(\varphi)[e] &= H^A_{at}(\varphi)[e] \text{ if } \varphi \text{ is atomic} \\
H^A(\neg\varphi)[e] &= -_1(H^A(\varphi)[e]) \\
H^A(\varphi \wedge \psi)[e] &= \wedge_1(H^A(\varphi)[e], H^A(\psi)[e]) \\
H^A(\varphi \vee \psi)[e] &= \vee_1(H^A(\varphi)[e], H^A(\psi)[e]) \\
H^A(\varphi \to \psi)[e] &= \to_1 (H^A(\varphi)[e], H^A(\psi)[e]) \\
H^A(\varphi \leftrightarrow \psi)[e] &= \leftrightarrow_1 (H^A(\varphi)[e], H^A(\psi)[e]) \\
H^A((\forall x)\varphi)[e] &= \min_{a \in A}(H^A(\varphi)[e(x/a)]) \\
H^A((\exists x)\varphi)[e] &= \max_{a \in A}(H^A(\varphi)[e(x/a)])
\end{aligned}$$

where $-_1, \wedge_1, \vee_1, \to_1, \leftrightarrow_1$ are the functions given by the truth tables in the part on propositional logic and $e(x/a)$ is an assignment assigning value a to variable x and otherwise identical to e. We can see that the value of a formula depends only on the assignment of free variables in the formula (we check all possible assignments for the bound variables in steps 7 and 8).

A structure $\mathcal{A}$ satisfies the formula φ if $H^A(\varphi) = 1$; we denote this fact as $\mathcal{A} \vDash \varphi[e]$, otherwise we write $\mathcal{A} \nvDash \varphi[e]$. We can easily check that all the following hold

$$\begin{aligned}
\mathcal{A} \vDash \neg\varphi[e] &\Leftrightarrow \mathcal{A} \nvDash \varphi[e] \\
\mathcal{A} \vDash (\varphi \wedge \psi)[e] &\Leftrightarrow \mathcal{A} \vDash \varphi[e] \text{ and } \mathcal{A} \vDash \psi[e] \\
\mathcal{A} \vDash (\varphi \vee \psi)[e] &\Leftrightarrow \mathcal{A} \vDash \varphi[e] \text{ or } \mathcal{A} \vDash \psi[e] \\
\mathcal{A} \vDash (\varphi \to \psi)[e] &\Leftrightarrow \mathcal{A} \vDash \varphi[e] \text{ implies } \mathcal{A} \vDash \psi[e] \\
\mathcal{A} \vDash (\varphi \leftrightarrow \psi)[e] &\Leftrightarrow \mathcal{A} \vDash \varphi[e] \text{ if and only if } \mathcal{A} \vDash \psi[e] \\
\mathcal{A} \vDash (\forall x)(\varphi)[e] &\Leftrightarrow \mathcal{A} \vDash \varphi[e(x/a)] \text{ for every } a \in A \\
\mathcal{A} \vDash (\exists x)(\varphi)[e] &\Leftrightarrow \mathcal{A} \vDash \varphi[e(x/a)] \text{ for some } a \in A
\end{aligned}$$

Furthermore, if t is substitutable for x in φ, then for every structure $\mathcal{A}$ and assignment e, $\mathcal{A} \vDash \varphi(x/t)[e]$ if and only if $\mathcal{A} \vDash \varphi[e(x/a)]$, where $a = t^A[e]$. If ψ is a variant of φ then $\mathcal{A} \vDash \varphi[e]$ if and only if $\mathcal{A} \vDash \psi[e]$.

As in propositional logic, we can generalize the notion above to validity in structure and in theory. Let φ be a formula of a language L, and $\mathcal{A}$ an L-structure. We say, that φ *is valid in* $\mathcal{A}$, denoted as $\mathcal{A} \vDash \varphi$, if $\mathcal{A} \vDash \varphi[e]$ for every $e : \text{Var} \to A$. We also say that $\mathcal{A}$ *satisfies* φ. Otherwise, we write $\mathcal{A} \nvDash \varphi$. The formula φ *is contradictory in* $\mathcal{A}$ if $\mathcal{A} \vDash \neg\varphi$, i.e. if $\mathcal{A} \nvDash \varphi[e]$ for every e.

We can easily check that for any structure $\mathcal{A}$ and formulas φ, ψ the following hold:

$$\mathcal{A} \vDash \varphi \Rightarrow \mathcal{A} \nvDash \neg\varphi \tag{1}$$

$$\mathcal{A} \vDash \varphi \wedge \psi \Leftrightarrow \mathcal{A} \vDash \varphi \text{ and } \mathcal{A} \vDash \psi \tag{2}$$

$$\mathcal{A} \vDash \varphi \vee \psi \Leftarrow \mathcal{A} \vDash \varphi \text{ or } \mathcal{A} \vDash \psi \tag{3}$$

$$\mathcal{A} \vDash \varphi \Leftrightarrow \mathcal{A} \vDash (\forall x)\varphi \tag{4}$$

Moreover, if φ and ψ are sentences, the implications in (1) and (3) are in fact equivalences. The last equivalence (4) also shows that $\mathcal{A} \vDash \varphi$ if and only if $\mathcal{A} \vDash \psi$, where ψ is the *universal closure* of φ, i.e. the formula $(\forall x_1)(\forall x_1) \dots (\forall x_n)\varphi$, where $x_1, \dots, x_n$ are all the free variables of φ.

A *theory* of a language L is any set T of formulas of L (the *axioms* of the theory). A *model of a theory* T is an L-structure $\mathcal{A}$ such that $\mathcal{A} \vDash \varphi$ for every $\varphi \in T$. We also write $\mathcal{A} \vDash T$ and say that $\mathcal{A}$ satisfies T. The *class of all models* of theory T is $M(T) = \{\mathcal{A} \in M(L) | \mathcal{A} \vDash T\}$. A formula is *valid in* T (true in T) ($T \vDash \varphi$) if $\mathcal{A} \vDash \varphi$ for every model $\mathcal{A}$ of T. Otherwise we write $T \nvDash \varphi$. A formula φ *is contradictory in* T if $T \vDash \neg\varphi$ and φ *is independent in* T if it is neither valid nor contradictory in T. For the empty theory T, we can omit T in the notation and $M(T) = M(L)$. In this case, ($\vDash \varphi$) means that the formula φ is *logically valid* (*a tautology*). A *consequence of* T is the set $\theta^L(T)$ of all sentences of L valid in T, i.e.

$$\theta^L(T) = \{\varphi \in \mathrm{Fm}_L \mid T \vDash \varphi \text{ and } \varphi \text{ is a sentence}\}.$$

Informal! The definitions above closely resemble those we saw in propositional logic; the main difference is in the definition of the model. In propositional logic, we could use truth assignments, while in predicate logic, the model is a structure. The structure, and its definitions of functions and relations in fact give the truth values to the atomic formulas. The atomic formulas then play the role of the propositional variables. Of course, in predicate logic, we also need to take care of the quantifiers, which brings another complexity to the definitions.

For example, the theory of orderings T is a theory in language $L = \langle \leq \rangle$ with axioms

$$\begin{aligned} x \leq x & \qquad \text{reflexivity,} \\ x \leq y \wedge y \leq x \to x = y & \qquad \text{antisymmetry,} \\ x \leq y \wedge y \leq z \to x \leq z & \qquad \text{transitivity.} \end{aligned}$$

The models of T (ordered sets) are structures $\langle S, \leq_S \rangle$. For example $\mathcal{A} = \langle \mathbb{N}, \leq \rangle$ or $\mathcal{B} = \langle \mathcal{P}(X), \subseteq \rangle$ for a set $X = \{0, 1, 2\}$. The formula $\varphi \equiv x \leq y \vee y \leq x$ is valid in $\mathcal{A}$, but not in $\mathcal{B}$, as $\mathcal{B} \nvDash \varphi[e]$ for $e(x) = \{0\}$ and $e(y) = \{1\}$, therefore it is independent in T. The sentence $\psi \equiv (\exists x)(\forall y)(y \leq x)$ is valid in $\mathcal{B}$ and contradictory in $\mathcal{A}$, and therefore also independent in T. Finally, the formula $\chi \equiv (x \leq y \wedge y \leq z \wedge z \leq x) \to (x = y \wedge y = z)$ is valid in T, denoted as $T \vDash \chi$.

We say that a theory T in a language L is *semantically inconsistent* if $T \vDash \bot$, otherwise T *is consistent*. A theory T *is complete*, if it is consistent, and every sentence of L is either valid or contradictory in T. An L-theory T *is an extension* of another theory T' in language L' if $L' \subseteq L$ and $\theta^{L'}(T') \subseteq \theta^L(T)$. The *extension is simple*, if $L' = L$ and it is *conservative* if $\theta^{L'}(T') = \theta^L(T) \cap \mathrm{Fm}_{L'}$. *Two theories are equivalent*, if one is an extension of the other and vice versa.

We also define a form of equivalence for two structures – we say that two L-structures $\mathcal{A}, \mathcal{B}$ are *elementarily equivalent*, denoted as $\mathcal{A} \equiv \mathcal{B}$ if

they satisfy the same sentences of L. It means that we cannot write a sentence in the language that would make any distinction between the two structures. Later, we will also define the isomorphism of structures and we will see that it is a stronger property, i.e. that any two isomorphic structures are elementarily equivalent, but not vice versa.

The definitions above lead to a simple observation – a theory T of language L is consistent if it has a model. It is complete if and only if it has a single model, up to elementary equivalence. A theory T is an extension of another theory T' in the same language L if and only if $M(T) \subseteq M(T')$ and the theories are equivalent if $M(T) = M(T')$.

We can transform the problem of validity in a theory into the problem of satisfiability of another theory, similarly to a proof by contradiction. For every theory T and sentence φ (in the same language), it holds $T \cup \{\neg\varphi\}$ is unsatisfiable if and only if $T \models \varphi$. Why? Because by the definitions, $T \cup \{\neg\varphi\}$ is unsatisfiable (i.e. has no model) if $\neg\varphi$ is not valid in any model of T, which means (and here we need the assumption that φ is a sentence) that φ is valid in every model of T and that in turn means $T \models \varphi$.

You also probably know the notion of substructure from other parts of mathematics. Let $\mathcal{A} = \langle A, \mathcal{R}^A, \mathcal{F}^A \rangle$ and $\mathcal{B} = \langle B, \mathcal{R}^B, \mathcal{F}^B \rangle$ be structures for $L = \langle \mathcal{R}, \mathcal{F} \rangle$. We say that $\mathcal{B}$ *is an (induced) substructure of* $\mathcal{A}$ ($\mathcal{B} \subseteq \mathcal{A}$) if $B \subseteq A$, $R^B = R^A \cap B^{\mathrm{ar}(R)}$ for every $R \in \mathcal{R}$, and $f^B = f^A \cap (B^{\mathrm{ar}(f)} \times B)$ for every $f \in \mathcal{F}$. A set C is a domain of some substructure of $\mathcal{A}$ if and only if it is closed under all functions of $\mathcal{A}$.[15] The representative substructure is then called a *restriction of* $\mathcal{A}$ *to* C and denotes as $\mathcal{A} \restriction C$.

[15] A set $C \subseteq A$ is closed under a function $f : A^{\mathrm{ar}(f)} \to A$, if $f(x_1, \ldots, x_{ar(f)}) \in C$ for all $x_1, \ldots, x_{\mathrm{ar}(f)} \in C$.

If we have a structure $\mathcal{A}$, its substructure $\mathcal{B}$ in a language L and a value assignment $e : \mathrm{Var} \to B$, then, obviously, for an open formula φ, $\mathcal{A} \models \varphi[e]$ if and only if $\mathcal{B} \models \varphi[e]$. The essential fact here is that e assigns only values from B and therefore the values of the terms are the same in both $\mathcal{A}$ and $\mathcal{B}$. The same then holds for the atomic formulas and by induction on the complexity of the formula also for general formulas. This simple observation however has an interesting corollary. For every open formula φ and a structure $\mathcal{A}$, the formula is valid in the structure $\mathcal{A}$ if and only if it is valid in every substructure $\mathcal{B}$ of $\mathcal{A}$. If a theory T contains only open axioms (a so-called *open theory*), this implies that every substructure of a model of T is also a model of T.

The last observation is important in case we want to check whether a theory T is openly axiomatizable, i.e. whether there is an equivalent theory T' that contains only open axioms. In such a case, we just need to check if every substructure of a model of the theory T is also a model of T. If it is, then T is openly axiomatizable, otherwise it is not.

Let $\mathcal{A} = \langle A, \mathcal{R}^A, \mathcal{F}^A \rangle$ be a structure and $X \subseteq A$. Let B be the smallest subset of A containing X and closed under all functions of $\mathcal{A}$ (including constants). Then the structure $\mathcal{A} \restriction B$ is denoted as $\mathcal{A}\langle X \rangle$ and is called *a substructure of* $\mathcal{A}$ *generated by the set* X.

Let $\mathcal{A}$ be a structure for the language L and $L' \subseteq L$. By omitting realizations of symbols that are not in L' we obtain a structure $\mathcal{A}'$ in L'

called the *reduct of $\mathcal{A}$ to the language L'*. The structure $\mathcal{A}$ is then called the expansion of $\mathcal{A}'$ into L.

The tableau method in first-order logic

As in propositional logic, we will use the tableau method to prove formulas in first order logic. While most of the ideas from propositional logic will be used again, the more complex structure of first-order formulas leads to additional rules in the tableau method (these are basically the atomic tableaux for the quantifiers). Also, the proofs of soundness and completeness will get more technical because we now need to also consider the structure of the terms, and because the models in first-order logic are mathematical structures rather than truth assignments. However, we will define the so-called canonical structure as a general structure for a language that prescribes the definitions of all function symbols; then we will again need to define only the truth values of atomic formulas, thus reducing most of the ideas to propositional logic.

In fact, propositional logic can be seen as a fragment of predicate logic, where we do not use the quantifiers and the propositional variables are represented by nullary predicate symbols.

In tableau proofs in predicate logic, we will work in a language that contains a countable number of new constants. Therefore it is important to show that adding new constants to the language does not change the validity of formulas in the language. This should be intuitive – adding constants without saying anything about them means that we only add symbols that can be used as names of specific elements in the structure. This is more formally demonstrated by the following theorem.

Theorem 10. *Let φ be a formula in a language L with free variables $x_1, \ldots, x_n$ and let T be a theory in L. Let L' be an extension of L with new constant symbols $c_1, \ldots, c_n$ and let T' denote the theory T in L'. Then*

$$T \vDash \varphi \text{ if and only if } T' \vDash \varphi(x_1/c_1, \ldots, x_n/c_n)\,.$$

Proof. ($\Rightarrow$) If $\mathcal{A}'$ is a model of T', let $\mathcal{A}$ be the reduct of $\mathcal{A}'$ to L. Since $\mathcal{A} \vDash \varphi[e]$ for every assignment e, we have $\mathcal{A} \vDash \varphi[e(x_1/c_1^{\mathcal{A}'}, \ldots, x_n/c_n^{\mathcal{A}'})]$, i.e. $\mathcal{A}' \vDash \varphi(x_1/c_1, \ldots, x_n/c_n)$.

($\Leftarrow$) For the other implication, if $\mathcal{A}$ is a model of T and e an assignment, let $\mathcal{A}'$ be the expansion of $\mathcal{A}$ into L' by setting $c_i^{\mathcal{A}'} = e(x_i)$ for every i. Since $\mathcal{A}' \vDash \varphi(x_1/c_1, \ldots, x_n/c_n)[e']$ for every assignment e', we have $\mathcal{A}' \vDash \varphi[e(x_1/c_1^{\mathcal{A}'}, \ldots, x_n/c_n^{\mathcal{A}'})]$, i.e. $\mathcal{A} \vDash \varphi[e]$. □

The basics of the tableau method in predicate logic are similar to the tableau method in propositional logic. A tableau is still a binary

tree that represents a search for a counterexample. The nodes are still labeled with entries, i.e. formulas with a sign T or F. However, this time, the formulas will be sentences. A branch is still contradictory if it contains $T\varphi$ and $F\varphi$ for a formula φ. A proof of a formula φ is still a contradictory tableau with $F\varphi$ as its root entry. If a counterexample exists, there will be a non-contradictory branch in the finished tableau that provides us with the counterexample. We will again define a systematic tableau that is always finished and in case it is a proof of a formula, it is also finite.

There are however some differences – we need to add atomic tableaux for the logical quantifiers. In these the quantified variables will be substituted with ground terms following some rules. We extend the language by new constant symbols that act as witnesses of the entries $T(\exists x)\varphi x$ and $F(\forall x)\varphi(x)$. In a finished non-contradictory branch for an entry $T(\forall x)\varphi(x)$, we will have the entries $T\varphi(x/t)$ for every ground term t of the extended language, and similarly for $F(\forall x)\varphi(x)$ and $F\varphi(x/t)$.

We have some assumptions in the tableau method in predicate logic. First of all, the formula we want to prove needs to be a sentence. This is not a problem – we can always prove the universal closure of a formula if it has free variables, as we already know that these two are equivalent. Furthermore, we also assume that the axioms of the theory are sentences – but that is also not a problem: we can again take the universal closures of the axioms. We also assume that the language L is countable. That also means that every theory in L is countable. We define L_C as the extension of L by new constant symbols $c_0, c_1, \ldots$. Then there are countably many ground terms of L_C. Let t_i denote the i-th ground term in some fixed enumeration. Finally, we start with the assumption that L is without equality, but we will deal with this one later.

$T\alpha$	$F\alpha$	$T(\varphi \wedge \psi)$ $T\varphi$ $T\psi$	$F(\varphi \wedge \psi)$ $F\varphi \quad F\psi$	$T(\varphi \vee \psi)$ $T\varphi \quad T\psi$	$F(\varphi \vee \psi)$ $F\varphi$ $F\psi$
$T(\neg\varphi)$ $F\varphi$	$F(\neg\varphi)$ $T\varphi$	$T(\varphi \to \psi)$ $F\varphi \quad T\psi$	$F(\varphi \to \psi)$ $T\varphi$ $F\psi$	$T(\varphi \leftrightarrow \psi)$ $T\varphi \quad F\varphi$ $T\psi \quad F\psi$	$F(\varphi \leftrightarrow \psi)$ $T\varphi \quad F\varphi$ $F\psi \quad T\psi$

Figure 10: The atomic tableaux for logical connectives. In the tableau, φ, ψ are sentences and α are atomic sentences.

The tableaux will again be constructed from atomic tableaux. In predicate logic, we still have the atomic tableaux for the logical connectives ($\vee, \wedge, \neg, \to, \leftrightarrow$). These are essentially the same as in propositional logic, but instead of having tableaux for propositional variables, we have them for atomic sentences α. These atomic tableaux are shown

in Figure 10. In the tableaux, φ and ψ denote formulas in L_C, and α denotes an atomic sentence in the same language.

Additionally, we also have atomic tableaux for the quantifiers. These are shown in Figure 11. Again, φ represents a formula of the language L_C with a free variable x, t is any ground term of L_C and c is a new constant symbol from $L_C \setminus L$.

$T(\forall x)\varphi(x)$	$F(\forall x)\varphi(x)$	$T(\exists x)\varphi(x)$	$F(\exists x)\varphi(x)$
$\vert$	$\vert$	$\vert$	$\vert$
$T\varphi(x/t)$	$F\varphi(x/c)$	$T\varphi(x/c)$	$F\varphi(x/t)$
for any term t	for a new constant c	for a new constant c	for any term t

Figure 11: The atomic tableaux for quantifiers

The constant symbol c represents a witness for the entry $T(\exists x)\varphi(x)$ or $F(\forall x)\varphi(x)$. These symbols must be new, i.e. not used anywhere else on the same branch in the tableau and also cannot be from the language L, as we do not want to assume anything about their value.

A *tableau from a theory* T is again a sequence $\tau_0, \tau_1, \ldots$ of finite tableaux from T, such that τ_{i+1} is formed from τ_i by steps 2 or 3 bellow, formally $\tau = \cup\tau_n$.

A *finite tableau from a theory* T is a binary tree labeled with entries defined inductively as follows:

1. Every atomic tableau from T is a finite tableau from T; in the cases $F(\forall x)\varphi(x)$ and $T(\exists x)\varphi(x)$ we may use any constant symbol $c \in L_C \setminus L$.
2. If P is an entry on a branch V in a finite tableau from T then by adjoining the atomic tableau from P at the end of the branch V we obtain a finite tableau from T. In the cases $F(\forall x)\varphi(x)$ and $T(\exists x)\varphi(x)$ we may only use constant symbols $c \in L_C \setminus L$ that do not appear on V.
3. If V is a branch in a finite tableau from T and $\varphi \in T$, then by adjoining $T\varphi$ at the end of V we obtain a finite tableau from T.
4. Every finite tableau is formed by finitely many applications of the steps above.

Similarly to the tableau method in propositional logic, we do not need to repeat an entry that is expanded again on a branch. However, we have to repeat it in the cases where the entry is $T(\forall x)\varphi(x)$, or $F(\exists x)\varphi(x)$. This convention is demonstrated in the tableaux in Figure 12.

Informal! The cases where we need to repeat the entry are those where we can choose any term in the atomic tableau and if we chose incorrectly, we want to have another attempt to guess correctly. By repeating the entry on the branch, we have another non-reduced entry of the same type that we will reduce later.

A *branch V is contradictory* if it contains entries $T\varphi$ and $F\varphi$ for some sentence φ, otherwise it is noncontradictory. A tableau τ is contradictory if all its branches are contradictory. A *tableau proof of a sentence* φ

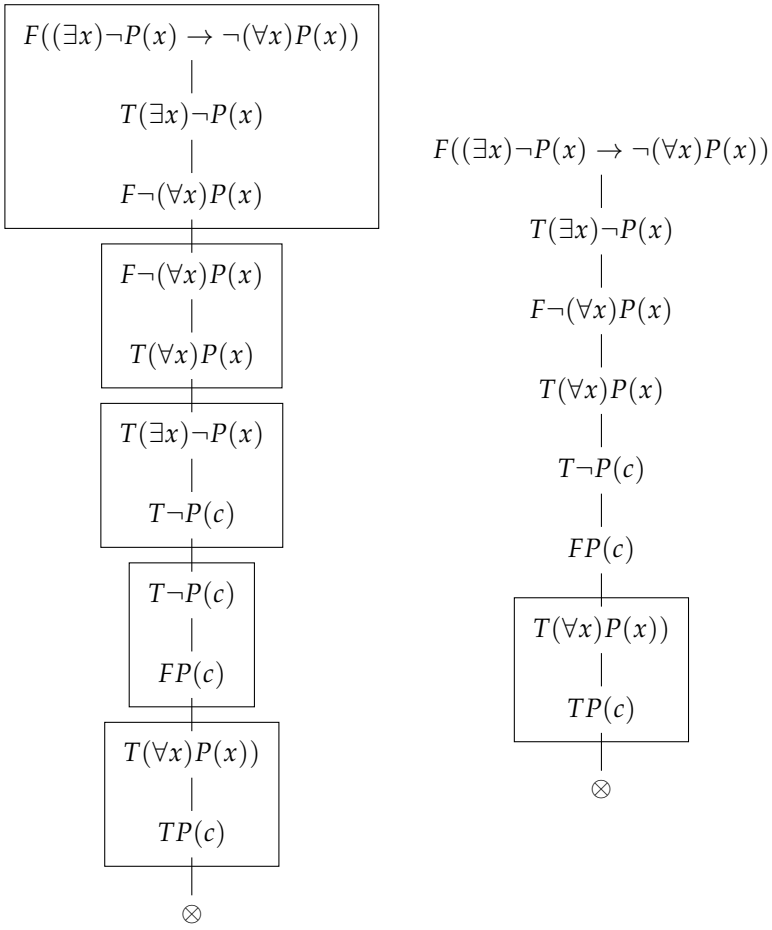

Figure 12: Example tableau. The rectangles on the left show the atomic tableaux used. The version on the right removes the repeated entries that can be removed; the entry in the rectangle in the right tableau must be repeated. c is a new constant symbol where it first appears in the tableau, and in the last step, we chose c as the term in the atomic tableau for $T(\forall x)P(x)$. The symbol $\otimes$ denotes a contradictory branch.

from T is a contradictory tableau from T with $F\varphi$ in the root. $T \vdash \varphi$ denotes that φ is tableau provable from T. A *refutation of a sentence* φ *by a tableau from* T is a contradictory tableau from T with $T\varphi$ in the root. A sentence is tableau refutable if there is a tableau refutation of the sentence.

Compared to the propositional version of the tableau method, the definition of a finished tableau is a bit more complicated – we need to account for the cases where we need to guess the correct terms and such atomic tableaux must be in the tableau for all the ground terms of L_C. This is reflected in the definition of a reduced entry. An occurrence on an entry P in a node v of a tableau τ is i-th if v has exactly $i-1$ predecessors labeled by P. The *occurrence of* P *is reduced* if P is neither of the form $T(\forall x)\varphi(x)$ nor $F(\exists x)\varphi(x)$ and P occurs in V as the root of an atomic tableau (it was already expanded on V); or if P is in the form $T(\forall x)\varphi(x)$ or $F(\exists x)\varphi(x)$, P has an $(i+1)$-th occurrence on V and V contains an entry $T\varphi(x/t_i)$ or $F\varphi(x/t_i)$, where t_i is the i-th ground term in L_C (in some enumeration of ground terms). Now, let V be a branch in a tableau τ from a theory T. We say that V is finished if it is contradictory, or every occurrence of an entry on V is reduced on V, and moreover V contains $T\varphi$ for every $\varphi \in T$. A tableau τ is finished if every branch in τ is finished.

We can now define systematic tableau in predicate logic. A system-

atic tableau is again always finished and, moreover, if it is a proof, it is finite, as we shall see later. Let R be an entry and $T = \{\varphi_0, \varphi_1, \dots\}$ a theory. The systematic tableau for R from T is the result $\tau = \cup\tau_n$ of the following construction:

1. τ_0 is the atomic tableau for R. If R is of the form $T(\exists x)\varphi(x)$ or $F(\forall x)\varphi(x)$ we take c_0 as the new constant. If R is of the form $T(\forall x)\varphi(x)$ or $F(\exists x)\varphi(x)$ we take t_1 as the term.

2. Let v be the leftmost node in the smallest possible level in tableau τ_n containing an occurrence of an entry P that is not reduced on some noncontradictory branch through v.

3. If P is neither $T(\forall x)\varphi(x)$ nor $F(\exists x)\varphi(x)$, let τ'_n be the tableau obtained from τ_n by adjoining the atomic tableau for P to every noncontradictory branch through v. If P is of the form $T(\exists x)\varphi(x)$ or $F(\forall x)\varphi(x)$, we take c_i as the new constant for the lowest possible i.

4. If P is either $T(\forall x)\varphi(x)$ or $F(\exists x)\varphi(x)$ and it has the i-th occurrence in v, let τ'_n be the tableau obtained from τ_n by adjoining the atomic tableau for P to every noncontradictory branch through v, where we take the term t_i for t.

5. Let t_{n+1} be the tableau obtained from τ'_n by adjoining $T\varphi_n$ to every noncontradictory branch that does not contain $T\varphi_n$ yet.

An example of a systematic tableau is shown in Figure 13.

As in propositional logic, every systematic tableau is finished. We can show this using the same method as in propositional logic. Let $\tau = \cup\tau_n$ be a systematic tableau from $T = \{\varphi_0, \varphi_1, \dots\}$ with a root entry R and let P be an entry in a node v of the tableau τ. There are only finitely many entries in levels above v, therefore if the occurrence of P in v was unreduced, it would be eventually found in step 2, and reduced in steps 3 or 4 of the construction above. Step 4 above ensures that for every $\varphi_n \in T$, $T\varphi_n$ is in the tableau no later than in τ_{n+1} on every noncontradictory branch. Therefore, all the branches in the tableau are finished.

We can also easily show that if a systematic tableau τ is a proof (from a theory T), it is finite. Assume that τ is infinite, then by König's lemma it contains an infinite branch. But this branch is noncontradictory, since we prolong only noncontradictory branches in the construction. But that contradicts τ being a proof, as proofs are contradictory tableaux.

Up until now, we have discussed the tableau method only in languages without equality. However, the principles are the same in languages with equality; we just need to add the equality axioms into the theory. The *equality axioms for language* L are

1. $x = x$,

2. $x_1 = y_1 \wedge \dots \wedge x_n = y_n \to f(x_1, \dots, x_n) = f(y_1, \dots y_n)$ for every n-ary function symbol f in L, and

3. $x_1 = y_1 \wedge \dots \wedge x_n = y_n \to (R(x_1, \dots, x_n) \to R(y_1, \dots y_n))$ for every n-ary relation symbol R in L (including "=").

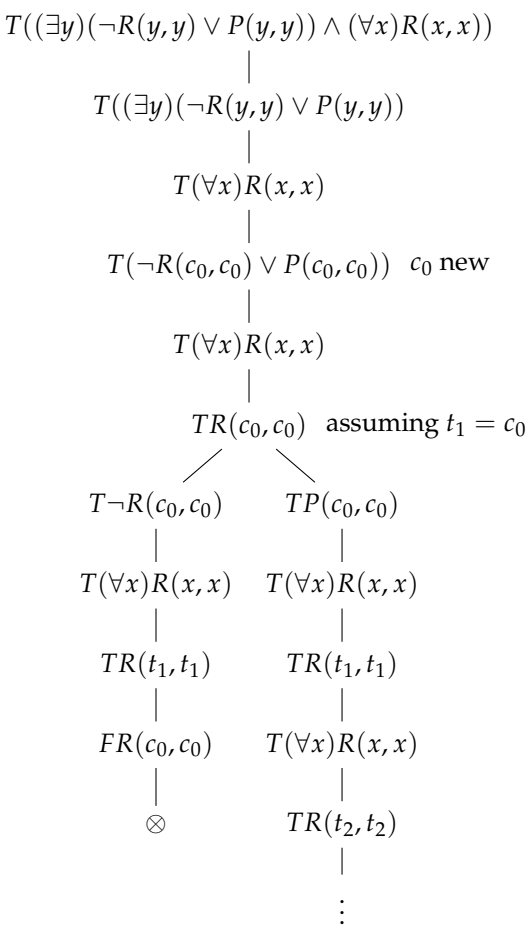

Figure 13: An example of a systematic tableau. The left branch is contradictory, while the right one is noncontradictory and finished but infinite.

The tableau proof from a theory T in language L with equality is a tableau proof from T^*, where T^* denotes the extension of T by adding the axioms of equality for L.

The problem is that the extended theory T^* can have models where equality is represented by a relation $=^A$ which is different from identity. This can be solved by considering the quotient structures by $=^A$ of these models. Let $\sim$ be an equivalence on A, $f : A \to A^n$, and $R \subseteq A^n$ for $n \in \mathbb{N}$. Then $\sim$ is a *congruence for the function* f if for every $x_1, \ldots, x_n, y_1, \ldots, y_n \in A : x_1 \sim x_2 \wedge \cdots \wedge x_n \sim y_n \Rightarrow f(x_1, \ldots, x_n) \sim =f(y_1, \ldots, y_n)$, and it is a *congruence for the relation* R if for every $x_1, \ldots, x_n, y_1, \ldots, y_n \in A : x_1 \sim x_2 \wedge \cdots \wedge x_n \sim y_n \Rightarrow R(x_1, \ldots, x_n) \Leftrightarrow R(y_1, \ldots, y_n)$.

Let an equivalence $\sim$ on A be a congruence for every function and relation in a structure $\mathcal{A} = \langle A, \mathcal{F}^A, \mathcal{R}^A \rangle$ of language $L = \langle \mathcal{F}, \mathcal{R} \rangle$. The *quotient structure of* $\mathcal{A}$ *by* $\sim$ is the structure ${}^{\mathcal{A}}/_{\sim} = \langle {}^{A}/_{\sim}, {\mathcal{F}^A}^{/_\sim}, {\mathcal{R}^A}^{/_\sim} \rangle$ where

$$f^{A/\sim}([x_1]_\sim, \ldots [x_n]_\sim) = [f^A(x_1, \ldots, x_n)]_\sim$$
$$R^{A/\sim}([x_1]_\sim, \ldots, [x_n]_\sim) \Leftrightarrow R^A(x_1, \ldots, x_n)$$

for each $f \in \mathcal{F}, R \in \mathcal{R}$, and $x_1, \ldots, x_n \in A$.

Axioms 1 and 3 of equality ensure that any relation $=^A$ that satisfies them is an equivalence, and axioms 2 and 3 ensure that the relation is

also a congruence. If we have a model $\mathcal{A} \vDash T^*$ then also $({}^{\mathcal{A}}/_{=_A}) \vDash T^*$. Moreover, equality is interpreted as identity in ${}^{\mathcal{A}}/_{=_A}$.

We can now prove the soundness of the tableau method in predicate logic. The proof again closely resembles the one from propositional logic. We will in fact also use a similar lemma. We say that a model $\mathcal{A}$ agrees with an entry P in a tableau if P is $T\varphi$ and $\mathcal{A} \vDash \varphi$ or if P is $F\varphi$ and $\mathcal{A} \vDash \neg\varphi$, i.e. $\mathcal{A} \nvDash \varphi$. Moreover, $\mathcal{A}$ agrees with a branch V if $\mathcal{A}$ agrees with every entry on V.

Lemma 5. *Let $\mathcal{A}$ be a model of a theory T of a language L that agrees with the root entry R in a tableau $\tau = \cup\tau_n$ from T. Then $\mathcal{A}$ can be expanded to a language L_C so that it agrees with some branch V in τ.*

Proof. We prove the lemma by induction on n. We will find a branch V_n in τ_n and an expansion $\mathcal{A}_n$ by constants c^A for all $c \in L_C \setminus L$ on V_n such that $\mathcal{A}_n$ agrees with V_n and $V_{n-1} \subseteq V_n$.

Assume we have a branch V_n in τ_n and an expansion $\mathcal{A}_n$ that agrees with V_n.

- If τ_{n+1} is obtained from τ_n without extending the branch V_n, take $V_{n+1} = V_n$ and $\mathcal{A}_{n+1} = \mathcal{A}_n$.
- If τ_{n+1} is obtained from τ_n by appending $T\varphi$ for some $\varphi \in T$ to the end of V_n, let V_{n+1} be the branch V_n with $T\varphi$ at the end and $\mathcal{A}_{n+1} = \mathcal{A}_n$. Since $\mathcal{A} \vDash \varphi$, $\mathcal{A}_{n+1}$ agrees with V_{n+1}.
- Otherwise τ_{n+1} is obtained from τ_n by appending an atomic tableau for an entry P on V_n to V_n. By induction we know that $\mathcal{A}$ agrees with P. If P is formed by a logical connective, we take $\mathcal{A}_{n+1} = \mathcal{A}$ and verify that V_n can always be extended to V_{n+1} (this is the same as in propositional logic). If P is in the form $T(\forall x)\varphi(x)$, let V_{n+1} be the unique extension of V_n to a branch τ_{n+1} by the entry $T\varphi(x/t)$. Let $\mathcal{A}_{n+1}$ be any expansion of $\mathcal{A}_n$ by new constants from t. Since $\mathcal{A}_n \vDash (\forall x)\varphi(x)$ also $\mathcal{A}_{n+1} \vDash \varphi(x/t)$. Analogously for P in the form $F(\exists x)\varphi(x)$. Finally, if P is in the form $T(\exists x)\varphi(x)$, let V_{n+1} be the unique extension of V_n to a branch in τ_{n+1}, i.e. by the entry $T\varphi(x/c)$. Since $\mathcal{A}_n \vDash (\exists x)\varphi(x)$ there is some $a \in A$ such that $\mathcal{A} \vDash \varphi(x)[e(x/a)]$ for every assignment e. Let $\mathcal{A}_{n+1}$ be the expansion of $\mathcal{A}_n$ by a new constant $c^A = a$. Then $\mathcal{A}_{n+1} \vDash \varphi(x/c)$. Analogously for P in the form $F(\forall x)\varphi(x)$.

The base step for $n = 0$ follows from the analysis of the atomic tableaux for the root entry R applying the assumption that $\mathcal{A}$ agrees with R. □

We can finally prove the soundness of the tableau method in first-order logic. The proof is in fact almost identical to the one in propositional logic.

Theorem 11. *For every theory T and sentence φ, if φ is tableau provable from T, then φ is valid in T, i.e. $T \vdash \varphi \Rightarrow T \vDash \varphi$.*

Proof. Let φ be tableau provable from T, i.e. there is a contradictory tableau from T with root entry $F\varphi$. Assume for contradiction that

φ is not valid in T, i.e. there is a model $\mathcal{A}$ of T in which φ is not true. However, in such a case $\mathcal{A}$ agrees with the root entry $F\varphi$ of the proof and therefore by the previous lemma it can be expanded to the language L_C so that it agrees with a branch in τ. But all the branches in τ are contradictory, thus it is not possible. □

Now we would like to prove the completeness of the tableau method in first-order logic. We will again use the branch in a non-contradictory tableau and a model that agrees with the branch in order to provide a counterexample. This time, the model will be the so-called canonical model. In the canonical model, the universe is formed by all the ground terms of the language and the representations of all function symbols are fixed. This means we can imagine the ground atomic formulas, informally, as complex names of propositional variables and therefore the proof of completeness in principle reduces to its equivalent in propositional logic.

Let V be a noncontradictory branch of a finished tableau from a theory T of a language $L = \langle \mathcal{F}, \mathcal{R} \rangle$. The *canonical model* from V is the L_C-structure $\mathcal{A} = \langle A, \mathcal{F}^A, \mathcal{R}^A \rangle$, where A is the set of all ground terms of the language L_C, $f^A(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ for every n-ary function symbol $f \in \mathcal{F} \cup (L_C \setminus L)$[16], and $t_1, \ldots, t_n \in A$, and $R^A(t_1, \ldots, t_n) \Leftrightarrow TR(t_1, \ldots, t_n)$ is an entry on V for every n-ary relation symbol $R \in \mathcal{R}$ and every $t_1, \ldots, t_n \in A$.

[16] The expression $f(t_1, \ldots, t_n)$ is a ground term of the language and therefore is in A.

If L is with equality, T^* is an extension of T by the axioms of equality for L. Equality will be interpreted in the model by some relation $=^A$. We also have $t_1 =^A t_2 \Leftrightarrow T(t_1 = t_2)$ is an entry of V. Since V contains all axioms of equality (it is finished), the relation $=^A$ is a congruence for all functions and relations in $\mathcal{A}$. If we require that equality is represented by identity, we take the quotient of the canonical model $\mathcal{A}$ by the congruence $=^A$. The *canonical model with equality* from V is the quotient ${}^{\mathcal{A}}/_{=^A}$.

Lemma 6. *The canonical model $\mathcal{A}$ from a noncontradictory finished branch V agrees with V.*

Proof. We will prove the lemma by induction on the structure of sentence φ in an entry on V.

- For atomic φ, if $T\varphi$ is on V then $\mathcal{A} \vDash \varphi$ by definition. If $F\varphi$ is on V, then $T\varphi$ is not on V since V is noncontradictory, so $\mathcal{A} \vDash \neg\varphi$ by definition.

- If $T(\varphi \wedge \psi)$ is on V, then $T\varphi$ and $T\psi$ are on V since V is finished. By induction $\mathcal{A} \vDash \varphi$ and $\mathcal{A} \vDash \psi$, thus $\mathcal{A} \vDash \varphi \wedge \psi$. For other logical connectives similarly (this step is the same as in the proof in propositional logic).

- If $T(\forall x)\varphi(x)$ is on V, then $T\varphi(x/t)$ is on V for every term $t \in A$ since V is finished. By induction $\mathcal{A} \vDash \varphi(x/t)$ for every $t \in A$ and thus $\mathcal{A} \vDash (\forall x)\varphi(x)$. Similarly for $F(\exists x)\varphi(x)$ on V.

- Finally, if $T(\exists x)\varphi(x)$ is on V, then $T\varphi(x/c)$ is on V for some $c \in A$. By induction, $\mathcal{A} \vDash \varphi(x/c)$ and thus $\mathcal{A} \vDash (\exists x)\varphi(x)$. Similarly for $F(\forall x)\varphi(x)$ on V.

□

We can finally prove the completeness of the tableau method. As always, the proof is very similar to the one in propositional logic.

Theorem 12. *For every theory T and sentence φ, if φ is valid in T, then φ is tableau provable from T, i.e. $T \vDash \varphi \Rightarrow T \vdash \varphi$.*

Proof. Let φ is valid in T. We will show that an arbitrary finished tableau τ from a theory T with the root entry $F\varphi$ is contradictory. Assume for a contradiction, that it is not, i.e. that there is a noncontradictory branch V in τ. By the previous lemma, there is a structure $\mathcal{A}$ for L_C that agrees with V, in particular with the root entry $F\varphi$, i.e. $\mathcal{A} \vDash \neg\varphi$. Let $\mathcal{A}'$ be the reduct of $\mathcal{A}$ to the language L, then $\mathcal{A}' \vDash \neg\varphi$. Since V is finished, it contains $T\psi$ for every $\psi \in T$. Thus $\mathcal{A}'$ is a model of T. But this contradicts the assumption that φ is valid in T (φ is not valid in the model $\mathcal{A}'$ of the theory T). Therefore the tableau τ is a proof of φ from T. □

As with propositional logic, we can again redefine the semantical terms using syntactical notions. In fact, this and the next paragraph are copied from the propositional part with only minor changes. First of all, we define the *set of theorems of L-theory T*

$$\mathrm{Thm}^L(T) = \{\varphi | \varphi \in \mathrm{Fm}_L, T \vdash \varphi\}.$$

We say that a theory T *is inconsistent* if $T \vdash \bot$, otherwise T *is consistent*. A theory T *is complete* if it is consistent and every sentence is provable or refutable from T, i.e. if $T \vdash \neg\varphi$ or $T \vdash \varphi$. A theory T in L is *an extension* of T' in L', if $L' \subseteq L$ and $\mathrm{Thm}^{L'}(T') \subseteq \mathrm{Thm}^L(T)$; the extension is *simple* if $L = L'$, and it is *conservative* if $\mathrm{Thm}^{L'}(T') = \mathrm{Thm}^L(T) \cap \mathrm{Fm}_{L'}$. Two *theories T and T' are equivalent* if T is an extension of T' and vice versa.

There are strong relations between the syntactic terms introduced above and the semantic terms introduced in the previous chapter. Most of these are corollaries of the soundness and completeness of the tableau method. For each theory T and sentences φ, ψ of a language L

1. $T \vdash \varphi$ if and only if $T \vDash \varphi$,
2. $\mathrm{Thm}^L(T) = \theta^L(T)$,
3. T is inconsistent if and only if T is unsatisfiable, i.e. it has no model,
4. T is complete if and only if T is semantically complete, i.e. it has a single model, up to elementary equivalence, and
5. (deduction theorem) $T \cup \{\varphi\} \vdash \psi$ if and only if $T \vdash \varphi \to \psi$.

A corollary of the proofs is the weak version of the Lövenheim-Skolem theorem.

Theorem 13. *Every consistent theory T of a countable language L without equality has a countably infinite model.*

Proof. Let τ be the systematic tableau from T with $F\bot$ in the root. Since τ is consistent, $\bot$ is not provable from T and therefore τ contains a noncontradictory branch V, and there exists a canonical model $\mathcal{A}$ from V (in language L_C). Since $\mathcal{A}$ agrees with V, its reduct to the language L is the desired countably infinite model of T. □

We needed the assumption that the theory is without equality because the canonical model with equality can be also finite (but it is always countable).

As in propositional logic, we can also prove the compactness theorem in first-order logic.

Theorem 14 (compactness)**.** *A theory T has a model if and only if every finite subset of T has a model.*

Proof. The implication from left to right is obvious. For the other implication, if T has no model, then it is inconsistent, i.e. $\bot$ is provable by a systematic tableau τ from T. Since τ is finite, $\bot$ is provable from some finite $T' \subseteq T$, i.e. T' has no model. □

The compactness theorem has an interesting corollary which gives us the so called non-standard model of the natural numbers. Let $\mathbb{N} = \langle \mathbb{N}, S, +, \cdot, 0, \leq \rangle$ be the standard model of the natural numbers, and let $\mathrm{Th}(\mathbb{N})$ be the theory consisting of all sentences valid in $\mathbb{N}$. For $n \in \mathbb{N}$, we denote $\underline{n} = S(S(S(S(...S(0)))))$ (n applications of the function S) the so called *n-th numeral*. Now, let us consider a theory T with a new constant symbol c, such that $T = \mathrm{Th}(\mathbb{N}) \cup \{\underline{n} \lneq c | n \in \mathbb{N}\}$. Every finite subset of this theory has a model, therefore also the whole theory T has a model. This is a *non-standard model of natural numbers*. Every sentence that is valid in $\mathrm{Th}(\mathbb{N})$ is also valid in this model, but it additionally contains an element that is greater than all natural numbers.

In mathematics, we very often define new functions or relations using formulas from the theory we work with. Now, we will show that such definitions do not in any way increase the strength of the theory, i.e. that by adding definitions of new symbols to a theory, we obtain a conservative extension of that theory. Before we get to that point, we will show a simple lemma that gives us a simple way to show that a theory is an (conservative) extension of another theory.

Lemma 7. *Let T be a theory of L and T' a theory of L' where $L \subseteq L'$.*

1. *T' is an extension of T if and only if the reduct $\mathcal{A}$ of every model $\mathcal{A}'$ of T' to the language L is a model of T,*

2. *T' is a conservative extension of T if T' is an extension of T and every model $\mathcal{A}$ of T can be expanded to the language L' on a model $\mathcal{A}'$ of T'.*

Proof. 1. If T' is an extension of T and $\varphi \in T$ then $T' \vDash \varphi$. Thus $\mathcal{A}' \vDash \varphi$ and also $\mathcal{A} \vDash \varphi$, which implies that $\mathcal{A}$ is a model of T. On the other

hand, if $\mathcal{A}$ is a model of T and $T \models \varphi$ for a φ of language L, then $\mathcal{A} \models \varphi$ and also $\mathcal{A}' \models \varphi$. Therefore $T' \models \varphi$ and T' is an extension of T.

2. If $T' \models \varphi$ where φ is of the language L and $\mathcal{A}$ is a model of T, then in its expansion $\mathcal{A}'$ that is a model of T' we have also $\mathcal{A}' \models \varphi$. Thus also $\mathcal{A} \models \varphi$, and hence $T \models \varphi$. Therefore T' is conservative. □

We now start by showing that adding a definition of a new relation does not change the strength of a theory. Let T be a theory of a language L and $\psi(x_1, \ldots, x_n)$ a formula of L with free variables $x_1, \ldots, x_n$. Let L' denote the extension of the language L with a new n-ary relation symbol R. The *extension of T by definition of R* with a formula ψ is the theory T' of L' obtained from T by adding the axiom $R(x_1, \ldots, x_n) \leftrightarrow \psi(x_1, \ldots, x_n)$. Obviously, in such a case every model of T can be uniquely expanded into a model of T' and therefore T' is a conservative extension of T. Moreover, we can "translate" each formula φ' of L' into a formula φ of L such that $T' \models \varphi' \leftrightarrow \varphi$. We just replace each sub-formula $R(t_1, \ldots, t_n)$ with $\psi'(x_1/t_1, \ldots, x_n/t_n)$, where ψ' is a suitable variant of ψ (such that every substitution can be performed).

Similarly, adding a definition of a function symbol does not increase the strength of the theory. Let T be a theory of a language L, $\psi(x_1, \ldots, x_n, y)$ an L-formula with free variables $x_1, \ldots, x_n, y$ such that $T \models (\exists y)\psi(x_1, \ldots, x_n, y)$ (existence) and $T \models \psi(x_1, \ldots, x_n, y) \wedge \psi(x_1, \ldots, x_n, z) \rightarrow y = z$ (uniqueness). Let L' be an extension of L with a new n-ary function symbol f. The *extension of T by definition of f* by the formula ψ is the theory T' of L' obtained from T by adding the axiom $f(x_1, \ldots, x_n) = y \leftrightarrow \psi(x_1, \ldots, x_n, y)$. Commonly, ψ is $t(x_1, \ldots, x_n) = y$ for a term t and variables $x_1, \ldots, x_n$. In such a case both existence and uniqueness always hold. Obviously, every model of T can be uniquely extended to a model of T' and therefore T' is again a conservative extension of T. Moreover, we can again "translate" the formula φ' in L' into a formula φ in L such that these two formulas are T'-equivalent (i.e. $T' \models \varphi' \leftrightarrow \varphi$). We show the translation only for formulas that contain the function f only once; for other formulas, we can repeat the process inductively. Let φ^* denote the formula obtained from φ' by replacing the term $f(t_1, \ldots, t_n)$ with a new variable z. Let φ be the formula $(\exists z)(\varphi^* \wedge \psi'(x_1/t_1, \ldots, x_n/t_n, y/z)$, where ψ' is a suitable variant of ψ. Now, if $\mathcal{A}$ is a model of T', e is an assignment, and $a = f^A(t_1, \ldots, t_n)[e]$, by the two conditions, $\mathcal{A} \models \psi'(x_1/t_1, \ldots, x_n/t_n, y/z)[e]$ if and only if $e(z) = a$. Thus $\mathcal{A} \models \varphi[e] \Leftrightarrow \mathcal{A} \models \varphi^*[e(z/a)] \Leftrightarrow \mathcal{A} \models \varphi'[e]$ for every assignment e, i.e. $\mathcal{A} \models \varphi' \leftrightarrow \varphi$ and so $T' \models \varphi' \leftrightarrow \varphi$.

The two previous paragraphs show that if we have a theory T' of the language L' which was obtained from T of L by successive definitions of relation and function symbols (*extension of T by definitions*) then every model of T can be uniquely expanded into a model of T', T' is a conservative extension of T, and for every formula φ' of L' there is a formula φ of L such that $T' \models \varphi' \leftrightarrow \varphi$.

Resolution in First-Order Logic

We now aim to introduce the resolution method in predicate logic. To this end, we will need to show that the problem of satisfiability of theories can be reduced to open theories. We will show that every theory has an open conservative extension and therefore the satisfiability of the theory can be expressed as the satisfiability of the open extension.

We say that two *theories T and T' are equisatisfiable* if T has a model if and only if T' has a model. A formula is in the prenex (normal) form (PNF) if it is written as $(Q_1x_1)\dots(Q_nx_n)\varphi'$, where Q_i denotes $\forall$ or $\exists$ and φ' is an open formula called the *matrix*. $(Q_1x_1)\dots(Q_nx_n)$ is the *prenex*. If all the quantifiers are $\forall$ then φ is a *universal formula*.

We obtain the equisatisfiable open theory T' by first transforming all axioms of T into prenex form. Then we remove the existential quantifiers (we will create the so-called Skolem variant of the formula) thus obtaining a universal formula. The matrices of the universal formulas will be the axioms of T'.

Prenex Normal Form

We start by transforming formulas into prenex normal form. The transformation is based on replacing some occurrences of a sub-formula ψ in a formula φ by an equivalent sub-formula ψ'. It is easy to show (by induction on the structure of formula φ) that the obtained formula is equivalent to φ.

Let Q denote $\forall$ or $\exists$ and $\bar{Q}$ denote the complementary quantifier. For all formulas φ and ψ such that x is not free in the formula ψ, the following equivalences (*conversion rules*) hold:

$$\begin{gathered}
\neg(Qx)\varphi \leftrightarrow (\bar{Q}x)\neg\varphi \\
((Qx)\varphi \wedge \psi) \leftrightarrow (Qx)(\varphi \wedge \psi) \\
((Qx)\varphi \vee \psi) \leftrightarrow (Qx)(\varphi \vee \psi) \\
((Qx)\varphi \to \psi) \leftrightarrow (\bar{Q}x)(\varphi \to \psi) \\
(\psi \to (Qx)\varphi) \leftrightarrow (Qx)(\psi \to \varphi)
\end{gathered}$$

All the equivalences can be proved by the tableau method. The assumption that x is not free in ψ is necessary in each rule above.

By induction on the structure of formula φ, we can show that for every formula φ there is an equivalent formula φ' in prenex normal form, i.e. $\vDash \varphi \leftrightarrow \varphi'$. We just apply the conversion rules above (and replace subformulas with suitable variants if needed).

Skolem Variant

If the prenex of the formula contains only universal quantifiers ($\forall$), we can remove them, thus obtaining an equivalent (and therefore also equisatisfiable) open formula. However, in many cases, some of the quantifiers will be existential. In such cases we can use the so-called Skolem variant as the equisatisfiable formula.

Let φ be a sentence of a language L in prenex normal form, let $y_1, \ldots, y_n$ be the existentially quantified variables in φ (in this order), and for every $i \leq n$ let $x_1, \ldots, x_{n_i}$ be the universally quantified variables in φ before y_i. Let L' be an extension of L with new n_i-ary function symbols f_i for all $i \leq n$. Let φ_S denote the formula obtained from φ by removing all the $(\exists y_i)$ and replacing each occurrence of y_i by $f_i(x_1, \ldots, x_{n_i})$. Then φ_S is the *Skolem variant* of φ.

For example, for the formula

$$(\exists y_1)(\forall x_1)(\forall x_2)(\exists y_2)(\forall x_3)R(x_1, y_1, x_2, y_2, x_3)$$

the Skolem variant is

$$(\forall x_1)(\forall x_2)(\forall x_3)R(x_1, f_1, x_2, f_2(x_1, x_2), x_3)\,.$$

Informal! The new function symbols provide witnesses for the existentially quantified variables. We need functions, as these can be different based on the previously universally quantified variables. That is also the reason why the functions all have the previously quantified variables as their parameters. The existentially quantified variables do not need to be included in the parameters, as the function can "compute" them from the universally quantified ones.

We will now prove that the Skolem variant φ_S of φ is equisatisfiable with φ.

Lemma 8. *Let φ be a sentence $(\forall x_1) \ldots (\forall x_n)(\exists y)\psi$ of L and φ' be a sentence $(\forall x_1) \ldots (\forall x_n)\psi(y/f(x_1, \ldots, x_n))$ where f is a new function symbol. Then*

1. *the reduct $\mathcal{A}$ of every model $\mathcal{A}'$ of φ' to the language L is a model of φ, and*
2. *every model of φ can be expanded into a model $\mathcal{A}'$ of φ'.*

Proof. Let $\mathcal{A}' \vDash \varphi'$ and $\mathcal{A}$ be the reduct of $\mathcal{A}'$ to L. Since $\mathcal{A} \vDash \psi[e(y/a)]$ for every assignment e where $a = (f(x_1, \ldots, x_n)^{\mathcal{A}'})[e]$, we have also $\mathcal{A} \vDash \varphi$.

On the other hand, let $\mathcal{A} \vDash \varphi$. There is a function $f^A : A^n \to A$ such that for every assignment e, $\mathcal{A} \vDash \psi[e(y/a)]$ where $a = f^A(e(x_1), \ldots, e(x_n))$, and thus the expansion $\mathcal{A}'$ of $\mathcal{A}$ by function f^A is a model of φ. □

If φ' is a Skolem variant of φ then both statements above also hold and therefore φ and φ' are equisatisfiable.

It is important to realize that a formula and its Skolem variant are not equivalent. For example, $\varphi_S \equiv (\forall x)P(x, f(x))$ is a Skolem

variant of $\varphi \equiv (\forall x)(\exists y)P(x,y)$. Let $\mathcal{A} = \langle\{0,1\}, P^A, f^A\rangle$, where $P^A = \{(0,0),(1,1)\}$ and $f^A(0) = 1, f^A(1) = 0$. Then $\mathcal{A} \vDash \varphi$, but $\mathcal{A} \nvDash \varphi_S$. On the other hand, if φ_S is valid in a structure, φ also is, as f gives the y for which the formula holds, therefore, for a formula φ and its Skolem variant φ_S, we always have $\vDash \varphi_S \to \varphi$.

The difference between equivalence and equisatisfiability in this case is caused by the fact that the new functions in the Skolem variant are not in any way limited to cases where they provide the witnesses for the existence and can therefore be defined arbitrarily, as we saw above. If we defined the function in the "intended" way (in this case as the identity on A), the Skolem variant would hold in such a structure. Therefore, it is also satisfiable.

The transformation to prenex normal form and the following creation of Skolem variant gives us the possibility to reduce the question of satisfiability to open theories, i.e. for every theory, there is an equisatisfiable open theory. This is a corollary of the so-called Skolem theorem.

Theorem 15 (Skolem)**.** *Every theory T has an open conservative extension T^*.*

Proof. We assume T is in closed form (otherwise we can take the universal closures of all the axioms of T), and let L be its language. We first replace each axiom of T by an equivalent formula in prenex normal form, thus obtaining theory T°. Next, we replace each axiom in T° by its Skolem variant, which gives us a theory T' in a language $L' \supseteq L$. Since every reduct of every model of T' to L is a model of T, T' is an extension of T. Furthermore, every model of T can be expanded to a model of T' and therefore T' is a conservative extension of T. Every axiom of T' is a universal sentence, therefore we can take the matrices of these sentences to obtain an open theory T° equivalent to T'. T° is the open conservative extension of T. □

Herbrand Model

We can even reduce the problem of satisfiability to propositional logic and show that if an open theory is unsatisfiable, we can demonstrate it via ground terms. For example, in the language $L = \langle P, R, f, c\rangle$ the theory $T = \{P(x,y) \vee R(x,y), \neg P(c,y), \neg R(x,f(x))\}$ is unsatisfiable because the following conjunction of finitely many ground instances of axioms of T is unsatisfiable:

$$P(c,f(c)) \vee R(c,f(c)) \wedge \neg P(c,f(c)) \wedge \neg R(c,f(c))\,.$$

This can also be seen as a unsatisfiable propositional formula $(p \vee r) \wedge \neg p \wedge \neg r$.

A *ground instance* of formula φ with free variables $x_1, \ldots, x_n$ is $\varphi(x_1/t_1, \ldots, x_n/t_n)$ where $t_1, \ldots, t_n$ are ground terms.

The reduction of satisfiability to propositional logic is done using the notion of Herbrand models. Let $L = \langle \mathcal{R}, \mathcal{F}\rangle$ be a language with at least one constant symbol (we can add a new constant symbol, if

needed). The *Herbrand universe* for L is the set of all ground terms of L. An L-structure $\mathcal{A}$ is a *Herbrand structure*, if its domain A is the Herbrand universe for L and for each n-ary function symbol $f \in \mathcal{F}$, $t_1, \ldots, t_n \in A$, $f^A(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$. A *Herbrand model* of a theory T is a Herbrand structure that is a model of T.

We can see that the definition of the Herbrand universe and of the function symbols in the Herbrand structure is the same as in the canonical model, however, Herbrand structures do not specify the relations. This is important, as every canonical model is also a Herbrand model.

The reduction of unsatisfiability to propositional logic is formally given by Herbrand's theorem.

Theorem 16 (Herbrand)**.** *Let T be an open theory of a language L without equality and with at least one constant symbol. Then either T has a Herbrand model, or there are finitely many ground instances of axioms of T whose conjunction is unsatisfiable, and thus T has no model.*

Proof. Let T' be the set of all ground instances of axioms of T. Consider a finished systematic tableau τ from T' in the language L (without adding new constant symbols) with the root entry $F\bot$. If the tableau τ contains a noncontradictory branch V, the canonical model from V is a Herbrand model. Otherwise, τ is contradictory, i.e. $T' \vdash \bot$. Therefore τ is also finite and $\bot$ is provable from finitely many formulas of T', i.e. their conjunction is unsatisfiable. □

The Herbrand model also works in languages L with equality: we just need to take the extension T^* of T with the axioms of equality and if T^* has a Herbrand model $\mathcal{A}$ we take its quotient by $=^A$.

A corollary of Herbrand's theorem is that an open theory T of a language L with at least one constant symbol is satisfiable if and only if the theory T' of all ground instances of axioms of T is satisfiable. Why? If T has a model $\mathcal{A}$, every instance of each axiom of T is valid in $\mathcal{A}$, and thus $\mathcal{A}$ is also a model of T'. If T is unsatisfiable, then by Herbrand's theorem there are finitely ground instances of axioms of T that are not satisfiable and therefore T' is also unsatisfiable.

Another corollary says that for every open $\varphi(x_1, \ldots, x_n)$ of a language L with at least one constant symbol, the formula $(\exists x_1) \ldots (\exists x_n)\varphi$ is valid if and only if there exist mn ground terms t_{ij} of L for some m such that $\varphi(x_1/t_{11}, \ldots, x_n/t_{1n}) \vee \cdots \vee \varphi(x_1/t_{m1}, \ldots, x_n/t_{mn})$ is a tautology. We know that $(\exists x_1) \ldots (\exists x_n)\varphi$ is valid if and only if its negation $(\forall x_1) \ldots (\forall x_n)(\neg\varphi)$ is unsatisfiable, which is equivalent to $\neg\varphi$ being unsatisfiable. Herbrand's theorem for $\{\neg\varphi\}$ gives us finitely many (m) ground instances of $\{\neg\varphi\}$ such that their conjunction ψ is unsatisfiable. The negation of ψ is the desired tautology.

We will now discuss the resolution method in predicate logic. It is again a refutation procedure, i.e. it aims to show that a formula or theory are not satisfiable. It assumes the formulas are open and in CNF. These are then represented in the clausal form as in propositional logic. Now, *literals* are atomic formulas and their negations, *a clause* is a finite set of literals (□ denotes the empty clause) and a *formula in clausal form*

is a set of clauses. The resolution rule in first-order logic is based on the rule in propositional logic, but is more general – it allows us to resolve through literals that are different, if they are unifiable (i.e. if there is such a substitution that they become identical). Resolution in first-order logic is thus based on the resolution in propositional logic and unification.

Resolution

Herbrand's theorem actually gives us an (ineffective) way, how to perform the resolution in first-order logic. If we have an input formula S in clausal form and we assume the language has at least one constant symbol, we can create the set S' of all ground instances of all clauses in S. If we now consider the atomic sentences as names of propositional variables, we may view S' as a propositional formula in clausal form. We may now verify that it is unsatisfiable using the propositional version of the resolution method. While this method in principle works, it is very inefficient, as we need to work with large numbers of ground instances. Therefore, instead of grounding, we use suitable substitutions in order to perform the resolution on as general clauses as possible.

A *substitution* is a finite set $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$, where $x_i, i \leq n$ are distinct variables, t_i are terms and the term t_i is not x_i. If all t_i are ground terms, σ is a *ground substitution*; if t_i are distinct variables, σ is a *renaming of variables*. An *instance of an expression E by substitution* $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$ is the expression $E\sigma$ obtained from E by simultaneously replacing all occurrences of x_i by t_i. For a set S of expressions, let $S\sigma = \{E\sigma | E \in S\}$. For two substitutions $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$ and $\tau = \{y_1/s_1, \ldots, y_m/s_m\}$ we define the *composition of σ and τ* as $\sigma\tau = \{x_i/t_i\tau | x_i \in X, t_i\tau \text{ is not } x_i\} \cup \{y_j/s_j | y_j \in Y \setminus X\}$ where $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$. For every expression E and substitutions σ, τ, ρ it holds (without proof) that $(E\sigma)\tau = E(\sigma\tau)$ and $(\sigma\tau)\rho = \sigma(\tau\rho)$.

A unification is a special substitution. Let $S = \{E_1, \ldots, E_n\}$ be a finite set of expressions. A *unification for S* is a substitution σ such that $E_1\sigma = E_2\sigma = \cdots = E_n\sigma$, i.e. $S\sigma$ is a singleton. We say that *S is unifiable*, if S has a unification. A unification σ of S is a *most general unification (mgu)* if for every unification τ of S there is a substitution λ such that $\tau = \sigma\lambda$. Obviously, if σ and τ are two different most general unifications of S, they differ only in renaming of variables.

The general resolution rule in first-order logic is then defined in the following way: let C_1, C_2 be clauses with distinct variables such that $C_1 = C_1' \sqcup \{A_1, \ldots, A_n\}$ and $C_2 = C_2' \sqcup \{\neg B_1, \ldots, \neg B_m\}$, and let σ be a most general unifier of $S = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$. The *resolvent* of C_1 and C_2 is $C = C_1'\sigma \cup C_2'\sigma$.

For example, in the clauses $\{P(x), Q(x,z)\}$ and $\{\neg P(y), \neg Q(f(y), y)\}$ we can unify $S = \{Q(x,z), Q(f(y),y)\}$ applying the most general unification $\sigma = \{x/f(y), z/y\}$ and resolve to a clause $\{P(f(y)), \neg P(y)\}$.

The resolution proof and related notions are then defined in almost

the same way as in propositional logic. A resolution proof (deduction) of a clause C from a formula S is a finite sequence $C_0, \ldots, C_n = C$, such that for every $i \leq n$, we have $C_i = C'_i\sigma$ for some $C'_i \in S$ and a renaming of variables σ, or C_i is a resolvent of some previous clauses. A clause C is (resolution) provable from S ($S \vdash_R C$), if it has a resolution proof from S. A refutation of a formula S is a resolution proof of $\square$ from S. S is resolution refutable if $S \vdash_R \square$.

It remains to show how to find the most general unifications we need in the resolution method. There is in fact a simple algorithm to find them: let S be a finite set of expressions and p be the leftmost position in which some expressions of S differ. Then the difference in S is the set $D(S)$ of subexpressions of all expressions of S starting at the position p. For example $S = \{P(x,y), P((f(x),z), P(z,f(x)\}$ has $D(S) = \{x, f(x), z\}$.

The input of the algorithm is a nonempty finite set of expressions S, its output is the most general unification σ of S or information that S is not unifiable. The algorithm performs the steps bellow:

1. Let $S_0 \leftarrow S, \sigma_0 \leftarrow \emptyset, k \leftarrow 0$.

2. If S_k is a singleton, output the substitution $\sigma = \sigma_0\sigma_1 \ldots \sigma_k$.

3. If $D(S_k)$ contains a variable x and a term t with no occurrence of x, let $\sigma_{k+1} \leftarrow \{x/t\}, S_{k+1} \leftarrow S_k\sigma_{k+1}, k \leftarrow k+1$ and go to step 2.

4. Otherwise output "S is not unifiable".

The occurrence check in step 3 can be expensive and slows the algorithm down significantly. Therefore some implementations (e.g. those in most Prolog interpreters) ignore that. However, it can lead to infinite loops.

We now show the correctness of the unification algorithm – the unification algorithm outputs a correct answer in finite time for any input S, i.e. a most general unification σ of S or it detects that S is not unifiable. The algorithm eliminates one variable in each iteration, therefore it finishes in finite time. If it ends negatively after k iterations, $D(S_k)$ is not unifiable and therefore also S is not. If it outputs $\sigma = \sigma_0\sigma_1 \ldots \sigma_k$, clearly σ is a unification of S. We only need to show it is a most general one, i.e. for every unification τ of S there is a substitution λ such that $\tau = \sigma\lambda$. We will show that in this case, $\lambda = \tau$ and therefore $\tau = \sigma\tau$ (this property will also be important in the proof of the lifting lemma later; let us call this property (*)). Let τ be a unification of S; we will show that $\tau = \sigma_0\sigma_1 \ldots \sigma_i\tau$ for every $i \leq k$. It obviously holds for $k = 0$. Let $\sigma_{i+1} = \{x/t\}$ and assume $\tau = \sigma_0\sigma_1 \ldots \sigma_i\tau$. It suffices to show that $v\sigma_{i+1}\tau = v\tau$ for every variable τ. If $v \neq x$, $v\sigma_{i+1} = v$ and it holds. Otherwise $v = x$ and $v\sigma_{i+1} = x\sigma_{i+1} = t$. Since τ unifies $S_i = S\sigma_0\sigma_1 \ldots \sigma_i$ and both the variable x and term t are in $D(S_i)$ (otherwise σ_{i+1} would be different), τ has to unify x and t, i.e. $t\tau = x\tau$ as required.

Soundness and Completeness

In order to show the soundness of the resolution method, we first show the soundness of the general resolution rule.

Lemma 9. *Let C be a resolvent of clauses C_1 and C_2. For every L-structure $\mathcal{A}$, if $\mathcal{A} \models C_1$ and $\mathcal{A} \models C_2$, then $\mathcal{A} \models C$.*

Proof. Let $C_1 = C_1' \sqcup \{A_1, \ldots, A_n\}$ and $C_2 = C_2' \sqcup \{\neg B_1, \ldots, \neg B_m\}$, let σ be a most general unifier of $S = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$, and let $C = C_1'\sigma \cup C_2'\sigma$. Since C_1, C_2 are open, also $\mathcal{A} \models C_1\sigma$ and $\mathcal{A} \models C_2\sigma$. We have $C_1\sigma = C_1'\sigma \cup \{S\sigma\}$ and $C_2\sigma = C_2' \cup \{\neg(S\sigma)\}$. We show $\mathcal{A} \models C[e]$ for every assignment e. If $\mathcal{A} \models S\sigma[e]$, then $\mathcal{A} \models C_2'\sigma[e]$, and thus $\mathcal{A} \models C[e]$, otherwise $\mathcal{A} \not\models S\sigma[e]$, so $\mathcal{A} \models C_1'\sigma[e]$ and thus $\mathcal{A} \models C[e]$. □

Theorem 17 (soundness of resolution)**.** *If S is resolution refutable, then S is unsatisfiable.*

Proof. Let $S \vdash_R \square$. Suppose $\mathcal{A} \models S$ for some structure $\mathcal{A}$. By the soundness of the general resolution rule we also have $\mathcal{A} \models \square$, which is not possible. □

The proof of the completeness of resolution in predicate logic is based on its completeness in propositional logic. The connection between the propositional and predicate levels is given by the lifting lemma.

Lemma 10 (lifting)**.** *Let $C_1^* = C_1\tau_1$ and $C_2^* = C_2\tau_2$ be ground instances of clauses C_1 and C_2 with distinct variables and let C^* be a resolvent of C_1^* and C_2^*. Then there exists a resolvent C of C_1 and C_2 such that $C^* = C\tau_1\tau_2$ is a ground instance of C.*

Proof. Assume C^* is a resolvent of C_1^* and C_2^* through some literal $P(t_1, \ldots, t_k)$. We show that the resolution step provides the desired C. We have $C_1 = C_1' \sqcup \{A_1, \ldots, A_n\}$ and $C_2 = C_2' \sqcup \{\neg B_1, \ldots, \neg B_m\}$, where $\{A_1, \ldots, A_n\}\tau_1 = \{P(t_1, \ldots, t_k\}$ and also $\{\neg B_1, \ldots, \neg B_m\}\tau_2 = \{\neg P(t_1, \ldots, t_k)\}$. Thus $(\tau_1\tau_2)$ unifies $S = \{A_1, \ldots, A_n, B_1, \ldots B_m\}$ and if σ is a most general unification of S from the unification algorithm, then $C = C_1'\sigma \cup C_2'\sigma$ is a resolvent of C_1, C_2. We know (by the property (*)) that $(\tau_1\tau_2) = \sigma(\tau_1\tau_2)$ and hence $C\tau_1\tau_2 = (C_1'\sigma \cup C_2'\sigma)\tau_1\tau_2 = C_1'\sigma\tau_1\tau_2 \cup C_2'\sigma\tau_1\tau_2 = C_1'\tau_1 \cup C_2'\tau_2 = (C_1 \setminus \{A_1, \ldots, A_n\})\tau_1 \cup (C_2 \setminus \{\neg B_1, \ldots, \neg B_m\})\tau_2 = (C_1^* \setminus \{P(t_1, \ldots, t_k\}) \cup (C_2^* \setminus \{\neg P(t_1, \ldots, t_k)\}) = C^*$. □

We can now easily show by induction on the length of the resolution proof for a set S' of all ground instances of clauses of a formula S, that if $S' \vdash_R C'$ (on the propositional level) where C' is a ground clause, then $C' = C\sigma$ for some clause C and a ground substitution σ such that $S \vdash_R C$ (on the predicate level). This leads to the completeness of the resolution method.

Theorem 18 (completeness of resolution)**.** *If S is unsatisfiable, then $S \vdash_R \square$.*

Proof. If S is unsatisfiable, then by the corollary of Herbrand's theorem the set S' of all ground clauses is also unsatisfiable. By the completeness of resolution in propositional logic, $S' \vdash_R \square$. By the above paragraph, there is a clause C and a ground substitution σ such that $\square = C\sigma$ and $S \vdash_R C$. But the only clause that has $\square$ as a ground instance is the clause $C = \square$. $\square$

This concludes the part of the lecture dedicated to predicate logic. We closely followed the previous part on propositional logic and extended the ideas to the first-order logic. The differences are mostly given by the more expressive language of predicate logic. We again started the discussion with syntax and semantics, and we saw that models in first order logic are structures. Then, we showed two formal proof methods – the tableau method and resolution.

In the next (and last) part, we will first discuss the basics of model theory and then we will show the limits of the formal systems.

Part III

Model Theory and Incompleteness

Basics of model theory and decidability

In the last part of the lecture, we will discuss the basics of model theory, decidability of theories and the incompleteness of some theories. Model theory is a rather modern branch of logic that rapidly developed in the 1990s. As its name suggests, it studies the models of theories and answers (among others) questions like: what is the number of models of a theory T up to isomorphisms, is the theory T complete, etc.

The decidability of theories deals with the problem of whether we can algorithmically decide whether a sentence is provable in a theory T or not. If it is, the theory is called decidable.

Finally, we will discuss the so-called Gödel incompleteness theorems that state that once a theory is sufficiently strong, it is incomplete. We will also show that truth cannot be defined in logic.

Model Theory

We have actually already seen some terms that are considered a part of model theory. For example, we know that two structures are elementarily equivalent if they satisfy the same sentences. We have also defined the theory of a structure $\mathcal{A}$ – $\mathrm{Th}(\mathcal{A})$ as the set of all sentences valid in $\mathcal{A}$. This set is also a theory (it is a set of formulas). Moreover, $\mathrm{Th}(\mathcal{A})$ is always a complete theory, if $\mathcal{A} \vDash T$, then $\mathrm{Th}(\mathcal{A})$ is a simple (complete) extension of T, and if $\mathcal{A} \vDash T$ and T is complete, then $\mathrm{Th}(\mathcal{A})$ is equivalent to T, i.e. $\mathrm{Th}(\mathcal{A}) = \theta^L(T)$. We can also easily see that for every models $\mathcal{A}, \mathcal{B}$ of a theory T, $\mathcal{A} \equiv \mathcal{B}$ if and only if $\mathrm{Th}(\mathcal{A})$ and $\mathrm{Th}(\mathcal{B})$ are equivalent theories.

Before we discuss some of the ideas of model theory further, we define some common algebraic theories.

1. the *theory of groups* in the language $L = \langle +, -, 0 \rangle$ with equality has axioms

$$\begin{aligned} x + (y + z) &= (x + y) + z \\ 0 + x &= x = x + 0 \\ x + (-x) &= 0 = (-x) + x \end{aligned}$$

2. the *theory of Abelian groups* has moreover the axiom $x + y = y + x$

3. the *theory of rings* in $L = \langle +, -, \cdot, 0, 1 \rangle$ has moreover axioms

$$\begin{aligned} &1 \cdot x = x = x \cdot 1 \\ &x \cdot (y \cdot z) = (x \cdot y) \cdot z \\ &x \cdot (y + z) = x \cdot y + x \cdot z \\ &(x + y) \cdot z = x \cdot z + y \cdot z \end{aligned}$$

4. the *theory of commutative rings* has moreover the axiom $x \cdot y = y \cdot x$, and

5. the *theory of fields* in the same language has additional axioms

$$\begin{aligned} &x \neq 0 \to (\exists y)(x \cdot y = 1) \\ &0 \neq 1 \end{aligned}$$

Another important theory is the *theory of dense linear orders* $DeLO^*$ of the language $L = \langle \leq \rangle$ with equality that has axioms

$$\begin{aligned} &x \leq x \\ &x \leq y \wedge y \leq x \to x = y \\ &x \leq y \wedge y \leq z \to x \leq z \\ &x \leq y \vee y \leq x \\ &x < y \to (\exists x)(x < z \wedge z < y) \\ &(\exists x)(\exists y)(x \neq y) \end{aligned}$$

where $x < y$ means $x \leq y \wedge x \neq y$. Let φ be the sentence $(\exists x)(\forall y)(x \leq y)$ and let ψ be the sentence $(\exists x)(\forall y)(y \leq x)$. We will see that $DeLO^*$ has the following four simple complete extensions (and none other):

$$\begin{aligned} DeLO &= DeLO^* \cup \{\neg\varphi, \neg\psi\} \\ DeLO^+ &= DeLO^* \cup \{\neg\varphi, \psi\} \\ DeLO^- &= DeLO^* \cup \{\varphi, \neg\psi\} \\ DeLO^\pm &= DeLO^* \cup \{\varphi, \psi\} \end{aligned}$$

We already know the Lövenheim-Skolem theorem – let T be a consistent theory of a countable language L. If L is without equality, then T has a countably infinite model. If L is with equality, then T has a model that is countable (finite or countably infinite).

A corollary of the theorem is that for every structure $\mathcal{A}$ of a countable language without equality, there exists a countably infinite structure $\mathcal{B}$ such that $\mathcal{A} \equiv \mathcal{B}$. Why? We know that $\mathrm{Th}(\mathcal{A})$ is consistent since it has a model $\mathcal{A}$ and by the Lövenheim-Skolem theorem it has a countably infinite model $\mathcal{B}$. Since $\mathrm{Th}(\mathcal{A})$ is complete $\mathcal{A}$ must be elementarily equivalent to $\mathcal{B}$.

Similarly for theories in language with equality, but we additionally need the assumption that $\mathcal{A}$ is infinite: for every infinite structure $\mathcal{A}$ of a countable language without equality, there exists a countably infinite structure $\mathcal{B}$ such that $\mathcal{A} \equiv \mathcal{B}$. The proof is similar to the one above. This time we also know that the sentence "there are exactly n elements"

is not valid in $\mathcal{A}$ for any n and therefore also not in $\mathcal{B}$ and therefore $\mathcal{B}$ must be infinite.

These corollaries are quite strong, as is demonstrated in the following proof of existence of a countable algebraically closed field. We say that a field $\mathcal{A}$ is *algebraically closed* if every polynomial (of non-zero degree) has a root in $\mathcal{A}$, i.e. we have for each $n \in \mathbb{N}$

$$\mathcal{A} \vDash (\forall x_{n-1}) \dots (\forall x_0)(\exists y)(y^n + x_{n-1} \cdot y^{n-1} + \dots + x_1 \cdot y + x_0 = 0)\,.$$

For example, the field $\mathbb{C} = \langle \mathbb{C}, +, -, \cdot, 0, 1 \rangle$ of complex numbers is algebraically closed, while the fields of real numbers $\mathbb{R}$ or rational numbers $\mathbb{Q}$ are not. But since $\mathbb{C}$ is closed and infinite, by the previous corollary, there is a countable structure elementarily equivalent to $\mathbb{C}$ and therefore also algebraically closed.

We now want to introduce the ω-categorical criterium of completeness. To this end, we first need to define an isomorphism of structures. Let $\mathcal{A}$ and $\mathcal{B}$ be structures of a language $L = \langle \mathcal{F}, \mathcal{R} \rangle$. A bijection $h : A \to B$ is a *isomorphism of structures $\mathcal{A}$ and $\mathcal{B}$* if both $h(f^A(a_1, \dots, a_n)) = f^b(h(a_1), \dots, h(a_n))$ for every n-ary function symbol $f \in \mathcal{F}$ and every $a_1, \dots, a_n \in A$, and $R^A(a_1, \dots, a_n) \Leftrightarrow R^B(h(a_1), \dots, h(a_n))$ for every n-ary relation symbol $R \in \mathcal{R}$ and every $a_1, \dots, a_n \in A$. We say that $\mathcal{A}$ and $\mathcal{B}$ are isomorphic (via h) ($A \simeq B$) if there is an isomorphism h of $\mathcal{A}$ and $\mathcal{B}$. We also say that $\mathcal{A}$ is isomorphic with $\mathcal{B}$. An automorphism of a structure $\mathcal{A}$ is an isomorphism of $\mathcal{A}$ with $\mathcal{A}$.

It is easy to show that isomorphisms preserve semantics, i.e. let $\mathcal{A}$ and $\mathcal{B}$ be structures of a language $L = \langle \mathcal{F}, \mathcal{R} \rangle$, a bijection $h : A \to B$ is an isomorphism of $\mathcal{A}$ and $\mathcal{B}$ if and only if both $h(t^A[e]) = t^B[he]$ for every term t and $e : \mathrm{Var} \to A$, and $\mathcal{A} \vDash \varphi[e] \Leftrightarrow \mathcal{B} \vDash \varphi[he]$ for every formula φ and $e : \mathrm{Var} \to A$. This gives us a corollary that for every structures $\mathcal{A}$ and $\mathcal{B}$ of the same language $\mathcal{A} \simeq \mathcal{B} \Rightarrow \mathcal{A} \equiv \mathcal{B}$, i.e. if two structures are isomorphic, they are also elementarily equivalent. The other implication does not generally hold, as we can have two elementarily equivalent structures (e.g. the field of complex numbers $\mathbb{C}$ and the countable algebraically closed field from the previous example) that are elementarily equivalent and not isomorphic (because they have different cardinality). However, the following lemma shows that the implication holds for finite structures in a language with equality.

Lemma 11. *For every finite structures $\mathcal{A}$ and $\mathcal{B}$ of a language with equality $\mathcal{A} \equiv \mathcal{B} \Rightarrow \mathcal{A} \simeq \mathcal{B}$.*

Proof. First of all, we can see that $|A| = |B|$ as the sentence "there are exactly n elements can be expressed in L". For a finite language L, we can write a formula φ that defines $\mathcal{A}$ up to isomorphism.[17] This formula holds both in $\mathcal{A}$ and $\mathcal{B}$ since they are elementarily equivalent and therefore they are also isomorphic.

For an infinite language L: there is only a finite number of bijections between A and B; assume, for contradiction, that none of them is an isomorphism. For each bijection h_i choose a relation R_i in L that is not preserved in this bijection. Let $L' \subseteq L$ is the language that contains

[17] The sentence says "there are exactly n elements $a_1, \dots, a_n$ satisfying exactly those atomic formulas on function values and relations that are valid in the structure $\mathcal{A}$".

only these relations. Obviously L' is finite. Then the reducts of $\mathcal{A}$ and $\mathcal{B}$ to L' are elementarily equivalent and therefore isomorphic (by the previous paragraph) which is a contradiction. □

A corollary of this lemma is that if a complete theory T in a language with equality has a finite model, then all models of T are isomorphic.

An *isomorphic spectrum* of a theory T is given by the number $I(\kappa, T)$ of mutually non-isomorphic models of T for every cardinality κ. A *theory T is κ-categorical* if it has exactly one model of cardinality κ (up to isomorphism), i.e. $I(\kappa, T) = 1$.

For example, the theory *DeLO* is ω-categorical. Let $\mathcal{A}, \mathcal{B} \vDash DeLO$, with $A = \{a_i\}_{i\in\mathbb{N}}$ and $B = \{b_i\}_{i\in\mathbb{N}}$. By induction on n we can find injective $h_n \subseteq h_{n+1} \subset A \times B$ preserving the ordering such that $\{a_i\}_{i<n} \subseteq \mathrm{dom}(h_n)$ and $\{b_i\}_{i<n} \subseteq \mathrm{rng}(h_n)$. Then $\mathcal{A} \simeq \mathcal{B}$ via $h = \cup h_n$.

Similarly, we obtain that $\mathcal{A} = \langle \mathbb{Q}, \leq \rangle, \mathcal{A} \upharpoonright [0,1), \mathcal{A} \upharpoonright (0,1]$, and $\mathcal{A} \upharpoonright [0,1]$ are up to isomorphism all countable models of $DeLO^*$. Therefore

$$I(\kappa, DeLO^*) = \begin{cases} 0 & \text{if } \kappa \in \mathbb{N}\,, \\ 4 & \text{if } \kappa = \omega\,. \end{cases}$$

This finally leads to the ω-categorical criterium of completeness (similar criteria also hold for cardinalities bigger than ω):

Theorem 19. *Let L be a finite or countably infinite language.*

1. *If a theory T in L without equality is ω-categorical, it is complete.*
2. *If a theory T in L with equality is ω-categorical and without finite models, it is complete.*

Proof. Every model of T is elementarily equivalent with some countably infinite model of T, but such model is unique up to isomorphism, therefore all models of T are elementarily equivalent and T is complete. □

The fact that isomorphisms preserve semantics also gives us a simple condition for sets that can be defined in structures. The set defined by a formula $\varphi(\bar{x}, \bar{y})$ in structure $\mathcal{A}$ from parameters $\bar{b} \in \mathcal{A}^{|\bar{b}|}$ is $\varphi^{\mathcal{A},\bar{b}}(\bar{x}, \bar{y}) = \{\bar{a} \in \mathcal{A}^{|\bar{x}|} | \mathcal{A} \vDash \varphi[e(\bar{x}/\bar{a}, \bar{y}/\bar{b}\}$. A set $D \subseteq \mathcal{A}^n$ is definable in the structure $\mathcal{A}$ from parameters $\bar{b} \in \mathcal{A}^{|\bar{y}|}$ if there is a formula $\varphi(\bar{x}, \bar{y})$ such that $D = \varphi^{\mathcal{A},\bar{b}}(\bar{x}, \bar{y})$. It is easy to show that definable sets map to themselves in automorphism.

Lemma 12. *Let $D \subseteq A^n$ is a set definable in the structure $\mathcal{A}$ from parameters $\bar{b}$ and h is an automorphism on $\mathcal{A}$ identical on $\bar{b}$. Then $h[D] = D$.*

Proof. Let $D = \varphi^{\mathcal{A},\bar{b}}(\bar{x}, \bar{y})$, then for every $\bar{a} \in A^n$ we have $\bar{a} \in D \Leftrightarrow A \vDash \varphi[e(\bar{x}/\bar{a}, \bar{y}/\bar{b})] \Leftrightarrow \varphi[e \circ h(\bar{x}/\bar{a}, \bar{y}/\bar{b})] \Leftrightarrow \varphi[e(\bar{x}/h(\bar{a}), \bar{y}/h(\bar{b}))] \Leftrightarrow \varphi[e(\bar{x}/h(\bar{a}), \bar{y}/\bar{b})] \Leftrightarrow h(\bar{a}) \in D$. □

Decidability

Now, we would like to know if a given problem is algorithmically decidable (e.g. if a sentence is provable in a theory). We can formally

define the notion of algorithm, for example using Turing machines. Decision problems can then be encoded into sets of natural numbers corresponding to the positive instances (with answer "yes"). For example, $SAT = \{\ulcorner\varphi\urcorner | \varphi \text{ is a satisfiable formula }\}$, where $\ulcorner\varphi\urcorner$ is the natural number representing the formula φ. We say that a set $A \subseteq \mathbb{N}$ *is recursive* if there is an algorithm that for every input $x \in \mathbb{N}$ halts (stops) and correctly tells whether or not $x \in A$. We say that such an algorithm *decides* $x \in A$. We say that a *set* $A \subseteq \mathbb{N}$ *is recursively enumerable* if there is an algorithm that for every input $x \in \mathbb{N}$ halts if and only if $x \in A$. We say that such an algorithm *recognizes* $x \in A$. Equivalently, A is recursively enumerable if there is an algorithm that generates (enumerates) all elements of A.

Lemma 13. *For every $A \subseteq \mathbb{N}$ A is recursive if and only if both A and $\bar{A}$ are recursively enumerable.*

Proof. ($\Rightarrow$) If A is recursive, there is an algorithm that always halts and returns whether $x \in A$. We can make an algorithm that halts only if $x \in A$ (check if $x \in A$, if it is, halt, otherwise enter an infinite loop). Therefore A is recursively enumerable. Similarly for $\bar{A}$.

($\Leftarrow$) If both A and $\bar{A}$ are recursively enumerable, there are two algorithms P_1 and P_2 such that P_1 recognizes $x \in A$ and P_2 recognizes $x \in \bar{A}$. We can start both these algorithms in parallel (make one step of P_1, then one step of P_2, then another step of P_1 and so on). For each $x \in \mathbb{N}$, one of these algorithms eventually halts. If P_1 halts first $x \in \mathbb{N}$, otherwise $x \notin \mathbb{N}$. Therefore A (and also $\bar{A}$) is recursive. □

In logic, we are especially interested if a given theory is algorithmically decidable. We assume a recursive language L. A *theory T of L is decidable*, if $\mathrm{Thm}(T)$ is recursive, otherwise T is *undecidable*. This means that a theory is decidable if there is an algorithm that for each sentence φ decides whether $T \vdash \varphi$ or not.

For every theory T of L with recursively enumerable axioms, $\mathrm{Thm}(T)$ is recursively enumerable – the construction of systematic tableau from T with root $F\varphi$ assumes a given enumeration of axioms of T. Since T has recursively enumerable axioms, the construction provides an algorithm that recognizes $T \vdash \varphi$.

If the theory T from the previous paragraph is additionally complete, then $\mathrm{Thm}(T)$ is recursive, i.e. T is decidable. Similarly to the proof of the lemma above, we can start the construction of systematic tableaux from T with roots $F\varphi$ and $T\varphi$. Since T is complete, then $T \nvdash \varphi$ if and only if $T \vdash \neg\varphi$ for every sentence φ. The construction thus provides an algorithm that decides $T \vdash \varphi$.

We now know that complete theories (with recursively enumerable axioms) are decidable, however, completeness is a rather strong assumption. In fact, it is enough, if all the simple complete extensions of a theory T are recursively enumerable, i.e. if there is an algorithm $\alpha(i, j)$ that generates the i-th axiom of th j-th extension (in some enumeration) or announces that such an axiom does not exist.

So, if a theory T has recursively enumerable axioms and the set of all simple complete extensions of T is recursively enumerable, then T

is decidable. By the previous observation there is an algorithm that recognizes $T \vdash \varphi$ (the construction of systematic tableau). On the other hand, if $T \nvdash \varphi$ then $T' \vdash \neg\varphi$ for some simple complete extension T' of T. We can construct the systematic tableaux with root $T\varphi$ from all the extensions in parallel. In the i-th step we construct tableaux up to level i for the first i extensions. One of these tableaux is eventually a proof of $T' \vdash \neg\varphi$ and therefore this construction recognizes $T \nvdash \varphi$. Therefore, T is decidable.

There are many theories that are decidable, although they are not complete, for example:

1. the *theory of pure equality* with no axioms in language $L = \langle\rangle$ with equality,
2. the *theory of unary predicate* with no axioms in language $L = \langle U\rangle$ with equality, where U is a unary relation symbol,
3. the theory of dense linear orders $DeLO^*$,
4. the theory of algebraically closed fields in $L = \langle +, =, \cdot, 0, 1\rangle$ with the axioms of fields and moreover the axioms for all $n \geq 1$

$$(\forall x_{n-1}) \dots (\forall x_0)(\exists y)(y^n + x_{n-1} \cdot y^{n-1} + \dots + x_1 \cdot y + x_0 = 0)\,,$$

5. the theory of Abelian groups, and
6. the theory of Boolean algebras.

Axiomatizability

Another interesting question is whether we can effectively (i.e. algorithmically, recursively) describe common mathematical structures. We say that a class $K \subseteq M(L)$ is *recursively axiomatizable* if there is a recursive theory T of language L with $M(T) = K$. A *theory is recursively axiomatizable* if $M(T)$ is recursively axiomatizable, i.e. if there is an equivalent recursive theory.

For example, for every finite structure $\mathcal{A}$ of a finite language with equality the theory $\mathrm{Th}(\mathcal{A})$ is recursively axiomatizable. Thus, $\mathrm{Th}(\mathcal{A})$ is decidable. Let $A = \{a_1, \dots, a_n\}$, $\mathrm{Th}(\mathcal{A})$ can be axiomatized by a single sentence that describes $\mathcal{A}$. The sentence is of the form "there are exactly n elements $a_1, \dots, a_n$ satisfying exactly those atomic formulas on function values and relations that are valid in the structure $\mathcal{A}$".

For example the following structures $\mathcal{A}$ have recursively axiomatizable $\mathrm{Th}(\mathcal{A})$:

1. $\langle \mathbb{Z}, \leq\rangle$ by the theory of discrete linear orderings,
2. $\langle \mathbb{Q}, \leq\rangle$ by the theory of dense linear ordering without ends ($DeLO$),
3. $\langle \mathbb{N}, S, 0\rangle$ by the theory of successor with zero,
4. $\langle \mathbb{N}, S, +, 0\rangle$ by so called Presburger arithmetic,
5. $\langle \mathbb{R}, +, -, \cdot, 0, 1\rangle$ by the theory of real closed fields, and

6. $\langle \mathbb{C}, +, -, \cdot, 0, 1 \rangle$ by the theory of algebraically closed fields with characteristic 0.

By the previous observation, for all the above structures $\mathcal{A}$, the theory $\mathrm{Th}(\mathcal{A})$ is decidable.

Apart from recursive axiomatizability, we may also be interested if a class of structures can be axiomatized by theories with other properties. For example, a *class $K \subseteq M(L)$ is openly axiomatizable* if there is an open theory T such that $M(T) = K$. A *theory is openly axiomatizable* if $M(T)$ is openly axiomatizable. We already know that if T is openly axiomatizable then every substructure of a model of T is also a model of T. The other implication also holds and gives a useful criterium to test if a theory is openly axiomatizable. For example, the *DeLO* theory is not openly axiomatizable, because a finite substructure of its model is not a model of *DeLO*.

Another interesting case is finite axiomatizability. A class of structures $K \subseteq M(L)$ *is finitely axiomatizable* if there is a finite theory T such that $M(T) = K$. Similarly, *a theory T is finitely axiomatizable* if $M(T)$ is finitely axiomatizable. We again have a criterium to test if a class is finitely axiomatizable.

Theorem 20. *Let $K \subseteq M(L), \bar{K} = M(L) \setminus K$, where L is a language. Then K is finitely axiomatizable if and only if both K and $\bar{K}$ are axiomatizable.*

Proof. ($\Rightarrow$) If T is a finite axiomatization of K then the theory with the single sentence $\bigvee_{\varphi \in T} \neg\varphi$ axiomatizes K.

($\Leftarrow$) Let T, S be theories of L such that $M(T) = K, M(S) = \bar{K}$. Then $M(S \cup T) = M(T) \cap M(S) = \emptyset$. By compactness, there are finite theories $T' \subseteq T$ and $S' \subseteq S$ such that $M(S' \cup T') = \emptyset$. Then $M(T) \subseteq M(T') \subseteq \overline{M(S')} \subseteq \overline{M(S)} = M(T)$ and therefore $M(T) = M(T')$. □

There are actually classes of structures that are not axiomatizable at all. The compactness theorem gives us the following lemma:

Lemma 14. *If a theory T has for each $n \in \mathbb{N}$ at least one n-element model, it has an infinite model.*

Proof. For languages without equality this is true by the Lövenheim-Skolem theorem. In languages with equality, we create an extension T' of T such that $T' = T \cup \{c_i \neq c_j | i \neq j\}$ in a language with countably many new constant symbols c_i. Every finite part of T' has a model by the assumption and therefore T' also has a model according to the compactness theorem. This model must be infinite, and its reduct to the original language is the desired infinite model of T. □

However, that means that if a theory has for each $n \in \mathbb{N}$ at least one n-element model, the class of its finite models is not axiomatizable, e.g. the class of finite groups of finite fields are not axiomatizable in first-order logic.

Incompleteness

We will now show the limits of all formal logical systems. In the beginning of the 20^{th} century, Hilbert proposed the so-called Hilbert's program, whose goal was to ground all existing mathematical theories to a finite, complete set of axioms. We will show that such a program is unattainable for key areas of mathematics (as shown by Gödel in 1931). More specifically, we will show that arithmetic on natural numbers represented by the structure $\underline{\mathbb{N}} = \langle \mathbb{N}, S, +, \cdot, 0, \leq \rangle$ is not recursively axiomatizable (this is a corollary of Gödel's incompleteness theorem).

We will start by introducing a first approximation of $\mathrm{Th}(\underline{\mathbb{N}})$. So-called *Robinson arithmetic* Q has finitely many axioms:

1. $S(x) \neq 0$
2. $S(x) = S(y) \to x = y$
3. $x + 0 = x$
4. $x + S(y) = S(x + y)$
5. $x \cdot 0 = 0$
6. $x \cdot S(x) = x \cdot y + x$
7. $x \neq 0 \to (\exists y)(x = S(y))$
8. $x \leq y \leftrightarrow (\exists z)(z + x = y)$

Robinson arithmetic Q is rather weak – it does not even prove the commutativity or associativity of $+, \cdot$ or the transitivity of $\leq$. On the other hand it suffices to prove for example existential sentences on numerals that are true in $\underline{\mathbb{N}}$. For example, for $\varphi(x, y)$ in the form $(\exists z)(x + z = y)$ it is $Q \vdash \varphi(\underline{1}, \underline{2})$, where $\underline{1} = S(0)$ and $\underline{2} = S(S(0))$.

A stronger approximation of $\mathrm{Th}(\underline{\mathbb{N}})$ is an extension of Robinson arithmetic called *Peano arithmetic* or *PA*. It has all the axioms of Robinson arithmetic plus the *scheme of induction* – for every formula $\varphi(x, \vec{y})$ of L the axiom

$$(\varphi(0, \vec{y}) \wedge (\forall x)(\varphi(x, \vec{y}) \to \varphi(S(x), \vec{y}))) \to (\forall x)\varphi(x, \vec{y})\,.$$

Peano arithmetic is a quite common approximation of $\mathrm{Th}(\underline{\mathbb{N}})$. It proves all the basic properties that are true in $\underline{\mathbb{N}}$, but it is still incomplete, i.e. there are sentences that are true in $\underline{\mathbb{N}}$ but not independent in *PA*.

One could hope that there is another, even stronger, extension of Robinson arithmetic that recursively axiomatizes $\mathrm{Th}(\underline{\mathbb{N}})$, but we will show that for every consistent recursively axiomatized extension T of Robinson arithmetic there is a sentence true in $\underline{\mathbb{N}}$ and unprovable in T (this is the first Gödel's incompleteness theorem).

Before we get to the incompleteness theorem, we will show another rather surprising result – there is no algorithm that can decide whether a given sentence is logically true. The proof of this proposition is based on the solution to the 10^{th} Hilbert's problem that asks to find an algorithm that determines in finitely many steps whether a given Diophantine equation is an arbitrary number of variables and with integer coefficients has an integer solution (*Diophantine equations* are of the form $p(x_1, \ldots, x_n) = 0$ for some polynomial p with integer coefficients). The solution to this problem was found in 1970, and the result

is that the problem of existence of integer solutions to a given Diophantine equation with integer coefficients in algorithmically undecidable. (We will not prove it.) But that also means that there is no algorithm to determine for given polynomials $p(x_1, \ldots, x_n), q(x_1, \ldots, x_n)$ with natural coefficients whether

$$\underline{\mathbb{N}} \vDash (\exists x_1) \ldots (\exists x_n)(p(x_1, \ldots, x_n) = q(x_1, \ldots, x_n)) .$$

Assume now that there is an algorithm that can decide the logical truth of sentences. In particular, such an algorithm could decide the logical truth of the sentence $\psi \to \varphi$, where ψ is the conjunction of closures of axioms of the Robinson arithmetic and φ is the formula $(\exists x_1) \ldots (\exists x_n)(p(x_1, \ldots, x_n) = q(x_1, \ldots, x_n))$ for some polynomials p and q with natural coefficients, i.e. the algorithm decides whether $\vDash \psi \to \varphi$, according to the completeness theorem, this is equivalent to $\vdash \psi \to \varphi$, which is again equivalent (by the deduction theorem) to $Q \vdash \varphi$. Because Q proves existential sentences on numerals that are true in $\underline{\mathbb{N}}$,[18] the algorithm also decides $\underline{\mathbb{N}} \vDash \varphi$ which is not possible by the answer to the 10^{th} Hilbert's problem.

[18] More precisely, we have for every existential formula $\varphi(x_1, \ldots, x_n)$ in arithmetic $Q \vdash \varphi(x_1/\underline{a_1}, \ldots, x_n/\underline{a_n})$ if and only if $\underline{\mathbb{N}} \vDash \varphi[e(x_1/a_1, \ldots, x_n/a_n)]$

The proof of Gödel's incompleteness theorem is based on so-called arithmetization and on self-reference. Arithmetization shows that the syntactic notions of proofs can be expressed as formulas in arithmetic. We start by assigning numbers to all symbols of the language and then create a function that assigns values to sequences (i.e. terms, formulas, and so on). In this way, all the finite objects (variables, terms, formulas, tableaux, proofs) can be assigned a natural number (code). All these codes can be computed recursively and are reversible (i.e. we can recursively compute the original object from the code). Let $\ulcorner \varphi \urcorner$ denote the code assigned to formula φ and let $\underline{\varphi}$ denote the numeral representing $\ulcorner \varphi \urcorner$.

If T has a recursive axiomatization then the predicate $\mathrm{Prf}_T \subseteq \mathbb{N}^2$ defined as

$$\mathrm{Prf}_T(x, y) \Leftrightarrow y \text{ is a proof of } x$$

is recursive. We can easily create an algorithm that first decodes x and y and then checks if y is indeed a tableau proof of x – axioms of T are recursive, thus we can check if a given entry in a tableau is an axiom of T, and the atomic tableaux can also be checked recursively. If, moreover, T extends Robinson arithmetic Q, the relation Prf_T can be represented by a formula Prf_T such that for every $x, y \in \mathbb{N}$, $Q \vdash Prf_T(\underline{x}, \underline{y})$ if $\mathrm{Prf}_T(x, y)$ and $Q \vdash \neg Prf_T(\underline{x}, \underline{y})$ otherwise.

The formula $Prf_T(x, y)$ expresses that y is a proof of x in T. Therefore, the formula $(\exists y) Prf_T(x, y)$ expresses that x is provable in T. If $T \vdash \varphi$ then $\underline{\mathbb{N}} \vDash (\exists y) Prf_T(\underline{x}, y)$ and moreover $T \vdash (\exists y) Prf_T(\underline{x}, y)$.

The other important part of the proof of the incompleteness theorem is self-reference. A self-referential sentence can mention itself. For example, "This sentence has 24 letters" is an example of a self-reference. However, such direct self-reference is not always available in formal systems. On the other hand, direct reference to another sentence is often available, like in "The following sentence has 32 letters 'The following sentence has 32 letters' ". However, it is not self-referential. We

can use direct reference to create self-reference, like in "The following sentence written once more and then once again between quotation marks has 116 letters 'The following sentence written once more and then once again between quotation marks has 116 letters' ".

Under some assumptions, there is a self-referential formula ψ that expresses "this formula satisfies condition φ" for any condition φ, as demonstrated by the next theorem.

Theorem 21 (Fixed-point theorem)**.** *Let T be a consistent extension of Robinson arithmetic. For every formula $\varphi(x)$ there is a sentence ψ such that $T \vdash \psi \leftrightarrow \varphi(\underline{\psi})$.*

Proof. We will only show the idea of the proof. Consider the doubling function d such that for every formula $\chi(x)$: $d(\lceil\chi(x)\rceil) = \lceil\chi(\underline{\chi(x)})\rceil$. It can be shown that such a formula is expressible in T; assume it is by a term also called d.

Then, for every formula $\chi(x)$ of the language of the theory T holds that

$$T \vdash d(\underline{\chi(x)}) = \underline{\chi(\underline{\chi(x)})}\,.$$

We can now take ψ as $\varphi(d(\underline{\varphi(d(x))}))$. It suffices to verify that $T \vdash d(\underline{\varphi(d(x))}) = \underline{\psi}$. It follows from the above equation for $\chi(x)$ being $\varphi(d(x))$ since in this case

$$T \vdash d(\underline{\varphi(d(x))}) = \underline{\varphi(d(\underline{\varphi(d(x))}))}\,.$$

□

A consequence of the previous theorem is that we cannot define truth in any consistent extension of Robinson arithmetic. We say that a formula $\tau(x)$ *defines truth* in theory T of arithmetical language if for every sentence φ it holds that $T \vdash \varphi \leftrightarrow \tau(\underline{\varphi})$. In the proof we will use the principle of self-reference and the liar's paradox. The sentence φ expresses "This sentence is not true in T".

Theorem 22 (Undefinability of truth)**.** *Let T be a consistent extension of Robinson arithmetic. Then T has no definition of truth.*

Proof. Assume for contradiction that there is a formula $\tau(x)$ that defines truth in T. By the fixed-point theorem for $\neg\tau(x)$ there is a sentence φ such that $T \vdash \varphi \leftrightarrow \neg\tau(\underline{\varphi})$. As τ defines truth this means that $T \vdash \varphi \leftrightarrow \neg\varphi$ which is not possible in a consistent theory. □

We can finally prove Gödel's first incompleteness theorem, which shows that whatever we do, there is not a consistent recursively axiomatizable extension of Robinson arithmetic that is equivalent to $\mathrm{Th}(\underline{\mathbb{N}})$.

Theorem 23 (Gödel's First Incompleteness Theorem)**.** *For every consistent axiomatizable extension T of Robinson arithmetic there is a sentence true in $\underline{\mathbb{N}}$ and unprovable in T.*

Proof. Let $\varphi(x)$ be $\neg(\exists y)Prf_T(x, y)$, i.e. it says x is not provable in T. By the fixed point theorem there is a sentence ψ_T such that $T \vdash \psi_T \leftrightarrow$

$\neg(\exists y)Prf_T(\underline{\psi_T}, y)$. Such a ψ_T is equivalent to a formula that expresses that ψ_T is not provable in T, and the equivalence holds both in T and $\underline{\mathbb{N}}$.

First, we show that ψ_T is not provable in T. If it is, i.e. if $T \vdash \psi_T$, then ψ_T is contradictory in $\underline{\mathbb{N}}$, therefore $\underline{\mathbb{N}} \models (\exists y)Prf_T(\underline{\psi_T}, y)$ and moreover $T \vdash (\exists y)Prf_T(\underline{\psi_T}, y)$. Thus also $T \vdash \neg\psi_T$, which is impossible since T is consistent.

Now we show ψ_T is true in $\underline{\mathbb{N}}$. If not, i.e. $\underline{\mathbb{N}} \models \neg\psi_T$, then $\underline{\mathbb{N}} \models (\exists y)Prf_T(\underline{\psi_T}, y)$, therefore $T \vdash \psi_T$ which is not possible as shown in the previous paragraph. □

If we additionally add the assumption that $\underline{\mathbb{N}} \models T$, then T is incomplete, because if it were complete then $T \vdash \neg\psi_T$ and thus $\underline{\mathbb{N}} \models \neg\psi_T$, which contradicts $\underline{\mathbb{N}} \models \psi_T$. This also shows that $\mathrm{Th}(\underline{\mathbb{N}})$ is not recursively axiomatizable, because $\mathrm{Th}(\underline{\mathbb{N}})$ is a consistent extension of Robinson arithmetic and has a model $\underline{\mathbb{N}}$. Suppose $\mathrm{Th}(\underline{\mathbb{N}})$ is recursively axiomatizable. However, by the previous discussion, $\mathrm{Th}(\underline{\mathbb{N}})$ is incomplete, which is a contradiction with $\mathrm{Th}(\underline{\mathbb{N}})$ being complete (as $\mathrm{Th}(\mathcal{A})$ is complete for any structure $\mathcal{A}$).

Gödel's theorem was strengthened by Rosser. Rosser's theorem states that every consistent recursively axiomatized extension T of Robinson arithmetic has an independent sentence and is thus incomplete. Thus the assumption $\underline{\mathbb{N}} \models T$ was not needed above.

Gödel's second incompleteness theorem shows that the consistency of a theory cannot be proved in the theory itself. More formally, let Con_T denote the sentence $\neg(\exists y)Prf_T(\underline{0 = 1}, y)$. We have $\underline{\mathbb{N}} \models Con_T \Leftrightarrow T \nvdash 0 = \underline{1}$. Thus, Con_T represents "T is consistent".

Theorem 24 (Gödel's Second Incompleteness Theorem)**.** *For every consistent recursively axiomatized extension T of Peano arithmetic it holds that Con_T is unprovable in T.*

Proof. Let ψ_T be the sentence from the proof of the first Gödel's incompleteness theorem, i.e. "this is not provable in T". We have shown in the proof that if T is consistent, then ψ_T is not provable in T, i.e. $Con_T \rightarrow \psi_T$. If T is an extension of Peano arithmetic the proof can be formalized within the theory of T itself. Hence $T \vdash Con_T \rightarrow \psi_T$. Since T is consistent, we have that $T \nvdash \psi_T$, therefore if follows that $T \nvdash Con_T$. □

Because Peano arithmetic is a recursively axiomatizable extension of itself and Con_T is not provable in PA, there must be a model $\mathcal{A}$ of Peano arithmetic such that $\mathcal{A} \models (\exists y)Prf_{PA}(\underline{0 = 1}, y)$. The model $\mathcal{A}$ must be a so-called non-standard model of arithmetic and the witness for the existence in the sentence must be some non-standard element of the model (other than a value of a numeral).

Similarly, there is also a consistent recursively axiomatizable extension T of Peano arithmetic such that $T \vdash \neg Con_T$. If we choose $T = PA \cup \{\neg Con_{PA}\}$, then T is consistent, because $T \nvdash Con_{PA}$. Moreover, $T \vdash \neg Con_{PA}$, i.e. T proves the inconsistency of $PA \subseteq T$, and thus

also $T \vdash \neg Con_T$. Again, all models of T must be non-standard, i.e. $\underline{\mathbb{N}}$ is not a model of T.

Gödel's second incompleteness theorem actually states that a recursively axiomatized extension of Peano arithmetic cannot prove its own consistency, unless it is inconsistent. Since Zermelo-Frankel set theory (ZFC) is strong enough to define the required arithmetic notions, this also means that if ZFC is consistent then Con_{ZFC} is unprovable in ZFC. From a more philosophical point of view, this means that we actually cannot prove if modern mathematics (based on ZFC) is consistent, unless it is inconsistent.

This concludes the last part of the introductory lecture to logic. The goal of this part was to introduce more advanced parts of logic and provide some strong results that can be used to analyze a given theory – we provided useful criteria for completeness and decidability of theories. We have also shown the limits of formal systems, mostly represented by Gödel's incompleteness theorem and by the existence of classes of structures that are not axiomatizable in first-order logic.

Contents

Introduction 3

About these lecture notes 4

Preliminaries 5

I Propositional Logic 11

Propositional Formulas and Models 13

Syntax of Propositional Logic 13

Semantics of Propositional Logic 14

Normal Forms 16

Logical theories 18

Satisfiability of Propositional Formulas 20

Formal Proof Systems 23

Tableau Method 23

Soundness and Completeness 27

Hilbert systems 30

Resolution method 31

Linear resolution 35

LI-resolution 35

Resolution in Prolog 36

II First-Order Logic 39

Basic Syntax and Semantics 41

First-order formulas and theories 41

Semantics of first-order logic 44

The tableau method in first-order logic 51

Resolution in First-Order Logic 63

Prenex Normal Form 63

Skolem Variant 64

Herbrand Model 65

Resolution 67

Soundness and Completeness 69

III Model Theory and Incompleteness 71

Basics of model theory and decidability 73

Model Theory 73

Decidability 76

Axiomatizability 78

Incompleteness 80

List of Figures

1 The labeled ordered tree representing the formula $(p \wedge q) \rightarrow q$. 9

2 The formation tree representing the formula $p \wedge q \rightarrow \neg(p \vee s)$. 14

3 The atomic tableaux 24

4 Example tableau. The rectangles on the left show the atomic tableaux used. The version on the right removes the repeated entries. The symbol $\otimes$ denotes a contradictory branch. 25

5 Example tableau. Both left and middle branches are finished. The left one is also contradictory, while the middle one is noncontradictory. The right branch is not finished. 26

6 The resolution proof of $S \vdash_R \square$. 33

7 An example of a reduction tree. 34

8 A formation tree of the term $(S(x) + y) \cdot x$. 42

9 A formation tree of the formula $(\forall x)(x \cdot y \leq (S(x) + y) \cdot x)$. Moreover, $x \cdot y$ and $(S(x) + y) \cdot x$ are roots of formation trees of the terms included in the formula. 43

10 The atomic tableaux for logical connectives. In the tableau, φ, ψ are sentences and α are atomic sentences. 52

11 The atomic tableaux for quantifiers 53

12 Example tableau. The rectangles on the left show the atomic tableaux used. The version on the right removes the repeated entries that can be removed; the entry in the rectangle in the right tableau must be repeated. c is a new constant symbol where it first appears in the tableau, and in the last step, we chose c as the term in the atomic tableau for $T(\forall x)P(x)$. The symbol $\otimes$ denotes a contradictory branch. 54

13 An example of a systematic tableau. The left branch is contradictory, while the right one is noncontradictory and finished but infinite. 56

List of Tables

1 The semantics of logical connectives 14

List of Algorithms

Todo list