Part I

Propositional logic

Chapter 1

Syntax and Semantics of Propositional Logic

Syntax is a set of formal rules for creating well-formed sentences consisting of words (in the case of natural languages) or formal expressions consisting of symbols (e.g., statements in a programming language). In contrast, *semantics* describes the meaning of such expressions. The relationship between syntax and semantics is fundamental to all of logic and is therefore the key to its understanding.

1.1 Syntax of Propositional Logic

First, we define the formal 'statements' with which we will work in propositional logic.

1.1.1 Language

The *language* of propositional logic is determined by a non-empty set of *propositional variables* $\mathbb{P}$ (also called *atomic propositions* or *atoms*). This set can be finite or infinite, but it will usually be countable[1] (unless otherwise specified), and it will have a fixed ordering. For propositional variables, we will typically use the notation p_i (from the word "proposition"), but for better readability, especially if $\mathbb{P}$ is finite, we will also use $p, q, r, \ldots$. For example:

$$\mathbb{P}_1 = \{p, q, r\}$$
$$\mathbb{P}_2 = \{p_0, p_1, p_2, p_3, \ldots\} = \{p_i \mid i \in \mathbb{N}\}$$

In addition to propositional variables, the language also includes *logical symbols*: symbols for logical connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ and parentheses $(,)$. However, for simplicity, we will talk about the "language $\mathbb{P}$".

Remark 1.1.1. If we need to formally express the ordering of the propositional variables in $\mathbb{P}$, we imagine it as a bijection $\iota\colon \{0, 1, \ldots, n-1\} \to \mathbb{P}$ (for a finite, n-element language) or $\iota\colon \mathbb{N} \to \mathbb{P}$ (if $\mathbb{P}$ is countably infinite). In our examples, $\iota_1(0) = p$, $\iota_1(1) = q$, $\iota_1(2) = r$, and $\iota_2(i) = p_i$ for all $i \in \mathbb{N}$.[2]

[1] This is important in many applications in computer science, as uncountable sets cannot fit into a (even infinite) computer.

[2] The set of natural numbers $\mathbb{N}$ includes zero, see standard ISO 80000-2:2019.

1.1.2 Proposition

The basic building block of propositional logic is a *proposition*, also called *propositional formula*. It is a finite string composed of propositional variables and logical symbols according to certain rules. Atomic propositions are propositions, and we can further create propositions from simpler propositions and logical symbols: for example, for the logical connective $\wedge$, we write first the symbol '(', then the first proposition, the symbol '$\wedge$', the second proposition, and finally the symbol ')'.

Definition 1.1.2 (Proposition)**.** A *proposition* (*propositional formula*) in the language $\mathbb{P}$ is an element of the set $\mathrm{PF}_{\mathbb{P}}$ defined as follows: $\mathrm{PF}_{\mathbb{P}}$ is the smallest set satisfying[3]

- for every atomic proposition $p \in \mathbb{P}$, $p \in \mathrm{PF}_{\mathbb{P}}$,
- for every proposition $\varphi \in \mathrm{PF}_{\mathbb{P}}$, $(\neg\varphi)$ is also an element of $\mathrm{PF}_{\mathbb{P}}$,
- for every $\varphi, \psi \in \mathrm{PF}_{\mathbb{P}}$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \to \psi)$, and $(\varphi \leftrightarrow \psi)$ are also elements of $\mathrm{PF}_{\mathbb{P}}$.

Propositions are usually denoted by Greek letters φ, ψ, χ (φ from the word "formula"). To avoid listing all four binary logical connectives, we sometimes use a placeholder symbol $\square$. Thus, the third point of the definition could be expressed as:

- for every $\varphi, \psi \in \mathrm{PF}_{\mathbb{P}}$ and $\square \in \{\wedge, \vee, \to, \leftrightarrow\}$, $(\varphi \square \psi)$ is also an element of $\mathrm{PF}_{\mathbb{P}}$.

A *subproposition* (*subformula*) is a substring that is itself a proposition. Note that all propositions are necessarily finite strings, created by applying a finite number of steps from the definition to their subpropositions.

Example 1.1.3. The proposition $\varphi = ((p \vee (\neg q)) \leftrightarrow (r \to (p \wedge q)))$ has the following subpropositions: $p, q, (\neg q), (p \vee (\neg q)), r, (p \wedge q), (r \to (p \wedge q)), \varphi$.

A proposition in the language $\mathbb{P}$ does not have to contain all atomic propositions from $\mathbb{P}$ (and it cannot if $\mathbb{P}$ is an infinite set). Therefore, it will be useful to denote by $\mathrm{Var}(\varphi)$ the set of atomic propositions occurring in φ.[4] In our example, $\mathrm{Var}(\varphi) = \{p, q, r\}$.

We introduce abbreviations for two special propositions: $\top = (p \vee (\neg p))$ (*tautology*) and $\bot = (p \wedge (\neg p))$ (*contradiction*), where $p \in \mathbb{P}$ is fixed (e.g., the first atomic proposition from $\mathbb{P}$). Thus, the proposition $\top$ is always true, and the proposition $\bot$ is always false.

When writing propositions, we may omit some parentheses for better readability. For example, the proposition φ from Example 1.1.3 can be represented by the string $p \vee \neg q \leftrightarrow (r \to p \wedge q)$. We omit outer parentheses and use priority of operators: $\neg$ has the highest priority, followed by $\wedge, \vee$, and finally $\to, \leftrightarrow$ have the lowest priority. Furthermore, the notation $p \wedge q \wedge r \wedge s$ means the proposition $(p \wedge (q \wedge (r \wedge s)))$, and similarly for $\vee$.[5][6]

[3] This kind of definition is called *inductive*. It can also be naturally expressed using a *formal grammar*, see the course NTIN071 Automata and Grammars.

[4] If we do not specify the language of a proposition (and if it is not clear from the context), we mean that it is in the language $\mathrm{Var}(\varphi)$.

[5] Due to the associativity of $\wedge, \vee$, the placement of parentheses does not matter.

[6] Sometimes finer priorities are introduced, $\wedge$ often has higher priority than $\vee$, and $\to$ has higher priority than $\leftrightarrow$. Also, sometimes $p \to q \to r$ is written instead of $(p \to (r \to q))$, although $\to$ is not associative and so here the placement of parentheses does matter. We prefer to avoid both of those conventions.

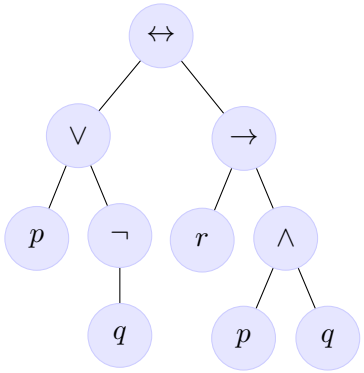


Figure 1.1: Tree of the proposition $\varphi = ((p \vee (\neg q)) \leftrightarrow (r \rightarrow (p \wedge q)))$

1.1.3 Tree of a Proposition

In the definition of a proposition, we chose *infix* notation (with parentheses) purely for human readability. Nothing would prevent us from using *prefix* ("Polish") notation, i.e., defining propositions as follows:

- every atomic proposition is a proposition, and
- if φ, ψ are propositions, then $\neg\varphi$, $\wedge\varphi\psi$, $\vee\varphi\psi$, $\rightarrow\varphi\psi$, and $\leftrightarrow\varphi\psi$ are also propositions.

The proposition $\varphi = ((p \vee (\neg q)) \leftrightarrow (r \rightarrow (p \wedge q)))$ would then be written as $\varphi = \leftrightarrow\vee p\neg q\rightarrow r\wedge pq$. We could also use *postfix* notation and write $\varphi = pq\neg\vee rpq\wedge\rightarrow\leftrightarrow$. The essential information about a proposition is actually contained in its tree structure, which captures how it is composed of simpler propositions, similar to the tree of an arithmetic expression.

Example 1.1.4. The tree of the proposition $\varphi = ((p \vee (\neg q)) \leftrightarrow (r \rightarrow (p \wedge q)))$ is illustrated in Figure 1.1. Notice also how the subpropositions of φ correspond to subtrees. The proposition φ is obtained by traversing the tree from the root, and at each node:

- if the label is an atomic proposition, write it out,
- if the label is a negation: write '(¬', recursively call the child, write ')',
- otherwise (for binary logical connectives): write '(', call the left child, write the label, call the right child, write ')'.[7]

Now we will define the tree of a proposition formally, *by induction on the structure of the proposition*:[8]

Definition 1.1.5 (Tree of a Proposition)**.** The *tree of a proposition* φ, denoted $\mathrm{Tree}(\varphi)$, is a rooted ordered tree, defined inductively as follows:

[7] Prefix and postfix notations would be obtained similarly, but we do not write parentheses, and the label is written immediately upon entering or just before leaving the node.

[8] Once we have the tree of a proposition defined, we can understand induction on the structure of the proposition as induction on the depth of the tree. For now, understand it as induction on the number of steps in Definition 1.1.2 by which the proposition was created. Alternatively, induction on the length of the proposition or the number of logical connectives would work as well.

- If φ is an atomic proposition p, Tree(φ) contains a single node, and its label is p.
- If φ is of the form $(\neg\varphi')$, Tree(φ) has a root labeled $\neg$, and its single child is the root of Tree(φ').
- If φ is of the form $(\varphi' \square \varphi'')$ for $\square \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, Tree($\varphi$) has a root labeled $\square$ with two children: the left child is the root of the tree Tree(φ'), and the right child is the root of Tree(φ'').

Exercise 1.1. Prove that every proposition has a uniquely determined proposition tree, and vice versa.

1.1.4 Theory

In practical applications, we do not express the desired properties with a single proposition — it would have to be very long and complex and difficult to work with — but with many simpler propositions.

Definition 1.1.6 (Theory)**.** A *theory* in the language $\mathbb{P}$ is any set of propositions in $\mathbb{P}$, that is, any subset $T \subseteq \mathrm{PF}_{\mathbb{P}}$. The individual propositions $\varphi \in T$ are also called *axioms*.

Finite theories could be replaced by a single proposition: the conjunction of all their axioms. But that would not be practical. Moreover, we also allow infinite theories (a trivial example is the theory $T = \mathrm{PF}_{\mathbb{P}}$), and the empty theory $T = \emptyset$.[9]

1.2 Semantics of Propositional Logic

In our logic, the semantics are given by one of two possible values: *True* or *False*. (In other logical systems, semantics can be more interesting.)

1.2.1 Truth Value

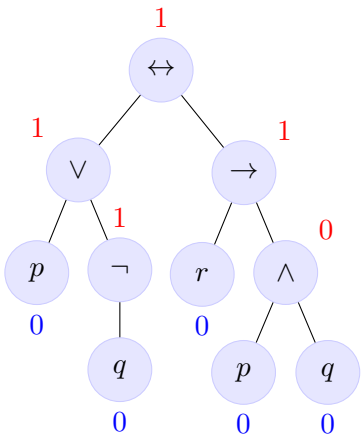
Propositions can be assigned one of two possible truth values: *True* (1) or *False* (0). Atomic propositions represent simple, indivisible statements (hence the name '*atomic*'); their truth value must be assigned to correspond to what we want to model (that is why we call them *propositional variables*). Once we *assign* truth values to the atomic propositions, the truth value of any compound proposition is uniquely determined and can be easily calculated according to the tree of the proposition:

Example 1.2.1. Let us calculate the truth value of the proposition $\varphi = ((p \vee (\neg q)) \leftrightarrow (r \rightarrow (p \wedge q)))$ for the assignments (a) $p = 0$, $q = 0$, $r = 0$ and (b) $p = 1$, $q = 0$, $r = 1$. We proceed from the leaves towards the root, similarly to evaluating an arithmetic expression. The proposition φ is *true* under assignment (a) and *false* under assignment (b). See Figure 1.2.

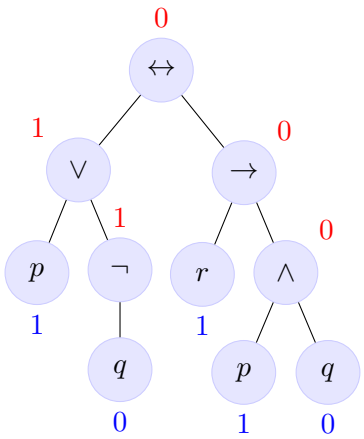
Logical connectives in the inner nodes are evaluated according to their *truth tables*, see Table 1.1.[10]

[9] Infinite theories are useful, for example, for describing the development of a system over (discrete) time steps $t = 0, 1, 2, \ldots$. The empty theory is not useful for anything, but it would be awkward to formulate statements about logic if theories had to be non-empty.

[10] Let us recall once again that disjunction is not exclusive, i.e., $p \vee q$ is true even if both p and q are true, and that implication is purely logical, i.e., $p \rightarrow q$ is true whenever p is false.



(a) $p = 0$, $q = 0$, $r = 0$



(b) $p = 1$, $q = 0$, $r = 1$

Figure 1.2: Truth value of a proposition

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

Table 1.1: Truth tables of logical connectives.

1.2.2 Propositions and Boolean Functions

To formalize the truth value of a proposition, we first look at the relationship between propositions and Boolean functions.

A *Boolean function* is a function $f\colon \{0,1\}^n \to \{0,1\}$, meaning that the input is an n-tuple of zeros and ones, and the output is 0 or 1. Each logical connective represents a Boolean function. In the case of negation, it is a unary function $f_\neg(x) = 1 - x$, while the other logical connectives correspond to binary functions described in Table 1.2.

Definition 1.2.2 (Truth Function)**.** The *truth function* of a proposition φ in a finite language $\mathbb{P}$ is the function $f_{\varphi,\mathbb{P}}\colon \{0,1\}^{|\mathbb{P}|} \to \{0,1\}$ defined inductively:

- If φ is the i-th atomic proposition from $\mathbb{P}$, then $f_{\varphi,\mathbb{P}}(x_0, \ldots, x_{n-1}) = x_i$,
- If $\varphi = (\neg\varphi')$, then

$$f_{\varphi,\mathbb{P}}(x_0, \ldots, x_{n-1}) = f_\neg(f_{\varphi',\mathbb{P}}(x_0, \ldots, x_{n-1})),$$

$f_\wedge(x,y)$:

	0	1
0	0	0
1	0	1

$f_\vee(x,y)$:

	0	1
0	0	1
1	1	1

$f_\to(x,y)$:

	0	1
0	1	1
1	0	1

$f_\leftrightarrow(x,y)$:

	0	1
0	1	0
1	0	1

Table 1.2: Boolean functions of logical connectives

- If $(\varphi' \square \varphi'')$ where $\square \in \{\wedge, \vee, \to, \leftrightarrow\}$, then

$$f_{\varphi,\mathbb{P}}(x_0, \ldots, x_{n-1}) = f_{\square}(f_{\varphi',\mathbb{P}}(x_0, \ldots, x_{n-1}), f_{\varphi'',\mathbb{P}}(x_0, \ldots, x_{n-1})).$$

Example 1.2.3. Let us calculate the truth function of the proposition $\varphi = ((p \vee (\neg q)) \leftrightarrow (r \to (p \wedge q)))$ in the language $\mathbb{P}' = \{p, q, r, s\}$:

$$f_{\varphi,\mathbb{P}'}(x_0, x_1, x_2, x_3) = f_{\leftrightarrow}(f_{\vee}(x_0, f_{\neg}(x_1)), f_{\to}(x_2, f_{\wedge}(x_0, x_1)))$$

Truth value of the proposition φ for the truth assignment $p = 1$, $q = 0$, $r = 1$, $s = 1$ is calculated as follows (compare with Figure 1.2(b)):

$$\begin{aligned} f_{\varphi,\mathbb{P}'}(1,0,1,1) &= f_{\leftrightarrow}(f_{\vee}(1, f_{\neg}(0)), f_{\to}(1, f_{\wedge}(1,0))) \\ &= f_{\leftrightarrow}(f_{\vee}(1,1), f_{\to}(1,0)) \\ &= f_{\leftrightarrow}(1,0) \\ &= 0 \end{aligned}$$

Observation 1.2.4. *The truth function of a proposition φ over $\mathbb{P}$ depends only on the variables corresponding to the atomic propositions in* $\mathrm{Var}(\varphi) \subseteq \mathbb{P}$.

Thus, even if we have a proposition φ in an *infinite* language $\mathbb{P}$, we can restrict ourselves to the language $\mathrm{Var}(\varphi)$ (which is finite) and consider the truth function over this language.

1.2.3 Models

A given truth assignment of propositional variables is a representation of the 'real world' (system) in our chosen 'formal world,' hence it is also called a *model*.

Definition 1.2.5 (Model of a Language)**.** A *model* of a language $\mathbb{P}$ is any truth assignment $v\colon \mathbb{P} \to \{0,1\}$. The *set of (all) models of a language* $\mathbb{P}$ is denoted by $\mathrm{M}_{\mathbb{P}}$:

$$\mathrm{M}_{\mathbb{P}} = \{v \mid v\colon \mathbb{P} \to \{0,1\}\} = \{0,1\}^{\mathbb{P}}$$

Models will be denoted by letters v, u, w, etc. (v from the word 'valuation'). A model of a language is therefore a function, formally a set of pairs (input, output). For example, for the language $\mathbb{P} = \{p, q, r\}$ and the truth assignment where p is true, q false, and r true, we have the model

$$v = \{(p,1), (q,0), (r,1)\}.$$

For simplicity, we will write it as $v = (1,0,1)$. For the language $\mathbb{P} = \{p, q, r\}$, we have $2^3 = 8$ models:

$$\mathrm{M}_{\mathbb{P}} = \{(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)\}$$

Remark 1.2.6. Formally speaking, we identify the set $\{0,1\}^{\mathbb{P}}$ with the set $\{0,1\}^{|\mathbb{P}|}$ using the ordering ι of the language $\mathbb{P}$ (see Remark 1.1.1). Specifically, instead of the element $v = \{(p,1), (q,0), (r,1)\} \in \{0,1\}^{\mathbb{P}}$, we write $(1,0,1) = (v \circ \iota)(0,1,2) = (v(\iota(0)), v(\iota(1)), v(\iota(2))) \in \{0,1\}^{|\mathbb{P}|}$ (where we allow the functions v, ι to act 'component-wise').[11] If this seems confusing, imagine the model v as a set of atomic propositions that are true, i.e., $\{p, r\} \subseteq \mathbb{P}$, our notation $v = (1,0,1)$ is then the characteristic vector of this set. This identification will be used henceforth without further notice.

[11] Alternatively, we could require (at least for countable languages) that the language *be* $\mathbb{P} = \{0, 1, 2, \ldots\}$ and use symbols p_0, p_1, p, q, r only for readability.

1.2.4 Validity

We are now ready to define the key concept of logic, *validity* of a proposition in a given model. Informally, a proposition is valid in a model (i.e., under a specific truth assignment of atomic propositions) if its truth value, as calculated in Example 1.2.1, equals 1. In the formal definition, we will use the truth function of the proposition (Definition 1.2.2).[12]

Definition 1.2.7 (Validity of a Proposition in a Model, Model of a Proposition)**.** Given a proposition φ in a language $\mathbb{P}$ and a model $v \in \mathrm{M}_{\mathbb{P}}$, if $f_{\varphi,\mathbb{P}}(v) = 1$, we say that the proposition φ is *valid* in the model v, v is a *model* of φ, and we write $v \models \varphi$. The set of all models of the proposition φ is denoted by $\mathrm{M}_{\mathbb{P}}(\varphi)$.

Models of a language that are not models of φ will sometimes be called *non-models* of φ. They form the complement of the set of models of φ. Using the standard notation for function inverse, we can write:

$$\mathrm{M}_{\mathbb{P}}(\varphi) = \{v \in \mathrm{M}_{\mathbb{P}} \mid v \models \varphi\} = f^{-1}_{\varphi,\mathbb{P}}[1]$$
$$\overline{\mathrm{M}_{\mathbb{P}}(\varphi)} = \mathrm{M}_{\mathbb{P}} \setminus \mathrm{M}_{\mathbb{P}}(\varphi) = \{v \in \mathrm{M}_{\mathbb{P}} \mid v \not\models \varphi\} = f^{-1}_{\varphi,\mathbb{P}}[0]$$

If the language is clear from the context, we can simply write $\mathrm{M}(\varphi)$. We must be really sure, though: for example, in the language $\mathbb{P} = \{p, q\}$ we have

$$\mathrm{M}_{\{p,q\}}(p \to q) = \{(0,0), (0,1), (1,1)\},$$

while in the language $\mathbb{P}' = \{p, q, r\}$ we would have

$$\mathrm{M}_{\mathbb{P}'}(p \to q) = \{(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,1,0), (1,1,1)\}.$$

Definition 1.2.8 (Validity of a Theory, Model of a Theory)**.** Let T be a theory in a language $\mathbb{P}$. The theory T is *valid* in a model v if every axiom $\varphi \in T$ is valid in v. In this case, we also say that v is a *model* of T, and we write $v \models T$. The set of all models of the theory T in a language $\mathbb{P}$ is denoted by $\mathrm{M}_{\mathbb{P}}(T)$.

When dealing with a finite theory, or adding a finite number of new axioms to a theory, we will use the following simplified notation:

- $\mathrm{M}_{\mathbb{P}}(\varphi_1, \varphi_2, \ldots, \varphi_n)$ instead of $\mathrm{M}_{\mathbb{P}}(\{\varphi_1, \varphi_2, \ldots, \varphi_n\})$,
- $\mathrm{M}_{\mathbb{P}}(T, \varphi)$ instead of $\mathrm{M}_{\mathbb{P}}(T \cup \{\varphi\})$.

Note that $\mathrm{M}_{\mathbb{P}}(T, \varphi) = \mathrm{M}_{\mathbb{P}}(T) \cap \mathrm{M}_{\mathbb{P}}(\varphi)$, $\mathrm{M}_{\mathbb{P}}(T) = \bigcap_{\varphi \in T} \mathrm{M}_{\mathbb{P}}(\varphi)$, and for a finite theory (similarly for countable theories), we have

$$\mathrm{M}_{\mathbb{P}}(\varphi_1) \supseteq \mathrm{M}_{\mathbb{P}}(\varphi_1, \varphi_2) \supseteq \mathrm{M}_{\mathbb{P}}(\varphi_1, \varphi_2, \varphi_3) \supseteq \cdots \supseteq \mathrm{M}_{\mathbb{P}}(\varphi_1, \varphi_2, \ldots, \varphi_n).$$

We can use this when finding models by brute force.

Example 1.2.9. We can find models of the theory $T = \{p \vee q \vee r, q \to r, \neg r\}$ (in the language $\mathbb{P} = \{p, q, r\}$) as follows. First, we find the models of the proposition $\neg r$:

$$\mathrm{M}_{\mathbb{P}}(r) = \{(x, y, 0) \mid x, y \in \{0, 1\}\} = \{(0,0,0), (0,1,0), (1,0,0), (1,1,0)\},$$

then we determine in which of these models the proposition $q \to r$ is valid:

[12] For *validity*, we use the symbol $\models$, which we read as 'satisfies' or 'models', in LaTeX as \models.

- $(0,0,0) \models q \to r$,
- $(0,1,0) \not\models q \to r$,
- $(1,0,0) \models q \to r$,
- $(1,1,0) \not\models q \to r$,

Thus $\mathrm{M}_{\mathbb{P}}(r, q \to r) = \{(0,0,0),(1,0,0)\}$. The proposition $p \vee q \vee r$ is valid only in the second of these models, so we get

$$\mathrm{M}_{\mathbb{P}}(r, q \to r, p \vee q \vee r) = \mathrm{M}_{\mathbb{P}}(T) = \{(1,0,0)\}.$$

This procedure is more efficient than determining the sets of models of the individual axioms and taking their intersection. (But much less efficient than using a formal proof system, such as the tableau method, which we will see later.)

1.2.5 Additional semantic notions

Following the notion of validity, we will use several other notions. For some properties, several different names are in use, depending on the context in which the property is discussed.

Definition 1.2.10 (Semantic notions)**.** We say that a proposition φ (in a language $\mathbb{P}$) is

- *true*, *tautology*, *valid (in logic/logically)*, and we write $\models \varphi$, if it is valid in every model (of the language $\mathbb{P}$), $\mathrm{M}_{\mathbb{P}}(\varphi) = \mathrm{M}_{\mathbb{P}}$,
- *false*, *contradictory*, if it has no model, $\mathrm{M}_{\mathbb{P}}(\varphi) = \emptyset$,[13]
- *independent*, if it is valid in some model, and it is not valid in some other model, i.e., it is neither true nor false, $\emptyset \subsetneq \mathrm{M}_{\mathbb{P}}(\varphi) \subsetneq \mathrm{M}_{\mathbb{P}}$,
- *satisfiable*, if it has some model, i.e., it is not false, $\mathrm{M}_{\mathbb{P}}(\varphi) \neq \emptyset$.

Furthermore, we say that propositions φ, ψ (in the same language $\mathbb{P}$) are *(logically) equivalent*, we write $\varphi \sim \psi$, if they have the same models, i.e.,

$$\varphi \sim \psi \text{ if and only if } \mathrm{M}_{\mathbb{P}}(\varphi) = \mathrm{M}_{\mathbb{P}}(\psi).$$

Example 1.2.11. For example, the following hold:

- propositions $\top$, $p \vee q \leftrightarrow q \vee p$ are true,
- propositions $\bot$, $(p \vee q) \wedge (p \vee \neg q) \wedge \neg p$ are false,
- propositions $p, p \wedge q$ are independent, and also satisfiable, and
- the following propositions are equivalent:
  - $p \sim p \vee p \sim p \vee p \vee p$,
  - $p \to q \sim \neg p \vee q$,

[13] Note that being *false* is not the same as not being *true*!

– $\neg p \to (p \to q) \sim \top$.

The notions from Definition 1.2.10 can also be relativized with respect to a given theory. This means that we restrict the individual definitions to the models of this theory:

Definition 1.2.12 (Semantic Notions Relative to a Theory)**.** Let T be a theory in a language $\mathbb{P}$. We say that a proposition φ in the language $\mathbb{P}$ is

- *true in* T, a *consequence of* T, *valid in* T, and we write $T \models \varphi$, if φ is valid in every model of the theory T, i.e., $\mathrm{M}_{\mathbb{P}}(T) \subseteq \mathrm{M}_{\mathbb{P}}(\varphi)$,
- *false in* T, *contradictory in* T, if it is not valid in any model of T, i.e., $\mathrm{M}_{\mathbb{P}}(\varphi) \cap \mathrm{M}_{\mathbb{P}}(T) = \mathrm{M}_{\mathbb{P}}(T, \varphi) = \emptyset$.
- *independent in* T, if it is valid in some model of T, and not valid in some other model of T, i.e., it is neither true in T nor false in T, $\emptyset \subsetneq \mathrm{M}_{\mathbb{P}}(T, \varphi) \subsetneq \mathrm{M}_{\mathbb{P}}(T)$,
- *satisfiable in* T, *consistent with* T, if it is valid in some model of T, i.e., it is not false in T, $\mathrm{M}_{\mathbb{P}}(T, \varphi) \neq \emptyset$.

And we say that propositions φ, ψ (in the same language $\mathbb{P}$) are *equivalent in* T, T*-equivalent*, we write $\varphi \sim_T \psi$ if they hold in the same models of T, i.e.,

$$\varphi \sim_T \psi \text{ if and only if } \mathrm{M}_{\mathbb{P}}(T, \varphi) = \mathrm{M}_{\mathbb{P}}(T, \psi).$$

Note that for the empty theory $T = \emptyset$, we have $\mathrm{M}_{\mathbb{P}}(T) = \mathrm{M}_{\mathbb{P}}$ and the above concepts for T coincide with the original ones. Again, we illustrate the concepts with several examples:

Example 1.2.13. Let $T = \{p \vee q, \neg r\}$. The following hold:

- propositions $q \vee p$, $\neg p \vee \neg q \vee \neg r$ are true in T,
- the proposition $(\neg p \wedge \neg q) \vee r$ is false in T,
- propositions $p \leftrightarrow q, p \wedge q$ are independent in T, and also satisfiable, and
- p and $p \vee r$ are T-equivalent, $p \sim_T p \vee r$ (but $p \not\sim p \vee r$).

1.2.6 Universality of Logical Connectives

In the language of propositional logic, we use the following logical connectives: $\neg, \wedge, \vee, \to, \leftrightarrow$. This is not the only possible choice; to build a fully-fledged logic, we could make do, for example, only with negation and implication,[14] or negation, conjunction, and disjunction.[15] And as we will see below, we could use other logical connectives as well. Our choice is the golden middle path between expressiveness on the one hand and succintness of syntactic definitions and proofs on the other.

What do we mean by saying that logic is fully-fledged? We say that a set of logical connectives S is *universal*[16] if any Boolean function f can be expressed as the truth function $f_{\varphi,\mathbb{P}}$ of some proposition φ built from the logical connectives in S (where $|\mathbb{P}| = n$ if f is

[14] Negation is needed to describe the state of a system, and implication to describe behavior over time.
[15] These are sufficient to build logical circuits.
[16] Some people would say *[functionally] complete*.

an n-ary function). Equivalently, for any finite language $\mathbb{P}$ (say n-element) and any set of models $M \subseteq \mathrm{M}_{\mathbb{P}}$, there must exist a proposition φ such that $\mathrm{M}_{\mathbb{P}}(\varphi) = M$. (The equivalence of these two statements follows from the fact that if we have a Boolean function f and choose $M = f^{-1}[1]$, then $f_{\varphi,\mathbb{P}} = f$ if and only if $\mathrm{M}_{\mathbb{P}}(\varphi) = M$.)

Proposition 1.2.14. *The sets of logical connectives* $\{\neg, \wedge, \vee\}$ *and* $\{\neg, \to\}$ *are universal.*

Proof. Let us have a function $f\colon \{0,1\}^n \to \{0,1\}$, or equivalently a set of models $M = f^{-1}[1] \subseteq \{0,1\}^n$. Our language will be $\mathbb{P} = \{p_1, \ldots, p_n\}$. If the set M contained only one model, say $v = (1,0,1,0)$, we could represent it with the proposition $\varphi_v = p_1 \wedge \neg p_2 \wedge p_3 \wedge \neg p_4$, which says 'I must be the model v.' For a general model v, we would write the proposition φ_v as follows:

$$\varphi_v = p_1^{v_1} \wedge p_2^{v_2} \wedge \cdots \wedge p_n^{v_n} = \bigwedge_{i=1}^{n} p_i^{v(p_i)} = \bigwedge_{p \in \mathbb{P}} p^{v(p)},$$

where we introduce the following useful notation: $p^{v(p)}$ is the proposition p if $v(p) = 1$, and the proposition $\neg p$ if $v(p) = 0$.

If the set M contains more models, we say 'I must be at least one of the models from M':

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

Clearly, $\mathrm{M}_{\mathbb{P}}(\varphi_M) = M$, or equivalently $f_{\varphi_M,\mathbb{P}} = f$. (If $M = \emptyset$, then by definition $\bigvee_{v \in M} \varphi_v = \bot$.)[17]

The universality of $\{\neg, \to\}$ follows from the universality of $\{\neg, \wedge, \vee\}$ and the fact that conjunction and disjunction can be expressed using negation and implication: $p \wedge q \sim \neg(p \to \neg q)$ and $p \vee q \sim \neg p \to q$. □

Remark 1.2.15. Note that in the construction of the proposition φ_M, it is crucial that the set M is finite (it has at most 2^n elements). If it were infinite, the symbol '$\bigvee_{v \in M}$' would mean 'disjunction' of infinitely many propositions, and thus the result would not be a finite string, i.e., 'φ_M' would not be a proposition. (If we have a countably infinite language $\mathbb{P}'$, then not every subset $M \subseteq \mathrm{M}_{\mathbb{P}'}$ can be represented by a proposition—there are uncountably many such subsets, while propositions are only countably many.)

What other logical connectives could we use? Nullary Boolean functions,[18] i.e. constants 0, 1, could be introduced as symbols TRUE and FALSE; we will suffice with propositions $\top, \bot$. There are four unary Boolean functions ($4 = 2^{2^1}$), but negation is the only 'interesting' one: the others are $f(x) = x$, $f(x) = 0$, and $f(x) = 1$. There are more interesting binary logical connectives that occur naturally, for example:

- NAND or *Sheffer's stroke*, sometimes denoted as $p \uparrow q$, where $p \uparrow q \sim \neg(p \wedge q)$,
- NOR or *Peirce's arrow*, sometimes denoted as $p \downarrow q$, where $p \downarrow q \sim \neg(p \vee q)$,
- XOR, or *exclusive-OR*, sometimes denoted as $\oplus$, where $p \oplus q \sim (p \vee q) \wedge \neg(p \wedge q)$, i.e. the sum of truth values modulo 2.

[17] Similar to how the sum of an empty set of summands equals 0.

[18] In formalizing mathematics or computer science, a function of arity 0 means it has no inputs, so the output cannot depend on the input and is constant. Formally, these are functions $f\colon \emptyset \to \{0,1\}$. If this is confusing, assume that functions must have an arity of at least 1, and instead of 'nullary function,' say 'constant.'

Exercise 1.2. Express $(p \oplus q) \oplus r$ using $\{\neg, \wedge, \vee\}$.

Exercise 1.3. Show that {NAND} and {NOR} are universal.

Exercise 1.4. Consider the ternary logical connective IFTE, where $IFTE(p, q, r)$ is satisfied if and only if 'if p then q else r'. Determine the truth table of this logical connective (i.e., the function f_{IFTE}) and show that $\{\text{TRUE}, \text{FALSE}, \text{IFTE}\}$ is universal.

1.3 Normal Forms

Let us recall that propositions are equivalent if they have the same set of models. For each proposition, there exist infinitely many equivalent propositions; it is often useful to express a proposition in a 'nice' (useful) 'shape', i.e., to find an equivalent proposition of that 'shape'. This concept of 'shape' in mathematics is called a *normal form*. We will introduce two most common normal forms: *conjunctive normal form (CNF)* and *disjunctive normal form (DNF)*.

The following terminology and notation are needed:

- A *literal* ℓ is either a propositional variable p or the negation of a propositional variable $\neg p$. For a propositional variable p, denote $p^0 = \neg p$ and $p^1 = p$. If ℓ is a literal, then $\bar{\ell}$ denotes the *opposite literal* to ℓ. If $\ell = p$ (a *positive literal*), then $\bar{\ell} = \neg p$; if $\ell = \neg p$ (a *negative literal*), then $\bar{\ell} = p$.
- A *clause* is a disjunction of literals $C = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_n$. A *unit clause* is a single literal ($n = 1$) and the *empty clause* ($n = 0$) is interpreted as $\bot$.
- A proposition is in *conjunctive normal form (CNF)* if it is a conjunction of clauses. The *empty CNF proposition* is $\top$.
- An *elementary conjunction* is a conjunction of literals $E = \ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_n$. A *unit elementary conjunction* is a single literal ($n = 1$). The *empty elementary conjunction* ($n = 0$) is $\top$.
- A proposition is in *disjunctive normal form (DNF)* if it is a disjunction of elementary conjunctions. The *empty DNF proposition* is $\bot$.

Example 1.3.1. The proposition $p \vee q \vee \neg r$ is in CNF (it is a single clause) as well as in DNF (it is a disjunction of unit elementary conjunctions). The proposition $(p \vee q) \wedge (p \vee \neg q) \wedge \neg p$ is in CNF, while the proposition $\neg p \vee (p \wedge q)$ is in DNF.

Example 1.3.2. The proposition φ_v from the proof of Proposition 1.2.14 is in CNF (it is a conjunction of unit clauses, i.e., literals) and also in DNF (it is a single elementary conjunction). The proposition φ_M is in DNF.

Observation 1.3.3. *Note that a proposition in CNF is a tautology if and only if each of its clauses contains a pair of opposite literals. Similarly, a proposition in DNF is satisfiable if and only if not every elementary conjunction contains a pair of opposite literals.*

1.3.1 On Duality

Note that if we interchange the values for truth and falsehood in propositional logic, i.e., 0 and 1, the truth table for negation remains the same, while conjunction becomes disjunction, and vice versa. This concept is called *duality*; we will see many examples of this in logic.

We have $\neg(p \wedge q) \sim (\neg p \vee \neg q)$, and from *duality*, we also know that $\neg(\neg p \vee \neg q) \sim (\neg\neg p \wedge \neg\neg q)$, from which we can easily deduce $\neg(p \vee q) \sim (\neg p \wedge \neg q)$.[19] More generally, n-ary Boolean functions f, g are *dual* to each other if $f(\neg x) = \neg g(x)$. If we have a proposition φ built from $\{\neg, \wedge, \vee\}$ and we interchange $\wedge$ and $\vee$, and negate the propositional variables (i.e., interchange literals with their opposite literals), we obtain a proposition $\psi \sim \neg\varphi$ (i.e., the models of φ are the non-models of ψ and vice versa), and the functions $f_{\varphi,\mathbb{P}}$ and $f_{\psi,\mathbb{P}}$ are dual to each other.

The notion of DNF is dual to the notion of CNF; 'is a tautology' is dual to 'is not satisfiable', thus the previous observation can be understood as an example of duality. For every statement in propositional logic, we obtain a *dual* statement 'for free', resulting from the interchange of $\wedge$ and $\vee$, truth and falsehood.

1.3.2 Conversion to Normal Forms

We have already encountered the disjunctive normal form in the proof of Proposition 1.2.14. The key part of the proof can be formulated as follows: 'If the language is finite, any set of models can be *axiomatized* by a proposition in DNF'. From duality, we also obtain axiomatization in CNF since the complement of a set of models is also a set of models:

Proposition 1.3.4. *Given a finite language $\mathbb{P}$ and any set of models $M \subseteq \mathrm{M}_\mathbb{P}$, there exists a proposition φ_{DNF} in DNF and a proposition φ_{CNF} in CNF such that $M = \mathrm{M}_\mathbb{P}(\varphi_{\mathrm{DNF}}) = \mathrm{M}_\mathbb{P}(\varphi_{\mathrm{CNF}})$. Specifically:*

$$\varphi_{\mathrm{DNF}} = \bigvee_{v \in M} \bigwedge_{p \in \mathbb{P}} p^{v(p)}$$

$$\varphi_{\mathrm{CNF}} = \bigwedge_{v \in \overline{M}} \bigvee_{p \in \mathbb{P}} \overline{p^{v(p)}} = \bigwedge_{v \notin M} \bigvee_{p \in \mathbb{P}} p^{1-v(p)}$$

Proof. For the proposition φ_{DNF}, see the proof of Proposition 1.2.14, where each elementary conjunction describes one model. The proposition φ_{CNF} is dual to the proposition φ'_{DNF} constructed for the complement $M' = \overline{M}$. Alternatively, we can prove it directly: the models of the clause $C_v = \bigvee_{p \in \mathbb{P}} p^{1-v(p)}$ are all models except v, $\mathrm{M}_C = \mathrm{M}_\mathbb{P} \setminus \{v\}$, so each clause in the conjunction excludes one non-model. □

Proposition 1.3.4 provides a method for converting a proposition to disjunctive or conjunctive normal form:

Example 1.3.5. Consider the proposition $\varphi = p \leftrightarrow (q \vee \neg r)$. First, we find the set of models: $M = \mathrm{M}(\varphi) = \{(0,0,1), (1,0,0), (1,1,0), (1,1,1)\}$. Then, we find the propositions φ_{DNF} and φ_{CNF} according to Proposition 1.3.4. Those have the same models as φ and are therefore equivalent with it.

We find the proposition φ_{DNF} by constructing an elementary conjunction for each model that enforces precisely that model:

$$\varphi_{\mathrm{DNF}} = (\neg p \wedge \neg q \wedge r) \vee (p \wedge \neg q \wedge \neg r) \vee (p \wedge q \wedge \neg r) \vee (p \wedge q \wedge r)$$

For constructing φ_{CNF}, we need the *non-models* of φ, $\overline{M} = \{(0,0,0), (0,1,0), (0,1,1), (1,0,1)\}$. Each clause excludes one non-model:

$$\varphi_{\mathrm{CNF}} = (p \vee q \vee r) \wedge (p \vee \neg q \vee r) \wedge (p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee \neg r)$$

[19] Since p, q are propositional variables, they can be assigned both values 0 and 1, thus we can interchange them with their opposite literals.

Corollary 1.3.6. *Every proposition (in any, even infinite, language $\mathbb{P}$) is equivalent to some proposition in CNF and some proposition in DNF.*

Proof. Even if the language $\mathbb{P}$ is infinite, the proposition φ contains only finitely many propositional variables, so we can use Proposition 1.3.4 for the language $\mathbb{P}' = \mathrm{Var}(\varphi)$ and the set of models $M = \mathrm{M}_{\mathbb{P}'}(\varphi)$. Since $M = \mathrm{M}_{\mathbb{P}'}(\varphi_{\mathrm{DNF}}) = \mathrm{M}_{\mathbb{P}'}(\varphi_{\mathrm{CNF}})$, we have $\varphi \sim \varphi_{\mathrm{DNF}} \sim \varphi_{\mathrm{CNF}}$. □

Exercise 1.5. Describe how one can easily generate models from a DNF proposition and non-models from a CNF proposition.

Remark 1.3.7. When can a *theory* be axiomatized by a proposition in DNF or CNF? Consider the language $\mathbb{P}' = \mathrm{Var}(T)$ (i.e., all propositional variables occurring in the axioms of T). If T in the language $\mathbb{P}'$ has finitely many models (i.e., $\mathrm{M}_{\mathbb{P}'}(T)$ is finite), we can construct a proposition in DNF, and if it has finitely many *non-models*, we can construct a proposition in CNF. Generally, however, not every theory can be axiomatized by a *single* proposition in CNF or DNF. We can always convert individual axioms to CNF (or DNF), and we can also axiomatize the theory using (potentially infinitely many) clauses.

This method of conversion to CNF or DNF requires knowledge of the set of models of the proposition, so it is quite inefficient. Additionally, the resulting normal form can be very long. We will now show another method.

Conversion Using Equivalent Transformations

We use the following observation: If we replace some subproposition ψ of a proposition φ with an equivalent proposition ψ', the resulting proposition φ' will also be equivalent to φ. We will first demonstrate the method on an example:

Example 1.3.8. Again, we will convert the proposition $\varphi = p \leftrightarrow (q \vee \neg r)$. First, we eliminate the equivalence, expressing it as a conjunction of two implications. In the next step, we remove the implications using the rule $\varphi \to \psi \sim \neg\varphi \vee \psi$:

$$\begin{aligned} p \leftrightarrow (q \vee \neg r) &\sim (p \to (q \vee \neg r)) \wedge ((q \vee \neg r) \to p) \\ &\sim (\neg p \vee q \vee \neg r) \wedge (\neg(q \vee \neg r) \vee p) \end{aligned}$$

Now, imagine the tree of the proposition; in the next step, we want to push the negations as low as possible in the tree, just above the leaves: we use $\neg(q \vee \neg r) \sim \neg q \wedge \neg\neg r$ and eliminate the double negation $\neg\neg r \sim r$. We obtain the proposition

$$(\neg p \vee q \vee \neg r) \wedge ((\neg q \wedge r) \vee p)$$

At this point, we leave the literals untouched and apply the distributivity of $\wedge$ over $\vee$, or vice versa, depending on whether we want DNF or CNF. For conversion to CNF, we use the transformation $(\neg q \wedge r) \vee p \sim (\neg q \vee p) \wedge (r \vee p)$, which pushes the $\vee$ symbol lower in the tree. (Draw the tree!) This gives us a proposition in CNF; for better readability, we sort the literals in the clauses:

$$(\neg p \vee q \vee \neg r) \wedge (p \vee \neg q) \wedge (p \vee r)$$

For conversion to DNF, we proceed similarly by repeatedly applying distributivity. Here, we start from the CNF form and combine each literal from the first clause with each literal from the second and each literal from the third clause. Note that the same literal does not need to

be repeated twice in the elementary conjunction, and if the elementary conjunction contains a pair of opposite literals, it is contradictory and can be omitted in the DNF. We can also omit an elementary conjunction E if we have another elementary conjunction E' such that E' contains all the literals contained in E, e.g., $E = (p \wedge \neg r)$ and $E' = (p \wedge q \wedge \neg r)$. (Think about why, and formulate the dual simplification when converting to CNF.) The resulting proposition in DNF is:

$$(\neg p \wedge \neg q \wedge r) \vee (p \wedge q \wedge r) \vee (p \wedge \neg r)$$

We now list all equivalent transformations needed for the method. The proof that every proposition can be converted to DNF and CNF can be easily carried out by induction on the structure of the proposition (depth of the proposition tree).

- Implications and equivalences:

$$\varphi \to \psi \sim \neg\varphi \vee \psi$$
$$\varphi \leftrightarrow \psi \sim (\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi)$$

- Negations:

$$\neg(\varphi \wedge \psi) \sim \neg\varphi \vee \neg\psi$$
$$\neg(\varphi \vee \psi) \sim \neg\varphi \wedge \neg\psi$$
$$\neg\neg\varphi \sim \varphi$$

- Conjunctions (conversion to DNF):

$$\varphi \wedge (\psi \vee \chi) \sim (\varphi \wedge \psi) \vee (\varphi \wedge \chi)$$
$$(\varphi \vee \psi) \wedge \chi \sim (\varphi \wedge \chi) \vee (\psi \wedge \chi)$$

- Disjunctions (conversion to CNF):

$$\varphi \vee (\psi \wedge \chi) \sim (\varphi \vee \psi) \wedge (\varphi \vee \chi)$$
$$(\varphi \wedge \psi) \vee \chi \sim (\varphi \vee \chi) \wedge (\psi \vee \chi)$$

As we will see in the next chapter, CNF is much more important in practice than DNF (even though they are dual concepts). For describing a real system, it is more natural to express it as a conjunction of many simpler properties than as a single very long disjunction. There are many other forms of representing Boolean functions. Similar to data structures, we choose the appropriate form of representation depending on the operations we need to perform on the function.[20]

[20] See, for example, the course NAIL031 Representations of Boolean Functions.