Lec-5-HW-1, TM basics

(Problem 0)----------
Design a Turing Machine (TM), T_sub, that does unary decrement by one. Assume a legal, initial tape consists of a contiguous set of cells, each containing a "1", surrounded by blank tape to both the left and right. Assume the read-write head is initially positioned on the first blank cell to the right of the string of 1s. Have T_sub decrement from the left end of the string. Show an initial tape, the position of the read-write head, and a diagram showing T_sub's states with state transition arrows labeled with "input-symbol / output symbol / move" notation (a "state transition diagram"). Have T_sub halt after decrementing, leaving a legal input tape as output, with its read-write head positioned as it was at startup (to the right of the rightmost "1"). Assume any legal input is any positive integer. Use unary encoding: "1" for 0, "11" for 1, "111" for 2, and so on. You may indicate blank cells as containing the symbol "0", or you may use "#" as the symbol for a blank cell.

(PROBLEM 1)----------
Below is shown a part of a TM tape with an encoding of some TM, M. (We show only a part of the encoded M.) The encoding symbol set is {1, [,], {, }, #}. Note that exactly one of these symbols is present in a tape cell.

... # # # [1] { 1 # 1 # 1 1 # 1 1 # 1 1 } { 1 # 1 1 # 1 # 1 # 1 } ...

Here's the interpretation of the above characters:
- # = blank tape cell
- [= a tape cell containing the symbol "["
-] = a tape cell containing the symbol "]"
- { = a tape cell containing the symbol "{"
- } = a tape cell containing the symbol "}"
- 1 = a tape cell containing the symbol "1"

The part inside braces "{...}" represents a state transition rule of the form,

{ state, input-symbol, output-symbol, move, next-state }

Numbering for M's states and symbols is unary starting from 0. So, M's states are assumed be state-0, state-1, and so on, and are encoded as "1", "11", and so forth. Likewise, M's symbols are symbol-0, symbol-1, etc., and are encode "1", "11", etc. For moves, assume "1" indicates "move Left", and "11" indicates "move Right". The portion of the tape between the brackets "[...]" indicates M's current state, which initially is its start-state.

Draw the portion of M's state-transition diagram described on the tape. That is, draw and label whatever states are indicated, and draw arrows showing state transitions. Label the arrows "input/output/move" notation such as "2/3/R", which stands for "on input of symbol-2, output symbol-3 and move R". Also, interpret what the portion of M does.

(PROBLEM 2)----------
Is there any limit on how large a TM is? That is, is there a largest TM, in the sense that it has the maximum number of states, or the maximum number of symbols, or the maximum combined number of states and symbols? Is there any limit on the largest TM that can be described using the encoding scheme in Problem-1?

(Problem 3) ----------
This question asks for TM designs. Use state diagrams as shown in class. Show your symbol set. Draw state-transition diagrams, indicating the starting state. Show the initial tape configuration and the starting position of your TM's R/W-head relative to the input on its tape.

1. Design a TM, M-left, that moves left four cells from its starting position, and halts.

2. Now extend the design of M-left so that instead of halting, it reads whatever symbol it finds there, moves back to its starting position, writes that symbol in the initial cell, and then halts.

3. Design M-right, exactly a mirror image of M-left: it moves right four cells and copies the symbol found there to the starting cell.

4. Use the two machines above to build TM "M" that looks at the symbol in its starting cell and acts like M-left if it finds a "0" or acts like M-right if it finds a "1". You may use M-left and M-right as hierarchical sub-parts; that is, for instance, draw a box, label it "M-left" with a circle inside the box for M-left's start state and another for its halt state, the show a state transition from some state of your machine to M-left's start state. If your machine continues working after entering M-left's halt state, draw a state transition for M-left's halt state to the next state. The assumption is that any of M-left's transitions that enter its halt state actually transition to whatever state you have indicated its halt state transitions to.

(Problem 4) ---------------------------------
A finite state machine (FSM) that has "accepting" states and no outputs is called a language recognizer: if, when it runs out of input, it is in an accepting state, then the input stream is a string in the language the machine recognizes. These languages are called "regular", and there is a grammar for describing such a language's syntax called "regular expressions" (see unix "REGEX" man pages). Regular expressions are used to specify the strings to search for in almost all programs that do text searching. There is a one-to-one correspondence between regular languages and accepting FSMs. (The text search is actually implemented as a simulation of the corresponding FSM.) For instance, (a+)b*([c|d]e)* is a regular expression: "(a+)" says start with one or more "a"s, "b*" says next comes zero or more "b"s, "[c|d]e" says a "c" or "d" then "e" follows; so, "([c|d]e)*" says the string ends in zero or more "ce"s or "de"s. For instance, "aabbbdececede" is accepted. Show a state-transition diagram for an FSM accepting the language described by that regular expression. Here is the rule table for a machine accepting the language ([c|d]e)*:

state	input	next-state
A	c	B
A	d	B
B	e	C
C	c	B
C	d	B

The start state is A; states A and C are accepting states (it will accept an empty string). All unspecified transitions go to an "error", non-accepting state that halts the machine.

(Problem 5) -------------------------
In what follows, we will specify a machine's rules in this format:

{current-state, input, output, move, next-state}

For example, here is a machine that moves left forever:

§ States = {A} (ie., it has only a single state, "A".)
§ Symbols = {0, 1, #}
§ Start state = A
§ Rules:
□ {A, 0, 0, L, A}
□ {A, 1, 1, L, A}
□ {A, #, #, L, A}

Explain what the following machine does (Hint: draw its state machine diagram and trace out its execution for some sample inputs):

§ Set of states = {A, B, Halt}
§ Set of symbols = {0, 1, #}
§ Start state = A.
§ Transition rules:
□ {A, 0, 0, L, A}
□ {A, 1, 1, L, B}
□ {A, #, #, L, A}
□ {B, 0, 1, R, A}
□ {B, 1, 1, L, Halt}
□ {B, #, #, L, A}

(PROBLEM 6)----------
Simple counting can be done as above by using a fixed number of states, but in general, a TM should be able to handle any size of input. (Arbitrary counting can be done by using the tape to keep track of the count). For our purposes, computation is defined by what TMs can do. Take it for granted that computer programs essentially describe TMs using a special language (e.g., C). Consider a TM, T_abc, that produces all combinations of symbols using the symbol set, {a, b, c}, in order by length. For instance, T_abc would start out producing,

a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, aac, aba, abb, abc, aca, acb, acc, baa, bab, ...

on its tape. Note that this machine never halts, and that if you wait long enough, it will produce any finite string in a finite amount of time.

Since the strings get arbitrarily long, any equivalent computer program could not use fixed-size variables to construct the strings. The program might use a linked list to remember the last output string, but eventually the string would be too long and exceed the size of the memory. We can imagine that then one would have to resort to using disks and so forth. For our TMs, the tape is infinite; so, we can ignore that problem.

Sketch a method for generating the strings that does not depend on fixed resources such as registers or memory or counting by states. Hint, use copying and additional marker symbols. It is probably easier to think of this in TM terms rather than as a program (in C, for instance). You can suppose you have T_copy available that makes a fixed number of copies of a string delimited by specific special symbols, say e.g., double quotes. Assume T_copy produces the copies to the right side of current output. Argue that your T_abc, if completely fleshed out, plausibly has a finite number of states and a fixed and finite set of symbols.