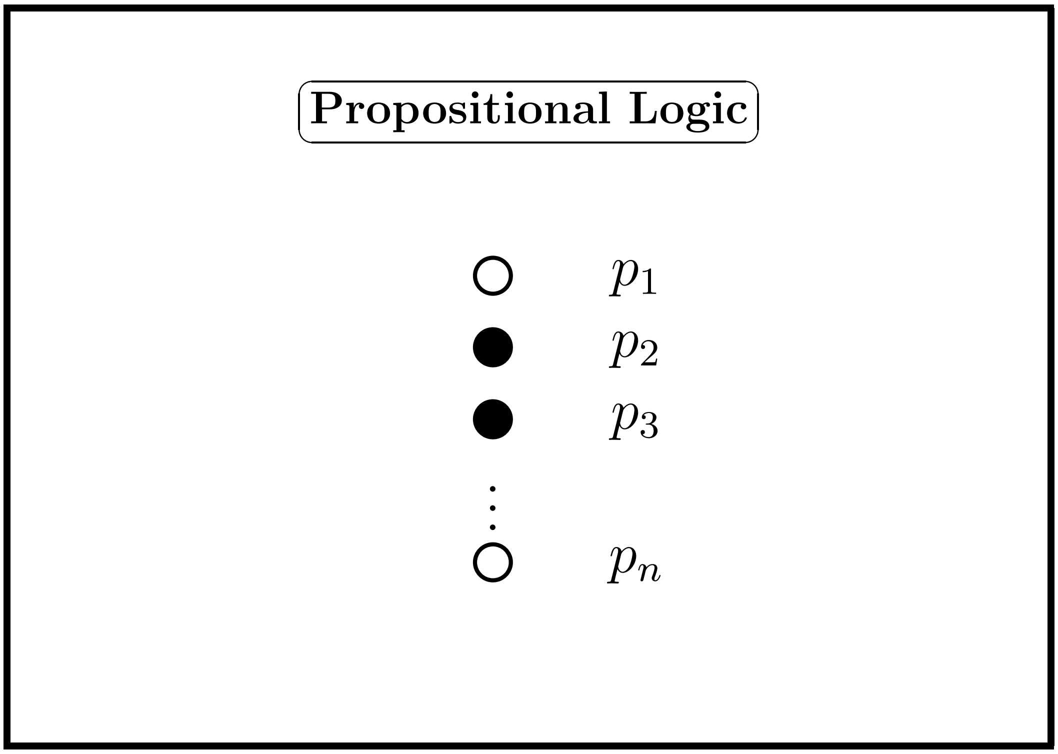
1 Propositional Logic

Propositional logic is the logic of individual bits.



1.1 Syntax and Semantics

So, how do we describe such a situation? We can have a statement for every bit, saying that it is set. The rest is just composing statements, say, by "and" and "or"; we also allow $\top$ and $\bot$ as "logical constants", that is, as statements that are always true or always false, respectively.

In the following definition note the difference the language we use to speak *about formulae* and the actual syntax of the *formulae themselves*. The symbols "φ" and "ψ" are used to *denote* arbitrary formulae, but are not part of the syntax themselves (our formulae do not contain Greek letters). The symbols "(", ")", "$\wedge$", "$\vee$", "$\bot$", and "$\top$", however, *are* part of the syntax.

Syntax of Propositional Logic

The set $\mathsf{PL}[p_1, \ldots, p_n]$ of *propositional formulae over* $p_1, \ldots, p_n$ is freely generated as follows.

- $\top$, $\bot$, and all $p_i \in \{p_1, \ldots, p_n\}$ are propositional formulae *(so called "atomic formulae")*.
- If φ is a propositional formula, then so is $\neg\varphi$.
- If φ and ψ are propositional formulae, then so are $(\varphi \wedge \psi)$ and $(\varphi \vee \psi)$.

Here the "freely generated" (sometimes also called "inductively defined") means that the set $\mathsf{PL}[p_1, \ldots, p_n]$ is the *smallest* set with the given closure properties and, moreover, for every term we can read off—in a unique way!—in which way it is generated from these rules.

Note that we added enough parentheses so that for every string there is at most one way how it could have been generated by the above rules. This allows us to take these strings as formulae and still consider them freely generated.

Such definitions are quite common in many areas of logic and theoretical computer science. This definition principle gives us immediately a principle to define or proof something for *all* formulae. Note that to prove something for all formulae, it is sufficient to know that all formulae can be obtained by one of the rules; for defining a function on formulae, however, we also need that formulae are generated in a unique way.

Let's show this principle at a simple example. Our language $\mathsf{PL}[p_1, \ldots, p_n]$ of propositional logic is parametrised by the set $\{p_1, \ldots, p_n\}$ of allowed propositional variables. These symbols may be used in formulae of $\mathsf{PL}[p_1, \ldots, p_n]$. However, they do not have to be used. For $\varphi \in \mathsf{PL}[p_1, \ldots, p_n]$, the set $\mathsf{fv}(\varphi)$ contains the variables that actually *are* used. We will see that only the value of those bit matter for a formula to be true or false.

$$
\begin{array}{lcl}
\mathsf{fv}(\top) = \mathsf{fv}(\bot) & = & \emptyset \\
\mathsf{fv}(p_i) & = & \{p_i\} \\
\mathsf{fv}(\neg\varphi) & = & \mathsf{fv}(\varphi) \\
\mathsf{fv}((\varphi \wedge \psi)) = \mathsf{fv}((\varphi \vee \psi)) & = & \mathsf{fv}(\varphi) \cup \mathsf{fv}(\psi)
\end{array}
$$

As mentioned, we regard propositional logic as the logic of bit strings. So our structures for $\mathsf{PL}[p_1, \ldots, p_n]$ are n-tuples $\underline{\mathrm{a}} = (a_1, \ldots, a_n) \in \{0,1\}^n$. The variable p_i is interpreted as a_i. We extend this (again, using the free generation of the formulae) to a value $\varphi[\underline{\mathrm{a}}] \in \{0,1\}$ by the following truth table.

Evaluation of a Boolean Formula

For $\varphi \in \mathsf{PL}[p_1, \ldots, p_n]$ and $\underline{\mathrm{a}} = (a_1, \ldots, a_n) \in \{0,1\}^n$ we define $\varphi[\underline{\mathrm{a}}] \in \{0,1\}$ as follows.

- $\top[\underline{\mathrm{a}}] = 1$, $\bot[\underline{\mathrm{a}}] = 0$, $p_i[\underline{\mathrm{a}}] = a_i$
- $(\neg\varphi)[\underline{\mathrm{a}}] = \urcorner(\varphi[\underline{\mathrm{a}}])$
- $(\varphi \wedge \psi)[\underline{\mathrm{a}}] = (\varphi[\underline{\mathrm{a}}]) \mathbin{\text{⩓}} (\psi[\underline{\mathrm{a}}])$,
  $(\varphi \vee \psi)[\underline{\mathrm{a}}] = (\varphi[\underline{\mathrm{a}}]) \mathbin{\text{⩔}} (\psi[\underline{\mathrm{a}}])$

The functions $\urcorner, \text{⩓}, \text{⩔}$ are given by

x	y	$\urcorner y$	$x \mathbin{\text{⩓}} y$	$x \mathbin{\text{⩔}} y$
0	0	1	0	0
0	1	0	0	1
1	0		0	1
1	1		1	1

Note the difference between the *symbols* $\neg, \wedge, \vee$ and the *functions* $\urcorner\colon \{0,1\} \to \{0,1\}$, $\text{⩓}, \text{⩔}\colon \{0,1\} \times \{0,1\} \to \{0,1\}$. However, from the context is usually clear whether a function or a name of a function is needed in some particular place. So we follow the usual laziness and often write $\neg, \wedge, \vee$ as a shorthand for $\urcorner, \text{⩓}, \text{⩔}$. Nevertheless, you should always be able to tell, whether a symbol is meant, or whether that shorthand is used in a particular place.

Of course, when evaluating a formula, only those variables mentioned in the formula matter. The lemma expressing this fact is usually known as "Coincidence Lemma". We'll prove it as an exercise to see how one can argue on the inductive structure of an inductively defined object.

Lemma ("Coincidence Lemma").
$\varphi \in \mathsf{PL}[p_1, \ldots, p_n]$, $\underline{\mathrm{a}} = (a_1, \ldots, a_n) \in \{0,1\}^n$, $\underline{\mathrm{b}} = (b_1, \ldots, b_n) \in \{0,1\}^n$.
Assume $a_i = b_i$ for all i s.t. $p_i \in \mathsf{fv}(\varphi)$.
Then $\varphi[a_1, \ldots, a_n] = \varphi[b_1, \ldots, b_n]$.

Proof.
If $\varphi = p_i$, then $p_i \in \mathsf{fv}(\varphi)$. Hence $\varphi[a_1, \ldots, a_n] = p_i[a_1, \ldots, a_n] = a_i = b_i = \varphi[b_1, \ldots, b_n]$.
If $\varphi = \psi \wedge \psi'$ then $\varphi[a_1, \ldots, a_n] = (\psi \wedge \psi')[a_1, \ldots, a_n] = \psi[a_1, \ldots, a_n] \wedge \wedge\psi[a_1, \ldots, a_n]' \overset{IH}{=} \psi[b_1, \ldots, b_n] \wedge \wedge\psi[b_1, \ldots, b_n]' = \varphi[b_1, \ldots, b_n]$.
...

Logic is usually formulated in terms of the model relation $\models$ that says whether a formula holds in a particular structure. Even though our interpretation is formulated in terms of 0 and 1, it is obvious that we think of 1 as being true and of 0 as being false. This motivates writing $\underline{\mathrm{a}} \models \varphi$ for $\varphi[\underline{\mathrm{a}}] = 1$.

Model Relation for Propositional Logic

Writing $\underline{a} \models \varphi$ for $\varphi[\underline{a}] = 1$ we obtain the following.

- $\underline{a} \models \top$ always holds and $\underline{a} \models \bot$ never holds
- $\underline{a} \models \neg\varphi$ holds iff $\underline{a} \models \varphi$ does *not* hold
- $\underline{a} \models \varphi \wedge \psi$ holds if $\underline{a} \models \varphi$ *and* $\underline{a} \models \psi$ both hold.
  $\underline{a} \models \varphi \vee \psi$ holds if $\underline{a} \models \varphi$ holds *or* $\underline{a} \models \psi$ holds.

1.2 Boolean Functions

We have seen that propositional logic can be used to define functions $\{0,1\}^n \to \{0,1\}$. An obvious question arising is "Can we define all functions in that way?".

The answer is "Yes" and we will work towards the following Theorem.

$\boxed{\textbf{Expressive Completeness}}$

Theorem. For every $f\colon \{0,1\}^n \to \{0,1\}$ there is some $\varphi \in \mathsf{PL}[p_1,\dots,p_n]$ such that for all $\underline{\mathrm{a}} \in \{0,1\}^n$ we have $f(\underline{\mathrm{a}}) = \varphi[\underline{\mathrm{a}}]$.

To start our task, we build up things up from simpler tasks. How to represent that a variable has a particular value? How to build a function that is 1 at precisely one point $\underline{\mathrm{a}}$? Once this is achieved, we use that there are only finitely many $\underline{\mathrm{a}} \in \{0,1\}^n$; so we can just enumerate the positions where f is 1.

Proof.
Write x^1 for x and write x^0 for $\neg x$.
Then $\underline{a} \models x_i^\delta$ iff $a_i = \delta$.

Define
$$\chi_{\underline{a}} = \bigwedge_i x_i^{a_i}$$
Then $\underline{b} \models \chi_{\underline{a}}$ iff $\underline{b} = \underline{a}$.

Finally set
$$\varphi = \bigvee_{f(\underline{a})=1} \chi_{\underline{a}}$$

Carefully inspecting the proof, we note that we have proven even more.

Expressive Completeness

Theorem. For every $f\colon \{0,1\}^n \to \{0,1\}$ there is some $\varphi \in \mathsf{PL}[p_1, \ldots, p_n]$ such that for all $\underline{a} \in \{0,1\}^n$ we have $f(\underline{a}) = \varphi[\underline{a}]$.

In fact, φ can be chosen to be of the form

$$\bigvee_j \bigwedge_i \xi_{ij} \qquad \text{with } \xi_{ij} \in \{x_i, \neg x_i\}$$

(in "disjunctive normal form")

In a similar way, we can show that every boolean function can be represented in conjunctive normal form. Working out the details will be (part of) your homework.

This completeness result also justifies why we didn't include more connectives in our language: having "and", "or" and "not" is enough. Nevertheless, there are quite a few abbreviations commonly used; the most important ones are "$\rightarrow$" and "$\leftrightarrow$".

$$\begin{aligned} \phi \rightarrow \psi \;&\equiv\; \neg\phi \vee \psi \\ \phi \leftrightarrow \psi \;&\equiv\; (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi) \end{aligned}$$

In fact, our choice of connectives is not minimal. A smaller number of connectives suffices to define all boolean functions.

Other Complete Sets of Connectives

- $\wedge$, $\neg$. *Indeed,* $x \vee y = \neg((\neg x) \wedge (\neg y))$.
- $\vee$, $\neg$
- `nand` where

x	y	x `nand` y
0	0	1
0	1	1
1	0	1
1	1	0

Indeed, $\neg x = x \texttt{ nand } x$ *and* $x \wedge y = \neg(x \texttt{ nand } y)$.

There are more complete set of connectives; one will be shown in your homework.

We have seen that we can express every boolean function by a propositional formula. Our construction yielded formulae in disjunctive normal form of size $\mathcal{O}(n \cdot 2^n)$. We now can ask, whether we can do better by using formulae not in disjunctive normal form. The answer is, that we cannot get rid of the exponential growths. More precisely, we have the following theorem.