CS311 / MATH352 - AUTOMATA AND COMPLEXITY THEORY

Homework # 7

Max. Points: 100

Due Date: Wednesday, February 15, 2006 3:25PM

Problem No. 1 (5*3=15 points)
Decide whether the following statements are True or False. You must justify your answer to earn any credit.

i) The set of all possible Java programs forms a regular language.

False. Because in Java programs, we have to keep count of opening and closing parentheses which can't be done by NFA or DFA. So, such programs won't make the language regular.
ii) Every ambiguous CFG can be converted into an equivalent unambiguous CFG.

**False.** Some ambiguous CFG's can't be converted into equivalent unambiguous CFG. See example in section 5.4.4 of your textbook (figure 5.22).

iii) Every non-deterministic push-down automaton has an equivalent deterministic push-down automaton.

**False.** Non-determinism is more powerful than determinism in case of pushdown automata. E.g. \( L = \{a^n b^m c^p : m,n,p \geq 0, \text{ and } m! = n \text{ or } m! = p\} \) can be accepted by an NPDA, but no DPDA exists for it. Reason for non-existence of DPDA is that it involves non-determinism regarding whether to satisfy \( m! = n \) or \( m! = p \). We don't know in advance, which of these statements will qualify our valid string.

iv) Every non-deterministic Turing machine has an equivalent deterministic Turing machine.

**True.** Idea is that use a multi-tape deterministic TM to simulate non-deterministic TM. First tape records the input. Other tapes are used in some fashion to store all possible computations of non-deterministic TM. As a result, it requires exponentially many steps in \( n \) to simulate a computation of \( n \) steps by the non-deterministic TM.
v) Every Turing machine can be simulated by a PDA with two stacks.

<table>
<thead>
<tr>
<th>True. Use one stack for some manipulation. When you have to use a symbol below top of first stack, pop desired contents from first stack and push them into second stack. You can switch pop and push operations between two stacks and can do any manipulation.</th>
</tr>
</thead>
</table>

**Problem No. 2 (5*3=15 points)**
Examine the formal definition and other relevant literature of Turing machine to answer the following questions. You must justify your answer to earn any credit. Just Yes / No will give you no credit.

  a) Can a Turing machine ever write blank symbol on its tape?

<table>
<thead>
<tr>
<th>Yes. There is no restriction on writing blank in tape for ever.</th>
</tr>
</thead>
</table>

  b) Can the tape alphabet \(\Gamma\) be the same as the input alphabet \(\Sigma\)?

<table>
<thead>
<tr>
<th>No, blank tape symbol is always in (\Gamma), but it can’t be part of (\Sigma). Initially, input tape contains all blanks. So, blank is a necessary part of (\Gamma).</th>
</tr>
</thead>
</table>
c) Can a Turing machine’s head ever be in the same location in two successive steps?

Yes, stay-option TM’s do so, and they are equivalent to standard TM’s. So, standard TM’s can also do so.

d) Can a Turing machine contain just a single state?

Yes. TM’s can have just a single state. E.g. TM for \( L = \{a^*\} \) may be shown using a single state which is starting as well as final state.

e) Can a single Turing machine simulate functionality of two different TM’s at the same time?

Yes. Copy input contents to another tape (for a multi-tape TM). Simulate functionality (all manipulations) of TM1 on first tape. Then simulate functionality (all manipulations) of TM2 on second tape.
Problem No. 3 (20 points)
Say that a write-once Turing machine is a single tape TM that can alter each tape square at most once (including the input portion of tape). Show that this variant of Turing machine model is equivalent to the ordinary TM model.
(Your proof may be informal, but it must be sensible)

Part I:
Ordinary TM simulates a write-once TM.
We will use a multi-tape TM as ordinary TM (we know that multi-tape TM is equivalent to standard TM). In a multi-tape TM, we use a lot of tapes. As we can write at most one, so after one alteration, if we need more alteration on an already altered cell, we will copy the current tape contents to the next tape. (We will modify our transition accordingly). Keep on this copying process to the next tape, until all required alterations have been done.

Part II:
A write-once TM simulates an ordinary TM.
It’s trivial. A write-once TM is an ordinary TM, which alters the tape contents at most once.

Hence, they are equivalent.
**Problem No. 4** (20 points)
Show that (single-tape) Turing machines that cannot write on the portion of the tape containing the input string recognize only regular languages.
*(Your proof may be informal, but it must be sensible)*

TM's have memory and counting capability, while DFA's don't have. To memorize, TM's use special tape symbols and write on tape containing input. So, already read portion of tape is recognized. Machine can reach back to a marked position. So, we know where the head was last time. When we restrict the TM's to change any thing on tape containing input, no marker can be put on tape. So, no counting or memorization exists.

What such tapes can do in this situation?
They start reading input but can't remember what they had read before. This is the exact property which DFA's hold. Even PDA's can remember in a limited fashion, but such TM's can't remember any thing. So, they also don't accept CFL's. Hence, they are equivalent to DFA's, and can recognize only regular languages.
**Problem No. 5 (30 points)**
a) Give state diagram, formal description and meaning of each state (single line meaning per state) of Turing machine which decides following language L:

\[ L = \{w#w \mid w \in \{0, 1\}^*\}, \quad \Sigma = \{0, 1 \#, \ \square\} \]

(8+2+5 points)

Here is the TM \((Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})\):

![Turing Machine Diagram]

(8+2+5 points)
b) Prove that all words in L are accepted by the Turing machine suggested by you.  
(5 points)

We will prove it by induction:

Base Case: w#w is accepted by this TM, when |w|=0, i.e. # is accepted.
Also, when |w|=1, the concerned branch of TM can verify w#w in a single step.

Inductive Step: We assume that w#w is accepted for any w, where |w|=k.
We will prove that v#v is also accepted for any v, where |v|=k+1.
It means that, v may be of at most two forms. One is w concatenated with a single alphabet, and other is possibility is that any single alphabet followed by w.
First form is trivial. As we know that for string of length 1, our base case supports us. So, leading single alphabet can easily be matched and according to hypothesis w is also matched successfully. So, aw can be matched with aw after #.
In second part, according to our hypothesis, w is successfully match.
After it, a single alphabet can be matched according to our base case.
So, v#v will accepted for any v, where |v|=k+1.

c) Prove that all words not in L are rejected by the Turing machine suggested by you.  
(5 points)

In this part, we will also use induction.

Base Step: a#, #a and a#b are not accepted by this TM, where a and b are single alphabets. And a is not equal to b. In just one step, we can see this case using our TM.

Inductive Step: As we have three base cases, so we have three possibilities here. If w# is rejected by TM, then wa# will also be rejected. If #w is rejected, then #aw will also be rejected. (If part is inductive hypothesis and in then part we are showing for k+1 length).
If w1#w2 is rejected, then a aw1#aw2, w1a#w2a, aw1#bw2 and w1a#w2b will also be rejected. How???
w1#w2 is our inductive hypothesis. All other cases can easily be proved by using inductive hypothesis and base case. Considering all possible combinations of hypothesis and base cases, we get the desired proof.
d) Give the sequence of configurations that the above Turing machine enters when started on the following input strings: (2+3 points)

i) 01# >

q001# > q11# > 1q1# > 1#q2
Halted in non-final state.

ii) 101#101

q0101#101 > q601#101 > 0q61#101 > 01q6#101 > 01#q7101
> 01#Xq801 > 01#q8X01 > 01q8#X01 > 0q81#X01 > q801#X01
> q901#X01 > q11#X01 > 1q1#X01 > 1#q2X01 > 1#Xq201
> 1#XXq31 > 1#Xq3X1 > 1#q3XX1 > 1q3#XX1 > q31#XX1 > q41#XX1
> q6#XX1 > #q7XX1 > #Xq7X1 > #XXq71 > #XXq8X > #Xq8XX
> #q8XXX > q8#XXX > q9#XXX > #q10XXX > #Xq10XX
> #XXq10X > #XXXq10 > #XXXq11
Halted in a final state.