# Homework Assignment #10
# Due: December 5, 2016 at 4:00 p.m.

**1.** Consider the context-free grammar $G$ with starting symbol $S$ and terminals `a` and `b` and rules

$$\begin{aligned} S &\to S\texttt{a} \\ S &\to \texttt{b}SS \\ S &\to S\texttt{b}S \\ S &\to SS\texttt{b} \\ S &\to \texttt{a} \end{aligned}$$

Prove that every string generated by $G$ has more `a`'s than `b`'s.

**2.** Consider the following context-free grammar for a simple programming language. The set of terminals is $\{\texttt{0}, \texttt{1}, \texttt{2}, \texttt{3}, \texttt{4}, \texttt{5}, \texttt{6}, \texttt{7}, \texttt{8}, \texttt{9}, \texttt{R}, \texttt{[}, \texttt{]}, =, \neq, \texttt{-}, \texttt{+}, >, \texttt{d}, \texttt{e}, \texttt{f}, \texttt{h}, \texttt{i}, \texttt{l}, \texttt{n}, \texttt{s}, \texttt{t}, \texttt{w}\}$ and the set of variables is $\{\langle\text{PROGRAMME}\rangle, \langle\text{STATEMENT}\rangle, \langle\text{NUM}\rangle, \langle\text{DIGIT}\rangle\}$. The starting symbol is $\langle\text{PROGRAMME}\rangle$.

$$\begin{aligned} \langle\text{PROGRAMME}\rangle &\to \langle\text{STATEMENT}\rangle\langle\text{PROGRAMME}\rangle \\ \langle\text{PROGRAMME}\rangle &\to \varepsilon \\ \langle\text{STATEMENT}\rangle &\to \texttt{if R[}\langle\text{NUM}\rangle\texttt{] = 0 then }\langle\text{PROGRAMME}\rangle\texttt{ else }\langle\text{PROGRAMME}\rangle\texttt{ end if} \\ \langle\text{STATEMENT}\rangle &\to \texttt{while R[}\langle\text{NUM}\rangle\texttt{]} \neq \texttt{0 }\langle\text{PROGRAMME}\rangle\texttt{ end while} \\ \langle\text{STATEMENT}\rangle &\to \texttt{R[}\langle\text{NUM}\rangle\texttt{]++} \\ \langle\text{STATEMENT}\rangle &\to \texttt{R[}\langle\text{NUM}\rangle\texttt{]--} \\ \langle\text{NUM}\rangle &\to \langle\text{DIGIT}\rangle \\ \langle\text{NUM}\rangle &\to \langle\text{DIGIT}\rangle\langle\text{NUM}\rangle \\ \langle\text{DIGIT}\rangle &\to \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \end{aligned}$$

These programmes manipulate the contents of an infinite array of registers $\texttt{R[}0, 1, 2, \ldots\texttt{]}$. Each register stores a natural number. The value in the register $\texttt{R[}n\texttt{]}$ can be incremented and decremented by the instructions $\texttt{R[}n\texttt{]++}$ and $\texttt{R[}n\texttt{]--}$. Exception: if the value stored in $\texttt{R[}n\texttt{]}$ is 0, then the statement $\texttt{R[}n\texttt{]--}$ has no effect. This ensures that the value stored in each register is always a natural number. The `if` and `while` have their usual meaning. It turns out that this very simple programming language is strong enough to simulate anything that can be done on a Turing machine.

The following example programme, which we will call $\text{COPY}(i, j, k)$ (where $i, j, k$ are distinct natural numbers), is a programme that copies the contents of $\texttt{R[}i\texttt{]}$ into $\texttt{R[}j\texttt{]}$, using a third register $\texttt{R[}k\texttt{]}$ as temporary storage. The programme is broken up into separate lines and indented to make it easier to read. The text enclosed in brace brackets {...} are just comments and are not really part of the programme.

```
while R[k] ≠ 0              {Set R[k] to 0}
   R[k]--
end while
while R[j] ≠ 0              {Set R[j] to 0}
   R[j]--
end while
while R[i] ≠ 0              {Move value in R[i] to R[k] while changing R[i] to 0}
   R[i]--
   R[k]++
end while
while R[k] ≠ 0              {Move value in R[k] into both R[i] and R[j]}
   R[k]--
   R[i]++
   R[j]++
end while
```

Write a programme that sets the value of `R[2]` to the product of the values stored in `R[0]` and `R[1]`. (The values in `R[0]` and `R[1]` should be the same at the end of your programme as they were at the beginning.) In other words, your programme should have the same effect as the Java instruction `R[2] = R[0] * R[1]`. You may use the COPY$(i, j, k)$ programme as a subroutine and you may use any other registers as temporary storage. Do not assume anything about the values stored in any registers when your programme begins, except that they are natural numbers. You need not prove your algorithm is correct. You may add comments to your programme to help the marker understand what the programme is doing.