CS120: Intro. to Algorithms and their Limitations Hesterberg & Vadhan

Section 8: Logic and Resolution

Harvard SEAS - Fall 2022 *Oct. 27, 2022*

1 Propositional Logic

Definition 1.1 (informal)**.** A *boolean formula* φ is a formula built up from a finite set of variables, say $x_0, \ldots, x_{n-1}$, using the logical operators $\wedge$ (AND), $\vee$ (OR), and $\neg$ (NOT), and parentheses.

Every boolean formula φ on n variables defines a boolean function, which we'll abuse notation and also denote by $\varphi : \{0,1\}^n \to \{0,1\}$, where we interpret 0 as false and 1 as true, and give $\wedge, \vee, \neg$ their usual semantics (meaning).

Definition 1.2. A *literal* is a variable (e.g. x_i) or its negation ($\neg x_i$).

A boolean formula is in *disjunctive normal form (DNF)* if it is the OR of a sequence of *terms*, each of which is the AND of a sequence of literals.

A boolean formula is in *conjunctive normal form (CNF)* if it is the AND of a sequence of *clauses*, each of which is the OR of a sequence of literals.

Example:

$$\varphi_{maj}(x_0, x_1, x_2) = (x_0 \wedge x_1) \vee (x_1 \wedge x_2) \vee (x_2 \wedge x_0)$$

is a DNF formula. It evaluates to 1 if at least two of its inputs are 1. For instance, $\varphi_{maj}(1,1,0) = 1$ and $\varphi_{maj}(1,0,0) = 0$.

Lemma 1.3. *For every boolean function $f : \{0,1\}^n \to \{0,1\}$, there are boolean formulas φ and ψ in DNF and CNF, respectively, such that $f \equiv \varphi$ and $f \equiv \psi$, where we use $\equiv$ to indicate equivalence as functions, i.e. $f \equiv g$ iff $\forall x : f(x) = g(x)$.*

Question 1.4. Can there be multiple ways to represent the same boolean function f as a CNF or DNF formula?

2 Satisfiability

We're interested in finding variable assignments that will *satisfy* (and hence the name) the boolean formulas we just described.

Input	**:** A boolean formula φ on n variables
Output	**:** An $\alpha \in \{0,1\}^n$ such that $\varphi(\alpha) = 1$, or $\perp$ if no satisfying assignment exists

Computational Problem Satisfiability

Input	**:** A CNF formula φ on n variables
Output	**:** An $\alpha \in \{0,1\}^n$ such that $\varphi(\alpha) = 1$, or $\perp$ if no satisfying assignment exists

Computational Problem CNF-Satisfiability

Input	: A DNF formula φ on n variables
Output	: An $\alpha \in \{0,1\}^n$ such that $\varphi(\alpha) = 1$, or $\perp$ if no satisfying assignment exists

Computational Problem DNF-Satisfiability

Question 2.1. We mentioned special cases of Satisfiability, 2SAT and 3SAT, in lecture. What *two* constraints differentiate the 3SAT problem from the normal Satisfiability problem?

3 Reductions to SAT

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

- Graph Coloring (Lecture 14)
- Longest Path (SRE 4, upcoming)
- Independent Set (ps7)
- and many more—all the computational problems we've seen in class so far (except a few mentioned in lecture 0) reduce efficiently to SAT.

Remember our definition of a **reduction**: If Problem A *reduces to* Problem B, we can solve Problem A using an oracle for Problem B. Intuitively, you can think of this as saying that Problem A is *no harder than* Problem B. Now, we're going to apply those same principles to reduce to satisfiability problems. Typically, a reduction with one oracle call goes like this:

Algorithm for Problem A by reducing to Problem B

- Pre-process the input: $R_{pre}(x)$.
- Call the oracle, which solves Problem B on the transformed input: $B(R_{pre}(x))$.
- Post-process the output: $R_{post}(B(R_{pre}(x)))$. Typically $R_{post}(y)$ outputs $\perp$ iff $y = \perp$; we will assume this in the correctness proof below.

After specifying the reduction, you will often be asked to prove the correctness of the reduction. In other words, you need to show that a solution to Problem B on the transformed input can be used to construct a solution to Problem A on the original input.

A typical proof of correctness of a reduction from $A = (\mathcal{I}, f)$ to $B = (\mathcal{J}, g)$

- $x \in \mathcal{I}$ and $A(x) = \emptyset$ implies that $g(R_{pre}(x)) = \emptyset$. This implies that we correctly output $\perp$ when there is no solution to Problem A.
- $x \in \mathcal{I}$ and $A(x) \neq \emptyset$ implies that $R_{pre}(x) \in \mathcal{J}$ and $g(R_{pre}(x)) \neq \emptyset$. Furthermore, if $y \in g(R_{pre}(x))$, then $R_{post}(y) \in f(x)$. This implies that when there is a solution to Problem A on input x, we will find one.

And just like that, we've leveraged a solution to Problem B to solve Problem A. When we complete a reduction, we also need to bound the total pre- and post-processing runtime. If that runtime is polynomial, we say that there is a *polynomial-time reduction* from Problem A to Problem B.

Question 3.1. (IndependentSet $\leq$ SAT)

1. Show how, given $n, k \in \mathbb{N}$, to construct in time $O(nk)$ a CNF formula $\psi_{n,k}(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{m-1})$ such that:

$$\exists y \ \psi_{n,k}(x, y) = 1 \Leftrightarrow \text{at least } k \text{ of the } x_i\text{'s are 1.}$$

   Here $x = (x_0, \ldots, x_{n-1})$ and $y = (y_0, \ldots, y_{m-1})$. (Hint: for each $i = 0, \ldots, n-1$ and $j = 0, \ldots, k$, introduce a variable $y_{i,j}$ representing whether or not at least j of the variables $x_0, \ldots, x_{i-1}$ equal 1. Write clauses that force $y_{i,j}$ to have the correct value assuming that $y_{i-1,j}$ and $y_{i-1,j-1}$ have the correct value.)

2. Recall the IndependentSet-ThresholdSearch (IS-TS) problem from Problem Set 5: given a graph G and a number $k \in \mathbb{N}$, find an independent set of size at least k in G (if one exists). Using Part 1, show that IS-TS $\leq_{O(nk+m),O(nk+m)}$ SAT. (Here the size parameters of the IS-TS problem are the number n of vertices, the number m of edges, and the threshold k. For this problem, we define the size of a SAT instance to be the sum of the lengths of its clauses.)

Question 3.2. Prove that SAT reduces to 3SAT.

4 Resolution

Definition 4.1 (clause simplification)**.** Given a clause B, Simplify(B) returns 1 if B contains both a literal and its negation, and otherwise removes duplicates of literals from B and sorts the variables in B according to a fixed ordering on the variables (e.g. x_0, x_1, ...).

Definition 4.2 (resolution rule)**.** For clauses C and D, define

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

Examples:

$$(x_1 \vee x_3) \diamond (\neg x_2 \vee x_4) = 1$$

$$(x_2 \vee \neg x_1) \diamond (x_2 \vee x_3 \vee x_4) = 1$$

$$(x_1 \vee x_2) \diamond (\neg x_2 \vee x_3 \vee x_4) = (x_1 \vee x_3 \vee x_4)$$

If we can resolve more than one variable, it seems like we would have 2 options. For example,

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

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

But notice that in both cases they will be true (because this is a tautology), and so we resolve the clause to 1 (this is part of the simplification).

Moreover, we always remove duplicate literals:

$$(x_1 \vee \neg x_2 \vee x_4) \diamond (x_1 \vee x_2 \vee x_3) = (x_1 \vee x_3 \vee x_4).$$

Lastly, recall that the empty clause is equal to false:

$$(x_3) \diamond (\neg x_3) = \emptyset = FALSE.$$

From now on, we can think of a CNF formula as a set of clauses $\mathcal{C}$.

Definition 4.3. 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

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

Lemma 4.4. *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. In particular, if $C \diamond D$ is the empty clause, then $\mathcal{C}$ is unsatisfiable.*

Lemma 4.5. *For all clauses C and D, $C \diamond (C \diamond D) = 1$*

Question 4.6. Resolve the following clauses:

1. $(x_1 \vee \neg x_2) \diamond (x_3 \vee \neg x_4)$.
2. $(x_3 \vee x_4 \vee \neg x_2) \diamond (x_2 \vee x_3 \vee x_1)$.
3. $(x_2) \diamond (\neg x_2)$.
4. $(x_1 \vee x_2 \vee x_3) \diamond (x_1 \vee \neg x_2 \vee \neg x_3)$.

We can use the resolution rule to decide the satisfiability of a CNF formula φ: keep adding resolvents until we find the empty clause (in which case we know φ is unsatisfiable by Lemma 4.4) or cannot generate any more new clauses.

We will use the following algorithm for this process: `ResolutionInOrder`. Note that there are actually many variants of resolution, based on which order we choose the clauses. For this week's

pset, you should process the clauses in the order that the following pseudocode indicates:

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

If the algorithm finds an empty set, then we conclude that φ is unsatisfiable. On the other hand, if the algorithm finishes the loops without having found an empty set, we conclude that φ is satisfiable.

Note that the role of the variable f above is to ensure that we do not resolve a clause C_i with another clause that has been "created" by C_i. E.g., if $C_4 = C_1 \diamond C_3$, then we should not resolve C_1 with C_4 (although notice that we do for C_j). Again, it might be useful to think of the previous algorithm as maintaining a list of clauses that grow as we resolve more clauses.

Question 4.7. Why does resolution always terminate?

Question 4.8. Run the algorithm `ResolutionInOrder` on the following φ:

$$\varphi(x_1, x_2, x_3, x_4) = (x_2 \vee x_4 \vee \neg x_1) \wedge (x_2 \vee \neg x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_3) \wedge (x_2 \vee \neg x_1).$$

Question 4.9. Run the algorithm `ResolutionInOrder` on the following ϕ:

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

Question 4.10. Prove Lemma 4.5. That is, if C and D are clauses, then $C \diamond (C \diamond D) = 1$.

Question 4.11. Find clauses C, D, and E such that $(C \diamond D) \diamond (C \diamond E) = 0$