Applying this rule with respect to commutativity of $\vee$ means, for example, that both the formulas $(\phi \vee (\phi \wedge \psi))$ and $((\phi \wedge \psi) \vee \phi)$ can be transformed by the rule to ϕ where in both cases $p = \epsilon$. Rules are always applied modulo associativity and commutativity of $\wedge$, $\vee$.

The procedure is depicted in Algorithm 4. Although computing ac for Step 2 is not practical in general, because the function is exponentially growing, the test $\mathrm{ac}(\psi[\phi]_p) > \mathrm{ac}(\psi[P]_p \wedge \mathrm{def}(\psi, p, P))$ can be computed in constant time after a linear time processing phase, see the implementation comment above.

Algorithm 4: ocnf(ϕ)

Input : A formula ϕ.
Output: A formula ψ in CNF satisfiability preserving to ϕ.
1 **whilerule** *(***ElimRedI**(ϕ)*,***ElimRedV**(ϕ)*,***ElimRedVII**(ϕ)*)* **do** ;
2 **SimpleRenaming**(ϕ) on beneficial 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;

Applying Algorithm 4 to the formula $\neg((P \vee Q) \leftrightarrow (P \rightarrow (Q \wedge \top)))$ of Example 2.5.3 results in the transformation depicted in Figure 2.10, page 119. Looking at the result it is already very close to $\neg Q$, as it contains the clause $(\neg Q \vee \neg Q)$. Removing duplicate literals in clauses and removing clauses containing complementary literals from the result yields

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

which is even closer to just $\neg Q$. The first two clauses can actually be removed because they are subsumed by $\neg Q$, i.e., considered as multisets, $\neg Q$ is a subset of these clauses. Subsumption will be introduced in Section 2.6. Logically, they can be removed because $\neg Q$ has to be true for any satisfying assignment of the formula and then the first two clauses are satisfied anyway.

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

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

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

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.

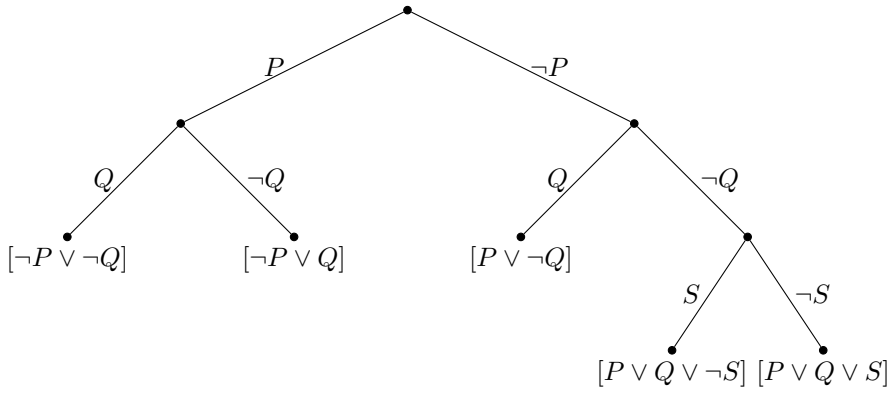
($\Rightarrow$) The traditional method of proving resolution completeness are *semantic trees*. A *semantic tree* is a binary tree where the edges are labeled with literals such that: (i) edges of children of the same parent are labeled with L and $\mathrm{comp}(L)$, (ii) any node has either no or two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



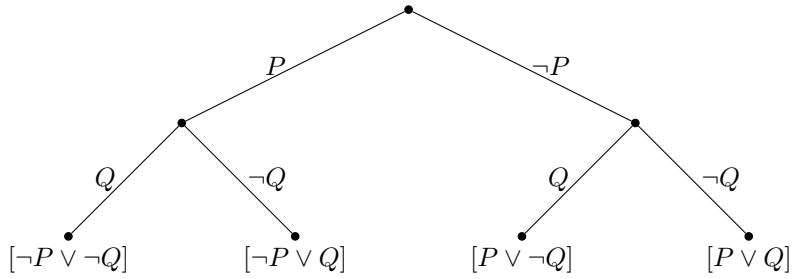
The first inference cuts the rightmost branch

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

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.

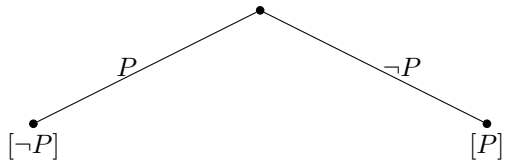
2 $N_1 \Rightarrow_{\mathrm{RES}} N_1 \cup \{P \vee Q \vee Q\}$
3 $N_2 \Rightarrow_{\mathrm{RES}} N_2 \cup \{P \vee Q\}$

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 2.13.

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 $\quad (N \uplus \{C_1, C_2\}) \Rightarrow_{\text{RES}} (N \cup \{C_1\})$

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

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:

$$\begin{array}{ll}
(\{P \vee Q,\ P \vee \neg Q,\ \neg P \vee Q,\ \neg P \vee \neg Q\}) \\
\Rightarrow_{\mathrm{RES}}^{\mathrm{SubRes}} & (\{P \vee Q,\ P,\ \neg P \vee Q,\ \neg P \vee \neg Q\}) \\
\Rightarrow_{\mathrm{RES}}^{\mathrm{Subumption}} & (\{P,\ \neg P \vee Q,\ \neg P \vee \neg Q\}) \\
\Rightarrow_{\mathrm{RES}}^{\mathrm{SubRes}} & (\{P,\ Q,\ \neg P \vee \neg Q\}) \\
\Rightarrow_{\mathrm{RES}}^{\mathrm{SubRes}} & (\{P,\ Q,\ \neg Q\}) \\
\Rightarrow_{\mathrm{RES}}^{\mathrm{SubRes}} & (\{P,\ Q,\ \bot\})
\end{array}$$

where I abbreviated the rule Subumption 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.

$$\begin{array}{ll}
(\{P \vee Q,\ P \vee \neg Q,\ \neg P \vee Q,\ \neg P \vee \neg Q\}) \\
\Rightarrow_{\mathrm{RES}}^{\mathrm{Resolution}} & (\{P \vee Q,\ P \vee \neg Q,\ \neg P \vee Q,\ \neg P \vee \neg Q,\ Q \vee \neg Q\})
\end{array}$$

For three variables, the respective clause set is

$$\begin{array}{l}
(\{P \vee Q \vee R,\ P \vee \neg Q \vee R,\ \neg P \vee Q \vee R,\ \neg P \vee \neg Q \vee R, \\
\ P \vee Q \vee \neg R,\ P \vee \neg Q \vee \neg R,\ \neg P \vee Q \vee \neg R,\ \neg P \vee \neg Q \vee \neg R\})
\end{array}$$