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) polarity 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 ψ.

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 ψ, the position p where the variable P is introduced, the definition of P is given by

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

The polarity dependent definition of some predicate P introduces fewer clauses in case $\text{pol}(\psi, p)$ 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 [81, 71]. 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 ϕ 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 $\phi \Rightarrow_{\text{SimpRen}} \phi[P_1]_{p_1}[P_2]_{p_2} \ldots [P_n]_{p_n} \wedge \text{def}(\phi, p_1, P_1) \wedge \ldots \wedge \text{def}(\phi[P_1]_{p_1}[P_2]_{p_2} \ldots [P_{n-1}]_{p_{n-1}}, p_n, P_n)$
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

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 ϕ to be renamed. Where are several choices. Following Plaisted and Greenbaum [71], 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 $P \wedge (P \rightarrow \phi)$ 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, 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 ϕ or (ii) pq is an obvious position, $q \neq \epsilon$, 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.

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

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 ϕ 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

$$[\neg P_3] \rightarrow [P_4] \wedge (P_1 \rightarrow (Q \wedge R)) \wedge (P_2 \rightarrow (\neg Q \leftrightarrow \neg R)) \quad \wedge \\ (P_3 \rightarrow (\neg P \vee P_1)) \wedge (P_4 \rightarrow (P \vee P_2))$$

where I applied in both cases a polarity dependent definition of the freshly introduced propositional 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 $\mathrm{pol}(\chi, p) \in \{0, 1\}$

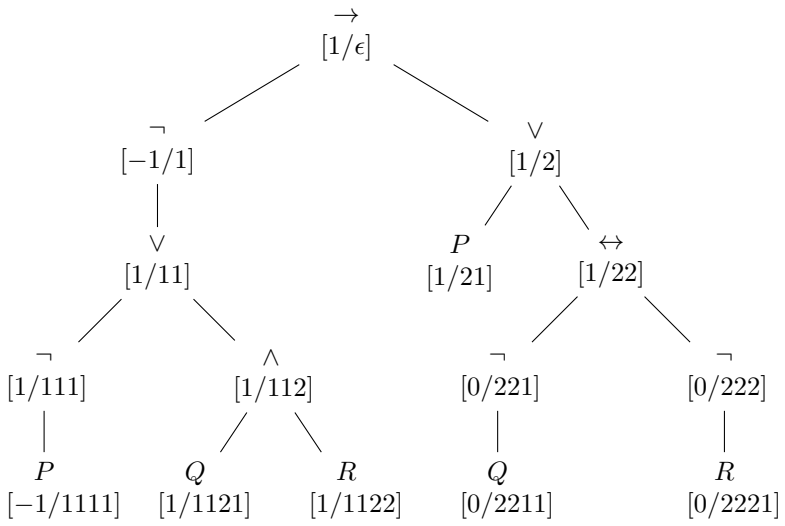


Figure 2.9: Tree representation of $[\neg(\neg P \vee (Q \wedge R))] \rightarrow [P \vee (\neg Q \leftrightarrow \neg R)]$ where each node is annotated with its [polarity/position].

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 $\leftrightarrow$.

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 ϕ be a formula and ϕ' a renamed CNF of ϕ computed by acnf. Then any (partial) model $\mathcal{A}$ of ϕ' is also a model for ϕ.

Algorithm 3: acnf(ϕ)

Input : A formula ϕ.
Output: A formula ψ in CNF satisfiability preserving to ϕ.

```
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;
```

$$
\begin{aligned}
&\neg((P \vee Q) \leftrightarrow (P \rightarrow (Q \wedge \top))) \\
&\Rightarrow^{\text{Step 1}}_{\text{ACNF}} \ \neg((P \vee Q) \leftrightarrow (P \rightarrow Q)) \\
&\Rightarrow^{\text{Step 3}}_{\text{ACNF}} \ \neg(((P \vee Q) \wedge (P \rightarrow Q)) \vee (\neg(P \vee Q) \wedge \neg(P \rightarrow Q))) \\
&\Rightarrow^{*,\text{Step 4}}_{\text{ACNF}} \ \neg(((P \vee Q) \wedge (\neg P \vee Q)) \vee (\neg(P \vee Q) \wedge \neg(\neg P \vee Q))) \\
&\Rightarrow^{*,\text{Step 5}}_{\text{ACNF}} \ ((\neg P \wedge \neg Q) \vee (P \wedge \neg Q)) \wedge ((P \vee Q) \vee (\neg P \vee Q)) \\
&\Rightarrow^{*,\text{Step 6}}_{\text{ACNF}} \ (\neg P \vee P) \wedge (\neg P \vee \neg Q) \wedge (\neg Q \vee P) \wedge (\neg Q \vee \neg Q) \wedge (P \vee Q \vee \neg P \vee Q)
\end{aligned}
$$

Figure 2.10: Example Advanced CNF Transformation

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 2.2.8. (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 ϕ' it holds $\mathcal{A}(\phi|_p) = \mathcal{A}(P)$ and hence $\mathcal{A}(\phi) = 1$. □

Note that Proposition 2.5.6 does not hold the other way round. Whenever a formula is manipulated by introducing fresh symbols, the truth of the original formula does not depend on the truth of the fresh symbols. For example, consider the formula

$$\phi \vee \psi$$

which is renamed to

$$\phi \vee P \wedge P \rightarrow \psi$$

.

Then any interpretation $\mathcal{A}$ with $\mathcal{A}(\phi) = 1$ is a model for $\phi \vee \psi$. It is not necessarily a model for $\phi \vee P \wedge P \rightarrow \psi$. If $\mathcal{A}(P) = 1$ and $\mathcal{A}(\psi) = 0$ it does not satisfy $\phi \vee P \wedge P \rightarrow \psi$.

$$
\begin{array}{ll}
& \neg((P \vee Q) \leftrightarrow (P \rightarrow (Q \wedge \top))) \\
\Rightarrow_{\mathrm{OCNF}}^{\text{Step 1}} & \neg([(P \vee Q) \leftrightarrow (P \rightarrow Q)]) \\
\Rightarrow_{\mathrm{OCNF}}^{\text{Step 3}} & \neg([(P \vee Q) \wedge (P \rightarrow Q)] \vee [\neg(P \vee Q) \wedge \neg(P \rightarrow Q)]) \\
\Rightarrow_{\mathrm{OCNF}}^{\text{Step 2}} & \neg([(P \vee Q) \wedge (\neg P \vee Q)] \vee [\neg(P \vee Q) \wedge \neg(\neg P \vee Q)]) \\
\Rightarrow_{\mathrm{OCNF}}^{*,\text{Step 3}} & (\neg[(P \vee Q) \wedge (\neg P \vee Q)] \wedge \neg[\neg(P \vee Q) \wedge \neg(\neg P \vee Q)]) \\
\Rightarrow_{\mathrm{OCNF}}^{*,\text{Step 3}} & [\neg(P \vee Q) \vee \neg(\neg P \vee Q)] \wedge [(P \vee Q) \vee (\neg P \vee Q)] \\
\Rightarrow_{\mathrm{OCNF}}^{*,\text{Step 3}} & [(\neg P \wedge \neg Q) \vee (P \wedge \neg Q)] \wedge [(P \vee Q) \vee (\neg P \vee Q)] \\
\Rightarrow_{\mathrm{OCNF}}^{*,\text{Step 4}} & [(\neg P \vee P) \wedge (\neg P \vee \neg Q) \wedge (\neg Q \vee P) \wedge (\neg Q \vee \neg Q)] \wedge [P \vee Q \vee \neg P \vee Q]
\end{array}
$$

Figure 2.15: Example Optimized CNF Transformation

2.6 Propositional Resolution

The propositional resolution calculus operates on a set of clauses and tests unsatisfiability. This enables advanced CNF transformation and, in particular, renaming, see Section 2.5.3. In order to check validity of a formula ϕ we check unsatisfiability of the clauses generated from $\neg\phi$.

Recall, see Section 2.1, that for clauses I switch between the notation as a disjunction, e.g., $P \vee Q \vee P \vee \neg R$, and the multiset notation, e.g., $\{P, Q, P, \neg R\}$. This makes no difference as we consider $\vee$ in the context of clauses always modulo AC. Note that $\bot$, the empty disjunction, corresponds to $\emptyset$, the empty multiset. Clauses are typically denoted by letters C, D, possibly with subscript.

The *resolution calculus* consists of the inference rules *Resolution* and *Factoring*. So, if we consider clause sets N as states, $\uplus$ is disjoint union, we get the inference rules

$$\textbf{Resolution} \quad (N \uplus \{C_1 \vee P, C_2 \vee \neg P\}) \Rightarrow_{\mathrm{RES}} (N \cup \{C_1 \vee P, C_2 \vee \neg P\} \cup \{C_1 \vee C_2\})$$

$$\textbf{Factoring} \quad (N \uplus \{C \vee L \vee L\}) \Rightarrow_{\mathrm{RES}} (N \cup \{C \vee L \vee L\} \cup \{C \vee L\})$$

Theorem 2.6.1. The resolution calculus is sound and complete:
N is unsatisfiable iff $N \Rightarrow_{\mathrm{RES}}^{*} N'$ and $\bot \in N'$ for some N'

Proof. ($\Leftarrow$) Soundness means for all rules that $N \models N'$ where N' is the clause set obtained from N after applying Resolution or Factoring. For Resolution it is sufficient to show that $C_1 \vee P, C_2 \vee \neg P \models C_1 \vee C_2$. This is obvious by a case analysis of valuations satisfying $C_1 \vee P, C_2 \vee \neg P$: if P is true in such a valuation so must be C_2, hence $C_1 \vee C_2$. If P is false in some valuation then C_1 must be true and so $C_1 \vee C_2$. Soundness for Factoring is obvious this way because it simply removes a duplicate literal in the respective clause.

($\Rightarrow$) The traditional method of proving resolution completeness are *semantic trees*. A *semantic tree* is a binary tree where the edges are labeled with literals such that: (i) edges of children of the same parent are labeled with L and $\mathrm{comp}(L)$, (ii) any node has either no or two children, and (iii) for any path from

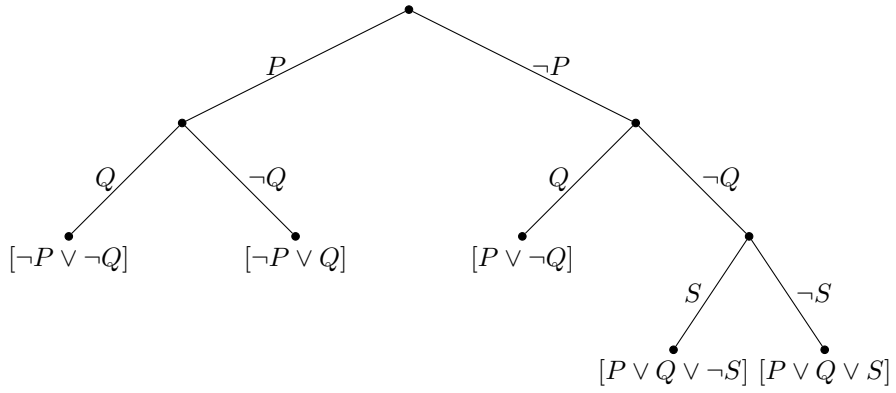
the root to a leaf, each propositional variable occurs at most once. Therefore, each path corresponds to a partial valuation. Now for an unsatisfiable clause set N there is a finite semantic tree such that for each leaf of the tree there is a clause from N that is false with respect to the partial valuation at that leaf. By structural induction on the size of the tree we prove completeness. If the tree consists of the root node, then $\bot \in N$. Now consider two sister leaves of the same parent of this tree, where the edges are labeled with L and $\mathrm{comp}(L)$, respectively. Let C_1 and C_2 be the two false clauses at the respective leaves. If some C_i does neither contain L or $\mathrm{comp}(L)$ then C_i is also false at the parent, finishing the case. So assume both C_1 and C_2 contain L or $\mathrm{comp}(L)$: $C_1 = C_1' \vee L$ and $C_2 = C_2' \vee \neg L$. If C_1 (or C_2) contains further occurrences of L (or C_2 of $\mathrm{comp}(L)$), then the rule Factoring is applied to eventually remove all additional occurrences. Therefore, eventually $L \notin C_1'$ and $\mathrm{comp}(L) \notin C_2'$. Note that if some C_i contains both L and $\mathrm{comp}(L)$, the clause is a tautology, contradicting the assumption that C_i is false at its leaf. A resolution step between these two clauses on L yields $C_1' \vee C_2'$ which is false at the parent of the two leaves, because the resolvent neither contains L nor $\mathrm{comp}(L)$. Furthermore, the resulting tree is smaller, proving completeness. □

In the proof of Theorem 2.6.1 it is not required that the semantic tree for some clause set is minimal. Instead, in case it is not minimal, one of the leaf clauses is simply moved to the parent level and the tree shrinks. The proof can also be done using minimal semantic trees. A semantic tree is *minimal* if no clause can be moved upwards without violating a semantic tree property. However, this complicates the proof a lot, because after a resolution step, the resulting semantic tree is not guaranteed to be minimal anymore. Sometimes minimality assumptions help in proving completeness, see the completeness proof for propositional superposision, Section 2.7, but sometimes they complicate proofs a lot.

Example 2.6.2 (Resolution Refutation Showing the Respective Semantic Tree)**.** Consider the clause set

$$N_0 = \{\neg P \vee Q,\ P \vee \neg Q,\ \neg P \vee \neg Q,\ P \vee Q \vee S,\ P \vee Q \vee \neg S\}$$

and the below sequence of semantic trees and resolution steps. The leaves are always labeled with clauses that are falsified at the respective partial valuation:



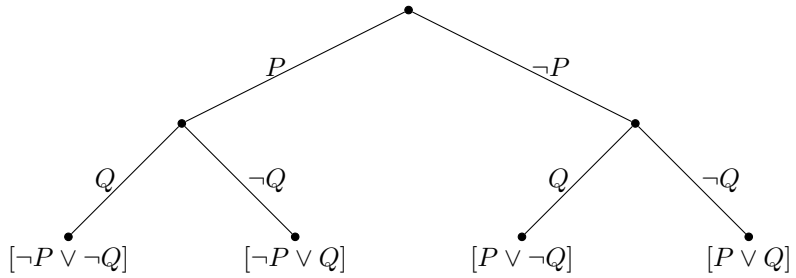
The first inference cuts the rightmost branch

1 $N_0 \Rightarrow_{\text{RES}} N_0 \cup \{P \vee P \vee Q \vee Q\}$

by resolving on literal S. The clause set of the i^{th} inference is always referred N_i, e.g., the above resulting clause set is $N_1 = N_0 \cup \{P \vee P \vee Q \vee Q\}$. The duplicate literals can be eliminated by two factoring steps.

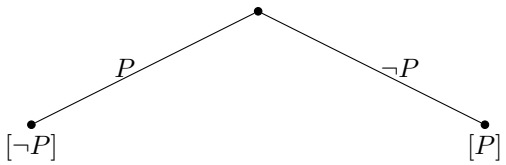
2 $N_1 \Rightarrow_{\text{RES}} N_1 \cup \{P \vee Q \vee Q\}$
3 $N_2 \Rightarrow_{\text{RES}} N_2 \cup \{P \vee Q\}$

and the semantic tree is cut using the clause $P \vee Q$.



The next inferences result in cuts to both the left branch and the right branch by resolving on the respective Q literals and removing resulting duplicate literal occurrences by Factoring applications.

4 $N_3 \Rightarrow_{\text{RES}} N_3 \cup \{\neg P \vee \neg P\}$
5 $N_4 \Rightarrow_{\text{RES}} N_4 \cup \{\neg P\}$
6 $N_5 \Rightarrow_{\text{RES}} N_5 \cup \{P \vee P\}$
7 $N_6 \Rightarrow_{\text{RES}} N_6 \cup \{P\}$



Finally, a resolution step between the clauses P and $\neg P$ yields the empty clause $\bot$.

Example 2.6.3 (Resolution Completeness)**.** The semantic tree for the clause set

$$P \vee Q \vee S,\ \neg P \vee Q \vee S,\ P \vee \neg Q \vee S,\ \neg P \vee \neg Q \vee S,$$
$$P \vee Q \vee \neg S,\ \neg P \vee Q \vee \neg S,\ P \vee \neg Q \vee \neg S,\ \neg P \vee \neg Q \vee \neg S$$

is shown in Figure 2.16.

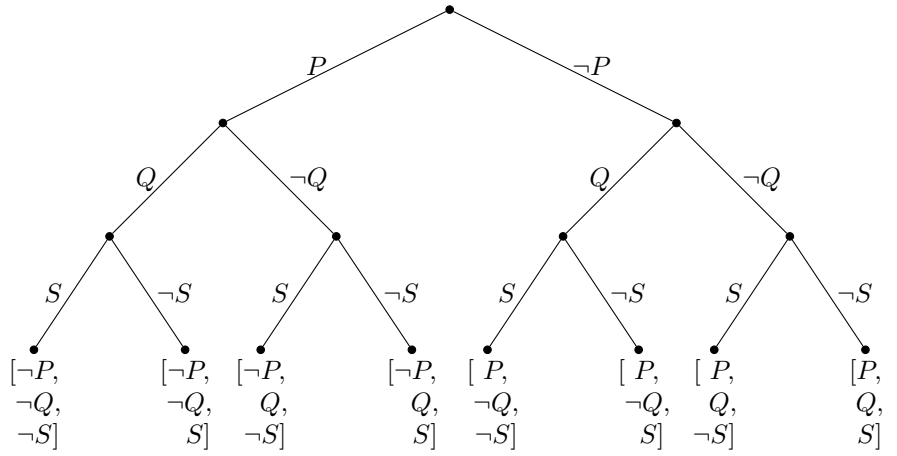


Figure 2.16: Semantic tree representation of $\{P \vee Q,\ \neg P \vee Q,\ P \vee \neg Q,\ \neg P \vee \neg Q \vee S,\ \neg P \vee \neg Q \vee \neg S\}$ where each leaf is labeled with the literals that falsify the partial valuation at that leaf.

The resolution calculus is complete just by using Resolution and Factoring. But the rules always extend a clause set. It gets larger both with respect to the number of clauses and the overall number of literals. It is practically very important to keep clause sets small. Therefore, so called *reduction rules* have been invented that actually reduce a clause set with respect to the number of clauses or overall number of literals.

The crucial question is whether adding such rules preserves completeness. This can become non-obvious. For the resolution calculus, the below rules are commonly used.

Subsumption $(N \uplus \{C_1, C_2\}) \Rightarrow_{\mathrm{RES}} (N \cup \{C_1\})$
provided $C_1 \subset C_2$

Tautology Deletion $(N \uplus \{C \vee P \vee \neg P\}) \Rightarrow_{\mathrm{RES}} (N)$

Condensation $(N \uplus \{C_1 \vee L \vee L\}) \Rightarrow_{\mathrm{RES}} (N \cup \{C_1 \vee L\})$

Subsumption Resolution where $C_1 \subseteq C_2$ $(N \uplus \{C_1 \vee L, C_2 \vee \mathrm{comp}(L)\}) \Rightarrow_{\mathrm{RES}} (N \cup \{C_1 \vee L, C_2\})$

Note the different nature of inference rules and reduction rules. Resolution and Factorization only add clauses to the set whereas Subsumption, Tautology Deletion and Condensation delete clauses or replace clauses by "simpler" ones. In the next section, Section 2.7, I will show what "simpler" means. For the resolution calculus, the semantic tree proof can actually be reformulated incorporating the four reduction rules, see Exercise **??**.

Example 2.6.4 (Refutation by Simplification)**.** Consider the clause set

$$N = \{P \vee Q,\ P \vee \neg Q,\ \neg P \vee Q,\ \neg P \vee \neg Q\}$$

that can be deterministically refuted by Subsumption Resolution:

$$\begin{aligned}
&(\{P \vee Q,\ P \vee \neg Q,\ \neg P \vee Q,\ \neg P \vee \neg Q\})\\
\Rightarrow^{\mathrm{SubRes}}_{\mathrm{RES}} \quad &(\{P \vee Q,\ P,\ \neg P \vee Q,\ \neg P \vee \neg Q\})\\
\Rightarrow^{\mathrm{SubRes}}_{\mathrm{RES}} \quad &(\{P,\ \neg P \vee Q,\ \neg P \vee \neg Q\})\\
\Rightarrow^{\mathrm{SubRes}}_{\mathrm{RES}} \quad &(\{P,\ Q,\ \neg P \vee \neg Q\})\\
\Rightarrow^{\mathrm{SubRes}}_{\mathrm{RES}} \quad &(\{P,\ Q,\ \neg Q\})\\
\Rightarrow^{\mathrm{SubRes}}_{\mathrm{RES}} \quad &(\{P,\ Q,\ \bot\})
\end{aligned}$$

where I abbreviated the rule Subsumption Resolution by SubRes.

While the above example can be refuted by the rule Subsumption Resolution, the Resolution rule itself may derive redundant clauses, e.g., a tautology.

$$\begin{aligned}
&(\{P \vee Q,\ P \vee \neg Q,\ \neg P \vee Q,\ \neg P \vee \neg Q\})\\
\Rightarrow^{\mathrm{Resolution}}_{\mathrm{RES}} \quad &(\{P \vee Q,\ P \vee \neg Q,\ \neg P \vee Q,\ \neg P \vee \neg Q,\ Q \vee \neg Q\})
\end{aligned}$$

For three variables, the respective clause set is

$$\begin{aligned}
(\{&P \vee Q \vee R,\ P \vee \neg Q \vee R,\ \neg P \vee Q \vee R,\ \neg P \vee \neg Q \vee R,\\
&P \vee Q \vee \neg R,\ P \vee \neg Q \vee \neg R,\ \neg P \vee Q \vee \neg R,\ \neg P \vee \neg Q \vee \neg R\})
\end{aligned}$$

The above deterministic, linear resolution refutation, Example 2.6.4, cannot be simulated by the tableau calculus without generating an exponential overhead, see also the comment on page 43. At first, it looks strange to have the same rule, namely Factorization and Condensation, both as a reduction rules and as an inference rule. On the propositional level there is obviously no difference and it is possible to get rid of one of the two. In Section 3.10 the resolution calculus is lifted to first-order logic. In first-order

logic Factorization and Condensation are actually different, i.e., a Factorization inference is no longer a Condensation simplification, in general. They are separated here to eventually obtain the same set of rules propositional and first-order logic. This is needed for a proper lifting proof of first-order completeness that us actually reduced to the ground fragment of first-order logic that can be considered as a variant of propositional logic.

Proposition 2.6.5. The reduction rules Subsumption, Tautology Deletion, Condensation and Subsumption Resolution are sound.

Proof. This is obvious for Tautology Deletion and Condensation. For Subsumption we have to show that $C_1 \models C_2$, because this guarantees that if $N \cup \{C_1\}$ has a model, $N \uplus \{C_1, C_2\}$ has a model too. So assume $\mathcal{A}(C_1) = 1$ for an arbitrary $\mathcal{A}$. Then there is some literal $L \in C_1$ with $\mathcal{A}(L) = 1$. Since $C_1 \subseteq C_2$, also $L \in C_2$ and therefore $\mathcal{A}(C_2) = 1$. Subsumption Resolution is the combination of a Resolution application followed by a Subsumption application. □

Theorem 2.6.6 (Resolution Termination)**.** If reduction rules are preferred over inference rules and no inference rule is applied twice to the same clause(s), then $\Rightarrow^{+}_{\mathrm{RES}}$ is well-founded.

Proof. If reduction rules are preferred over inference rules, then the overall length if a clause cannot exceed n, where n is the number of variables. Multiple occurrences of the same literal are removed by rule Condensation, multiple occurrences of the same variable with different sign result in an application of rule Tautology Deletion. The number of such clauses can be overestimated by 3^n because every variable occurs at most once positively, negatively or not at all in clause. Hence, there are at most $2n3^n$ different resolution applications. □

Of course, what needs to be shown is that the strategy employed in Theorem 2.6.6 is still complete. This is not completely trivial. This result becomes a particular instance of superposition completeness. Exercise **??** contains the completeness part when the reduction rules are preferred over inference rules.

2.7 Propositional Superposition

Superposition was originally developed for first-order logic with equality [9]. Here I introduce its projection to propositional logic. Compared to the resolution calculus superposition adds (i) ordering and selection restrictions on inferences, (ii) an abstract redundancy notion, (iii) the notion of a partial model, based on the ordering for inference guidance, and (iv) a *saturation* concept.

Definition 2.7.1 (Clause Ordering)**.** Let $\prec$ be a total strict ordering on Σ. Then $\prec$ can be lifted to a total ordering on literals by $\prec \subseteq \prec_L$ and $P \prec_L \neg P$ and

2.9 Conflict Driven Clause Learning (CDCL)

The CDCL calculus tests satisfiability of a finite set N of propositional clauses. Similar to DPLL, Section 2.8, the CDCL calculus explicitely builds a candidate model for a clause set. If such a sequence of literals $L_1, \ldots, L_n$ satisfies the clause set N, it is done. If not, there is a false clause $C \in N$ with respect to $L_1, \ldots, L_n$. Now instead of just backtracking through the literals $L_1, \ldots, L_n$ as done in DPLL, CDCL generates an additional clause, called *learned clause*, that actually guarantees that the subsequence of $L_1, \ldots, L_n$ that caused C to be false will not be generated anymore. This causes CDCL to be exponentially more powerful in proof length than its predecessor DPLL, Section 2.8, or Tableau, Section 2.4, see Theorem 2.14.2. The learned clause is always a resolvent from clauses in N, so CDCL can be viewed as a combination of DPLL (Tableau) and Resolution. More precisely, it can be understood as a resolution variant where a partial model assumption triggers which resolvents are actually generated. In this regard it is similar to propositional Superposition, Section 2.7, where a model assumption generated out of an a priori total ordering on the propositional variables triggers the relevant resolution steps, see the proof of propositional superposition completeness, Theorem 2.7.11. I investigate the connection between model assumptions, proof length, completeness and orderings in Section 2.11.

For any clause set N, I assume that $\bot \notin N$ and that the clauses in N do not contain duplicate literal occurrences. Furthermore, duplicate literal occurrences are always silently removed during rule applications of the calculus. A CDCL problem state is a five-tuple $(M; N; U; k; D)$ where M a sequence of annotated literals, called a *trail*, N and U are sets of clauses, $k \in \mathbb{N}$, and D is a non-empty clause or $\top$ or $\bot$, called the *mode* of the state. The set N is initialized by the problem clauses where the set U contains all newly learned clauses that are consequences of clauses from N derived by resolution. In particular, the following states can be distinguished:

$(\epsilon; N; \emptyset; 0; \top)$	is the start state for some clause set N
$(M; N; U; k; \top)$	is a final state, if $M \models N$ and all literals from N are defined in M
$(M; N; U; k; \bot)$	is a final state, where N has no model
$(M; N; U; k; \top)$	is an intermediate model search state if $M \not\models N$
$(M; N; U; k; D)$	is a backtracking state if $D \notin \{\top, \bot\}$

Literals in $L \in M$ are either annotated with a number, a level, i.e., they have the form L^k meaning that L is the $k-th$ guessed decision literal, or they are annotated with a clause that forced the literal to become true. A literal L is of *level* k with respect to a problem state $(M; N; U; j; C)$ if L or $\text{comp}(L)$ occurs in M and L itself or the first decision literal left from L ($\text{comp}(L)$) in M is annotated with k. If there is no such decision literal then $k = 0$. A clause D is of *level* k with respect to a problem state $(M; N; U; j; C)$ if k is the maximal level of a literal in D. Recall C is a non-empty clause or $\top$ or $\bot$. The rules are