1 Resolution

1.1 Overview

Resolution is a proof system that operates on clause (ORs of literals).

1. It has no axioms.
2. Single inference rule, “cut”.

Given clauses \( x_i \lor C \) (rest of clause) and \( \neg x_i \lor D \), derive \( C \lor D \) (\( x_i \lor \neg x_i \) are complementary pairs)

1.2 Soundness?

Suppose some \( x \in X \) satisfies both \( x_i \lor C \) and \( \neg x_i \lor D \), each of \( x_i \lor C \) and \( \neg x_i \lor D \) must have a satisfied literal and it can’t be that \( x_i \) and \( \neg x_i \) are respectively satisfying them.

1.3 How to use?

Resolution is usually used to obtain proof by contradiction. Suppose KB is a CNF, \( C_1 \land \ldots \land C_k \). (we assume KB is consistent, i.e., it has at least one satisfying assignment.) Suppose we wish to prove \( KB \models \phi \) for some \( \phi \). We take \( \neg \phi \) (which by de Morgans law, converts to a CNF) and we show that \( \neg \phi \land C_1 \land \ldots \land C_k \) cannot be satisfied. This shows that \( x : KB(x) \subset x : \phi(x) = 1 \), i.e., \( KB \models \phi \).

We achieve this with resolution by giving a derivation of the “empty clause” \( \bot \) we take to be unsatisfiable. Why is this true?

On the final step, we apply cut to \( x_i \) and \( \neg x_i \), which cannot be simultaneously satisfied. Since cut preserves satisfying assignments, the original formula must be unsatisfiable.

1.4 Example

Recall \( A \Rightarrow B \) is a shorthand for \( \neg A \lor B \). So our chaining rules, e.g., \([\text{dog} \land \text{young}] \Rightarrow \text{puppy} \) can be taken to be clauses, \( \neg \text{dog} \lor \neg \text{young} \lor \text{puppy} \), etc. Suppose I wished to prove, given

\[ \neg \text{dog} \lor \neg \text{young} \lor \text{puppy}, \text{dog}, \neg \text{puppy} \lor \text{mischief} \Rightarrow \neg \text{young} \lor \text{mischief}. \]

We would want to refute

\( (\text{young} \land \neg \text{mischief}) \land (\neg \text{dog} \lor \neg \text{young} \lor \text{puppy}) \land \text{dog} \land (\neg \text{puppy} \lor \text{mischief}) \)

by using RES.
Since we only used each intermediate derivation once, its a treelike proof.

2 SAT-solver

Most algorithms for SAT (SAT-solvers) implicitly search for resolution refutations.

**Basic SAT-solver** A partial assignment, $\rho \in \{0, 1, *\}^n$, $\rho_i$ fixes $x_i$ to a Boolean value when $\rho_i \in \{0, 1\}$, otherwise $x_i$ is unspecified/free. The operation of plugging in a partial assignment is called applying a restriction. (We say that $x \in X$ is consistent with $\rho$ if whenever $\rho_i \neq *$, $\rho_i = x_i$)

**Definition 1** For a CNF: $\phi = C_1 \land \ldots \land C_k$, restriction of $\phi$ under $\rho$, $\phi/\rho$ is defined as follows.

For a clause $C_i$, if some element of $C_i$ (either $x_k$ or $\neg x_j$) is satisfied by $\rho$, then $C_j|\rho = 1$. (and will be omitted from $\phi/\rho$).

Otherwise, $C_j|\rho$ is the OR over the literals of $C_i$ on attributes unspecified by $\rho$.

if some $C_j|\rho = \bot$, then $\phi/\rho = 0$.

elseif every $C_j|\rho = 1$, $\phi/\rho = 1$ and otherwise $\phi/\rho = \bigwedge_{i:C_i|\rho \neq 1} C_i|\rho$.

We have a natural notion of composition of the restrictions. Given $\sigma$ that is a partial assignment on $\{x_i : \rho_i = *\}$, then $\tau = \phi \rho$ is the partial assignment on $X$, s.t. $\tau_i = \begin{cases} \rho_i, \rho_i \neq * \\ \sigma_i, \text{otherwise.} \end{cases}$

Notice $(\phi|\rho)|\sigma = \phi|\tau$.

2.1 Native SAT

*NaiveSAT*(\phi):

**IF** $\phi = 0$

*Return* Fail
IF $\phi = 1$
Return Empty partial assignment.
ELSE
choose some arbitrary $x_j \in \phi$
IF $NaiveSAT(\phi|x_j = 1) = \rho(\neq \text{Fail})$
Return $[x_j = 1]\rho$
ELSEIF $NaiveSAT(\phi|x_j = 0) = \rho(\neq \text{Fail})$
Return $[x_j = 0]\rho$
ELSE
Return Fail

2.2 Three Modifications

1. Pure literals: attributes that only appear positively at negatively in $\phi$ are first fixed to the appropriate (satisfying) value.
2. Unit propagation: If there are any clauses consisting of a single literal (unit clauses) choose a next attribute assignment to satisfy one of those.
3. Clause learning and resets: Whenever we find a partial assignment $\rho$ that falsifies $\phi$, derive a new clause from $\rho$, e.g., if $\rho$ satisfies exactly the conjunction $l_i \land \ldots \land l_k$, include $C_{\rho} = \neg(l_i \land \ldots \land l_k)$ in $\phi$. Occasionally, drop the partial assignment $\rho$ but keep the new clauses and restart.

When any of these algorithms fails to find a satisfying partial assignment, a trace of its execution gives a RES refutation. (Treelike RES for Naive SAT and DPLL)

2.3 Native RES

$NaiveRES(\phi)$:

IF $\phi = 0$
Return Fail
IF $\phi = 1$
Return Empty partial assignment.
ELSE
choose some arbitrary $x_j \in \phi$
IF $NaiveRES(\phi|x_j = 1) = \rho$ (not ref)
Return $[x_j = 1]\rho$
ELSEIF $NaiveSAT(\phi|x_j = 0) = \rho$ (not ref)
Return $[x_j = 0]\rho$
ELSEIF $NaiveRES(\phi|x_j = 1) = \pi(1)$ is a refutation of $\phi$ or if $NaiveRES(\phi|x_j = 0) = \pi(0)$ is a refutation of $\phi$
Return the appropriate $\pi(b)$
ELSE
Return $\pi(0), \pi(1), \perp$ (for $\phi$)

Claim 2 $Naive RES$ returns a refutation of ANY unsatisfiable CNF.

Proof By induction on number of variables.
**Base Case** 0 free variables, then either $\phi = 1$ (SAT $\perp$) or obtained $\phi = \perp \land \ldots$ in which case asserting $\perp$ is sufficient.

**Induction Step** If $\phi$ is unsatisfiable then both $\phi|x_j=1$ and $\phi|x_j=0$ are unsatisfiable, and have $n - 1$ variables. So, by induction hypothesis, Naive RES returns a refutation of each.

When we apply the corresponding derivation to $\phi$, either one of them actually still derives $\perp$ (in which case, we are done) OR they derive unit clauses on $x_j$ that were falsified by $x_j = b$.

Since one has $x_j = 1$ and the other has $x_j = 0$, these must be $\pi'(0)$ derives $x_j$ and $\pi'(1)$ derives $\neg x_j$, so applying cut to the final step, this final proof is treelike if $\pi'(0)$ and $\pi'(1)$ were.

**Remark** Since Naive RES returns a refutation whenever $\phi$ is unsatisfiable, RES is complete for refuting CNFs. The size of the RES proof we generate is the lower bound on the running time of the SAT-solver. (only large treelike RES refutations ⇒ DPLL slow, only large RES refutations ⇒ CDCL slow)

**CDCL**: Conflict Directed Clause Learning

### 3 Complexity Measures for Resolution

**Most natural** Proof length (steps)

- **Ideally** Wed like a SAT-solver / RES reasoning algorithm that runs in poly time in $n + \text{length}$.

- **Open Question** Does any CDCL SAT-solver achieve this time complexity?

**Second Measure** Space (Clause) complexity

- Roughly of clauses I need to remember simultaneously in order to follow a RES proof. (Much like the size of a working memory to follow a proof.)

Consider the following modified notion of a proof system.

Associated with each line of the proof is a set of representations called the blackboard initially empty. The next step of the proof must satisfy the relation $R$ with some $\pi_1, \ldots, \pi_i$, that appear in the blackboard on that step. The resulting next step representation is itself added to the blackboard. After any step, any subset of the representation may be erased, i.e. omitted, from the next blackboard. The clause space of a proof is now the maximum of representations appearing on any blackboard.

**Remark** An equivalent characterization of treelike proof is that whenever a representation is used on a step, it is then erased.