Computation Theory (COMP 170), Fall 2020
Recitation 07

[1] Enumeration Machines

As part of some proofs of decidability, we need to rely upon the ability of Turing machines not only read in inputs and confirm they have the right basic form, but also to simulate other machines, and produce inputs for those machines. As part of such a process, suppose we wanted a TM to produce all of the binary integers, one at a time, in order.

It turns out it's slightly easier to generate binary numbers reversed, that is, with the least-significant bit first. E.g. 6, which is normally written in binary as 110, would instead be written as 011. So we'll generate our binary numbers in that reverse format. Our sequence of binary numbers will thus be:

$$0, 1, 01, 11, 001, 101, \ldots$$

Note that only one binary number should be on the tape at a time. We're simply trying to enumerate these strings one-by-one.

a. Write an algorithm, in plain language, for generating these strings on the tape of a TM. For convenience, assume that the least significant bit is furthest to the left, so that the numbers don't have to be shifted over on the tape as you go.

b. Give a transition diagram for a TM that implements this algorithm. (Note that it will not need an accept or reject state, since it is not deciding anything, only writing out numbers to its tape.)

c. Give the configurations of the TM from the previous question as it writes out the first four binary strings (starting with an empty tape):

$$0, 1, 10, 11$$

[2] DFA Emptiness

Consider the following language, which is the complement of the language E_{DFA} given in Sipser, Theorem 4.4 (3rd ed., p. 196):

$$NE_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) \neq \emptyset\}$$

If we assume that the alphabet of A is binary, can we use the following algorithm to decide NE_{DFA}?

1. Generate each of the possible binary strings, in order; for each, simulate A on that string.

2. If A accepts any given input string, then *accept.* If this never happens, then *reject.*

[3] Closure Properties

Suppose A and B are both recursive (a.k.a. Turing decidable) languages with total TM M_A and M_B such that $L(M_A) = A$ and $L(M_B) = B$. Since M_A and M_B are total, we know that

$$\begin{aligned} M_A \text{ accepts } x &\iff x \in A & \qquad M_A \text{ rejects } x &\iff x \notin A \\ M_B \text{ accepts } x &\iff x \in B & \qquad M_B \text{ rejects } x &\iff x \notin B \end{aligned}$$

We can prove that $A \cup B$ is recursive (a.k.a. Turing decidable) by constructing a total TM M from M_A and M_B as follows:

M = "On input x:

1. Run M_A on x. If M_A accepts x, *accept*; Otherwise (M_A rejects x), proceed.
2. Run M_B on x. If M_B accepts x, *accept*. Otherwise, *reject*."

Then by definition of $A \cup B$, M, and the properties of M_A and M_B stated above:

$$\begin{aligned} x \in A \cup B &\iff x \in A \text{ or } x \in B & \qquad x \notin A \cup B &\iff x \notin A \text{ and } x \notin B \\ &\iff M_A \text{ or } M_B \text{ accept } x & &\iff M_A \text{ and } M_B \text{ reject } x \\ &\iff M \text{ accepts } x. & &\iff M \text{ rejects } x. \end{aligned}$$

Use a similar technique to show that A^* is decidable. For best results, use the decidable proof paradigm resource to help structure your proof.