**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)
- 3D Matching (pset 7)
- 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:

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