Logic for Computer Science - Lecture Notes

Alexandru Ioan Cuza University, Iași
Faculty of Computer Science
University Year 2022-2023

Ștefan Ciobâcă
Andrei Arusoaie
Rodica Condurache
Cristian Masalagiu

Contents

Chapter 1

Introduction

If Arithmetic is the science that studies numbers and operations with numbers, then Logic is the science that studies *propositions* and operations with propositions.

For example, if in Arithmetic we notice that the sum of two even numbers is an even number, then in Logic we could notice that the disjunction of two true sentences is also true.

Logic sits at the intersection of philosophy, mathematics and computer science and has experienced its greatest development starting with the 1950s, because of its numerous applications in Computer Science.

In this course, we will study at an introductory level *propositional logic* and *first-order logic*.

Propositional logic is extremely simple, but the concepts that we study, the methods that we learn and the issues that we face in propositional logic generalize to other more complex logics. Moreover, propositional logic corresponds intimally to the internal organization of computers at an abstract level, in the sense that electronic circuits can be modeled as formulae in propositional logic. Propositional logic has a rich and mathematically interesting theory (examples: compactness theorem, Craig's interpolation theorem). The *satisfiability problem* for propositional logic has many applications in computer science. It is especially important both from a theoretical viewpoint (being the canonical NP-complete problem) and from a practical viewpoint as well (with applications in program verification, circuit verification, combinatorial optimization, and others).

First-order logic is an extension of propositional logic and also has numerous application in computer science, but also in mathematics. For example, all of the math that you have studied in highschool is based on a so-called first-order logic theory called **ZFC** (the **Z**ermelo–**F**raenkel set theory, together with the Axiom of **C**hoice). In Computer Science, applications of first-order logic

appear in the fields of descriptive complexity, relational databases, software and hardware verification, and others. Additionally, several other logics (for example, higher-order logics) have applications in programming languages, the fundamentals of mathematics, type theory, etc.

Chapter 2

Informal Propositional Logic

Propositional logic is the logic of propositions, connected among themselves by *logical connectives* such as *or*, *and* and *not*. In this chapter, we study the basics of propositional logic.

2.1 Propositions

A *proposition* is a statement that is either true or false. Propositions are sometimes called *sentences*. Here are examples of propositions:

1. *I wear a blue shirt.*
2. *You own a laptop computer and a tablet computer, but no smartphone.*
3. $2 + 2 = 4$. (*Two plus two is four.*)
4. $1 + 1 = 1$. (*One plus one is one.*)
5. $1 + 1 \neq 1$. (*One plus one is not one.*)
6. *If* $1 + 1 = 1$*, then I'm a banana.*
7. *All natural numbers are integers.*
8. *All rational numbers are integers.*

Here are examples of things that are not propositions:

1. *Red and Black* (not a statement);

2. π (not a statement);
3. *Is it raining?* (question, not a statement);
4. *Go fish!* (imperative);
5. x *is greater than* 7 (we have here a *predicate of* x; once we set a value for x, the predicate becomes a proposition);
6. *This sentence is false.* (although a statement, it is not a proposition, since it is not either true or false: if it were true, it would need to be false and vice-versa).

Sometimes it is debatable whether something is truly a proposition. For example, we generally agree that *Snow is white* is true, but someone might argue that they have seen black snow, so the truth value of *Snow is white* is put in question. Arguing about whether something is a proposition or not is more a matter of philosophical logic than computer science logic and we will therefore not be too concerned about these sort of issues.

2.2 Atomic Propositions

Some propositions are atomic, in that they cannot be decomposed further into smaller propositions:

1. *I wear a blue shirt.*
2. *You own a laptop computer.*
3. $2 + 2 = 4$. (*Two plus two is four.*)

2.3 Conjunctions

Others however seem to be composed of smaller parts. For example, the proposition *I play games often and I study very well* is composed of two smaller propositions: *I play games often* and *I study very well*, joined together by *and*. When two propositions φ and ψ are joined by an *and*, the resulting proposition φ *and* ψ is called a *conjunction* (*the conjunction of* φ *and* ψ). The propositions φ and ψ are called the *conjuncts* of the proposition φ *and* ψ.

A conjunction is true if both of its conjuncts are true. For example, the proposition *I play games often and I study very well.* is true if both *I play games often* and *I study very well* are true. In particular, as I do not play games often, this proposition is false (when I say it).

Note that a conjunction need not use explicitly the word *and*. For example, the proposition *It is raining outside, but I have an umbrella* is also a conjunction, and it conjuncts are *It is raining outside* and *I have an umbrella*. This particular conjunction uses the adversative conjunction *but*.

Exercise 1. *Find the conjuncts of* I play at home and I study at school..

Exercise 2. *Give an example of a conjunction that is false and an example of a conjunction that is true.*

2.4 Disjunctions

Disjunctions are propositions linked together by *or*. For example, *I can install the software on my smartphone or on my tablet* is a disjunction between *I can install the software on my smartphone* and *I can install the software on my tablet.*. The two parts of the disjunction are called the *disjuncts*.

In the example above, note that the English grammar allows us to omit *I can install the software ...* in the second disjunct, as it is implicit in our understanding of the language. However, when we find the disjuncts, is helps to state them explicitly.

Exercise 3. *Find the disjuncts of* I will buy a laptop or a tablet. *Pay attention! The two disjuncts must be propositions (some words in them could be implicit and not appear in the text).*

A disjunction is true if at least one of the disjuncts is true. For example, *I am Darth Vader or I teach* is true because *I teach* is true (I do not have to worry about being Darth Vader). *I teach or I program* is also true (it happens that both disjuncts are true).

This meaning of disjunctions is called the *inclusive or*. It is standard in mathematics. Sometimes people use *or* in natural language to mean *exclusive or*. For example *Either white wins or black wins in a game of go* is an example where the *or* is exclusive. The meaning of the sentence is that *white wins* or *black wins*, but not both. Here is an example of a false proposition that uses *exclusive or*: *Either I program or I teach* (hint: false because I do both). When you see *either* in a sentence, it is a sign that you are dealing with an *exclusive or*.

In the following, by *disjunction* we will understand *inclusive or* (the standard interpretation in mathematics).

Exercise 4. *Give an example of a false disjunction and an example of a true disjunction.*

Exercise 5. *When is a disjunction* φ or ψ *false (depending on the truth value of* φ *and* ψ*)?*

2.5 Implications

Implications are propositions of the form *if* φ *then* ψ. The proposition φ is called the *antecedent* and the proposition ψ is called the *conclusion* (or *consequent*) of the implication.

An example of an implication is *If I get a passing grade in Logic, I will buy everyone beer.* The antecedent is *I get a passing grade in Logic.* and the conclusion is *I will buy everyone beer.* When is an implication true? Actually, it is easier to say when it is false. An implication is false if and only if the antecedent is true, but the conclusion is false. Assume that I got a passing grade in Logic. Therefore, the proposition *I get a passing grade in Logic* is true. However, I will not buy beer for everyone (just a few select friends). Therefore the proposition *I will buy everyone beer* is false. Therefore the implication *If I get a passing grade in Logic, I will buy everyone beer* as a whole is false (antecedent is true, but conclusion is false).

The meaning of implications is worth a more detailed discussion as it is somewhat controversial. This is mostly because implication as we understand it in mathematics can sometimes be subtly different from implication as we understand it in everyday life. In everyday life, when we say *If I pass Logic, I buy beer*, we understand that there is a cause-and-effect relation between passing Logic and buying beer. This subtle cause-and-effect relation is evident in a number of *if-then* statements that we use in real life: *If I have money, I will buy a car*, *If you help me, I will help you*, etc. We would never connect two unrelated sentences with an implication: the proposition *If the Earth is round, then 2+2=4* would not be very helpful, even though it is true (both the antecedent and the conclusion are true).

This implication that we use in mathematics is called *material implication* or *truth functional implication*, because the truth value of the implication as a whole depends only on the truth values of the antecedent and the conclusion, not on the antecedent and the conclusion itself. This meaning of implication sometimes does not correspond to the meaning of natural language implications, but it turns out that it is the only sensible interpretation of implications in mathematics (and computer science).

In particular, we will take both the propositions *If the Earth is flat, then* $2 + 2 = 5$ and *If the Earth is flat, then* $2 + 2 = 4$ to be true, because the antecedent is false. Implications that are true because the antecedent is false are called *vacuously true.*

Exercise 6. *What are the truth values of* If $2 + 2 = 4$, then the Earth is flat *and* If $2 + 2 = 5$, then the Earth is flat*?*

The truth value of an implication *if* φ *then* ψ, depending on the truth values of its antecedent φ and its conclusion ψ, is summarized in the truth-table below:

φ	ψ	if φ then ψ
false	*false*	*true*
false	*true*	*true*
true	*false*	*false*
true	*true*	*true*

The following example aims at convincing you that the truth table above is the only reasonable one. You must agree that every natural number is also an integer. Otherwise put, the proposition *for any number x, if x is a natural, then x is an integer* is true. In particular, you will agree that the proposition above holds for $x = -10$, $x = 10$ and $x = 1.2$. In particular, the propositions *If* -10 *is a natural, then* -10 *is an integer*, *If* 10 *is a natural, then* 10 *is an integer* and *If* 1.2 *is a natural, then* 1.2 *is an integer* must all be true. This accounts for the first, second and fourth lines of the truth table above (typically, the second line is controversial). As for the third line, false is the only reasonable truth value for an implication *if φ then ψ* where φ is true but ψ is false. Otherwise, we would have to accept propositions such as *If* $2 + 2 = 4$*, then* $2 + 2 = 5$ (antecedent $2 + 2 = 4$ true, conclusion $2 + 2 = 5$ false) as being true.

Implications are sometimes subtle to spot and identify correctly. For example, in the proposition *I will pass Logic only if I study hard* (emphasis on *only if*), the antecedent is *I will pass Logic* and the conclusion is *I study hard*. In particular, the above proposition does not have the same meaning as *If I study hard, then I will pass Logic*.

> Pay attention! In propositions of the form *I will pass Logic only if I study*, the antecedent is *I will pass Logic*, and the consequent is *I study*. This proposition does not have the same meaning as *if I study, then I will pass Logic*.

Implications can sometimes not make use of *if*. For example, take the proposition *I will pass Logic or I will drop school* (apparently a disjunction). This most likely meaning of this proposition is *If I do not pass Logic, then I will drop school*. Thankfully, both of these propositions are equivalent, in a sense that we shall study in the following lectures.

2.6 Negations

A proposition of the form *it is not the case that φ* (or simply *not ψ*) is the *negation* of φ. For example, *It is not raining* is the negation of *It is raining*.

The negation of a proposition takes the opposite truth value. For example, as I am writing this text, the proposition *It is raining* is false, and therefore the proposition *It is not raining* is true.

Exercise 7. *Give an example of a false proposition that uses both a negation and a conjunction.*

2.7 Equivalences

A proposition of the form φ *if and only if* ψ is called an *equivalence* or *double implication*. Such a proposition, as a whole, is true if φ and ψ have the same truth value (both false or both true).

For example, when I am writing this text, *It is raining if and only if it is snowing* is true. Why? Because both of the propositions *It is raining* and *It is snowing* are false.

Exercise 8. *What is the truth value of the proposition* The number 7 is odd if and only if 7 is a prime.*?*

Equivalences are, semantically speaking, conjunctions of two implications: φ *if and only if* ψ gives the same information as

$$\underbrace{\varphi \text{ if } \psi}_{\text{the reverse implication}} \quad \text{and} \quad \underbrace{\varphi \text{ only if } \psi}_{\text{the direct implication}} .$$

The proposition φ *if* ψ is the same as *if* ψ*, then* φ (but it has a different topic). The proposition φ *only if* ψ has the same meaning as *if* φ*, then* ψ, as we dicussed in the previous section on implications.

2.8 Logical Connectives

The words *and*, *or*, *if-then*, *not*, *only if*, *if-and-only-if* (and other similar phrases) are called *logical connectives*, as they can be used to connect smaller propositions in order to obtain larger propositions.

Pay attention! A proposition is *atomic* in propositional logic only if it cannot be decomposed into smaller propositions separated by the connectives discussed above. For example, the proposition *every natural number is an integer* is an atomic proposition (in propositional logic).

The same proposition is not necessarily atomic in other logics. For example, in first-order logic (which we study in the second half of the semester), we have additional logical connectives called *quantifiers* that can be used to construct *every natural number is an integer* from smaller propositions. Therefore, *every natural number is an integer* is not atomic in first-order logic.

2.9 Ambiguities in Natural Language

We have described above the language of propositional logic: atomic propositions connected by *and*, *or*, *not*, etc. So far, our approach has used English. However, English (or any other natural language) is not suitable for our purposes because it exhibits imprecisions in the form of ambiguitites.

Here are a few examples of ambigious propositions:

1. *John and Mary are married* (meaning 1: John and Mary are married to each other; meaning 2: John and Mary are married, but not necessarily to each other).

2. *It is not true that John is tall and Jane is short* (meaning 1: John is not tall and Jane is short; meaning 2: the conjunction *John is tall and Jane is short* is false).

In the study of the laws of logic, such ambiguities can get in the way, just like a wrong computation could impact the resistance of buildings or bridges in civil engineering.

The state of the art in Logic for over 2.000 years, from Aristotle up to the development of Symbolic Logic in the 19th century, has been to use natural language. Symbolic logic (formal logic) has changed the game by introducing languages so precise that there is no risk of misunderstandings.

Such ambiguities impede the study of propositional logic. Therefore we will design a *formal language*, the language of propositional logic, where no ambiguity can occur. The first such language that we study will be the language of *propositional logic*.

Normally, when people say *formal*, they mean it in a bad way, such as having to dress or act formally to go to dinner.

However, in computer science (and in mathematics), formal is a good thing: it means making things so precise that there is no possibility of misunderstanding.

2.10 Exercise Sheet

Exercise 9. *Establish which of the following phrases are propositions:*

1. You own a laptop computer.

2. Snow is white.

3. Snow is not white.

4. My father goes to work and I go to school.

5. It is raining outside, but I have an umbrella.
6. Either it will rain tomorrow, or it won't rain.
7. If I get a passing grade in Logic, I will celebrate.
8. $2 + 2 = 4$. *(*Two plus two is four.*)*
9. Red and Black.
10. π.
11. Is it raining?
12. Let's go fishing!
13. x is greater than 7.
14. This sentence is false.

Exercise 10. *For all propositions that you identified, establish whether they are atomic or molecular. If molecular, establish whether they are conjunctions, negations, etc.*

Chapter 3

The Formal Syntax of Propositional Logic

Next, we will study the formal language of propositional logic.

By *syntax* we generally understand a set of rules for writing correctly. As mentioned previously, in this chapter we develop an artificial language, that we call *formal propositional logic* (or simply *propositional logic*). The formal syntax of propositional logic refers to the rules for writing correctly in this language.

3.1 Alphabets in Computer Science

A set is called an *alphabet* in computer science if we use the elements of the set to make up words. The elements of an alphabet are called *symbols*. Most of the time, an alphabet is a finite set (but not in the case of propositional logic).

What is the difference between an alphabet and a set? A priori, none. The intention matters – what we plan on doing with the elements of the set/alphabet. We use the elements of an alphabet to create *words*. A *word* over an alphabet is a sequence of symbols in the alphabet.

Example 11. *Consider the alphabet* $X = \{0, 1\}$. *The strings/sequences of symbols* `001010`, `101011` *and* `1` *are examples of words of the alphabet* X.

The empty word, formed of 0 symbols of the alphabet, is usually denoted by ϵ.

3.2 The Alphabet of Propositional Logic

The language of propositional logic is made of *propositional formulae*, which model propositions, as discussed in the previous chapter.

Propositional formulae are strings (sequences of characters) over the *alphabet of propositional logic*.

The alphabet of propositional logic is the union of the following sets:

1. $A = \{\mathtt{p}, \mathtt{q}, \mathtt{r}, \mathtt{p}', \mathtt{q}_1, \ldots\}$ is an infinite set of *propositional variables* that we fix from the very beginning;

2. $\{\neg, \wedge, \vee\}$ is the set of *logical connectives*;

3. $\{(,)\}$ is the set of auxiliary symbols; in our case, it consists of two symbols called *brakets*.

The set $L = A \cup \{\neg, \wedge, \vee, (,)\}$ is called the alphabet of propositional logic. Here are a few examples of words over L:

1. $))\mathtt{p} \vee \wedge$;
2. $\vee \vee \neg(\mathtt{p})$;
3. $\mathtt{p}$;
4. $\mathtt{ppp}$;
5. $\neg(\mathtt{p} \vee \mathtt{q})$.

Words, or strings, are simply sequences of symbols of the alphabet L. Some of these words will be formulae or, equivalently, well-formed formulae (wff). Some authors prefer to use the terminology *wff*, but we will simply use *formula* by default in these lecture notes.

As an example, the last word above, $\neg(\mathtt{p} \vee \mathtt{q})$ is a formula of propositional logic, but $\vee \vee \neg(\mathtt{p})$ is not. The following definition captures exactly the set of propositional formulae.

3.3 Propositional Formulae

Definition 12 (The Set of Propositional Formulae ($\mathbb{PL}$))**.** *The set of formulae of propositional logic, denoted $\mathbb{PL}$ from hereon, is the smallest set of words over L satisfying the following conditions:*

• *Base Case. Any propositional variable, seen as a 1-symbol word, is in $\mathbb{PL}$ (equivalently, $A \subseteq \mathbb{PL}$);*

• *Inductive Step i. If* $\varphi \in \mathbb{PL}$*, then* $\neg\varphi \in \mathbb{PL}$ *(equivalently, if the word* φ *is a propositional formula, then so is the word starting with the symbol* $\neg$ *and continuing with the symbols in* φ*);*

• *Inductive Step ii. If* $\varphi_1, \varphi_2 \in \mathbb{PL}$*, then* $(\varphi_1 \wedge \varphi_2) \in \mathbb{PL}$ *(equivalently, exercise);*

• *Inductive Step iii. If* $\varphi_1, \varphi_2 \in \mathbb{PL}$*, then* $(\varphi_1 \vee \varphi_2) \in \mathbb{PL}$ *(equivalently, exercise);*

Here are examples of elements of $\mathbb{PL}$:

$$\mathtt{p} \quad \mathtt{q} \quad \neg\mathtt{p} \quad \neg\mathtt{q} \quad \neg\mathtt{p}' \quad \neg\neg\mathtt{p}_1 \quad (\mathtt{p} \vee \mathtt{q}) \quad (\mathtt{p} \wedge \mathtt{q})$$

$$\neg(\mathtt{p} \vee \mathtt{q}) \quad (\neg\mathtt{p} \wedge \neg\mathtt{q}) \quad \neg(\neg\neg\mathtt{p} \vee \mathtt{p}) \quad ((\mathtt{p} \vee \mathtt{q}) \wedge \mathtt{r}) \quad (\mathtt{p} \vee (\mathtt{q} \wedge \mathtt{r}))$$

Here are examples of words not in $\mathbb{PL}$:

$$\mathtt{pp} \qquad \mathtt{q}\neg\mathtt{q} \qquad \mathtt{q} \wedge \neg\mathtt{p} \qquad \mathtt{p} + \mathtt{q}$$

The definition of the set $\mathbb{PL}$ is an example of an *inductive definition*. Such definitions are really important in computer science and it is a must to understand them very well. In inductive definitions of sets, there are usually some base cases, which say what *base* elements are part of the set and some inductive cases, which explain how to obtain new elements of the set from old elements. Another important part of an inductive definition is the minimality constraint, which says that nothing other than what is provable by the base case(s) and the inductive case(s) belongs to the set. It can be shown that the above set $\mathbb{PL}$ exists and is unique, but the proof is beyond the scope of this course.

3.4 Showing That a Word Is In $\mathbb{PL}$

We can show that a word belongs to $\mathbb{PL}$ by explaining how the rules of the inductive definition were applied. Here is an example of a proof that $\neg(\mathtt{p} \vee \mathtt{q}) \in \mathbb{PL}$:

1. $\mathtt{p} \in \mathbb{PL}$ (by the Base Case, because $\mathtt{p} \in A$);
2. $\mathtt{q} \in \mathbb{PL}$ (by the Base Case, as $\mathtt{q} \in A$);
3. $(\mathtt{p} \vee \mathtt{q}) \in \mathbb{PL}$ (by the Inductive Case iii, with $\varphi_1 = \mathtt{p}$ and $\varphi_2 = \mathtt{q}$);
4. $\neg(\mathtt{p} \vee \mathtt{q}) \in \mathbb{PL}$ (by the Inductive Case i, with $\varphi = (\mathtt{p} \vee \mathtt{q})$).

We can rearrange the above into an equivalent *annotated construction tree* for the formula $\neg(p \vee q)$:

$$\dfrac{\dfrac{\dfrac{}{p \in \mathbb{PL}}\ \textit{Base Case} \qquad \dfrac{}{q \in \mathbb{PL}}\ \textit{Base Case}}{(p \vee q) \in \mathbb{PL}}\ \textit{Inductive Case iii}}{\neg(p \vee q) \in \mathbb{PL}}\ \textit{Inductive Case i}$$

You will often see this notation in Computer Science and it is worth getting to know it. Each line is called an inference; below each line is the conclusion of the inference and above the lines are the hypotheses (0, 1 or more). Besides each line is the name of the rule that was applied.

If we drop all annotation, we obtain the following bare *construction tree* for the formula:

$$\dfrac{\dfrac{\overline{p} \qquad \overline{q}}{(p \vee q)}}{\neg(p \vee q)}$$

It is easy to see that a word belongs to $\mathbb{PL}$ iff there is a construction tree for it.

3.5 The Main Connective of a Formula

A formula that consists of a single propositional variable, such as p or q, is called an *atomic formula*. This explains why the set A of propositional variables is called A (A stands for *atomic*).

More complex formulae, such as $\neg p$ or $p \vee q$, are called *molecular* (to distinguish them from atomic formulae).

Each molecular formula has a *main connective*, which is given by the last inference in its construction tree. For example, the main connective of the formula $\neg(p \vee q)$ is $\neg$ (the negation), while the main connective of the fomrula $(\neg p \vee q)$, is $\vee$ (the disjunction). We call formulae whose main connective is $\wedge$ *conjunctions*. Similarly, if the main connective of a formula is a $\vee$, it is a *disjunction* and if the main connective is $\neg$, then it is a *negation*.

3.6 Showing That a Word Is Not in $\mathbb{PL}$

It is somewhat more difficult (or rather more verbose) to show that a word is not in $\mathbb{PL}$.

If the word is not over the right alphabet, such as the three-symbol word $\mathtt{p} + \mathtt{q}$, which uses a foreign symbol $+$, then we can easily dismiss it as $\mathbb{PL}$ only contains words over L.

However, if the word is over the right alphabet and we want to show that it is not part of $\mathbb{PL}$, we must make use of the minimality condition. Here is an example showing that $(\mathtt{p}\neg\mathtt{q}) \notin \mathbb{PL}$:

Let's assume (by contradiction) that $(\mathtt{p}\neg\mathtt{q}) \in \mathbb{PL}$. Then, by the minimality condition, it follows that $(\mathtt{p}\neg\mathtt{q}) \in \mathbb{PL}$ must be explained by one of the rules Base Case, Inductive Case i – iii.

But we cannot apply the Base Case to show that $(\mathtt{p}\neg\mathtt{q}) \notin \mathbb{PL}$, since $(\mathtt{p}\neg\mathtt{q}) \notin A$.

But we cannot apply the Inductive Case i either, because there is no formula φ such that $(\mathtt{p}\neg\mathtt{q}) = \neg\varphi$ (no matter how we would choose $\varphi \in \mathbb{PL}$, the first symbol of the word on the lhs would be $($, while the first symbol of the word on the rhs would be $\neg$).

But we cannot apply the Inductive Case ii either, because there are no formulae $\varphi_1, \varphi_2 \in \mathbb{PL}$ such that $(\mathtt{p}\neg\mathtt{q}) = (\varphi_1 \vee \varphi_2)$ (no matter how we would choose $\varphi_1, \varphi_2 \in \mathbb{PL}$, the word on the lhs would not contain the symbol $\neg$, while while the the word on the rhs would contain it).

By similar reasons, we cannot apply Inductive Case iii either.

But this contradicts our assumption and therefore it must be the case that $(\mathtt{p}\neg\mathtt{q}) \notin \mathbb{PL}$.

3.7 Unique Readability

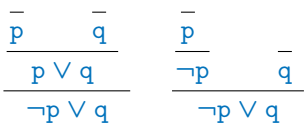
The definition of $\mathbb{PL}$ has an important property called *unique readability*:

Theorem 13 (Unique Readability of Propositional Formulae)**.** *For any word $\varphi \in \mathbb{PL}$, there is a unique construction tree.*

The property above is also sometimes called unambiguity of the grammar. Proving the Unique Readability Theorem is beyond the scope of the present lecture notes. Instead we will try to understand the importance of the property above by showing how an alternative, fictive, definition of the set $\mathbb{PL}$ would be ambiguous:

Beginning of Wrong Definition of $\mathbb{PL}$.

> Imagine that the Inductive Case ii would read:
>
> $\vdots$
>
> 3. (Inductive Step ii) If $\varphi_1, \varphi_2 \in \mathbb{PL}$, then $\varphi_1 \vee \varphi_2 \in \mathbb{PL}$.

> ⋮
>
> The only difference would be that we do not place parantheses around disjunctions. With this alternative, fictive, definition of $\mathbb{PL}$, we have that the word $\neg p \vee q \in \mathbb{PL}$. However, this word has two different construction trees:
>
> 
>
>
> Such an ambiguity would be very troubling, because we would not know if $\neg p \vee q$ is a disjunction (if we would use the construction tree on the right) or a negation (the construction tree on the left). In fact, avoiding such syntactic ambiguities was the main reason why we left natural language and began studying formal logic.

End of Wrong Definition of $\mathbb{PL}$.

I hope that you can now see that the Unique Readability Theorem is non-trivial (as an exercise to the more mathematically inclined students, I suggest that you try to prove it) and that it is a very important property of our definition of formulae, since it essential says that any propositional formula can be read unambiguously.

3.8 Object-language and meta-language

A peculiarity in logic is that we study propositions (we analyse, for example, their truth values), but in order to perform such study, we use a form of reasoning that also uses propositions. For example, in our study, we could make the following argument:

> *The proposition I do not go to school is false if the proposition I go to school is true.*

Note that the sentences that are the object of our study (*I do not go to school* and *I go to school*) are propositions, but so is the entire statement *The proposition I do not go to school is false if the proposition I go to school is true*. In this way, it seems that we are performing a circular reasoning, in which we study our own method of study.

This apparent difficulty does not occur in the case of arithmetic, for example. In arithmetic, we study numbers and operations over numbers, and we reason about numbers using propositions.

The difficulty can be solved by strictly separating *the object language* from the *meta-language*. The object language is the language that we study ($\mathbb{PL}$), and the meta-language is the language that we use to perform the study (English).

> The object language is the language that represents the object of our study (in our case, propositional logic). The meta-language is the language that we use to communicate about the object of our study (in our case, English).

In this course, to more easily differentiate between the two, we make the following convention: all elements in the object language are written in sans-serif blue font (for example, $(\mathsf{p} \wedge \mathsf{q})$), and the statement in the meta-language are written using *a regular black font* (for example, *any atomic formulae is a formula*).

> It is extremly important to differentiate correctly between object language and meta-language. To this effect, the lecture notes use a typographic convention. The elements of the meta-language are written in regular black font. The elements of the object-language are writte as follows: $(\mathsf{p} \wedge \mathsf{q})$. For this reason, if you print out the lecture notes, please print them in color.

3.9 Exercise Sheet

Exercise 14. *Show that the following words are* propositional formulae *(i.e., element of $\mathbb{PL}$), by explaining which are the construction steps (base case, respectively one of the three inductive steps):*

1. $\neg\mathsf{q}$;
2. $(\mathsf{p}_1 \wedge \mathsf{q})$;
3. $\neg(\mathsf{p} \vee \mathsf{q})$;
4. $(\neg\mathsf{p} \vee \neg\mathsf{q})$;
5. $\neg(\neg\mathsf{p} \vee (\mathsf{q} \wedge \neg\mathsf{q}))$.

Exercise 15. *Show that the following words over the alphabet L are not elements of $\mathbb{PL}$(hint: show that none of the four construction rules applies):*

1. $((\neg)\mathtt{q})$;
2. $\mathtt{q} \wedge \neg$;
3. $\mathtt{pq}$;
4. $\mathtt{p} \wedge \mathtt{q}$;
5. $(\mathtt{p}\neg\mathtt{q})$;
6. $(\mathtt{p}) \wedge (\mathtt{q})$.

Exercise 16. *Which of the following are formulae (in $\mathbb{PL}$) and which are not:*

1. $\mathtt{p}_1$;
2. $\mathtt{p}_1 \vee \mathtt{q}_1$;
3. $(\mathtt{p}_1 \vee \mathtt{q}_1)$;
4. $(\neg\mathtt{p}_1 \vee \mathtt{q}_1)$;
5. $((\neg\mathtt{p}_1) \vee \mathtt{q}_1)$;
6. $(\neg\mathtt{p})$?

Exercise 17. *Give 5 examples of interesting formulae (with many variables/operators, etc.).*

Chapter 4

Functions Defined Recursively on $\mathbb{PL}$

Reminder. *Recall that when X is a set, by 2^X we denote the* powerset of X, *that is, the set of all subsets of X. For example, if $X = \{1, 2, 3\}$, then $2^X = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$. When X is finite, note that $|2^X| = 2^{|X|}$, which partly explains this notation. You might have used the notation $\mathcal{P}(X)$ instead of 2^X in highschool.*

Whenever a set is inductively defined, we can define recursive functions that operate on the elements of the set, function which make use of the *structure* of the elements.

Here is an example of a function computing the set of *subformulae* of a formula:

$subf : \mathbb{PL} \to 2^{\mathbb{PL}}$, defined by:

$$subf\Big(\varphi\Big) = \begin{cases} \{\varphi\}, & \text{if } \varphi \in A; \\ \{\varphi\} \cup subf\Big(\varphi'\Big), & \text{if } \varphi = \neg\varphi' \text{ and } \varphi' \in \mathbb{PL}; \\ \{\varphi\} \cup subf\Big(\varphi_1\Big) \cup subf\Big(\varphi_2\Big), & \\ \qquad \text{if } \varphi = (\varphi_1 \wedge \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}; & \\ \{\varphi\} \cup subf\Big(\varphi_1\Big) \cup subf\Big(\varphi_2\Big), & \\ \qquad \text{if } \varphi = (\varphi_1 \vee \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}. & \end{cases}$$

Here is how we can compute $subf\Big(\varphi\Big)$ when $\varphi = \neg(\mathtt{p} \vee \mathtt{q})$:

$$\begin{aligned} subf\Big(\neg(\mathtt{p} \vee \mathtt{q})\Big) &= \{\neg(\mathtt{p} \vee \mathtt{q})\} \cup subf\Big((\mathtt{p} \vee \mathtt{q})\Big) \\ &= \{\neg(\mathtt{p} \vee \mathtt{q})\} \cup \{(\mathtt{p} \vee \mathtt{q})\} \cup subf\Big(\mathtt{p}\Big) \cup subf\Big(\mathtt{q}\Big) \\ &= \{\neg(\mathtt{p} \vee \mathtt{q})\} \cup \{(\mathtt{p} \vee \mathtt{q})\} \cup \{\mathtt{p}\} \cup \{\mathtt{q}\} \\ &= \{\neg(\mathtt{p} \vee \mathtt{q}), (\mathtt{p} \vee \mathtt{q}), \mathtt{p}, \mathtt{q}\}. \end{aligned}$$

Notice that we use two different types of brackets in the computation above. On one hand, we have the brackets that surround the argument to the function *subf*, and on the other hand we have the usual brackets that are part of the alphabet L. The former brackets are the usual brakets in mathematics, while the later brackets are simply symbols in our alphabet. In order to differentiate between them, we adopt the convention to use bigger brakets for the function call and the usual brackets in blue for the auxiliary symbols.

Note that both are important. For example, it would be an error to write $subf\Big(\mathtt{p} \vee \mathtt{q}\Big)$ instead of $subf\Big((\mathtt{p} \vee \mathtt{q})\Big)$, as the word $\mathtt{p} \vee \mathtt{q} \notin \mathbb{PL}$ is not a formula.

The function *subf* is the first in a long line of functions defined recursively on the structure of a formula $\varphi \in \mathbb{PL}$. These functions are called *structurally recursive*.

It is very important to understand how such functions operate in order to understand the remainder of this course. Here are several more examples of such functions.

4.1 The Abstract Syntax Tree of a Formula

Here is an example of a function computing the *abstract syntax tree* of a formula:

$ast : \mathbb{PL} \to Trees$, defined by:

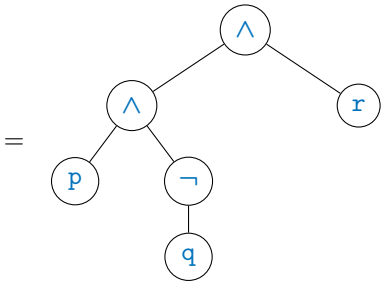
$$ast\Big(\varphi\Big) = \begin{cases} \varphi, & \text{if } \varphi \in A; \\ \neg \\ ast\Big(\varphi'\Big), & \text{if } \varphi = \neg\varphi' \text{ and } \varphi' \in \mathbb{PL}; \\ \vee \\ ast\Big(\varphi_1\Big) \quad ast\Big(\varphi_2\Big), & \text{if } \varphi = (\varphi_1 \wedge \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}; \\ \wedge \\ ast\Big(\varphi_1\Big) \quad ast\Big(\varphi_2\Big), & \text{if } \varphi = (\varphi_1 \vee \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}. \end{cases}$$

Here is the computation of the abstract syntax tree of the formula $((\mathtt{p} \wedge \neg\mathtt{q}) \wedge \mathtt{r})$.

$$ast\Big(((\mathtt{p} \wedge \neg\mathtt{q}) \wedge \mathtt{r})\Big) = \wedge \quad ast\Big((\mathtt{p} \wedge \neg\mathtt{q})\Big) \quad ast\Big(\mathtt{r}\Big)$$

$$= \wedge \quad \mathtt{r} \quad ast\Big(\mathtt{p}\Big) \quad ast\Big(\neg\mathtt{q}\Big)$$

$$= \wedge \quad \mathtt{r} \quad \mathtt{p} \quad \neg \quad ast\Big(\mathtt{q}\Big)$$



Note that *Trees* is the set of labeled, rooted trees, considered here intuitively, without a formal definition.

Note that abstract syntax trees are very important. In fact, conceptually, a propositional formula *is* its abstract syntax tree. However, because writing trees is cumbersome, we have resort to the more conventional left-to-right notation.

4.2 Other Examples of Recursively Defined Functions

Here are three more examples of functions defined by structural recursion on formulae.

The first function, $height : \mathbb{PL} \to \mathbb{N}$, computes the *height* of the ast o formula:

$$height\Big(\varphi\Big) = \begin{cases} 1, & \text{if } \varphi \in A; \\ 1 + height\Big(\varphi'\Big), & \text{if } \varphi = \neg\varphi' \text{ and } \varphi' \in \mathbb{PL}; \\ 1 + max\Big(height\Big(\varphi_1\Big), height\Big(\varphi_2\Big)\Big), & \text{if } \varphi = (\varphi_1 \wedge \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}; \\ 1 + max\Big(height\Big(\varphi_1\Big), height\Big(\varphi_2\Big)\Big), & \text{if } \varphi = (\varphi_1 \vee \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}. \end{cases}$$

The function $max : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, which occurs in the definition of *height*, is the well-known function that computes the maximum between two natural numbers.

The next function, $size : \mathbb{PL} \to \mathbb{N}$, computes the *size* of the abstract syntax tree of a formula (i.e., the number of nodes):

$$size\Big(\varphi\Big) = \begin{cases} 1, & \text{if } \varphi \in A; \\ 1 + size\Big(\varphi'\Big), & \text{if } \varphi = \neg\varphi' \text{ and } \varphi' \in \mathbb{PL}; \\ 1 + size\Big(\varphi_1\Big) + size\Big(\varphi_2\Big), & \text{if } \varphi = (\varphi_1 \wedge \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}; \\ 1 + size\Big(\varphi_1\Big) + size\Big(\varphi_2\Big), & \text{if } \varphi = (\varphi_1 \vee \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}. \end{cases}$$

The final example function, $prop : \mathbb{PL} \to 2^A$, compute the set of propositional variables occurring in a formula:

$$prop = \begin{cases} \{\varphi\}, & \text{if } \varphi \in A; \\ prop, & \text{if } \varphi = \neg\varphi' \text{ and } \varphi' \in \mathbb{PL}; \\ prop \cup prop, & \text{if } \varphi = (\varphi_1 \wedge \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}; \\ prop \cup prop, & \text{if } \varphi = (\varphi_1 \vee \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}. \end{cases}$$

Make sure that you are proficient at understanding, defining and computing with functions such as those given above.

4.3 Proofs by Structural Induction

We will sometimes do proofs by *structural induction*. You are already familiar with proofs by induction on the naturals.

In order to prove a theorem of the form

$$\text{For all } n \in \mathbb{N}, \text{ it is true that } P(n),$$

the mathematical induction principle postulates that it is sufficient to show that:

1. (Base Case) $P(0)$ holds;
2. (Induction Case) $P(k)$ implies $P(k+1)$, for all $k \in \mathbb{N}$.

Proofs by structural induction generalize the principle above to any inductively defined set such as $\mathbb{PL}$.

For the case of $\mathbb{PL}$, the structural induction principle is that, in order to show a theorem of the form

For any propositional formula $\varphi \in \mathbb{PL}$, it is true that $P\Big(\varphi\Big)$,

it is sufficient to prove that:

1. (Base Case) $P\Big(\varphi'\Big)$ holds for any atomic formula $\varphi' \in A$ (otherwise put, the property P holds of every atomic formula);
2. (Inductive Case i) $P\Big(\neg\varphi'\Big)$ holds whenever $P\Big(\varphi'\Big)$ holds;
3. (Inductive Case ii) $P\Big((\varphi_1 \wedge \varphi_2)\Big)$ holds whenever $P\Big(\varphi_1\Big)$ and $P\Big(\varphi_2\Big)$ hold;
4. (Inductive Case iii) $P\Big((\varphi_1 \vee \varphi_2)\Big)$ holds whenever $P\Big(\varphi_1\Big)$ and $P\Big(\varphi_2\Big)$ hold.

Here is an example of a theorem and its proof by structural induction:

Theorem 18 (Example of Theorem Provable by Structural Induction)**.** *For any propositional formula φ, we have that $height\Big(\varphi\Big) \leq size\Big(\varphi\Big)$.*

Proof: We proceed by structural induction (the property $P(\varphi)$ is simply that $height\Big(\varphi\Big) \leq size\Big(\varphi\Big)$). It is sufficient to show that:

1. (Base Case) $height\Big(\varphi'\Big) \leq size\Big(\varphi'\Big)$ for any propositional variable $\varphi' \in A$.

   But by definition, $height\Big(\varphi'\Big) = 1$ and $size\Big(\varphi'\Big) = 1$ when $\varphi' \in A$. And $1 \leq 1$, which is what we had to show.

2. (Inductive Case i) Assuming that $height\Big(\varphi'\Big) \leq size\Big(\varphi'\Big)$, we show that $height\Big(\neg\varphi'\Big) \leq size\Big(\neg\varphi'\Big)$.

   But by definition, $height\Big(\neg\varphi'\Big) = 1 + height\Big(\varphi'\Big)$ and $size\Big(\neg\varphi'\Big) = 1 + size\Big(\varphi'\Big)$. And $1 + height\Big(\varphi'\Big) \leq 1 + size\Big(\varphi'\Big)$ (what we had to show), as we already know $height\Big(\varphi'\Big) \leq size\Big(\varphi'\Big)$.

3. (Inductive Case ii) Assuming that $height\Big(\varphi_1\Big) \leq size\Big(\varphi_1\Big)$ and $height\Big(\varphi_2\Big) \leq size\Big(\varphi_2\Big)$, we show $height\Big((\varphi_1 \vee \varphi_2)\Big) \leq size\Big((\varphi_1 \vee \varphi_2)\Big)$.

But, by definition, $height\Big((\varphi_1 \vee \varphi_2)\Big) = 1 + max\Big(height\Big(\varphi_1\Big), height\Big(\varphi_2\Big)\Big)$ and, as $max(a,b) \leq a+b$ (for any naturals $a, b \in \mathbb{N}$), we have that $height\Big((\varphi_1 \vee \varphi_2)\Big) \leq 1 + height\Big(\varphi_1\Big) + height\Big(\varphi_2\Big)$. But $height\Big(\varphi_i\Big) \leq size\Big(\varphi_i\Big)$ $(1 \leq i \leq 2)$ by our hypothesis, and therefore $height\Big((\varphi_1 \vee \varphi_2)\Big) \leq 1 + size\Big(\varphi_1\Big) + size\Big(\varphi_2\Big)$. But, by definition, $size\Big((\varphi_1 \vee \varphi_2)\Big) = 1 + size\Big(\varphi_1\Big) + size\Big(\varphi_2\Big)$, and therefore $height\Big((\varphi_1 \vee \varphi_2)\Big) \leq size\Big((\varphi_1 \vee \varphi_2)\Big)$, what we had to prove.

4. (Inductive Case iii) Similar to Inductive Case ii.

q.e.d.

4.4 Exercise Sheet

Exercise 19. *Compute, using the function subf, the set of subformulae of the following formulae:*

1. $((\mathtt{p} \wedge \neg\mathtt{q}) \wedge \mathtt{r})$; *2.* $((\mathtt{p} \vee \neg\mathtt{q}) \wedge \mathtt{r})$; *3.* $\neg((\mathtt{p} \vee \neg\mathtt{q}) \wedge \mathtt{r})$.

Exercise 20. *Compute the abstract syntax trees of the following formulae:*

1. $((\mathtt{p} \wedge \neg\mathtt{q}) \wedge \mathtt{r})$;
2. $((\mathtt{p} \vee \neg\mathtt{q}) \wedge \mathtt{r})$;
3. $\neg((\mathtt{p} \vee \neg\mathtt{q}) \wedge \mathtt{r})$;
4. $(\neg(\mathtt{p} \vee \neg\mathtt{q}) \wedge \mathtt{r})$.

Exercise 21. *Recall the recursive definition of the function height* $: \mathbb{PL} \to \mathbb{N}$*, which computes, given a formula, the height of its abstract syntax tree. Compute the height of the formulae shown in Exercise 20.*

Exercise 22. *Recall the recursive definition of the function size* $: \mathbb{PL} \to \mathbb{N}$*, which computes the number of nodes in abstract syntax tree of a formula. Compute the size of the formulae shown in Exercise 20.*

Exercise 23. *Recall the recursive definition of the function prop* $: \mathbb{PL} \to 2^A$*, which computes, given a formula, the set of propositional variables occuring in the formula. Compute the set of propositional variables occuring in the formulae shown in Exercise 20.*

Exercise 24. *Show by structural induction that* $height(\varphi) < size(\varphi) + 1$ *for any formula* $\varphi \in \mathbb{PL}$.