Survival Guide to First Order Logic (**FOL**)

Tristan Stérin Christopher-Lloyd Simon

July 2020

Contents

Foreword

First Order Logic (**FOL**) is one way to think about mathematical thinking: it is a formal approach to the concepts of truth and deduction. FOL gives a model of what mathematicians are doing when they are doing mathematics. The main strength of this perspective, with Gödel's theorems as pinacle, is that, by allowing to reason about reasoning, it identifies essential limitations of the deductive method – as modelled by FOL. Quite surprisingly, these limitations appear already when considering mathematical objects "as natural" as the natural numbers with addition and multiplication: $(\mathbb{N}, +, \times)$. However, the devil is in the details and, in the case of FOL it is very easy to get lost in these details. Hence, the goal of this document is to present FOL halfway between being completely formal and having an informal discussion with the hope that the reader will find some sort of conceptual clarity in this presentation. We believe that making parallels with Computability Theory is very fruitful when considering FOL hence this document develops and exploits the computational approach without restrain. All along the document, we will be explicit each time we make an **Assumption** at the meta level (i.e. the level from which we write not the level of what we are writing about), which gives a good indication of what is assumed when practicing FOL. We follow roughly the same presentation as in [1] which is an excellent, very accessible, and very technically precise presentation of FOL that can be freely downloaded online[1].

Finally, it is important to note that, although having been historically dominant[2], FOL is far from being the only way to think about mathematical thinking, a lot of other *logics* exist: intuitionistic logic, second order logic, higher order logics, modal logics, temporal logics, geometric logics et caetera. Hence, there is nothing universal in FOL: all what we are going to say is valid if you decide to accept the meta-level assumptions and logical axioms[3] of FOL.

[1]`https://minerva.geneseo.edu/a-friendly-introduction-to-mathematical-logic/`

[2]`https://philosophy.stackexchange.com/questions/2617/how-did-first-order-logic-come-to-be-the-dominant-formal-logic`

[3]For instance, a logical axiom of FOL is that "P or not P" always hold for any statement P (excluded middle). That axiom is not accepted by intuitionistic logicians, see `https://en.wikipedia.org/wiki/Intuitionistic_logic`.

1 Syntax

Assumption 1. We have a naïve concept of *collection* of objects (also called *class*). We also have a concept of *equality* between collections: two collections are equal iff they contain the same elements.

1.1 Language

In this section we talk only about syntax: there is no meaning associated to the strings of symbols that we define, we only say what are valid strings of symbols (in practice, *trees* of symbols) in First Order Logic (**FOL**). For instance, when we talk about "function symbols" we assume nothing about the concept of function, that is just the name of a class of symbols (which will be justified later when we talk about semantic).

Definition 2 (Language)**.** A language $\mathcal{L}$ is given by the following symbols, each coming with a notion of arity, which for now is just a non-negative integer :

1. A collection of constant symbols, with arity 0
2. A collection of function symbols with their arity
3. A collection of relation symbols with their arity
4. The equality relation symbol denoted $=$ with arity 2

Important. All along this document we will try not to use logical symbols (such as $=$) at the meta-level. For instance we will use $:\equiv$ instead of $=$ when we want to define or state that two objects are the same at the meta-level (the level from which we write, not the level "inside" a given FOL language). We will also use plain english sentences such as "this equals that".

Example 3. For instance, the language of arithmetic is $\mathcal{L}_{\mathbb{N}} :\equiv (0, S^1, +^2, \times^2, <^2)$ where 0 is a constant symbol, $S, +, x$ are function symbols and $<$ a relation symbol. Arities are given in exponents.

Remark 4. As functions are a special kind of relations, one could expose the whole theory with only relation symbols. However, the exposition is clearer when having function symbols so we keep them. Point 4 of Definition 2 is forcing all languages to have the equality symbol. At the moment it is not justified since this symbol "means" nothing, but it will become crucial when talking about semantic because we are studying First Order Logic With Equality. Equality is a difficult concept[4]. Finally, we assume nothing about the cardinality of the different collections that are involved in the definition of a language, in particular, they do not have to be finite.

1.2 Terms

Assumption 5. We have access to a countably infinite collection of *variable symbols* $\mathcal{V} :\equiv \{x, y, \ldots\}$, calld variables for short.

Given a language $\mathcal{L}$, we now construct *syntactic terms*, which are formal trees (again representing nothing but themselves yet). Each node is labeled by a symbol whose arity is equal to the number of edges descending from the node.

[4]see `https://math.stackexchange.com/questions/363168/first-order-logic-without-equality` and `http://rg1-teaching.mpi-inf.mpg.de/autrea-ss10/script/lecture17.pdf`

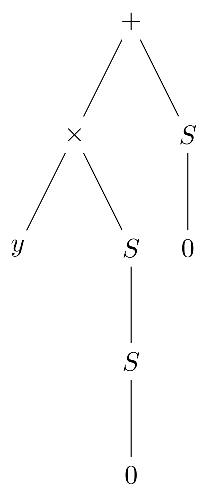

Figure 1: Example of a term in $\mathcal{L}_{\mathbb{N}} :\equiv (0, S^1, +^2, \times^2, <^2)$ (Definition 6).

Definition 6 (Term)**.** Given a language $\mathcal{L}$, a $\mathcal{L}$-term is a finite tree where:

- Leaves are either constant symbols or *variables symbols* (Assumption 5)
- Internal nodes are function symbols. Their arity gives the branching factor

The definition is recursive: children of internal nodes are either internal nodes or leafs. We write $\mathcal{T}_{\mathcal{L}}$ the collection of all $\mathcal{L}$-terms.

Example 7. Figure 1 depicts a term in the language of arithmetic $\mathcal{L}_{\mathbb{N}} :\equiv (0, S^1, +^2, \times^2, <^2)$. That term is of complexity 4. Although terms are intrinsically trees, they can be represented with traditional *infix* notation, in the case of Figure 1: $(y \times SS0) + S0$. Or in *prefix* notation (never ambiguous): $+ \times y\ SS0\ S0$. Again, all of this is purely syntactic, so for instance the term $+\ 0\ S0$ is different from $+\ S0\ 0$.

Remark 8. Terms represent the mathematical (formal) objects that one can conceive in a given language.

1.3 Formulae

Once we have mathematical objects, we can start talking about them: we can construct statements, called *formulae.* Note once again, that we do not yet give a meaning to formulae: they are only syntactic trees.

Definition 9 (Formula)**.** Given a language $\mathcal{L}$, a $\mathcal{L}$-formula is a finite tree where:

- Leaves are relations of terms, i.e. relation symbols applied to as many terms as their arity. Relations of terms are called *atomic formulae*
- Internal nodes are *logical connectors*, they come in two categories:

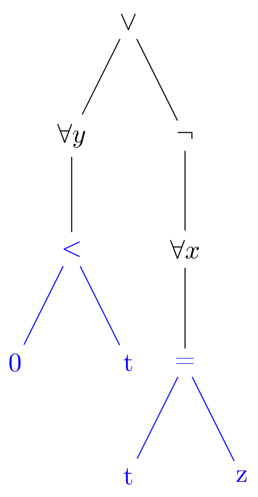

Figure 2: A formula in $\mathcal{L}_{\mathbb{N}}$ with t the term of Figure 1. Leaves (relations of terms) are in blue.

1. Propositional logic connectors: $\vee^2, \wedge^2, \neg^1, \rightarrow^2$ respectively named "inclusive or", "and","not" and "implies"
2. Universal and existential quantifiers: $\forall x$ (for all) and $\exists x$ (there exists) both of arity 1 and where x can be replaced by any variable in $\mathcal{V}$

We write $\mathcal{F}_{\mathcal{L}}$ the collection of all $\mathcal{L}$-formulae.

Example 10. Figure 2 gives an example of formula where t is the term of Figure 1. This formula is of complexity 3. Note that the variable x does not appear below the quantifier $\forall x$ which, although unusual when doing mathematics, is completely fine here.

Remark 11. A more minimalistic approach could construct logic formulae by using only two connectors: the Sheffer stroke[5] and the universal quantifier $\forall x$. That is because the Sheffer stroke can implement all propositional logic connectors and because, $(\exists x)(\phi)$ will be semantically equivalent to $\neg(\forall x)(\neg\phi)$.

Definition 12 (Boundedness)**.** In a formula, an occurrence of a variable symbol is *free* if no ancestor of that occurrence in the formula's tree is a quantifier using that variable symbol. It is *bound* otherwise. If *any* occurrence of a variable symbol is free then we say that the variable symbol is free. A formula with no free variable symbol (i.e. all occurrences of used variable symbols are bound) is called a *sentence*.

Example 13. In the formula of Figure 9, with t the term defined in Figure 6:

- There is one bound occurrence of the variable symbol y (in the leftmost t).
- There is one free occurrence of the variable symbol y (in the rightmost t).
- The only occurrence of the variable symbol z is free hence z is free.

[5]`https://en.wikipedia.org/wiki/Sheffer_stroke`

2 Semantic

Assumption 14 (Relation)**.** We have a naïve concept of *relation*: a k-ary relation over a collection $\mathcal{C}$ is a collection of k-tuples of elements of $\mathcal{C}$.

Assumption 15 (Function)**.** We have a naïve concept of *total function* with domain $\mathcal{C}_0$ and co-domain $\mathcal{C}_1$: it maps any element of the collection $\mathcal{C}_0$ to an element of the collection $\mathcal{C}_1$.

2.1 Models

So far, we have considered terms and formulae purely as formal objects. Now, we are going to give them a meaning, to *interpret* them. A term is interpreted within a *model*:

Definition 16 (Model)**.** A model $\mathfrak{A}$ for a language $\mathcal{L}$ (also called $\mathcal{L}$-structure) is given by:

1. A non-empty collection of objects A, called the *universe* of $\mathfrak{A}$
2. An interpretation for each constant symbol: i.e. for each constant symbol c, an element $c^{\mathfrak{A}}$ of A
3. An interpretation for each function symbol: i.e. for each k-ary function symbol f, a function (Assumption 15) $f^{\mathfrak{A}}$ from A^k to A
4. An interpretation for each relation symbol: i.e. for each k-ary relation symbol R, a k-ary relation (Assumption 14) R over A

We say that $\mathfrak{A}$ is a $\mathcal{L}$-model.

Example 17 (The standard model of arithmetic)**.** The standard model of arithmetic, $\mathfrak{N}$, is expressed in $\mathcal{L}_{\mathbb{N}} :\equiv (0, S^1, +^2, \times^2, <^2)$, with universe $\mathbb{N}$ the collection $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots\}$ and where constant symbol 0 is interpreted by $\mathbf{0}$, function symbol S is interpreted as being the natural successor function ($\mathbf{0} \mapsto \mathbf{1},\ \mathbf{1} \mapsto \mathbf{2}, \ldots$), function symbols $+, \times$ are interpreted to be $\boldsymbol{+}, \boldsymbol{\times}$ standard addition and multiplication operations and $<$ is interpreted as $\boldsymbol{<}$ the standard order relation.

Remark 18 (Reality)**.** Note the shocking aspect of Example 17 and the repeated use of the word *standard.* We are not constructing the standard model of arithmetic, we point our finger at it: we claim its existence and treat it as part of our reality. Hence $\mathfrak{N}$ becomes a *real* object, accessible out there (as opposed to specific to one's mind) by whoever is claiming its existence too.

Furthermore, in FOL, any tentative to *construct* $\mathfrak{N}$ is doomed to fail as we will see that no finite collection of axioms can characterize an infinite model up to isomorphism (i.e symbol renaming), see Theorem **??**.

2.2 Evaluating terms

Definition 19 (Variable assignment)**.** Given a model $\mathfrak{A}$ with universe A a variable assignment v is a total function from $\mathcal{V}$ to A: it gives an interpretation of variable symbols.

Given a model and a variable assignment, it becomes easy to inductively evaluate terms:

Definition 20 (Term evaluation)**.** Let $\mathfrak{A}$ be a $\mathcal{L}$-model with universe A and v a variable assignment. Let $\mathbf{eval}_{\mathfrak{A}}[v]$ (or just $\mathbf{eval}[v]$ when the model is given by context), the function from $\mathcal{T}_{\mathcal{L}}$ (the set of all $\mathcal{L}$-terms) to A inductively defined as follows:

1. If the term t is a constant symbol then $\mathbf{eval}[v](t) :\equiv t^{\mathfrak{A}}$
2. If the term t is a variable symbol then $\mathbf{eval}[v](t) :\equiv v(t)$
3. If the term t if of the form $f(t_1, \dots, t_k)$ then $\mathbf{eval}[v](t) :\equiv f^{\mathfrak{A}}(\mathbf{eval}[v](t_1), \dots, \mathbf{eval}[v](t_k))$

Example 21. In the standard model of arithmetic, $\mathfrak{N}$, with v such that $v(y) :\equiv \mathbf{4}$, we have: $\mathbf{eval}[v]((y \times SS0) + S0) :\equiv (\mathbf{4} \times \mathbf{2}) + \mathbf{1} :\equiv \mathbf{9}$.

Remark 22. Note that $\mathbf{eval}[v]$ is putting an equivalence relation on terms: semantic "merges" syntactically different trees into the same object which is described by the value returned by $\mathbf{eval}[v]$. Computation starts playing a role here: evaluating a term is a computation.

2.3 Evaluating formulae: defining truth

Definition 23 (Boolean algebra)**.** Let $\mathbb{B} :\equiv \{\mathbf{True}, \mathbf{False}\}$. We define **not** from $\mathbb{B}$ to $\mathbb{B}$ the standard negation: $\mathbf{not}(\mathbf{True}) :\equiv \mathbf{False}$ and $\mathbf{not}(\mathbf{False}) :\equiv \mathbf{True}$. Similarly we define $\mathbf{or}, \mathbf{and}, \mathbf{imp}$ from $\mathbb{B}^2$ to $\mathbb{B}$ the standard "inclusive or", "and", "implication". "Inclusive or" means that the value **False** is reached only for $\mathbf{or}(\mathbf{False}, \mathbf{False}) :\equiv \mathbf{False}$.

Remark 24. The truth table of standard implication is not immediate: $\mathbf{imp}(\mathbf{False}, \mathbf{False}) :\equiv \mathbf{imp}(\mathbf{False}, \mathbf{True}) :\equiv \mathbf{imp}(\mathbf{True}, \mathbf{True}) :\equiv \mathbf{True}$ and $\mathbf{imp}(\mathbf{True}, \mathbf{False}) :\equiv \mathbf{False}$. The intuition is as follows: anything can be deduced from false and only true can be deduced from true.

We can now define truth in a structure: we simply have to recursively evaluate our formulae's trees under the rules of Boolean algebra.

Definition 25 (Truth evaluation)**.** Let $\mathfrak{A}$ be a $\mathcal{L}$-model with universe A and v a variable assignment. Let $\mathbf{truth}_{\mathfrak{A}}[v]$ (or just $\mathbf{truth}[v]$ when the model is given by context), the function from $\mathcal{F}_{\mathcal{L}}$ (the set of all $\mathcal{L}$-formulae) to $\mathbb{B}$ inductively defined as follows:

1. If the formula ϕ is of the form $t_1 = t_2$ then $\mathbf{truth}[v](\phi) :\equiv \mathbf{True}$ iff $\mathbf{eval}[v](t_1)$ is equal to $\mathbf{eval}[v](t_2)$
2. If the formula ϕ is of the form $R(t_1, \dots, t_k)$ then $\mathbf{truth}[v](\phi) :\equiv \mathbf{True}$ iff the k-tuple $(\mathbf{eval}[v](t_1), \dots, \mathbf{eval}[v](t_k))$ is in the collection $R^{\mathfrak{A}}$
3. If the formula ϕ is of the form $\neg(\phi')$ then $\mathbf{truth}[v](\phi) :\equiv \mathbf{not}(\mathbf{truth}[v](\phi'))$. Similar definition for symbols $\vee$, $\wedge$ and $\rightarrow$ respectively associated to operators **or**, **and** and **imp** (Definition 23).
4. If the formulae ϕ is of the form $(\forall x)(\phi)$ then $\mathbf{truth}[v](\phi) :\equiv \mathbf{True}$ iff for all a in A, $\mathbf{truth}[v_{x\,:\,a}](\phi) :\equiv \mathbf{True}$ with $v_{x\,:\,a}$ the same assignment as v except for $v_{x\,:\,a}(x) :\equiv a$
5. If the formulae ϕ is of the form $(\exists x)(\phi)$ then $\mathbf{truth}[v](\phi) :\equiv \mathbf{True}$ iff there exists a in A such that $\mathbf{truth}[v_{x\,:\,a}](\phi) :\equiv \mathbf{True}$ with $v_{x\,:\,a}$ the same assignment as v except for $v_{x\,:\,a}(x) :\equiv a$

Please refer to Section 2.4 which defines the usual notation associated to truth: $\mathfrak{A} \vDash \phi[v]$, $\mathfrak{A} \vDash \phi$, $\mathfrak{A} \vDash \Sigma[v]$, $\mathfrak{A} \vDash \Sigma$ and $\Sigma \vDash \Gamma$.

Remark 26. Note also that for sentences (formulae without free variable, Definition 12), $\mathbf{truth}[v]$ is independent of v. In consequence, inside a model, truth of sentences is *absolute*: a sentence evaluates either to **True** or **False**. The famous concept of "undecidability" that we will explore in Section **??** *is not* an alternative to truth (see Remark **??**).

Remark 27. The definition of truth can be a deception at first sight. Indeed, at each Point of the definition, we are doing the same thing: we explain what truth is by taking it to the meta level (note for instance the absence of symbols at Point 1 when we say "is equal to"). While it maybe feels alright for Point 3, where we are simply applying Boolean algebra operators (i.e. doing a finite computation), Point 4 can feel like we are cheating. How can we know, in an infinite universe, if something holds "for all" the elements? The point is that when defining truth, we are not at the level of "knowing" (that will be the goal of proofs), We are at a level where we are confronted to a reality which is independent of any tentative to describe/axiomatize/prove. And it is the conclusion of Gödel incompleteness theorems (see Theorem **??**) that, given a "reasonable" system of axioms, one cannot prove all true statements in $\mathfrak{N}$.

Remark 28. In connection to the previous Remark, we can see an obvious difference between evaluating terms and evaluating formulae. While they are both finite objects, one can evaluate a term with a computer but cannot evaluate a formulae with a computer: for infinite models we will get stuck at each unbounded $(\forall x)$ e.g. for instance, in $\mathfrak{N}$, any $(\forall x)$ not followed by a bounding condition on x (of the form $x < t$).

A computer can try to evaluate formulae. Indeed, because $(\forall x)(\phi)$ is semantically equivalent to $\neg(\exists x)(\neg\phi)$, a computer can try to find a counter-example to ϕ by looping through all x. But nothing says if the computer will stop (in fact if $(\forall x)(\phi)$ is true it will never stop). We start seeing here a link between the Halting Problem of ideal computers (e.g. Turing Machines) and FOL.

Finally, *valid* $\mathcal{L}$-formulae are formulae which are true in any $\mathcal{L}$-model for any variable assignment:

Definition 29 (Valid formula)**.** A $\mathcal{L}$-formula ϕ is *valid* iff for any $\mathcal{L}$-model $\mathfrak{A}$ and variable assignement v we have: $\mathbf{truth}_{\mathfrak{A}}[v](\phi)$ equals **True**. We write $\vDash \phi$ (see Section 2.4).

Example 30. By Definition 25, given any model and variable assignment the formula $x = x$ evaluates to **True**. Hence the formula $x = x$ is valid, i.e. we have $\vDash x = x$. Hence we also have $\vDash (\forall x)(x = x)$ which is a canonical example of valid sentence (formula with no free variable).

Definition 31 (Canonical tautology and contradiction)**.** The canonical tautology is the formula $\top :\equiv (\forall x)(x = x)$ which can be expressed in any FOL language and is always valid (see Example 30). The canonical contradiction is $\bot :\equiv \neg((\forall x)(x = x))$ which can be expressed in any FOL language and evaluates to **False** in any model for any variable assignment.

2.4 Summary: notation for truth

In logic, very compact notation are commonly used in order to have, later down the road, compact theorem statements. We summarize all the notation related to truth here so that the reader can come back to this Section when needed.

In the following, given a language $\mathcal{L}$ (Definition 2), ϕ is a $\mathcal{L}$-formula (Definition 9), Σ and Γ are collections of $\mathcal{L}$-formulae, v a variable assignment (Definition 19), $\mathfrak{A}$ is a $\mathcal{L}$-model (Definition 16). We define the following notation:

1. $\mathfrak{A} \vDash \phi[v]$ iff $\mathbf{truth}_{\mathfrak{A}}[v](\phi) :\equiv$ **True** (Definition 25). This statement (meta-statement) reads "model $\mathfrak{A}$ satisfies formula ϕ with assignment v"

2. $\mathfrak{A} \vDash \phi$ iff we have $\mathfrak{A} \vDash \phi[v]$ for all assignments v. It reads "model $\mathfrak{A}$ satisfies formula ϕ"

3. $\mathfrak{A} \vDash \Sigma[v]$ iff we have $\mathfrak{A} \vDash \phi[v]$ for all ϕ in Σ

4. $\mathfrak{A} \vDash \Sigma$ iff we have $\mathfrak{A} \vDash \Sigma[v]$ for all assignments v

5. $\Sigma \vDash \Gamma$ iff in any model $\mathfrak{B}$ such that $\mathfrak{B} \vDash \Sigma$ we have $\mathfrak{B} \vDash \Gamma$. It reads "$\Sigma$ **logically implies** Γ". If Σ contains only one formula ϕ we write $\phi \vDash \Gamma$, similarly if Γ contains only ψ we write $\Sigma \vDash \psi$

6. $\emptyset \vDash \phi$ or $\vDash \phi$ when ϕ is valid (i.e. true in every model for every assignment, Definition 29). Note that it is a particular case of the previous Point

2.5 Equivalence between models

There are principally two notions of equivalence between two $\mathcal{L}$-models $\mathfrak{A}$ and $\mathfrak{B}$: **isomorphism**, written $\mathfrak{A} \cong \mathfrak{B}$ and **elementary equivalence** written $\mathfrak{A} \equiv \mathfrak{B}$. Isomorphism implies elementary equivalence but the converse does not hold.

Isomorphism corresponds to a renaming operation which preserves constants, the structure of terms and relations:

Definition 32 (Isomorphism)**.** Two $\mathcal{L}$-models $\mathfrak{A}$ and $\mathfrak{B}$ with respective universes A and B are isomorphic, and we write $\mathfrak{A} \cong \mathfrak{B}$, iff there exists a bijection i from A to B, called isomorphism, such that:

1. For all constant symbols c, we have $i(c^{\mathfrak{A}}) :\equiv c^{\mathfrak{B}}$

2. For all k-ary function symbols f and $a_1, \ldots, a_k$ in A^k we have: $i(f^{\mathfrak{A}}(a_1, \ldots, a_k)) :\equiv f^{\mathfrak{B}}(i(a_1), \ldots, i(a_k))$

3. For all k-ary relation symbols R and $a_1, \ldots, a_k$ in A^k then $(a_1, \ldots, a_k)$ is in $R^{\mathfrak{A}}$ iff $(i(a_1), \ldots, i(a_k))$ is in $R^{\mathfrak{B}}$

Remark 33. Although isomorphism corresponds "only" in a renaming procedure, it is highly non-trivial, even in the finite case, to decide whether two models are isomorphic or not. For instance, the problem of deciding whether two finite graphs (which are finite models of the language of graphs) are isomorphic or not is not known to be solvable by a machine in polynomial time[6] (it is solvable by a machine in exponential time as it "suffices" to try all the permutations of the first graph until we eventually find, or not find, the second graph). We can relate this discussion to the fact that we already know, when practicing mathematics, that showing that two things "are the same" can require a lot of efforts.

Elementary equivalence is intimately related to truth: two $\mathcal{L}$-models are elementary equivalent if the collection of formulae that they satisfy are the same. If they have the same *theory*:

Definition 34 (Theory of a model)**.** Let $\mathfrak{A}$ be a $\mathcal{L}$-model. The theory of model $\mathfrak{A}$ is $Th(\mathfrak{A})$ the collection $\{\phi \mid \phi\ \mathcal{L}\text{-formula such that } \mathfrak{A} \vDash \phi\}$.

Definition 35 (Elementary equivalence)**.** Two $\mathcal{L}$-models $\mathfrak{A}$ and $\mathfrak{B}$ are elementary equivalent, and we write $\mathfrak{A} \equiv \mathfrak{B}$, iff $Th(\mathfrak{A})$ and $Th(\mathfrak{B})$ are the same.

[6]Mysteriously enough, it is one of the only problems, with integer factorisation, which is: (a) in the complexity class NP, (b) not known to be in P, (c) not known to be NP-complete (i.e. able to simulate all the problems of the class NP), see `https://en.wikipedia.org/wiki/Graph_isomorphism`.

Let's state the relation between those two notions:

Theorem 36. Let $\mathfrak{A}$ and $\mathfrak{B}$ be two $\mathcal{L}$-models. Then $A \cong B \Rightarrow A \equiv B$ but $A \equiv B \not\Rightarrow A \cong B$.

Sketch proof. The proof that $A \cong B \Rightarrow A \equiv B$ can be done by induction on formulae. Indeed, isomorphism preserves the structure and semantic of formulae so the proof is not hard.

However, at this point we don't really have any tools to show that $A \equiv B \not\Rightarrow A \cong B$. We can only informally mention that, in the language of arithmetic, $\mathcal{L}_{\mathbb{N}} :\equiv (0, S^1, +^2, \times^2, <^2)$, there exists a *non-standard* model of arithmetic $\mathfrak{N}^*$, elementary equivalent to $\mathfrak{N}$, but whose universe contains an element ω (called *non-standard integer*) such that the sentence $\phi_n :\equiv t_n < \omega$ evaluates to **True** for all n where $t_n :\equiv \underbrace{S \dots S}_{n \text{ times}} 0$. Non-isomorphism follows directly: there is no suitable assignment for ω in the universe of $\mathfrak{N}$. The essence of why we have $\mathfrak{N} \equiv \mathfrak{N}^*$ is because the language of arithmetic, $\mathcal{L}_{\mathbb{N}}$, is not rich enough to "talk about" ω. True statements are the same in $\mathfrak{N}$ and $\mathfrak{N}^*$ because they cannot involve ω (or any non-standard integer) in what they are stating, they are "blind" to its existence, see TODO. □

Models of arithmetic are models elementary equivalent to $\mathfrak{N}$ (given the proof of the previous Theorem, they are not necessarily *standard*, i.e. isomorphic to $\mathfrak{N}$):

Definition 37 (Models of arithmetic)**.** A model of arithmetic is a $\mathcal{L}_{\mathbb{N}}$-model elementary equivalent to $\mathfrak{N}$.

Remark 38. There seems to be a confusion where "model of arithmetic" sometimes refers to a model in which, for instance, Peano axioms (PA) are satisfied (see Section 53). But, by Gödel theorems (Section 7), not all models of PA are elementary equivalent to $\mathfrak{N}$: there exists some sentences true in $\mathfrak{N}$ (i.e. what we see as *arithmetical truths*) which are not true in some other model satisfying PA and vice versa. See Section 3 and Section 7.

3 Proofs

Defining deduction in FOL, athough it will match with what we intuitively regard as a proof, is not immediate and contains technical difficulties. That is maybe because the act of "deducing" something is intrinsically technical. We leave most of the technical aspects (namely the logical axioms and deduction rules of FOL) of FOL deduction to Appendix **??** which can be ignored at first. Note that, because we are more interested in reasoning about proofs than making proofs themselves, it is not completely crucial to be aware of all the details of the FOL deduction system and, in some sense, the reader who practices mathematics is, at least inconsciously, familiar with these details. Historically, there has been multiple different approaches to formalizing deduction in FOL, such as *natural deduction* or *sequent calculus*, but, we follow a different approach which is the one of [1][7].

Importantly, note that all the constructions of this Section are mainly **formal**: we are only defining a formal system which generates objects that we call proofs. In particular, while we prepare the ground for it, we are not connecting the concept of proof to the concept of truth yet. That is the goal of Section 4.

[7]The interested reader will find all the technicalities that we omit in [1].

3.1 Defining proofs

Intuitively, a proof is not a crazy object: it can be seen as a finite, ordered, succession of formulae, where some formulae are *axioms* (i.e we do not have to prove them), and where the other formulae can be "logically deduced" from the preceding ones. Here are the desirable properties that our deduction system, which is composed of logical axioms and *deduction rules*, will have:

1. A proof involves only a finite number of formulae and uses a finite number of deduction rules (or *inference rules*).
2. An algorithm can decide whether a formula is a logical axiom or not and if a given application of a deduction rule is correct or not.
3. Altogether, an algorithm can decide whether a proof is correct or not: it recognises axioms and is able to verify each deduction rule.

Finding a proof can be hard and can require imagination but checking the proof is "easy" (i.e. mechanical, algorithmic). Formally, proofs are defined as follows:

Definition 39 (FOL logical axioms)**.** Let Λ be the set of $\mathcal{L}$-formulae called FOL *logical axioms.* Appendix **??** gives all the logical axioms of FOL.

Example 40. For instance, for any variable symbol x, the formula $x = x$ is a logical axiom of FOL. Apart from dealing with equality, the other logical axioms of FOL describe the use of quantifiers and allow *universal instantiation* and *existential generalization*, see Appendix **??**.

Definition 41 (FOL deduction rules)**.** We have a collection of ordered pairs (Γ, ϕ), called FOL *deduction rules*, where Γ is a *finite* collection of $\mathcal{L}$-formulae and ϕ a $\mathcal{L}$-formula. Appendix **??** gives all the deduction rules of FOL.

Example 42. An important example of class of deduction rules is *propositional consequence* (rules of "type **PC**"). Propositional consequence embeds all the theorems of *propositional logic.* For example, *modus ponens*, for two sentences ϕ and ψ then $(\{\phi, \phi \to \psi\}, \psi)$ is a deduction rule of type PC. Verifying PC rules amounts to replace (in the right way, see Appendix **??**) formulae by propositional variables, here, $(\{A, A \to B\}, B)$ and verify that, under the rules of Boolean algebra (Definition 23), any truth assignment which satisfies both A and $A \to B$ also satisfies B (i.e. evaluates B to **True**). In other word, to verify that $(A \wedge (A \to B)) \to B$ is a *tautology* (true for any truth assignment). Verifying propositional tautologies can be done algorithmically by looping over all the 2^n truth assignments of the n propositional variables and evaluating truth at each step. Note however that there is no known efficient algorithm (polynomial time in n) for solving this problem in general[8]. The other class of FOL deduction rules deals with how to "safely" manipulate universal and existential quantifiers, see Appendix **??**.

Definition 43 (Proof)**.** Let Σ be a collection of $\mathcal{L}$-formulae (called *theoretical axiom* or *nonlogical axioms*) and D a finite ordered sequence of $\mathcal{L}$-formulae $(\phi_1, \phi_2, \dots, \phi_n)$. Then D is a proof of ϕ_n from Σ if for each $1 \leq i \leq n$:

1. ϕ_i is in Λ (is a logical axiom) or

[8]Related to the problem called SAT in complexity theory, which is complete for the class NP: in particular, any polynomial time running algorithm can be reformulated as corresponding to particular instances of SAT, see `https://en.wikipedia.org/wiki/Boolean_satisfiability_problem`.

2. ϕ_i is in Σ (is a nonlogical axiom) or
3. There is a deduction rule (Γ, ϕ) such that $\Gamma \subseteq \{\phi_1, \ldots, \phi_{i-1}\}$.

We write $\Sigma \vdash \phi$ to mean that there is a proof D from Σ of ϕ (ϕ is the last formula of the proof D). It reads "Σ proves ϕ".

Remark 44. Logical axioms are related to the internal mechanics of FOL deduction, together with deduction rules, they constitute the essence of how FOL is modelling mathematical deduction (see Appendix **??**). Nonlogical axioms are specified by mathematicians when they start axiomatizing their theories – see for instance the axioms of arithmetic, Definition 53. If we want all proofs to be verifiable by a computer we need to guarantee that nonlogical axioms, like logical axioms, can be recognised by a decision procedure.

Definition 45 (Thm_Σ)**.** Let Σ be a collection of $\mathcal{L}$-formulae. Then, the collection of formulae that can be deduced from Σ is written $Thm_\Sigma :\equiv \{\phi \mid \phi \text{ is a } \mathcal{L}\text{-formula and } \Sigma \vdash \phi\}$. The set Thm_Σ is said to be an *axiomatic theory*. Elements of Thm_Σ are called *theorems*.

Remark 46. Note that Thm_Σ is countable and, because verifying proofs is an algorithmic procedure, it is possible to algorithmically enumerate (also called *recursively enumerate*) all valid proofs starting from Σ and so, all the theorems in Thm_Σ. However, for any given ϕ, deciding whether ϕ is in Thm_Σ or not wont be possible in the general case (see Remark **??**). Said otherwise, while we can always algorithmically enumerate Thm_Σ, in general, we cannot know for sure if some given ϕ will eventually appear in the enumeration or not. Said otherwise, in general, we cannot algorithmically enumerate the **complement** of Thm_Σ (because if we always could, the previous point would be settled: ϕ would eventually appear in enumerating Thm_Σ or its complement providing us with a decision procedure).

3.2 Properties of the proof system

A good way to get more familiar with FOL deduction system is to be aware of its properties. We list some of them here. Note again that we are not yet connecting the concept of proof to the concept of truth. We are purely talking about the properties of a formal system. In the following, Σ is an **arbitrary** collection of $\mathcal{L}$-formulae and ϕ a formula.

Lemma 47 (Universal closure)**.** $\Sigma \vdash \phi$ iff $\Sigma \vdash (\forall x)(\phi)$

Remark 48. Some authors require theorems (and axioms) to be sentences (no free variables) instead of formulae. This restriction does not really matter in FOL because of the above lemmas which implies that the universal closure of any formula in Thm_Σ is also in Thm_Σ. By the same argument, we can replace each formula of Σ by their universal closure without really changing the collection of formulae that we can prove.

Theorem 49 (Deduction theorem)**.** Let θ be a sentence (no free variables). Then:

$$\Sigma \vdash \theta \to \phi \text{ iff } \Sigma \cup \{\theta\} \vdash \phi$$

Remark 50. The deduction theorem may sound a bit tautological at first sight. However its proof is non trivial (in the $\Leftarrow$ direction) and it validates something that we do often when doing mathematics. When we want to prove an implication, we can add the premise to our set of hypothesis and derive the conclusion. The subtle point is that, by doing so, we can reconstruct a valid proof (seen as a formal object) of the implication we were starting with. Finally, because we have not insisted on the details of FOL deduction (see Appendix **??**) we cannot really justify why θ is asked to be a sentence. At the intuitive level we can only say that it is because it prevents making any kind of assumptions on the variables present in θ.

Theorem 51 (Proof by contradiction)**.** Recall that $\bot :\equiv \neg((\forall x)(x = x))$. We have:

1. $\Sigma \cup \{\bot\} \vdash \phi$
2. $\Sigma \vdash \phi$ iff $\Sigma \cup \{\neg\phi\} \vdash \bot$

Remark 52 (Intuitionism)**.** The first Point says that you can prove anything from something false ($\bot$ is the canonical "false" as it is expressible in any FOL language and evaluates to **False** in all models). The second Point says that FOL deduction admits proof by contradiction. This point has been source of intellectual polemics in the 20th century as the school of intuitionism (see [3]) refuses the idea of proof by contradiction. The reason being that it allows non-constructive proofs: showing that an object exists without constructing it. Famously, for instance, the non-constructive proof of "there exists a, b irrational numbers such that a^b is rational", see [2]. In that particular case, constructive proofs are also known (take $a = \sqrt{2}, b = \log_2(9)$, we have $a^b = 3$) however there exists cases where a non-constructive proof exists but no constructive proof is known, see [2].

3.3 Axioms of arithmetic

The collection of non-logical axioms which, historically, has accompanied the development of FOL, are the axioms of arithmetic. Also called "Robinson axioms" and expressed in the language of arithmetic $\mathcal{L}_{\mathbb{N}} :\equiv (0, S^1, +^2, \times^2, <^2)$ they are the following:

Definition 53 (The axioms of arithmetic)**.** The axioms of arithmetic is the finite collection N composed of the following $\mathcal{L}_{\mathbb{N}}$-sentences:

1. $(\forall x)\, \neg(Sx = 0)$
2. $(\forall x)(\forall y)\, [Sx = Sy \to x = y]$
3. $(\forall x)\, [x + 0 = x]$
4. $(\forall x)(\forall y)\, [x + Sy = S(x + y)]$
5. $(\forall x)\, [x \times 0 = 0]$
6. $(\forall x)(\forall y)\, [x \times Sy = (x \times y) + x]$
7. $(\forall x)\, \neg(x < 0)$
8. $(\forall x)(\forall y)\, [x < Sy \leftrightarrow (x < y \vee x = y)]$
9. $(\forall x)(\forall y)\, [x < y \vee x = y \vee y < x]$

Remark 54. The axioms of arithmetic are quite weak. For instance, from N, it is not possible to prove the commutativity of $+$. Anticipating Section 7, we say that "the commutativity of $+$ is *independent* of N" or "the commutativity of $+$ is *undecidable* in N". Hence we don't have equality between $Thm_N :\equiv \{\phi \mid N \vdash \phi\}$ and $Th(\mathfrak{N}) :\equiv \{\phi \mid \mathfrak{N} \vDash \phi\}$ since commutativity of addition is verified in $\mathfrak{N}$ (the standard model of arithmetic, see Example 17). One could question the validity of these axioms if we can't even prove that $x + y = y + x$. However, it is the whole point of FOL and Gödel theorems, to show that for any *reasonable* axiomatization of arithmetic, such sentences exist: they are true in $\mathfrak{N}$, but cannot be derived from the axioms. Not all arithmetical truth are provable, said otherwise, there is no reasonable Σ such that Thm_Σ is equal to $Th(\mathfrak{N})$, see Section 7. Nevertheless, the axioms of N are powerful enough to show some natural arithmetical facts such as $N \vdash \bar{a} + \bar{b} = \overline{a + b}$. For any a and b natural numbers, $a + b$ the standard integer addition and $\bar{a}$ the $\mathcal{L}_\mathbb{N}$-term $\underbrace{S \dots S}_{a \text{ times}} 0$. See [1] (Lemma 2.8.4) for other example of deductions from N.

Peano's axioms add the induction scheme to the axioms of arithmetic:

Definition 55 (Peano's axiom)**.** The axioms of Peano, PA, are N, the axioms of arithmetic together with the induction scheme: $[\phi(0) \wedge (\forall x)\,(\phi(x) \rightarrow \phi(Sx))] \rightarrow (\forall x)\phi(x)$ for all $\mathcal{L}_\mathbb{N}$-formula with only one free variable x. Contrary to N, the collection PA is not finite.

Remark 56. So far we never introduced notation such as $\phi(0)$ which only means, assuming ϕ has only one free variable x, "replace all occurrences of the variable x by the term 0".

An other historically important collection of axioms are ZF the axioms of set theory, expressed in the language of set theory which consists only of one binary relation symbol: $\mathcal{L}_{\mathrm{ST}} :\equiv \{\in\}$. For instance, the existence of the empty set is the axiom $\exists x \forall y\ \neg(y \in x)$.

3.4 Summary: properties of axiomatic theories

Let Σ be a set of formulae and $Thm_\Sigma :\equiv \{\phi \mid \Sigma \vdash \phi\}$ be its associated axiomatic theory (Definition 45). The following are properties that Thm_Σ can have:

1. **Consistency**: there is no derivation of $\bot :\equiv \neg(\forall x)(x = x)$ from Σ. We write $\Sigma \nvdash \bot$. Note that next Section will prove that the consistency of a collection of axioms is equivalent to them having a model (Theorem 57) i.e. Σ is consistent iff there exists a model in which Σ is satisfied.

2. **Enumerability**: there is an algorithmic procedure to enumerate all deductions in Thm_Σ. Note that, given FOL proof system, if the set of nonlogical axioms Σ is decidable (i.e. a computer can recognise nonlogical axioms) then Thm_Σ is enumerable.

3. **Decidability**: there is an algorithmic procedure to determine whether a given ϕ is in Thm_Σ or not.

4. **Completeness**: any ϕ is either proved (ϕ is in Thm_Σ) or refuted ($\neg\phi$ is in Thm_Σ). We have that (Completeness + Enumerability) implies Decidability: in order to decide whether or not ϕ is in Thm_Σ or not, enumerate Thm_Σ until, by completeness, you find a proof of ϕ or $\neg\phi$. Decidability implies Enumerability but it does not imply Completeness: we still can have ϕ such that neither ϕ or $\neg\phi$ is in Thm_Σ.

4 Soundness, completeness and compactness of FOL

We have spent some efforts in order to define the notions of truth and proofs and it is time to start connecting them to one another. In this section, we are going to prove fundamental statements about FOL itself: FOL is *sound*, *complete*[9] and *compact*. Soundness of FOL is a property that you would hope to get in any logical system: if a formula ϕ is deduced from Σ, then Σ logically implies ϕ, i.e. it is true in **all** models satisfying Σ. Completeness is a property which is less immediate: it states that if a statement if true in all models satisfying a collection of axiom then there is a proof of it. Completeness is equivalent to the existence of a model for any consistent set of axioms (Lemma 58), it is worth mentioning that some logics, Second Order Logic for instance, are not complete (and not compact).

Theorem 57 (Fundamental theorem of FOL)**.** Let Σ be a collection of $\mathcal{L}$-formulae and ϕ a $\mathcal{L}$-formula. Then $\Sigma \vDash \phi$ iff $\Sigma \vdash \phi$.

Proof.

- Soundness: if $\Sigma \vdash \phi$ then $\Sigma \vDash \phi$. This is not the hard direction. Proceed by induction on the complexity of ϕ and use the fact that FOL logical axioms are valid (true in all models, for all assignments) and that FOL deduction rules preserve truth. See [1] for the entire proof.

- Completeness: (a) if $\Sigma \vDash \phi$ then $\Sigma \vdash \phi$. This requires more work. First notice that the statement is equivalent to: (b) if $\Sigma \vDash \bot$ then $\Sigma \vdash \bot$. This is because from (b) we have: (c) if $\Sigma \cup \{\neg\phi\} \vDash \bot$ then $\Sigma \cup \{\neg\phi\} \vdash \bot$, then by supposing $\Sigma \vDash \phi$ we get $\Sigma \cup \{\neg\phi\} \vDash \bot$ which from (c) gives $\Sigma \cup \{\neg\phi\} \vdash \bot$ which, because FOL admits proof by contradiction (Theorem 51), we get $\Sigma \vdash \phi$.

  Note that the statement: (b) if $\Sigma \vDash \bot$ then $\Sigma \vdash \bot$ reads "if no model satisfies Σ then Σ is inconsistent". Which by contraposition amounts to: (d) "if Σ is consistent then Σ has a model" (i.e. there is model in which Σ is satisfied). There "only" remains to construct such a model, this is the goal of Lemma 58.

□

[9] Completeness of First Order Logic is not related to the completeness of an axiomatic theory as defined in Section 3.4. The fact that those words are the same can create a confusion at first.

Lemma 58 (Henkin model)**.** Let Σ be a consistent set of $\mathcal{L}$-axioms, meaning $\Sigma \not\vdash \perp$ – i.e. there is no deduction of $\perp :\equiv \neg(\forall x)(x = x)$ from Σ. Then there exists a $\mathcal{L}$-model $\mathfrak{M}$ such that $\mathfrak{M} \vDash \Sigma$. The model constructed in this Lemma is called *Henkin* model of Σ or *syntactic* model of Σ.

5 Computability

5.1 Algorithms

Gödel incompleteness theorems are intimately linked to the existence of functions "that we cannot compute". In order to highlight this link we need to talk about the basis of *computability theory*. In this document, we already have used the concept of *algorithm* (for instance in Section 3 when defining proofs) quite loosely, with no formal attempt do define what they are. We are going to go further in that direction by assuming out loudly that you *intrinsically* know what they are:

Assumption 59 (Algorithm)**.** We have a naïve concept of *algorithm*.

Remark 60 (Church-Turing thesis)**.** The fact that, with Assumption 59 we allow ourselves to talk about algorithms without a formal, mathematical, definition is motivated by the *Church-Turing thesis*. Indeed, historically, a lot of different models were invented to talk about algorithms: Turing machines, Church's lambda-calculus, Gödel's recursive functions, Minsky's counter machines, Post's tag systems, Generalised Collatz Maps, elementary 1D cellular automaton "rule 110", Conway's "game of life", and more. However, although these models can be very different looking, they are all, as far as algorithms are concerned, *equivalent*: there is no computation that one of these models can do that another cannot do[10] and that is because they can all simulate one another. This leads to Church-Turing thesis which is of a philosophical nature: the concept of *algorithm* is more essential than the actual model you use to specify them[11]. More informally: algorithms are independent of the programming language you use. We could make another analogy with numbers: the concept of number is independent of the base you use to represent them.

If we do not place ourselves in a specific model of computation, let's give some properties that algorithms have:

1. **(Program).** An algorithm, in a given model of computation, admits a finite description called its *program*. Importantly, the set of programs of a given length is finite too.
2. **(Input).** An algorithm admits a finite (possibly empty) *input*.
3. **(Halt).** An algorithm, on a given input, either *halts* or does not halt.
4. **(Output).** An algorithm, on a given input, if it halts, delivers a (possible empty) *output*.

Definition 61 (Computable function)**.** A total function $f : \mathbb{N} \to \mathbb{N}$ is said to be *computable* if there is an algorithm such that for any input $x \in \mathbb{N}$, the algorithm halts and produces the output $f(x) \in \mathbb{N}$. The set of computable functions is denoted by $\mathcal{C} \subset \mathbb{N}^{\mathbb{N}}$.

[10]Between models, there can be differences in time/space efficiency but that is the topic of *Complexity theory*.

[11]The model of computation needs to be powerful enough (for instance Finite State Automata cannot simulate arbitrary Turing Machines) and if it is, is then called a Turing-complete model (Finite State Automata *are not* Turing-complete).

Remark 62. By Church-Turing thesis (Remark 60) the set of computable functions is independent of the model of computation which is used. The concept of algorithm is more essential than the concept of computable function since computable functions do not account for algorithm which do not halt on all inputs.

Example 63. Very early in life, we learn that $+ : \mathbb{N}^2 \to \mathbb{N}$ is a computable function[12]: we are taught the algorithm which performs the addition of two numbers (written in base 10).

5.2 Non-computable functions

Because the set of programs of a given length is finite, the set of all programs is countable. Hence, the set of computable functions $f : \mathbb{N} \to \mathbb{N}$ is countable too. Thus, because $\mathbb{N} \to \mathbb{N}$ is not countable, there must exist some non-computable function, i.e. a function for which no algorithm can give the value of $f(x)$ for all x. Can we give an example of such an uncomputable function? Turing gave a famous, canonical, answer: the *halting problem.* The halting problem, which is the function deciding if an arbitrary program halts on an arbitrary input, *is not* computable:

Theorem 64 (The halting problem)**.** Let $\mathcal{H} = \{E(p, x) \mid \text{such that program } p \in \mathcal{P} \text{ halts on input } x \in \mathbb{N}\}$ with $E : \mathcal{P} \times \mathbb{N} \to \mathbb{N}$ some computable encoding for program/input pairs in the Turing-complete model of computation in use. Then the *halting problem*, $h : \mathbb{N} \to \{0, 1\}$, which is the characteristic function of $\mathcal{H}$, is not computable.

6 Expressivity of FOL

7 Gödel incompleteness theorems

Theorem 65 (First incompleteness theorem)**.** Suppose that A is a consistent and decidable set of nonlogical axioms expressed in $\mathcal{L}_{\mathbb{N}}$. Then there is a $\mathcal{L}_{\mathbb{N}}$-sentence θ such that $\mathfrak{N} \vDash \theta$ but $A \nvdash \theta$.

8 Going further

References

[1] Christopher Leary and Lars Kristiansen. *A friendly introduction to mathematical logic.* Milne Library, SUNY Geneseo, Geneseo, NY, 2015. URL: `https://minerva.geneseo.edu/a-friendly-introduction-to-mathematical-logic/`.

[2] Wikipedia. Constructive proof — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Constructive%20proof&oldid=969930319`, 2020. [Online; accessed 31-July-2020].

[3] Wikipedia. Intuitionism — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Intuitionism&oldid=955009666`, 2020. [Online; accessed 31-July-2020].

[12]Note that $\mathbb{N}^2$ is bijection with $\mathbb{N}$