

# Algorithms & Complexity: Lecture 3, Completeness and Reductions

Sam Barrett

March 4, 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**

### 1.2.2 Conversion to negation-free form

A formula is negation free when there are no occurrences of  $\neg$  or  $\rightarrow$ . However these are still permitted in the form of negated variables which are still literals.

Every formula is equivalent to one in negation-free form. Simply push in each negation to a literal using de Morgan's laws:

$$\begin{aligned}\neg(\psi \vee \psi') &= (\neg\psi) \wedge (\neg\psi') \\ \neg(\psi \wedge \psi') &= (\neg\psi) \vee (\neg\psi')\end{aligned}$$

Each push is  $O(n)$ -time, so the overall conversion is in  $O(n^2)$  time.

### 1.2.3 Negation-free to CNF

A negation-free formula  $\phi$  can be converted to a CNF formula  $\phi'$  using extra free variables that are **equisatisfiable** with  $\phi$ . This means that the new formula will be satisfiable iff  $\phi$  is satisfiable.

This is done via induction on  $\phi$ :

- The case where  $\phi$  is a literal is clear, literals are already in CNF
- The case where  $\phi$  is a conjunction is clear, it is already in CNF
- What if  $\phi$  is a disjunction?

For any variable  $c$  and clause  $\phi$ , the formula  $c \rightarrow \phi$  is equivalent to the clause  $\bar{c} \vee \phi$ .

Therefore, any variable  $c$  and CNF formula  $\phi$ , the formula  $c \rightarrow \phi$  is equivalent to a CNF formula by the law:

$$c \rightarrow (\psi \wedge \psi') = (c \rightarrow \psi) \wedge (c \rightarrow \psi')$$

For any CNF formulas  $\phi$  and  $\phi'$ , the formula  $\phi \vee \phi'$  is equisatisfiable with:

$$(c \vee c') \wedge (c \rightarrow \phi) \wedge (c' \rightarrow \phi')$$

Which is equivalent to a CNF formula, obtained in  $O(n)$  time. Thereby, we have a conversion to CNF in  $O(n^2)$

### 1.2.4 CNF to 3CNF

In CNF, each clause is of the form:

$$a \rightarrow (b_0 \vee \dots \vee b_{n-1} \vee c \vee c')$$

is equisatisfiable with:

$$\begin{aligned} & (a \rightarrow b_0 \vee d_0) \\ & \wedge (d_0 \rightarrow b_1 \vee d_1) \\ & \dots \\ & \wedge (d_{n-2} \rightarrow b_{n-1} \vee d_{n-1}) \\ & \wedge (d_{n-1} \rightarrow c \vee c') \end{aligned}$$

Where  $d_0, \dots, d_{n-1}$  are *fresh* variables.

This gives an  $O(n)$  time conversion to 3CNF

## 1.3 Satisfiability

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

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

### 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  $\phi$ , and an interpretation  $u$ ,

- the length of  $u$  is no longer than that of  $\phi$
- 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  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  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 of  $L'$ , 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 undecidable .

This is a very useful property and allows us to easily prove the decidability or undecidability of problems without explicitly 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  $L \in \mathbf{NP}$  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  $x \in L$ .

The variables

- 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)

The constraints

- 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  $q_{\text{accept}}$ .

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

### 3.2 Reduction to SAT

By converting a formula to an equisatisfiable CNF formula in  $O(n^2)$  time (See Section 1.2.3), we show that Formula-SAT  $\leq_P$  SAT.

### 3.3 Reduction to 3SAT

By converting a CNF formula to an equisatisfiable 3CNF formula in  $O(n)$  time we show that SAT  $\leq_P$  3SAT

### 3.4 Proving NP-completeness

We previously outlined how to prove a problems is **NP**-complete in Section 2.3.1. We have shown that 3SAT reduces to **NP**-hard thus satisfying the second point.

## 4 Logspace reductions

We know that  $\mathbf{L} \subseteq \mathbf{P}$ , i.e every decision problem that can be solved in logspace can be solved in polynomial time.

### 4.1 Requirements

- We will write  $L \leq_L L'$  when there is a logspace reduction from  $L$  to  $L'$ .
- We *want* the identity reduction to be logspace, i.e.  $L \leq_L L$ .
- We want a composite of logspace reductions to be logspace, so that:

$$L \leq_L L' \leq_L L'' \implies L \leq_L L''$$

- If you have two languages that are related,  $L \leq_L L'$  then  $L' \in \mathbf{L}$  should imply  $L \in \mathbf{L}$
- Every logspace reduction should be polytime, so that,  $\leq_L$  implies  $\leq_P$ .

### 4.2 Logspace reduction: definition

We impose the following requirements on a function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ :

- $f$  must be **polynomially bounded**, meaning, there must be a  $c$  such that, for every bitstring  $x$ ,  $|f(x)| \leq |x|^c$  holds true.
- We can test in logarithmic space whether a particular position in the output it within or outside of the length of  $f(x)$ . Formally:  
The set of pairs  $\langle x, i \rangle$ , s.t.  $i \leq |f(x)|$  **must** be in  $\mathbf{L}$
- We can test whether a particular position  $\langle x, i \rangle$  gives a result of 1 or not,  $f(x)_i = 1$  must be in  $\mathbf{L}$ . This is referred to as the **bitwise** problem for  $f$ . (This is the most important condition)

### 4.3 Composing logspace reductions

Suppose we have two logspace reductions  $f$  and  $g$ , then the composite function  $x \mapsto g(f(x))$  is also logspace, we are going to show that this is the case:

To compute  $g(f(x))_i$  (the  $i^{th}$  bit of the result) using three work tapes (A,B,C), we assume we can compute  $f$  and  $g$  using one work tape each. We assign  $f$  work tape C and  $g$  work tape A.

We cannot use tape B as an input tape for  $g$  as this would take too much space, resulting in a non-logspace computation. Instead, we use a *virtual* input tape. This means that the current input position  $j$  is stored on work tape B (using a logarithmic amount of space), and in each step we work out  $f(x)_j$ , using work tape C.

All of these components are logspace, meaning our composition function  $x$  is also logspace as if  $L \leq_L L'$  then  $L' \in \mathbf{L}$  implies  $L \in \mathbf{L}$ .

### 4.4 Logspace reduction are polytime

Let  $f$  be a logspace reduction. The bitwise problem is in  $\mathbf{L}$  and therefore also in  $\mathbf{P}$ .

So, for any  $x$ , the length of  $f(x)$  is polynomially bounded and each bit can be computed in polynomial time, allowing us to compute  $f(x)$  in polynomial time (polynomially many steps over polynomially bits).

### 4.5 Application: P-completeness

Just as polytime reductions give a reasonable notion of  $\mathbf{NP}$ -completeness, so logspace reductions give a reasonable notion of  $\mathbf{P}$ -completeness.

With  $\mathbf{P}$ -completeness, we cannot simply look to the degree of the polynomial to determine how *hard* it is, as these are infinite and so  $\mathbf{P}$  would be the same as  $\mathbf{P}$ -complete. We instead need a different measure.

**Definition 1** (*P-completeness*) *A problem is P-complete if it is in P and every problem in P logspace-reduces to it.*