Lecture 13: NP-completeness

Valentine Kabanets

November 3, 2016

1 Polytime mapping-reductions

We say that A is *polytime reducible* to B if there is a polytime computable function f (a reduction) such that, for every string x, $x \in A$ iff $f(x) \in B$. We use the notation "$A \leq_p B$." (This is the same as a notion of mapping reduction from Computability we saw earlier, with the only change being that the reduction f be *polytime* computable.)

It is easy to see

Theorem 1. *If $A \leq_p B$ and $B \in \mathsf{P}$, then $A \in \mathsf{P}$.*

2 NP-completeness

2.1 Definitions

A language B is NP-*complete* if

1. B is in NP, and
2. every language A in NP is polytime reducible to B (i.e., $A \leq_p B$).

In words, an NP-complete problem is the "hardest" problem in the class NP.

It is easy to see the following:

Theorem 2. *If B is NP-complete and B is in P, then NP = P.*

It is also possible to show (Exercise!) that

Theorem 3. *If B is NP-complete and $B \leq_p C$ for some C in NP, then C is NP-complete.*

2.2 "Trivial" NP-complete problem

The following "scaled down version of A_{NTM}" is NP-complete.

$$A^p_{NTM} = \{\langle M, w, 1^t\rangle \mid \text{NTM } M \text{ accepts } w \text{ within } t \text{ steps}\}$$

Theorem 4. *The language A^p_{NTM} defined above is NP-complete.*

Proof. First, A^p_{NTM} is in NP, as we can always simulate a given nondeterministic TM M on a given input w for t steps, so that our simulation takes time $poly(|\langle M\rangle|, |w|, t)$ (polynomial in the input size).

To argue that every language $L \in \mathsf{NP}$ reduces to A^p_{NTM}, we take an NTM M deciding L in time n^c, for some constant $c > 0$. We have that $x \in L$ iff M accepts x within $|x|^c$ steps. Thus, $x \in L$ iff $\langle M, x, 1^{|x|^c}\rangle \in A^p_{NTM}$. So the required polytime reduction from L to A^p_{NTM} maps x to $\langle M, x, 1^{|x|^c}\rangle$; it is easy to see that the output of this reduction is indeed computable in deterministic time polynomial in the input size. □

2.3 Natural NP-complete problems

The fact that A^p_{NTM} is NP-complete is pretty simple, and not surprising. What is surprising is that many *natural* problems (not involving Turing machines) also turn out to be NP-complete! One of the first such natural problems shown to be NP-complete was

$$\mathsf{SAT} = \{\langle \phi(x_1, \ldots, x_n)\rangle \mid \text{propositional formula } \phi \text{ is satisfiable}\}.$$

Theorem 5 (Cook-Levin Theorem)**.** SAT *is* NP*-complete.*

Proof. (1) SAT is in NP. (Easy.) (2) Every language L in NP is polytime reducible to SAT. This is what we need to show.

Idea: Take any language L in NP. This L is decided by some NTM M in time n^c, for some constant c. Given M and any input string x, we will construct a formula ϕ_x such that: M accepts x iff ϕ_x is satisfiable.

Intuitively, we can construct such a ϕ_x simulating the computation of a TM because every computer (including the TM) can be implemented using chips/circuits that are built from logical operations like AND, OR, and NOT — precisely the operations used in logical formulas like our ϕ_x.

In more detail, observe that M accepts x iff there is an accepting computation of M on x. That is, there is a sequence of configurations $conf_1, \ldots, conf_{n^c}$ such that:

1. $conf_1$ is the start configuration $(q_{start}x)$,
2. $conf_{i+1}$ follows from $conf_i$ according to the transition rules of M,
3. some $conf_j$ is an accepting configuration.

Our formula ϕ_x will have propositional variables to encode the sequence of configurations of TM M on input x. The formula ϕ_x is satisfiable by an assignment to its variables iff the sequence of configurations encoded by this assignment actually corresponds to a valid accepting computation of M on x. In other words, the formula ϕ_x must check all condition (1)–(3) stated above.

The most interesting (and non-trivial) condition to check is (2). For this, we define a notion of a *window*. For each position i, j of the tableau of computation of M on x, the value of cell (i, j) (at time i, in position j of the tape) is determined by the values of the cells $(i-1, j-1)$, $(i-1, j)$, and $(i-1, j+1)$. Intuitively, to know what happened in position j, you need to know what was there before, and whether the TM was scanning that position or one of its immediate neighbours ($j-1$ or $j+1$). The reason is that a TM can move only one position (left or right) in a single step of computation.

We say that a window (i, j) is *legal* if the value of cell (i, j) (which includes both the tape contents of position j at time i, and whether that position was scanned by the TM, and if so, in what state) is consistent with the values of the cells $(i-1, j-1)$, $(i-1, j)$, and $(i-1, j+1)$. It is not hard to see that a configuration at time i is correctly obtained from configuration at time $i-1$ iff all windows (i, j) (over $1 \leq j \leq n^c$) are legal.

Checking if a given window is legal can be represented by a propositional formula (of constant size) which "hard-wires" the transition function of the given TM M. A conjunction of such windows over all possible positions yields a formula to check if a configuration correctly follows from a previous configuration.

All in all, we get a propositional formula ϕ_x of size $poly(n)$. This formula can be constructed efficiently (in time $poly(n)$) given the description of M and an input x of length n. Thus we get a required reduction from L to SAT. □

3 Is there life between P and NP-complete?

Assuming that P ≠ NP, one can show the existence of problems in NP that are not NP-complete and not in P. We state the following without proof; the proof is a subtle diagonalization argument.

Theorem 6 (Ladner)**.** *Assuming* NP ≠ P*, there exists a language* $L \in$ NP *such that*

1. $L \notin$ P*, and*
2. L *is not* NP*-complete.*