Resolution Theorem Proving

While I might argue that with modern programming languages, writing a program that will perform proof search in the sequent calculus is not very hard, in the early days of automated theorem proving the rich syntax of the full logic, and the variety of rules that need to be attempted at each step in the search were a daunting representational problem. Therefore, researchers looked for ways to simplify the domain, and, hence, the process.

The system that became prevalent, **Resolution**, is still the most widely implemented theorem proving technique.

Using Resolution as a Validity Checker

Resolution is, in fact, a *satisfiability* checker. Given a set of formulas it tells you whether they are mutually satisfiable.

In order to use it to check validity (and logical consequence) we will appeal to our earlier theorem that says that a formula is valid if and only if its negation is unsatisfiable.

Using Resolution as a Consequence Checker

We want to determine whether

$$\Gamma \models A$$

for some set of formulas Γ and and formula A. This is equivalent to showing that:

$$\models \Gamma_\wedge \Rightarrow A$$

where $\Gamma_\wedge$ is the conjunction of the formulas in Γ. But, since if Φ is valid, $\neg\Phi$ is unsatisfiable, we will apply resolution to:

$$\neg(\Gamma_\wedge \Rightarrow A)$$

If that formula is unsatisfiable, the original consequence holds. Otherwise, not.

First, though it is informative to use the equivalences to simplify the formula.

$$\begin{aligned} \neg(\Gamma_\wedge \Rightarrow A) &\leftrightarrow \neg((\neg\Gamma_\wedge) \vee A) \\ &\leftrightarrow (\neg(\neg\Gamma_\wedge)) \wedge (\neg A) \\ &\leftrightarrow \Gamma_\wedge \wedge \neg A \end{aligned}$$

So, we are testing the mutual satisfiability of the original assumptions and the negation of the conclusion.

Conjunctive Normal Form

Resolution's simplicity is based on the recognition that, because of the results we have about functionally complete sets of operators, the syntax of formulas can be significantly simplified by reduction to certain *normal forms*.

Resolution is dependent on the use of one particular normal form: **Conjunctive Normal Form**, or **CNF**. A CNF formula is one which is a conjunction of disjunctions of literals (positive or negative atoms). That is, a formula of the form:

$$(l_{11} \vee \cdots \vee l_{1n_1}) \wedge \cdots \wedge (l_{m1} \vee \cdots \vee l_{mn_m})$$

where each l_{ij} is either of the form p or of the form $\neg p$ for some propositional letter p.

Conjunctive Normal Form

Theorem (2.10)**:** For every formula $\Phi \in PROP$, there is a formula $\Phi_{CNF} \in PROP$ such that $\Phi \leftrightarrow \Phi_{CNF}$ and Φ_{CNF} is in conjunctive normal form.

Proof. The proof is in the form of a recursive algorithm that produces the equivalent CNF formula. (To be precise, you should prove that this algorithm terminates for all well-formed inputs and that the output is in CNF.):

1. Replace uses of $\Rightarrow$ and $\equiv$ with uses of $\vee$, $\wedge$, and $\neg$ using the equivalences:

   $$A \Rightarrow B \leftrightarrow \neg A \vee B \qquad A \equiv B \leftrightarrow (\neg A \vee B) \wedge (\neg B \vee A)$$

2. Push negations inwards using the equivalences:

   $$\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B \qquad \neg(A \wedge B) \leftrightarrow \neg A \vee \neg B$$
   $$\neg\neg A \leftrightarrow A$$

3. Use the distributivity laws for $\vee$ over $\wedge$:

   $$A \vee (B \wedge C) \leftrightarrow (A \vee B) \wedge (A \vee C)$$
   $$(A \wedge B) \vee C \leftrightarrow (A \vee C) \wedge (B \vee C)$$

Conjunctive Normal Form

So, for example:

$$(\neg p \Rightarrow \neg q) \Rightarrow (p \Rightarrow q)$$

$$\updownarrow$$

Conjunctive Normal Form

And:

$$(hj \Rightarrow (bij \Rightarrow ((hj \Rightarrow (bij \Rightarrow bd)) \Rightarrow bd)))$$

$$\updownarrow$$

Conjunctive Normal Form

Definition: A disjunction of literals is called a *clause*. A formula in CNF is also refered to as being in *Clause Normal Form*.

Note that we can discard the operators and talk about manipulating sets of sets of literals. The CNF formula above can be represented by the set:

$$\{\{l_{11}, \ldots, l_{1n_1}\}, \ldots, \{l_{m1}, \ldots, l_{mn_m}\}\}$$

The rest of the discussion will be stated in terms of this representation.

The Resolution Rule

Definition: The *Resolution Rule* is stated as follows:

> Let C_1 and C_2 be clauses such that there is a literal $l \in C_1$ with $l^c \in C_2$. C_1 and C_2 are said to be *clashing* clauses and to *clash* on the literals l and l^c. C, the *resolvent* of C_1 and C_2, is the clause:
>
> $$Res(C_1, C_2) = (C_1 - \{l\}) \cup (C_2 - \{l^c\})$$
>
> The clauses C_1 and C_2 are refered to as the *parent clauses*.

For example:

$$a \vee b \vee \neg c \qquad b \vee c \vee \neg e$$

$$\neg a \vee b \vee \neg c \qquad \neg a \vee \neg b \vee c \vee \neg e$$

The Resolution Rule

Lemma:(2.10.12) Given two clauses, C_1 and C_2, their resolvent, C, is satisfiable if and only if the parent clauses C_1 and C_2 are mutually satisfiable.

Proof.

Resolution Procedure

Definition: The *Resolution Procedure* is given as:

> Let S be a set of clauses and define $S_0 = S$. Assume that S_i has been constructed. *Choose* clashing clauses $C_1, C_2 \in S_i$ and let $C = Res(C_1, C_2)$. If $C = \square$ (the empty clause) then terminate the procedure – S is unsatisfiable. Otherwise, construct $S_{i+1} = S_i \cup \{C\}$. If $S_{i+1} = S_i$ for all pairs of clashing clauses, terminate the procedure – S is satisfiable.

Definition: A derivation of $\square$ from a set of clauses is called a *refutation* of the clauses, since it says they are not mutually satisfiable.

For Example:

$$p \qquad \neg p \vee q \qquad \neg r \qquad \neg p \vee \neg q \vee r$$