Honors Theory, Spring 2002, Yap
Homework 4

Exercises on machines models (Turing, Pointer). Note that you do not need to hand in problems that has 0 points.

This is due on Monday March 18. But for any problem that you solve and submit BEFORE the midterm, I will give you the solution to that problem.

1. (0 Points) In the lecture notes and in class, we show that for any $c \in \mathrm{N}$ and any k-tape M, we can construct a $(k+1)$-tape M' with the following property: (i) each tape symbol of M' encodes c consecutive tape symbols of M, (ii) M' begins by converting the input word into this compressed from on its extra tape (so that it can now read c input symbols at a time), and (iii) M' simulates c steps of N using only 6 steps. (Call the 6 steps a "phase".)

   Can you reduce the number "6" in a phase?
   (i) Improve the number 6 in case $k = 1$.
   (ii) When $k > 1$, can you improve 6 to 5?

   Before we go into the solution, we should say what the ground rules for this game is: let c be arbitrary. We want to simulate c steps of a multitape TM M using a constant (hopefully less than 6) steps of a new multitape TM N. We assume that each tape symbol of N can represent c symbols of M's tape symbols. That is all.

   SOLUTION (i): When $k = 1$, we can improve 6 to 4 as follows: If the index of the current supercell is i, we know that in the next c steps of M, we only need to modify the supercells $i-1$ or $i+1$, but not both. Because we have only one worktape ($k = 1$), we can predict whether it should be $i-1$ or $i+1$. Hence, in one step, we can move into $i-1$ or $i+1$, as the case maybe. Without loss of generality, say it is $i-1$. Next, we know exactly what the new contents of supercells i and $i-1$ will be after c steps of M. Hence, in 2 more steps, we can update supercels $i-1$ and i. Thus only 3 steps are needed. Moreover, we can ensure that at the end of the 3rd step, we are scanning the correct supercell (either $i-1$ or i).

   SOLUTION (ii): We describe a 5-step simulation phase, but it is more complicated than the 6-step simulation. Let us focus our attention on the head motion of M' in one of the worktapes, but you must imagine that some corresponding head motion is happening on the other worktapes. The action of M' depends on all these head motions. Also, the input tape is just a special case of the worktape, so we do not need to treat it specially.

   Suppose we are currently scanning supercell i on a worktape, at the beginning of a sumulation phase. In the 6-step simulation, we assume that the current head of M is somewhere inside supercell i. Two "delayed actions" can now happen: (I) We allow the possibility that the current head of M is either at supercell j where $j \in \{i-1, i, i+1\}$. (II) We allow the possibility that the current supercell of M does not really contain the latest supersymbol! Instead, this supersymbol is remembered in the state of M, and it will be written out in the first step of the next phase.

   CASE (A): Suppose the head of M is inside supercell i. Then in three steps, we can scan supercells $i-1$, i and finally $i+1$. Now we know (based on corresponding information from the other heads), which two supercells need to be updated. In two more steps, we can update them. However, our superhead need not end up on the supercell j which contains the head of M. However, we have $|j-i| \leq i$, and M knows this.

   CASE (B): Suppose the head of M is inside supercell $i+1$. Then in three steps, we can scan $i+1, i+2$ and back to $i+1$. How we know (based on corresponding information from the other heads), the two supercells need to be updated. They are either $i, i+1$ or $i+1, i+2$. Then in two more steps, we can update them (possibly with delays of types (I) or (II)).

2. (5+15 Points) Construct a pointer machine PM that adds two dyadic numbers.
   (i) Describe the dyadic algorithm for addition. Is it simpler or more complex than binary addition?
   (ii) We need some I/O conventions for pointer machines (PM's). Let Δ be the *alphabet* for the PM, and G be the computation graph of the PM. If $w \in \Delta^*$, let $[w]_G$ denote the node in G represented by w. For instance, the empty string ε represents the center $[\varepsilon]$. Assume a special symbol $\beta \in \Delta$ that denotes the *bit value* of any node $w \in \Delta^*$ in the following sense: if $[w.\beta] = [\varepsilon]$, then we say that the node $[w]$ stores the 1 bit, else it stores tbe 0 bit. How do we represent a list of nodes? Assume a special "next symbol" $\nu \in \Delta$ such that $[w.\nu]$ represents the next node after $[w]$, provided $[w, \nu] \neq [\varepsilon]$; otherwise we say there is no next node. Define the list headed by w to be the sequence

$$L(w) := ([w], [w\nu], [w\nu^2], \ldots, [w\nu^{n-1}]),$$

terminated by the first n such that $[w\nu^n] = [\varepsilon]$. The list $L(w)$ may be infinite (in the case of a loop). The binary string represented by w is simply the sequence of bit string stored in $L(w)$. Input are indicated by two linear lists $L(\alpha)$ and $L(\alpha^2)$, where $\alpha \in \Delta$ is a special "input symbol". When the computation ends, we want the output string to stored in the list $L(\alpha^3)$.

SOLUTION (i): It suffices to only consider single digit addition, with or without carries. The remaining details follow the usual binary addition algorithm. Since addition is commutative, we only need to consider 7 output possibilities:

$$0+0=1, \quad 0+1=00, \quad 1+1=01, \quad 0+0+0=00, \quad 0+0+1=01, \quad 0+1+1=10, \quad 1+1+1=11$$

This seems more complex (say, in terms of logical circuitry if we have to construct hardware) than binary addition. Dyadic addition has two types of carry, but binary addition has only one type of carry. We have to consider four types of outputs: $0+0=0, 0+1=1, 1+1=10$ when there are no carries. If there is a carry, the only new kind of output is obtained as $1+1+1=11$.

SOLUTION (ii): Now we have to implement the above algorithm on a PM. With the stated I/O convention, we introduce more pointer symbols $A, C, 0, 1 \in \Delta$. The nodes $[A]$ and $[A^2]$ are the current nodes in the first and second arguments, respectively. Also, $[A^3]$ is the current output node. The value of $[C]$ is either $[\varepsilon]$ or [0] or [1], corresponding to the various carry possibilities. To initialize, we have

$$A = \alpha; A^2 = \alpha^2; A^3 = new; C = \varepsilon;$$

To add the current bits, we check for each of the 7 cases of one bit addition: for instance, the case $0+0=1$ amounts to the following code:

If $(A \neq \varepsilon \wedge A^2 \neq \varepsilon \wedge C = \varepsilon)$ then
$A^3.\beta = \varepsilon; C = \varepsilon;$
$A^3.\nu = new; A = A.\nu; A^2 = A^2.\nu$

The case $0+1+1=10$ has three possibilities, depending on whether "0" is in the first argument, second argument or in the carry. But for illustration, we only show the case where the first argument is a 0.

If $(A \neq \varepsilon \wedge A^2 = \varepsilon \wedge C = 1)$ then
$A^3.\beta = A^3; C = 1;$
$A^3.\nu = new; A = A.\nu; A^2 = A^2.\nu$

We leave the remaining details to the reader.

3. (30 Points) Verify the Polynomial Simulation Thesis (L), for Turing machines and Pointer machines. We want to verify two main cases, depending on the mode μ and resource ρ:
(i) (μ, ρ) = (Fundamental Mode, Time). More precisely, $DTIME(n^{O(1)}) = DTIME_{PM}(n^{O(1)})$. The latter class is the class of languages accepted by PM's in deterministic polynomial time.
(ii) (μ, ρ) = (Nondeterministic Mode, Space). That is $NSPACE(n^{O(1)}) = NSPACE_{PM}(n^{O(1)})$.

Please only sketch the simulations, at a fairly high level (but you should give details when necessary). Try not to write more than 4 pages for this problem. [Remember the words of the wise speaker, "if I have more time, I would have given a shorter speech".]

4. (10x3 Points) There are three possible responses to the following statements: True, False or Unknown. "Unknown" reflects the uncertainty that only assume the results in chapter 2. If your answer is True or False, you must give brief reasons in order to obtain full credit. For Unknown, discuss relevant results.
(a) T/F/P: $NSPACE(n) \subseteq DTIME(O(1)^n)$.
(b) T/F/P: $NTIME(n) \subseteq DTIME(2^n)$.
(c) T/F/P: $NREVERSAL(n) \subseteq DREVERSAL(O(1)^n)$.

SOLUTION:
(a) True. $NSPACE(n) \subseteq DTIME(n^3 O(1)^n) = DTIME(O(1)^n)$.
(b) Unknown. We know that $NTIME(n) \subseteq DTIME(O(1)^n)$. But there is no single constant c such that we can prove $NTIME(n) \subseteq DTIME(c^n)$.
(c) We will accept two kinds of answers from you: We know that $NREVERSAL(2) = RE$. So the question amounts to asking if $RE = DREVERSAL(O(1)^n)$. Based on the results of Chapter 2, the answer is "Unknown". But using hierarchy theorem in chapter 6, we actually know the answer is "False".