CS 374 Spring 2018
Homework 4

Nathaniel Murphy (njmurph3@illinois.edu)
Tanvi Modi (tmodi3@illinois.edu)
Marianne Huang (mhuang46@illinois.edu

Problem 3

Let us define a few functions that will help in defining M_G. ($\mathbb{W}$ represents the set of whole numbers)

- Define $\#_{ops}: \ R_G \to \mathbb{W}$ such that $\#_{ops}(\gamma) = \ \#$ distinct options in non-terminal γ
- Define a bijection $c: \ (R_G, \mathbb{W}) \to w$, where $w \in (\Sigma_G \cup \Gamma_G)^*$
- Define a bijection $r: \ \mathbb{W} \to R_G$, such that $r(0) = S_G$

Let us now define $M_G = (\Gamma, \square, \Sigma, Q, s, \text{accept}, \text{reject}, \delta)$:

- $\Gamma = \Sigma_G \cup \Gamma_G \cup \{\triangleright, \square\}$
- $\square = \square$
- $\Sigma = \Sigma_G$
- $Q = \{\text{start, rt, accept, reject, temp}\} \cup \{(w, i) \mid 0 \le i \le |w|\} \cup \{(w_{ij}, k) \mid w_{ij} = c(r(i), j), 0 \le k < |w_{ij}\}$
- $s = \text{start}$
- accept $=$ accept
- reject $=$ reject
- Let us now define δ:

$$\delta(q, t_1, t_2) \begin{cases} \big((w,0), (t_1, 0), (\triangleright, 1)\big) & \text{if } q = \text{ start} \\ \big((w, i+1), (w[k-i], -1), (t_2, 0)\big) & \text{if } q = (w, i), \ k = |w|, \ i < |w| \\ \big(\{(w_{0j}, 0) \mid 0 \le j < \#_{ops}(S_G)\}, (t_1, 0), (\square, 0)\big) & \text{if } q = (w, i), \ i = |w| \\ \big(\{(w_{ij}, 0) \mid 0 \le j < \#_{ops}(t_2)\}, (t_1, 0), (\square, 0)\big) & \text{if } t_2 \in \Gamma_G \\ \big((w_{ij}, k+1), (t_1, 0), (w_{ij}[l-k], 1)\big) & \text{if } q = (w_{ij}, k), \ \ell = |w_{ij}|, \ k < \ell \\ \big(\text{temp}, (t_1, 0), (\square, -1)\big) & \text{if } q = (w_{ij}, k), \ k = |w_{ij}| \\ \big(\text{rt}, (t_1, 1), (\square, -1)\big) & \text{if } q \in \{\text{temp, rt}\}, \ t_2 \in \Sigma \ \wedge \ t_1 = t_2 \\ \big(\text{reject}, (t_1, 0), (t_2, 0)\big) & \text{if } t_2 \in \Sigma \ \wedge \ t_1 \ne t_2 \\ \big(\text{accept}, (t_1, 0), (t_2, 0)\big) & \text{if } t_1 = \square \ \wedge \ t_2 = \triangleright \end{cases}$$

Notice that transitioning into a $\{(w_{ij}, 0)\}$ state may spawn multiple threads if $\#_{ops}(R) > 1$ for a non-terminal R.

When defining the states, the {start, accept, reject} states are trivial to understand, however, some explanation may be required for the {temp, rt} states. The temp state is the state that the machine goes into right after it has finished writing an entire string from a non-terminal to the tape. This makes it so that if the last letter it wrote was a terminal symbol, the machine will enter the {rt} state, and if the last symbol that it writes to the tape is a non-terminal, it will go back into a $\{(w_{ij}, k)\}$ state, the set of states used to non-deterministically write the non-terminals. The rt state is just for reading non-terminals from tape 2 and comparing them with the non-terminals on tape 1. Notice that it advances tape 1's head by 1 and tape 2's head by negative 1 because the string written to tape 2 is inverted. With that being said, notice the way that the machine writes a non-terminal to tape 2. It writes it backwards to ensure that the beginning of that non-terminal is at the end of the tape, because that is how the algorithm was designed. Finally, being in a (w, i) state means that the machine has written the first i characters of the input string to tape 1 and will continue to do so.

The functions defined at the top are only to simplify my definitions of the states. $\#_{ops}$ simply maps a rule to a whole number, that whole number being the number of paths (for lack of a better word) to choose from that rule. c intuitively 'indexes' the options within a rule r so that they may be accessed by using whole number indices. And finally, r indexes the rules of a context free grammar, so that they may be accessed by using whole number indices.