Notes on Satisfiability-Based Problem Solving
Resolution and Backtracking

David Mitchell
mitchell@cs.sfu.ca
September 27, 2013

This is a preliminary draft of these notes. Please do not distribute without permission. Corrections and comments are welcome.

In this section, we describe the resolution proof system and its relationship to the backtracking algorithm for SAT.

Terms and Conventions In this section \( \Gamma \) always denotes a set of clauses, and \( \alpha \) a (possibly partial) truth assignment to its atoms. \( \Gamma \) has \( n \) distinct atoms, \( m \) clauses, and \( l \) literal occurrences. “CNF Formula” and “set of clauses” are regarded as synonyms.

1 Propositional Resolution

To demonstrate satisfiability of a formula requires only exhibiting a satisfying assignment. It may not be easy to find such an assignment, but once found it is easy to check. Demonstrating unsatisfiability (or logical implication) requires showing that no assignment satisfies a formula. For all but very small formulas, generating the truth table is too inefficient. Here, we introduce the resolution proof system, a relatively simple tool for demonstrating unsatisfiability of CNF formulas.

Definition 1. If \( B = (L \lor B') \) and \( C = (L \lor C') \) are clauses (where \( B' \) and \( C' \) are possibly empty disjunctions of literals), then the resolvent of \( B \) and \( C \) on \( L \) is the clause \( (B' \lor C') \).

Described in set notation, the resolvent of clauses \( B \) and \( C \) on literal \( L \) is \( (B \cup C) - \{L, \bar{L}\} \).

The rule is illustrated graphically by:

\[
\frac{(L \lor A) (\bar{L} \lor B)}{(A \lor B)}.
\]

In (1), the bottom clause is called the resolvent of \( (L \lor A) \) and \( (\bar{L} \lor B) \) on \( L \). The top clauses are called the antecedents, and we say that the two antecedents clash on \( L \).
Proposition 1. The resolution rule is sound. That is, every truth assignment that satisfies both antecedents also satisfies their resolvent.

Proof. Let \( \alpha \) be a truth assignment for the atoms of the antecedents. If \( \alpha(L) = \text{false} \), then \( \alpha \) must map some literal in \( A \) to true, since it satisfies \( (L \lor A) \), so \( \alpha \) satisfies \( (A \lor B) \). If \( \alpha(L) = \text{true} \), then \( \alpha \) satisfies some literal in \( B \), and thus satisfies the resolvent clause.

Definition 2. A resolution derivation \( \Pi \) of clause \( C \) from clause set \( \Gamma \) is a sequence \( \Pi = C_0, C_1, \ldots, C_n \) of clauses with \( C_n = C \) and each clause \( C_i \) is either contained in \( \Gamma \) or is obtained by resolution from two clauses \( C_j, C_k \) with \( j, k < i \).

Example 1. \( (A \lor B \lor C), (\overline{A} \lor B), (B \lor C), (\neg C \lor D), (\neg D \lor B), (\neg C \lor B), (B) \) is a resolution derivation of \( (B) \) from \( (A \lor B \lor C), (\overline{A} \lor B), (\neg C \lor D), (\neg D \lor B) \).

Defining a derivation as a sequence of clauses means derivations are strings, which is desirable formally. However, it is not very convenient for displaying derivations, because the structure is not transparent. Therefore, we usually present derivations graphically, as in (1) and Example 2.

Example 2. Graphical presentation of the derivation of Example 1.

\[
\begin{array}{cccc}
(A \lor B \lor C) & (\overline{A} \lor B) & (\overline{C} \lor D) & (\overline{D} \lor B) \\
(B \lor C) & (B \lor C) & (B \lor C) & (B \lor C) \\
\end{array}
\]

A simple inductive application of the transitivity of logical implication shows that every clause which can be obtained from \( \Gamma \) by a resolution derivation is logically implied by \( \Gamma \). If follows that, if we can derive the empty clause from \( \Gamma \) by resolution, then \( \Gamma \) is unsatisfiable. We will show later that the reverse holds as well.

Exercise 1. Prove that, if there is a resolution derivation \( \Pi \) of clause \( C \) from \( \Gamma \), then \( \Gamma \models C \).

Definition 3. A resolution refutation of \( \Gamma \) is a resolution derivation of the empty clause from \( \Gamma \).

Example 3. Let \( \Gamma \) be the clause set

\[
(P \lor R), (\overline{Q} \lor \overline{P}), (Q \lor R), (R \lor Q \lor S), (S \lor Q \lor \overline{U}), (P \lor \overline{R} \lor \overline{S}), (\overline{P} \lor Q \lor \overline{R}), (P \lor Q \lor U)
\]
Then the following is a resolution refutation of $\Gamma$:

\[
\begin{array}{c}
(P \lor Q \lor R) (Q \lor R) \\
(P \lor Q) (Q \lor P) \\
(P \lor R \lor S) (S \lor R) \\
(P \lor R) (P \lor R)
\end{array}
\]

Exercise 2. Construct a resolution refutation of each of the following sets of clauses. Here, we use a more concise hybrid set/clause notation, in which $(P \lor Q)$ is written $(P, Q)$.

1. $(P, \overline{Q}, R), (R, \overline{P}), (P, Q), (P, R, \overline{Q}), (R, Q), (P, \overline{R})$
2. $(V, \overline{P}, \overline{Q}), (\overline{V}, \overline{T}, P), (P, Q), (V, R), (U, S), (\overline{V}, T), (\overline{V}, U, \overline{S}), (P, \overline{R}), (U, T, P)$

Theorem 1. A set $\Gamma$ of clauses has a resolution refutation if and only if it is unsatisfiable.

We say that resolution is refutationally sound and complete.

Remark 1. A proof system is said to be complete if $A$ can be derived from $\Phi$ whenever $\Phi \models A$. Resolution is not complete in this sense, even if we restrict $\Phi$ to be a set of clauses. For example, $(A \lor B)$ logically implies $(A \lor B \lor C)$ but there is no way to derive the latter from the former by the resolution rule.

2 Refutation Completeness of Resolution

Here we show one way to demonstrate that every unsatisfiable clause set has a resolution refutation. We assume that $\Gamma$ is a finite, unsatisfiable set of clauses. (In practice, the finite case is usually our concern. Moreover, by the compactness theorem for propositional logic, any unsatisfiable clause set has a finite unsatisfiable sub-set.)

Given clause set $\Gamma$ construct a complete semantic tree $T = T_\Gamma$ for $\Gamma$ as follows. Begin with a single node, the root, and label it with an atom $L$ from $\Gamma$. Add two children. Label the edge to one child $L$ and the edge to the other child $\overline{L}$. As the tree is constructed, associate to each vertex $v$ the partial truth assignment $\tau_v$ determined by the literals labelling edges on the path from the root to $v$. Now, apply the following rule as long as there is a leaf $l$ such that $\tau_l$ is not a total assignment for $\Gamma$:

Let $l$ be a leaf with $\tau_l$ not total, and let $L$ be an atom of $\Gamma$ for which $\tau_l(L)$ is not defined. Label $l$ with $L$, and add two children to $l$, labelling the edge to one child with $L$ and the other with $\overline{L}$. sucht that $\tau_l$ is not a total assignment for $\Gamma$. 

Let $l$ be a leaf with $\tau_l$ not total, and let $L$ be an atom of $\Gamma$ for which $\tau_l(L)$ is not defined. Label $l$ with $L$, and add two children to $l$, labelling the edge to one child with $L$ and the other with $\overline{L}$. 

3
$T$ is a perfect binary tree, and the leaves of $T$ enumerate the truth assignments for the atoms of $\Gamma$. Label the nodes of $T$ with clauses as follows. (Each node will be labelled by an atom and a clause.) Label each leaf $l$ with a clause from $\Gamma$ that is made false by $\tau_l$. (There is one, because $\Gamma$ is unsatisfiable.) Now, as long as there is a node without a clause label:

Let $v$ be a node of $T$ with no clause label, for which both children have clause labels. Let $C_1, C_2$ be the clause labels of the children of $v$, and $L$ be the atom labelling $v$. If $C_1$ and $C_2$ clash on $L$ label $v$ with the resolvent of $C_1, C_2$. Otherwise, choose one of $C_1, C_2$ that contains neither $L$ nor $\overline{L}$, and label $v$ with it.

**Proposition 2.** Let $T$ be a complete semantic tree for an unsatisfiable clause set $\Gamma$. Then every vertex $v$ of $T$ is labelled by a clause $C_v$ such that $\tau_v(C_v) = \text{false}$.

**Proof.** (Sketch:) We use induction on the structure of $T$. If $v$ is a leaf, the property holds by choice of $C_v$. Otherwise, let $v_1, v_2$ be the children of $v$, and $L$ the atom labelling $v$. By inductive hypothesis, $\tau_{v_1}(C_{v_1}) = \tau_{v_2}(C_{v_2}) = \text{false}$. By construction, $\tau_{v_1}$ and $\tau_{v_2}$ agree on all atoms except $L$. Therefore, the only possible clashing literal for $C_{v_1}$ and $C_{v_2}$ is $L$. If $C_{v_1}$ and $C_{v_2}$ clash, their resolvent does not contain $L$ or $\overline{L}$. Otherwise, at least one of them contains neither $L$ nor $\overline{L}$. In either case, $v$ is labelled by a clause satisfying the claim. □

**Corollary 1.** Let $T$ be a complete semantic tree for unsatisfiable clause set $\Gamma$. The root of $T$ is labelled with the empty clause.

**Proof.** (Completeness of Resolution) Let $T$ be constructed from $\Gamma$ as described, and let $\Pi = C_1, C_2, \ldots C_s$ be a listing of the clauses labelling nodes of $T$ that respects the partial order induced by $T$. That is, if $C_j$ occurs in $T$ on the path from $C_i$ to the root of $T$, then $i < j$. By construction of $T$ and $\Pi$, $\Pi$ is a resolution derivation from $\Gamma$, and since the root of $T$ is labelled with the empty clause, it is a refutation of $\Gamma$. □

**Exercise 3.** Let $\Gamma$ be $(\overline{P}, \overline{Q}, \overline{R}), (R, \overline{P}), (P, Q), (P, \overline{Q}, R), (P, R, \overline{Q}), (R, Q), (P, \overline{R})$. Construct a complete semantic tree for $\Gamma$, and label it with clauses as described in the lead-up to Proposition 2.

### 3 Resolution and Unit Propagation

In unit propagation, if $\Gamma$ has a unit clause $(L)$, we replace each clause of the form $(\overline{L} \lor A)$ with $(A)$. Observing that $(A)$ is the resolvent of $(L)$ and $(\overline{L} \lor A)$, we see a close relationship between unit propagation and “unit resolution” - resolution when one antecedent clause is unit. It is not hard to show that any unsatisfiable 2-CNF or Horn formula has a
linear-sized resolution refutation. These refutations can be constructed explicitly by simple modifications of the algorithms we have already seen for 2-SAT and Horn-SAT.

## 4 Resolution and Backtracking

The simplest algorithm for SAT that is a bit smarter than enumerating the truth table for the input formula is backtracking, as given in Algorithm 1.

---

**Algorithm 1** Backtracking for SAT

```plaintext
1: // $\Gamma$ is satisfiable if BT-SAT($\Gamma, \emptyset$) returns “SAT”.
2: procedure BT-SAT($\Gamma, \tau$) // $\Gamma$ is a set of clauses, $\tau$ a partial truth assignment.
3: if $\tau(C) = \text{false}$ for some $C \in \Gamma$ then
4: return “UNSAT”.
5: else if $\tau$ is total then
6: return “SAT”
7: else
8: $L \leftarrow$ a literal from $\Gamma$ for which $\tau(L)$ is undefined.
9: if BT-SAT($\Gamma, \tau \cup \{L\}$) returns “SAT” then
10: return “SAT”
11: else
12: return BT-SAT($\Gamma, \tau \cup \{\overline{L}\}$)
13: end if
14: end if
15: end procedure
```

One may view the backtracking algorithm as performing a depth-first search of a semantic tree for $\Gamma$. However, the algorithm does not traverse the complete tree, but stops exploring a path as soon as the partial assignment on the path makes a clause false. Thus, the algorithm will normally explore fewer truth assignments than the truth table method. Moreover, if the backtracking algorithm can determine a formula is unsatisfiable by making $n$ assignments (or $n$ recursive calls), then there is a resolution refutation of the formula containing at most $n + 1$ clauses.