HW1 Solution Set
CPSC 468/568: Computational Complexity
(Spring 2015)

1 Problem 1.4

> Define a bidirectional TM to be a TM whose tapes are infinite in both directions. Show that for every $f : \{0,1\}^* \to \{0,1\}^*$ and time-constructible $T : \mathbb{N} \to \mathbb{N}$, if f is computable in time $T(n)$ by a bidirectional TM M, then it is computable in time $4T(n)$ by a standard (unidirectional) TM $\tilde{M}$.

We complete the proof sketch of claim 1.8. We take a bidirectional tape and "fold" the tape at index 0 to produce a unidirectional tape as shown in claim 1.5. The new alphabet is made up of $|\Gamma^2|$ elements, each a two-tuple $(x, y) : x, y \in \Gamma$ of every possible combination of the elements of Γ. We replace every original rule $(q_i, x) = (q_j, y, z)$ with $(q_i, (x, \perp)) = (q_j, (y, \perp), z)$, and $(q'_i, (\perp, x)) = (q'_j, (\perp, y), \neg z)$. Note: $\neg L = R, \neg R = L, \neg S = S$.

The q states preserve the original direction of movement, while the q' states reverse said direction. For the case where the head reaches the start position, we add the rules $(q_i, (\triangleright, \triangleright)) = (q'_j, (\triangleright, \triangleright), R)$ and $(q'_i, (\triangleright, \triangleright)) = (q_j, (\triangleright, \triangleright), R)$ – this lets us switch the "mode" of the machine.

This machine runs in the same number of steps as the original machine except for the case where we cross the start symbol, which takes 2 steps. Thus, the unidirectional machine runs in time at most 2T(n).

2 Problem 1.5

> Define a TM M to be *oblivious* if its head movements do not depend on the input but only on the input length.That is, M is oblivious if for every input $x \in \{0,1\}^*$ and $i \in \mathbb{N}$, the location of each of M's heads at the i^{th} step of execution on input x is only a function of $|x|$ and i. Show that for every time-constructible $T : \mathbb{N} \to \mathbb{N}$, if $L \in \mathbf{DTIME}(T(n))$, then there is an oblivious TM that decides L in time $O(T(n)^2)$. Furthermore, show that there is such a TM that uses only *two tapes*: one input tape and one work/output tape.

Use the same construction as in Claim 1.6 of the text: "The TM M' encodes the k tapes of M (including its input and output tapes) on a single

tape by using locations $1, k+1, 2k+1, ...$ to encode the first tape, locations $2, k+2, 2k+2, ...$ to encode the second tape etc. For every symbol a in M's alphabet, M' will contain both the symbol a and the symbol $\hat{a}$. In the encoding of each tape, exactly one symbol will be of the "hat type", indicating that the corresponding head of M is positioned in that location. M' uses the input and output tape in the same way M does. To simulate one step of M, the machine M' makes two sweeps of its work tape: first it sweeps the tape in the left-to-right direction and records to its state register the k symbols that are hatted. Then M' uses M's transition function to determine the new state, symbols, and head movements and sweeps the tape back in the right-to-left direction to update the encoding accordingly."

We really only need to make a few changes to this to make M' into an oblivious TM M'':

- When M' updates the encoding in its right-to-left sweep, it may need to move the ˆ marker to the right. However, M'' cannot stop and move right when it finds the hat, because it is oblivious. Therefore, the head of M'' must, on its right-to-left sweep, move **LRL** every time M' would move **L**, so its movement pattern will look like **LRLLRLLRL**... That way, the head always has the opportunity to move the ˆ marker either left or right when it is encountered.

- We need to know how far out M'' needs to sweep. One option is to calculate $T(n)$ in advance and place a special marker at spot $kT(n)+1$, so that we always sweep out to $T(n)+1$ and back to the beginning. Alternatively, we simply place a marker at spot $k+1$ in step 1, and at the end of every sweep right, we have M'' move the marker k cells to the right. This will work because the original machine M cannot have gone beyond spot i in step i of its calculation.

- We also need to handle the case where M might halt at different times depending on the input. We know that M is computable in time $T(n)$ for some *time constructible* T. This means that there exists a machine M_T such that on all inputs of length n, M_T halts in exactly $T(n)$ steps. We can simulate M and M_T simultaneously on the same input by adding m tapes (this includes input/output tapes) for simulation of M_T to M''. States of the machine M'' can then be considered as pairs of states of the above form corresponding to M and M_T. M' will continue sweeping back and forth in the oblivious manner described above until M_T halts.

Because we're sweeping out to a maximum distance of $kT(n)$, and we do it $T(n)$ times, the machine M'' runs in time $O(T(n)^2)$.

3 Problem 1.12

A partial function from $\{0,1\}^*$ to $\{0,1\}^*$ is a function that is not necessarily defined on all its inputs. We say that a TM M computes a partial function f if for every x on which f is defined, $M(x) = f(x)$ and for every x on which f is not defined M gets into an infinite loop when executed on input x.If $\mathcal{S}$ is a set of partial functions, we define $f_{\mathcal{S}}$ to be the Boolean function that on input α outputs 1 iff M_α computes a partial function in $\mathcal{S}$. Rice's Theorem says that for every nontrivial $\mathcal{S}$ (a set that is not the empty set nor the set of all partial functions computable by some Turing machine), the function $f_{\mathcal{S}}$ is not computable.

(a) Show that Rice's Theorem yields an alternative proof for Theorem 1.11 by showing that the function HALT is not computable.

(b) Prove Rice's Theorem.

1.12 (a)

Let $\mathcal{S}$ be the set of all partial functions defined on a given input x. If the function HALT is computable, then we can compute $f_{\mathcal{S}}$ by simply checking if the program halts - if it does, then the function must be defined. This contradicts Rice's theorem, so HALT cannot be computable.

1.12 (b)

We assume that Rice's theorem is false. We want to compute HALT(α, x) (does TM M_α halt on input x?).

Let $\mathcal{S}$ be some nontrivial set of partial functions. Without loss of generality, assume that $\mathcal{S}$ does not contain the function NIL that is not defined on any input.

We create a machine M that first runs M_α on the input, then computes a partial function $g \in \mathcal{S}$, and finally outputs the result of the latter computation. Thus, if M_α halts, M is equivalent to a TM that computes g.

We now take the function computed by M and check whether it satisfies $f_{\mathcal{S}}$ (in which case the first part of the computation must have halted) – if so, we output 1 , otherwise, we output 0. So, for any given TM and input, we can check whether HALT is satisfied. This is not possible, so Rice's Theorem must be true.

4 Problem 2.13

Recall that a reduction f from an **NP**-language L to an **NP**-language L' is *parsimonious* if the number of certificates of f is equal to the number

of certificates of $f(x)$.

(a) Prove that the reduction from every **NP**-language L to SAT presented in the proof of Lemma 2.11 can be made parsimonious.

(b) Show a parsimonious reduction from SAT to 3SAT.

2.13 (a)

The textbook's presentation of Cook's Theorem seems to already be parsimonious. To see this, we need to show that there is a bijection between the set of certificates of the initial TM M, and the set of satisfying assignments of the final formula ϕ_x. First, we show an injection: suppose we have two distinct certificates u_1 and u_2 such that $M(x, u_1) = M(x, u_2) = 1$. Any satisfying assignment of ϕ_x must *contain* the bits of the corresponding certificate, and so it's clear that u_1 and u_2 must correspond to distinct satisfying assignments. Second, we show a surjection: consider any satisfying assignment v. v consists of $y, z_1, z_2, ..., z_{T(|x|)}$. ϕ_x checks that the first $|x|$ bits of y are exactly the same as the bits of x, so those are predetermined. Furthermore, given x and the remaining bits of y (which are meant to be interpreted as the certificate u), all of the snapshots z_i are fully determined, as M is a deterministic TM.

The ambiguity is how the formula ϕ_x verifies that M outputs 1. The phrase "M outputs 1" is not well defined. If it means that M is in an accept state and its output tape head is at a cell reading the bit "1", then this property can easily be verified on the final snapshot $z_{T(|x|)}$, and so we conclude that the reduction of Cook's Theorem is already parsimonious. However, if we are not willing to assume this definition, then the reduction in the blook might allow for more than one "accept" condition. In order to account for this, we clear the work tape and place a 1 in the leftmost cell before halting in an accept state, so that there is now exactly one condition under which the machine can accept, ensuring parsimony.

2.13 (b)

We define parsimonious reduction $f : SAT \to 3SAT$ as a repeated application of function g, where g maps each clause $C = x_1 \vee ... \vee x_k$ in SAT instance x to a set of clauses as follows:

- If $k < 4$, $g(C) = C$
- Else, $g(C) = (z \vee x_1 \vee x_2) \wedge (x_1 \to \bar{z}) \wedge (x_2 \to \bar{z}) \wedge g(\bar{z} \vee x_3 \vee ... \vee x_k)$
  $= (z \vee x_1 \vee x_2) \wedge (\bar{z} \vee \bar{x}_1) \wedge (\bar{z} \vee \bar{x}_2) \wedge g(\bar{z} \vee x_3 \vee ... \vee x_k)$

The idea here is that it's just like the reduction in the book, except we force z to be false whenever either x_1 or x_2 is true. This guarantees that whenever x is satisfied by some assignment of variables, there is a unique extension to that assignment that satisfies $f(x)$. If x_1 and x_2 are both false, z is forced to true; otherwise, z is forced to false. Furthermore, if we take any satisfying assignment of $g(C)$ and remove all the assignments to new variables, it's clear by induction that the resulting assignment satisfies C (if z is false, then $x_1 \vee x_2$ is true; if z is true, then $x_3 \vee ... \vee x_k$ is true by induction).

5 Problem 2.14

> Cook used a somewhat different notion of reduction: A language L is *polynomial-time Cook reducible* to a language L' if there is a polynomial time TM M that, given an oracle for deciding L', can decide L. An oracle for L' is a magical extra tape given to M, such that whenever M writes a string on this tape and goes into a special "invocation" state, then the string – in a single step! – gets overwritten by 1 or 0 depending upon whether the string is or is not in L'; see Section 3.4 for a more precise definition.
>
> Show that the notion of cook reducibility is transitive and that 3SAT is Cook-reducible to TAUTOLOGY.

Cook reducibility is transitive

In the following, we use "reducible" to mean *polynomial-time Cook reducible*. Let L be reducible to L', and let L' be reducible to L''. We want to show that L is reducible to L''.

We have a TM M_L that recognizes L in polynomial time $O(n^{c_1})$ using an oracle $O_{L'}$. We also have a TM $M_{L'}$ that recognizes L' in time $O(n^{c_2})$ using an oracle $O_{L''}$. ($O_{L'}$ and $O_{L''}$ recognize L' and L'', respectively.)

We simply compose the two reductions. We replace $O_{L'}$ with $M_{L'}$ – a "false" oracle. Now, M_L runs in polynomial time $O(n^{c_1} n^{c_2}) = O(n^{c_1 \times c_2})$, using the oracle $O_{L''}$. So, L is reducible to L''.

3SAT is Cook-reducible to TAUTOLOGY

A 3SAT formula $f(x)$, is unsatisfiable if $\forall x, f(x) = 0$. So, the negation thereof, $\neg f(x)$, is a tautology.

We take the negation of the input to 3SAT, pass it to a TAUTOLOGY oracle, and then output the negation of the result. Thus, 3SAT is Cook-reducible to TAUTOLOGY.

6 Problem 2.32

> Prove that if every *unary* **NP**-language is in **P** then **EXP=NEXP**. (A language L is unary iff it is a subset of $\{1\}^*$, see Exercise 2.30.)

We use the proof of Theorem 2.22 from the textbook, which shows that if **P=NP**, then **EXP=NEXP**.

Instead of the actual value x, we simply use the unary representation of x, which has length $O(2^{|x|})$. This gives us a unary **NP** language L'_{pad} – the rest of the proof is identical to what is given in the book.