# 2 Propositional Logic

Propositional logic

- logic of truth values
- decidable (but NP-complete)
- can be used to describe functions over a finite domain
- important for hardware applications (e. g., model checking)

## 2.1 Syntax

- propositional variables
- logical symbols
  $\Rightarrow$ Boolean combinations

### Propositional Variables

Let $\Pi$ be a set of *propositional variables*.

We use letters $P$, $Q$, $R$, $S$, to denote propositional variables.

### Propositional Formulas

$F_\Pi$ is the set of propositional formulas over $\Pi$ defined as follows:

$$
\begin{array}{rcll}
F, G, H & ::= & \bot & \text{(falsum)} \\
& | & \top & \text{(verum)} \\
& | & P, \quad P \in \Pi & \text{(atomic formula)} \\
& | & \neg F & \text{(negation)} \\
& | & (F \wedge G) & \text{(conjunction)} \\
& | & (F \vee G) & \text{(disjunction)} \\
& | & (F \rightarrow G) & \text{(implication)} \\
& | & (F \leftrightarrow G) & \text{(equivalence)}
\end{array}
$$

### Notational Conventions

- We may omit brackets according to the following rules:
  - $\neg \quad >_p \quad \vee \quad >_p \quad \wedge \quad >_p \quad \rightarrow \quad >_p \quad \leftrightarrow$ (binding precedences)
  - $\vee$ and $\wedge$ are left-associative,
    i. e., $F \vee G \vee H$ means $(F \vee G) \vee H$.
  - $\rightarrow$ is right-associative,
    i. e., $F \rightarrow G \rightarrow H$ means $F \rightarrow (G \rightarrow H)$.

## 2.2 Semantics

In *classical logic* (dating back to Aristoteles) there are "only" two truth values "true" and "false" which we shall denote, respectively, by 1 and 0.

There are *multi-valued logics* having more than two truth values.

### Valuations

A propositional variable has no intrinsic meaning. The meaning of a propositional variable has to be defined by a valuation.

A $\Pi$-*valuation* is a map

$$\mathcal{A} : \Pi \rightarrow \{0, 1\}.$$

where $\{0, 1\}$ is the set of *truth values*.

### Truth Value of a Formula in $\mathcal{A}$

Given a $\Pi$-valuation $\mathcal{A}$, the function $\mathcal{A}^* : \Sigma$-formulas $\rightarrow \{0, 1\}$ is defined inductively over the structure of $F$ as follows:

$$\begin{aligned}
\mathcal{A}^*(\bot) &= 0 \\
\mathcal{A}^*(\top) &= 1 \\
\mathcal{A}^*(P) &= \mathcal{A}(P) \\
\mathcal{A}^*(\neg F) &= \mathsf{B}_\neg(\mathcal{A}^*(F)) \\
\mathcal{A}^*(F \,\rho\, G) &= \mathsf{B}_\rho(\mathcal{A}^*(F), \mathcal{A}^*(G)) \quad \text{for} \quad \rho \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}
\end{aligned}$$

where $\mathsf{B}_\rho$ is the Boolean function associated with $\rho$ defined by the usual truth table.

For simplicity, we write $\mathcal{A}$ instead of $\mathcal{A}^*$.

We also write $\rho$ instead of $\mathsf{B}_\rho$, i. e., we use the same notation for a logical symbol and for its meaning (but remember that formally these are different things.)

## 2.3 Models, Validity, and Satisfiability

$F$ is *valid* in $\mathcal{A}$ ($\mathcal{A}$ is a *model* of $F$; $F$ holds under $\mathcal{A}$):

$$\mathcal{A} \models F \;:\Leftrightarrow\; \mathcal{A}(F) = 1$$

$F$ is *valid* (or is a *tautology*):

$$\models F \;:\Leftrightarrow\; \mathcal{A} \models F \text{ for all } \Pi\text{-valuations } \mathcal{A}$$

$F$ is called *satisfiable* if there exists an $\mathcal{A}$ such that $\mathcal{A} \models F$. Otherwise $F$ is called *unsatisfiable* (or *contradictory*).

### Entailment and Equivalence

*$F$ entails (implies) $G$* (or *$G$ is a consequence of $F$*), written $F \models G$, if for all $\Pi$-valuations $\mathcal{A}$ we have $\mathcal{A} \models F \;\Rightarrow\; \mathcal{A} \models G$.

$F$ and $G$ are called *equivalent*, written $F \models\!\!\models G$, if for all $\Pi$-valuations $\mathcal{A}$ we have $\mathcal{A} \models F \;\Leftrightarrow\; \mathcal{A} \models G$.

**Proposition 2.1** *$F \models G$ if and only if $\models (F \to G)$.*

**Proof.** ($\Rightarrow$) Suppose that $F$ entails $G$. Let $\mathcal{A}$ be an arbitrary $\Pi$-valuation. We have to show that $\mathcal{A} \models F \to G$. If $\mathcal{A}(F) = 1$, then $\mathcal{A}(G) = 1$ (since $F \models G$), and hence $\mathcal{A}(F \to G) = 1$. Otherwise $\mathcal{A}(F) = 0$, then $\mathcal{A}(F \to G) = \mathsf{B}_{\to}(0, \mathcal{A}(G)) = 1$ independently of $\mathcal{A}(G)$. In both cases, $\mathcal{A} \models F \to G$.

($\Leftarrow$) Suppose that $F$ does not entail $G$. Then there exists a $\Pi$-valuation $\mathcal{A}$ such that $\mathcal{A} \models F$, but not $\mathcal{A} \models G$. Consequently, $\mathcal{A}(F \to G) = \mathsf{B}_{\to}(\mathcal{A}(F), \mathcal{A}(G)) = \mathsf{B}_{\to}(1, 0) = 0$, so $(F \to G)$ does not hold in $\mathcal{A}$. □

**Proposition 2.2** *$F \models\!\!\models G$ if and only if $\models (F \leftrightarrow G)$.*

**Proof.** Analogously to Prop. 2.1. □

Entailment is extended to sets of formulas $N$ in the "natural way":

$N \models F$ if for all $\Pi$-valuations $\mathcal{A}$:
if $\mathcal{A} \models G$ for all $G \in N$, then $\mathcal{A} \models F$.

Note: formulas are always finite objects; but sets of formulas may be infinite. Therefore, it is in general not possible to replace a set of formulas by the conjunction of its elements.

### Validity vs. Unsatisfiability

Validity and unsatisfiability are just two sides of the same medal as explained by the following proposition.

**Proposition 2.3** *$F$ is valid if and only if $\neg F$ is unsatisfiable.*

**Proof.** ($\Rightarrow$) If $F$ is valid, then $\mathcal{A}(F) = 1$ for every valuation $\mathcal{A}$. Hence $\mathcal{A}(\neg F) = \mathsf{B}_\neg(\mathcal{A}(F)) = \mathsf{B}_\neg(1) = 0$ for every valuation $\mathcal{A}$, so $\neg F$ is unsatisfiable.

($\Leftarrow$) Analogously. □

Hence in order to design a theorem prover (validity checker) it is sufficient to design a checker for unsatisfiability.

In a similar way, entailment $N \models F$ can be reduced to unsatisfiability:

**Proposition 2.4** *$N \models F$ if and only if $N \cup \{\neg F\}$ is unsatisfiable.*

### Checking Unsatisfiability

Every formula $F$ contains only finitely many propositional variables. Obviously, $\mathcal{A}(F)$ depends only on the values of those finitely many variables in $F$ under $\mathcal{A}$.

If $F$ contains $n$ distinct propositional variables, then it is sufficient to check $2^n$ valuations to see whether $F$ is satisfiable or not.
$\Rightarrow$ truth table.

So the satisfiability problem is clearly deciadable (but, by Cook's Theorem, NP-complete).

Nevertheless, in practice, there are (much) better methods than truth tables to check the satisfiability of a formula. (later more)

### Substitution Theorem

**Proposition 2.5** *Let $F$ and $G$ be equivalent formulas, let $H$ be a formula in which $F$ occurs as a subformula.*

*Then $H$ is equivalent to $H'$ where $H'$ is obtained from $H$ by replacing the occurrence of the subformula $F$ by $G$. (Notation: $H = H[F]$, $H' = H[G]$.)*

**Proof.** The proof proceeds by induction over the formula structure of $H$.

Each of the formulas $\bot$, $\top$, and $P$ for $P \in \Pi$ contains only one subformula, namely itself. Hence, if $H = H[F]$ equals $\bot$, $\top$, or $P$, then $H = F$, $H' = G$, and $H$ and $H'$ are equivalent by assumption.

If $H = H_1 \wedge H_2$, then either $F$ equals $H$ (this case is treated as above), or $F$ is a subformula of $H_1$ or $H_2$. Without loss of generality, assume that $F$ is a subformula of $H_1$, so $H = H_1[F] \wedge H_2$. By the induction hypothesis, $H_1[F]$ and $H_1[G]$ are equivalent. Hence, for every valuation $\mathcal{A}$, $\mathcal{A}(H') = \mathcal{A}(H_1[G] \wedge H_2) = \mathcal{A}(H_1[G]) \wedge \mathcal{A}(H_2) = \mathcal{A}(H_1[F]) \wedge \mathcal{A}(H_2) = \mathcal{A}(H_1[F] \wedge H_2) = \mathcal{A}(H)$.

The other boolean connectives are handled analogously. □

### Some Important Equivalences

**Proposition 2.6** *The following equivalences are valid for all formulas $F, G, H$:*

$$
\begin{array}{rll}
(F \wedge F) \leftrightarrow F & & \\
(F \vee F) \leftrightarrow F & & \textit{(Idempotency)} \\
(F \wedge G) \leftrightarrow (G \wedge F) & & \\
(F \vee G) \leftrightarrow (G \vee F) & & \textit{(Commutativity)} \\
(F \wedge (G \wedge H)) \leftrightarrow ((F \wedge G) \wedge H) & & \\
(F \vee (G \vee H)) \leftrightarrow ((F \vee G) \vee H) & & \textit{(Associativity)} \\
(F \wedge (G \vee H)) \leftrightarrow ((F \wedge G) \vee (F \wedge H)) & & \\
(F \vee (G \wedge H)) \leftrightarrow ((F \vee G) \wedge (F \vee H)) & & \textit{(Distributivity)}
\end{array}
$$

$$
\begin{array}{rll}
(F \wedge (F \vee G)) \leftrightarrow F & & \\
(F \vee (F \wedge G)) \leftrightarrow F & & \textit{(Absorption)} \\
(\neg\neg F) \leftrightarrow F & & \textit{(Double Negation)} \\
\neg(F \wedge G) \leftrightarrow (\neg F \vee \neg G) & & \\
\neg(F \vee G) \leftrightarrow (\neg F \wedge \neg G) & & \textit{(De Morgan's Laws)} \\
(F \wedge G) \leftrightarrow F, \textit{ if } G \textit{ is a tautology} & & \\
(F \vee G) \leftrightarrow \top, \textit{ if } G \textit{ is a tautology} & & \\
(F \wedge G) \leftrightarrow \bot, \textit{ if } G \textit{ is unsatisfiable} & & \\
(F \vee G) \leftrightarrow F, \textit{ if } G \textit{ is unsatisfiable} & & \textit{(Tautology Laws)}
\end{array}
$$

$$
\begin{array}{rll}
(F \leftrightarrow G) \leftrightarrow ((F \rightarrow G) \wedge (G \rightarrow F)) & & \textit{(Equivalence)} \\
(F \rightarrow G) \leftrightarrow (\neg F \vee G) & & \textit{(Implication)}
\end{array}
$$

## 2.4 Normal Forms

We define *conjunctions* of formulas as follows:

$\bigwedge_{i=1}^{0} F_i = \top$.

$\bigwedge_{i=1}^{1} F_i = F_1$.

$\bigwedge_{i=1}^{n+1} F_i = \bigwedge_{i=1}^{n} F_i \wedge F_{n+1}$.

and analogously *disjunctions:*

$\bigvee_{i=1}^{0} F_i = \bot$.

$\bigvee_{i=1}^{1} F_i = F_1$.

$\bigvee_{i=1}^{n+1} F_i = \bigvee_{i=1}^{n} F_i \vee F_{n+1}$.

### Literals and Clauses

A *literal* is either a propositional variable $P$ or a negated propositional variable $\neg P$.

A *clause* is a (possibly empty) disjunction of literals.

### CNF and DNF

A formula is in *conjunctive normal form (CNF, clause normal form)*, if it is a conjunction of disjunctions of literals (or in other words, a conjunction of clauses).

A formula is in *disjunctive normal form (DNF)*, if it is a disjunction of conjunctions of literals.

Warning: definitions in the literature differ:

are complementary literals permitted?
are duplicated literals permitted?
are empty disjunctions/conjunctions permitted?

Checking the validity of CNF formulas or the unsatisfiability of DNF formulas is easy:

A formula in CNF is valid, if and only if each of its disjunctions contains a pair of complementary literals $P$ and $\neg P$.

Conversely, a formula in DNF is unsatisfiable, if and only if each of its conjunctions contains a pair of complementary literals $P$ and $\neg P$.

On the other hand, checking the unsatisfiability of CNF formulas or the validity of DNF formulas is known to be coNP-complete.

### Conversion to CNF/DNF

**Proposition 2.7** *For every formula there is an equivalent formula in CNF (and also an equivalent formula in DNF).*

**Proof.** We consider the case of CNF.

Apply the following rules as long as possible (modulo associativity and commutativity of $\wedge$ and $\vee$):

Step 1: Eliminate equivalences:

$$(F \leftrightarrow G) \;\Rightarrow_K\; (F \rightarrow G) \wedge (G \rightarrow F)$$

Step 2: Eliminate implications:

$$(F \rightarrow G) \;\Rightarrow_K\; (\neg F \vee G)$$

Step 3: Push negations downward:

$$\neg(F \vee G) \;\Rightarrow_K\; (\neg F \wedge \neg G)$$
$$\neg(F \wedge G) \;\Rightarrow_K\; (\neg F \vee \neg G)$$

Step 4: Eliminate multiple negations:

$$\neg\neg F \;\Rightarrow_K\; F$$

Step 5: Push disjunctions downward:

$$(F \wedge G) \vee H \;\Rightarrow_K\; (F \vee H) \wedge (G \vee H)$$

Step 6: Eliminate $\top$ and $\bot$:

$$(F \wedge \top) \;\Rightarrow_K\; F$$
$$(F \wedge \bot) \;\Rightarrow_K\; \bot$$
$$(F \vee \top) \;\Rightarrow_K\; \top$$
$$(F \vee \bot) \;\Rightarrow_K\; F$$
$$\neg\bot \;\Rightarrow_K\; \top$$
$$\neg\top \;\Rightarrow_K\; \bot$$

Proving termination is easy for steps 2, 4, and 6; steps 1, 3, and 5 are a bit more complicated.

For step 1, we can prove termination in the following way: We define a function $\phi$ from formulas to positive integers such that $\phi(\bot) = \phi(\top) = \phi(P) = 1$, $\phi(\neg F) = \phi(F)$, $\phi(F \wedge G) = \phi(F \vee G) = \phi(F \rightarrow G) = \phi(F) + \phi(G)$, and $\phi(F \leftrightarrow G) = 2\phi(F) + 2\phi(G) + 1$. Observe that $\phi$ is constructed in such a way that $\phi(F) > \phi(G)$ implies $\phi(H[F]) > \phi(H[G])$ for all formulas $F$, $G$, and $H$. Using this property, we can show that whenever a formula $H'$ is the result of applying the rule of step 1 to a formula $H$, then $\phi(H) > \phi(H')$. Since $\phi$ takes only positive integer values, step 1 must terminate.

Termination of steps 3 and 5 is proved similarly. For step 3, we use function $\mu$ from formulas to positive integers such that $\mu(\bot) = \mu(\top) = \mu(P) = 1$, $\mu(\neg F) = 2\mu(F)$, $\mu(F \wedge G) = \mu(F \vee G) = \mu(F \rightarrow G) = \mu(F \leftrightarrow G) = \mu(F) + \mu(G) + 1$. Whenever a formula $H'$ is the result of applying a rule of step 3 to a formula $H$, then $\mu(H) > \mu(H')$. Since $\mu$ takes only positive integer values, step 3 must terminate.

For step 5, we use a function $\nu$ from formulas to positive integers such that $\nu(\bot) = \nu(\top) = \nu(P) = 1$, $\nu(\neg F) = \nu(F) + 1$, $\nu(F \wedge G) = \nu(F \rightarrow G) = \nu(F \leftrightarrow G) = \nu(F) + \nu(G) + 1$, and $\nu(F \vee G) = 2\nu(F)\nu(G)$. Again, if a formula $H'$ is the result of applying a rule of step 5 to a formula $H$, then $\nu(H) > \nu(H')$. Since $\nu$ takes only positive integer values, step 5 terminates, too.

The resulting formula is equivalent to the original one and in CNF.

The conversion of a formula to DNF works in the same way, except that conjunctions have to be pushed downward in step 5. □

### Complexity

Conversion to CNF (or DNF) may produce a formula whose size is *exponential* in the size of the original one.

### Satisfiability-preserving Transformations

The goal

"find a formula $G$ in CNF such that $F \models\!\!\!\models G$"

is unpractical.

But if we relax the requirement to

"find a formula $G$ in CNF such that $F \models \bot \Leftrightarrow G \models \bot$"

we can get an efficient transformation.

Idea: A formula $F[F']$ is satisfiable if and only if $F[P] \wedge (P \leftrightarrow F')$ is satisfiable (where $P$ is a new propositional variable that works as an abbreviation for $F'$).

We can use this rule recursively for all subformulas in the original formula (this introduces a linear number of new propositional variables).

Conversion of the resulting formula to CNF increases the size only by an additional factor (each formula $P \leftrightarrow F'$ gives rise to at most one application of the distributivity law).

### Optimized Transformations

A further improvement is possible by taking the *polarity* of the subformula $F$ into account.

Assume that $F$ contains neither $\rightarrow$ nor $\leftrightarrow$. A subformula $F'$ of $F$ has *positive polarity* in $F$, if it occurs below an even number of negation signs; it has *negative polarity* in $F$, if it occurs below an odd number of negation signs.

**Proposition 2.8** *Let $F[F']$ be a formula containing neither $\rightarrow$ nor $\leftrightarrow$; let $P$ be a propositional variable not occurring in $F[F']$.*

*If $F'$ has positive polarity in $F$, then $F[F']$ is satisfiable if and only if $F[P] \wedge (P \rightarrow F')$ is satisfiable.*

*If $F'$ has negative polarity in $F$, then $F[F']$ is satisfiable if and only if $F[P] \wedge (F' \rightarrow P)$ is satisfiable.*

**Proof.** Exercise. □