Logic for Computer Science and Artificial Intelligence
Logic for Computer Science and Artificial Intelligence

Ricardo Caferra
# Table of Contents

**Preface** ........................................ xi

**Chapter 1. Introduction** ............................. 1
   1.1. Logic, foundations of computer science, and applications of logic to computer science .......................... 1
   1.2. On the utility of logic for computer engineers .......................... 3

**Chapter 2. A Few Thoughts Before the Formalization** ........... 7
   2.1. What is logic? ....................................... 7
       2.1.1. Logic and paradoxes ................................. 8
       2.1.2. Paradoxes and set theory .............................. 9
       2.1.2.1. The answer .................................. 10
       2.1.3. Paradoxes in arithmetic and set theory .................. 13
       2.1.3.1. The halting problem ............................ 13
       2.1.4. On formalisms and well-known notions ................... 15
       2.1.4.1. Some “well-known” notions that could turn out to be difficult to analyze ....................... 19
       2.1.5. Back to the definition of logic ....................... 23
       2.1.5.1. Some definitions of logic for all .................... 24
       2.1.5.2. A few more technical definitions ................. 24
       2.1.5.3. Theory and meta-theory (language and meta-language) ............................................. 30
       2.1.6. A few thoughts about logic and computer science ......... 30
   2.2. Some historic landmarks ............................. 32

**Chapter 3. Propositional Logic** .............................. 39
   3.1. Syntax and semantics ................................. 40
       3.1.1. Language and meta-language ......................... 43
# Table of Contents

5.8. A decidable class: the monadic class .................................. 202  
5.8.1. Some decidable classes ........................................... 205  
5.9. Limits: Gödel’s (first) incompleteness theorem ..................... 206

Chapter 6. Foundations of Logic Programming .......................... 213  
6.1. Specifications and programming .................................... 213  
6.2. Toward a logic programming language ................................ 219  
6.3. Logic programming: examples ...................................... 222  
6.3.1. Acting on the execution control: cut “/” ....................... 229  
6.3.1.1. Translation of imperative structures .......................... 231  
6.3.2. Negation as failure (NAF) ...................................... 232  
6.3.2.1. Some remarks about the strategy used by LP and  
  negation as failure .................................................. 238  
6.3.2.2. Can we simply deduce instead of using NAF? .............. 239  
6.4. Computability and Horn clauses .................................... 241

Chapter 7. Artificial Intelligence .......................................... 245  
7.1. Intelligent systems: AI ............................................. 245  
7.2. What approaches to study AI? ...................................... 249  
7.3. Toward an operational definition of intelligence .................... 249  
7.3.1. The imitation game proposed by Turing ....................... 250  
7.4. Can we identify human intelligence with mechanical  
  intelligence? .................................................................. 251  
7.4.1. Chinese room argument .......................................... 252  
7.5. Some history .......................................................... 254  
7.5.1. Prehistory .......................................................... 254  
7.5.2. History ............................................................. 255  
7.6. Some undisputed themes in AI ....................................... 256

Chapter 8. Inference ............................................................ 259  
8.1. Deductive inference ................................................... 260  
8.2. An important concept: clause subsumption ......................... 266  
8.2.1. An important problem ............................................ 268  
8.3. Abduction ............................................................. 273  
8.3.1. Discovery of explanatory theories ............................... 274  
  8.3.1.1. Required conditions ......................................... 275  
8.4. Inductive inference .................................................... 278  
8.4.1. Deductive inference .............................................. 279  
8.4.2. Inductive inference .............................................. 280  
8.4.3. Hempel’s paradox (1945) ....................................... 280  
8.5. Generalization: the generation of inductive hypotheses .......... 284  
  8.5.1. Generalization from examples and counter examples ....... 288
### Chapter 9. Problem Specification in Logical Languages

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>Equality</td>
<td>291</td>
</tr>
<tr>
<td>9.1.1</td>
<td>When is it used?</td>
<td>292</td>
</tr>
<tr>
<td>9.1.2</td>
<td>Some questions about equality</td>
<td>292</td>
</tr>
<tr>
<td>9.1.3</td>
<td>Why is equality needed?</td>
<td>293</td>
</tr>
<tr>
<td>9.1.4</td>
<td>What is equality?</td>
<td>293</td>
</tr>
<tr>
<td>9.1.5</td>
<td>How to reason with equality?</td>
<td>295</td>
</tr>
<tr>
<td>9.1.6</td>
<td>Specification without equality</td>
<td>296</td>
</tr>
<tr>
<td>9.1.7</td>
<td>Axiomatization of equality</td>
<td>297</td>
</tr>
<tr>
<td>9.1.8</td>
<td>Adding the definition of $=$ and using the resolution method</td>
<td>297</td>
</tr>
<tr>
<td>9.1.9</td>
<td>By adding specialized rules to the method of semantic tableaux</td>
<td>299</td>
</tr>
<tr>
<td>9.1.10</td>
<td>By adding specialized rules to resolution</td>
<td>300</td>
</tr>
<tr>
<td>9.1.10.1</td>
<td>Paramodulation and demodulation</td>
<td>300</td>
</tr>
<tr>
<td>9.2</td>
<td>Constraints</td>
<td>309</td>
</tr>
<tr>
<td>9.3</td>
<td>Second Order Logic (SOL): a few notions</td>
<td>319</td>
</tr>
<tr>
<td>9.3.1</td>
<td>Syntax and semantics</td>
<td>324</td>
</tr>
<tr>
<td>9.3.1.1</td>
<td>Vocabulary</td>
<td>324</td>
</tr>
<tr>
<td>9.3.1.2</td>
<td>Syntax</td>
<td>325</td>
</tr>
<tr>
<td>9.3.1.3</td>
<td>Semantics</td>
<td>325</td>
</tr>
</tbody>
</table>

### Chapter 10. Non-classical Logics

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1</td>
<td>Many-valued logics</td>
<td>327</td>
</tr>
<tr>
<td>10.1.1</td>
<td>How to reason with $p$-valued logics?</td>
<td>334</td>
</tr>
<tr>
<td>10.1.1.1</td>
<td>Semantic tableaux for $p$-valued logics</td>
<td>334</td>
</tr>
<tr>
<td>10.2</td>
<td>Inaccurate concepts: fuzzy logic</td>
<td>337</td>
</tr>
<tr>
<td>10.2.1</td>
<td>Inference in FL</td>
<td>348</td>
</tr>
<tr>
<td>10.2.1.1</td>
<td>Syntax</td>
<td>349</td>
</tr>
<tr>
<td>10.2.1.2</td>
<td>Semantics</td>
<td>349</td>
</tr>
<tr>
<td>10.2.2</td>
<td>Herbrand’s method in FL</td>
<td>350</td>
</tr>
<tr>
<td>10.2.2.1</td>
<td>Resolution and FL</td>
<td>351</td>
</tr>
<tr>
<td>10.3</td>
<td>Modal logics</td>
<td>353</td>
</tr>
<tr>
<td>10.3.1</td>
<td>Toward a semantics</td>
<td>355</td>
</tr>
<tr>
<td>10.3.1.1</td>
<td>Syntax (language of modal logic)</td>
<td>357</td>
</tr>
<tr>
<td>10.3.1.2</td>
<td>Semantics</td>
<td>358</td>
</tr>
<tr>
<td>10.3.2</td>
<td>How to reason with modal logics?</td>
<td>360</td>
</tr>
<tr>
<td>10.3.2.1</td>
<td>Formal systems approach</td>
<td>360</td>
</tr>
<tr>
<td>10.3.2.2</td>
<td>Translation approach</td>
<td>361</td>
</tr>
<tr>
<td>10.4</td>
<td>Some elements of temporal logic</td>
<td>371</td>
</tr>
<tr>
<td>10.4.1</td>
<td>Temporal operators and semantics</td>
<td>374</td>
</tr>
<tr>
<td>10.4.1.1</td>
<td>A famous argument</td>
<td>375</td>
</tr>
</tbody>
</table>
# Table of Contents

10.4.2. A temporal logic ........................................ 377  
10.4.3. How to reason with temporal logics? ................... 378  
10.4.3.1. The method of semantic tableaux .................... 379  
10.4.4. An example of a PL for linear and discrete time: PTL  
(or PLTL) ................................................................ 381  
10.4.4.1. Syntax ...................................................... 381  
10.4.4.2. Semantics .................................................. 382  
10.4.4.3. Method of semantic tableaux for PLTL           
(direct method) ..................................................... 383  

Chapter 11. Knowledge and Logic: Some Notions .................. 385  
11.1. What is knowledge? ............................................ 386  
11.2. Knowledge and modal logic .................................. 389  
11.2.1. Toward a formalization .................................... 389  
11.2.2. Syntax ......................................................... 389  
11.2.2.1. What expressive power? An example ................. 389  
11.2.2.2. Semantics .................................................. 389  
11.2.3. New modal operators ....................................... 391  
11.2.3.1. Syntax (extension) ....................................... 391  
11.2.3.2. Semantics (extension) ................................. 391  
11.2.4. Application examples ..................................... 392  
11.2.4.1. Modeling the muddy children puzzle ............... 392  
11.2.4.2. Corresponding Kripke worlds ......................... 392  
11.2.4.3. Properties of the (formalization chosen for the)   
knowledge ............................................................. 394  

Chapter 12. Solutions to the Exercises ............................. 395  

Bibliography .......................................................... 515  

Index ................................................................. 517
These notes result from a certain conception of knowledge transmission and from the experience gained after several years of teaching at the Grenoble Institute of Technology (Ensimag).

If the table of contents is interpreted too literally, the task at hand is infeasible: each of the themes developed in the different chapters has been the topic of thousands of pages (books, monographs, articles, popularization books, etc.) published by numerous authors, and some of these pages are of the highest scientific quality. On top of this, we must consider all the information available on the Internet.

The aim of these notes, which is probably ambitious but hopefully not disproportionate, is to attempt to provide a unified overview of the concepts and techniques that are useful in many well-identified domains of modern computer science and artificial intelligence. It is difficult to find all these topics in the same document, and they should also be a good starting point for a reader wishing to explore further topics.

Conceptual rigor will always be preferred to formal rigor. This approach is essential for the transmission of knowledge in the modern world, especially for those domains in which the readers will have to keep acquiring additional knowledge throughout their professional life.

The presentation method of all the topics will always be the same: informal description of the topic under consideration (motivation) ~ historical background ~ examples ~ possible conceptualizations ~ comparative analysis ~ formalization ~ technical aspects.

Of course, the algorithmic point of view is privileged and for almost every considered problem, the goal is to design an algorithm capable of solving it. Examples play a crucial role in these notes: they have been chosen so as to guide in the...
conceptualization of pertinent abstractions for classes of problems. Digressions and remarks allow for an in-depth view of some of the topics and for the discovery of their relationship with other topics and other domains. Exercises are an essential complement to the topics treated, which cannot be understood and assimilated without solving them (solutions to the exercises are included in the final chapter). It is clear that the material treated here has already been discussed in other books. However, some of these topics are approached in an original manner in this book.

In addition to carrying out their original goal within a university syllabus, these pages will hopefully be agreeable to the reader and will also be an incentive to those wanting to know more and ask further questions.
Chapter 1

Introduction

We briefly analyze the relationship between logic and computer science, by focusing successively on two points of views, in a somewhat natural order. We first underline the importance of logic in the foundations of computer science and how it is used in many computer science domains. Then we explain why logic is useful for computer engineers.

1.1. Logic, foundations of computer science, and applications of logic to computer science

Trying to underline the importance of logic for computer science in the 21st Century is the same as trying to reinvent the wheel. Indeed, in 1969, C.A.R. Hoare wrote:

Computer programming is an exact science, in that all the properties of a program and all the consequences of executing it can, in principle, be found out from the text of the program itself by means of purely deductive reasoning.

More recently, in 1986, Hoare stated that “computers are mathematical machines and computer programs are mathematical expressions”.

Of course, in our defence of logic, we will not make use of arguments relying on Hoare’s renowned expertise, as these arguments may turn out to be fallacious; however, we attempt to shed some light on this kind of sentence and to convince the reader that the importance of logic for computer science is a fact and not a matter of opinion.
Logical concepts have been of prime importance in computer science, and this is still the case nowadays.

Instead of making a potentially tedious enumeration, we shall simply mention two typical concepts of computer science, production rules and formal languages, the origins of which are seldom mentioned and which were, respectively, invented by logicians in 1921 (Post) and 1879 (Frege).

We cannot overstate the importance of studying the foundations and the history of a discipline, as proved by the following quote. In 1936, A. Turing introduced his notion of an abstract computer, as part of his solution to one of the problems posed by D. Hilbert. Yet, H. Aiken (one of the pioneers of computer science) wrote in 1956 (as quoted by M. Davis):

If it should turn out that the basic logics of a machine designed for the numerical solution of differential equations coincide with the logics of a machine intended to make bills for a department store, I would regard this as the most amazing coincidence that I have ever encountered.

Such a statement would make any undergraduate-level computer scientist smile today.

Another point of view, which does not have such a good press currently, is the philosophical point of view. To convince oneself of its importance, it suffices to recall the role of intuitionisms in computer science (constructive program synthesis, etc.). Several philosophical questions arise naturally in elementary logic (paradox, truth value, possible worlds, intention, extension, etc.).

The importance of temporal logic in modern computer science is undeniable, and is due (including for philosophical motivations) to philosophers–logicians such as A. N. Prior (see section 10.4).

The philosophical point of view is essential to understand a topic, and of course, understanding is crucial from a practical point of view.

An example is the popular term ontology. J. McCarthy borrowed it in the 1970s for philosophy. Currently, in computer science and artificial intelligence (AI) the meaning of this term has connections (even though they may not be that obvious) with its original philosophical meaning, i.e. “Theory of being as such – the branch of metaphysics that deals with the nature of being, as opposed to the study of their memberships or attributes”.

We conclude this section by mentioning three so-called theoretical topics, of the utmost practical importance:
The NP-completeness of the consistency problem (satisfiability and validity) of classical propositional calculus (SAT), which was proved by Cook in 1971. The wide array of applications of propositional calculus (verification of critical systems, intelligent systems, robotics, constraints, etc.) provides an idea of the importance of this result.

– The study of finite structures, which is closely related to computer science and has numerous applications in databases, multi-agent systems, etc.

– Non-classical logics (modal, temporal and multi-valued logics) are extremely useful in, e.g. program analysis and knowledge representation (in particular in distributed knowledge representation).

1.2. On the utility of logic for computer engineers

Although no one could say for sure which concepts and techniques will be useful to a computer scientist in, say, 10, 20, 30, or 40 years, or even if computer scientists will still be around by then, human beings will probably not have changed that much and will have to cope with an increasingly complex world (at least in developed countries). Abstract concepts will turn out to be more and more useful from the point of view of understanding and efficiency (e.g. it suffices to keep in mind all the advantages – conception, encoding, and updating – that offer very high-level languages).

It is also important to remember that one of the most efficient and rewarding recipes to success from an economical point of view, especially in the modern world, is originality. History shows that the original ideas that have led to important progress in science and techniques are mostly a consequence of an in-depth analysis (and possibly a revision) of principles.

Some of the fundamental notions of logic are used by computer engineers (sometimes with different names and presentations). For example, proposition, definition, semantics, inference, language/meta-language, intention, intension, extension, subclass of formulas, and model.

The notion of proposition from an intuitionistic point of view, for example, represents an intention, a task or a problem. The notion of definition, which has been studied for centuries by logicians, is closely related to that of specification and is used in the process of program construction (folding and unfolding rules).

The role of logic as a language for software specification/verification/development is unchallenged. The notion of semantics leads directly to that of compilation (the assignment of a semantics can be viewed as a translation). Inference is a way of expliciting information and is therefore strongly related to algorithms design. The notion of intention is related not only to that of specification but also to that of system modeling (and not only program modeling), in which it is also necessary to model the
environment (including users). The notions of intension and extension are used, for example, in languages such as Datalog (studied in logic and databases). The notion of decidable subclasses of formulas is an example of a problem whose nature changes when one of its subproblems is considered. The notion of models in the context of abduction (which is used for example in diagnostics, or in the understanding of natural languages) is similar to that of empirical sciences.

These are, of course, very general concepts, but let us mention some concrete examples that have extremely important practical applications. The Davis–Putnam method is used, for example, in system validation. The notion (algorithm) of subsumption is used, among other things, in knowledge representation languages. This notion is essential in so-called ontologies (considered as sets of concepts, relationships between these concepts and means to reason on these concepts), which are widely used in taxonomies and sometimes defined as the explicit specification of a simplified and abstract view of a world we wish to represent.

The notion (algorithm) of unification has been at the intersection of logic and computer science for many years. This algorithm is specified in a very natural way as a set of inference or rewriting rules (see section 4.2). Databases are one of the traditional areas of application of computer science. We can view a relational database as a first-order structure, and first-order logic as a query language (see remark 6.1). The search for increasingly powerful query languages shows the practical need for logics with a greater expressive power.

These examples should be enough to convince the reader of the importance of logic for computer engineers.

To answer those for whom the incentive to learn logic can only come from its recent applications or its relationship to recent applications, we enumerate some applications that have been receiving increasing attention.

The first application is multi-agent systems, which are used in important domains such as robotics, the Internet, etc. These systems can possess extremely complex configurations, and it is becoming necessary to have (formal) techniques to reason in (and on) these systems. The notions and tools that are used include temporal logic, knowledge representation logics, deduction, abduction (i.e. the discovery of hypotheses that are capable of explaining an observation or permit us to reach a conclusion), and the notion of proof that can be used to convince an agent (in particular a human agent) of the pertinence of a piece of information, etc.

Modal logics (temporal logic, dynamic logic, etc.) are used in the industry, for example, to capture reactive behaviors, or in concurrent systems, etc.
An increasing number of computer engineers now work in the economic and financial industry. In these disciplines, the modeling of the actors and their knowledge (beliefs) of the other actors (on the market, in the society) is indispensable for comprehension and decision. Assertions such as “X knows (believes) that Y knows (believes) that Z knows (believes) that…” can be treated formally by knowledge (belief) logics.

In a science popularization article that was published in a well-known journal in May 2001, we could read:

The semantic web is… an extension of the current one.

[...]

For the semantic web to function, computers must have access to structured collections of information and sets of inference rules that they use to conduct automated reasoning.

Logic is very important for natural language processing, which has many applications (e.g. on the Internet, for information retrieval, in question answering systems, etc).

Interdisciplinarity has reached the most recent computer science applications, such as multimedia indexing (i.e. the way of finding multimedia documents in digital libraries), where the roles of semantics and inference are essential.

We conclude this section with some considerations that are more directly related to the technical problems that arise in computer system programming.

– It is possible to prove that a program is not correct, but it is generally impossible to prove that a program is correct using examples. The importance of detecting potential errors in programs (or in integrated circuit designs) is evidenced by two dramatic examples that took place 30 years apart.

The spacecraft Mariner 1 (July 1962) was meant to land on Venus but failed because of an error in the on-board program (a syntactically correct instruction, very close to the intended one, had an entirely different meaning).

Logical techniques have been developed to prove program correctness, to detect programming errors and to construct programs that satisfy a given specification. Most of these techniques can be performed automatically. To reason automatically on a program, it is necessary to have a formal semantics, a formal logical theory, and an automated theorem prover for this theory.

In general, we are interested in verifying specifications, i.e. in proving properties satisfied by the specifications rather than in proving properties satisfied by the programs themselves.
Nowadays, people use more and more certified software. The practical importance of this notion cannot be overstated (it suffices to reflect on the importance of having certified software in a plane or a hospital). More recently (mid-1990’s), errors were discovered in a digital circuit that had already been commercialized. It had been sold by the biggest manufacturer of digital circuits. This error occurred when rare data were provided to the circuit.

This error served as an argument to those who advocated for the necessity of replacing simulation with exhaustive tests by formal verification, so as to prove the correctness of a design before the construction phase.

Theorem proving and model checking are two techniques that enable us to certify the correctness of the aforementioned digital circuits. Logic (both classical and non-classical) plays a key role in these two approaches.

Some resounding successes in software and hardware engineering have proved that these approaches can indeed be applied to real-world problems.

Formal verification had typically been a neglected domain because it was considered too theoretical, but now, it can have a huge financial impact.

- Logic programming consists of using logic as a programming language. The programming language Prolog, as well as others that followed, in particular, constraint logic programming languages, arose naturally from the notions and methods that are studied in logic.

- Some logical paradoxes, in particular, the one named Russell’s paradox, are closely related to the halting problem, i.e. the existence of an algorithm that can decide whether any program provided as an input will halt or not. The impossibility of creating such an algorithm is well known.

- Over the past few years, there has been a boom of computer systems that exhibit an intelligent behavior (such systems are mostly studied in the discipline known as AI).

The principles that are used to design these systems are closely related to classical and non-classical logic. The role of logic in social sciences must not be discarded. Logic is considered as a preferred tool in, e.g. the study of intelligent interactions.
Chapter 2

A Few Thoughts Before the Formalization

2.1. What is logic?

We cannot give a formal answer to this question right away (we will get back to it though). In order to be understood, the answer would require some hindsight on the topic about to be studied.\(^1\)

Try to understand what mathematics is about by only relying on its definition: mathematics, the science of quantity and space.

To choose an answer would not be very helpful right now, because a lack of criteria and references to concrete cases make it difficult to judge the pertinence of the answer.

A problem that is closely related to the one under consideration inspired the following thought to a famous philosopher (D. Hume):

The fact that ideas should logically entail one another is nothing but a fact, no more understandable by itself than any fact of the material world.

And this one from another less famous philosopher:

A logical formula is the expression of a natural phenomenon, as is gravitation or the growth of a tree.

\(^1\) This remark is valid for any topic about to be studied.
2.1.1. Logic and paradoxes

Let us return to the history of logic and try to analyze some of the well-known concepts that are involved.

Logical difficulties arose very early in philosophy, religious “treaties”, and literature. Here are two examples:

– the liar’s paradox, due to Eubulides of Miletus (see digression 2.2): I am lying;
– the sentence: this sentence is false;
– the version of the liar’s paradox due to Epimenides of Knossos (6th Century BC): All Cretans are liars.

Are these really paradoxes?

What is a paradox?

Etymologically (15th Century AD): paradox: contrary to common opinion (doxa: opinion, from which originated heterodox and paradox).

Other definitions characterize paradoxes as argumentations (or assertions) that lead to a contradiction (logical paradoxes).

Paradoxes are sometimes associated with results (obtained by using correct reasoning) that are contrary to intuition and common sense, thus provoking surprise and perplexity.

One may also call a paradox a proposition that seems true (false) and is actually false (true).

Consider the following story (an excerpt from a masterpiece of universal literature):

A soldier is given the order to ask every person about to cross a bridge the following question:

(*) “What have you come here for?”
– If the person tells the truth, he is allowed to cross the bridge.
– If the person is lying, he must be hanged next to the bridge.

Someone arrives, and when asked question (*), shows the gallows next to the bridge and replies: “I have come to be hanged in these gallows”.

Imagine how embarrassed the soldier must feel, as he must either allow someone who lied to cross the bridge, or hang someone who was telling the truth!
There also exist paradoxes that occur in board games.

The rule Every rule has exceptions gives rise to problems. As it is a rule, it must admit some exceptions. Which means that there exist rules that do not admit any exception.

Here is another one:

i) The sentence below is false.
ii) The sentence above is true.
   If (i) is T then (ii) is F hence (i) is F.
   If (i) is F then (ii) is T hence (i) is T.
   If (ii) is F then (i) is T hence (ii) is T.
   If (ii) is T then (i) is F hence (ii) is F.

2.1.2. Paradoxes and set theory

Perhaps those paradoxes that had the most impact are those that involve set theory: probably because of the importance of set theory in mathematics, and particularly in the foundations of mathematics.

Bolzano introduced (in 1847) the notion of a “set” for the first time:

A set is a collection of elements the order of which is not pertinent, and nothing essential is changed by only changing this order.

But it is Cantor who developed set theory.

In naive set theory, according to Cantor, a set is:

Any collection of objects from our intuition or our thoughts, that are both defined and different.

… But allowing the use of any property to define a set can be dangerous:

There exist sets that do not contain themselves. For example:

The set of prime numbers $< 150$

There are others that contain themselves. For example:

The set of all ideas.
The catalog of all catalogs.
From a more abstract point of view:

The set of all sets.

It is simple to show that accepting as a set all the sets in the universe \( U \) leads to a problem.

If this is a set, it must contain itself. But the set of its subsets \( \mathcal{P}(U) \) is also a set, and \( \text{card}(\mathcal{P}(U)) > \text{card}(U) \). Thus, there would exist a set containing more sets than the universe itself!

The set of all the sets that can be described in English by fewer than twenty words.

Bertrand Russell came up in 1902 with a “set” that leads to a paradox, which is known as “Russell’s paradox”:

Let \( B \) denote the set of all sets \( A \) such that \( A \) is not an element of \( A \) (i.e. the set of all sets that do not contain themselves).

The existence of \( B \) leads to a contradiction:

- If \( B \in B \), then \( B \) contains a set that contains itself \( (B) \), and \( B \) must not contain \( B \) (as \( B \) only contains those sets that do not contain themselves), hence \( B \notin B \).
- If \( B \notin B \), then as \( B \) does not contain itself, \( B \) must contain \( B \) (as \( B \) contains all the sets that do not contain themselves), hence \( B \in B \).

Russell’s paradox was popularized as the “barber’s paradox”. A barber must shave every man who does not shave himself. How about him? Must he shave himself? If he does, then he is shaving someone who is shaving himself. Thus, he must not shave himself. If he does not shave himself, then he will not be shaving every man who does not shave himself. Hence, he must shave himself.

The origin of this problem is the axiom of abstraction (also called the axiom of naive comprehension): Given a property, there exists a set whose members are exactly the entities satisfying this property. In other words, for every predicate \( P \), there exists a set whose elements are all the objects (and no other) that satisfy \( P \):

\[
\exists x \forall y (y \in x \iff P(y))
\]

2.1.2.1. The answer

The paradoxes on set theory were believed to be caused by the definition of incomplete objects as sets. Such objects should not be considered as sets.

When a set \( S \) and an object \( ob \) are defined in such a way that:
i) $ob$ is an element of $S$;

ii) the definition of $ob$ depends on $S$.

the definition is said to be *impredicative* (Poincaré). This is the same as what is commonly called a *vicious circle*.

To avoid paradoxes, impredicative definitions must be considered as illegitimate.

In set theory, this leads to the distinction between a *class* and a *proper class*.

A class: $\{x \mid P(x)\}$ ($P$ predicate symbol).

Sets are classes, but not all classes are sets:

$\{x \mid x \in x\}$ (or $\{x \mid x \notin x\}$) is not a set but it is a *proper class*.

Sets are complete entities.

Classes are incomplete entities.

In set theory, a constructive definition of a hierarchy of sets is provided (as usual, $\mathcal{P}(X)$ represents the set of all the subsets of a set $X$):

$\mathcal{F}_0 = \emptyset$

$\mathcal{F}_{n+1} = \mathcal{P}(\mathcal{F}_n)$

$\mathcal{F}_\omega = \bigcup_i \mathcal{F}_i$

It is worth mentioning that impredicative but still important definitions are used in mathematics, such as the definition of the *least upper bound* (it is among all the bounds under consideration), *maximum of a function on an interval* (the maximum is among all the considered values), etc.

These kinds of definitions have also proved their value in computer science (communicating systems and streams).

A set belongs to a family of sets.

The *axiom of choice* (AC) and the *continuum hypothesis* (CH) are of great importance in set theory and in the foundations of mathematics.

We present three versions of the AC:

**Version 1:**

For every set $X$ of non-empty sets $y$, there exists a function $f$ with domain $X$, such that $f(y) \in y$ for all $y \in X$. (It is necessary to use this version only when $X$ is infinite).
**Version 2:**

Let $S$ denote a set of paired disjointed, non-empty sets. There exists a set $C$ containing exactly one element in each member of $S$.

**Version 3 (formalization of version 1):**

$$\forall X \exists f[f \text{ a function with domain } X \land \forall z (z \in X \land \exists u. u \in z \Rightarrow f(z) \in z)]$$

The CH states that between $\aleph_0$ (the cardinality of $\mathbb{N}$) and $\aleph_1$ (the cardinality of $\mathbb{R}$, also denoted by $2^{\aleph_0}$), there is no other transfinite cardinal.

The generalized CH states that the sequence of transfinite cardinals is:

$$\aleph_0, \aleph_1 = 2^{\aleph_0}, \aleph_2 = 2^{\aleph_1}, \ldots \aleph_{n+1} = 2^{\aleph_n}, \ldots$$

The *method* used to construct this sequence is to consider a set and the set of its subsets. For $\aleph_0$ (card $\mathbb{N}$), the method is the one shown in exercise 3.1.

For the others, assume that there exists a bijection $f$ between $E$ and $\mathcal{P}(E)$. By diagonalization (as in exercise 3.1), we can prove that $f$ cannot be onto.

Gödel proved (in 1940) that if the ZF (Zermelo–Fraenkel) axiomatization is consistent (i.e. does not permit us to deduce a formula and its negation), then ZF + AC and ZF + AC + HC are also consistent. This means that AC and HC cannot be refuted.

Paul Cohen proved (in 1963) that AC and HC are independent from the other axioms in ZF, which means that AC and HC cannot be proved if ZF is consistent. HC cannot be proved in ZF + AC.

*ZF and AC are undecidable in ZF.*

**DIGRESSION 2.1.–** (sets and the AC in constructivism). For intuitionists or constructivists, or at least for most of them, a set $E$ is well defined if and only if:

i) we are told how to construct an element of $E$;

ii) we are told how to prove that two elements in $E$ are equal;

iii) we are given a proof that the equality defined in (ii) is an equivalence relation.

If only item (i) is satisfied, $E$ is called a pre-set (it is possible to define sets, using pre-sets as a starting point).

A set $E$ is completely presented iff for any of its elements, one can “read” evidence that it belongs to $E$ (for example, $\mathbb{Q}$ is completely presented, as for every $\frac{m}{n}$, it is possible to verify that $m$ and $n$ have no common factor).
Constructivists consider the AC as essentially non-constructive; however, they accept some (constructive) versions that are provable in constructive mathematics (the name "axiom" was kept out of respect for tradition in classical mathematics). We provide the version named denumerable choice:

\[ \text{AC}_\aleph_0: E \subseteq \mathbb{N}^2 \land \forall m \exists n (\langle m, n \rangle \in E) \Rightarrow \exists f: \mathbb{N} \to \mathbb{N} | \forall m (\langle m, f(m) \rangle \in E) \]

2.1.3. Paradoxes in arithmetic and set theory

Example 2.1.– (Berry’s paradox). Consider the formalization of arithmetic based on Peano’s axioms (see example 3.8).

Let \( A \) denote the set of all natural numbers that can be defined in English by a sentence containing at most a thousand characters (letters or separation symbols). \( A \) is therefore finite, and there must exist natural numbers that do not belong to \( A \). The sentence:

\[ n \text{ is the smallest integer that cannot be defined by an English sentence containing at most a thousand characters} \]

contains fewer than a thousand characters and defines a natural number \( n \). Hence, \( n \) is a member of \( A \). However, by definition, \( n \) does not belong to \( A \).

We shall soon return to the notions of theory and meta-theory.

2.1.3.1. The halting problem

The idea underlying Russell’s paradox can be used to prove the undecidability (insolubility) of the halting problem. This idea can be characterized as the possibility for an expression designating an object to refer to a whole that the object belongs to.

The problem can be stated as follows:

Halting problem: given a program in a language (with the same expressive power as the Turing machine)\(^2\) and some input data\(^3\) (or an initial state of its input tape), decide whether this program (or Turing machine) halts or not.

To prove that this problem can be solved for a certain language, it suffices to provide a program that solves it.

\( ^2 \) Of course, leaving this language unspecified does not entail any loss of generality in the statement.

\( ^3 \) This can be any input data.
However, to prove that it cannot be solved, it is necessary to use an indirect method (for example reductio ad absurdum).

It is possible to encode a program as a set of input data (see, e.g. section 5.9).

We therefore assume that we can write a program $A$, which decides for any program $P$ whether $P$ halts or not, i.e. $A(P)$ returns true if and only if $P$ halts.

We define the program $D$ of Figure 2.1.
- Program $D$ halts if $D$ does not halt;
- Program $D$ does not halt if $D$ halts.

Contradiction. Hence $A$ cannot exist.

DIGRESSION 2.2.– (on the importance of paradoxes). Paradoxes can be viewed as questions about the foundations of our reasonings, of our definitions, of our levels of language, etc. They have greatly influenced traditional logic, and also modern logic, set theory, etc. The principles of paradoxes can be used to obtain fundamental results in mathematical logic (for example Berry’s paradox and the proof of Gödel’s incompleteness theorem, see section 5.9) or in computability (the halting problem).

A typical historical figure that came up with several paradoxes is Eubulides of Miletus (circa 384–322 BC), who was contemporary with Aristotle, with whom he was at odds and against whom he produced several writings.

He is believed to be the father of eristic arguments (i.e. related to controversies, specious reasoning sophisticated quibbling).

It seems like the refutation of these arguments played a central role in the design of Aristotelian logic.
He is the one who came up with the liar’s paradox, and that of the heap of sand (see example 8.1). The latter poses the problem of the relationship between the continuum and the discrete.

There also exist paradoxes (that are not necessarily logical paradoxes but do illustrate the etymology of the word “paradox”) in other scientific domains.

For example:

The paradox known as the Tocqueville effect: revolutions always start when a totalitarian regime becomes more liberal.

The Einstein–Podolsky–Rosen paradox involving quantum physics.

Bertrand’s paradox: one has to determine the probability that a randomly chosen chord in a circle is longer than the side of the equilateral triangle that admits the same circle as a circumcircle. Using the definition of Laplace (probability = number of favorable cases ÷ total number of cases) leads to two different values depending on the way one chooses to define the favorable cases. And both choices are equally reasonable.

2.1.4. On formalisms and well-known notions

G. Frege (19th to 20th Century) was one of the greatest logicians in history. He introduced the notions of formal systems, quantifiers, and proofs that are still used today.

The formalism he proposed was too cumbersome, as evidenced by the following examples.

---

4 Legend has it that the poet and philosopher Philitas of Cos was subject to a deteriorating health as he tried to analyze this paradox.
This formalism was not used very much afterwards (which is not surprising!)\textsuperscript{5}.

We shall often ask ourselves questions about notions that seem natural. To convince ourselves that important problems are often hidden by our habits, it is enlightening to ponder, for example, on a word that can often be found in mathematical statements:

\textbf{Does there exist} \ldots such that \ldots ?

1) According to formalists, \textbf{existence} means consistency, non-contradiction.

2) According to intuitionists\textsuperscript{6}, \textbf{existence} means construction, meaning.

\footnotesize
\begin{itemize}
  \item \textsuperscript{5} One cannot overstate the importance of formalism. For example, McColl proposed $P : Q$ as a notation for the implication $P$ implies $Q$. But the symmetry in this notation suggests that $P$ and $Q$ have the same role... it is thus a bad formalism.
  \item \textsuperscript{6} Some authors prefer to use the term \emph{constructivists} (and \emph{constructivism}), which they believe better mirrors the underlying philosophy. Brouwer, one of the founders of constructivism, viewed mathematics as the activity of carrying out constructions (a primitive and somewhat vague concept) in the mind of an ideal mathematician.
\end{itemize}