Patrick Pei
CS3252-01
Due: 03/21/17

Assignment 8:

*5.1.3

Base case: any regular expression with 0 operators can be represented by the mapping of start symbol S to any character c (this includes ε)

$S \rightarrow c$.

Inductive step: suppose that for any regular expression, it has been produced by a CFG with the same language. Then, it is enough to prove for each of the operators of regular expressions that the CFGs constructed are also regular. (+, concatenate, *)

1. union

$S \rightarrow S_1 \mid S_2$

equivalent to

$S \rightarrow S_1$

$S \rightarrow S_2$

R_1 R_2 are regular expressions

S_1 S_2 are the set of strings matched respectively by R_1 and R_2.

Thus, S will match either the expression specified by R_1 or R_2 depending on which production was used.

2. concatenation

$S \rightarrow S_1 S_2$

Thus, S will start from the start state and match exactly the strings matched by the concatenation of regular expressions R_1, R_2

3. Kleene Closure

$S \rightarrow S_i S_i \mid \varepsilon$

Finally since every regular expression has an equivalent CFG, every regular language is a context free language.

*5.4.5

a. The following grammar:

$S \rightarrow A1B$

$A \rightarrow 0A \mid \varepsilon$

$B \rightarrow 0B \mid 1B \mid \varepsilon$ is unambiguous by inspection

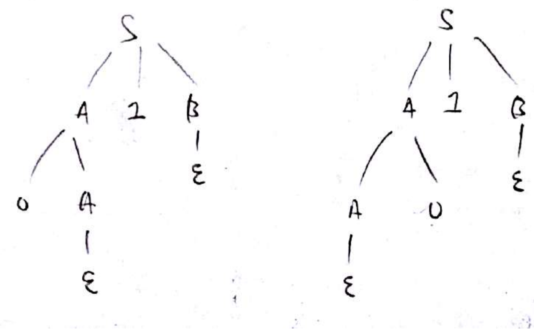
Clearly, any derivation of the described grammer contains a 1 and since the productions of A only produce 0's, the 1 produced by the start symbol is guaranteed to be the first 1 in the string. Also, these 0's generated by A are one by one added to the left with no ambiguity possible. Thus, it can be concluded that the entire grammar is unambiguous since after the 1st 1, B produces any pattern of 0 and 1's all added to the right of the current string strictly.

b. $S \rightarrow A1B$

$A \rightarrow 0A \mid A0 \mid \varepsilon$

$B \rightarrow 0B \mid 1B \mid \varepsilon$

01 parse trees:



*5.1.4

a. proof: every right-linear grammar generates a regular language

A CFG is said to be right-linear if:

1. each production body has at most one variable
2. that one variable is at the right end

In other words, all variables will be of the form $A \to wB \mid w$

$B \to wC \mid w$

...

where w is some string of 0 or more terminals

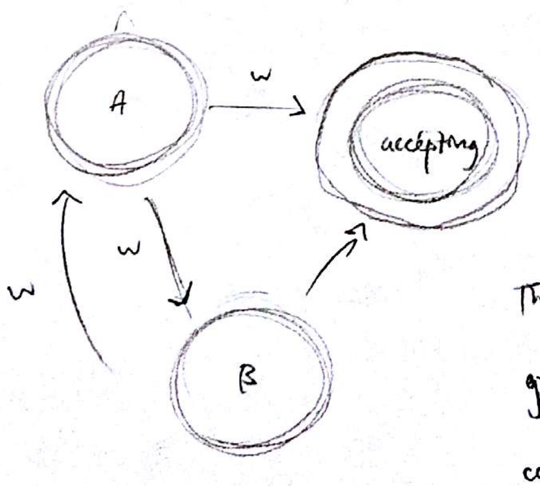
Thus, it suffices to prove an example right-linear CFG generates a regular language by showing that the derivation process is representable by a finite state automaton.

The example is:

$A \to wB \mid w$

$B \to wA \mid w$

then, for each grammar variable, a state will be assigned to it in the automaton. Furthermore, an accepting state must be added. finally, the extended transition function $\hat{\delta}$ can be used to create state transitions representing productions.



note that w in each case can be distinct and that the transitions between may be of an unspecified length (ε, 1, 2+)

Thus, the very representation proves that right-linear grammars generate a regular language (this can obviously be extended for different right-linear CFGs of various variables / productions.

* 5.1.4b

Proof: every regular language has a right-linear grammar.

For any regular language, there is a corresponding DFA representation.
Thus, it suffices to show that this DFA accepts the language of the right-linear grammar.

D is a DFA such that $D = (Q, \Sigma_1, \delta, q_0, F)$ where $Q = \{q_0$ through $q_n\}$
and $\Sigma_1 = \{a_0$ through $a_m\}$

Then the right-linear grammar $G = (V, \Sigma_1, S, P)$ can be formed where

V, the variables are the states q_0 through q_n (this is similar to the process of 5.1.4a previously described)

S, the start symbol is clearly q_0.

Then, T the terminals will be the set of $\{a_0$ through $a_m\}$

Finally, the productions, P are formed by analyzing each transition from q_i to q_k represented by $\hat{\delta}(q_i, a_j) = q_k$ such that $q_i \rightarrow a_j q_k$, thus enforcing right-linearity. Finally, each accepting state maps to ε.

By the above procedure, each production per transition allows for acceptance of all strings described by the regular language since transitions are followed to create strings of the form a_i.

*6.2.1 $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

b. PDA P that accepts by final state:

$$P = (\{q, p\}, \{0, 1\}, \{x, Z_0\}, \delta, q, Z_0, \{p\})$$

$\delta(q, 0, Z_0) = \{(q, xZ_0)\}$

$\delta(q, 0, x) = \{(q, xx)\}$

$\delta(q, 1, x) = \{(q, \varepsilon)\}$

$\delta(q, \varepsilon, x) = \{(p, x)\}$ more 0's than 1's at the end

$\delta(q, \varepsilon, Z_0) = \{(p, Z_0)\}$ equal number of 0's and 1's

c. $P = (\{q, p\}, \{0, 1\}, \{Z_0, x, Y\}, \delta, q, Z_0, \{p\})$

use x and y's to keep balance

by final state acceptence

$\delta(q, 0, Z_0) = \{(q, xZ_0)\}$

$\delta(q, 0, x) = \{(q, xx)\}$

$\delta(q, 0, Y) = \{(q, \varepsilon)\}$

$\delta(q, 1, Z_0) = \{(q, YZ_0)\}$

$\delta(q, 1, Y) = \{(q, YY)\}$

$\delta(q, 1, x) = \{(q, \varepsilon)\}$

$\delta(q, \varepsilon, Z_0) = \{(p, Z_0)\}$