Proof. A consequence of the proof of Theorem 2.5.6 □

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 $\neg[(P \wedge \neg(Q \vee \neg R)) \rightarrow (Q \wedge R)]$.

2.6 Normal Forms

In order to check the status of a formula ϕ via truth tables, the truth table contains a column for the subformulas of ϕ and all valuations for its variables. Any shape of ϕ is fine in order to generate the respective truth table. For the superposition calculus (Section 2.8) and the CDCL (Conflict Driven Clause Learning) calculus (Section 2.10) I introduce in the next two sections, the shape of ϕ is restricted. Both 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 transformed into an equivalent formula in CNF.

Definition 2.6.1 (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 L_j are literals. A disjunction of literals $L_1, \dots, L_n$ is called a *clause.*

$\boxed{\texttt{T}}$

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? For the above Definition 2.6.1 the answer is "yes" to all three questions. 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 2.12).

$\boxed{\texttt{C}}$

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 or DNF normal forms for ϕ can be computed in linear time in the size of ϕ. More concrete, 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 **??**) computes a satisfiability preserving formula, i.e., the linear time CNF of $\neg\phi$ is unsatisfiable iff $\neg\phi$ is.

Proposition 2.6.2. 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. □

2.6.1 Basic CNF/DNF Transformation

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

Algorithm 2: bcnf(ϕ)

```
   Input  : A propositional formula $\phi$.
   Output: A propositional formula $\psi$ equivalent to $\phi$ in CNF.
 1 whilerule (ElimEquiv($\phi$)) do ;
 2 ;
 3 whilerule (ElimImp($\phi$)) do ;
 4 ;
 5 whilerule (ElimTB1($\phi$),...,ElimTB6($\phi$)) do ;
 6 ;
 7 whilerule (PushNeg1($\phi$),...,PushNeg3($\phi$)) do ;
 8 ;
 9 whilerule (PushDisj($\phi$)) do ;
10 ;
11 return $\phi$;
```

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 2.6.3. Consider the formula $\neg((P \lor Q) \leftrightarrow (P \rightarrow (Q \land \top)))$ and the application of $\Rightarrow_{\mathrm{BCNF}}$ depicted in Figure 2.7. Already for this simple formula the CNF transformation via $\Rightarrow_{\mathrm{BCNF}}$ becomes quite messy. Note that the CNF result in Figure 2.7 is still highly redundant. If I remove all disjunctions that are trivially true, because they contain a propositional literal and its negation, the result becomes

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

Figure 2.6: Basic CNF/DNF Transformation Rules

$$(P \lor \neg Q) \lor (\neg Q \lor \neg P) \land (\neg Q \lor \neg Q)$$

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

$$(P \lor \neg Q) \lor (\neg Q \lor \neg P) \land \neg Q.$$

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 $\neq Q$ *subsumes* the formulas $P \lor \neg Q$ and $\neg Q \lor \neg P$ in this sense. The notion of subsumption will be discussed in detail for clauses in Section 2.7.

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 I will present an algorithm $\Rightarrow_{\mathrm{ACNF}}$ that actually generates $\neg Q$ out of $\neg((P \lor Q) \leftrightarrow (P \rightarrow (Q \land \top)))$ and does this without generating the mess of formulas $\Rightarrow_{\mathrm{BCNF}}$ does. Please recall that the above rules apply modulo commutativity of $\lor$, $\land$, e.g., the rule ElimTB1 is both applicable to the formulas $\phi \land \top$ and $\top \land \phi$.

$\neg((P \lor Q) \leftrightarrow (P \rightarrow (Q \land \top)))$
$\Rightarrow^{\text{Step 1}}_{\text{BCNF}}$ $\neg([(P \lor Q) \rightarrow (P \rightarrow (Q \land \top))] \land [(P \rightarrow (Q \land \top)) \rightarrow (P \lor Q)])$
$\Rightarrow^{\text{Step 2}}_{\text{BCNF}}$ $\neg([\neg(P \lor Q) \lor (P \rightarrow (Q \land \top))] \land [(P \rightarrow (Q \land \top)) \rightarrow (P \lor Q)])$
$\Rightarrow^{\text{Step 2}}_{\text{BCNF}}$ $\neg([\neg(P \lor Q) \lor (P \rightarrow (Q \land \top))] \land [\neg(P \rightarrow (Q \land \top)) \lor (P \lor Q)])$
$\Rightarrow^{\text{Step 2}}_{\text{BCNF}}$ $\neg([\neg(P \lor Q) \lor (P \rightarrow (Q \land \top))] \land [\neg(\neg P \lor (Q \land \top)) \lor (P \lor Q)])$
$\Rightarrow^{\text{Step 2}}_{\text{BCNF}}$ $\neg([\neg(P \lor Q) \lor (\neg P \lor (Q \land \top))] \land [\neg(\neg P \lor (Q \land \top)) \lor (P \lor Q)])$
$\Rightarrow^{\text{Step 3}}_{\text{BCNF}}$ $\neg([\neg(P \lor Q) \lor (\neg P \lor Q)] \land [\neg(\neg P \lor Q) \lor (P \lor Q)])$
$\Rightarrow^{\text{Step 4}}_{\text{BCNF}}$ $\neg([(\neg P \land \neg Q) \lor (\neg P \lor Q)] \land [\neg(\neg P \lor Q) \lor (P \lor Q)])$
$\Rightarrow^{\text{Step 4}}_{\text{BCNF}}$ $\neg([(\neg P \land \neg Q) \lor (\neg P \lor Q)] \land [(\neg\neg P \land \neg Q) \lor (P \lor Q)])$
$\Rightarrow^{\text{Step 4}}_{\text{BCNF}}$ $\neg([(\neg P \land \neg Q) \lor (\neg P \lor Q)] \land [(\neg\neg P \land \neg Q) \lor (P \lor Q)])$
$\Rightarrow^{*,\text{Step 4}}_{\text{BCNF}}$ $[(\neg\neg P \lor \neg\neg Q) \land (\neg\neg P \land \neg Q)] \lor [(\neg\neg\neg P \lor \neg\neg Q) \land (\neg P \land \neg Q)]$
$\Rightarrow^{*,\text{Step 4}}_{\text{BCNF}}$ $[(P \lor Q) \land (P \land \neg Q)] \lor [(\neg P \lor Q) \land (\neg P \land \neg Q)]$
$\Rightarrow^{*,\text{Step 5}}_{\text{BCNF}}$ $(P \lor Q \lor \neg P \lor Q) \land (P \lor Q \lor \neg P) \land (P \lor Q \lor \neg Q) \land (P \lor \neg P \lor$
$Q) \land (P \lor \neg P) \land (P \lor \neg Q) \land (\neg Q \lor \neg P \lor Q) \land (\neg Q \lor \neg P) \land (\neg Q \lor \neg Q)$

Figure 2.7: Example Basic CNF Transformation

I Figure 2.1 contains more potential for simplification. For example, the idempotency equivalences $(\phi \land \phi) \leftrightarrow \phi$, $(\phi \lor \phi) \leftrightarrow \phi$ 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 $(P \lor Q) \land (Q \lor P)$ does this way not reduce to $(Q \lor P)$. A solution is to consider identity modulo commutativity. But then identity modulo commutativity and associativity (AC) as in $((P \lor Q) \lor R) \land (Q \lor (R \lor P)$ 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 $\lor$ or $\land$ with flexible arity, to normalize nested occurrences of the connectives, and finally to sort the arguments using some total ordering. Applying this to $((P \lor Q) \lor R) \land (Q \lor (R \lor P)$ with ordering $R > P > Q$ the result is $(Q \lor P \lor R) \land (Q \lor P \lor R)$. 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.

2.6.2 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) polarity dependant transformations. The before studied Example 2.6.3 serves already as a nice motivation for (i) and (iii). Firstly, removing $\top$ from the formula $\neg((P \lor Q) \leftrightarrow (P \rightarrow (Q \land \top)))$ first and not at the end of the algorithm obviously shortens the overall process. Secondly, if the equivalence is replaced polarity dependant, i.e., using the equivalence $(\phi \leftrightarrow \psi) \leftrightarrow (\phi \land \psi) \lor (\neg\phi \land \neg\psi)$ and not the one used in rule ElimEquiv applied before, a lot of redundancy generated by $\Rightarrow_{\text{BCNF}}$ is prevented. In general, if $\psi[\phi_1 \leftrightarrow \phi_2]_p$ and $\text{pol}(\psi, p) = -1$ then for CNF transformation do $\psi[(\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge \neg\phi_2)]_p$ and if $\text{pol}(\psi, p) = 1$ do $\psi[(\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)]_p$

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

1. If $\text{pol}(\psi, p) = 1$, then $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \wedge (P \rightarrow \phi)$ is satisfiable.
2. If $\text{pol}(\psi, p) = -1$, then $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \wedge (\phi \rightarrow P)$ is satisfiable.
3. If $\text{pol}(\psi, p) = 0$, 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 ψ, the position p where the variable P is introduced definition of P is given by

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

For renaming there are several choices which subformula to choose. Obviously, since a formula has only linearly many subformulas, renaming every subformula works [20, 17]. Basically this is what I show below. In the following section a renaming variant is introduced that produces smallest CNFs.

SimpleRenaming $\phi \Rightarrow_{\text{SimpRen}} \phi[P_1]_{p_1}[P_2]_{p_2}\ldots[P_n]_{p_n} \wedge \text{def}(\phi, p_1, P_1) \wedge \ldots \wedge \text{def}(\phi[P_1]_{p_1}[P_2]_{p_2}\ldots[P_{n-1}]_{p_{n-1}}, p_n, P_n)$

provided $\{p_1, \ldots, p_n\} \subset \text{pos}(\phi)$ 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 ϕ

Actually, the rule SimpleRenaming does not provide an effective way to compute the set $\{p_1, \ldots, p_n\}$ of positions in ϕ to be renamed. Where are several choices. Following Plaisted and Greenbaum [17], the set contains all positions from ϕ that do not point to a propositional variable or a negation symbol. In addition, renaming position ϵ does not make sense because it would generate the formula $P \land (P \rightarrow \phi)$ which results in more clauses than just ϕ. 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 ϕ, let's call it the set of obvious positions, is still preventing the explosion and given by the rules: (i) if $\phi|_p$ is an equivalence and there is a position $q < p$ such that $\phi|_q$ is either an equivalence or disjunctive in ϕ then p is an obvious position (ii) if $\phi|_{pq}$ is a conjunctive formula in ϕ, $\phi|_p$ is a disjunctive formula in ϕ and for all positions r with $p < r < pq$ the formula $\phi|_r$ is not a conjunctive formula then pq is an obvious position. A formula $\phi|_p$ is conjunctive in ϕ 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 ϕ if $\phi|_p$ is a disjunction or implication and $\mathrm{pol}(\phi,p) \in \{0,1\}$ or $\phi|_p$ is a conjunction and $\mathrm{pol}(\phi,p) \in \{0,-1\}$.

Consider as an example the formula

$$[\neg(\neg P \lor (Q \land R))] \rightarrow [P \land (\neg Q \leftrightarrow \neg R)]$$

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

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

ElimEquiv2 $\chi[(\phi \leftrightarrow \psi)]_p \;\Rightarrow_{\mathrm{ACNF}}\; \chi[(\phi \land \psi) \lor (\neg\phi \land \neg\psi)]_p$
provided $\mathrm{pol}(\chi,p) = -1$

$$\begin{array}{l}
\neg((P \lor Q) \leftrightarrow (P \rightarrow (Q \land \top))) \\
\Rightarrow^{\text{Step 1}}_{\text{ACNF}} \ \neg((P \lor Q) \leftrightarrow (P \rightarrow Q)) \\
\Rightarrow^{\text{Step 3}}_{\text{ACNF}} \ \neg(((P \lor Q) \land (P \rightarrow Q)) \lor (\neg(P \lor Q) \land \neg(P \rightarrow Q))) \\
\Rightarrow^{*,\text{Step 4}}_{\text{ACNF}} \ \neg(((P \lor Q) \land (\neg P \lor Q)) \lor (\neg(P \lor Q) \land \neg(\neg P \lor Q))) \\
\Rightarrow^{*,\text{Step 5}}_{\text{ACNF}} \ ((\neg P \land \neg Q) \lor (P \land \neg Q)) \land ((P \lor Q) \lor (\neg P \lor Q)) \\
\Rightarrow^{*,\text{Step 6}}_{\text{ACNF}} \ (\neg P \lor P) \land (\neg P \lor \neg Q) \land (\neg Q \lor P) \land (\neg Q \lor \neg Q) \land (P \lor Q \lor \neg P \lor Q)
\end{array}$$

Figure 2.8: Example Advanced CNF Transformation

2.6.3 Renaming Optimized for small CNF

Here I suggest to rename a subformula if the eventual number of generated clauses by bcnf decreases after renaming [5, 16]. The below function ac computes