Notes on Satisfiability-Based Problem Solving

Resolution and Backtracking

David Mitchell
mitchell@cs.sfu.ca
February 5, 2020

This is a preliminary draft. Please use freely but do not distribute without permission. Corrections and suggestions are welcome.

In this section, we describe the propositional resolution proof system and it's relationship to the backtracking algorithm for SAT.

Terms and Conventions In this section Γ denotes a set of clauses over a set S of n atoms, with size (number of clauses) m, and total size (number of literal occurrences) l. τ is a (possibly partial) truth assignment for S. "CNF Formula" and "set of clauses" are regarded as synonyms. $\square$ denotes the empty clause.

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. This can be done by producing the truth table, but this is practical only for very small for formulas. Here, we introduce the propositional resolution proof system, a relatively simple tool for demonstrating unsatisfiability of CNF formulas.

Definition 1. Let $B = (L \vee B')$ and $C = (\overline{L} \vee C')$ be clauses, where B' and C' are (possibly empty) disjunctions of literals. We may resolve clauses B and C on L to derive the clause $(B' \vee C')$. B and C are the antecedents, L the clashing atom, and $(B' \vee C')$ the resolvent.

In set notation, the resolvent of clauses B and C on literal L is $(B \cup C) - \{L, \overline{L}\}$. The rule is illustrated graphically by:

$$\frac{(L \vee A) \quad (\overline{L} \vee B)}{(A \vee B)}. \tag{1}$$

Proposition 1. *The resolution rule is sound. (That is, every truth assignment that satisfies both antecedents also satisfies their resolvent.)*

Proof. Let τ be a truth assignment that satisfies both of the antecedents. If $\tau(L) = \text{false}$, then by assumption τ makes some literal in A true, so τ satisfies $(A \vee B)$. Symmetrically, if $\tau(L) = \text{true}$, then τ satisfies some literal in B, so also satisfies the resolvent. □

Definition 2. A resolution derivation Π of clause C from clause set Γ is a sequence $\Pi = C_0, C_1, \ldots C_n$ of clauses such that $C_n = C$ and each clause C_i is either an element of Γ or is obtained by resolution from two clauses C_j, C_k of Π with $j, k < i$.

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

In a derivation presented as a string, it may not be easy to see the structure of the deriviation. Therefore, we usually present derivations graphically, sometimes in the manner of (1) and Example 2.

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

$$\dfrac{\dfrac{(A \vee B \vee C) \qquad (\overline{A} \vee B)}{(B \vee C)} \qquad \dfrac{(\overline{C} \vee D) \qquad (\overline{D} \vee B)}{(B \vee \overline{C})}}{(B)}$$

Proposition 2. *If there is a resolution derivation of a clause C from Γ, then $\Gamma \models C$.*

Proposition 2 asserts that resolution is a sound derivation system. This can be proven by a simple inductive application of transitivity of logical implication.

Exercise 1. *Prove Proposition 2.*

Definition 3. A resolution derivation of the empty clause from Γ is called a *resolution refutation* of Γ.

By Proposition 2, if we can derive the empty clause from Γ then $\Gamma \models \square$. So, every assignment that satisfies Γ satisfies $\square$, but $\square$ is unsatisfiable, so Γ must be also. Thus, a resolution refutation is a proof that Γ is unsatisfiable. In Section 2 we will show that every unsatisfiable set of clauses has a resolution refutation.

Example 3. *Let Γ be the clause set*

$$(P \vee R), (\overline{Q} \vee \overline{P}), (Q \vee R), (\overline{R} \vee P \vee S), (S \vee Q \vee \overline{U}), (P \vee \overline{R} \vee \overline{S}), (\overline{P} \vee Q \vee \overline{R}), (P \vee Q \vee U)$$

Then the following is a resolution refutation of Γ*:*

$$\dfrac{\dfrac{\dfrac{(\overline{P} \vee Q \vee \overline{R}) \qquad (Q \vee R)}{(\overline{P} \vee Q)} \qquad (\overline{Q} \vee \overline{P})}{(\overline{P})} \qquad \dfrac{\dfrac{(P \vee \overline{R} \vee \overline{S}) \qquad (\overline{R} \vee P \vee S)}{(P \vee \overline{R})} \qquad (P \vee R)}{(P)}}{()}$$

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* $(P \vee Q)$ *is written* (P, Q)*.*

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

2 Refutation Completeness of Resolution

Here we show that every unsatisfiable clause set has a resolution refutation. The method we use is not the most elegant, but it makes an important connection between resolution and certain algorithms explicit. We assume Γ is a finite, unsatisfiable set of clauses. (By the compactness theorem for propositional logic, any infinite unsatisfiable clause set has a finite unsatisfiable sub-set.)

Given clause set Γ, first construct a *complete semantic tree* $T = T_\Gamma$ for Γ. Each node v of T will be labelled by a partial truth assignment τ_v, and an atom L for which τ_v is undefined. Construct T inductively as follows. Begin with a single node, the root, labelled with the empty partial truth assignment. Now, apply the following rule, as long as the tree has a leaf v such that τ_v is not a total assignment for Γ:

> Let v be a leaf with τ_v not total, and let L be an atom of Γ for which $\tau_v(L)$ is not defined. Label v with L, and add two children to l. Label one child with $\tau \cup L$ and the edge to that child with L. Label the other edge with $\overline{L}$ and the other child with $\tau \cup \overline{L}$.

The resulting tree is T. Notice that, for each node v of T, the assignment τ_v is the partial truth assignment determined by the literals labelling edges on the path from the root to v. When completed, T is a perfect binary tree of height n, and the leaves of of T enumerate the 2^n truth assignments for the atoms of Γ.

Now, label all nodes of T with clauses, by applying the following rules:

1. Label each leaf l with a clause from Γ that is made false by τ_l. (There is such a clause, because Γ is unsatisfiable.)
2. If there is a node of T with no clause label, let v be such a node 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 3. *Let T be a complete semantic tree for an unsatisfiable clause set Γ, labelled by clauses according to Rules 1 and 2. Then every vertex v of T is labelled by a clause C_v such that $\tau_v(C_v) = false$.*

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 $v1, v2$ be the children of v, and L the atom labelling v. By inductive hypothesis, $\tau_{v1}(C_{v1}) = \tau_{v2}(C_{v2}) =$ false. By construction, τ_{v1} and τ_{v2} agree on all atoms except L. Therefore, the only possible clashing literal for C_{v1} and C_{v2} is L. If C_{v1} and C_{v2} 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 Γ. The root of T is labelled with the empty clause.*

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

Theorem 1. *A set Γ of clauses has a resolution refutation if and only if it is unsatisfiable.*

Proof. One direction is Proposition 2. The other is implied by our labelled semantic tree construction as follows. Let T be constructed from Γ as described, and let $\Pi = C_1, C_2, \ldots C_s$ be an enumeration of the clauses labelling nodes of T such that that, if C_j is a descendant of C_i in T then $i < j$. By construction of T and Π, Π is a resolution derivation from Γ, and since the root of T is labelled with the empty clause, it is a refutation of Γ. □

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

3 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

```
1:  // Γ is satisfiable iff BT-SAT(Γ,∅) returns "SAT".
2:  procedure BT-SAT( Γ, τ ) // Γ is a set of clauses, τ a partial truth assignment.
3:      if τ(C) = false for some C ∈ Γ then
4:          return "UNSAT".
5:      else if τ is total then
6:          return "SAT"
7:      else
8:          L ← a literal from Γ for which τ(L) is undefined.
9:          if BT-SAT( Γ, τ ∪ {L}) returns "SAT" then
10:             return "SAT"
11:         else
12:             return BT-SAT( Γ, τ ∪ {L̄})
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 Γ. Performing a depth-first search of the complete semantic tree amounts to constructing the truth table. However, the backtracking algorithm, for almost all formulas, does not traverse the complete semantic tree, but stops exploring a branch as soon as the partial assignment on the branch makes a clause false. Thus, the backtracking tree normally has far fewer than 2^n leaves. (Nonetheless, there are many families of formulas for which this naive form of backtracking is inefficient in practice, but better algorithms are effective.)

Our construction leads to the interesting observation that, if the backtracking algorithm can determine a formula is unsatisfiable by making s assignments of values to atoms (or s recursive calls), then there is a resolution refutation of the formula containing at most $s + 1$ clauses.