Algorithms & Complexity: Lecture 3, Completeness and Reductions

Sam Barrett

February 8, 2021

1 SAT and its variants

1.1 Propositional connectives

A basic reminder of Propositional logic and connectives:

- `T` (`True`) and `F` (`False`) are the propositional **constants**
- $a \wedge b$ is `True` if both a and b are `True` , otherwise `False` . **Conjunction**
- $a \vee b$ is `True` if either a or b is `True` , otherwise `False` . **Disjunction**
- $\neg a$ is `True` if a is `False` and vice versa. **Negation**
- $a \rightarrow b$ is `True` if either a is `False` or b is `True` , but `False` otherwise. **Implication**

> **Lemma 1** ***A propositional expression can be evaluated in linear time.*** *This is done using the shunting yard algorithm to translate into postfix notation and then evaluating using a stack.*

1.2 Conjunctive normal form

A formula is in CNF when it is a conjunction of disjunctions of variables and their negations.

For example,

$$(u_0 \vee \bar{u_1} \vee u_2) \wedge (u_1 \vee \bar{u_2} \vee u_3) \wedge \underbrace{(u_0 \vee \bar{u_2} \vee \bar{u_3})}_{\text{clause}}$$

Where in the above example $\bar{u}$ is the negation of u.

The disjunctions within the formula are called **clauses** and the variables are called **literals**

A clause can be written as $u \rightarrow (v \vee w \vee x)$ rather than $\bar{u} \vee v \vee w \vee x$

1.2.1 3CNF formulae

A CNF formula is **3CNF** when each clause has at most 3 literals

Note: any 3CNF clause can be written as an implication

1.3 Satisfiability

Satisfiability is the process of answering questions of the form: *Over the variables* p, q, r *is the formula* $(\neg(q \to p) \wedge r) \vee (p \wedge q)$ satisfiable?

In this particular example, the answer is *yes*, in the case where $p = \mathtt{F}$ and $q = r = \mathtt{T}$

1.3.1 Formula-SAT

`Formula-SAT` is the set of all formulas that are satisfiable.

`Formula-SAT` is in **NP**, this is the case as given a formula ϕ, and an interpretation u,

- the length of u is no longer than that of ϕ
- it takes linear time to test whether it is a satisfying assignment by Lemma 1.

1.3.2 SAT

`SAT` is the set of CNF formulae that are satisfiable. Since `SAT` is a special case of `Formula-SAT` (which is in **NP**), it too is in **NP**

1.3.3 3SAT

`3SAT` is the set of 3CNF formulae that are satisfiable. Again, since it is a special case of `SAT`, it too is in **NP**.

2 Reductions

We often want to reduce a problem in mathematics/ Computer science to another, simpler or more understood problem. Intuitively, this can be thought of in the same way as reducing the problem of making *profiteroles* to the problem(s) of making cream-filled pastries and making chocolate sauce.

Let L and L' be languages.

A (many-to-one) **reduction** from L to L' is a function $f : \{0,1\}^* \to \{0,1\}^*$ such that for any bitstring, x we have $x \in L$ iff $f(x) \in L'$.

Or, more plainly, if we know how to decide membership fo L', then the reduction enables us to decide membership of L.

2.1 Computable reductions

We write $L \leq_m L'$ when there is a reduction from L to L' that is **computable**. From this we can see that:

- If L' is decidable, then L is decidable
- If L is undecidable (*e.g. Halting problem*), then L' is undeciable .

This is a very useful property and allows us to easily prove the deciability or undecidability of problems without explicity having to prove them. We will **not** look any closer in this module.

2.2 Polynomial time reductions

We write $L \leq_P L'$ when there is a reduction from L to L' that is **polynomial time**.

- If L' is in **P**, then L is also in **P**
- If L' is in **NP**, then L is also in **NP**

2.3 NP-Completeness

A language L is **NP**-hard if **every** language in **NP** has a polynomial-time reduction to it.

Therefore, if L is in **P** and **NP**-hard then **P** = **NP**!

If L is in **NP** and also **NP**-hard, we say that it is **NP**-complete. These are the *hardest* problems in **NP**.

2.3.1 Proving NP-completeness

To prove that a problem is **NP**-complete:

- One must show that it is in **NP**
- One must show that some other **NP**-hard problem reduces to it.

3 The Cook-Levin theorem

Theorem 1 *3SAT is* **NP***-complete*

We know that 3SAT is in **NP**. Therefore, to show that it is **NP**-complete, we must show it to also be in **NP**-hard.

For any language $L \in$ **NP** we want to give a polytime reduction from L to 3SAT.

We will approach this in order from Formula-SAT $\rightarrow$ SAT $\rightarrow$ 3SAT

3.1 Reducing to Formula-SAT

Since L is in **NP** there must be a nondeterministic Turing machine which decides it.

Say that M is a NDTM for the language L, using an input tape, a work tape and an alphabet $\{\triangleright, \square, 0, 1\}$ with 50 states and a running time and space usage of at most n^3, where n is the size of the input.

From this, we must convert a bitstring x of length n into a propositional logic formula that is satisfiable iff $x \in L$.

<u>The variables</u>

- Let $a_{i,j,s}$ say that at time i, cell j of the work tape contains symbol s. Here $i, j < n^3$
- Let $b_{i,j}$ say that, at time i the input head is in position j. Here $i < n^3$ and $j < n$.
- Let $c_{i,j}$ say that, at time i, the work head is in position j. Here $i, j < n^3$
- Let $d_{i,q}$ say that, at time i the current work state is q. here $i < n^3$ and $q < 50$ (as per machine definition)

<u>The constraints</u>

- For any time i, each cell j contains only one symbol and there is only one current state.
- The configurations at time i and time $i+1$, and the input, are related by the transition function.

  This is stated locally, meaning if, at time i the state at time $i+1$ is determined only by adjacent states.
- At some time $i < n^3$, the current state is $q_{\texttt{accept}}$.

Putting these things together gives a formula of size $O(n^3)$, It is satisfiable iff the bitstring x is acceptable ($x \in L$).

4 Logspace reductions