ZHIYUAN COLLEGE, SHANGHAI JIAOTONG UNIVERSITY

CS389: Computational Complexity Homework I

Zihao Ye

October 24, 2016

PROBLEM 1.5

PROOF

It's special form of **PROBLEM 1.6**, since only two tapes are available, we need to combine the contents of work tapes into one tape. Specifically, we use a tuple $(c_1, c_2, \cdots, c_k)$ in one cell to represent the information of tape heads in all tapes. The operations on tape i should be implemented on the i-th dimension of the tuples in UTM.

$O(T(n)\log T(n)) \subseteq O(T(n)^2)$, so we prove **PROBLEM 1.6** first.

PROBLEM 1.6

Section 1.7 has shown us a way to construct a UTM with log-amplification. To construct an oblivious UTM with log-amplification, we need to modify the rule a bit.

Firstly, to make sure the UTM terminates at a exact time given the input length n, we need to attach a counter to the original TM. UTM terminates iff the counter reaches $T(n)$. Since $T(n)$ is a time-constructible function, it's possible to compute $T(n)$ first(don't start the counter first) given 1^n, by adding some additional tapes to the original TM.

Secondly, the UTM stated in section 1.7 is not obvious since it relies on the content of input. To make it oblivious, we do some modifications on the rules:

1. Firstly, initialize all zones $L_i, R_i, i = 0, 1, 2, \cdots$ with 2^i non-buffer elements($|L_i| = 2^{i+1} = |R_i|$).
2. Suppose it's the k-th movement of the original TM, let j be the most significant bit that has just turned from 0 to 1. Suppose $\delta(j) = \sum_{i=0}^{j} \#L_i - 2^i$, Sweeping from L_i to R_i and readjust the number of elements in $L_j, L_{j-1}, \cdots, L_0, R_0, \cdots, R_{j-1}, R^j$ to

$$2^j + \delta(j) \pm 1, 2^{j-1}, \cdots, 2^0, 2^0, \cdots, 2^{j-1}, 2^j - \delta(j) \mp 1$$

In which + or − only depend on whether you want to move toward right or left, and they will be included in $\delta(j)$ after operation.

We use $OP(j)$ to denote this operation.

It's obvious that during the execution, $|L_i| + |R_i| = 2^{i+1}$ is an invariant. Now we just need to prove that $\forall j, 2^j + \delta(j) \in [0, 2^{j+1}]$.

Since elements number in $L_i, L_{i-1}, \cdots, L_0, R_0, \cdots, R_i$ will be resized to $2^i, 2^{i-1}, \cdots, 2^0, 2^0, \cdots, 2^i$ once $OP(x), x > i$ has been operated, then $\delta(i) = 0$.

The round gap between two nearst $OP(x), x > i$ (or from initialization to first $OP(x), x > i$) is 2^i, thus $\delta(i)$ will never exceed 2^i or lower than -2^i. Thus for all i, $OP(i)$ is valid.

Total time complexity is $\sum_{i=0}^{\log(T(n))} (2 \times 2^{i+2} - 1) \times \frac{T(n)}{2^{i+1}} = O(T(n) \log T(n))$.

PROBLEM 1.9

To simulate a RAM TM, we need two tapes: t_1 as memory, t_2 as output. Since the address is unbounded, we can't just save/load symbols at their address(finding the address requires a lot of time).

Whenever we need to write a symbol, save the address $\llcorner i \lrcorner$ and symbol σ at the tail of t_1. If needed, we can add a divider. Thus our tape looks like this:

$$\llcorner i_1 \lrcorner \sigma_1 \dagger \llcorner i_2 \lrcorner \sigma_2 \dagger \llcorner i_3 \lrcorner \sigma_3 \dagger \cdots$$

When we need to load a symbol at some address, look up this address in the tape(from the back forward). If the tape head encounters a zone with the address we need, output its symbol at t_2. Otherwise, output 0 at t_2. Then reset the tape head.

It takes at most $T(n)$ time to save/load, as we know there are no more than $T(n)$ access to RAM, thus we derive the total time complexity is $O(T(n)^2)$.

PROBLEM 1.13

(A)

$$\text{BIT}(n, i) = \text{DIVIDES}(p_i, n)$$

(B)

$$\text{COMPARE}(n, m, i, j) = \forall x, (x < i) \vee (x > j) \vee (\text{BIT}(n, x) = \text{BIT}(m, x))$$

(C)

For convenience, assume M is a single-tape TM(similarly hereinafter). Firstly, we encode every symbol in M's alphabetΣ as a binary string(from $(0\cdots0)_2$ to $(|\Sigma|)_2$). Then we need to encode every state of M as a binary string(from $(0\cdots0)_2$ to $(|S|)_2$). Thus we derive the binary representation of a configuration of M in the following way:

$$\mathrm{BR}(C) = \ddagger c_1 \dagger c_2, \cdots \dagger c_l \ddagger i \ddagger s \ddagger$$

In which l represents the "useful" length of the single tape(which means c_l is not empty, and $\forall j > l, \mathrm{empty}(c_j)$), $c_1, c_2, \cdots c_l$ represent the content of the cells. i is the current position of tape head while s is the current state.

By encoding † as 01, ‡ as 10, 0 as 00, 1 as 11. We derive an unambiguous binary representation of the configuration.

(D)

Let C be the initial configuration of M given input x, and let Len(s) be the length of a binary string s, define m as:

$$m = \prod_{i=1}^{\mathrm{Len(BR}(C))} p_i^{\mathrm{BR}(C)_i}$$

Then we can derive the INIT function:

$$\mathrm{INIT}_{M,x}(n) = \mathrm{COMPARE}(n, m, 1, \mathrm{Len}(\mathrm{BR}(C)))$$

(E)

We can construct a parser to parse any component of a binary string s, suppose $\mathrm{Comp}(s, i)$ allow us to fetch the i-th component divided by ‡ from the binary string representation s of the configuration and transform them to the normal form(11 to 1, 00 to 0, according to the last process in **(c)**. But when $i = 1$, we don't need to do this since we still have to deal with another divider †).

To finish HALT function, we need to be able to decode the binary representation s from n, which could defined as:

$$s = \mathrm{Bin}(n) = \mathrm{BIT}(n,1)\mathrm{BIT}(n,2)\cdots\mathrm{BIT(n,\ k)}$$

In which k is the least number to satisfy that $\mathrm{BIT}(n,1)\cdots\mathrm{BIT(n,\ k)}$ contains 4 dividers ‡.

Suppose $\mathrm{Term}_M(i)$ returns whether the i-th state of M is terminal state.

$$\mathrm{HALT}_M(n) = \mathrm{Term}_M\left(\llcorner \mathrm{Comp}(\mathrm{Bin}(n), 3) \lrcorner_2\right)$$

(F)

Let $State(n)$ be the current state of the configuration correspond to the encoding n:

$$\text{State}(n) = S\left(\llcorner \text{Comp}(\text{Bin}(n), 3) \lrcorner_2\right)$$

Let $Cont(n)$ be the content of the current cell of the configuration correspond to the encoding n:

$$\text{Cont}(n) = \text{Comp}(\text{Bin}(n), 1)\left[\llcorner \text{Comp}(\text{Bin}(n), 2) \lrcorner_2\right]$$

Then it is possible to construct the following three functions:

Succ-State(n, m)**:**

State(m) is the successor of State(n) in M according to State(n) and Cont(n)

Succ-Pos(n, m)**:**

Comp$(\text{Bin}(m), 2)$ is the successor of Comp$(\text{Bin}(n), 2)$ in M according to State(n) and Cont(n)

Succ-Cells(n, m)**:**

Comp$(\text{Bin}(m), 1)$ is the successor of Comp$(\text{Bin}(m), 1)$ in M according to State(n) and Cont(n)

The third one requires some careful examinations, and the "useful" length of them may not be the same. Then we can derive NEXT as:

$$\text{NEXT}(n, m) = \text{Succ-State}(n, m) \wedge \text{Succ-Pos}(n, m) \wedge \text{Succ-Cells}(n, m)$$

(G)

Define Enc(s) as:

$$\text{Enc}(s) = \prod_{i=1}^{l} p_i^{s_i}$$

Then it's convenient to derive that:

$$\text{VALID}_M(m, t) = \text{NEXT}(\text{Enc}(\text{BR}(x_1)), \text{Enc}(\text{BR}(x_2))) \wedge \cdots \wedge \text{NEXT}(\text{Enc}(\text{BR}(x_{t-1})), \text{Enc}(\text{BR}(x_t)))$$

(H)

$$\text{HALT}_{M,x}(t) = \exists m, \text{VALID}_M(m, t) \wedge \text{INIT}_{M,x}(x_1)$$

While in which x_1 is the first element of m(according to the convention in **(g)**).

(I)

From the construction below, and with the help of computable TRUE-EXP, one can construct a Turing Machine H to determine whether a TM M terminates given input x:

$$H(M,x) = \text{TRUE-EXP}\left(\exists m, \text{HALT}_{M,x}(t)\right)$$

Design another TM $T(P)$ as:

1. while(1); if $H(P,P) = 1$.
2. return 0; if $H(P,P) = 0$.

Run $H(T,T)$, $H(T,T) = 1 \Longrightarrow H(T,T) = 0, H(T,T) = 0 \Longrightarrow H(T,T) = 1$, which leads to contradiction.

PROBLEM 1.15

(A)

To show that choosing a different base of representation will make no difference to the class **P**, we need to do Karp-reduction from L_S^b to L_S^2, and from L_S^2 to L_S^b.

$L_S^b \Longrightarrow L_S^2$: $\llcorner n \lrcorner_b$ can be transformed to $\llcorner n \lrcorner_2$ by Horner's method, which could be described as:

$$\llcorner n \lrcorner_2 = ((n_k \times b + n_{k-1}) \times b + \cdots) \times b + n_0$$

In which $k = O(\log_b(n))$, this algorithms takes at most k multiplication and at most k addition, both requires at most $O(\log_2(b) \times \log_2(n))$ time, the total time complexity is $O(\log_2(n)^2)$ which is a polynomial of input length $\log_2(n)$.

$L_S^2 \Longrightarrow L_S^b$: $\llcorner n \lrcorner_2$ can be transformed to $\llcorner n \lrcorner_b$ by iteratively divide $\llcorner n \lrcorner_2$ by $\llcorner b \lrcorner_2$, and records the remainder one by one. The total time complexity is $\log_b(n) \times \log_2(n)\ times \log_2(b) = O(\log_2(n)^2)$, which is also a polynomial of input length $\log_2(n)$.

Thus we derive $L_S^b \in \mathbf{P}$ iff $L_S^2 \in \mathbf{P}$.

(B)

We need to enumerate every number in (l,k) to see whether it divides n, however, each check takes $O(n)$ time. If $x \in (l,k)$ divides n, we need to check whether it is a prime, which requires an enumeration over $(1,n)$ and in each round we need $O(n)$ to calculate the remainder.

Overall, the time complexity is $O((k-l) \times n \times n \times n) = O((k-l)n^3)$, which is a polynomial of input size $n+k+l$.

The main difference between unary representation and other representations is that in other presentations, we are required to use $0, 1, p-1$ given the base p while in unary presentation, we are required to use 1 other than 0, since 0 can make nothing.

ACKNOWLEDGEMENT

Thanks Tianyao Chen for his creative idea on 1.6.