

A Compendium of First-Order Logic

Fabio Somenzi
University of Colorado at Boulder
Fabio@Colorado.EDU

August 25, 2023

Abstract

These notes provide a short introduction to first-order logic. More extensive and in-depth treatments are found in the cited works.

1 Introduction

Logic is the main mathematical tool employed in computer-aided verification. These notes give a quick introduction to first-order logic. They assume that the reader already has some familiarity with propositional logic, which many students encounter under the name of Boolean algebra. (Another name you may have seen used is “sentential logic.”) In particular, these notes assume familiarity with the standard Boolean connectives, their basic algebraic properties, and with truth tables. A short summary may be found in Appendix A. While notation varies considerably in books that cover Boolean algebra as a foundation for digital logic design, standard modern notation in books devoted to mathematical logic is \wedge for conjunction, \vee for disjunction, and \neg for negation. Our symbol for implication is \rightarrow , while the one for equivalence is \leftrightarrow .

When parsing logic expressions, it is customary to give highest precedence to negation, followed by conjunction, followed by the other operators in some order. Rather than specifying the order in full, we’ll use parentheses to avoid ambiguity. We’ll also use parentheses to avoid ambiguity when writing implications, which are neither commutative nor associative. (Some authors interpret $p \rightarrow q \rightarrow r$ as $p \rightarrow (q \rightarrow r)$, while others interpret it as $(p \rightarrow q) \rightarrow r$.)

First-order logic adds quantifiers to the set of Boolean connectives. While their symbols are standard, we have to spend a few words on their syntax. We shall write¹

$$\forall x . \exists y . y > x$$

¹Our punctuation is a restricted version of a system designed to minimize clutter by removing parentheses. See, for instance, [17, pp. 41–42]. Full parentheses suppression is not popular nowadays, whereas the version we adopt is quite common.

to state that for every x there is a y such that $y > x$. If we want to limit the scope of a quantifier, we use parentheses, as in this example:

$$(\forall x . P(x)) \vee (\forall x . Q(x)) ,$$

which means something quite different from

$$\forall x . P(x) \vee Q(x) .$$

The order of quantifiers matters: $\forall x . \exists y . y = x + 1$ says something quite different from $\exists y . \forall x . y = x + 1$. A loftier example can be drawn from topology. A map f between metric spaces (X, d_X) and (Y, d_Y) is continuous if, and only if,

$$\forall \varepsilon > 0 . \forall x \in X . \exists \delta > 0 . \forall x' \in X . d_X(x, x') < \delta \rightarrow d_Y(f(x), f(x')) < \varepsilon .$$

Changing the order of quantification gives the stronger condition for *uniform* continuity:

$$\forall \varepsilon > 0 . \exists \delta > 0 . \forall x \in X . \forall x' \in X . d_X(x, x') < \delta \rightarrow d_Y(f(x), f(x')) < \varepsilon .$$

It's a stronger condition because the same δ must work for all x .

2 Rigorous Arguments

Logic may be defined as the study of valid arguments. As a discipline, it has been practiced for about two and a half millennia, but we'll jump over most of that period and land in the early twentieth century, when mathematicians were interested in more rigorous ways to conduct their business. Specifically, a system of reasoning that allows one to deduce contradictory conclusions is, according to most, very unsatisfactory. The development of analysis, however, led to the development of set theory, which in turn led to the discovery of paradoxes like *Russell's Paradox*:

Consider the set of all sets that are not members of themselves. Is it a member of itself?

You may have come across this question in other formulations. For instance,

A set is *normal* if it does not contain itself. Is the set of all normal sets normal?

If we call this set R , we ask whether $R \in R$. Suppose it is; then, like all members of R , it does not contain itself; that is, $R \notin R$. The assumption has led to a contradiction; hence it is untenable. Suppose then R does not contain itself; that is, $R \notin R$. But then, by definition of R , it must be $R \in R$. We have reached a contradiction again. We can sum up the situation like this:

$$\text{Let } R = \{x \mid x \notin x\}, \text{ then } R \in R \Leftrightarrow R \notin R .$$

The conclusion we draw is that a theory of sets that allows us to talk about the set of all sets that are not members of themselves is not “safe” because it leads to contradictions. (More on paradoxes may be found in Section 7.)

Non-Euclidean geometries showed that *intuition* has limits. The proof of independence of the axiom of the parallels from the other axioms of Euclid’s geometry was also a conceptually important step, because it was a negative result: The proof that certain proofs are impossible within certain systems. Note that the set of Euclid’s axioms minus the parallel axiom is an *incomplete* theory: Neither the parallel axiom nor its negation follows from the other axioms.

This kind of developments pushed Mathematics towards the *axiomatic approach*. Hilbert proposed that the consistency of mathematical theories be established by *finitary methods* that could not lead to a contradiction.

3 Peano Arithmetic and Induction

As an example of axiomatic theory let’s take a look at *Peano’s arithmetic*, which is an axiomatization of the theory of the natural numbers. The number 0 and a unary operator s (successor) are given. The Peano axioms are:

1. $\forall x . s(x) \neq 0$.
2. $\forall x, y . x \neq y \rightarrow s(x) \neq s(y)$.
3. Induction: Let A be a subset of the natural numbers with the following properties: $0 \in A$, and $s(x) \in A$ whenever $x \in A$. Then A must be the set of all natural numbers.

The number 0 is explicitly mentioned in this theory; we may wonder what happened to the familiar numerals 1, 2, 3, Well, 1 is $s(0)$, 2 is $s(s(0))$, and so on. The s -based unary notation is quite cumbersome; hence we give numbers the familiar shorter names, which also make computation easier.

Hilbert wanted (more or less) to prove that no contradiction could be proved from the axioms above by the rules of first-order logic.

As an example of proof in Peano arithmetic, we show the existence of a predecessor for all natural numbers except 0. That is,

$$\forall x . x = 0 \vee \exists y . x = s(y) . \quad (1)$$

We prove by induction $\forall x . \varphi(x)$, where $\varphi(x)$ is $x = 0 \vee \exists y . x = s(y)$. For the base case we note that $\varphi(0)$ is $0 = 0$ disjoined with something else. Since equality is reflexive, $\varphi(0)$ clearly holds. For the inductive step, $\varphi(s(n))$ is $s(n) = 0 \vee \exists y . s(n) = s(y)$. Noting that $s(n) = 0$ is false because of the first axiom, we instantiate y as n to get $s(n) = s(n)$, which obviously holds.

Hilbert’s program called for the use of first-order logic. In reality, induction as shown above is a *second-order* axiom. Second-order means that we quantify over sets of elements of our universe of discourse [36], while in first-order logic

we can only quantify over individuals. Note that in the following formula A ranges over *sets* of natural numbers.

$$\forall A \subseteq \mathbb{N}. (0 \in A \wedge (\forall x \in A. s(x) \in A)) \rightarrow A = \mathbb{N} .$$

A first-order theory of arithmetic requires replacing the second-order axiom with an *axiom schema*, which generates infinitely many axioms: one axiom for each *definable* subset of the natural numbers. For instance, the set of numbers that are either 0 or have a predecessor is defined by the following formula:

$$\varphi_1(x) := x = 0 \vee \exists y. x = s(y) ,$$

obtained from (1) by removing the quantifier on x : number x is in the set if, and only if, it makes the formula true. The axiom schema generates the axiom

$$(\varphi_1(0) \wedge (\forall z. \varphi_1(z) \rightarrow \varphi_1(s(z)))) \rightarrow \forall x. \varphi_1(x) ,$$

which is what we used above in our proof that $\forall x. \varphi_1(x)$. Let's consider instead the formula

$$\varphi_2(x) := \exists y. x = s(y) .$$

The axiom schema supplies

$$(\varphi_2(0) \wedge (\forall z. \varphi_2(z) \rightarrow \varphi_2(s(z)))) \rightarrow \forall x. \varphi_2(x) .$$

In this case, $\varphi_2(0)$ does not hold, and in fact $\varphi_2(x)$ defines the set of all natural number except 0. As a final example, consider

$$\varphi_3(x) := x \neq s(x) ,$$

which says that x is not its own successor. Even such a basic fact as $\forall x. \varphi_3(x)$ needs a proof in the axiomatic approach; that is, it must be shown to be a consequence of the axioms. The induction axiom in this case is

$$(\varphi_3(0) \wedge (\forall y. \varphi_3(y) \rightarrow \varphi_3(s(y)))) \rightarrow \forall x. \varphi_3(x) .$$

We can easily check that $\varphi_3(0)$ follows from $\forall x. s(x) \neq 0$, which is our first axiom. With the help of the second axiom, we can then see that

$$\begin{aligned} \varphi_3(x) &\Leftrightarrow x \neq s(x) \\ &\Rightarrow s(x) \neq s(s(x)) \\ &\Leftrightarrow \varphi_3(s(x)) . \end{aligned}$$

If we want to talk about addition and multiplication, we need further first-order axioms that characterize them in terms of the successor function:

4. $\forall x. x + 0 = x.$
5. $\forall x, y. x + s(y) = s(x + y).$

$$6. \forall x. x \cdot 0 = 0.$$

$$7. \forall x, y. x \cdot s(y) = (x \cdot y) + x.$$

The resulting *Peano arithmetic* is a weaker theory than the second-order theory because induction only applies to definable subsets. Not all subsets of the natural numbers are definable, as we shall later see. This weakness notwithstanding, we can prove the usual theorems of number theory. A few simple examples can be found in Appendix B.

4 First-Order Languages

Propositional logic (also known as *Boolean logic*) has numerous important applications, but is inadequate for many other common reasoning tasks. Consider the formula $\varphi = \forall x. x < 3 \vee x > 2$. We may introduce two propositions $p = x < 3$ and $q = x > 2$. Then we would rewrite φ as $p \vee q$, which is not a propositional *tautology*. (A tautology is a statement that is true by virtue of its logical form.)

However, φ is a *theorem* of the natural numbers (and other number systems). If a program contains an **if** statement with condition $x < 3 \vee x > 2$, we want to be able to conclude that the **else** branch will never execute. This draws us to the study of *first-order logic* and *first-order languages*.

The main new features of first-order logic with respect to propositional logic, as seen in the previous example, are the ability to talk about a domain (e.g., the real numbers), and the ability to *quantify*. First-order languages include *function* symbols like $+$ and *relation* symbols like $<$.

In studying formal languages, we separate their *syntax* from their *semantics*. The former tells us which formulae we may write (are *well-formed*); the latter tells us what well-formed formulae mean. In logic, in particular, semantics tells us which sentences are true. We shall define both syntax and semantics shortly. Now, however, we want to develop a bit of intuition that will help us to see through the definitions.

The truth of a formula depends on its *interpretation*. The formula $\forall x. \exists y. x + y = 0$ is true of the integers, rationals, reals, and complex numbers, but not of the natural numbers. A formula like $\forall x. x \cdot x = x$ is true if we *interpret* the dot \cdot as set intersection or the minimum of two numbers. These examples show that choosing an interpretation involves both selecting a domain of discourse and assigning meaning to the function and relation symbols.

Let us consider the *language of graphs*. An undirected graph is specified by its *edge relation*. Vertex a is connected to vertex b if, and only if, $E(a, b)$. Each interpretation of the relation symbol E corresponds to a different graph. Simple graphs are those that satisfy $\forall x. \neg E(x, x)$. Undirected graphs satisfy $\forall x. \forall y. E(x, y) \rightarrow E(y, x)$. What we can derive from these two axioms without further assumptions on E is true of all simple, undirected graphs.

4.1 Syntax of First-Order Languages

We are now ready to get more formal. A *first-order language* consists of *symbols*, *terms* and *formulae*. The symbols include the *signature* (i.e., the non-logical symbols), the variables, the logical symbols, and punctuation. The logical symbols are: \neg (negation), \wedge (conjunction), \vee (disjunction), \rightarrow (implication), \leftrightarrow (equivalence), \exists (existential quantification), and \forall (universal quantification). Punctuation includes dot, comma, and parentheses. Non-logical symbols include constant, function, and relation symbols. To keep things simple, we assume that the symbols are countable.

Terms are built up inductively from variables, constant symbols, and function symbols. For instance, suppose our variables are v_1, v_2, \dots , and we have one constants c and two function symbols f and g such that f takes two arguments and g takes one. Then

$$v_3, c, g(c), f(f(v_1, c), g(v_2))$$

are four terms. Saying that terms are built inductively implies that the only terms of the language are those obtained by finitely many applications of the rules above. Instead of “inductively,” we could say “recursively.”

The meaning of terms is not the business of syntax, but it helps our intuition to think of terms as signifying elements of the domain of discourse. For example, if the domain is the set of the real numbers, and a and b are constants symbols, they are terms that designate two real numbers, while $a + b$ is a term that designates a real number.

Formulae are built up inductively from relations among terms (e.g., $t_1 = t_2$, $t_1 < t_2$, $P(t_2)$, $Q(t_1, t_2, t_3)$), Boolean connectives applied to formulae (e.g., $\varphi_1 \vee \varphi_2$), and quantification (e.g., $\exists v_1. \varphi$). The formulae obtained by applying a relation symbol to a set of terms are called *atoms* or *atomic formulae*. The formulae that conform to the syntax rules are often called *well-formed formulae* (*wffs*). An example of formula is

$$\forall x. \forall y. x \leq y \vee y \leq x .$$

In this formula, x and y are terms, $x \leq y$ and $y \leq x$ are atomic (sub)-formulae, and $x \leq y \vee y \leq x$, $\forall y. x \leq y \vee y \leq x$ and the entire formula are non-atomic (sub)-formulae. We shall usually abbreviate a formula like the one above as follows:

$$\forall x, y. x \leq y \vee y \leq x .$$

In illustrating the definition we used some familiar relation symbols (equality, inequality) and some generic relation symbols (P and Q). We'll stick to the convention that whenever we use a familiar symbol, its meaning is the familiar one. (We shall *not* use $=$ to mean “different.” While this rule is a convention, it's a most reasonable one.) Whenever we use a generic relation symbol like P or Q above, we'll make it clear whether we have a predefined meaning in mind or not. For instance, it's not hard to convince ourselves that

$$\neg \exists y. \forall x. Q(x, y) \leftrightarrow \neg Q(x, x)$$

holds regardless of which two-place relation Q designates. (Later we'll say, "regardless of the interpretation of binary relation symbol Q .".) No matter how y is chosen, choosing x to be the same as y makes $Q(x, y) \leftrightarrow \neg Q(x, x)$ false.

We are getting ahead of ourselves, though, because saying that a formula is true involves semantics, and we still need to finish our discussion of syntax, to which we now return.

Which non-logical symbols may appear in a formula depends on the language signature. Just as for the function symbols, each relation symbol takes a fixed number of arguments, known as the *arity* of the symbol. The arity of each function and relation symbol is also specified by the signature.

The details of the definition of first-order language vary depending on the source one looks up and are largely unimportant for our outline. Some authors, for example, find it more convenient to regard constants as functions with zero arguments. The important fact to notice is that constants are used to refer to distinguished elements of the domain of discourse. (Note that "distinguished" does not mean "distinct": two constants may refer to the same element.)

Let's apply the definition we have just seen to graphs. Suppose we call the variables v_1, v_2, \dots as before. The language of graphs we consider has neither constant nor function symbols, and two binary relation symbols: E and $=$. ("Binary" means that their arity is 2.) This may seem a very laconic language, but we can write interesting sentences nonetheless:

$$(\forall v_1. \exists v_2. E(v_1, v_2)) \wedge (\forall v_1, v_2, v_3. (E(v_1, v_2) \wedge E(v_1, v_3)) \rightarrow v_2 = v_3) .$$

Variables may occur either *free* or *bound* in a formula. In $\forall x. x \cdot y = 1$, x occurs bound (by the universal quantifier) and y occurs free. A variable is free in a formula if, and only if, it has at least one free occurrence in the formula.

For example, variable x is free in $x = y \wedge \forall x. x > 0$, because it has one free occurrence. It doesn't matter that it also has a bound occurrence. We usually avoid confusion by renaming the bound variable ($x = y \wedge \forall z. z > 0$). We shall see that we can rename bound variables like this without changing the "meaning" of the formula.

A *sentence* is a formula in which there are no *free variables*. That is, all variable occurrences are *bound* by quantifiers. We can ask whether a sentence is either true or false. For a formula with free variables this simple question does not make sense. We will better appreciate this distinction once we have discussed the semantics of first-order languages, which we consider next.

4.2 Semantics of First-Order Languages

A key notion we need to introduce is that of *structure*. We have argued that whether $\forall x. \exists y. x + y = 0$ is true depends on whether we interpret it as a sentence about the integers or the natural numbers, and on whether $+$ designates the usual addition operation or some other function.² Structures allow us to

²If the idea that $+$ may designate something different from regular, old addition seems preposterous, consider floating point addition in most programming languages, which is often written $+$, but is *not associative*: $(a + b) + c$ may overflow, while $a + (b + c)$ may not.

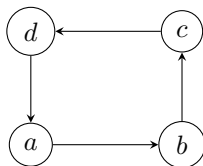


Figure 1: A four-vertex cycle.

formalize this kind of reasoning.

A structure for a first-order language consists of a nonempty *universe of discourse* D , and an *interpretation* of all non-logical symbols of the language. A constant symbol is interpreted as an element of D . A functions symbol of *arity* k is interpreted as a function from D^k to D . A relation symbol of arity k is interpreted as a subset of D^k . Note that the interpretation does not specify the values of variables, even though variables range over D .

As an example, consider the first-order language that has constant symbol 0 , unary function symbol s (successor) and binary relation symbol $=$. The natural numbers are one of the possible structures for this language with the natural numbers \mathbb{N} as domain D and the following interpretation: the constant symbol 0 is interpreted as the natural number 0 , the function symbol s is interpreted as the successor function, and the relation symbol $=$ has its usual meaning of equality relation.

If “ 0 is interpreted as 0 ” bothers you, replace the constant symbol with **0** (boldface) and the pain should abate. After a while, the idea that constant symbols and elements of D , while conceptually distinct, may be written the same way will stop chafing you. Usually the context will tell you whether 0 designates a constant symbol or an element of the domain.

Figure 1 shows a graphical representation of a structure for the language of graphs. (Unsurprisingly, it looks like a graph.) The domain is $D = \{a, b, c, d\}$ and E is interpreted as the binary relation $\{(a, b), (b, c), (c, d), (d, a)\}$. For completeness, we also mention that $=$ has the standard interpretation as the binary relation $\{(a, a), (b, b), (c, c), (d, d)\}$. In future examples, the standard interpretation of the equality symbol will be tacitly assumed.

Now that we know what we mean by “structure” in the context of logic, we can define the truth value of a formula. At the end of the previous section we alluded to the fact that sentences are either true or false of a structure, but the truth value of a formula with free variables depends on the values of those variables. Since formulae with free variables appear as subformulae of most sentences, and we want to define truth inductively—that is, from the ground up—we need some circumspection. Specifically, we use an *environment* to specify values for the free variables.

Another key observation is that terms appear as building blocks of formulae. Hence our inductive definition must provide a way to assign values to terms too. Note that terms take values in D , while formulae take truth values. With our

syntax rules, terms appear inside formulae, not the other way around.³

Given a formula $\varphi(x_1, \dots, x_n)$ of a first-order language, where x_1, \dots, x_n are (distinct) *free* variables; a structure S with domain D , which interprets the non-logical symbols in φ ; and an *environment* e that assigns values from D to x_1, \dots, x_n , we define the truth of $\varphi(x_1, \dots, x_n)$ in S and e by induction on the structure of φ .

If $n = 0$, there are no free variables, φ is a *sentence* and the environment is immaterial. Even if we are ultimately interested in sentences, we need to allow free variables in φ in our definition because subformulae of a sentence may contain free variables. For instance, x is free in subformula $x \geq 0$ of sentence $\forall x. x \geq 0$.

Let $\llbracket \sigma \rrbracket_S$ be the interpretation of (non-logical) symbol σ in structure S . We associate a *value* $\llbracket t \rrbracket_{S,e}$ from D to every *term* t as follows:

- $\llbracket c \rrbracket_{S,e} = \llbracket c \rrbracket_S$ for a constant symbol c ;
- $\llbracket x \rrbracket_{S,e} = e(x)$ for a variable symbol x ;
- $\llbracket f(t_1, \dots, t_k) \rrbracket_{S,e} = \llbracket f \rrbracket_S(\llbracket t_1 \rrbracket_{S,e}, \dots, \llbracket t_k \rrbracket_{S,e})$ for a function symbol f of arity k .

Quite simply, the value of a constant symbol is whatever the interpretation assigns to it; the value of a variable is whatever the environment assigns to it; the value of a function application term is obtained by applying the interpretation of the function symbol to the values of the argument terms. The value of a term is also known as its *denotation*.

As an example, consider the formula $\forall x. \exists y. x + y = 0$ and suppose S is the set of integers with the usual interpretation of $+$. If $e(x) = 1$ and $e(y) = 2$, then

$$\llbracket x + y \rrbracket_{S,e} = \llbracket x \rrbracket_{S,e} + \llbracket y \rrbracket_{S,e} = e(x) + e(y) = 1 + 2 = 3 .$$

We now consider formulae. Let $e[d/x]$ be the environment that is identical to e , except that $e[d/x](x) = d$. We write $S, e \models \varphi$ if φ is true in S and e . (The symbol \models is called *double turnstile* and denotes the *satisfaction relation*.)

- $S, e \models R(t_1, \dots, t_k)$ if, and only if, $(\llbracket t_1 \rrbracket_{S,e}, \dots, \llbracket t_k \rrbracket_{S,e}) \in \llbracket R \rrbracket_S$ for a relation symbol R of arity k .
- $S, e \models \neg \varphi$ if, and only if, $S, e \not\models \varphi$.
- $S, e \models \varphi \wedge \psi$ if, and only if, both $S, e \models \varphi$ and $S, e \models \psi$.
- $S, e \models \varphi \vee \psi$ if, and only if, either $S, e \models \varphi$ or $S, e \models \psi$.
- $S, e \models \exists x. \varphi(x)$ if, and only if, $S, e[d/x] \models \varphi(x)$ for some $d \in D$.
- $S, e \models \forall x. \varphi(x)$ if, and only if, $S, e[d/x] \models \varphi(x)$ for all $d \in D$.

³If, for example, the syntax included ternary operators that map a formula and two terms to a term, there would be formulae inside terms. Defining the semantics would get more complicated, but nothing conceptually deep would come from this (common) extension.

The interpretation of *equality* ($=$) is fixed:

- $S, e \models t_1 = t_2$ if, and only if, $\llbracket t_1 \rrbracket_{S,e} = \llbracket t_2 \rrbracket_{S,e}$.

The two equal signs in the line above denote different things. The first one is the relation symbol in the first-order language; the second is the symbol for equality in the *metalinguage*, which is the language we use to talk about the first-order language. The metalanguage is usually a mixture of natural language and mathematical notation. Put simply, $S, e \models t_1 = t_2$ if $\llbracket t_1 \rrbracket_{S,e}$ and $\llbracket t_2 \rrbracket_{S,e}$ are the same element of D .

A structure S for a first-order language such that a sentence φ of the language is true in S ($S \models \varphi$) is a *model* of φ . This extends to sets of sentences Γ in the obvious way: a structure S for a first-order language such that all sentences of the language in set Γ are true in S (written $S \models \Gamma$) is a *model* of Γ .

4.3 Satisfiability, Validity, Definability

A sentence φ is *satisfiable* if, and only if, it is true in some structure for the language. A sentence φ of a first-order language is *valid* (written $\models \varphi$) if and only if it is true in all structures for the language. Equivalently, a φ is satisfiable if, and only if, it has a model, and is valid if all structures for the language of φ are models of φ .

A set of sentences Γ *logically implies* a sentence φ (written $\Gamma \models \varphi$) if, and only if, every model of Γ is a model of φ . That is, $S \models \Gamma$ implies $S \models \varphi$. Even though we use the same symbol, when we write $\Gamma \models \varphi$ we mean a different relation than we write $S \models \varphi$. To stress this fact, we say that S is in *satisfaction relation* with φ , while Γ is in *consequence relation* with φ .

Both validity and logical implication can be reduced to satisfiability. Sentence φ is valid if, and only if, $\neg\varphi$ is not satisfiable. Sentence φ is logically implied by sentence ψ if, and only if, $\psi \rightarrow \varphi$ is valid.

This last claim is an instance of the *semantic deduction theorem*.

Theorem 1. *If Γ is a set of sentences and φ_1, φ_2 are sentences, then*

$$\Gamma \cup \{\varphi_1\} \models \varphi_2 \text{ if, and only if, } \Gamma \models \varphi_1 \rightarrow \varphi_2 .$$

Proof. $\Gamma \cup \{\varphi_1\} \not\models \varphi_2$ if, and only if, there exists a structure S such that $S \models \Gamma$, $S \models \varphi_1$, and $S \not\models \varphi_2$. Equivalently, there exists S such that $S \models \Gamma$ and $S \not\models \varphi_1 \rightarrow \varphi_2$. This holds if, and only if, $\Gamma \not\models \varphi_1 \rightarrow \varphi_2$. We have shown that

$$\Gamma \cup \{\varphi_1\} \not\models \varphi_2 \text{ if, and only if, } \Gamma \not\models \varphi_1 \rightarrow \varphi_2 .$$

The contrapositive of this claim is our theorem. □

Note that by using the contrapositive in the proof, we didn't have to explicitly deal with the case in which $\Gamma \cup \{\varphi_1\}$ has no models. As an example of application of Theorem 1, let's start from the observation that $\{p, q\}$ logically implies p because any model of both p and q must be a model of p . Applying

semantic deduction twice we get $\{p, q\} \models p$ if, and only if, $\{p\} \models q \rightarrow p$ if and only if $\models p \rightarrow (q \rightarrow p)$, from which we conclude that $p \rightarrow (q \rightarrow p)$ is a valid sentence.

A k -ary relation R on the universe of discourse of a structure S is *definable* in S if, and only if, there is a formula φ in free variables v_1, \dots, v_k that defines R in S . This means that the k -tuples in R are precisely those that, assigned to v_1, \dots, v_k , make φ true. For example, the formula $\exists v_2. v_1 = v_2 + v_2$ in the free variable v_1 defines the set of even natural numbers (a unary relation).

Not all relations on the natural numbers are definable in Peano arithmetic, since there are more subsets of \mathbb{N} (uncountably many) than there are formulae (countably many). We run out of formulae even if we restrict ourselves to unary relations.

4.4 First-Order Theories

A *first-order theory* is a set of sentences T of some first-order language that is *closed under implication*:

$$T \models \sigma \Rightarrow \sigma \in T .$$

The sentences of T are the *theorems* of the theory. A set of *axioms* for T is a subset of T from which all of T can be obtained by implication.

Alternatively, a theory is defined by a set of sentences taken as axioms. The theorems are then all the sentences that are implied by the theory. In the formula above, the symbol \Rightarrow is part of the metalanguage, whereas we use \rightarrow to denote implication in the formulae of the language.

An example of theory defined by a set of axioms is the first-order Peano arithmetic of Section 3. The use of an axiom schema for induction implies that the number of axioms is infinite.

Another example is provided by the theory of *groups*, whose signature may be chosen to contain the binary function symbol \circ (the group operation), the constant e (the identity), and the binary equality relation. The theory of groups is then defined by the following axioms:

1. $\forall x, y, z. x \circ (y \circ z) = (x \circ y) \circ z$ (associativity of \circ);
2. $\forall x. x \circ e = e \circ x = x$ (existence of identity);
3. $\forall x. \exists y. x \circ y = y \circ x = e$ (existence of inverses).

The choice of signature (and consequently axioms) is not unique. In the case of groups, one may include a unary function for the inverse, or omit the constant for the identity.

A theory is *effectively generated* (alternatively, *recursively enumerable*) if its axioms are a recursively enumerable set. That is, there is an algorithm that outputs a list of the axioms. Obviously, every theory defined by a finite set of axioms is effectively generated. In the presence of axiom schemata, the algorithm does not terminate. What matters is that if we pick a particular axiom, the algorithm will output it in finite time. Both Peano arithmetic and the theory of groups are effectively generated.

5 Deductive Calculi

In the previous section we have defined what is meant for truth of a formula. Let's summarize. A sentence φ of first-order language L is either true ($S \models \varphi$) or false ($S \not\models \varphi$) in a structure S for L (an L -structure), which consists of a domain and an interpretation of the non-logical symbols of L .

For a formula φ with free variables, the values of those variables must be specified for us to assign a truth value to φ . Another useful way to look at it, is that a formula with k free variables defines a k -ary relation. In particular, a formula with one free variable is often called a *predicate*.

Once we have defined the satisfaction relation $S \models \varphi$, we can say what it means for a formula φ of language L to be satisfiable (there exists at least one L -structure S such that $S \models \varphi$) or valid (for all L -structures S , $S \models \varphi$).

Finally, φ is a tautology if its truth “only depends on its propositional structure.” What this means is that a tautology in a first-order language L is a formula that is obtained starting from a propositional tautology (e.g., $(P \rightarrow Q) \rightarrow (\neg Q \rightarrow \neg P)$) and replacing each propositional variable with a formula of L . For example, we may replace P with $\forall x.R(x)$ and Q with $y = 0 \wedge \exists z.y = z$. We obtain

$$((\forall x.R(x)) \rightarrow (y = 0 \wedge \exists z.y = z)) \rightarrow (\neg(y = 0 \wedge \exists z.y = z) \rightarrow \neg(\forall x.R(x))) ,$$

which is guaranteed valid by virtue of its construction. An example of a valid sentence that is not a tautology is provided by

$$(\exists x.P(x) \vee Q(x)) \leftrightarrow ((\exists x.P(x)) \vee (\exists x.Q(x))) .$$

Any structure for the language of φ is a potential witness for the satisfiability of sentence φ . Suppose we want to prove that the rational numbers are *dense*, that is, that they satisfy

$$\varphi = \forall x, y. x < y \rightarrow \exists z. x < z \wedge z < y .$$

The formula is not true of the integers: therefore φ is not valid. If we just ask whether φ is satisfiable, the witnessing structure may be the reals, or the reals in the interval $[0, 1]$. We may also be handed a structure with $|D| = 3$ in which $x < y$ is interpreted as “ x is different from y .”

In such cases, we have not really reached our objective of proving a theorem about the rational numbers. Therefore, in practice, we may want to fix some features of the structure S such that $S \models \varphi$. Ideally, in so doing we also gain efficiency in our search. More on this later; right now, we focus on the problem we have to solve when we impose no constraints on the structure S besides $S \models \varphi$.

The definitions we have recalled are perfectly good, but if we think of what it takes to check whether a specific formula is satisfiable (or valid) we are immediately faced with difficulties. How are we supposed to single out a structure S in which our sentence φ holds among the infinitely many structures for the

language of φ ? If there is one structure, we may luck out and find it, but what if φ is unsatisfiable? Can we ever stop and report that conclusion?

Suppose, on the other hand, that we have selected a structure S and we need to check whether $S \models \varphi$. (This is what goes under the name of *model checking*.) How do we proceed when S is not finite?

We look for a different approach to simply sifting through structures and rummaging in each of them; this is what we take up next. We'll make significant progress by introducing the notion of (formal) *proof*. (However, not all difficulties will go away: first-order validity is only *semidecidable*.)

A *deductive calculus* (or *proof system*) consists of a set of *logical axioms* and *inference rules*. If we are interested in proving the truth of sentences, we better choose axioms that are valid. For instance, we may choose as axioms tautologies of certain (simple) shapes. There's also some latitude in choosing the inference rules; of course, they should prevent one from deriving false conclusions from true premises. They should also be "powerful enough," in a sense that will be made precise later. Popular inference rules include *modus ponens* and *universal generalization*. Modus ponens says that from p and $p \rightarrow q$ we can infer q . Universal generalization says that if we can infer $\varphi(a)$ from a set of assumptions Γ , then we can infer also $\forall x. \varphi(x)$, provided a does not appear in Γ and x does not occur in $\varphi(a)$.

Suppose we have fixed a proof system. A sentence φ is *provable* from a set of first-order formulae Γ ($\Gamma \vdash \varphi$) if there is a sequence of sentences ending in φ and such that each sentence is a *logical axiom*, a formula in Γ , or is derived from previous sentences by an *inference rule*. Such a sequence of sentences is a *proof* of φ . As an example, consider this (semi-formal) proof that a group in which each element is its own inverse is Abelian (that is, commutative).

$$\begin{aligned}
& \forall x. x \circ x = e \\
& \forall x. (x \circ e) \circ x = e \\
& \forall x, y. (x \circ (y \circ y)) \circ x = e \\
& \forall x, y. ((x \circ y) \circ y) \circ x = e \\
& \forall x, y. (x \circ y) \circ (y \circ x) = e \\
& \forall x, y. ((x \circ y) \circ (y \circ x)) \circ (y \circ x) = e \circ (y \circ x) \\
& \forall x, y. (x \circ y) \circ ((y \circ x) \circ (y \circ x)) = e \circ (y \circ x) \\
& \forall x, y. (x \circ y) \circ e = e \circ (y \circ x) \\
& \forall x, y. x \circ y = y \circ x .
\end{aligned}$$

While this proof may seem quite detailed, note that going from the first to the second line is justified by the axiom of existence of the identity ($\forall x. x \circ e = x$) and the rule of *replacement*. Associativity is used twice to obtain the fifth line and then again to obtain the seventh line. A proof according to the definition of "provable" above would spell out all these details. A proof found in a textbook on group theory would take advantage of associativity to drop parentheses that do not improve readability and skip a few steps.

A sequence is often a linearization of a tree. We often come up with proofs that look like trees, but we can then order the formulae in a sequence. Since the proof of φ ends with φ and each formula is of finite length, the whole proof is a finite sequence of finite sequences of symbols: that is, a proof is a finite object.

Note that the *syntactic consequence* symbol \vdash takes meaning in the context of a specific proof system. When we talk about multiple proof systems at once, we annotate the symbols, as in \vdash_G , \vdash_H , and so on. Also note that a proof of a sentence in one proof system is often not a proof in another proof system. So we may need to distinguish a G -proof from an H -proof.

5.1 Semantic Tableaux

We now outline a way to prove validity of first-order sentences known as the *semantic tableau* approach. The connection between semantic tableaux and formal proofs as we just defined them will become apparent in Section 5.2.

We start from propositional logic. The resulting procedure will look somewhat like CDCL (which we shall see later): It produces a tree, but in a different way.⁴ Then we consider the semantic tableau method for first-order logic. (Those familiar with translation of LTL to Büchi automata may recall that the automaton is also called a tableau automaton.) Some examples of semantic tableaux are adapted from [38] and [3].

To prove the validity of a sentence with the semantic tableau method we try to find a model to its negation by analyzing its structure. If the sentence φ is the conjunction of two subformulae, $\varphi = \varphi_1 \wedge \varphi_2$, then a model for φ must be a model for both φ_1 and φ_2 . For $\varphi = \varphi_1 \vee \varphi_2$, we need to find either a model of φ_1 or a model of φ_2 . Similar reasoning applies to all Boolean connectives. As an example, the proof of validity of $(p \vee (q \wedge r)) \rightarrow ((p \vee q) \wedge (p \vee r))$ is shown in Figure 2. This formula is half of the distributivity property of disjunction over conjunction. Each formula in the tableau of Figure 2 is labeled with an index on the left. The number on the right of a formula is the index of the formula that was used to produce it. The first formula of the tableau is the negation of the formula we want to prove valid. An implication is false if, and only if, its antecedent is true and its consequent is false. Therefore we enter the antecedent as line (2) and the negation of the consequent as line (3) in the tableau. Our search for a model of (1) has turned into the search for a model of (1)–(3). We then look at line (2), which is a disjunction; there are two ways to satisfy it, corresponding to lines (4) and (5). Since they are alternatives, they go into different nodes of the tableau. This means that we are looking for either a model of (1)–(4) or a model of (1)–(3), (5).

Line (5) is a conjunction; hence we add lines (6) and (7) to the same node of the tableau. We now use line (3), which produces lines (8)–(11). Note that we apply DeMorgan’s laws to the negation of a conjunction, turning it into the disjunction of two negations, and that each of the two existing branches

⁴We could also use truth tables to prove validity of propositional formulae, but truth tables do not “scale up” to first-order logic.

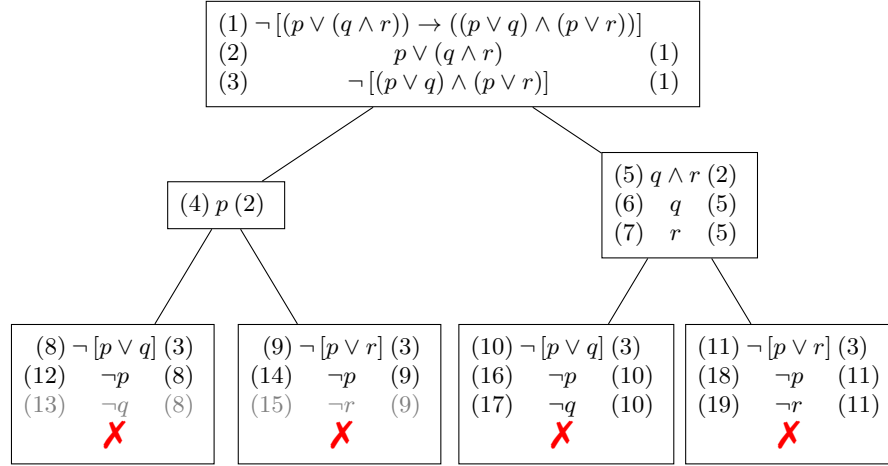


Figure 2: Example of semantic tableau for propositional logic.

is expanded in two ways. Let us focus on the leftmost branch, and expand line (8) into lines (12) and (13). Now we notice that (4) and (12) contradict each other; Hence the leftmost branch is *closed*, which means that it is proved to be unsatisfiable. Continuing in similar fashion, we close all four branches of Figure 2, which means that (1), the negation of the give formula, is unsatisfiable.

There are two types of rule: one applies to formulae like $p \wedge q$ or $\neg(p \rightarrow q)$ and does not cause branching. We call this type “ α -rule.” The other type applies to formulae like $p \vee q$ and $p \rightarrow q$ and causes branching. We call this type “ β -rule.” In the example, lines (2)–(3), (6)–(7), and (12)–(19) were produced by α -rules. The other lines were produced by β -rules.

A couple of remarks are in order. In a tableau for propositional logic, we only need to deal with a line once if we enter the formulae we derive from it in all branches reachable from that formula at once. We also apply α -rules right away, so as to keep the tableau as simple as possible. That’s what was done in producing Figure 2 and explains the line indices. Moreover, we always “push negations inside” until we have literals. Every new formula is simpler than the one from which it is derived; this means that building the tableau must terminate. While we have not detailed an actual procedure, we have seen the essential elements of it: as long as there are unused lines, expand one and mark it as used. If the tableau for φ has at least one branch that is not closed at termination, then that branch identifies a satisfying assignment for φ .

We now look at the extension of the tableau approach to first-order logic. We only consider languages with no function symbols. This may seem very limiting, but, given a formula φ in a language L with constant or function symbols, there are a related language L' and a formula φ' of L' such that φ is satisfiable if and only if φ' is. Since even for simple φ , the derived formula φ' tend to be unwieldy, we shall not spend time on the transformation that produces it, but it’s good to

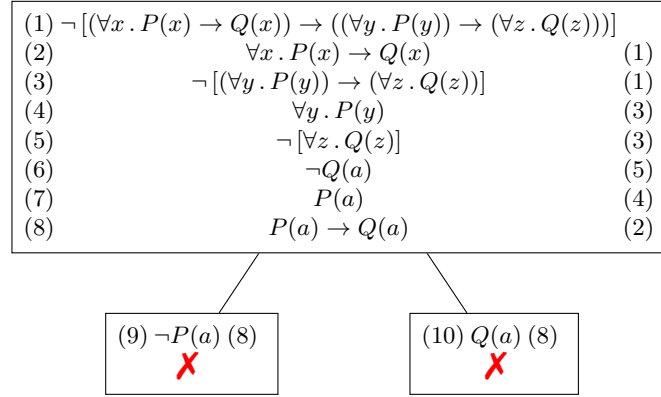


Figure 3: Example of semantic tableau for first-order logic.

know that it exists. It is also possible to extend the semantic tableau approach to deal with function symbols, but we will not do so.

We shall also refrain from using any relation symbol whose interpretation is predetermined. Specifically, we shall not use equality. Again, this looks like a severe blow to the applicability of the tableau approach, but it isn't. We can conjoin our formula with other formulae that constrain the interpretation of $=$ to the equality relation. These additional formulae are indeed the axioms of equality.

To recap, we are given a sentence of a first-order language with neither function symbols nor interpreted relation symbols. We want to prove its validity by an extension of the tableau approach we have outlined for propositional logic. First and foremost, we need to deal with quantifiers. Let's first see how it is done on an example. The proof of validity of $(\forall x. P(x) \rightarrow Q(x)) \rightarrow ((\forall y. P(y)) \rightarrow (\forall z. Q(z)))$ is shown in Figure 3. We apply the usual rules for Boolean connectives to obtain lines (2)–(5). Lines (1) and (3) at this point have been “used,” while two more formulae start with a quantifier and the last one is the negation of a formula that starts with a quantifier. We pick line (5) and reason as follows: If it is not the case that $\forall z. Q(z)$, then there is an element, let's call it a , such that $\neg Q(a)$. We enter this formula as line (6) of the tableau. Here, a is a *fresh* constant symbol: a constant symbol that does not appear yet in the tableau. We insist on freshness so that a may stand for *any* element of the domain—as long as it satisfies $\neg Q$.

Our attention now shifts to line (4): if $P(y)$ must hold for all y , then, in particular, $P(a)$ must hold. This becomes line (7) of the tableau. For universal quantification, we do not insist on freshness of the constant symbol. On the contrary, we usually want to make sure that the universal claim holds for those elements whose existence we have previously postulated—like a in this case. With similar process we produce line (8) from line (2). Lines (6)–(8) are easily seen to be inconsistent. Our semi-systematic procedure, however, takes one more step, producing lines (9) and (10); the former conflicts with line (7), and

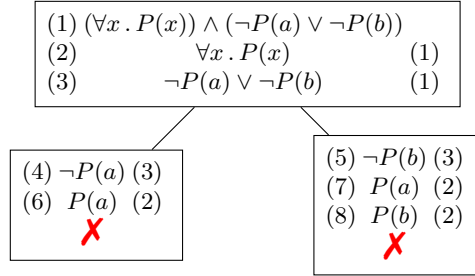


Figure 4: Semantic tableau requiring multiple instantiations.

the latter conflicts with line (6). Therefore both branches close and the sentence of line (1) is declared unsatisfiable.

Summing up, when we instantiate an existential quantifier or a negated universal quantifier—we apply a δ -rule—we introduce a fresh constant symbol, but when we instantiate a universal quantifier or a negated existential quantifier (a γ -rule) we use a symbol that is already present. Another important distinction, which is not highlighted by our last example, is that we may need to instantiate the same universal quantifier more than once—in fact, we cannot in general bound the number of its instantiations. When we apply a γ -rule, we do not mark the formula as “used;” if we did, we would not be able to prove the unsatisfiability of

$$(\forall x. P(x)) \wedge (\neg P(a) \vee \neg P(b)) ,$$

whose tableau is shown in Figure 4. This is big news, because it implies that the procedure is no longer guaranteed to terminate.

A “proof” of validity of $(\forall x. P(x) \vee Q(x)) \rightarrow ((\forall y. P(y)) \vee (\forall z. Q(z)))$ is shown in Figure 5. This sentence is half the claim that universal quantification distributes over disjunction. In fact, it’s the implication that is *not* valid. Let’s see what happens when we build a tableau for its negation. The tableau shown in Figure 5 cannot be further extended. We say that it is *complete*. All lines that could be used once have been used, and line (2), which may be used multiple times, has been instantiated for both a and b .

The branch that goes through formula (12) is not closed, though, because no contradiction was reached. Hence we can construct a model for the sentence of line (1):

- The domain is a two-element set $D = \{d_a, d_b\}$.
- The constants a and b are interpreted as d_a and d_b , respectively.
- The predicates P and Q are interpreted as follows:

$$\llbracket P \rrbracket(d_a) = \perp \quad \llbracket P \rrbracket(d_b) = \top \quad \llbracket Q \rrbracket(d_a) = \top \quad \llbracket Q \rrbracket(d_b) = \perp .$$

There are other models as well, both finite and infinite. What distinguishes them is that they all have at least two elements that interpret a and b such that $\llbracket Q(a) \rrbracket$ and $\llbracket P(b) \rrbracket$ are true, while $\llbracket Q(b) \rrbracket$ and $\llbracket P(a) \rrbracket$ are false.

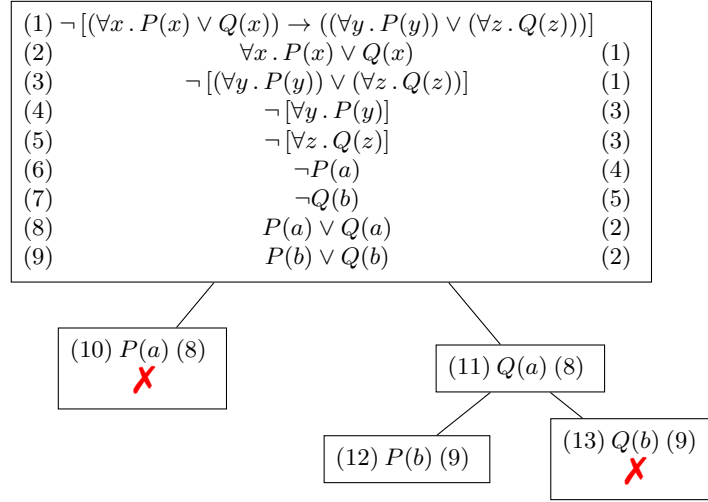


Figure 5: Semantic tableau for a satisfiable first-order logic sentence.

As a final example of first-order tableau consider the following set of sentences, which only has infinite models:

$$\begin{aligned}
 &\forall x. \exists y. E(x, y) \\
 &\forall x. \neg E(x, x) \\
 &\forall x. \forall y. \forall z. (E(x, y) \wedge E(y, z)) \rightarrow E(x, z) .
 \end{aligned}$$

These sentences are modeled (for instance) by the natural numbers with E interpreted as $<$. Suppose there exists a model S for these sentences whose domain D has cardinality $n \in \mathbb{N}$. Starting from an arbitrary element $d_0 \in D$ and applying the first axiom n times we generate a sequence of $n + 1$ elements of the domain, d_0, \dots, d_n , which cannot all be distinct. Hence, there are i and j , with $0 \leq i < j \leq n$ such that $d_i = d_j$. Now, the second axiom (irreflexivity of E) prescribes $\neg E(d_i, d_j)$, while the third axiom (transitivity of E) implies $E(d_i, d_j)$. This is a contradiction; hence the required finite S does not exist.

Before we attempt to build a tableau to show the satisfiability (not the validity in this case) of our three sentences a few observations are in order. First, the tableau approach is easily extended to finite sets of sentences by initializing the first node to all given sentences. So, lines (1)–(3) of our tableau will be the three axioms. Second, all three axioms start with a universal quantifier. In such a case, we introduce a fresh constant symbol in the language. Since the domain of a model is assumed non-empty, this operation is always permitted. Finally, an axiom like $\forall x. \forall y. \forall z. (E(x, y) \wedge E(y, z)) \rightarrow E(x, z)$ is the source of a lot of work when building a tableau. As an example, the instantiation

$$(E(a_1, a_1) \wedge E(a_1, a_1)) \rightarrow E(a_1, a_1)$$

produces, when the Boolean connectives are expanded, three branches: two

(1)	$\forall x . \exists y . E(x, y)$	
(2)	$\forall x . \neg E(x, x)$	
(3)	$\forall x . \forall y . \forall z . (E(x, y) \wedge E(y, z)) \rightarrow E(x, z)$	
(4)	$\exists y . E(a_1, y)$	(1)
(5)	$\neg E(a_1, a_1)$	(2)
(6)	$E(a_1, a_2)$	(4)
(7)	$\neg E(a_2, a_2)$	(2)
(8)	$\exists y . E(a_2, y)$	(1)
(9)	$E(a_2, a_3)$	(8)
(10)	$\neg E(a_3, a_3)$	(2)
(11)	$E(a_1, a_3)$	(3)
(12)	$\exists y . E(a_3, y)$	(1)
(13)	$E(a_3, a_4)$	(12)
(14)	$\neg E(a_4, a_4)$	(2)
(15)	$E(a_1, a_4)$	(3)
(16)	$E(a_2, a_4)$	(3)
(17)	$\exists y . E(a_4, y)$	(1)
	\dots	

Figure 6: Initial fragment of an infinite semantic tableau.

contain $\neg E(a_1, a_1)$, which is also obtained from $\forall z . \neg E(z, z)$; the third is immediately closed because it contains $E(a_1, a_1)$, which conflicts with $\forall z . \neg E(z, z)$. We'll cut a few corners and only use the transitivity axiom when it produces a meaningful addition to a branch, without causing it to close.

What do we expect our tableau to look like? We know that there is a model: the natural numbers. Hence the tableau will not close. However, all models are infinite. We do not expect to find a complete open branch either: we saw in the previous example how to derive a finite model from one such branch. We are left with the option that the construction of the tableau will not terminate, and that is precisely what happens in Figure 6.

Our examples have illustrated three possible scenarios: an unsatisfiable sentence, whose tableau closed; a satisfiable sentence with a finite model, whose tableau completed with an open branch; and a satisfiable sentence with no finite models, whose tableau was infinite. Are there other pertinent scenarios? Is it possible for the tableau construction to go on forever even when the sentence is unsatisfiable or has finite models?

Our vaguely defined procedure may well go on forever even on a sentence that has a closed tableau. The main reason is that we have stated no rule that forces us to instantiate all quantifiers. Consider, for instance, adding the symmetry axiom

$$\forall s, t . E(s, t) \rightarrow E(t, s)$$

to the three axioms above. Then the resulting set of sentences is unsatisfiable because $E(a_1, a_2)$ implies $E(a_2, a_1)$ by symmetry. Applying the transitivity

axiom to these two atoms allows us to infer $E(a_1, a_1)$, which contradicts the irreflexivity axiom. However, we can still build the infinite branch of Figure 6 if we keep postponing the instantiation of the symmetry axiom. In the absence of additional constraints, we are not forced to instantiate it because there's always something else we can do to continue the tableau construction.

This observation points in the direction of adding some constraints that impose a “systematic” use of all available universally quantified formulae. We'll just mention that there are several ways to do so. The interested reader may consult [38, 3, 14, 10]. The important consequence of these constraints is that the resulting semantic tableau procedures are guaranteed to find a (finite) closed tableau whenever applied to an unsatisfiable formula.

Though we have mostly waved our hands, we have at least provided some intuition for the conclusion that the semantic tableau approach is *sound and complete for validity*. This means that a sentence is valid if, and only if, its negation has a closed tableau and that such a tableau is found by systematic procedures that compute semantic tableaux. If a sentence is not valid, however, the construction of the tableau may not terminate. (Note, however, that we cannot effectively use non-termination to conclude non-validity.)

5.2 From Tableaux to Proofs

A closed tableau for $\neg\varphi$ shows that any attempt to build a model of $\neg\varphi$ runs into a contradiction. Hence it proves that φ is valid. However, the tableau does not, at first glance, conform to our definition of proof as a list of sentences, in which each sentence is either an axiom, or is derived from previous sentences by application of an inference rule. We now show how we can easily derive such a list-shaped proof from the tableau. As we did when we introduced the semantic tableaux, we start from the propositional case.

The proof system we outline is a Gentzen-style proof system. We choose it because of its close relation with tableaux. As all proof systems, ours has axioms and inference rules. The axioms are sets of formulae that contain both an atomic formula and its negation. For instance, $\{p(x), q(a, y), \neg p(x)\}$. Since we interpret each set as a disjunction, each axiom is a tautology. Inference rules consist of zero or more *premises* and a *conclusion*: if we have proved the premises, application of the rule allows us to prove the conclusion. Collectively, premises and conclusions are called *judgments*. When we write $\vdash \Gamma$ in our proofs, we judge that the disjunction of the sentences in Γ is valid. When writing inference rules, premises and conclusions are usually separated by a horizontal line, as in this example:

$$\frac{\vdash \Gamma \cup \{\alpha_1, \alpha_2\}}{\vdash \Gamma \cup \{\alpha_1 \vee \alpha_2\}}$$

This inference rule says that, if at some point we have proved a set of sentences that contains α_1 and α_2 , we have also proved the set of sentences obtained by replacing α_1 and α_2 by their disjunction. (Γ is a possibly empty set of sentences.)

Since a set of sentences is interpreted as their disjunction, this rule simply relies on the commutativity and associativity of disjunction.

In general, rules may include assumptions, which are sets of formulae interpreted as conjunctions of their members:

$$\frac{\Delta \vdash \Gamma \cup \{\alpha_1, \alpha_2\}}{\Delta \vdash \Gamma \cup \{\alpha_1 \vee \alpha_2\}} (\vee \text{ right})$$

This says that if $\Gamma \cup \{\alpha_1, \alpha_2\}$ is provable from assumptions Δ , then $\Gamma \cup \{\alpha_1 \vee \alpha_2\}$ is provable from the same assumptions. A judgment of the form $\Delta \vdash \Gamma$ is a *sequent*. If Δ is the empty set, it can be omitted. The label to the right of the formula is (a short form of) the name of the rule. The full name would be something like “ \vee introduction to the right (of the turnstile).” We shall not need assumptions to connect semantic tableaux to proofs, because we shall be dealing with proofs of validity, and not with proofs of logical consequence.

From α_1 and α_2 , we could infer $\neg(\neg\alpha_1 \wedge \neg\alpha_2)$ instead of $\alpha_1 \vee \alpha_2$; or perhaps, $\neg\alpha_1 \rightarrow \alpha_2$. We allow ourselves full freedom in that regard and, in keeping with the established practice, refer to all these related rules as α -rules. We also tacitly suppress double negations, though in a more detailed treatment, we would introduce a rule for that very purpose. For propositional validity, we only need another family of rules, the β rules, exemplified by:

$$\frac{\vdash \Gamma_1 \cup \{\beta_1\} \quad \vdash \Gamma_2 \cup \{\beta_2\}}{\vdash \Gamma_1 \cup \Gamma_2 \cup \{\beta_1 \wedge \beta_2\}} (\wedge \text{ right})$$

This rule is easy to interpret when Γ_1 and Γ_2 are empty. It then says that if at some points we have proved both β_1 and β_2 , then we have also proved $\beta_1 \wedge \beta_2$.

Let’s see this method at work by proving half of the contrapositive property: $(P \rightarrow Q) \rightarrow (\neg Q \rightarrow \neg P)$. To avoid clutter, we omit the braces around set member lists.

$$\frac{\frac{\frac{\vdash P, \neg P, Q \quad \vdash \neg Q, \neg P, Q}{\vdash \neg[P \rightarrow Q], \neg P, Q} (\beta)}{\vdash \neg[P \rightarrow Q], \neg Q \rightarrow \neg P} (\alpha)}{\vdash (P \rightarrow Q) \rightarrow (\neg Q \rightarrow \neg P)} (\alpha)$$

The two premises on the first line are axioms because they contain pairs of complementary literals. Let us use the tableau approach to prove the validity of our sentence. Instead of writing each new formula on a separate line, every time we apply a rule we write all the formulae that are not yet “used up” on the same line. Figure 7 shows what we get. We then take the tableau and turn it upside down in Figure 8. A comparison with the Gentzen-style proof shows that the trees are isomorphic, and all formulae that appear in one node are the negation of those that appear in the corresponding node of the other tree. In fact, the proof was mechanically derived from the tableau by this device.

Why does it work? The leaf that terminates a closed branch of the tableau corresponds to an inconsistent set of formulae; the disjunction of their negations

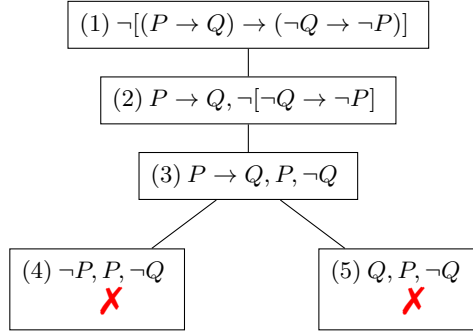


Figure 7: Semantic tableau redrawn to look more like a Gentzen-style proof.

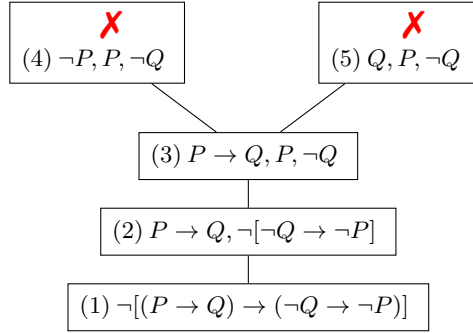


Figure 8: The semantic tableau of Figure 7 turned upside-down.

is therefore valid and can be taken as an axiom. The inference rules of our proof system have been chosen to reverse the effects of the tableau rules. We may also appreciate how much easier it is to build a tableau than to guess the right axioms and sequence of inference rules that prove the desired validity result.

To extend the translation of tableaux to proofs so that it covers first-order logic we need two more rules that reverse the tableau rules for quantifiers. The rule for existential quantifiers is

$$\frac{\vdash \Gamma \cup \{\exists x. \varphi(x), \varphi(a)\}}{\vdash \Gamma \cup \{\exists x. \varphi(x)\}} (\gamma)$$

Now that we know what's going on, we can flip the rule and negate antecedent and consequent. We then recognize the tableau rule for universal quantifiers: if $\forall x. \neg\varphi(x)$ is true, then, for any constant symbol a , $\neg\varphi(a)$ must also be true. We can also provide a direct interpretation for the γ -rule: if we know that $\varphi(x)$ is true for some x , we may not need to carry along also the formula that gives that x the name a .

The rule for universal quantifiers is derived by reversing the tableau rule for existential quantifiers. When an existentially quantified formula is expanded in

$$\begin{array}{c}
\frac{\frac{\frac{\frac{\vdash \neg \forall x . P(x) \rightarrow Q(x), \textcolor{green}{P}(a), \quad \vdash \neg \forall x . P(x) \rightarrow Q(x), \neg \textcolor{green}{Q}(a),}{\neg \forall y . P(y), \neg P(a), Q(a)} \quad \neg \forall y . P(y), \neg P(a), Q(a)}{\vdash \neg \forall x . P(x) \rightarrow Q(x), \neg \forall y . P(y), \neg P(a), Q(a)} (\beta)}{\vdash \neg \forall x . P(x) \rightarrow Q(x), \neg (P(a) \rightarrow Q(a)), \neg \forall y . P(y), \neg P(a), Q(a)} (\gamma)} \\
\frac{\vdash \neg \forall x . P(x) \rightarrow Q(x), \neg \forall y . P(y), \neg P(a), Q(a)}{\vdash \neg \forall x . P(x) \rightarrow Q(x), \neg \forall y . P(y), Q(a)} (\gamma)}{\vdash \neg \forall x . P(x) \rightarrow Q(x), \neg \forall y . P(y), \forall z . Q(z)} (\delta)} \\
\frac{\vdash \neg \forall x . P(x) \rightarrow Q(x), \neg \forall y . P(y), \forall z . Q(z)}{\vdash \neg \forall x . P(x) \rightarrow Q(x), (\forall y . P(y)) \rightarrow (\forall z . Q(z))} (\alpha)}{\vdash (\forall x . P(x) \rightarrow Q(x)) \rightarrow ((\forall y . P(y)) \rightarrow (\forall z . Q(z)))} (\alpha)
\end{array}$$

Figure 9: A proof derived from a first-order semantic tableau. The formulae involved in the application of each rule are highlighted.

the tableau construction, a fresh constant is introduced and the formula is used up. Negating and flipping, we obtain the following rule:

$$\frac{\vdash \Gamma \cup \{\varphi(a)\}}{\vdash \Gamma \cup \{\forall x . \varphi(x)\}} (\delta)$$

The freshness condition translates into the requirement that a should not appear in any formula in Γ .

As an example, we show a proof derived from the tableau of Figure 3 in Figure 9. The sequence of applied rules is β , γ , γ , δ , α , α . A remark that is rather obvious for those who have gotten thus far is that the trees obtained from the tableaux can be put in linear form to obtain the list of formulae prescribed by our definition of proof. One simple way to do so is to do a post-order traversal of the proof tree.

In the rest of these notes we shall not make much practical use of the deductive calculus we have developed in this section. Our main purpose was to show that one could be devised. However, before we conclude this section, we want to make a few observations. First, we ignored assumptions in our rules because we were concerned with proofs of validity. We can make our calculus more versatile by adding these inference rules:

$$\frac{\Gamma \vdash \{\varphi\} \cup \Delta}{\Gamma \cup \{\neg \varphi\} \vdash \Delta}$$

$$\frac{\Gamma \cup \{\neg \varphi\} \vdash \Delta}{\Gamma \vdash \{\varphi\} \cup \Delta}$$

Using these and the α -rules, we can then introduce a derived rule:

$$\frac{\Gamma \cup \{\varphi_1\} \vdash \{\varphi_2\} \cup \Delta}{\Gamma \vdash \{\varphi_1 \rightarrow \varphi_2\} \cup \Delta}$$

which is a statement of the *deduction theorem* for our proof system. It is important enough that we spell it out.

Theorem 2. *If $\Gamma \cup \{\varphi_1\} \vdash \varphi_2$ then $\Gamma \vdash \varphi_1 \rightarrow \varphi_2$.*

The proof of this rule involves proving the soundness of the two inference rules used to derive it. We leave this to the reader as a useful exercise. Instead, we state another rule that will come in handy in the next section:

$$\frac{\Gamma \vdash \varphi \quad \Gamma \vdash \neg\varphi}{\Gamma \vdash \psi}$$

where ψ is *any* sentence. This rule deserves a comment. Why are we allowed to use it? Because whenever $\Gamma \models \varphi$ and $\Gamma \models \neg\varphi$, it is also true that $\Gamma \models \psi$. The reason is simply that the only way for both $\Gamma \models \varphi$ and $\Gamma \models \neg\varphi$ to be true is that Γ has no models. Does ψ hold in all models of Γ ? Why, yes!

6 Characterizing First-Order Logic

In the previous section we have achieved an important goal: we have described a deductive calculus in which we can write a proof of any valid first-order sentence. Less surprising, but at least as important, is the fact that if we get a proof for a sentence, then this sentence is valid. Let us also recall that, thanks to the deduction theorem, checking logical consequence ($\Gamma \models \varphi$) can be reduced to checking validity ($\models \varphi$). Hence, we also have a proof procedure that allows us to check whether $\Gamma \models \varphi$. We now proceed to explore some important consequences of our discovery.

6.1 Consistency, Soundness, and Completeness

A set of sentences Γ is *inconsistent* if, and only if, for some sentence φ , both $\Gamma \vdash \varphi$ and $\Gamma \vdash \neg\varphi$. If Γ is consistent, on the other hand, there exists a sentence φ such that $\Gamma \not\vdash \varphi$. Clearly, the inconsistency of a set of sentences depends on the proof system in use. In the one introduced in Section 5.2 an inference rule guarantees that if a set of assumptions is inconsistent, then any sentence ψ can be proved from it. This conclusion, however, applies under very general assumptions. In a proof system in which modus ponens is an inference rule and the tautology $\varphi \rightarrow (\neg\varphi \rightarrow \psi)$ is a theorem, one can apply modus ponens twice, first with φ to prove $\neg\varphi \rightarrow \psi$, and then with that and $\neg\varphi$ to prove ψ . Unlike the proponents of *paraconsistent logics*, in these notes we embrace the view that inconsistency has disastrous consequences and we want to avoid it like the plague. That desire leads us to define the soundness of a deductive calculus.

A deductive calculus is *sound* if, and only if, $\Gamma \vdash \varphi \Rightarrow \Gamma \models \varphi$. In a sound deductive calculus we can only prove from Γ sentences that are true in all models of Γ : If we find a proof of φ from Γ , then φ is true in all models of Γ .

Theorem 3. *A deductive calculus is sound if, and only if, every satisfiable set of sentences is consistent.*

Proof. Suppose the deductive calculus is sound. If there is a model S of Γ (i.e., Γ is satisfiable), then in this model every sentence φ is either true or false; that is, either $S \models \varphi$ or $S \not\models \varphi$. Fix one φ and, without loss of generality, suppose the latter. By definition of the consequence relation, $\Gamma \not\models \varphi$, which implies, by the contrapositive of the soundness condition, that $\Gamma \not\vdash \varphi$. In words, there is no derivation of φ from Γ in the deductive calculus. Since φ was chosen arbitrarily, we conclude that for every φ , either $\Gamma \not\vdash \varphi$ or $\Gamma \not\vdash \neg\varphi$, which is equivalent to saying that Γ is consistent.

In the other direction, suppose the deductive calculus is unsound. Then there are Γ and φ such that $\Gamma \vdash \varphi$ but $\Gamma \not\models \varphi$. The latter implies that there is a structure S such that $S \models \Gamma$ and $S \models \neg\varphi$. This means that $\Gamma \cup \{\neg\varphi\}$ is satisfiable (namely, in S). If it were also consistent, we'd have $\Gamma \cup \{\neg\varphi\} \vdash \varphi$, but this would contradict $\Gamma \vdash \varphi$, because a proof of φ from Γ is also a proof of φ from $\Gamma \cup \{\neg\varphi\}$. We have to accept that some satisfiable sets may be inconsistent. \square

A deductive calculus is complete if, and only if, $\Gamma \models \varphi$ implies $\Gamma \vdash \varphi$.

Theorem 4. *If a deductive calculus is complete, then any consistent set of sentences is satisfiable (has a model).*

Proof. If Γ is consistent, there is a sentence φ such that $\Gamma \not\vdash \varphi$; completeness implies that $\Gamma \not\models \varphi$. This means that there is a model S of Γ in which φ is false. This model S witnesses the satisfiability of Γ . \square

In the following lemma we don't pin down the deductive system completely, but we assume that it is strong enough to deduce any sentence from a contradiction.⁵

Lemma 1. *If, for some Γ and φ , $\Gamma \not\vdash \varphi$ then $\Gamma \cup \{\neg\varphi\}$ is consistent.*

Proof. By contraposition. If $\Gamma \cup \{\neg\varphi\}$ is inconsistent, then $\Gamma \cup \{\neg\varphi\} \vdash \varphi$ because any sentence can be proved from inconsistent assumptions. By Theorem 2, $\Gamma \vdash \neg\varphi \rightarrow \varphi$, from which we derive $\Gamma \vdash \varphi$. \square

Theorem 5. *If every consistent set of sentences is satisfiable, then the deductive calculus is complete.*

Proof. Suppose for some Γ and φ , $\Gamma \not\vdash \varphi$. Then by Lemma 1, $\Gamma \cup \{\neg\varphi\}$ is consistent; therefore it has a model. This means that there is a model of Γ in which φ is false, which implies $\Gamma \not\models \varphi$. By contraposition we obtain completeness. \square

The deductive calculus we have developed in Sections 5.1 and 5.2 is both sound and complete. Though we have not proved this claim ourselves, some very smart people say so, and we'll take their word for it.

Theorem 6 (Completeness). *There are sound and complete deductive calculi for first-order logic.*

⁵This ability is known as *Ex falso quodlibet*.

In particular, it was Kurt Gödel who first proved that first-order logic is complete. It's important that we understand what this claim actually entails. In particular, we should not forget that logical consequence in first-order logic is *semidecidable*:

Theorem 7 (Church). *First-order logic is undecidable.*

Intuitively, we can enumerate the proofs until we find the right one (the set of proofs is only countably infinite) but if there is no proof, we don't know when to stop. If we impose enough restriction on the structure of the formulae that are allowed, we may get a decidable *fragment* of first-order logic [5]. The truth of the matter, though, is that these restrictions need to be rather severe. Another limitation of first-order logic is the subject of Section 6.3.

6.2 Three Important Theorems

The completeness theorem for first-order logic has an immediate yet important consequence in the compactness theorem.

Theorem 8 (Compactness). *A set of first-order sentences has a model if, and only if, every finite subset of it has a model.*

Proof. If Γ is an unsatisfiable set of sentences, then it is inconsistent; therefore there exists a proof of some contradiction ψ from it: $\Gamma \vdash \psi$. This proof is finite in length by definition; therefore it only involves finitely many sentences from Γ . Said otherwise, it is a proof that a finite subset Δ of Γ is inconsistent. But then, Δ is unsatisfiable. In sum, if Γ is unsatisfiable, it has a finite subset that is unsatisfiable. The other direction of the proof is straightforward: if Γ is satisfiable, then all its subsets must be. \square

An equivalent formulation of compactness is that a theory Γ is consistent if, and only if, every finite subset of Γ is consistent.⁶ Among the uses of the compactness theorem are the proofs that certain properties are not expressible in first-order languages. This corollary of Theorem 8 comes in handy for that purpose.

Corollary 1. *If a first-order sentence φ has finite models of arbitrary cardinality, then it has an infinite model.*

Proof. The first-order sentence

$$\psi_n = \exists x_1, \dots, x_n. \bigwedge_{1 \leq i < j \leq n} x_i \neq x_j$$

is satisfied by precisely the models of cardinality greater than or equal to n (for $n > 1$). Therefore, the countable first-order theory $\{\psi_n \mid n > 1\}$ only has infinite models. Suppose φ has finite models of arbitrarily large cardinality, but

⁶For the connection to the topological notion of compact space, see [12, Chapter VI, Exercise 2.5].

no infinite model, and consider the theory $\Gamma = \{\varphi\} \cup \{\psi_n \mid n > 1\}$. All finite subsets of Γ have models; they have to be larger than the larger n such that ψ_n is in the subset, but that's OK because φ has arbitrarily large finite models. By compactness, then, Γ should have models too. However, the full set of ψ_n 's forbid finite models, and φ has no infinite ones. The assumption that such a φ exists must be rejected and the corollary follows. \square

Corollary 1 lets us show that first-order logic cannot express graph reachability. One proof goes as follows. Suppose there existed a formula $\psi(x, y)$ with two free variables in the language of graphs that is true of two vertices of a graph if and only if the vertices are connected. Then $\forall x, y. \psi(x, y)$ is a sentence asserting that a graph is *strongly connected*.

We can write sentences φ_1 and φ_2 asserting that all nodes of a graph have exactly one incoming and one outgoing edge. In fact, we saw one of them in Section 4.1, and the other is similar. The graphs that satisfy $\varphi_1 \wedge \varphi_2 \wedge \forall x, y. \psi(x, y)$ are cycles. For any n , there exists a cycle with n vertices; hence by Corollary 1, there should be an infinite cycle, but that's not possible! (You should try to work out the details of why it is so.)

In Section 5.1, p. 18 we have seen a first-order theory that only has infinite models and in the proof of Corollary 1 we have made use of the theory of all infinite structures. At the other end of the spectrum, a model of this sentence must have a domain of cardinality 1:

$$\forall x, y. x = y .$$

Corollary 1, however, immediately shows that there is no first-order theory of finite structures. Such theory would have models of arbitrary sizes, but no infinite model. It also follows that we do need infinitely many sentences to characterize infinite structures. Otherwise, negating their conjunction would give us a sentence that characterizes all finite structures.

Another central result of first-order logic that deals with the cardinality of models is the following.

Theorem 9 (Löwenheim-Skolem). *If a countable first-order theory has an infinite model, then it has a countably infinite model too.*

The method of the semantic tableau of Section 5.1 explains why this theorem holds when the theory consists of one sentence only. In a nutshell, if the sentence is satisfiable, at least one branch of its tableau remains open. Such a branch defines a model of the sentence. While the branch may be infinite, it is only countably so, as long as there are only countably many symbols.

We leave a real proof and the extension to more general settings to the interested reader. Instead we turn to an immediate, yet momentous consequence of the more general theorem that asserts that if a theory has an infinite model, it has models of all infinite cardinalities. We say that a theory is *categorical* if it has exactly one model, up to isomorphism.

Corollary 2. *No first-order theory with an infinite model is categorical.*

Said otherwise, first-order logic cannot control the type of infinity of a model. An interesting questions you may want to ponder: What does Corollary 2 say about first-order Peano arithmetic? Also consider the so-called *Skolem's paradox*: Set theory can be axiomatized in first-order logic (in more than one way). Such first-order theories have a countable model. Cantor's theorem, on the other hand, proves the existence of uncountable sets. How can uncountable sets exist in a countable model? Tackling these questions is beyond the scope of these notes, but hopefully your curiosity will be piqued.

We conclude this section with the statement of a theorem that we may regard as the ultimate vindication of the importance of the compactness and Löwenheim-Skolem theorems.

Theorem 10 (Lindström). *First-order logic is the strongest logic for which the compactness and Löwenheim-Skolem theorems hold.*

Here, strength refers to a logic's ability to characterize classes of structures. Informally, we could say that if you want both compactness and Löwenheim-Skolem, you can't go beyond first-order. For instance, graph reachability is expressible in second-order logic, in which neither compactness nor Löwenheim-Skolem holds. (Consequently, second-order logic is *incomplete*.) This is the point of departure of *descriptive complexity theory* [24].

While a proof of Lindström's Theorem is beyond the scope of these notes, it is not difficult to see that compactness does not hold in second-order logic. If we allow variables that range over relations, we can write a single sentence that characterizes infinite structures. We assert the existence of a binary relation on the universe of discourse D that is transitive and irreflexive, and whose domain is the whole of D :

$$\varphi = \exists R. (\forall x, y, z. (R(x, y) \wedge R(y, z)) \rightarrow R(x, z)) \wedge (\forall x. \neg R(x, x) \wedge \exists y. R(x, y)) .$$

Contrast this situation with the one of Corollary 1, where we used the theory $\{\psi_n \mid n > 1\}$ with infinitely many sentences to characterize infinite structures. If we consider $\{\neg\varphi\} \cup \{\psi_n \mid n > 1\}$, we realize it is an unsatisfiable set of sentences such all its finite subsets are satisfiable. This denies compactness. Now, recall that we inferred compactness of first-order logic from its completeness. By contraposition, second-order logic has no complete deductive calculus.

6.3 Gödel's Incompleteness Theorems

In Section 6.1 we have seen that there exist complete deductive calculi for first-order logic. Here we introduce a related but different notion: completeness of a theory. A theory T is complete if, and only if, for every sentence φ of the language, either $T \models \varphi$ or $T \models \neg\varphi$.

Theorem 11 (Incompleteness 1). *In every first-order theory T that is recursively enumerable and consistent, and that is powerful enough to describe arithmetic, there is a sentence φ such that in T it is not possible to prove either φ or $\neg\varphi$.*

Table 1: Encoding of logical symbols.

symbol	\neg	\vee	\rightarrow	\exists	$=$	0	s	$($	$)$	$,$
number	1	2	3	4	5	6	7	8	9	10

Equivalently, there exists a model of T and φ as well as a model of T and $\neg\varphi$. The meaning of “powerful enough to describe arithmetic” is vague. Suffice it to say that Peano arithmetic meets that requirement.

The completeness result (Theorem 6) concerns the deductive calculus. The incompleteness result concerns the theory. It says that there is no way to find a set of recursively enumerable axioms so that the resulting first-order theory describes arithmetic well enough that all its models essentially agree with arithmetic on what sentences are theorems.

Truth in the standard model of arithmetic is not equivalent to provability in any recursively enumerable axiomatized first-order theory.

Theorem 12 (Incompleteness 2). *The consistency of a first-order theory of arithmetic can be written in the theory and proved equivalent to the formula that Gödel proved to be unprovable. Hence, consistency cannot be proved inside the theory.*

The proof of the second incompleteness theorem is relatively straightforward once the first theorem is established. However, that is not easy. Here we give a few hints on how Gödel celebrated proof proceeds.

We start by introducing Gödel’s numbers, which allow us to uniquely encode proofs as positive integers. Let p_i be the i -th prime. Let $\gamma(\sigma)$ be the Gödel number of σ . Assign positive integers to the logical symbols of the alphabet. For instance, following [31], we have the encoding of Table 1. Numerical variables are assigned prime integers from 11 onward. So, if the variables are x_1, x_2, \dots , then x_1 is encoded as 11, x_2 is encoded as 13, x_3 is encoded as 17, and so on. Propositional variables are assigned the squares of primes greater than 10. Finally, predicate and function symbols are assigned the cubes of primes greater than 10.

Next, we define the encoding of a formula. Suppose the formula is a sequence $\sigma_1, \dots, \sigma_n$ of n symbols. As before, p_i is the i -th prime. Then, the Gödel number of $\sigma_1, \dots, \sigma_n$ is

$$\gamma(\sigma_1, \dots, \sigma_n) = \prod_{1 \leq i \leq n} p_i^{\gamma(\sigma_i)} .$$

Finally, the Gödel number of a sequence of formulae f_1, \dots, f_m is

$$\gamma(f_1, \dots, f_m) = \prod_{1 \leq i \leq m} p_i^{\gamma(f_i)} .$$

The important property of this mapping is that it is *injective*. As an example, suppose p is the first propositional variable:

$$\gamma((p \vee p) \rightarrow p) = 2^8 \cdot 3^{11^2} \cdot 5^2 \cdot 7^{11^2} \cdot 11^9 \cdot 13^3 \cdot 17^{11^2} .$$

In the other direction:

$$\gamma^{-1}(243000000) = \gamma^{-1}(2^6 \cdot 3^5 \cdot 5^6) = "0 = 0" .$$

Not all numbers are Gödel numbers. (Try 77760.)

Gödel's numbers allow the *arithmetization of syntax*. What is meant by that is that a theory that deals with arithmetic can also express sentences about its own sentences; in other words, it can express *metamathematical* sentences. It is not too difficult to see that " φ_1 is a prefix of φ_2 " translates into something related to " $\gamma(\varphi_1)$ divides $\gamma(\varphi_2)$." Likewise, with a little faith one can believe that a (messy) arithmetic formula encodes " $\varphi_1, \dots, \varphi_n$ is a proof of φ_n ."

We can now attempt a very short summary of Gödel's proof. There is a two-variable formula $\pi(x, y)$ in the theory of arithmetic such that $\pi(m, n)$ holds if, and only if, n is a proof of m . This fact can be used to write a formula G that encodes the sentence "formula G cannot be proved."

A slightly longer summary of the proof goes like this: There is a formula $\omega(x, y)$ such that $\omega(m, n)$ holds if, and only if, m is the Gödel number of a formula ϕ that has one free variable and n is the Gödel number of a proof of $\phi(m)$.

Let $\psi(x)$ be the formula $\forall y. \neg \omega(x, y)$. If ϕ is a formula with one free variable with Gödel number m , then $\psi(m)$ tells us that there is no proof of $\phi(m)$. (Because there is no y that is the Gödel number of such a proof.)

Let p be the Gödel number of ψ itself, and let G be the sentence $\psi(p)$. Then G asserts its own unprovability. In fact, $\psi(p)$ tells us that there is no proof of the formula $\phi(p)$, where $\gamma(\phi) = p$. That is, it tells us that there is no proof of $\psi(p)$.

Since G asserts its own unprovability, it must be unprovable (since a proof of G would be a proof that G has no proof). Since G asserts its unprovability and is unprovable, it is true. Since it is true, it cannot be disproved.

Gödel's second incompleteness theorem says that C implies G , where C is the statement that the logical system is consistent, and G is the sentence "This sentence is not provable." Suppose G were false. Then "This sentence is not provable" would be provable and the logic system would be inconsistent because it would prove a false statement. By the contrapositive, if the logic system is consistent, G must be true. Because of the implication we have proved, if we could prove C within the logic system, we could apply *modus ponens* and prove G , which is known not to be possible.

7 The Barber's Paradox

In a town lives a barber who is a man and who shaves all (and only) the men of the town who do not shave themselves. Does the barber shave himself?

If we assume that the barber shaves himself we contradict the constraint that only those who don't shave themselves be shaven by the barber. On the other hand, if the barber doesn't shave himself, we contradict the constraint that all those who don't shave themselves be shaven by the barber.


(1)	$\exists y . \forall x . S(x, y) \leftrightarrow \neg S(x, x)$	
(2)	$\forall x . S(x, b) \leftrightarrow \neg S(x, x)$	(1)
(3)	$S(b, b) \leftrightarrow \neg S(b, b)$	(2)
		

Figure 10: Semantic tableau for the barber’s paradox.

Since we have reached a contradiction, we have to look at the assumptions we made. Of course, we also need to take a careful look at our argument. If we made no mistakes in the deductions, though, we conclude that our assumptions are inconsistent.

For the barber’s paradox, the assumption that we need to reject is that such a barber exists. We now show the inconsistency of such assumption with the help of first-order logic. To that effect, let our universe of discourse be the men living in that special town and let $S(x, y)$ be the relation that is true when x is shaven by y . We claim that

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

We can read this sentence in English as follows: there exists no man in town who shaves every man who doesn’t shave himself. We now prove this claim as a theorem of first-order logic, in a way that does not make any assumption about the town where the barber is supposed to live. Our proof is not going to rely on the meaning of “shaves” either. What we are really going to prove is this more general statement:

Given any set M and any binary relation $S \subseteq M \times M$, there is no member y of M such that $S(x, y) \leftrightarrow \neg S(x, x)$ holds for all members x of M .

Our proof uses the semantic tableau method, which is shown in Figure 10. We build the tableau for the negation of the sentence we wish to prove valid. We first instantiate the existential quantifier with an arbitrary constant b , and then we instantiate the universal quantifier with the same constant. If we were pedants, we would expand the “if, and only if” operator. The tableau would branch, but it would still close very quickly.

With the help of our little theorem we have shown that no such barber as the one postulated in the paradox exists. From inconsistent assumptions we can derive contradictions, and that’s exactly what we did. We next shown how the barber’s paradox relates to set theory and to the paradox about normal sets of Section 2.

Let S be the set membership relation; that is, $S(x, y)$ if, and only if, x and y are sets and $x \in y$. Then our theorem says that

$$\neg \exists y . \forall x . (x \in y) \leftrightarrow (x \notin x) .$$

If we read aloud this sentence, it says that there is no set y that contains all sets that do not contain themselves. Said otherwise, the set of all normal sets doesn’t

exist. Assuming otherwise leads to a contradiction. No wonder we stumbled into one.

At this point we may feel that these paradoxes were little more than curiosities and that now that we’ve figured them out we should just move on. We would be missing an important observation, though. Many find it easier to accept that no barber who satisfies the given constraints exists, than to accept that there is no set of all normal sets.

In fact, in the early attempts to systematize set theory the assumption was made that one can always build the set of those individuals that have a given property. In formula, we write

$$\exists y . \forall x . (x \in y) \leftrightarrow P(x) \text{ ,}$$

where P is a formula with x as its only free variable. This assumption—which looked reasonable—goes under the name of *principle of comprehension* (or *principle of abstraction*). At this point of our narrative we can easily notice that taking $P(x)$ to be $x \notin x$ directly contradicts our first-order theorem. If we want set theory to be consistent, we have to renounce the (unrestricted) principle of comprehension and accept that certain “things” we can talk about in natural languages are not sets.

One important development that was spurred by the paradoxes we have examined (though not only by them) is the axiomatization of set theory. The axioms of ZFC (Zermelo-Fraenkel set theory with Choice) are carefully selected so that the “set” of all normal sets is not a set. In particular, these axioms include *restricted comprehension*. (Zermelo called it the *axiom of selection* and this name is still in use.) Restricted comprehension allows one to form a *subset* of a given set by selecting its elements that have a given property P .

Appendix

A Basic Facts about Propositional Logic

Warning: If the rest of this document is a compendium, this appendix is more aptly dubbed a concentrate. It is not supposed to teach you propositional logic. Instead, it is meant to be a check-list of useful facts about propositional logic and to provide a summary of the notation we use in this course.

Propositional logic deals with the combination of *sentence symbols* (or *propositions*) by means of connectives. The *Boolean connectives* we use are shown in Table 2, in which we define all other connectives in terms of conjunction and negation. This choice is not unique. Among the other sets of *adequate* connectives are disjunction and negation, as well as implication and negation. In digital logic design it is customary to define connectives NAND and NOR (the negation of conjunction and disjunction, respectively). If one allows NANDs with one argument, then all other connectives can be defined in terms of NAND. (Try it!) The same is true of NOR.

name	symbol	a.k.a.	definition
conjunction	\wedge	AND	
negation	\neg	NOT	
disjunction	\vee	OR	$p \vee q = \neg(\neg p \wedge \neg q)$
implication	\rightarrow		$p \rightarrow q = \neg p \vee q$
equivalence	\leftrightarrow	XNOR	$p \leftrightarrow q = (p \rightarrow q) \wedge (q \rightarrow p)$
symmetric difference	$p \oplus q$	XOR	$(p \wedge \neg q) \vee (\neg p \wedge q)$

Table 2: Boolean connectives.

Note that “implication” as defined in Table 2 has nothing to do with cause and effect. In other words, when we write $p \rightarrow q$ we do not mean that q is true *because* p is; we only claim that q is true *whenever* p is. If the need arises to stress this distinction, people use the name *material implication* for the connective denoted by \rightarrow . While the definition of material implication may seem artificial, it is the only meaningful definition that is *truth functional*. That is, it is the only meaningful definition that can be described a truth-table with inputs p and q .

The sentence symbols go under various names; perhaps the most common of them is *Boolean variables*. No matter how we call them, they can take values true (\top) or false (\perp). Some texts use 1 for true and 0 for false, and moreover, they use \cdot for conjunction, $+$ for disjunction, and a line above a sentence for negation. When dealing with first-order logic, though, numbers also appear in formulae, and the notation we have chosen prevents confusion.

Given a formula of propositional logic, one question we often ask is whether it is *satisfiable*; that is, whether we can interpret each sentence symbol as either true or false in such a way that the overall formula takes on the value true. A propositional formula that is true no matter how the sentence symbols in it are interpreted is a *tautology*. A problem closely related to satisfiability checking is tautology checking. A formula F is satisfiable if and only if $\neg F$ is not a tautology.

A conceptually simple way to check satisfiability is to try all possible interpretations of the sentence symbols in turn. The exhaustive enumeration of all possible truth assignments corresponds to building a *truth table* for the formula. This method is quite impractical for the problems that arise in verification, because the number of different interpretations grows exponentially with the number of sentence symbols, but at least in principle, it works for any propositional formula. We’ll see that this is not true for first-order logic. We’ll also talk at length about clever ways to check propositional satisfiability.

Instead of enumerating assignments, we can use theorems of propositional logic to simplify the formula until it’s easy to decide whether it is satisfiable or tautologous. By building truth tables, it is not difficult to check that the

following are *identities* of propositional logic. That is, they hold for all x and y .

$$\begin{array}{ll}
x \wedge y = y \wedge x & x \vee y = y \vee x \\
x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z) & x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z) \\
x \wedge \top = x & x \vee \perp = x \\
x \wedge \neg x = \perp & x \vee \neg x = \top .
\end{array}$$

These identities are known as *Huntington's postulates* [23] and can be taken as a set of axioms for propositional logic. From them, all identities of propositional logic can be derived. Among the most useful,

$$\begin{array}{l}
x \wedge x = x \\
x \vee x = x \\
x \wedge (y \wedge z) = (x \wedge y) \wedge z \\
x \vee (y \vee z) = (x \vee y) \vee z \\
\neg \neg x = x \\
x \vee (x \wedge y) = x \\
x \wedge (x \vee y) = x \\
x \vee (\neg x \wedge y) = x \vee y \\
x \wedge (\neg x \vee y) = x \wedge y \\
\neg(x \vee y) = \neg x \wedge \neg y \\
\neg(x \wedge y) = \neg x \vee \neg y \\
(x \wedge y) \vee (\neg x \wedge z) \vee (y \wedge z) = (x \wedge y) \vee (\neg x \wedge z) \\
(x \vee y) \wedge (\neg x \vee z) \wedge (y \vee z) = (x \vee y) \wedge (\neg x \vee z) .
\end{array}$$

Note how all these theorems satisfy the *duality* property. If you exchange \wedge and \vee and also exchange \top and \perp in an identity, you get another identity. This is because the postulates satisfy duality, which means that the dual of the proof of an identity proves the dual of the identity. Note also that there are other choices of axioms besides Huntington's. (In fact, Huntington himself came up with more than one set of axioms.)

A sentence symbol or its negation is a *literal*. We often consider propositional formulae of a restricted form. A conjunction of literals is called a *cube*, while a disjunction of literals is called a *clause*. Two types of *two-level* formulae are of particular interest: a formula in *disjunctive normal form* (DNF) is a disjunction of cubes. Another popular name for a DNF formula is *sum of products*. A formula in *conjunctive normal form* (CNF) is a conjunction of clauses. It is also known as a *product of sums*.

B Examples of Proofs in Peano Arithmetic

We show how to derive a few familiar properties of the natural numbers from the axioms of Peano arithmetic, which we repeat here for convenience:

1. $\forall x . s(x) \neq 0$.
2. $\forall x, y . x \neq y \rightarrow s(x) \neq s(y)$.
3. For each formula $\varphi(x, v_1, \dots, v_n)$ with free variables x, v_1, \dots, v_n ,
$$\forall v_1, \dots, v_n . [\varphi(0, v_1, \dots, v_n) \wedge (\forall y . \varphi(y, v_1, \dots, v_n) \rightarrow \varphi(s(y), v_1, \dots, v_n))] \rightarrow \forall x . \varphi(x, v_1, \dots, v_n) .$$
4. $\forall x . x + 0 = x$.
5. $\forall x, y . x + s(y) = s(x + y)$.
6. $\forall x . x \cdot 0 = 0$.
7. $\forall x, y . x \cdot s(y) = (x \cdot y) + x$.

One may feel that there is no need to prove that addition of natural numbers is commutative or, perhaps, that our proofs are very cumbersome and opaque for such obvious claims as they prove. The point of the proofs, however, is to show that those claims follow from Peano's axioms, so that every model of those axioms must have, say, commutative and associative "plus" and "times" operations.

Given our objective, we'll be quite detailed in our proofs so that we can see how the axioms are applied. We first tackle associativity of addition:

Theorem 13.

$$\forall x, y, z . (x + y) + z = x + (y + z) .$$

Proof. We proceed by induction on z ; that is, we prove by induction that

$$P(z) = \forall x, y . (x + y) + z = x + (y + z)$$

is true of all natural numbers. The axiom of induction tells us that this is true only if $P(0)$ holds. Therefore we check whether

$$\forall x, y . (x + y) + 0 = x + (y + 0)$$

holds. The axioms tell us that both sides simplify to $x + y$. Since equality is reflexive, we conclude that $P(0)$ holds. Next the axiom of induction asks us to show that if $P(z)$ holds then $P(s(z))$ also holds.

$$\begin{aligned} P(z) &\Leftrightarrow \forall x, y . (x + y) + z = x + (y + z) \\ &\Leftrightarrow \forall x, y . s((x + y) + z) = s(x + (y + z)) \\ &\Leftrightarrow \forall x, y . s((x + y) + z) = x + s(y + z) \\ &\Leftrightarrow \forall x, y . s((x + y) + z) = x + (y + s(z)) \\ &\Leftrightarrow \forall x, y . (x + y) + s(z) = x + (y + s(z)) \\ &\Leftrightarrow P(s(z)) . \end{aligned}$$

Since the inductive step went through, we conclude that $\forall z . P(z)$. □

In the proofs to come, we make use of these two lemmas, whose easy proofs are left to the reader.

Lemma 2. $\forall x, y, z. x + z = y + z \leftrightarrow x = y.$

Lemma 3. $\forall x. 0 + x = x.$

Note that we need to prove that $0 + x = x$ because the axioms only give us $x + 0 = x$ and we haven't proved commutativity of addition yet. Toward that end, we establish another lemma.

Lemma 4. $\forall x, y. s(x) + y = x + s(y).$

Proof. We prove by induction that

$$P(y) = \forall x. s(x) + y = x + s(y)$$

holds for all natural numbers. For $P(0)$ we write:

$$\begin{aligned} P(0) &\Leftrightarrow \forall x. s(x) + 0 = x + s(0) \\ &\Leftrightarrow \forall x. s(x) = x + s(0) \\ &\Leftrightarrow \forall x. s(x) = s(x + 0) \\ &\Leftrightarrow \forall x. s(x) = s(x) , \end{aligned}$$

which is true because equality is reflexive. For the inductive step, we have:

$$\begin{aligned} P(y) &\Leftrightarrow \forall x. s(x) + y = x + s(y) \\ &\Leftrightarrow \forall x. s(s(x) + y) = s(x + s(y)) \\ &\Leftrightarrow \forall x. s(x) + s(y) = x + s(s(y)) \\ &\Leftrightarrow P(s(y)) . \end{aligned}$$

Therefore the lemma is proved. □

At this point we can quickly prove commutativity of addition.

Theorem 14.

$$\forall x, y. y + x = x + y .$$

Proof. The base case follows from Lemma 3, while the inductive step is dispatched with the help of Lemma 4.

$$\begin{aligned} P(x) &\Leftrightarrow \forall y. y + x = x + y \\ &\Leftrightarrow \forall y. s(y + x) = s(x + y) \\ &\Leftrightarrow \forall y. y + s(x) = x + s(y) \\ &\Leftrightarrow \forall y. y + s(x) = s(x) + y \\ &\Leftrightarrow P(s(x)) . \end{aligned}$$

□

We now turn our attention to multiplication. We don't need induction—for a change—to show that $s(0)$ is the right multiplicative identity.

Theorem 15.

$$\forall x . x \cdot s(0) = x \quad .$$

Proof.

$$\begin{aligned} \forall x, y . x \cdot s(y) = x + (x \cdot y) &\Rightarrow \forall x . x \cdot s(0) = x + (x \cdot 0) \\ &\Leftrightarrow \forall x . x \cdot s(0) = x + 0 \\ &\Leftrightarrow \forall x . x \cdot s(0) = x \quad . \end{aligned}$$

□

Using the identity $s(x) = s(0) + x$, it is also easy to show that $s(0)$ is the left multiplicative identity as well. Now we show that multiplication distributes over addition:

Theorem 16.

$$\forall x, y, z . x \cdot (y + z) = (x \cdot y) + (x \cdot z) \quad .$$

Proof. As usual, we proceed by induction, showing that

$$P(z) = \forall x, y . x \cdot (y + z) = (x \cdot y) + (x \cdot z)$$

holds for all natural numbers. We leave the easy proof that $P(0)$ holds to the reader, and show that $P(z)$ is equivalent to $P(s(z))$.

$$\begin{aligned} P(s(z)) &\Leftrightarrow \forall x, y . x \cdot (y + s(z)) = (x \cdot y) + (x \cdot s(z)) \\ &\Leftrightarrow \forall x, y . x \cdot s(y + z) = (x \cdot y) + (x \cdot s(z)) \\ &\Leftrightarrow \forall x, y . x + (x \cdot (y + z)) = (x \cdot y) + (x \cdot s(z)) \\ &\Leftrightarrow \forall x, y . x + (x \cdot (y + z)) = (x \cdot y) + (x + (x \cdot z)) \\ &\Leftrightarrow \forall x, y . x + (x \cdot (y + z)) = ((x \cdot y) + x) + (x \cdot z) \\ &\Leftrightarrow \forall x, y . x + (x \cdot (y + z)) = (x + (x \cdot y)) + (x \cdot z) \\ &\Leftrightarrow \forall x, y . x + (x \cdot (y + z)) = x + ((x \cdot y) + (x \cdot z)) \\ &\Leftrightarrow \forall x, y . x \cdot (y + z) = (x \cdot y) + (x \cdot z) \\ &\Leftrightarrow P(z) \quad . \end{aligned}$$

□

The proof that $\forall x, y, z . (x + y) \cdot z = (x \cdot z) + (y \cdot z)$ is similar. Next, we show that multiplication is associative.

Theorem 17.

$$\forall x, y, z . (x \cdot y) \cdot z = x \cdot (y \cdot z) \quad .$$

Proof. Once again we leave the base case to the reader and we use the distributivity we just proved to show that $P(z)$ is equivalent to $P(s(z))$.

$$\begin{aligned}
P(s(z)) &\Leftrightarrow \forall x, y. (x \cdot y) \cdot s(z) = x \cdot (y \cdot s(z)) \\
&\Leftrightarrow \forall x, y. (x \cdot y) + ((x \cdot y) \cdot z) = x \cdot (y \cdot s(z)) \\
&\Leftrightarrow \forall x, y. (x \cdot y) + ((x \cdot y) \cdot z) = x \cdot (y + (y \cdot z)) \\
&\Leftrightarrow \forall x, y. (x \cdot y) + ((x \cdot y) \cdot z) = (x \cdot y) + (x \cdot (y \cdot z)) \\
&\Leftrightarrow \forall x, y. (x \cdot y) \cdot z = x \cdot (y \cdot z) \\
&\Leftrightarrow P(z) .
\end{aligned}$$

□

Finally, we prove that multiplication is commutative.

Theorem 18.

$$\forall x. x \cdot y = y \cdot x .$$

Proof. For

$$P(y) = \forall x. x \cdot y = y \cdot x ,$$

we let the reader take care of $P(0)$ and we show that $P(y)$ is equivalent to $P(s(y))$.

$$\begin{aligned}
P(y) &\Leftrightarrow \forall x. x \cdot y = y \cdot x \\
&\Leftrightarrow \forall x. x + (x \cdot y) = x + (y \cdot x) \\
&\Leftrightarrow \forall x. x \cdot s(y) = x + (y \cdot x) \\
&\Leftrightarrow \forall x. x \cdot s(y) = (s(0) \cdot x) + (y \cdot x) \\
&\Leftrightarrow \forall x. x \cdot s(y) = (s(0) + y) \cdot x \\
&\Leftrightarrow \forall x. x \cdot s(y) = s(y) \cdot x \\
&\Leftrightarrow P(s(y)) .
\end{aligned}$$

□

We can also characterize the ordering relation \leq by the following axiom:

$$\forall x, y. x \leq y \leftrightarrow \exists z. x + z = y .$$

Induction then allows us to prove theorems like

$$\forall x, y. x \leq y \vee y \leq x ,$$

which says that \leq is a total order.

C Using Z3 to Prove First-Order Satisfiability

Let's return to the sentence

$$(\forall x. P(x) \vee Q(x)) \rightarrow ((\forall y. P(y)) \vee (\forall z. Q(z))) , \quad (2)$$

whose semantic tableau is shown in Figure 5. Let's use Z3 [11] to prove that this sentence is not valid. Specifically, let's check the satisfiability of the negation of the sentence. While Z3 is not designed to solve this type of problems in which there is no background theory, it does a better job than most SMT solvers and certainly adequate for our current purpose. The input to Z3 (using the Python bindings) is shown in Figure 11. If, as expected, Z3 finds the formula satisfiable, we ask for a model, which is shown in Figure 12. We define an *abstract sort* `D`, so as to let Z3 come up with its own choice of domain for the model. We then define two unary relations `P` and `Q` over `D`. To do so, we define *abstract functions* from `D` to the `Bool` sort. These functions are abstract because they are declared with `Function` and have no definition. Let us examine the model of Figure 12. (It is helpful at this point to set our expectations by reviewing Figure 5.) The first line of the model says that the domain `D` has two elements. In Figure 5 these two elements are called *a* and *b*. Here they are called `D!val!0` and `D!val!1`. The function definitions that follow say that `P` and `Q` disagree on the two elements of `D` and in particular, both `(P D!val!0)` and `(Q D!val!1)` are false. This is precisely the conclusion we came to with the semantic tableau of Figure 5.

As an exercise, you can encode the formulae of Figures 3–4 and verify that Z3 returns **unsat**. For the set of formulae of Figure 6, Z3 returns **unknown** after churning for a while. However, if the symmetry axiom is added, then Z3 quickly establishes unsatisfiability.

As a final example, let's use Z3 to prove the simple theorem about groups that we saw in Section 5. The encoding of the problem is shown in Figure 13. Z3 returns **unsat**.

D Exercises

Exercise 1. Write a FOL sentence in the language with equality as the only relation symbol that characterizes all structures with one element.

Solution. For example, $\forall x. \forall y. x = y$.

Exercise 2. Write a Python script that allows Z3 to prove that the sentence of Exercise 1 is satisfiable.

Solution. One possible solution is shown in Figure 14.

Exercise 3. Write a FOL sentence in a language with one relation symbol *R*, but without equality symbol, that characterizes all structures with more than two elements.

```

1 from z3 import *
2
3 def print_model(m):
4     """Prints current model."""
5     # Interpretation of sort D.
6     print(D, ': ', m[D])
7     # Pretty-print interpretation of P.
8     print(P, ': ')
9     for v in m[D]:
10         print(' ', v, '->', m.eval(P(v)))
11     # Pretty-print interpretation of Q.
12     print(Q, ': ')
13     for v in m[D]:
14         print(' ', v, '->', m.eval(Q(v)))
15
16 def check_validity(sentence):
17     """Checks validity and possibly prints counterexample.
18         """
19     print(sentence)
20     s = Solver()
21     # To check validity, we check satisfiability of the
22     # negation.
23     s.add(Not(sentence))
24     result = s.check()
25     if result == sat:
26         print('Not valid. Model for the negation:')
27         print_model(s.model())
28     elif result == unsat:
29         print('Valid.')
30     else:
31         print('Unable to solve.')
32
33 # Declare uninterpreted sort (the domain of discourse).
34 D = DeclareSort('D')
35 # Declare two uninterpreted predicates. In Z3 they have to
36 # be declared as uninterpreted functions returning Bool.
37 P = Function('P', D, BoolSort())
38 Q = Function('Q', D, BoolSort())
39 # Declare quantification "constants." (Behind the scenes,
40 # Z3 creates the quantified variables from them.)
41 x,y,z = [Const(s, D) for s in ['x','y','z']]
42
43 # Sentence whose validity we want to check.
44 sentence = Implies(ForAll([x], Or(P(x), Q(x))),
45                     Or(ForAll([y], P(y)),
46                         ForAll([z], Q(z))))
47
48 check_validity(sentence)

```

Figure 11: Z3 encoding of sentence (2).


```

D : [D!val!1, D!val!0]
P :
  D!val!1 -> True
  D!val!0 -> False
Q :
  D!val!1 -> False
  D!val!0 -> True

```

Figure 12: Z3 model for the negation of sentence (2).

Solution. One way to solve this problem is to force R to satisfy the axioms of equality:

$$\begin{aligned}
& \forall x . R(x, x) \\
& \forall x . \forall y . R(x, y) \rightarrow R(y, x) \\
& \forall x . \forall y . \forall z . (R(x, y) \wedge R(y, z)) \rightarrow R(x, z) .
\end{aligned}$$

We can then express the existence of three distinct elements in the structure by

$$\exists x . \exists y . \exists z . \neg R(x, y) \wedge \neg R(x, z) \wedge \neg R(y, z) .$$

The required sentence is the conjunction of this formula and the axioms of equality. We can write a simpler formula, though, if we observe that reflexivity is all R needs to satisfy from the axioms of equality. That is, our sentence can be simplified to

$$(\forall w . R(w, w)) \wedge \exists x . \exists y . \exists z . \neg R(x, y) \wedge \neg R(x, z) \wedge \neg R(y, z) .$$

In this sentence R may be interpreted as a relation different from equality. For example, it could be interpreted as a non-strict total order. A minor variant is to impose that R be *irreflexive*, which leads to the following sentence:

$$(\forall w . \neg R(w, w)) \wedge \exists x . \exists y . \exists z . R(x, y) \wedge R(x, z) \wedge R(y, z) .$$

Exercise 4. Write a Python script that allows Z3 to prove that the sentence of Exercise 3 is satisfiable. Include axioms that force the interpretation of R to differ from $x = y$ and $x \neq y$.

Solution. The input of Figure 15 forces R to be irreflexive and antisymmetric. Z3 finds a model with three elements in which R is a strict total order.

Exercise 5. Write a FOL sentence in the language of graphs extended with unary relation symbols C_1, \dots, C_k , which is true of a simple undirected graph $G = (V, E)$ if, and only if, G is k -colorable.

Solution. A possible solution is:

$$\begin{aligned}
& \left(\forall x . C_1(x) \vee \dots \vee C_k(x) \right) \wedge \\
& \left(\forall x, y . E(x, y) \rightarrow [\neg(C_1(x) \wedge C_1(y)) \wedge \dots \wedge \neg(C_k(x) \wedge C_k(y))] \right) .
\end{aligned}$$

```

1 from z3 import *
2
3 s = Solver()
4
5 # Signature: group operation, inverse, and identity.
6 G = DeclareSort('G')
7 op = Function('op', G, G, G)
8 inv = Function('inv', G, G)
9 e = Const('e', G)
10 x,y,z = [Const(v, G) for v in ['x','y','z']]
11
12 # Axioms.
13 # Behavior of (right) identity.
14 A1 = ForAll([x], x == op(x,e))
15 # Behavior of (right) inverse.
16 A2 = ForAll([x], e == op(x, inv(x)))
17 # Associativity.
18 A3 = ForAll([x,y,z],
19             op(x,op(y,z)) == op(op(x,y),z))
20
21 # Assumption: every element is its own inverse.
22 I = ForAll([x], e == op(x,x))
23
24 s.add(A1, A2, A3, I)
25
26 # Claim: commutativity.
27 C = ForAll([x,y], op(x,y) == op(y,x))
28
29 # Let's check that every group such that every
30 # element is its own inverse is Abelian.
31
32 result = s.check(Not(C))
33 if result == unsat:
34     print('The group is Abelian.')
35 else:
36     print('The group is not Abelian!')
37     print(s.model())

```

Figure 13: A simple theorem about groups.

Exercise 6. Prove that an infinite simple undirected graph G is k -colorable if, and only if, every finite subgraph of G is k -colorable. (This is the De Bruijn-Erdős Theorem, with application to the Hadwiger-Nelson problem.)

Solution. If a graph—either finite or infinite—is k -colorable, then all its subgraphs are. In the other direction we want to apply the compactness theorem, since we showed that k -colorability can be defined in FOL. To that effect we further extend the language of graphs by adding one constant symbol for each

```

1 from z3 import *
2
3 # The signature only contains the equality symbol.
4 D = DeclareSort('D')
5
6 s = Solver()
7 # Constants from which Z3 derives the quantification
   variables.
8 x = Const('x', D)
9 y = Const('y', D)
10 s.add(ForAll([x, y], x == y))
11
12 res = s.check()
13 if res == sat:
14     print('D:', s.model()[D])
15 else:
16     print(res)

```

Figure 14: Z3 encoding for Exercise 2.

vertex. Then G is described by the theory

$$\Gamma = \{E(a, b) \mid ([a]_G, [b]_G) \text{ is an edge of } G\} .$$

This theory has G as model, as well as graphs that have subgraphs isomorphic to G .

Let φ be the sentence that expresses k -colorability and consider $\Gamma_+ = \{\varphi\} \cup \Gamma$. Every finite subset of Γ is satisfiable because it is modeled by a finite subgraph of G . Since each subgraph is known to be k -colorable, adding φ to the subset of Γ does not change its satisfiability. That is, all finite subsets of Γ_+ are satisfiable. Then, by compactness, Γ_+ is satisfiable. The models of Γ_+ are k -colorable graphs that have subgraphs isomorphic to G . If Γ_+ has at least one model, then G is also a model, because a graph isomorphic to a subgraph of a k -colorable graph is k -colorable.

The De Bruijn-Erdős Theorem is known to depend on the axiom of choice. For us this dependence is rather hidden. We should go all the way back to Section 4.1, where we assumed that symbols are countable. If the graph has uncountably many vertices, adding one constant symbol for each vertex breaks that assumption and forces us to deal with uncountable signatures and the axiom of choice.

References

- [1] P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Springer, second edition, 2002.

```

1 from z3 import *
2
3 # The signature only contains a binary relation symbol.
4 D = DeclareSort('D')
5 R = Function('R', D, D, BoolSort())
6
7 s = Solver()
8
9 x = Const('x', D)
10 y = Const('y', D)
11 z = Const('z', D)
12
13 # R is irreflexive.
14 s.add(ForAll(x, Not(R(x, x))))
15 # There exist three distinct elements in D.
16 s.add(Exists([x, y, z], And(R(x, y), R(x, z), R(y, z))))
17 # R is antisymmetric. This formulation of antisymmetry is
18 # equivalent to the usual one for irreflexive relations.
19 # This axiom forces an interpretation of R different
20 # from "~(x = y)."
21 s.add(ForAll([x, y], Or(Not(R(x, y)), Not(R(y, x)))))
22
23 res = s.check()
24 if res == sat:
25     print(s.model())
26 else:
27     print(res)

```

Figure 15: Z3 encoding for Exercise 4.

- [2] J. Bell and M. Machover. *A Course in Mathematical Logic*. North-Holland, 1977.
- [3] M. Ben-Ari. *Mathematical Logic for Computer Science*. Springer, third edition, 2013.
- [4] G. S. Boolos, J. P. Burgess, and R. C. Jeffrey. *Computability and Logic*. Cambridge, 2007.
- [5] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer, 1997.
- [6] A. R. Bradley and Z. Manna. *The Calculus of Computation: Decision Procedures with Applications to Verification*. Springer, 2007.
- [7] S. N. Burris. *Logic for Mathematics and Computer Science*. Prentice Hall, Upper Saddle River, 1998.

- [8] P. J. Cameron. Gödel's theorem. In T. Gowers, editor, *The Princeton Companion to Mathematics*, pages 700–702. Princeton University Press, 2008.
- [9] C. C. Chang and H. J. Keisler. *Model Theory*. Dover, third edition, 2012.
- [10] M. D'Agostino, D. M. Gabbay, R. Hähnle, and J. Posegga, editors. *Handbook of Tableau Methods*. Springer, 1999.
- [11] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 337–340, Budapest, Hungary, Mar. 2008.
- [12] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Springer-Verlag, New York, second edition, 1994.
- [13] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, second edition, 2001.
- [14] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, second edition, 1996.
- [15] J. H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper & Row, 1986.
- [16] K. Gödel. *On Formally Undecidable Propositions of Principia Mathematica and Related Systems*. Basic Books, 1962.
- [17] L. H. Hackstaff. *Systems of Formal Logic*. Reidel, 1966.
- [18] P. Hájek. Kurt Gödel, completeness, incompleteness. *Journal of Physics: Conference Series*, 82, 2007.
- [19] J. Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge, 2009.
- [20] H. Hermes. *Introduction to Mathematical Logic*. Springer, 1973.
- [21] P. G. Hinman. *Fundamentals of Mathematical Logic*. A K Peters, 2005.
- [22] W. Hodges. *A Shorter Model Theory*. Cambridge, 1997.
- [23] E. V. Huntington. Sets of independent postulates for the algebra of logic. *Transactions of the American Mathematical Society*, 5(3):288–309, July 1904.
- [24] N. Immerman. *Descriptive Complexity*. Springer, 1999.
- [25] S. C. Kleene. *Mathematical Logic*. John Wiley & Sons, 1967.
- [26] K. Kunen. *The Foundations of Mathematics*. College Publications, 2012.

- [27] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.
- [28] D. Marker. *Model Theory: An Introduction*. Springer, 2002.
- [29] D. Marker. Logic and model theory. In T. Gowers, editor, *The Princeton Companion to Mathematics*, pages 635–646. Princeton University Press, 2008.
- [30] E. Mendelson. *Introduction to Mathematical Logic*. Chapman & Hall, fourth edition, 1997.
- [31] E. Nagel and J. R. Newman. *Gödel’s Proof*. New York University Press, 1958.
- [32] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.
- [33] C. C. Pinter. *A Book of Set Theory*. Dover, 2014.
- [34] A. Prestel and C. N. Delzell. *Mathematical Logic and Model Theory*. Springer, 2011.
- [35] W. Rautenberg. *A Concise Introduction to Mathematical Logic*. Springer, third edition, 2010.
- [36] S. Shapiro. *Foundations without Foundationalism: A Case for Second-order Logic*. Oxford, 1991.
- [37] J. R. Shoenfield. *Mathematical Logic*. Addison-Wesley, 1967.
- [38] R. M. Smullyan. *First-Order Logic*. Dover, 1995.
- [39] R. M. Smullyan. *Logical Labiriths*. CRC Press, 2009.
- [40] R. M. Smullyan and M. Fitting. *Set Theory and the Continuum Problem*. Dover, 2010.
- [41] R. I. Soare. *Turing Computability*. Springer, 2016.
- [42] The Stanford Encyclopedia of Philosophy. <http://plato.stanford.edu>.
- [43] D. van Dalen. *Logic and Structure*. Springer, fifth edition, 2013.