Computation Theory (COMP 170), Fall 2020
Assignment 07

Answer each problem below to the best of your ability. Submit all parts by 9:00 AM on Monday, November 9. List your collaborators. Late homework is accepted within 24 hours for half credit. After 24 hours no credit is given. The first late assignment (up to 24 hours) per student incurs no penalty. **Make sure that your submission follows the formatting guidelines given at the end of this document.**

Reading: Sipser Chapter 5

[1] (*10 pts.*) **Out of Context**

We'll say a fully general grammar (or FGG, pronounced "fugg," I suppose), is a tuple $G = (N, \Sigma, P, S)$ just like a CFG, except the productions in P are no longer restricted to having a single non-terminal on the left. That is, productions can be anything of the form $(N \cup \Sigma)^* \to (N \cup \Sigma)^*$. So whereas production rules in a CFG always look something like

$$A \to BcA$$

(note the single A on the left), a production rule in a FGG can look something like

$$ABc \to CAAd$$

The language $L(G)$ is simply those strings in Σ^* derivable from S.

In this problem, you'll be proving that these grammars are just as powerful as Turing machines. That is, for any TM $M = (Q, \Sigma, \Gamma, \delta, q_s, q_a, q_r)$, there is an FGG G such that $L(G) = L(M)$. One challenge here is that grammars generate an entire language, whereas Turing machines take in a single string and say yes or no. So our grammar needs to be capable of simulating M on all possible inputs. Thus our grammar will proceed in two phases. First, it will generate an arbitrary starting configuration. Then, it will update that configuration as M would have.

a. To simulate M, we'll need non-terminals that describe pieces of configurations. To that end, we'll introduce two types of non-terminals:

$$\begin{bmatrix} a \\ b \\ q \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} a \\ b \\ \end{bmatrix}$$

for all $a, b \in \Gamma, q \in Q$. The top symbol represents an initial input character. The middle symbol represents the current tape symbol. And the bottom symbol represents the state and position of the read head (blank indicates the read head is elsewhere). First, create production rules that allow for the generation of arbitrary start configurations. Note that the top two symbols should always match. This part of the grammar can probably be context free.

b. Now show how to modify your grammar to allow it to simulate M (based on δ, presumably). Don't modify the top symbols of the non-terminals (we'll get to those in part c). If you consider only the bottom two symbols, the sequence of strings your grammar yields should correspond exactly to configurations M would step through. You may want to modify your solution in part a to keep track of the last used space on the tape. You should be able to get away with only a few symbols on the LHS of any production rule.

c. Finally, explain how to add production rules so that the grammar produces a string $x \in \Sigma^*$ if and only if M would have accepted x.

[2] (*6 pts.*) **Tough Decisions**

For each of the following questions, state (and briefly justify) whether it is decidable, Turing recognizable but not decidable, or not even Turing recognizable.

a. Given a DFA D, is $L(D)$ finite?

b. Given a TM M, is there any string on which M halts?

c. Given a TM M and a string x, does M ever read all of x?

[3] (*4 pts.*) **Closure Properties**

a. Suppose A and B are both Turing recognizable. We'd like to argue that $A \cup B$ is also Turing recognizable. Here's an attempt:

Since A and B are Turing recognizable, there exist Turing machines M_A and M_B such that $L(M_A) = A$ and $L(M_B) = B$. Create a new Turing machine M which on input x works as follows:

1. Simulate M_A on x. If it accepts, accept. If it rejects, continue to step 2.

2. Simulate M_B on x. If it accepts, accept. If it rejects, reject.

Does this work? Why or why not? If not, how can we fix it?

b. Suppose we similarly design a TM to show that $A \cap B$ is Turing recognizable.

1. Simulate M_A on x. If it rejects, reject. If it accepts, continue to step 2.

2. Simulate M_B on x. If it rejects, reject. If it accepts, accept.

Does this work? Why or why not?

Format requirements: work for COMP 170 should correspond to the following guidelines:

- Work must be in type-written format, with any diagrams rendered using software to produce professional-looking results. No hand-written or hand-drawn work will be graded.
- Work must be submitted in PDF format to Gradescope.
- Each answer should start on a new page of the document. When possible, try to limit answers to a single page each. (Thus, the answers to this homework must be no less than three pages, and preferably no more.)

You can find links to information about using LaTeX to produce type-written mathematical work,[1] and to a handy web-based tool for drawing finite-state diagrams, on the Piazza class site:

https://piazza.com/tufts/fall2020/comp170/resources

[1] LaTeX was used to produce this document.