Chapter 2

Propositional Logic

A logic is a formal language with a mathematically precise semantics. A formal language is rigorously defined by a grammar and there are efficient algorithms that can decide whether a string of characters belongs to the language or not. The semantics is typically a notion of truth based on the notion of an abstract model. Propositional logic is concerned with the logic of propositions. In propositional logic from the propositions "Socrates is a man" and "If Socrates is a man then Socrates is mortal" the conclusion "Socrates is mortal" can be derived. The logic is expressive enough to talk about propositions, but not, e.g., about individuals. This will be possible in first-order logic (Chapter 3), a proper extension of propositional logic.

Nevertheless, propositional logic is an interesting candidate for many applications. For example, our overall computer technology is based on propositions, i.e., bits that can either become true or false. The representation of numbers on a computer is based on fixed length bit-vectors rather than on the abstract concept of an arbitrarily large number as known from math. Hardware is designed on a "logical level" that meets to a large extend propositional logic and is, therefore, the currently most well-known application of propositional logic reasoning in computer science.

2.1 Syntax

Consider a finite, non-empty signature Σ of propositional variables, the "alphabet" of propositional logic. In addition to the alphabet "propositional connectives" are further building blocks composing the sentences (formulas) of the language. Auxiliary symbols such as parentheses enable disambiguation.

Definition 2.1.1 (Propositional Formula)**.** The set $\mathrm{PROP}(\Sigma)$ of *propositional formulas* over a signature Σ is inductively defined by:

PROP(Σ)	Comment
$\bot$	connective $\bot$ denotes "false"
$\top$	connective $\top$ denotes "true"
P	for any propositional variable $P \in \Sigma$
$(\neg\phi)$	connective $\neg$ denotes "negation"
$(\phi \wedge \psi)$	connective $\wedge$ denotes "conjunction"
$(\phi \vee \psi)$	connective $\vee$ denotes "disjunction"
$(\phi \rightarrow \psi)$	connective $\rightarrow$ denotes "implication"
$(\phi \leftrightarrow \psi)$	connective $\leftrightarrow$ denotes "equivalence"

where $\phi, \psi \in \mathrm{PROP}(\Sigma)$.

The above definition is an abbreviation for setting $\mathrm{PROP}(\Sigma)$ to be the language of a context free grammar $\mathrm{PROP}(\Sigma) = L((N, T, P, S))$ (see Definition 1.3.9) where $N = \{\phi, \psi\}$, $T = \Sigma \cup \{(,)\} \cup \{\bot, \top, \neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ with start symbol rules $S \Rightarrow \phi \mid \psi$, $\phi \Rightarrow \bot \mid \top \mid (\neg\phi) \mid (\phi \wedge \psi) \mid (\phi \vee \psi) \mid (\phi \rightarrow \psi) \mid (\phi \leftrightarrow \psi)$, $\psi \Rightarrow \phi$, and $\phi \Rightarrow P$, for every $P \in \Sigma$.

As a notational convention we assume that $\neg$ binds strongest and we omit outermost parenthesis. So $\neg P \vee Q$ is actually a shorthand for $((\neg P) \vee Q)$. For all other logical connectives parenthesis are explicitly shown if needed. The connectives $\wedge$ and $\vee$ are actually associative and commutative, see the next Section 2.2. Therefore, the formula $((P \wedge Q) \wedge R)$ can be written $P \wedge Q \wedge R$ without causing confusion.

The connectives $\wedge$ and $\vee$ are introduced as binary connectives. They are associative and commutative as already mentioned above. When implementing formulas both connectives are typically considered to be of variable arity. This saves both space and enables more efficient algorithms for formula manipulation.

Definition 2.1.2 (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.

The length of a clause C, i.e., the number of literals, is denoted by $|C|$ according to the cardinality of its multiset interpretation. A clause is called *Horn* if it contains at most one positive literal.

Automated reasoning is very much formula manipulation. In order to precisely represent the manipulation of a formula, we introduce positions.

Definition 2.1.3 (Position)**.** A *position* is a word over $\mathbb{N}$. The set of positions of a formula ϕ is inductively defined by

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

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 $\mathrm{pos}(\phi)$: $|\phi| := |\,\mathrm{pos}(\phi)|$. The *subformula* of ϕ at position $p \in \mathrm{pos}(\phi)$ 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 $p \in \mathrm{pos}(\phi)$ 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 $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

Example 2.1.4. The set of positions for the formula $\phi = (P \wedge Q) \rightarrow (P \vee Q)$ is $\mathrm{pos}(\phi) = \{\epsilon, 1, 11, 12, 2, 21, 22\}$. The subformula at position 22 is Q, $\phi|_{22} = Q$ and replacing this formula by $P \leftrightarrow Q$ results in $\phi[P \leftrightarrow Q]_{22} = (P \wedge Q) \rightarrow (P \vee (P \leftrightarrow Q))$.

A further prerequisite for efficient formula manipulation is the notion of the *polarity* of the subformula $\phi|_p$ of ϕ at position p. The polarity considers the number of "negations" starting from ϕ at position ϵ down to p. It is 1 for an even number of explicit or implicit negation symbols along the path, -1 for an odd number and 0 if there is at least one equivalence connective along the path.

Definition 2.1.5 (Polarity)**.** The *polarity* of the subformula $\phi|_p$ of ϕ at position $p \in \mathrm{pos}(\phi)$ is inductively defined by

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

Example 2.1.6. Reconsider the formula $\phi = (A \wedge B) \rightarrow (A \vee B)$ of Example 2.1.4. Then $\mathrm{pol}(\phi, 1) = \mathrm{pol}(\phi, 11) = -1$ and $\mathrm{pol}(\phi, 2) = \mathrm{pol}(\phi, 22) = 1$. For the formula $\phi' = (A \wedge B) \leftrightarrow (A \vee B)$ we get $\mathrm{pol}(\phi', \epsilon) = 1$ and $\mathrm{pol}(\phi', p) = 0$ for all other $p \in \mathrm{pos}(\phi')$, $p \neq \epsilon$.

2.2 Semantics

In *classical logic* there are two truth values "true" and "false" which we shall denote, respectively, by 1 and 0. There are *many-valued logics* [63] having more than two truth values and in fact, as we will see later on, for the definition of

some propositional logic calculi, we will need an implicit third truth value called "undefined".

Definition 2.2.1 ((Partial) Valuation)**.** A Σ-*valuation* is a map

$$\mathcal{A} : \Sigma \to \{0, 1\}.$$

where $\{0, 1\}$ is the set of *truth values*. A *partial* Σ-*valuation* is a map $\mathcal{A}' : \Sigma' \to \{0, 1\}$ where $\Sigma' \subseteq \Sigma$.

Definition 2.2.2 (Semantics)**.** A Σ-valuation $\mathcal{A}$ is inductively extended from propositional variables to propositional formulas $\phi, \psi \in \mathrm{PROP}(\Sigma)$ by

$$\begin{array}{rcl} \mathcal{A}(\bot) & := & 0 \\ \mathcal{A}(\top) & := & 1 \\ \mathcal{A}(\neg\phi) & := & 1 - \mathcal{A}(\phi) \\ \mathcal{A}(\phi \wedge \psi) & := & \min(\{\mathcal{A}(\phi), \mathcal{A}(\psi)\}) \\ \mathcal{A}(\phi \vee \psi) & := & \max(\{\mathcal{A}(\phi), \mathcal{A}(\psi)\}) \\ \mathcal{A}(\phi \to \psi) & := & \max(\{1 - \mathcal{A}(\phi),\ \mathcal{A}(\psi)\}) \\ \mathcal{A}(\phi \leftrightarrow \psi) & := & \text{if } \mathcal{A}(\phi) = \mathcal{A}(\psi) \text{ then 1 else 0} \end{array}$$

If $\mathcal{A}(\phi) = 1$ for some Σ-valuation $\mathcal{A}$ of a formula ϕ then ϕ is *satisfiable* and we write $\mathcal{A} \models \phi$. In this case $\mathcal{A}$ is a *model* of ϕ. If $\mathcal{A}(\phi) = 1$ for all Σ-valuations $\mathcal{A}$ of a formula ϕ then ϕ is *valid* and we write $\models \phi$. If there is no Σ-valuation $\mathcal{A}$ for a formula ϕ where $\mathcal{A}(\phi) = 1$ we say ϕ is *unsatisfiable*. A formula ϕ *entails* ψ, written $\phi \models \psi$, if for all Σ-valuations $\mathcal{A}$ whenever $\mathcal{A} \models \phi$ then $\mathcal{A} \models \psi$. Two formulas ϕ and ψ are called *equisatisfiable*, if ϕ is satisfiable iff ψ is satisfiable (not necessarily in the same models).

Accordingly, a formula ϕ is satisfiable, valid, unsatisfiable, respectively, with respect to a partial valuation $\mathcal{A}'$ with domain Σ', if for all valuations $\mathcal{A}$ with $\mathcal{A}(P) = \mathcal{A}'(P)$ for all $P \in \Sigma'$, the formula ϕ is satisfiable, valid, unsatisfiable, respectively, with respect to a $\mathcal{A}$.

I call the fact that some formula ϕ is satisfiable, unsatisfiable, or valid, the *status* of ϕ. Note that if ϕ is valid it is also satisfiable, but not the other way round.

Valuations of propositional logic collapse with *interpretations*. Given a formula ϕ and an interpretation (valuation) $\mathcal{A}$ such that $\mathcal{A}(\phi) = 1$ then the interpretation $\mathcal{A}$ is also called a *model* for ϕ.

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 $\{P \mid P \in \Sigma \text{ and } \mathcal{A}(P) = 1\}$.

Please note the subtle difference between an Herbrand interpretation and a valuation represented by a set of literals. The latter can be partial with respect to a formula whereas the former is always total by definition. For example, the empty Herbrand interpretation assigns false to all propositional variables.

For example, for the valuation $\mathcal{A} = \{P, \neg Q\}$ the truth value of $P \vee Q$ is $\mathcal{A}(P \vee Q) = 1$, for $P \vee R$ it is $\mathcal{A}(P \vee R) = 1$, for $\neg P \wedge R$ it is $\mathcal{A}(\neg P \wedge R) = 0$, and the status of $\neg P \vee R$ cannot be established by $\mathcal{A}$. In particular, $\mathcal{A}$ is a partial valuation for $\Sigma = \{P, Q, R\}$. A literal L is *defined* with respect to a partial valuation $\mathcal{A}$ if $L \in \mathcal{A}$ or $\mathrm{comp}(L) \in \mathcal{A}$.

Example 2.2.3. The formula $\phi \vee \neg\phi$ is valid, independently of ϕ. According to Definition 2.2.2 we need to prove that for all Σ-valuations $\mathcal{A}$ of ϕ we have $\mathcal{A}(\phi \vee \neg\phi) = 1$. So let $\mathcal{A}$ be an arbitrary valuation. There are two cases to consider. If $\mathcal{A}(\phi) = 1$ then $\mathcal{A}(\phi \vee \neg\phi) = 1$ because the valuation function takes the maximum if distributed over $\vee$. If $\mathcal{A}(\phi) = 0$ then $\mathcal{A}(\neg\phi) = 1$ and again by the before argument $\mathcal{A}(\phi \vee \neg\phi) = 1$. This finishes the proof that $\models \phi \vee \neg\phi$.

Theorem 2.2.4 (Deduction Theorem)**.** $\phi \models \psi$ iff $\models \phi \to \psi$

Proof. ($\Rightarrow$) Suppose that ϕ entails ψ and let $\mathcal{A}$ be an arbitrary Σ-valuation. We need to show $\mathcal{A} \models \phi \to \psi$. If $\mathcal{A}(\phi) = 1$, then $\mathcal{A}(\psi) = 1$, because ϕ entails ψ, and therefore $\mathcal{A} \models \phi \to \psi$. For otherwise, if $\mathcal{A}(\phi) = 0$, then $\mathcal{A}(\phi \to \psi) = \max(\{(1 - \mathcal{A}(\phi)), \mathcal{A}(\psi)\}) = \max(\{(1, \mathcal{A}(\psi)\}) = 1$, independently of the value of $\mathcal{A}(\psi)$. In both cases $\mathcal{A} \models \phi \to \psi$.
($\Leftarrow$) By contraposition. Suppose that ϕ does not entail ψ. Then there exists a Σ-valuation $\mathcal{A}$ such that $\mathcal{A} \models \phi$, $\mathcal{A}(\phi) = 1$ but $\mathcal{A} \not\models \psi$, i.e., $\mathcal{A}(\psi) = 0$. By definition, $\mathcal{A}(\phi \to \psi) = \max(\{(1 - \mathcal{A}(\phi)), \mathcal{A}(\psi)\}) = \max(\{(1 - 1), 0\}) = 0$, hence $\phi \to \psi$ does not hold in $\mathcal{A}$. □

So both writings $\phi \models \psi$ and $\models \phi \to \psi$ are actually equivalent. I extend the former notion to sets or sequences on the left denoting conjunction. For example, $\chi, \phi \models \psi$ is short for $\chi \wedge \phi \models \psi$.

Proposition 2.2.5. The equivalences of Figure 2.1 are valid for all formulas ϕ, ψ, χ.

Note that the formulas $\phi \wedge \psi$ and $\psi \wedge \phi$ are equivalent. Nevertheless, recalling the problem state definition for Sudokus in Section 1.1 the two states $(N; f(2,3) = 1 \wedge f(2,4) = 4; \top)$ and $(N; f(2,4) = 4 \wedge f(2,3) = 1; \top)$ are significantly different. For example, it can be that the first state can lead to a solution by the rules of the algorithm where the latter cannot, because the latter implicitly means that the square $(2,4)$ has already been checked for all values smaller than 4. This reveals the important point that arguing by logical equivalence in the context of a rule set manipulating formulas, a calculus, can lead to wrong results.

Lemma 2.2.6 (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 $\models \phi \leftrightarrow \phi[\chi]_p$.

Proof. By induction on $|p|$ and structural induction on ϕ. For the base step let $p = \epsilon$ and $\mathcal{A}$ be an arbitrary valuation.

$$\begin{aligned} \mathcal{A}(\phi) &= \mathcal{A}(\psi) && \text{(by definition of position)} \\ &= \mathcal{A}(\chi) && \text{(because } \mathcal{A} \models \psi \leftrightarrow \chi\text{)} \\ &= \mathcal{A}(\phi[\chi]_\epsilon) && \text{(by definition of replacement)} \end{aligned}$$

For the induction step the lemma holds for all positions p and has to be shown for all positions ip. By structural induction on ϕ, I show the cases where $\phi = \neg\phi_1$ and $\phi = \phi_1 \to \phi_2$ in detail. All other cases are analogous.

If $\phi = \neg\phi_1$ then showing the lemma amounts to proving $\models \neg\phi_1 \leftrightarrow \neg\phi_1[\chi]_{1p}$. Let $\mathcal{A}$ be an arbitrary valuation.

$$\begin{aligned} \mathcal{A}(\neg\phi_1) &= 1 - \mathcal{A}(\phi_1) && \text{(expanding semantics)} \\ &= 1 - \mathcal{A}(\phi_1[\chi]_p) && \text{(by induction hypothesis)} \\ &= \mathcal{A}(\neg\phi[\chi]_{1p}) && \text{(contracting semantics)} \end{aligned}$$

If $\phi = \phi_1 \to \phi_2$ then showing the lemma amounts to proving the two cases $\models (\phi_1 \to \phi_2) \leftrightarrow (\phi_1 \to \phi_2)[\chi]_{1p}$ and $\models (\phi_1 \to \phi_2) \leftrightarrow (\phi_1 \to \phi_2)[\chi]_{2p}$. Both cases are similar so I show only the first case. Let $\mathcal{A}$ be an arbitrary valuation.

$$\begin{aligned} \mathcal{A}(\phi_1 \to \phi_2) &= \max(\{(1 - \mathcal{A}(\phi_1)), \mathcal{A}(\phi_2)\}) && \text{(expanding semantics)} \\ &= \max(\{(1 - \mathcal{A}(\phi_1[\chi]_p)), \mathcal{A}(\phi_2)\}) && \text{(by induction hypothesis)} \\ &= \mathcal{A}((\phi_1 \to \phi_2)[\chi]_{1p}) && \text{(applying semantics)} \end{aligned}$$

□

Lemma 2.2.7 (Polarity Dependent Replacement)**.** Consider a formula ϕ, position $p \in \text{pos}(\phi)$, $\text{pol}(\phi, p) = 1$ and (partial) valuation $\mathcal{A}$ with $\mathcal{A}(\phi) = 1$. If for some formula ψ, $\mathcal{A}(\psi) = 1$ then $\mathcal{A}(\phi[\psi]_p) = 1$. Symmetrically, if $\text{pol}(\phi, p) = -1$ and $\mathcal{A}(\psi) = 0$ then $\mathcal{A}(\phi[\psi]_p) = 1$. If $\text{pol}(\phi, p) = 1$ and $\mathcal{A}(\psi) = 1$ then $\mathcal{A}(\phi) = \mathcal{A}(\phi[\psi]_p)$.

Proof. Exercise **??**: by induction on the length of p. □

Note that the case for the above lemma where $\text{pol}(\phi, p) = 0$ is actually Lemma 2.2.6.

C

The equivalences of Figure 2.1 show that the propositional language introduced in Definition 2.1.1 is redundant in the sense that certain connectives can be expressed by others. For example, the equivalence Eliminate $\to$ expresses implication by means of disjunction and negation. So for any propositional formula ϕ there exists an equivalent formula ϕ' such that ϕ' does not contain the implication connective. In order to prove this proposition the above replacement lemma is key.

commutative, they are equivalent. One or two columns in the truth table for the two subformulas? Again, saving a column is beneficial but in general, detecting equivalence of two subformulas may become as difficult as checking whether the overall formula is valid. A compromise, often performed in practice, are normal forms that guarantee that certain occurrences of equivalent subformulas can be found in polynomial time. For the running example, we can simply assume some ordering on the propositional variables and assume that for a disjunction of two propositional variables, the smaller variable always comes first. So if $P < Q$ then the normal form of $P \vee Q$ and $Q \vee P$ is in fact $P \vee Q$.

In practice, nobody uses truth tables as a reasoning procedure. Worst case, computing a truth table for checking the status of a formula ϕ requires $O(2^n)$ steps, where n is the number of different propositional variables in ϕ. But this is actually not the reason why the procedure is impractical, because the worst case behavior of all other procedures for propositional logic known today is also of exponential complexity. So why are truth tables not a good procedure? The answer is: because they do not adapt to the inherent structure of a formula. The reasoning mechanism of a truth table for two formulas ϕ and ψ sharing the same propositional variables is exactly the same: we enumerate all valuations. However, if ϕ is, e.g., of the form $\phi = P \wedge \phi'$ and we are interested in the satisfiability of ϕ, then ϕ can only become true for a valuation $\mathcal{A}$ with $\mathcal{A}(P) = 1$. Hence, 2^{n-1} rows of ϕ's truth table are superfluous. All procedures I will introduce in the sequel, automatically detect this (and further) specific structures of a formula and use it to speed up the reasoning process.

2.4 Propositional Tableaux

Like resolution, semantic tableaux were developed in the sixties, independently by Lis [44] and Smullyan [61] on the basis of work by Gentzen in the 30s [31] and of Beth [11] in the 50s. For an at that time state of the art overview consider Fitting's book [29].

In contrast to the calculi introduced in subsequent sections, semantic tableau does not rely on a normal form of input formulas but actually applies to any propositional formula. The formulas are divided into α- and β-formulas, where intuitively an α formula represents an (implicit) conjunction and a β formula an (implicit) disjunction.

Definition 2.4.1 (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 $\neg(\phi_1 \leftrightarrow \phi_2)$.

A common property of α-, β-formulas is that they can be decomposed into direct descendants representing (modulo negation) subformulas of the respective

formulas. Then an α-formula is valid iff all its descendants are valid and a β-formula is valid iff one of its descendants is valid. Therefore, the literature uses both the notions semantic tableaux and analytic tableaux.

Definition 2.4.2 (Direct Descendant)**.** Given an α- or β-formula ϕ, Figure 2.4 shows its direct descendants.

Duplicating ϕ for the α-descendants of $\neg\neg\phi$ is a trick for conformity. Any propositional formula is either an α-formula or a β-formula or a literal.

Proposition 2.4.3. 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.

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. A state actually represents a tree and this tree is called a tableau in the literature. So if a state is closed, the respective tree, the tableau is closed too. The tableau calculus is a calculus showing unsatisfiability of a formula. Such calculi are called *refutational* calculi. Later on soundness and completeness of the calculus imply that a formula ϕ is valid iff the rules of tableau produce a closed state starting with $N = \{(\neg\phi)\}$.

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.

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

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

For example, consider proving validity of the formula $(P \wedge \neg(Q \vee \neg R)) \rightarrow (Q \wedge R)$. Applying the tableau rules generates the following derivation:

$$
\begin{array}{l}
\{(\neg[(P \wedge \neg(Q \vee \neg R)) \rightarrow (Q \wedge R)])\} \\
\quad \alpha\text{-Expansion} \Rightarrow_{\mathrm{T}}^{*} \{(\neg[(P \wedge \neg(Q \vee \neg R)) \rightarrow (Q \wedge R)], \\
\qquad P \wedge \neg(Q \vee \neg R), \neg(Q \wedge R), P, \neg(Q \vee \neg R), \neg Q, \neg\neg R, R)\} \\
\quad \beta\text{-Expansion} \Rightarrow_{\mathrm{T}} \{(\neg[(P \wedge \neg(Q \vee \neg R)) \rightarrow (Q \wedge R)], \\
\qquad P \wedge \neg(Q \vee \neg R), \neg(Q \wedge R), P, \neg(Q \vee \neg R), \neg Q, \neg\neg R, R, \neg Q), \\
\qquad (\neg[(P \wedge \neg(Q \vee \neg R)) \rightarrow (Q \wedge R)], \\
\qquad P \wedge \neg(Q \vee \neg R), \neg(Q \wedge R), P, \neg(Q \vee \neg R), \neg Q, \neg\neg R, R, \neg R)\}
\end{array}
$$

The state after β-expansion is final, i.e., no more rule can be applied. The first sequence is not closed, whereas the second sequence is closed because it contains R and $\neg R$. Thus, the formula is not valid but satisfiable. A tree representation, where common formulas of sequences are shared, can be found in Figure 2.5. This is the traditional way of tableau presentation.

Theorem 2.4.4 (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.

Proof. It is sufficient to show the following: (i) if N is closed then the disjunction of the conjunction of all sequence formulas is unsatisfiable (ii) the two tableau rules preserve satisfiability.

Part (i) is obvious: if N is closed all its sequences are closed. A sequence is closed if it contains a formula and its negation. The conjunction of two such formulas is unsatisfiable.

Part (ii) is shown by induction on the length of the derivation and then by a case analysis for the two rules. α-Expansion: for any valuation $\mathcal{A}$ if $\mathcal{A}(\psi) = 1$ then $\mathcal{A}(\psi_1) = \mathcal{A}(\psi_2) = 1$. β-Expansion: for any valuation $\mathcal{A}$ if $\mathcal{A}(\psi) = 1$ then $\mathcal{A}(\psi_1) = 1$ or $\mathcal{A}(\psi_2) = 1$ (see Proposition 2.4.3). □

Theorem 2.4.5 (Propositional Tableau Terminates)**.** Starting from a start state $\{(\phi)\}$ for some formula ϕ, the relation $\Rightarrow_{\mathrm{T}}^{+}$ is well-founded.

Proof. Take the two-folded multiset extension of the lexicographic extension of $>$ on the naturals to triples (n, k, l) generated by the a measure μ. It is first defined on formulas by $\mu(\phi) := (n, k, l)$ where n is the number of equivalence symbols in ϕ, k is the sum of all disjunction, conjunction, implication symbols in ϕ and l is $|\phi|$. On sequences $(\phi_1, \ldots, \phi_n)$ the measure is defined to deliver a multiset by $\mu((\phi_1, \ldots, \phi_n)) := \{t_1, \ldots, t_n\}$ where $t_i = \mu(\phi_i)$ if ϕ_i is open in the sequence and $t_i = (0, 0, 0)$ otherwise. Finally, μ is extended to states N by computing the multiset $\mu(N) := \{\mu(s) \mid s \in N\}$.

Note, that α-, as well as β-expansion strictly extend sequences. Once a formula is closed in a sequence by applying an expansion rule, it remains closed forever in the sequence.

An α-expansion on a formula $\psi_1 \wedge \psi_2$ on the sequence $(\phi_1, \ldots, \psi_1 \wedge \psi_2, \ldots, \phi_n)$ results in $(\phi_1, \ldots, \psi_1 \wedge \psi_2, \ldots, \phi_n, \psi_1, \psi_2)$. It needs to be shown $\mu((\phi_1, \ldots, \psi_1 \wedge \psi_2, \ldots, \phi_n)) >_{\mathrm{mul}} \mu((\phi_1, \ldots, \psi_1 \wedge \psi_2, \ldots, \phi_n, \psi_1, \psi_2))$. In the second sequence $\mu(\psi_1 \wedge \psi_2) = (0, 0, 0)$ because the formula is closed. For the triple (n, k, l) assigned by μ to $\psi_1 \wedge \psi_2$ in the first sequence, it holds $(n, k, l) >_{\mathrm{lex}} \mu(\psi_1)$, $(n, k, l) >_{\mathrm{lex}} \mu(\psi_2)$ and $(n, k, l) >_{\mathrm{lex}} (0, 0, 0)$, the former because the ψ_i are subformulas and the latter because $l \neq 0$. This proves the case.

A β-expansion on a formula $\psi_1 \vee \psi_2$ on the sequence $(\phi_1, \ldots, \psi_1 \vee \psi_2, \ldots, \phi_n)$ results in $(\phi_1, \ldots, \psi_1 \vee \psi_2, \ldots, \phi_n, \psi_1)$, $(\phi_1, \ldots, \psi_1 \vee \psi_2, \ldots, \phi_n, \psi_2)$. It needs to be shown $\mu((\phi_1, \ldots, \psi_1 \vee \psi_2, \ldots, \phi_n)) >_{\mathrm{mul}} \mu((\phi_1, \ldots, \psi_1 \vee \psi_2, \ldots, \phi_n, \psi_1))$ and $\mu((\phi_1, \ldots, \psi_1 \vee \psi_2, \ldots, \phi_n)) >_{\mathrm{mul}} \mu((\phi_1, \ldots, \psi_1 \vee \psi_2, \ldots, \phi_n, \psi_2))$. In the derived sequences $\mu(\psi_1 \vee \psi_2) = (0, 0, 0)$ because the formula is closed. For the triple (n, k, l) assigned by μ to $\psi_1 \vee \psi_2$ in the starting sequence, it holds $(n, k, l) >_{\mathrm{lex}}$

$\mu(\psi_1)$, $(n, k, l) >_{\text{lex}} \mu(\psi_2)$ and $(n, k, l) >_{\text{lex}} (0, 0, 0)$, the former because the ψ_i are subformulas and the latter because $l \neq 0$. This proves the case. □

Theorem 2.4.6 (Propositional Tableau is Complete)**.** If ϕ is valid, tableau computes a closed state out of $\{(\neg\phi)\}$.

Proof. If ϕ is valid then $\neg\phi$ is unsatisfiable. Now assume after termination the resulting state and hence at least one sequence is not closed. For this sequence consider a valuation $\mathcal{A}$ consisting of the literals in the sequence. By assumption there are no opposite literals, so $\mathcal{A}$ is well-defined. I prove by contradiction that $\mathcal{A}$ is a model for the sequence. Assume it is not. Then there is a minimal formula in the sequence, with respect to the ordering on triples considered in the proof of Theorem 2.4.5, that is not satisfied by $\mathcal{A}$. By definition of $\mathcal{A}$ the formula cannot be a literal. So it is an α-formula or a β-formula. In all cases at least one descendant formula is contained in the sequence, is smaller than the original formula, false in $\mathcal{A}$ (Proposition 2.4.3) and hence contradicts the assumption. Therefore, $\mathcal{A}$ satisfies the sequence contradicting that $\neg\phi$ is unsatisfiable. □

Corollary 2.4.7 (Propositional Tableau generates Models)**.** Let ϕ be a formula, $\{(\phi)\} \Rightarrow_{\text{T}}^{*} N$ 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.

Proof. See Exercise **??**. □

Consider the example tableau shown in Figure 2.5. The open first branch corresponds to the valuation $\mathcal{A} = \{P, R, \neg Q\}$ which is a model of the formula $\neg[(P \wedge \neg(Q \vee \neg R)) \rightarrow (Q \wedge R)]$.

The tableau calculus naturally evolves out of the semantics of the operators. However, from a proof search and proof length point of view it has severe deficits. Consider, for example, the abstract tableau in Figure 2.6. Let's assume it is closed. Let's further assume that the closedness does not depend on the K_j, K'_j literals. Then there is an exponentially smaller closed tableau for the formula that consists of picking exactly one of the identical L_i, L'_i subtrees. The calculus does not "learn" from the fact that closedness does not depend on the K_j, K'_j literals. Actually, this can be overcome and one way of looking at CDCL, Section **??**, is to consider it as a solution to the problem of unnecessary repetitions of already closed branches. Concerning proof length, there are clause sets where an exponential blow up compared to resolution, Section 2.6, or CDCL, Section **??**, cannot be prevented. For example, on a clause set where every clause rules out exactly one valuation of n variables, the shortest resolution proof is exponentially shorter than the shortest tableau proof. In addition, the resolution proof can be found in a deterministic way by simplification, see Example 2.6.4. For two variables the respective clause set is $(P \vee Q) \wedge (P \vee \neg Q) \wedge (\neg P \vee Q) \wedge (\neg P \vee \neg Q)$.

2.5 Normal Forms

In order to check the status of a formula ϕ via truth tables, the truth table contains a column for each subformula of ϕ and all valuations for its variables. Any shape of ϕ is fine in order to generate the respective truth table. The superposition calculus (Section 2.7), The DPLL calculus (Section 2.8), and the CDCL (Conflict Driven Clause Learning) calculus (Section **??**) all operate on a normal form, i.e., the shape of ϕ is restricted. All those calculi accept only conjunctions of disjunctions of literals, a particular *normal form*. It is called *Clause Normal Form* or simply *CNF*. The purpose of this section is to show that an arbitrary formula ϕ can be effectively and efficiently transformed into a formula in CNF, preserving at least satisfiability. Efficient transformations are typically not equivalence preserving because they introduce fresh propositional variables. Superposition, DPLL, and CDCL are all refutational calculi, so a satisfiability preserving normal form transformation is fine.

2.5.1 Conjunctive and Disjunctive Normal Forms

Both conjunctive and disjunctive normal forms only use the operators $\wedge$ and $\vee$ on top of literals. So all other operators need to be translated into a combination of $\wedge$, $\vee$ and $\neg$ and eventually negations have to be pushed downwards the formula in front of atoms. The crucial operator is an equivalence $\leftrightarrow$, because a formula $\phi \leftrightarrow \psi$ is logically equivalent to the formula $(\neg\phi \vee \psi) \wedge (\neg\psi \vee \phi)$. However, in the letter formula the occurrences of ϕ and ψ have been duplicated. Replacing a formula of nested $\leftrightarrow$ occurrences that way results therefore in an exponentially larger formula.

A CNF is a conjunction of disjunction of literals, e.g., a formula $(P \vee \neq Q)(P \vee R)$. A formula containing only the operators $\wedge$, $\vee$ and literals can always be transformed into a conjunction of disjunctions via the application of the distributivity law. For example the formula $\phi \vee (\psi_1 \wedge \psi_2)$ results in $(\phi \vee \psi_1) \wedge (\phi \vee \psi_2)$ after pushing this disjunction inside. Again, similar to the effect of replacing an equivalence, the formula ϕ is duplicated. Turning a deep nesting of $\wedge$ operators below $\vee$ operators may therefore also result in an exponentially larger formula. A dual property holds for the disjunctive normal form.

In the sequel I'll define the respective normal forms and present various calculi and algorithms for normal form transformations. The more sophisticated algorithms, Algorithm 3, Algorithm 4, transform any formula into a satisfiability preserving CNF in linear time.

Definition 2.5.1 (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.

The definition of the propositional language, Definition 2.1.1, considers only binary conjunctions and disjunctions,. Both operators are AC (Associative and Commutative) thus an n-ary usage of the operators as well as a set notation is compatible with the semantics. Actually, I will use all three notations, binary operators, n-ary operators as well as set notations interchangeably, whatever fits best in the respective context.

So a CNF has the form $\bigwedge_i \bigvee_j L_j$ and a DNF the form $\bigvee_i \bigwedge_j L_j$ where the L_j are literals. In the sequel the logical notation with $\vee$ is overloaded with a multiset notation. Both the disjunction $L_1 \vee \ldots \vee L_n$ and the multiset $\{L_1, \ldots, L_n\}$ are clauses. For clauses the letters C, D, possibly indexed are used. Furthermore, a conjunction of clauses is considered as a set of clauses. Then, for a set of clauses, the empty set denotes $\top$. For a clause, the empty multiset denotes $\emptyset$ and at the same time $\bot$.

Although CNF and DNF are defined in almost any text book on automated reasoning, the definitions in the literature differ with respect to the "border" cases: (i) are complementary literals permitted in a clause? (ii) are duplicated literals permitted in a clause? (iii) are empty disjunctions/conjunctions permitted? The above Definition 2.5.1 answers all three questions with "yes". A clause containing complementary literals is valid, as in $P \vee Q \vee \neg P$. Duplicate literals may occur, as in $P \vee Q \vee P$. The empty disjunction is $\bot$ and the empty conjunction $\top$, i.e., the empty disjunction is always false while the empty conjunction is always true.

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 $\neg P$ (see Exercise **??**).

On the other hand, checking the unsatisfiability of CNF formulas or the validity of DNF formulas is coNP-complete. For any propositional formula ϕ there is an equivalent formula in CNF and DNF and I will prove this below by actually providing an effective procedure for the transformation. However, also because of the above comment on validity and satisfiability checking for CNF and DNF formulas, respectively, the transformation is costly. In general, a CNF or DNF of a formula ϕ is exponentially larger than ϕ as long as the normal forms need to be logically equivalent. If this is not needed, then by the introduction of fresh propositional variables, CNF normal forms for ϕ can be computed in linear time in the size of ϕ. More concretely, given a formula ϕ instead of checking validity the unsatisfiability of $\neg\phi$ can be considered. Then the linear time CNF normal form algorithm (see Section 2.5.3) is satisfiability preserving, i.e., the linear time CNF of $\neg\phi$ is unsatisfiable iff $\neg\phi$ is.

Proposition 2.5.2. For every formula there is an equivalent formula in CNF and also an equivalent formula in DNF.

Proof. See the rewrite systems $\Rightarrow_{\mathrm{BCNF}}$, and $\Rightarrow_{\mathrm{ACNF}}$ below and the lemmata on their properties. □

2.5.2 Basic CNF/DNF Transformation

The below algorithm bcnf is a basic algorithm for transforming any propositional formula into CNF, or DNF if the rule **PushDisj** is replaced by **PushConj**.

Algorithm 2: bcnf(ϕ)

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

```
1 whilerule (ElimEquiv(φ)) do ;
2 whilerule (ElimImp(φ)) do ;
3 whilerule (ElimTB1(φ),...,ElimTB6(φ)) do ;
4 whilerule (PushNeg1(φ),...,PushNeg3(φ)) do ;
5 whilerule (PushDisj(φ)) do ;
6 return φ;
```

In the sequel I study only the CNF version of the algorithm. All properties hold in an analogous way for the DNF version. To start an informal analysis of the algorithm, consider the following example CNF transformation.

Example 2.5.3. Consider the formula $\neg((P \vee Q) \leftrightarrow (P \rightarrow (Q \wedge \top)))$ and the application of $\Rightarrow_{\mathrm{BCNF}}$ depicted in Figure 2.8. Already for this simple formula the CNF transformation via $\Rightarrow_{\mathrm{BCNF}}$ becomes quite messy. Note that the CNF result in Figure 2.8 is highly redundant. If I remove all disjunctions that are trivially true, because they contain a propositional literal and its negation, the result becomes

$$(P \vee \neg Q) \wedge (\neg Q \vee \neg P) \wedge (\neg Q \vee \neg Q)$$

now elimination of duplicate literals beautifies the third clause and the overall formula into

$$(P \vee \neg Q) \wedge (\neg Q \vee \neg P) \wedge \neg Q.$$

Now let's inspect this formula a little closer. Any valuation satisfying the formula must set $\mathcal{A}(Q) = 0$, because of the third clause. But then the first two clauses are already satisfied. The formula $\neg Q$ *subsumes* the formulas $P \vee \neg Q$ and $\neg Q \vee \neg P$ in this sense. The notion of subsumption will be discussed in detail for clauses in Section 2.6. So it is eventually equivalent to

$$\neg Q.$$

The correctness of the result is obvious by looking at the original formula and doing a case analysis. For any valuation $\mathcal{A}$ with $\mathcal{A}(Q) = 1$ the two parts of the equivalence become true, independently of P, so the overall formula is false. For $\mathcal{A}(Q) = 0$, for any value of P, the truth values of the two sides of the equivalence are different, so the equivalence becomes false and hence the overall formula true.

After proving $\Rightarrow_{\mathrm{BCNF}}$ correct and terminating, in the succeeding section, Section 2.5.3, I will present an algorithm $\Rightarrow_{\mathrm{ACNF}}$ that actually generates a much more compact CNF out of $\neg((P \vee Q) \leftrightarrow (P \rightarrow (Q \wedge \top)))$ and does this without generating the mess of formulas $\Rightarrow_{\mathrm{BCNF}}$ does, see Figure 2.10. Applying