Problem 1. Computation trees for Nondeterministic TMs.

The following is a diagram of a part of some NTM computation tree on input abb, starting with the initial configuration; for the transitions (labels on edges), we only show the new character, head move, and new state (in that order).

Note: the configurations are being represented using the book’s notation

a) Under each node, write its address (defined as in the proof of theorem 3.16); the address of the root is $\varepsilon$.

b) List the addresses for all the nodes from the diagram in breadth-first order.

c) We are traversing the computation tree, and we have reached node 312; what are the transitions that need to be made to reach this node from the root of the tree? Show them as a list of 5-tuples: (current state, input char, new char, head move, new state).

d) Inside each node, write the configuration that corresponds to it, in textbook's notation (as we've done for the root).
Problem 2. Combining TM programs for decidable languages

Imagine you have two TM programs for the TM simulator, M1 and M2. The first decides some language L1, and the second L2. You have access to these programs but you know NOTHING else about what these machines do.

a) Describe how to use M1 to create a new program that will decide the complement of L1. What exactly will you modify or add?

b) Describe how to use M1 and M2 to create a new program that will decide L1 ∩ L2. What exactly will you modify or add?

c) Suppose that M1 and M2 recognize L1 and L2 rather than decide them. Can you use the same approach as above to create new programs that recognize the complement of L1 and L1 ∩ L2? Justify your answer carefully, using definitions of deciding vs. recognizing.

Note: Your answers to (a) and (b) should be general-purpose (no matter what M1 and M2 are like), but specific enough that I would be able to follow your instructions and create such a program.

Problem 3. 2-stack PDAs

We can define a class of 2-stack PDAs (or "2PDA" for short), which have 2 stacks instead of one. On any transition, 2PDAs can push and/or pop either or both stacks. (This is analogous to how multitape TMs can read/write multiple tapes on each transaction) Just as for regular PDAs, both stacks are initially empty.

a) Give a formal definition of 2PDA (including of the transitions). See hint below.

b) What would its configuration consist of? What would be an initial configuration? A final configuration? See hint below.

c) Given a current configuration of a 2PDA, how do we determine what the next transition should be? (give an algorithm)

d) Given a current configuration of a 2PDA and the next transition, how do we compute what the next configuration will be? (give an algorithm)

Hint: look at the definitions of a 2-tape vs. a 1-tape TM for inspiration.

Problem 4. 2PDA expressiveness

Here are 3 facts about 2PDAs:

1. It is possible to simulate any (single-tape) TM M with a 2PDA A_M. Given any configuration of M in textbook’s notation (for example “abaqcd”) we would use the left stack to hold the left half of M’s tape contents including state (“aba” in the example) and the right stack to hold the right half M’s tape contents (“cdq” in the example).
2. It is also possible to simulate any 2PDA with a three-tape TM, where the first tape of $M_4$ initially contains the input string and the other two are initially empty. The contents of the first tape never changes, and the other two correspond to $A$'s stacks (with topmost stack symbol on the right).

3. It is also the case that the class of TMs is more expressive than the class of regular (1-stack) PDAs.

*Based on the three facts above*, what can you conclude about the class of 2PDAs?

a) How does it compare with regular PDAs?

b) How does it compare with Turing Machines?

*State your answers as formal theorems and prove them.*