

CS273: Theory of Computation, Summer 2008

Homework 6

Due 3:00PM Monday, July 21, 2008 at SC 3303

Revision due 3:00PM Friday, July 25 2008 at SC 3303

1. [30 points; 10 per part] For each of the following languages, describe a Turing Machine that *decides* the language. Do not give a formal specification (i.e., the complete 7-tuple), but describe the high-level algorithm at a level of detail that makes it clear how each step is accomplished by the limited tape-head movements of a Turing Machine.

(a) $\{www \mid w \in \{a, b\}^*\}$

Solution: There are several ways to solve this problem. Perhaps the shortest to describe is one that uses nondeterminism. Scan left to right, and nondeterministically choose 2 positions to insert a # symbol, shifting everything to the right. Then we can repeatedly do the following:

- i. Rewind the tape head to the leftmost position
- ii. Find, mark, and remember the first unmarked a or b before the # sign. If there is none to find, scan right and make sure everything is either marked or a # sign, and accept, else reject.
- iii. Scan past the first # sign; find and mark the first unmarked a or b before the next # sign. If it is different than the previously remembered character, or if there is no such unmarked character, reject.
- iv. Scan past the second # sign; find and mark the first unmarked a or b before the first blank. If it is different than the previously remembered character, or if there is no such unmarked character, reject. Otherwise, repeat from step (ii).

■

(b) $\{a^n b^{n^2} \mid n \geq 0\}$

Solution: The idea behind this is to use two “nested loops” from 1 to n to count up to n^2 . Let’s suppose we can mark tape cells with a red mark, blue mark, or both (or neither). This would be implemented by adding extra characters to the tape alphabet. The blue mark will keep track of the outer nested loop, and the red mark will keep track of the inner loop.

- i. Rewind the tape head to the leftmost position. Find and mark blue the first non-blue-marked a . If there is none to find, scan right and accept if every b is marked, else reject.
- ii. Rewind the tape head to the leftmost position. Find and mark the first non-red-marked a . If there is none to find, erase all the red marks and repeat from step (i). Otherwise, mark the first unmarked b (the color of the marks on the b s is not important) and repeat this step. If there is no unmarked b , reject.

■

(c) $\{a^n b^{2^n} \mid n \geq 0\}$

Solution: The idea here is to mark off half of the bs for each a .

- i. Rewind the tape head to the leftmost position. Find and mark the first unmarked a . If there is none to find, scan right and accept if there is exactly one unmarked b ; else reject.
- ii. Scan to the right and mark every other unmarked b . If there were an odd number of unmarked bs (or no unmarked bs at all), reject. Repeat from step (i).

■

You may use any “enhanced” Turing Machine variants that we have shown in class to be equivalent to plain Turing Machines.

2. [15 points] [From Sipser] Suppose we define a variant of a Turing Machine in which δ has the form $Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, \text{RESET}\}$. If the transition specifies “R”, the tape head advances one cell to the right, as normal. However, if the transition specifies “RESET”, the tape head is sent all the way to the leftmost tape cell.

Show that this Turing Machine variant is exactly equivalent to the standard definition.

Solution: We should show that these R-RESET-TMs can simulate standard TMs. R-transitions are no problem. To simulate the standard TM taking a L-transition, we do the following:

- (a) Mark the current cell with a “red” mark, to indicate the TM’s original head position, and RESET. Our goal is to somehow move this “red” mark one cell to the left.
- (b) If the current cell (after RESET) is marked, we must be on the first cell of the tape, so we are done (this is exactly what a L-transition would have taken us).
- (c) Otherwise, mark the current cell with a “blue” mark. Our goal is to walk this mark to the right until it is just before the red mark.
- (d) Move to the right. If we land on the red-marked cell, then our blue mark is in the correct spot, so erase the red mark, RESET, scan R to find the blue mark again and then we’re in the right spot to simulate the next step of the TM (we’ll erase the blue mark as we leave).
- (e) Otherwise, we need to move the blue mark, so RESET, scan R to find the blue mark again. Erase it, move R, mark that cell blue, and repeat the previous step.

We should also show the other direction: that normal TMs can simulate these R-RESET-TMs. This is pretty straightforward. At the beginning, put down a special “left endmarker” character and shift the rest of the cells to the right. Then any time we want to simulate a RESET command, we scan to the left until we see the left endmarker character and move right one cell.

■

3. [15 points] [From Sipser] Suppose we define a variant of a Turing Machine in which δ has the form $Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, S\}$. If the transition specifies “R”, the tape head advances one cell to the right, as normal. If the transition specifies “S”, the tape head stays in the same cell.

What is the class of languages accepted by these kinds of machines? Prove that your answer is correct.

Solution: These TMs accept only regular languages. First of all, a R-S-TM can easily simulate a DFA. This is the easy direction, so we'll focus on the other direction: namely, that a DFA (or NFA) can simulate a R-S-TM.

Suppose we have a R-S-TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$. Since the TM can still write to its tape, and the symbols that it writes can influence the future behavior, it is not immediate that an NFA can simulate it. Our idea is to build an NFA with special states q'_{start}, q'_{acc} , and other states $Q \times \Gamma$. q'_{acc} will be the only accept state, and it will have self-loops on all inputs.

We will design the NFA so that after reading w , it is in state (q, a) to represent the fact that the TM is in state q , with tape head at the $|w|$ th cell of the tape, looking at character a (note: this may not be the last character of w , since the TM could have overwritten it with an a). We just need to describe how this invariant (of the meaning of the NFA states) is maintained by the NFA's transition function.

The TM will start in its q_0 state looking at the first character of the input word (whatever that character is), so we add a transition in the NFA from q'_{start} to (q_0, a) on NFA input a , for each $a \in \Sigma$.

Now, if the TM is in state q , looking at tape symbol a , and there is a TM transition $\delta(q, a) = (r, b, R)$, then after this transition, the TM would be in state r looking at the symbol corresponding to the next symbol of the input. So for this kind of TM transition, we add a transition in the NFA from (q, a) to (r, c) on NFA input c , for all $c \in \Sigma$. Note that b , the character written to the tape by the TM, is irrelevant since the TM moved R, never to return.

Otherwise, if the TM is in state q , looking at tape symbol a , and there is a TM transition $\delta(q, a) = (r, b, S)$, then after this transition, the TM would be in state r looking at symbol b , but in the same tape cell. In terms of our NFA, this means that the NFA should not consume its next input character. So for this kind of TM transition, we add an *epsilon-transition* in the NFA from (q, a) to (r, b) .

Finally, if the TM ever enters its accept state, the entire input is considered accepted. So our NFA should have an epsilon transition from (q_{acc}, a) to q'_{acc} for all $a \in \Sigma$. Then q'_{acc} will consume the rest of the NFA input characters and properly accept the NFA's input. ■

4. [20 points] [From Sipser] Show that a language is decidable if and only if there is an enumerator that enumerates the language in lexicographic order.

As a reminder, lexicographic order is where we enumerate all strings of length 0, then all of length 1, then all of length 2, and so on. Within each length, strings are ordered as in a dictionary.

Solution: (\Rightarrow) Suppose A is decidable, with decider M . Then we will construct the following lexicographic enumerator, which uses M as a subroutine:

Enumerator E :

- For $x = s_1, s_2, \dots$ (where s_1, \dots are the strings in Σ^* in lexicographic order):
 - Simulate M on input x . If M accepts, then print x

Clearly E only prints a string if M accepted it, and clearly it only prints strings in lexicographic order. Also, since M eventually halts on every input, the second line of the algorithm for E always halts (i.e., the main loop always keeps advancing). So if M accepts some string x , then E will eventually simulate M on x and then print x .

(\Leftarrow) Suppose we have a lexicographic enumerator E for the language A . First of all, if A is finite, then it is certainly decidable (in fact, regular). Otherwise, consider the case where A is infinite. Then we will construct the following decider for A , which uses E as a subroutine:

TM M : On input x :

- Start simulating E
- Each time E outputs a string y :
 - If $x = y$, then accept.
 - If $|y| > |x|$, then reject.

Clearly, if E ever outputs x , then it outputs x before it outputs any string of longer length, so M will accept. For the other case, suppose E never outputs x . Since A is infinite, E will eventually output a string longer than x and M will correctly reject. ■

5. **[Honors; 15 points]** A *one-counter automata (1CA)* is an NFA that is equipped with a counter which holds an integer value. The counter is initialized to zero. At every transition, the machine can check whether its counter is zero, and also decide to add or subtract 1 from the counter. The machine accepts if the counter is zero after reading the entire input.

- Give a formal definition for 1CAs, including the definition of acceptance.
- Describe a 1CA for the language $\{w \in \{a, b\}^* \mid 2\#a(w) > \#b(w)\}$.
- Show that every language recognizable by a 1CA is context-free.

Solution: A 1CA is a 4-tuple $M = (Q, \Sigma, \delta, q_0)$, where

- Q is a finite set of states,
- Σ is an alphabet,
- $\delta : Q \times (\Sigma \cup \varepsilon) \times \{=, \neq\} \rightarrow \mathcal{P}(Q \times \{-1, 0, 1\})$, and
- $q_0 \in Q$ is the start state.

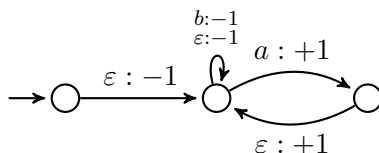
The intent is that δ takes arguments corresponding to the current state, current input character (or ε), and the status of the counter (whether it is zero or nonzero), then outputs a set of possible actions (next state, and quantity to add to the counter).

M accepts a string $x \in \Sigma^*$ if there is a way to write $x = a_1 \dots a_n$ (where each $a_i \in \Sigma \cup \{\varepsilon\}$), a sequence of states r_0, \dots, r_n , and sequence of integers c_0, \dots, c_n , such that:

- $r_0 = q_0, c_0 = 0$, and $c_n = 0$,
- If $c_i = 0$, then $(r_{i+1}, c_{i+1} - c_i) \in \delta(r_i, a_i, =)$.
- If $c_i \neq 0$, then $(r_{i+1}, c_{i+1} - c_i) \in \delta(r_i, a_i, \neq)$.

In other words, there should be a sequence of states and values for the counter that follow according to the transition function. The counter should start at 0 and end at 0.

Here’s a 1CA for the above language. This 1CA does not check whether the counter is 0. So a transition label “ $a : +1$ ” between states q and r means that in state q , when reading a , you can transition to state r and increment the counter (regardless of whether the counter was 0 or nonzero).



The machine first subtracts 1 from the counter. Then it adds 2 for every a and subtracts 1 for every b . It also can nondeterministically subtract 1 as many times as it likes, at any time. Thus the counter will always contain $-1 + 2\#a(w) - \#b(w) - N$, where $N \geq 0$ is the number of “extra times” it has subtracted. If this number is 0, then the machine accepts. This number can be zero only if $2\#a(w) - \#b(w) > 0$, which is what we want.

We can convert a 1CA to a GPDA by cleverly using the stack to simulate the counter. A stack containing just $\#$ will represent the counter being 0. A stack containing $\#$ at the bottom and k “ \oplus ” symbols will represent the number k . A stack containing $\#$ at the bottom and k “ \ominus ” symbols will represent the number $-k$. We start off by adding a new start state that pushes $\#$ to the stack to simulate initializing the counter to zero.

We can translate 1CA transitions to sets of GPDA transitions as follows:

For a 1CA transition that says:	Add these GPDA transitions:
On input a , counter nonzero, add 1	$a, \oplus \rightarrow \oplus \oplus$ $a, \ominus \rightarrow \varepsilon$
On input a , counter zero, add 1	$a, \# \rightarrow \oplus \#$
On input a , counter nonzero, subtract 1	$a, \ominus \rightarrow \ominus \ominus$ $a, \oplus \rightarrow \varepsilon$
On input a , counter zero, subtract 1	$a, \# \rightarrow \ominus \#$

Finally, at any time we can allow the GPDA to ε -transition to a new accept state, whenever the top of the stack has a $\#$ symbol, since the 1CA can accept any time the counter is zero. ■