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 $\phi$ requires $O(2^n)$ steps, where $n$ is the number of different propositional variables in $\phi$. 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 $\phi$ and $\psi$ sharing the same propositional variables is exactly the same: we enumerate all valuations. However, if $\phi$ is, e.g., of the form $\phi = P \wedge \phi'$ and we are interested in the satisfiability of $\phi$, then $\phi$ can only become true for a valuation $\mathcal{A}$ with $\mathcal{A}(P) = 1$. Hence, $2^{n-1}$ rows of $\phi$'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 [19] and Smullyan [29] on the basis of work by Gentzen in the 30s [13] and of Beth [3] in the 50s. For an at that time state of the art overview consider Fitting's book [12].

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 $\alpha$- and $\beta$-formulas, where intuitively an $\alpha$ formula represents an (implicit) conjunction and a $\beta$ formula an (implicit) disjunction.

**Definition 2.4.1** ($\alpha$-, $\beta$-Formulas)**.** A formula $\phi$ is called an $\alpha$-formula if $\phi$ 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 $\phi$ is called a $\beta$-formula if $\phi$ 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 $\alpha$-, $\beta$-formulas is that they can be decomposed into direct descendants representing (modulo negation) subformulas of the respective

formulas. Then an $\alpha$-formula is valid iff all its descendants are valid and a $\beta$-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 $\alpha$- or $\beta$-formula $\phi$, Figure 2.4 shows its direct descendants.

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

**Proposition 2.4.3.** For any valuation $\mathcal{A}$: (i) if $\phi$ is an $\alpha$-formula then $\mathcal{A}(\phi) = 1$ iff $\mathcal{A}(\phi_1) = 1$ and $\mathcal{A}(\phi_2) = 1$ for its descendants $\phi_1$, $\phi_2$. (ii) if $\phi$ is a $\beta$-formula then $\mathcal{A}(\phi) = 1$ iff $\mathcal{A}(\phi_1) = 1$ or $\mathcal{A}(\phi_2) = 1$ for its descendants $\phi_1$, $\phi_2$.

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 $\phi_i$ and $\phi_j$ 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 $\phi$ is valid iff the rules of tableau produce a closed state starting with $N = \{(\neg\phi)\}$.

A formula $\phi$ occurring in some sequence is called *open* if in case $\phi$ is an $\alpha$-formula not both direct descendants are already part of the sequence and if it is a $\beta$-formula none of its descendants is part of the sequence.

**$\alpha$-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 $\psi$ is an open $\alpha$-formula, $\psi_1$, $\psi_2$ its direct descendants and the sequence is not closed.

**$\beta$-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 $\psi$ is an open $\beta$-formula, $\psi_1$, $\psi_2$ 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 $\beta$-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 $\phi$ the tableau calculus computes $\{(\neg\phi)\} \Rightarrow_{\mathrm{T}}^{*} N$ and $N$ is closed, then $\phi$ 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. $\alpha$-Expansion: for any valuation $\mathcal{A}$ if $\mathcal{A}(\psi) = 1$ then $\mathcal{A}(\psi_1) = \mathcal{A}(\psi_2) = 1$. $\beta$-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 $\phi$, 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 $\mu$. It is first defined on formulas by $\mu(\phi) := (n, k, l)$ where $n$ is the number of equivalence symbols in $\phi$, $k$ is the sum of all disjunction, conjunction, implication symbols in $\phi$ 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 $\phi_i$ is open in the sequence and $t_i = (0, 0, 0)$ otherwise. Finally, $\mu$ is extended to states $N$ by computing the multiset $\mu(N) := \{\mu(s) \mid s \in N\}$.

Note, that $\alpha$-, as well as $\beta$-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 $\alpha$-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 $\mu$ 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 $\psi_i$ are subformulas and the latter because $l \neq 0$. This proves the case.

A $\beta$-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 $\mu$ 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 $\psi_i$ are subformulas and the latter because $l \neq 0$. This proves the case. □

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

*Proof.* If $\phi$ 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 $\alpha$-formula or a $\beta$-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 $\phi$ be a formula, $\{(\phi)\} \Rightarrow_{\text{T}}^{*} N$ and $s \in N$ be a sequence that is not closed and neither $\alpha$-expansion nor $\beta$-expansion are applicable to $s$. Then the literals in $s$ form a (partial) valuation that is a model for $\phi$.

*Proof.* A consequence of the proof of Theorem 2.4.6 □

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 2.9, 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 2.9, 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 2.6.3. 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

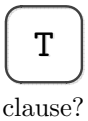
In order to check the status of a formula $\phi$ via truth tables, the truth table contains a column for each subformula of $\phi$ and all valuations for its variables. Any shape of $\phi$ is fine in order to generate the respective truth table. The superposition calculus (Section 2.7), The DPLL calculus (Section 2.8), and the CDCL (Conflict Driven Clause Learning) calculus (Section 2.9) all operate on a normal form, i.e., the shape of $\phi$ 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 $\phi$ can be effectively and efficiently transformed into an 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

**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.

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 $\phi$ 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 $\phi$ is exponentially larger than $\phi$ 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 $\phi$ can be computed in linear time in the size of $\phi$. More concretely, given a formula $\phi$ 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 $\Rightarrow_{\text{BCNF}}$, and $\Rightarrow_{\text{ACNF}}$ 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($\phi$)

**Input** : A propositional formula $\phi$.
**Output**: A propositional formula $\psi$ equivalent to $\phi$ 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 $\Rightarrow_{\text{BCNF}}$ depicted in Figure 2.8. Already for this simple formula the CNF transformation via $\Rightarrow_{\text{BCNF}}$ 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 $\Rightarrow_{\text{BCNF}}$ correct and terminating, in the succeeding section, Section 2.5.3, I will present an algorithm $\Rightarrow_{\text{ACNF}}$ 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 $\Rightarrow_{\text{BCNF}}$ does, see Figure 2.10. Applying standard redundancy elimination rules Tautology Deletion, Condensation, and Subsumption, see Section 2.6 and Section 2.7, then actually generates $\neg Q$ as the overall result. Please recall that the above rules apply modulo commutativity of $\vee$, $\wedge$, e.g., the rule ElimTB1 is both applicable to the formulas $\phi \wedge \top$ and $\top \wedge \phi$.

The equivalences in Figure 2.1 suggest more potential for simplification. For example, the idempotency equivalences $(\phi \wedge \phi) \leftrightarrow \phi$, $(\phi \vee \phi) \leftrightarrow \phi$ can be turned into simplification rules by applying them left to right. However, the way they are stated they can only be applied in case of identical subformulas. The formula $(P \vee Q) \wedge (Q \vee P)$ does this way not reduce to $(Q \vee P)$. A solution is to consider identity modulo commutativity. But then identity modulo commutativity and associativity (AC) as in $((P \vee Q) \vee R) \wedge (Q \vee (R \vee P)$ is still not detected. On the other hand, in practice, checking identity modulo AC is often too expensive. An elegant way out of this situation is to implement AC connectives like $\vee$ or $\wedge$ with flexible arity, to normalize nested occurrences of the connectives, and finally to sort the arguments using some total ordering. Applying this to $((P \vee Q) \vee R) \wedge (Q \vee (R \vee P)$ with ordering $R > P > Q$ the result is $(Q \vee P \vee R) \wedge (Q \vee P \vee R)$. Now complete AC simplification is back at the cost of checking for identical subformulas. Note that in an appropriate implementation, the normalization and ordering process is only done once at the start and then normalization and argument ordering is kept as an invariant.

### 2.5.3 Advanced CNF Transformation

The simple algorithm for CNF transformation Algorithm 2 can be improved in various ways: (i) more aggressive formula simplification, (ii) renaming, (iii) po-

larity dependant transformations. The before studied Example 2.5.3 serves already as a nice motivation for (i) and (iii). Firstly, removing $\top$ from the formula $\neg((P \vee Q) \leftrightarrow (P \rightarrow (Q \wedge \top)))$ first and not in the middle of the algorithm obviously shortens the overall process. Secondly, if the equivalence is replaced polarity dependant, i.e., using the equivalence $(\phi \leftrightarrow \psi) \leftrightarrow (\phi \wedge \psi) \vee (\neg\phi \wedge \neg\psi)$ and not the one used in rule ElimEquiv applied before, a lot of redundancy generated by $\Rightarrow_{\mathrm{BCNF}}$ is prevented. In general, if $\psi[\phi_1 \leftrightarrow \phi_2]_p$ and $\mathrm{pol}(\psi, p) = -1$ then for CNF transformation the equivalence is replaced by $\psi[(\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge \neg\phi_2)]_p$ and if $\mathrm{pol}(\psi, p) = 1$ by $\psi[(\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)]_p$ in $\psi$.

Item (ii) can be motivated by a formula

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

where Algorithm 2 generates a CNF with $2^{n-1}$ clauses out of this formula. The way out of this problem is the introduction of additional fresh propositional variables that *rename* subformulas. The price to pay is that a renamed formula is not equivalent to the original formula due to the extra propositional variables, but satisfiability preserving. A renamed formula for the above formula is

$$(P_1 \leftrightarrow (P_2 \leftrightarrow Q_1)) \wedge (Q_1 \leftrightarrow (P_3 \leftrightarrow Q_2)) \wedge \ldots$$

where the $Q_i$ are additional, fresh propositional variables. The number of clauses of the CNF of this formula is $4(n-1)$ where each conjunct $(Q_i \leftrightarrow (P_j \leftrightarrow Q_{i+1}))$ contributes four clauses.

**Proposition 2.5.4.** Let $P$ be a propositional variable not occurring in $\psi[\phi]_p$.

1. If $\mathrm{pol}(\psi, p) = 1$, then $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \wedge (P \rightarrow \phi)$ is satisfiable.

2. If $\mathrm{pol}(\psi, p) = -1$, then $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \wedge (\phi \rightarrow P)$ is satisfiable.

3. If $\mathrm{pol}(\psi, p) = 0$, then $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \wedge (P \leftrightarrow \phi)$ is satisfiable.

*Proof.* Exercise. □

So depending on the formula $\psi$, the position $p$ where the variable $P$ is introduced, the definition of $P$ is given by

$$\mathrm{def}(\psi, p, P) := \begin{cases} (P \rightarrow \psi|_p) & \text{if } \mathrm{pol}(\psi, p) = 1 \\ (\psi|_p \rightarrow P) & \text{if } \mathrm{pol}(\psi, p) = -1 \\ (P \leftrightarrow \psi|_p) & \text{if } \mathrm{pol}(\psi, p) = 0 \end{cases}$$

C The polarity dependent definition of some predicate $P$ introduces fewer clauses in case $\mathrm{pol}(\psi, p)$ has polarity 1 or -1. Still, even if always an equivalence is used to define predicates, for a properly chosen renaming the number of eventually generated clauses remains polynomial. Depending on the afterwards used calculus the former or latter results in a typically smaller search space. If a calculus relies on an explicitly building a partial model, e.g., CDCL, Section 2.9 and Section 2.10, then always defining predicates via equivalences is to be preferred. It guarantees that once the valuation of all variables in $\psi|_p$ is determined, also the value $P$ is determined by propagation. If a calculus relies on building inferences in a syntactic way, e.g., Resolution, Section 2.6 and Section 2.12, then using a polarity dependent definition of $P$ results in fewer inference opportunities.

For renaming there are several choices which subformula to choose. Obviously, since a formula has only linearly many subformulas, renaming every subformula works [30, 25]. However, this produces a number of renamings that do even increase the size of an eventual CNF. For example renaming in $\psi[\neg\phi]_p$ the subformulas $\neg\phi$ and $\phi$ at positions $p$, $p1$, respectively, produces more clauses than just renaming one position out of the two. This will be captured below by the notion of an *obvious position*. Then, in the following section a renaming variant is introduced that actually produces smallest CNFs. For all variants, renaming relies on a set of positions $\{p_1, \ldots, p_n\}$ that are replaced by fresh propositional variables.

**SimpleRenaming** $\phi \Rightarrow_{\mathrm{SimpRen}} \phi[P_1]_{p_1}[P_2]_{p_2} \ldots [P_n]_{p_n} \wedge \mathrm{def}(\phi, p_1, P_1) \wedge \ldots \wedge \mathrm{def}(\phi[P_1]_{p_1}[P_2]_{p_2} \ldots [P_{n-1}]_{p_{n-1}}, p_n, P_n)$
provided $\{p_1, \ldots, p_n\} \subset \mathrm{pos}(\phi)$ and for all $i, i+j$ either $p_i \parallel p_{i+j}$ or $p_i > p_{i+j}$ and the $P_i$ are different and new to $\phi$

The term $\phi[P_1]_{p_1}[P_2]_{p_2} \ldots [P_n]_{p_n}$ is evaluated left to right, i.e., a shorthand for $(\ldots((\phi[P_1]_{p_1})[P_2]_{p_2}) \ldots [P_n]_{p_n})$. Actually, the rule SimpleRenaming does not provide an effective way to compute the set $\{p_1, \ldots, p_n\}$ of positions in $\phi$ to be renamed. Where are several choices. Following Plaisted and Greenbaum [25], the set contains all positions from $\phi$ that do not point to a propositional variable or a negation symbol. In addition, renaming position $\epsilon$ does not make sense because it would generate the formula $P \wedge (P \rightarrow \phi)$ which results in more clauses than just $\phi$. Choosing the set of Plaisted and Greenbaum prevents the explosion in the number of clauses during CNF transformation. But not all renamings are needed to this end.

A smaller set of positions from $\phi$, called *obvious positions*, is still preventing the explosion and given by the rules: (i) $p$ is an obvious position if $\phi|_p$ is an equivalence and there is a position $q < p$ such that $\phi|_q$ is either an equivalence or disjunctive in $\phi$ or (ii) $pq$ is an obvious position if $\phi|_{pq}$ is a conjunctive formula in $\phi$, $\phi|_p$ is a disjunctive formula in $\phi$ and for all positions $r$ with $p < r < pq$ the formula $\phi|_r$ is not a conjunctive formula.

A formula $\phi|_p$ is conjunctive in $\phi$ if $\phi|_p$ is a conjunction and $\mathrm{pol}(\phi, p) \in \{0, 1\}$ or $\phi|_p$ is a disjunction or implication and $\mathrm{pol}(\phi, p) \in \{0, -1\}$. Analogously,

a formula $\phi|_p$ is disjunctive in $\phi$ if $\phi|_p$ is a disjunction or implication and $\mathrm{pol}(\phi, p) \in \{0, 1\}$ or $\phi|_p$ is a conjunction and $\mathrm{pol}(\phi, p) \in \{0, -1\}$.

**Example 2.5.5.** Consider as an example the formula

$$\phi = [\neg(\neg P \vee (Q \wedge R))] \rightarrow [P \vee (\neg Q \leftrightarrow \neg R)]\,.$$

Its tree representation as well as the polarity and position of each node is shown in Figure 2.9. Then the set of obvious positions is

$$\{22, 112\}$$

where 22 is obvious, because $\phi|_{22}$ is an equivalence and $\phi|_2$ is disjunctive, case (i) of the above definition. The position 112 is obvious, because it is conjunctive and $\phi|_{11}$ is a disjunctive formula, case (ii) of the above definition. Both positions are also considered by the Plaisted and Greenbaum definition, but they also add the positions $\{11, 2\}$ to this set, resulting in the set

$$\{2, 22, 11, 112\}.$$

Then applying SimpleRenaming to $\phi$ with respect to obvious positions results in

$$[\neg(\neg P \vee P_1)] \rightarrow [P \vee P_2] \wedge (P_1 \rightarrow (Q \wedge R)) \wedge (P_2 \rightarrow (\neg Q \leftrightarrow \neg R))$$

and applying SimpleRenaming with respect to the Plaisted Greenbaum positions results in

$$\begin{aligned}[\neg P_3] \rightarrow [P_4] \wedge (P_1 \rightarrow (Q \wedge R)) \wedge (P_2 \rightarrow (\neg Q \leftrightarrow \neg R)) \quad \wedge \\ (P_3 \rightarrow (\neg P \vee P_1)) \wedge (P_4 \rightarrow (P \vee P_2))\end{aligned}$$

where I applied in both cases a polarity dependent definition of the freshly introduced propoaitional variables. A CNF generated by bcnf out of the renamed formula using obvious positions results in 5 clauses, where the renamed formula using the Plaisted Greenbaum positions results in 7 clauses.

Formulas are naturally implemented by trees in the style of the tree in Figure 2.9. Every node contains the connective of the respective subtree and an array with pointers to its children. Optionally, there is also a back-pointer to the father of a node. Then a subformula at a particular position can be represented by a pointer to the respective subtree. The polarity or position of a subformula can either be a stored additionally in each node, or, if back-pointers are available, it can be efficiently computed by traversing all nodes up to the root.

The before mentioned polarity dependent transformations for equivalences are realized by the following two rules:

**ElimEquiv1** $\chi[(\phi \leftrightarrow \psi)]_p \;\Rightarrow_{\mathrm{ACNF}}\; \chi[(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)]_p$
provided $\mathrm{pol}(\chi, p) \in \{0, 1\}$

**ElimEquiv2** $\chi[(\phi \leftrightarrow \psi)]_p \Rightarrow_{\text{ACNF}} \chi[(\phi \wedge \psi) \vee (\neg\phi \wedge \neg\psi)]_p$
provided $\text{pol}(\chi, p) = -1$

Furthermore, the advanced algorithm eliminates $\top$ and $\bot$ before eliminating $\leftrightarrow$ and $\rightarrow$. Therefore the respective rules are added:

| | | | |
|---|---|---|---|
| **ElimTB7** | $\chi[\phi \rightarrow \bot]_p$ | $\Rightarrow_{\text{ACNF}}$ | $\chi[\neg\phi]_p$ |
| **ElimTB8** | $\chi[\bot \rightarrow \phi]_p$ | $\Rightarrow_{\text{ACNF}}$ | $\chi[\top]_p$ |
| **ElimTB9** | $\chi[\phi \rightarrow \top]_p$ | $\Rightarrow_{\text{ACNF}}$ | $\chi[\top]_p$ |
| **ElimTB10** | $\chi[\top \rightarrow \phi]_p$ | $\Rightarrow_{\text{ACNF}}$ | $\chi[\phi]_p$ |
| **ElimTB11** | $\chi[\phi \leftrightarrow \bot]_p$ | $\Rightarrow_{\text{ACNF}}$ | $\chi[\neg\phi]_p$ |
| **ElimTB12** | $\chi[\phi \leftrightarrow \top]_p$ | $\Rightarrow_{\text{ACNF}}$ | $\chi[\phi]_p$ |

where the two rules ElimTB11, ElimTB12 for equivalences are applied with respect to commutativity of $\leftrightarrow$.

**Algorithm 3:** acnf($\phi$)

**Input** : A formula $\phi$.
**Output**: A formula $\psi$ in CNF satisfiability preserving to $\phi$.

```
1 whilerule (ElimTB1(φ),...,ElimTB12(φ)) do ;
2 SimpleRenaming(φ) on obvious positions;
3 whilerule (ElimEquiv1(φ),ElimEquiv2(φ)) do ;
4 whilerule (ElimImp(φ)) do ;
5 whilerule (PushNeg1(φ),...,PushNeg3(φ)) do ;
6 whilerule (PushDisj(φ)) do ;
7 return φ;
```

For an implementation the Algorithm 3 can be further improved. For example, once equivalences are eliminated the polarity of each literal is exactly known. So eliminating implications and pushing negations inside is not needed. Instead the eventual CNF can be directly constructed from the formula.

**Proposition 2.5.6** (Models of Renamed Formulas)**.** Let $\phi$ be a formula and $\phi'$ a renamed CNF of $\phi$ computed by acnf. Then any (partial) model $\mathcal{A}$ of $\phi'$ is also a model for $\phi$.

*Proof.* By an inductive argument it is sufficient to consider one renaming application, i.e., $\phi' = \phi[P]_p \wedge \text{def}(\phi, p, P)$. There are three cases depending on the polarity. (i) if $\text{pol}(\phi, p) = 1$ then $\phi' = \phi[P]_p \wedge P \rightarrow \phi|_p$. If $\mathcal{A}(P) = 1$ then $\mathcal{A}(\phi|_p) = 1$ and hence $\mathcal{A}(\phi) = 1$. The interesting case is $\mathcal{A}(P) = 0$ and $\mathcal{A}(\phi|_p) = 1$. But then because $\text{pol}(\phi, p) = 1$ also $\mathcal{A}(\phi) = 1$ by Lemma 2.2.7. (ii) if $\text{pol}(\phi, p) = -1$ the case is symmetric to the previous one. Finally, (iii) if $\text{pol}(\phi, p) = 0$ for any $\mathcal{A}$ satisfying $\phi'$ it holds $\mathcal{A}(\phi|_p) = \mathcal{A}(P)$ and hence $\mathcal{A}(\phi) = 1$. □