| | |
|---|---|
| **CS1200: Intro. to Algorithms and their Limitations** | Anshu & Vadhan |
| Lecture 16: Resolution | |
| *Harvard SEAS - Fall 2024* | *Oct. 29, 2024* |

# 1 Announcements

- SRE 5 today
- Midterm regrade requests due tonight.
- Anurag's in person OH moved to Thur 11AM - 12 PM.
- A reminder that both the psets and lectures are part of the learning material.
- Pset5 reflection responses:
  - Most common questions students struggled with - Word-RAM to RAM simulations, Asymptotic runtime analysis and notation.
  - Students found that struggling through the problem was valuable and led to better understanding and problem solving skills.
  - Students learned how to break down problems and to ask for help, which is great!

# 2 Recap

- A *literal* is
- A boolean formula is in *conjunctive normal form (CNF)* if
- It will be convenient to also allow 1 (true) to be a clause. 0 (false) is already a clause:

| | |
|---|---|
| **Input** | **:** A CNF formula $\varphi$ on $n$ variables |
| **Output** | **:** An $\alpha \in \{0,1\}^n$ such that $\varphi(\alpha) = 1$ (if one exists) |

**Computational Problem** CNF-Satisfiability (SAT)

**Simplifying clauses.** Note terms and clauses may contain duplicate literals, but if a term or clause contains multiple copies of a variable $x$, it's equivalent to a term or clause with just one copy (since $x \vee x = x$ and $x \wedge x = x$). We can also remove any clause or term with both a variable $x$ and its negation $\neg x$, as that clause or term will be always true (in the case of a clause). We define a function Simplify which takes a clause and performs those simplifications:

Motivation for SAT (and logic problems in general): can encode many other problems of interest

- Graph Coloring (last time)
- Longest Path (SRE 5)
- Independent Set (section)
- Program Analysis (lec24)
- and much more (lec19)

Unfortunately, the fastest known algorithms for Satisfiability have worst-case runtime exponential in $n$. However, enormous effort has gone into designing heuristics that complete much more quickly on many real-world instances.

## 3 Resolution

SAT Solvers are algorithms to solve CNF-Satisfiability. Although they have worst-case exponential running time, on many "real-world" instances, they terminate more 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 we have 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).

**Definition 3.1** (resolution rule)**.** For clauses $C$ and $D$, define their *resolvent* to be

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

In the special case where $C = \ell, D = \neg\ell$, we use our definition from Lecture 15 that empty clause is always false and obtain

$$(\ell) \diamond (\neg\ell) =$$

**Intuition:** The intuition behind resolution can be seen from the following example. Consider two clauses $C_1 = (\neg x_0 \vee x_1)$ and $C_2 = (\neg x_1 \vee x_2)$. If both $C_1, C_2$ are required to be true (which is the goal of the CNF-Satisfiability problem), there is an implicit dependence between $x_0$ and $x_2$, as follows.

Following the definition, this is precisely the resolvent of $C_1, C_2$:

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

**Example 2:**

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

**Example 3:** We could also have a clause that appears to be resolvable in two ways:

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

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

**Definition 3.2.** 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 $\alpha$ satisfies all of the clauses in $\mathcal{C}$, or equivalently $\alpha$ satisfies the CNF formula

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

The following lemma says that adding resolvents does not change the set of satisfying assignments.

**Lemma 3.3.** *Let $\mathcal{C}$ be a set of clauses and let $C, D \in \mathcal{C}$. Then $\mathcal{C}$ and $\mathcal{C} \cup \{C \diamond D\}$ have the same set of satisfying assignments (if any).*

The following theorem tells us under what conditions a set of clauses is satisfiable or unsatisfiable.

**Theorem 3.4** (Resolution Theorem)**.** *Let $\mathcal{C}$ be a set of clauses over 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 $C \diamond D \in \mathcal{C}$. Then:*

1. *$\emptyset \in \mathcal{C}$ iff*

2. *If $\emptyset \notin \mathcal{C}$, then*

To turn this theorem into an algorithm, we can start with a set $\mathcal{C}$ of clauses from a CNF formula and keep adding resolvents until we cannot add any new ones. If we find the empty clause, we know $\varphi$ is unsatisfiable by Theorem 3.4.

There are many variants of resolution, based on different ways of choosing the order in which to resolve clauses. We give a particular version below, where starting with a set of clauses $C_0, C_1, \cdots, C_{m-1}$, simplify all the clauses in $\varphi$ and then:

1. Resolve $C_0$ with each of $C_1, \ldots, C_{m-1}$, adding any new clauses obtained from the resolution $C_m, C_{m+1}, \ldots$

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.

Note that this process will resolve every pair of clauses, except for resolving $C_i$ with resolvents of the form $C_i \diamond C_j$ for $j > i$. Omitting the latter is harmless by the following lemma:

**Lemma 3.5.** *For all clauses $C$ and $D$, $C \diamond (C \diamond D) = 1$*

*Proof.* $\square$

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

**Example 2:** $\psi(x_0, x_1, x_2, x_3) = (\neg x_0 \vee x_3) \wedge (x_0 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3)$

The resolution algorithm can be written as follows in pseudo-code:

`ResolutionInOrder(`$\varphi$`)`<br>
**Input** : A CNF formula $\varphi(x_0, \dots, x_{n-1})$<br>
**Output** : Whether $\varphi$ is `satisfiable` or `unsatisfiable`<br>
Let $C_0, C_1, \dots, C_{m-1}$ be the clauses in $\varphi$, after simplifying each clause;<br>
$i = 0$ ; `/* clause to resolve with others in current iteration */`<br>
$f = m$ ; `/* start of ‘frontier’ - new resolvents from current iteration */`<br>
$g = m$ ; `/* end of frontier */`<br>
**while** $f > i + 1$ **do**<br>
    **foreach** $j = i + 1$ *to* $f - 1$ **do**<br>
        $R = C_i \diamond C_j$;<br>
        **if** $R = 0$ **then return** `unsatisfiable`;<br>
        **else if** $R \notin \{C_0, C_1, \dots, C_{g-1}\}$ **then**<br>
            $C_g = R$;<br>
            $g = g + 1$;<br>
    $f = g$;<br>
    $i = i + 1$<br>
**return** `satisfiable`

Algorithm 15 raises two questions:

1. (Termination) Why does resolution always terminate? And what is its runtime?
2. (Correctness) Is Algorithm 15 correct? If it ever derives the empty clause $R = 0$, we know that $\varphi$ is unsatisfiable (why?) but if never generates the empty clause, can we be sure that $\varphi$ is satisfiable?

# 4 Termination and Efficiency

**Q:** Why does resolution terminate?

**A:**

**Q:** What is the runtime of resolution?

**A:**

However, in many cases, there is a *short* proof of unsatisfiability that resolution will find. One case is for the 2-SAT problem, defined as follows:

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

**Computational Problem** $k$-SAT

**Q:** What is the runtime of Resolution for 2-SAT?

**A:**