Propositional Logic: Operations

2.1.2 Definition (Atom, Literal, Clause)

A propositional variable P is called an *atom*. It is also called a *(positive) literal* and its negation $\neg P$ is called a *(negative) literal*.

The functions comp and atom map a literal to its complement, or atom, respectively: if $\mathrm{comp}(\neg P) = P$ and $\mathrm{comp}(P) = \neg P$, $\mathrm{atom}(\neg P) = P$ and $\mathrm{atom}(P) = P$ for all $P \in \Sigma$. Literals are denoted by letters L, K. Two literals P and $\neg P$ are called *complementary*.

A disjunction of literals $L_1 \vee \ldots \vee L_n$ is called a *clause*. A clause is identified with the multiset of its literals.

2.1.3 Definition (Position)

A *position* is a word over $\mathbb{N}$. The set of positions of a formula ϕ is inductively defined by

$$\begin{array}{rcl} \text{pos}(\phi) & := & \{\epsilon\} \text{ if } \phi \in \{\top, \bot\} \text{ or } \phi \in \Sigma \\ \text{pos}(\neg\phi) & := & \{\epsilon\} \cup \{1p \mid p \in \text{pos}(\phi)\} \\ \text{pos}(\phi \circ \psi) & := & \{\epsilon\} \cup \{1p \mid p \in pos(\phi)\} \cup \{2p \mid p \in \text{pos}(\psi)\} \end{array}$$

where $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

The prefix order $\leq$ on positions is defined by $p \leq q$ if there is some p' such that $pp' = q$. Note that the prefix order is partial, e.g., the positions 12 and 21 are not comparable, they are "parallel", see below.

The relation $<$ is the strict part of $\leq$, i.e., $p < q$ if $p \leq q$ but not $q \leq p$.

The relation $\parallel$ denotes incomparable, also called parallel positions, i.e., $p \parallel q$ if neither $p \leq q$, nor $q \leq p$.

A position p is *above* q if $p \leq q$, p is *strictly above* q if $p < q$, and p and q are *parallel* if $p \parallel q$.

The *size* of a formula ϕ is given by the cardinality of $\text{pos}(\phi)$: $|\phi| := |\,\text{pos}(\phi)|$.

The *subformula* of ϕ at position $p \in \text{pos}(\phi)$ is inductively defined by $\phi|_\epsilon := \phi$, $\neg\phi|_{1p} := \phi|_p$, and $(\phi_1 \circ \phi_2)|_{ip} := \phi_i|_p$ where $i \in \{1, 2\}$, $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

Finally, the *replacement* of a subformula at position $p \in \text{pos}(\phi)$ by a formula ψ is inductively defined by $\phi[\psi]_\epsilon := \psi$, $(\neg\phi)[\psi]_{1p} := \neg\phi[\psi]_p$, and $(\phi_1 \circ \phi_2)[\psi]_{1p} := (\phi_1[\psi]_p \circ \phi_2)$, $(\phi_1 \circ \phi_2)[\psi]_{2p} := (\phi_1 \circ \phi_2[\psi]_p)$, where $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

2.1.5 Definition (Polarity)

The *polarity* of the subformula $\phi|_p$ of ϕ at position $p \in \text{pos}(\phi)$ is inductively defined by

$$
\begin{array}{rcl}
\text{pol}(\phi, \epsilon) & := & 1 \\
\text{pol}(\neg\phi, 1p) & := & -\text{pol}(\phi, p) \\
\text{pol}(\phi_1 \circ \phi_2, ip) & := & \text{pol}(\phi_i, p) \quad \text{if } \circ \in \{\wedge, \vee\}, i \in \{1, 2\} \\
\text{pol}(\phi_1 \rightarrow \phi_2, 1p) & := & -\text{pol}(\phi_1, p) \\
\text{pol}(\phi_1 \rightarrow \phi_2, 2p) & := & \text{pol}(\phi_2, p) \\
\text{pol}(\phi_1 \leftrightarrow \phi_2, ip) & := & 0 \quad \text{if } i \in \{1, 2\}
\end{array}
$$

Valuations can be nicely represented by sets or sequences of literals that do not contain complementary literals nor duplicates.

If $\mathcal{A}$ is a (partial) valuation of domain Σ then it can be represented by the set
$\{P \mid P \in \Sigma \text{ and } \mathcal{A}(P) = 1\} \cup \{\neg P \mid P \in \Sigma \text{ and } \mathcal{A}(P) = 0\}$.

Another, equivalent representation are *Herbrand* interpretations that are sets of positive literals, where all atoms not contained in an Herbrand interpretation are false. If $\mathcal{A}$ is a total valuation of domain Σ then it corresponds to the Herbrand interpretation
$\{P \mid P \in \Sigma \text{ and } \mathcal{A}(P) = 1\}$.

2.2.4 Theorem (Deduction Theorem)

$\phi \models \psi$ iff $\models \phi \to \psi$

2.2.6 Lemma (Formula Replacement)

Let ϕ be a propositional formula containing a subformula ψ at position p, i.e., $\phi|_p = \psi$. Furthermore, assume $\models \psi \leftrightarrow \chi$. Then $\models \phi \leftrightarrow \phi[\chi]_p$.

Propositional Tableau

2.4.1 Definition (α-, β-Formulas)

A formula ϕ is called an α-formula if ϕ is a formula $\neg\neg\phi_1$, $\phi_1 \wedge \phi_2$, $\phi_1 \leftrightarrow \phi_2$, $\neg(\phi_1 \vee \phi_2)$, or $\neg(\phi_1 \rightarrow \phi_2)$.

A formula ϕ is called a β-formula if ϕ is a formula $\phi_1 \vee \phi_2$, $\phi_1 \rightarrow \phi_2$, $\neg(\phi_1 \wedge \phi_2)$, or $\neg(\phi_1 \leftrightarrow \phi_2)$.

2.4.2 Definition (Direct Descendant)

Given an α- or β-formula ϕ, its direct descendants are as follows:

α	Left Descendant	Right Descendant
$\neg\neg\phi$	ϕ	ϕ
$\phi_1 \land \phi_2$	ϕ_1	ϕ_2
$\phi_1 \leftrightarrow \phi_2$	$\phi_1 \rightarrow \phi_2$	$\phi_2 \rightarrow \phi_1$
$\neg(\phi_1 \lor \phi_2)$	$\neg\phi_1$	$\neg\phi_2$
$\neg(\phi_1 \rightarrow \phi_2)$	ϕ_1	$\neg\phi_2$

β	Left Descendant	Right Descendant
$\phi_1 \lor \phi_2$	ϕ_1	ϕ_2
$\phi_1 \rightarrow \phi_2$	$\neg\phi_1$	ϕ_2
$\neg(\phi_1 \land \phi_2)$	$\neg\phi_1$	$\neg\phi_2$
$\neg(\phi_1 \leftrightarrow \phi_2)$	$\neg(\phi_1 \rightarrow \phi_2)$	$\neg(\phi_2 \rightarrow \phi_1)$

2.4.3 Proposition ()

For any valuation $\mathcal{A}$:

(i) if ϕ is an α-formula then $\mathcal{A}(\phi) = 1$ iff $\mathcal{A}(\phi_1) = 1$ and $\mathcal{A}(\phi_2) = 1$ for its descendants ϕ_1, ϕ_2.

(ii) if ϕ is a β-formula then $\mathcal{A}(\phi) = 1$ iff $\mathcal{A}(\phi_1) = 1$ or $\mathcal{A}(\phi_2) = 1$ for its descendants ϕ_1, ϕ_2.

Tableau Rewrite System

The tableau calculus operates on states that are sets of sequences of formulas. Semantically, the set represents a disjunction of sequences that are interpreted as conjunctions of the respective formulas.

A sequence of formulas $(\phi_1, \ldots, \phi_n)$ is called *closed* if there are two formulas ϕ_i and ϕ_j in the sequence where $\phi_i = \mathrm{comp}(\phi_j)$.

A state is *closed* if all its formula sequences are closed.

The tableau calculus is a calculus showing unsatisfiability of a formula. Such calculi are called *refutational* calculi. Recall a formula ϕ is valid iff $\neg\phi$ is unsatisfiable.

A formula ϕ occurring in some sequence is called *open* if in case ϕ is an α-formula not both direct descendants are already part of the sequence and if it is a β-formula none of its descendants is part of the sequence.

Tableau Rewrite Rules

α-Expansion $N \uplus \{(\phi_1, \ldots, \psi, \ldots, \phi_n)\} \Rightarrow_T$
$N \uplus \{(\phi_1, \ldots, \psi, \ldots, \phi_n, \psi_1, \psi_2)\}$
provided ψ is an open α-formula, ψ_1, ψ_2 its direct descendants and the sequence is not closed.

β-Expansion $N \uplus \{(\phi_1, \ldots, \psi, \ldots, \phi_n)\} \Rightarrow_T$
$N \uplus \{(\phi_1, \ldots, \psi, \ldots, \phi_n, \psi_1)\} \uplus \{(\phi_1, \ldots, \psi, \ldots, \phi_n, \psi_2)\}$
provided ψ is an open β-formula, ψ_1, ψ_2 its direct descendants and the sequence is not closed.

Tableau Properties

2.4.4 Theorem (Propositional Tableau is Sound)

If for a formula ϕ the tableau calculus computes $\{(\neg\phi)\} \Rightarrow^*_{\mathrm{T}} N$ and N is closed, then ϕ is valid.

2.4.5 Theorem (Propositional Tableau Terminates)

Starting from a start state $\{(\phi)\}$ for some formula ϕ, the relation $\Rightarrow^+_{\mathrm{T}}$ is well-founded.

2.4.6 Theorem (Propositional Tableau is Complete)

If ϕ is valid, tableau computes a closed state out of $\{(\neg\phi)\}$.

2.4.7 Corollary (Propositional Tableau generates Models)

Let ϕ be a formula, $\{(\phi)\} \Rightarrow^*_{\mathrm{T}} N$ and $s \in N$ be a sequence that is not closed and neither α-expansion nor β-expansion are applicable to s. Then the literals in s form a (partial) valuation that is a model for ϕ.

Normal Forms

Definition (CNF, DNF)

A formula is in *conjunctive normal form (CNF)* or *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.

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

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

(ii) conversely, a formula in DNF is unsatisfiable, if and only if each of its conjunctions contains a pair of complementary literals P and $\neg P$

Basic CNF Transformation

ElimEquiv	$\chi[(\phi \leftrightarrow \psi)]_p \Rightarrow_{\mathrm{BCNF}} \chi[(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)]_p$
ElimImp	$\chi[(\phi \rightarrow \psi)]_p \Rightarrow_{\mathrm{BCNF}} \chi[(\neg\phi \vee \psi)]_p$
PushNeg1	$\chi[\neg(\phi \vee \psi)]_p \Rightarrow_{\mathrm{BCNF}} \chi[(\neg\phi \wedge \neg\psi)]_p$
PushNeg2	$\chi[\neg(\phi \wedge \psi)]_p \Rightarrow_{\mathrm{BCNF}} \chi[(\neg\phi \vee \neg\psi)]_p$
PushNeg3	$\chi[\neg\neg\phi]_p \Rightarrow_{\mathrm{BCNF}} \chi[\phi]_p$
PushDisj	$\chi[(\phi_1 \wedge \phi_2) \vee \psi]_p \Rightarrow_{\mathrm{BCNF}} \chi[(\phi_1 \vee \psi) \wedge (\phi_2 \vee \psi)]_p$
ElimTB1	$\chi[(\phi \wedge \top)]_p \Rightarrow_{\mathrm{BCNF}} \chi[\phi]_p$
ElimTB2	$\chi[(\phi \wedge \bot)]_p \Rightarrow_{\mathrm{BCNF}} \chi[\bot]_p$
ElimTB3	$\chi[(\phi \vee \top)]_p \Rightarrow_{\mathrm{BCNF}} \chi[\top]_p$
ElimTB4	$\chi[(\phi \vee \bot)]_p \Rightarrow_{\mathrm{BCNF}} \chi[\phi]_p$
ElimTB5	$\chi[\neg\bot]_p \Rightarrow_{\mathrm{BCNF}} \chi[\top]_p$
ElimTB6	$\chi[\neg\top]_p \Rightarrow_{\mathrm{BCNF}} \chi[\bot]_p$

Basic CNF Algorithm

Algorithm: 2 bcnf(ϕ)

Input : A propositional formula ϕ.
Output: A propositional formula ψ equivalent to ϕ in CNF.

whilerule *(***ElimEquiv**(ϕ)*)* **do** ;
whilerule *(***ElimImp**(ϕ)*)* **do** ;
whilerule *(***ElimTB1**(ϕ),. . .,**ElimTB6**(ϕ)*)* **do** ;
whilerule *(***PushNeg1**(ϕ),. . .,**PushNeg3**(ϕ)*)* **do** ;
whilerule *(***PushDisj**(ϕ)*)* **do** ;
return ϕ;

Advanced CNF Algorithm

For the formula

$$P_1 \leftrightarrow (P_2 \leftrightarrow (P_3 \leftrightarrow (\ldots (P_{n-1} \leftrightarrow P_n) \ldots)))$$

the basic CNF algorithm generates a CNF with 2^{n-1} clauses.