CS1200: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Lecture 18: Resolution

Harvard SEAS - Fall 2025 *2025-11-04*

1 Announcements

- Salil's in-person OH this week moved to Thursday 1:30-2:15pm.

Recommended Reading:

- Hesterberg–Vadhan 18

2 The Basic Resolution Algorithm

SAT Solvers are algorithms to solve CNF-Satisfiability. Although they have worst-case exponential running time, on many "real-world" instances, they terminate surprisingly quickly with either (a) a satisfying assignment, or (b) a "proof" that the input formula is unsatisfiable.

The best known SAT solvers implicitly use the technique of *resolution*. The idea of resolution is to repeatedly derive new clauses from the original clauses (using a valid deduction rule) until we either derive an empty clause (which is false, and thus gives a proof that the original formula is unsatisfiable) or we cannot derive any more clauses (in which case we can efficiently construct a satisfying assignment).

Our deduction rule will make use of a clause simplification procedure `Simplify`, which takes a clause C and simplifies it as follows:

1.

2.

3.

Definition 2.1 (resolution rule)**.** For clauses C and D, define their *resolvent* to be

$$C \diamond D = \begin{cases} \text{---------------------------} & \text{if } \ell \text{ is a literal s.t. } \ell \in C \text{ and } \neg\ell \in D \\ 1 & \text{if there is no such literal } \ell \end{cases}$$

Here $C - \{\ell\}$ means remove literal ℓ from clause C, and 1 represents `true`. As we will see below, if C and D can be resolved with respect to more than one literal ℓ, then for all choices of ℓ we will have $\texttt{Simplify}((C - \{\ell\}) \vee (D - \{\neg\ell\})) = 1$, so $C \diamond D$ is well-defined.

In the special case where $C = \ell, D = \neg\ell$, we obtain the empty clause ( ), which is always false:

$$(\ell) \diamond (\neg\ell) = (\,) = \text{FALSE}.$$

From now on, it will be useful to view a CNF formula as just a set $\mathcal{C}$ of clauses.

Definition 2.2. Let $\mathcal{C}$ be a set of clauses over variables $x_0, \ldots, x_{n-1}$. We say that an assignment $\alpha \in \{0,1\}^n$ *satisfies* $\mathcal{C}$ if α satisfies all of the clauses in $\mathcal{C}$, or equivalently α satisfies the CNF formula

$$\varphi(x_0, \ldots, x_{n-1}) = \bigwedge_{C \in \mathcal{C}} C(x_0, \ldots, x_{n-1}).$$

Intuition. The intuition behind resolution can be seen from the following example. Consider a CNF-SATISFIABILITY instance that includes two clauses $C = (\neg x_0 \vee x_1)$ and $D = (\neg x_1 \vee x_2)$, along with other clauses. Since a satisfying assignment must make both C_1 and C_2 true, there is an implicit dependence between x_0 and x_2:

Following the definition, this is precisely the resolvent:

$$C \diamond D = (\neg x_0 \vee x_1) \diamond (\neg x_1 \vee x_2) =$$

Example 2.3.

$$(x_0 \vee \neg x_1 \vee x_3 \vee \neg x_5) \diamond (x_1 \vee \neg x_4 \vee \neg x_5) =$$

Example 2.4. We could also have a clause that appears to be resolvable in two ways:

$$(x_0 \vee x_1 \vee \neg x_4) \diamond (\neg x_0 \vee x_2 \vee x_4) =$$

The following theorem gives us a criterion to decide if a set of clauses is satisfiable. Note that resolution plays a crucial rule here.

Theorem 2.5 (Resolution Theorem)**.** *Let $\mathcal{C}$ be a set of clauses over n variables $x_0, \ldots, x_{n-1}$. Suppose that $\mathcal{C}$ is* closed under resolution*, meaning that for every $C, D \in \mathcal{C}$, we have $C \diamond D \in \mathcal{C} \cup \{1\}$. Then:*

1. *( ) $\in \mathcal{C}$ iff*
2. *If ( ) $\notin \mathcal{C}$, then* `ExtractAssignment(`$\mathcal{C}$`)` *finds where* `ExtractAssignment` *is an algorithm described in Section 3.*

Thus we can obtain an algorithm for solving CNF-SATISFIABILITY as follows. We start with the set of clauses $C_0, C_1, \cdots, C_{m-1}$ that appear in the CNF φ, simplify all the clauses in φ and then:

1. Resolve C_0 with each of $C_1, \ldots, C_{m-1}$, adding any new clauses obtained from the resolution $C_m, C_{m+1}, \ldots$. If the empty clause ( ) is found, return `unsatisfiable`.

2. Resolve C_1 with each of $C_2, \ldots, C_{m-1}$ as well as with

3. Resolve C_2 with each of $C_3, \ldots, C_{m-1}$ as well as with

4. etc.

5. Run `ExtractAssignment` on the set of all clauses and return the satisfying assignment.

Pseudocode is given as Algorithm 2.2.

```
   ResolutionInOrder(φ):
   Input        : A CNF formula φ(x_0, ..., x_{n-1})
   Output       : Whether φ is satisfiable or unsatisfiable
 0 Let C_0, C_1, ..., C_{m-1} be the clauses in φ, after simplifying each clause;
 1 i = 0 ;  /* clause to resolve with others in current iteration */
 2 f = m ;     /* start of 'frontier' - new resolvents from current
     iteration */
 3 g = m ;                                          /* end of frontier */
 4 while f > i + 1 do
 5 |   foreach j = i + 1 to f − 1 do
 6 |   |   R = C_i ⋄ C_j;
 7 |   |   if R = 0 then return unsatisfiable;
 8 |   |   else if R ∉ {1, C_0, C_1, ..., C_{g-1}} then
 9 |   |   |   C_g = R;
10 |   |   |   g = g + 1;
11 |   f = g;
12 |   i = i + 1
13 return ExtractAssignment((C_0, C_1, ... C_{g-1}))
```

Algorithm 2.2: A Resolution-based SAT Algorithm

Example 2.6. Let's apply Algorithm 2.2 to the following formula:

$$\phi(x_0, x_1, x_2) = (\neg x_0 \vee x_1) \wedge (\neg x_1 \vee x_2) \wedge (x_0 \vee x_1 \vee x_2) \wedge (\neg x_2)$$

Example 2.7. $\psi(x_0, x_1, x_2, x_3) = (\neg x_0 \vee x_3) \wedge (x_0 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3)$

.

3 Assignment Extraction

We begin by describing the `ExtractAssignment` algorithm for finding a satisfying assignment to a set $\mathcal{C}$ of clauses that are closed under resolution and don't contain $(\ )$. Specifically, we generate our satisfying assignment one variable v at a time in the following manner:

1. If $\mathcal{C}$ contains a singleton clause (v), then
2. If it contains a singleton clause $(\neg v)$ then
3. If it contains neither (v) nor $(\neg v)$, then
4. $\mathcal{C}$ cannot contain both (v) and $(\neg v)$, because

Once we have assigned a variable to a value, we set that variable's value in every clause and simplify.

Formally, the algorithm works as follows:

`ExtractAssignment`($\mathcal{C}$):
Input : A closed and simplified set $\mathcal{C}$ of clauses over variables $x_0, \ldots, x_{n-1}$ such that $(\ ) \notin \mathcal{C}$
Output : An assignment $\alpha \in \{0,1\}^n$ that satisfies all of the clauses in $\mathcal{C}$

0 **foreach** $i = 0, \ldots, n-1$ **do**
1 **if** $(x_i) \in \mathcal{C}$ **then** $\alpha_i = 1$;
2 **else** $\alpha_i = 0$;
3 $\mathcal{C} = \mathcal{C}|_{x_i = \alpha_i}$, meaning that we set $x_i = \alpha_i$ and then simplify all clauses
4 **return** α

Algorithm 3.2: Assignment extraction algorithm

Example 3.1. Consider applying Algorithm 3.2 to the set of clauses derived from the formula in Example 2.7 above:

$$(\neg x_0 \vee x_3), (x_0 \vee \neg x_3), (\neg x_1 \vee x_2), (\neg x_2 \vee x_1), (\neg x_3), (\neg x_0)$$

4 Proof Sketches of Runtime and Correctness

Proof Sketch of Theorem 2.5. First, suppose $\mathcal{C}$ is such that $(\,) \in \mathcal{C}$.

Conversely, if $(\,) \notin \mathcal{C}$, then Item 2 says that `ExtractAssignment` will find a satisfying assignment, so $\mathcal{C}$ is satisfiable. Thus we turn to sketching the proof of Item 2. The key point is to show that after assigning a variable x_i as in `ExtractAssignment`, the set of clauses $\mathcal{C}|_{x_i=\alpha_i}$ satisfies the following:

□

Now, we consider the runtime and correctness of the resolution algorithm. Let $\mathcal{C}_{fin}$ be the final set of clauses produced in Algorithm 2.2. Let k_{fin} be the maximum *width* (number of literals) among the clauses in $\mathcal{C}_{fin}$.

Runtime. Before analysing the runtime of the resolution algorithm, let's understand why the resolution algorithm terminates.

We can now give a finer estimate of the runtime of the resolution algorithm.

Correctness:

5 Efficient algorithm for 2-SAT

Although the worst-case running time of resolution is exponential, the following is an important special case where it has polynomial runtime.

Input: A CNF formula φ on n variables in which each clause has width at most 2 (i.e. contains at most 2 literals)
Output: An $\alpha \in \{0,1\}^n$ such that $\varphi(\alpha) = 1$, or $\perp$ if no satisfying assignment exists

Computational Problem 2-SAT

Runtime of the resolution algorithm for 2-SAT**:** observe that on a 2-SAT instance φ, Algorithm 2.2 will never create a clause of size larger than 2:

Corollary 5.1. 2-SAT *can be solved in time* _____.

There is an algorithm known for 2-SAT with runtime $O(n + m)$, where m is the number of clauses, by reduction to finding strongly connected components of directed graphs. (This algorithm is covered in CS1240.) Unfortunately, just like with coloring, once we switch from $k = 2$ to $k = 3$, the best known algorithms have exponential ($O(c^n)$) worst-case runtimes.

6 SAT Solvers

Enormous effort has gone into designing SAT Solvers that perform well on many real-world satisfiability instances, often but not always avoiding the worst-case exponential complexity. These methods are very related to Resolution. In some sense, they can be viewed as interleaving the `ExtractAssignment` algorithm and Resolution steps, in the hope of quickly finding either a satisfying assignment or a proof of unsatisfiability.