Homework Four Solution– CSE 355

Due: 29 March 2012

Please note that there is more than one way to answer most of these questions. The following only represents a sample solution.

Problem 1: Linz 5.1.23 and 6.2.4

5.1.23: Find a context-free grammar for the set of all regular expressions on the alphabet $\{a, b\}$.

Let G be the grammar given by the following productions:

$$S \to a|b|\lambda|\emptyset|S + S|SS|S^*|(S)$$

Let R be the set of all regular expressions over $\{a, b\}$. Then any element in R must have one of the seven forms listed in the definition of regular expression given in Definition 3.1. These are precisely the only strings that G can generate, with each production representing one of the generation rules for regular expressions.

More formally (but not required in your solution), let $G = (\{S\}, \{a, b, \lambda, \emptyset, +, ^*, (,)\}, S, P)$, where P is the set of productions given above.

Now, we must show that $L(G) = R$. Let $r \in L(G)$. We will show that $r \in R$ by induction on the number of derivations needed to generate r.

- Base Case: Assume r is generated using 1 derivation. The only productions in P that end in a terminal will generate one of the following strings: $r = a$, $r = b$, $r = \lambda$ or $r = \emptyset$. In each case r is a regular expression and hence $r \in R$.
- Induction Hypothesis: Assume that if G generates r in fewer than n derivations then $r \in R$.
- Induction Step: Assume that G generates r in n derivations, with $n \geq 2$. Since r is not generated in only one production, one of the following productions of G must be used:
  - $S \to S + S$. In this case $S \Rightarrow S + S \Rightarrow^* r_1 + S \Rightarrow^* r_1 + r_2$, where r_1 and r_2 are regular expressions from the induction hypothesis (takes fewer than n derivation to generate since one derivation was used to get the sentential form $S + S$). Therefore, $r = r_1 + r_2$ and we get that $r \in R$.
  - $S \to SS$. In this case $S \Rightarrow SS \Rightarrow^* r_1S \Rightarrow^* r_1r_2$, where r_1 and r_2 are regular expressions from the induction hypothesis. Therefore, $r = r_1r_2$ and we get that $r \in R$.

- $S \to S^*$. In this case $S \Rightarrow S^* \Rightarrow^* r_1^*$, where r_1 is a regular expression from the induction hypothesis. Therefore, $r = r_1^*$ and we get that $r \in R$.
- $S \to (S)$. In this case $S \Rightarrow (S) \Rightarrow^* (r_1)$, where r_1 is a regular expression from the induction hypothesis. Therefore, $r = (r_1)$ and we get that $r \in R$.

In each case we get that $r \in R$

- Therefore, we conclude that if $r \in L(G)$ then $r \in R$. Whence, $L(G) \subseteq R$.

We still have to show that $R \subseteq L(G)$. Assume $r \in R$. We will show that $r \in L(G)$ by induction on the length of r.

- Base Case: Assume $|r| = 1$. Then r is a primitive regular expression and must have one of the following forms: $r = x$, where $x \in \{a, b, \lambda, \emptyset\}$. In each case $S \Rightarrow x$ will generate r. Thus, $r \in L(G)$.
- Induction Hypothesis: Assume that if $|r| < n$ then $S \Rightarrow^* r$, that is $r \in L(G)$.
- Induction Step: Assume that $|r| = n$, with $n \geq 2$. Since $|r| > 1$, r must have one of the following forms:
  - $r = r_1 + r_2$, where r_1 and r_2 are regular expressions, each with length less than n. By the induction hypothesis $S \Rightarrow^* r_1$ and $S \Rightarrow^* r_2$. Therefore, $S \Rightarrow S + S \Rightarrow^* r_1 + r_2 = r$. Thus, $r \in L(G)$.
  - $r = r_1 r_2$, where r_1 and r_2 are regular expressions, each with length less than n. By the induction hypothesis $S \Rightarrow^* r_1$ and $S \Rightarrow^* r_2$. Therefore, $S \Rightarrow SS \Rightarrow^* r_1 r_2 = r$. Thus, $r \in L(G)$.
  - $r = r_1^*$, where r_1 is a regular expressions with length less than n. By the induction hypothesis $S \Rightarrow^* r_1$. Therefore, $S \Rightarrow S^* \Rightarrow^* r_1^* = r$. Thus, $r \in L(G)$.
  - $r = (r_1)$, where r_1 is a regular expressions with length less than n. By the induction hypothesis $S \Rightarrow^* r_1$. Therefore, $S \Rightarrow (S) \Rightarrow^* (r_1) = r$. Thus, $r \in L(G)$.

  In each case we get that $r \in L(G)$
- Therefore, we conclude that if $r \in L(G)$ then $r \in R$. Whence, $R \subseteq L(G)$.

Therefore, we have that $L(G) = R$ and the claim is proved.

6.2.4: Transform the grammar with productions

$$\begin{array}{lcl} S & \to & abAB, \\ A & \to & baB|\lambda, \\ B & \to & BAa|A|\lambda. \end{array}$$

into Chomsky normal form.

Removing $\lambda-$productions: Here A and B are nullable variables. Then following the second step of construction in Theorem 6.3, we get

$$\begin{array}{lcl} S & \to & abAB|abA|abB|ab, \\ A & \to & baB|ba, \\ B & \to & BAa|A|Ba|Aa|a. \end{array}$$

Removing unit-productions(by Theorem 6.4):

$$\begin{aligned} S &\rightarrow abAB|abA|abB|ab, \\ A &\rightarrow baB|ba, \\ B &\rightarrow BAa|baB|ba|Ba|Aa|a. \end{aligned}$$

There are no useless productions in the grammar.

By step 1 of Theorem 6.6, we introduce variables C and D to substitute terminals a and b.

$$\begin{aligned} S &\rightarrow CDAB|CDA|CDB|CD, \\ A &\rightarrow DCB|DC, \\ B &\rightarrow BAC|DCB|DC|BC|AC|a, \\ C &\rightarrow a, \\ D &\rightarrow b. \end{aligned}$$

By step 2 of Theorem 6.6, we introduce variables to shorten the right sides of the production.

$$\begin{aligned} S &\rightarrow EF|EA|EB|CD \\ A &\rightarrow GB|DC, \\ B &\rightarrow BH|GB|DC|BC|AC|a, \\ E &\rightarrow CD, \\ F &\rightarrow AB, \\ G &\rightarrow DC, \\ H &\rightarrow AC, \\ C &\rightarrow a, \\ D &\rightarrow b. \end{aligned}$$

Now the grammar is in CNF.

Problem 2: Linz 7.1.17 and 7.2.5

7.1.17: An alternative to Definition 7.2 for language acceptance is to require the stack to be empty when the end of the input string is reached. Formally, an npda M is said to accept the language $N(M)$ by empty stack if

$$N(M) = \{w \in \Sigma^* : (q_0, w, z) \vdash_M^* (p, \lambda, \lambda)\},$$

where p is any element in Q. Show that this notion is effectively equivalent to Definition 7.2, in the sense that for any npda M there exists an npda $\widehat{M}$ such that $L(M) = N(\widehat{M})$, and vice versa.

We will give a procedure to convert an npda $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ into a new npda $\widehat{M} = (\widehat{Q}, \Sigma, \widehat{\Gamma}, \widehat{\delta}, \widehat{q_0}, z, F)$ such that $L(M) = N(\widehat{M})$.

1. Add new states q_s and q_f to Q. That is $\widehat{Q} = Q \cup \{q_s, q_f\}$.
2. Make q_s the start state of $\widehat{M}$. That is $\widehat{q_0} = q_s$. This is used to add a new bottom element to our stack.

3. Add a new stack symbol \$ to Γ. That is $\widehat{\Gamma} = \Gamma \cup \{\$\}$. This will be used to ensure that $\widehat{M}$ does not accept in a nonfinal state just because the stack of M is empty after reading the input.

4. Create $\widehat{\delta}$ as follows:

   – Add the transition $\widehat{\delta}(q_s, \lambda, z) = \{(q_0, z\$)\}$. This places our new stack symbol on the bottom of the stack.

   – Add the transitions $\widehat{\delta}(p, \lambda, a) = \{(q_f, \lambda)\}$, for all $p \in F$, $a \in \widehat{\Gamma}$. This nondeterministically checks that if we are in a final state, have no input, but have elements on the stack.

   – Add the transition $\widehat{\delta}(q_f, \lambda, a) = \{(q_f, \lambda)\}$, for all $a \in \widehat{\Gamma}$. This empties the stack in the new state, thus accepting.

   – All other transitions are the same as in δ. That is, $\widehat{\delta}(q, b, a) = \delta(q, b, a)$, for all $q \in Q$, $b \in \Sigma$, and $a \in \Gamma$.

5. This completes the creation of $\widehat{M}$

Now we must show that $L(M) = N(\widehat{M})$. If a string is not accepted by $L(M)$, then every state that it can reach with no input left to read must be a non-final state. This string will end in the same set of states in $\widehat{M}$ (since the only transition to q_f is from a final state). However, the stack must have the symbol \$ on it since the only way to pop it is from an accepting state or q_f. Therefore, the string is also not in $N(\widehat{M})$.

Conversely, if a string is in $L(M)$, then there must be some sequence of moves so that it ends in a final state with no input remaining to be read. Then the string would be in the same set of states in $\widehat{M}$ but would also be in q_f with an empty stack by following the transition from the final state and then emptying the stack. Therefore, the string is also in $N(\widehat{M})$. Thus, $L(M) = N(\widehat{M})$.

More formally (again not required), $w \in L(M)$ iff there is a sequence of moves in M, $(q_0, w, z) \vdash_M^* (p, \lambda, ab)$ with $p \in F$, $a \in \Gamma \cup \{\lambda\}$ and $b \in \Gamma^*$ iff there is a sequence of moves in $\widehat{M}$, $(q_s, w, z) \vdash_{\widehat{M}} (q_0, w, z\$) \vdash_{\widehat{M}}^* (p, \lambda, ab\$) \vdash_{\widehat{M}} (q_f, \lambda, b\$) \vdash_{\widehat{M}}^* (q_f, \lambda, \lambda)$ (since $\widehat{M}$ has the same moves as M except for the new start and from final states) iff $w \in N(\widehat{M})$.

7.2.5: Construct an npda corresponding to the grammar

$$\begin{aligned} S &\rightarrow aABB|aAA, \\ A &\rightarrow aBB|a, \\ B &\rightarrow bBB|A. \end{aligned}$$

The grammar does not have any $\lambda-$productions. So we can convert it into Greibach normal form. By applying Theorem 6.1 on last production,

$$\begin{aligned} S &\rightarrow aABB|aAA, \\ A &\rightarrow aBB|a, \\ B &\rightarrow bBB|aBB|a. \end{aligned}$$

By applying construction in Theorem 7.1, Define $M = (\{q_0, q_1, q_f\}, \Sigma, \{S, A, B, z\}, \delta, q_0, z, \{q_f\})$ with

$$\begin{aligned}
\delta(q_0, \lambda, z) &= \{(q_1, Sz)\}, \\
\delta(q_1, a, S) &= \{(q_1, ABB), (q_1, AA)\}, \\
\delta(q_1, a, A) &= \{(q_1, BB), (q_1, \lambda)\}, \\
\delta(q_1, b, B) &= \{(q_1, BB)\}, \\
\delta(q_1, a, B) &= \{(q_1, BB), (q_1, \lambda)\}, \\
\delta(q_1, \lambda, z) &= \{(q_f, z)\}.
\end{aligned}$$

Problem 3: Linz 5.2.11 and 6.1.6

5.2.11: Is it possible for a regular grammar to be ambiguous?

Yes. Consider the grammar

$$\begin{aligned}
S &\rightarrow A|\lambda \\
A &\rightarrow \lambda.
\end{aligned}$$

Then there are two leftmost derivations that generate the string λ. The first is $S \Rightarrow \lambda$ and the second is $S \Rightarrow A \Rightarrow \lambda$. Thus, this regular grammar is ambiguous.

Note as problem 5.2.9 states that regular languages are not inherently ambiguous. This is because one can make a DFA for the language and convert that DFA to a grammar. Since each string has only one path in a DFA the resulting grammar would also have only one derivation following the same path in the DFA.

6.1.6: Eliminate useless productions from

$$\begin{aligned}
S &\rightarrow a|aA|B|C, \\
A &\rightarrow aB|\lambda, \\
B &\rightarrow Aa, \\
C &\rightarrow cCD, \\
D &\rightarrow ddd.
\end{aligned}$$

Following the algorithm given in the proof of Theorem 6.2, we will first remove productions that cannot generate any strings. Let $V_1 = \emptyset$. Since $S \rightarrow a$, $A \rightarrow \lambda$ and $D \rightarrow ddd$ are productions we add S, A and D to V_1. Then, since $B \rightarrow Aa$ is a production we add B to V_1. There are no more variables that can produce a string following the algorithm. Removing these useless productions yields the grammar

$$\begin{aligned}
S &\rightarrow a|aA|B, \\
A &\rightarrow aB|\lambda, \\
B &\rightarrow Aa, \\
D &\rightarrow ddd.
\end{aligned}$$

Next we remove the unreachable variables. We will start with $V_2 = \{S\}$ since S is the start variabe, we always reach it. We can reach the variables A and B from S so they are added to V_2. A and B can only reach each other so our algorithm terminates. Since D is not in S_2 it cannot be

reached by any derivation of this grammar. Therefore we remove all productions with D, yielding the final grammar free of useless productions and variables

$$\begin{aligned} S &\rightarrow a|aA|B, \\ A &\rightarrow aB|\lambda, \\ B &\rightarrow Aa, \end{aligned}$$

Problem 4: Linz 7.1.4

Construct npda's that acccept the following languages on $\Sigma = \{a, b, c\}$.

(a) $L = \{a^n b^{2n} : n \geq 0\}$**.**

We will give the machine definition. It is recommended you draw this out.

Define $M = (\{q_0, q_1, q_2, q_3\}, \Sigma, \{a, z\}, \delta, q_0, z, \{q_0, q_3\})$ with $\delta(q_0, a, z) = \{(q_1, aaz)\}$, $\delta(q_1, a, a) = \{(q_1, aaa)\}$, $\delta(q_1, b, a) = \{(q_2, \lambda)\}$, $\delta(q_2, b, a) = \{(q_2, \lambda)\}$, $\delta(q_2, \lambda, z) = \{(q_3, \lambda)\}$.

Then M accepts L since it accepts the empty string and for each a seen it pushes two additional as onto the stack (3 total). Then it pops one a for each b seen guaranteeing there are twice as many bs as as. Finally it accepts if the input is done and there are no more as on the stack.

(b) $L = \{wcw^R : w \in \{a, b\}\}$**.**

We will give the machine definition. It is recommended you draw this out.

Define $M = (\{q_0, q_1, q_2\}, \Sigma, \{a, b, z\}, \delta, q_0, z, \{q_2\})$ with $\delta(q_0, a, a) = \{(q_0, aa)\}$, $\delta(q_0, b, a) = \{(q_0, ba)\}$, $\delta(q_0, a, b) = \{(q_0, ab)\}$, $\delta(q_0, b, b) = \{(q_0, bb)\}$, $\delta(q_0, a, z) = \{(q_0, az)\}$, $\delta(q_0, b, z) = \{(q_0, bz)\}$, $\delta(q_0, c, z) = \{(q_1, z)\}$, $\delta(q_0, c, a) = \{(q_1, a)\}$, $\delta(q_0, c, b) = \{(q_1, b)\}$, $\delta(q_1, a, a) = \{(q_1, \lambda)\}$, $\delta(q_1, b, b) = \{(q_1, \lambda)\}$, $\delta(q_1, \lambda, z) = \{(q_2, z)\}$.

Then M accepts L. It accepts c. For each a or b seen (if any) it pushes an a or b respectively onto the stack and stays in state q_0. When it encounters a c, it does not push anything on to the stack, but gets to the next state, q_1. While being in state q_1, if it encounters an a or b, it matches it with the a or b respectively on the stack and pops one a for each a or one b for each b seen guaranteeing that it matches w^R against the contents of the stack and the matching starts after it encounters a c. Finally it accepts if the input is done and there are no more as or b on the stack.

(c) $L = \{a^n b^m c^{n+m} : n \geq 0, m \geq 0\}$**.**

We will give the machine definition. It is recommended you draw this out.

Define $M = (\{q_0, q_1, q_2, q_3, q_4\}, \Sigma, \{a, z\}, \delta, q_0, z, \{q_0, q_4\})$ with $\delta(q_0, \lambda, z) = \{(q_1, z)\}$, $\delta(q_1, a, z) = \{(q_1, az)\}$, $\delta(q_1, a, a) = \{(q_1, aa)\}$, $\delta(q_1, \lambda, a) = \{(q_2, a)\}$, $\delta(q_1, \lambda, z) = \{(q_2, z)\}$, $\delta(q_2, b, z) = \{(q_2, az)\}$, $\delta(q_2, b, a) = \{(q_2, aa)\}$, $\delta(q_2, c, a) = \{(q_3, \lambda)\}$, $\delta(q_3, c, a) = \{(q_3, \lambda)\}$, $\delta(q_3, \lambda, z) = \{(q_4, \lambda)\}$.

Then M accepts L since it accepts the empty string and for each a seen (if any) it pushes an as onto the stack. Then it pushes an a onto the stack for each b seen (if any). Thus after reading all as and bs in the input the stack has $n_a(w) + n_b(w)$ as on it. It then pops one a for each c seen guaranteeing that $n_a(w) + n_b(w) = n_c(w)$ and they are in the right order. Finally it accepts if the input is done and there are no more as on the stack.

(d) $L = \{a^n b^{m+n} c^m : n \geq 0, m \geq 1\}$.

We will give the machine definition. It is recommended you draw this out.

Define $M = (\{q_0, q_1, q_2, q_3, q_4\}, \Sigma, \{a, b, z\}, \delta, q_0, z, \{q_4\})$ with $\delta(q_0, \lambda, z) = \{(q_1, z)\}$, $\delta(q_0, a, z) = \{(q_0, az)\}$, $\delta(q_0, a, a) = \{(q_0, aa)\}$, $\delta(q_0, b, a) = \{(q_1, \lambda)\}$, $\delta(q_0, b, z) = \{(q_1, bz)\}$, $\delta(q_1, b, a) = \{(q_1, \lambda)\}$, $\delta(q_1, b, z) = \{(q_2, bz)\}$, $\delta(q_2, b, b) = \{(q_2, bb)\}$, $\delta(q_2, c, b) = \{(q_3, \lambda)\}$, $\delta(q_3, c, b) = \{(q_3, \lambda)\}$, $\delta(q_3, \lambda, z) = \{(q_4, \lambda)\}$.

Then M accepts L. It does not accept the empty string. If it initially encounters a a (if any), it pushes an a onto the stack. Then it pushes an a onto the stack for each a seen (if any). Then it pops an a for every b seen, if any. Once the stack is empty and it sees a b or if it initially encountered a b, it pushes a b on to the stack. It then pops one b for each c seen guaranteeing that $n_a(w) + n_c(w) = n_b(w)$ and they are in the right order and $n \geq 0, m \geq 1$. Finally it accepts if the input is done and there are no more bs on the stack.

(e) $L = \{a^3 b^n c^n : n \geq 0\}$.

We will give the machine definition. It is recommended you draw this out.

Define $M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \Sigma, \{b, z\}, \delta, q_0, z, \{q_3, q_6\})$ with $\delta(q_0, a, z) = \{(q_1, z)\}$, $\delta(q_1, a, z) = \{(q_2, z)\}$, $\delta(q_2, a, z) = \{(q_3, z)\}$, $\delta(q_3, b, z) = \{(q_4, bz)\}$, $\delta(q_4, b, b) = \{(q_4, bb)\}$, $\delta(q_4, c, b) = \{(q_5, \lambda)\}$, $\delta(q_5, c, b) = \{(q_5, \lambda)\}$, $\delta(q_5, \lambda, z) = \{(q_6, \lambda)\}$.

Then M accepts L since the first three states ensure that the string starts with three as. Then if nothing else follows the string is accepted. For every b seen it pushes a b onto the stack. Then it pops one b for each c seen guaranteeing that the number of bs and cs match and they are in the right order. Finally it accepts if the input is done and there are no more bs on the stack.

(f) $L = \{a^n b^m : n \geq m \geq 3n\}$.

We will give the machine definition. It is recommended you draw this out.

Define $M = (\{q_0, q_1, q_2\}, \Sigma, \{a, b, z\}, \delta, q_0, z, \{q_2\})$ with $\delta(q_0, a, z) = \{(q_0, az)\}$, $\delta(q_0, a, z) = \{(q_0, aaz)\}$, $\delta(q_0, a, z) = \{(q_0, aaaz)\}$, $\delta(q_0, a, a) = \{(q_0, aa)\}$, $\delta(q_0, a, a) = \{(q_0, aaa)\}$, $\delta(q_0, a, a) = \{(q_0, aaaa)\}$, $\delta(q_0, b, a) = \{(q_1, \lambda)\}$, $\delta(q_1, b, a) = \{(q_1, \lambda)\}$, $\delta(q_1, \lambda, z) = \{(q_2, z)\}$.

Then M accepts L. It accepts empty string. The state q_0 keeps track of the n as encountered in the first part of the string and provides a chance for state q_1 to match the bs to anywhere between n and $3n$. It proceeds to state q_2 from state q_1 if the number of bs encountered is anywhere between n and $3n$.

(g) $L = \{w : n_a(w) = n_b(w) + 1\}$.

We will give the machine definition. It is recommended you draw this out.

Define $M = (\{q_0, q_1, q_2\}, \Sigma, \{a, b, z\}, \delta, q_0, z, \{q_2\})$ with $\delta(q_0, \lambda, z) = \{(q_1, az)\}$, $\delta(q_1, a, z) = \{(q_1, az)\}$, $\delta(q_1, a, a) = \{(q_1, aa)\}$, $\delta(q_1, a, b) = \{(q_1, \lambda)\}$, $\delta(q_1, b, z) = \{(q_1, bz)\}$, $\delta(q_1, b, a) = \{(q_1, \lambda)\}$, $\delta(q_1, b, b) = \{(q_1, bb)\}$, $\delta(q_1, \lambda, z) = \{(q_2, \lambda)\}$, $\delta(q_1, c, z) = \{(q_1, c)\}$, $\delta(q_1, c, a) = \{(q_a, a)\}$, $\delta(q_1, c, b) = \{(q_1, b)\}$.

Then M accepts L since it starts with one extra a on the stack and then uses the machine given in Example 7.4 to tell if the number of as and bs are the same. If they are and we started with one more a on the stack then the condition of L is met. Additionally, if it sees a c, it just reads it and does not modify the stack.

(h) $L = \{w : n_a(w) = 2n_b(w)\}$**.**

We will give the machine definition. It is recommended you draw this out.

Define $M = (\{q_0, q_1, q_2\}, \Sigma, \{a, b, z\}, \delta, q_0, z, \{q_2\})$ with $\delta(q_0, \lambda, z) = \{(q_2, z)\}$, $\delta(q_0, c, z) = \{(q_0, z)\}$, $\delta(q_0, c, a) = \{(q_0, a)\}$, $\delta(q_0, c, b) = \{(q_0, b)\}$, $\delta(q_0, a, z) = \{(q_0, az)\}$, $\delta(q_0, b, z) = \{(q_0, bbz)\}$, $\delta(q_0, a, a) = \{(q_0, aa\}$, $\delta(q_0, b, b) = \{(q_0, bbb)\}$, $\delta(q_0, a, b) = \{(q_0, \lambda)\}$, $\delta(q_0, b, a) = \{(q_0, b)\}$, $\delta(q_0, b, a) = \{(q_1, \lambda)\}$, $\delta(q_1, \lambda, a) = \{(q_0, \lambda)\}$.

Then M accepts L. It accepts empty string. If it sees a c, it just reads it and does not modify the stack. If it sees a a while the stack is empty or the top of stack has a, it pushes a a on to the stack. If it sees a b while the stack is empty or the top of stack has b, it pushes two bs to the stack. If it sees a a while top of stack has b, it pops b off the stack. If it encounters a b while top of stack has a, it could take two paths dependent on whether it is the last b it is encountering or not. If it is not the last b, it replaces the stack with a b. If it is the last b, it goes to state q_1 and pops off a from the stack. It then pops off another a from the stack (if any) and returns to state q_0. Pushing two bs while encountering each b and one a for each input a, and popping off symbols in the aforementioned manner ensures M accepts strings with $n_a(w) = 2n_b(w)$.

(i) $L = \{w : n_a(w) + n_b(w) = n_c(w)\}$**.**

We will give the machine definition. It is recommended you draw this out.

Define $M = (\{q_0, q_1\}, \Sigma, \{a, c, z\}, \delta, q_0, z, \{q_1\})$ with $\delta(q_0, a, z) = \{(q_0, az)\}$, $\delta(q_0, a, a) = \{(q_0, aa)\}$, $\delta(q_0, a, c) = \{(q_0, \lambda)\}$, $\delta(q_0, b, z) = \{(q_0, az)\}$, $\delta(q_0, b, a) = \{(q_0, aa)\}$, $\delta(q_0, b, c) = \{(q_0, \lambda)\}$, $\delta(q_0, c, z) = \{(q_0, cz)\}$, $\delta(q_0, c, a) = \{(q_0, \lambda)\}$, $\delta(q_0, c, c) = \{(q_0, cc)\}$, $\delta(q_0, \lambda, z) = \{(q_1, \lambda)\}$.

Then M accepts L since it keeps count of how many as and bs are read with the stack symbol a and how many cs with the symbol c. For each a and b seen it will cancel a c if that is on the top of the stack or add an a. Similarly, for each c seen, it will add a c to the stack or cancel an a if it is on top of the stack. Finally, we only accept if the number of cs is the same as the number of as and bs combined, that is they all cancel each other on the stack.

(j) $L = \{w : 2n_a(w) \leq n_b(w) \leq 3n_a(w)\}$**.**

We will give the machine definition. It is recommended you draw this out.

Define $M = (\{q_0, q_1, q_2, q_3\}, \Sigma, \{a, b, z\}, \delta, q_0, z, \{q_3\})$ with $\delta(q_0, a, z) = \{(q_0, aaz)\}$, $\delta(q_0, a, z) = \{(q_0, aaaz)\}$, $\delta(q_0, a, a) = \{(q_0, aaa\}$, $\delta(q_0, a, a) = \{(q_0, aaaa)\}$, $\delta(q_0, b, z) = \{(q_0, bz)\}$, $\delta(q_0, b, b) = \{(q_0, bb)\}$, $\delta(q_0, b, a) = \{(q_0, \lambda)\}$, $\delta(q_0, a, b) = \{(q_1, \lambda)\}$, $\delta(q_1, \lambda, b) = \{(q_0, \lambda)\}$, $\delta(q_1, \lambda, z) = \{(q_0, az)\}$, $\delta(q_1, \lambda, z) = \{(q_0, aaz)\}$, $\delta(q_1, \lambda, b) = \{(q_2, \lambda)\}$, $\delta(q_2, \lambda, b) = \{(q_0, \lambda)\}$, $\delta(q_2, \lambda, z) = \{(q_0, az)\}$, $\delta(q_0, c, z) = \{(q_0, c)\}$, $\delta(q_0, c, a) = \{(q_0, a)\}$, $\delta(q_0, c, b) = \{(q_0, b)\}$.

Then M accepts L. It accepts empty string. If it sees a c, it just reads it and does not modify the stack. The state q_0 keeps track of as and bs, and provides a chance for states q_0, q_1 and q_2 to match the as when b is encountered with a on the top of the stack by popping the stack, while tracking the condition $2n_a(w) \leq n_b(w) \leq 3n_a(w)$. q_1 and q_2 also deal with the cases when the bottom of the stack is reached without satisfying the condition but there are still input alphabets to be parsed. Finally, we only accept if the condition $2n_a(w) \leq n_b(w) \leq 3n_a(w)$ is satisfied.

(k) $L = \{w : n_a(w) < n_b(w)\}$**.**

We will give the machine definition. It is recommended you draw this out.

Define $M = (\{q_0, q_1\}, \Sigma, \{a, b, z\}, \delta, q_0, z, \{q_1\})$ with $\delta(q_0, a, z) = \{(q_0, az)\}$, $\delta(q_0, a, a) = \{(q_0, aa)\}$, $\delta(q_0, a, b) = \{(q_0, \lambda)\}$, $\delta(q_0, b, z) = \{(q_0, bz)\}$, $\delta(q_0, b, a) = \{(q_0, \lambda)\}$, $\delta(q_0, b, b) = \{(q_0, bb)\}$, $\delta(q_0, \lambda, b) = \{(q_1, \lambda)\}$, $\delta(q_0, c, z) = \{(q_0, c)\}$, $\delta(q_0, c, a) = \{(q_0, a)\}$, $\delta(q_0, c, b) = \{(q_0, b)\}$.

Then M accepts L since it follows Example 7.4 to tell if the number of as and bs are the same, but now it only accepts if there is a b on top of the stack. The symbol on top of the stack is always the symbol we have read more of thus far while processing the string. Therefore, if the input is done and there is a b at the top of the stack then there are more bs than as. Additionally, if it sees a c, it just reads it and does not modify the stack.

Problem 5: Linz 7.2.15

Find a context-free grammar that generates the language accepted by the npda $M = (\{q_0, q_1\}, \{a, b\}, \{A, z\}, \delta, q_0, z, \{q_1\})$, with transitions

$$\begin{aligned} \delta(q_0, a, z) &= \{(q_0, Az)\}, \\ \delta(q_0, b, A) &= \{(q_0, AA)\}, \\ \delta(q_0, a, A) &= \{(q_1, \lambda)\}. \end{aligned}$$

This satisfies condition 2 but not 1 (conditions in section 7.2 *Context−Free Grammars for Pushdown Automata*). In order to satisfy condition 1, we add two more rules to the CFG and make q_2 the only accepting state (q_1 now empties the stack).

$$\begin{aligned} \delta(q_0, a, z) &= \{(q_0, Az)\}, \\ \delta(q_0, b, A) &= \{(q_0, AA)\}, \\ \delta(q_0, a, A) &= \{(q_1, \lambda)\} \\ \delta(q_1, \lambda, A) &= \{(q_1, \lambda)\} \\ \delta(q_1, \lambda, z) &= \{(q_2, \lambda)\}. \end{aligned}$$

The last three transitions are of the form (7.5). So they yield the corresponding productions

$$\begin{aligned} (q_0 A q_1) &\to a, \\ (q_1 A q_1) &\to \lambda, \\ (q_1 z q_2) &\to \lambda. \end{aligned}$$

From the first two transitions we get the set of productions

$$\begin{aligned} (q_0 z q_0) &\to a(q_0 A q_0)(q_0 z q_0) | a(q_0 A q_1)(q_1 z q_0) | a(q_0 A q_2)(q_2 z q_0), \\ (q_0 z q_1) &\to a(q_0 A q_0)(q_0 z q_1) | a(q_0 A q_1)(q_1 z q_1) | a(q_0 A q_2)(q_2 z q_1), \\ (q_0 z q_2) &\to a(q_0 A q_0)(q_0 z q_2) | a(q_0 A q_1)(q_1 z q_2) | a(q_0 A q_2)(q_2 z q_2), \\ (q_0 A q_0) &\to b(q_0 A a_0)(q_0 A q_0) | b(q_0 A q_1)(q_1 A q_0) | b(q_0 A q_2)(q_2 A q_0), \\ (q_0 A q_1) &\to b(q_0 A a_0)(q_0 A q_1) | b(q_0 A q_1)(q_1 A q_1) | b(q_0 A q_2)(q_2 A q_1), \\ (q_0 A q_2) &\to b(q_0 A a_0)(q_0 A q_2) | b(q_0 A q_1)(q_1 A q_2) | b(q_0 A q_2)(q_2 A q_2). \end{aligned}$$

$(q_1 z q_0)$, $(q_1 A q_0)$, $(q_1 z q_1)$, $(q_1 A q_2)$, $(q_2 z q_0)$, $(q_2 z q_1)$, $(q_2 z q_2)$, $(q_2 A q_0)$, $(q_2 A q_1)$ and $(q_2 A q_2)$ do not occur at the left side of any production, so they are useless and can be eliminated. Also noting that the production $(q_0 A q_0) \to b(q_0 A q_0)(q_0 A q_0)$ has no way of terminating and is useless we can

eliminate it. Then there is no production with (q_0Aq_0) on the left hand side so we eliminate any production containing it on the right hand side. We are then left with the folloing shorter grammar

$$\begin{array}{lcl}(q_0Aq_1) & \rightarrow & a \\ (q_1Aq_1) & \rightarrow & \lambda, \\ (q_1zq_2) & \rightarrow & \lambda, \\ (q_0zq_2) & \rightarrow & a(q_0Aq_1)(q_1zq_2), \\ (q_0Aq_1) & \rightarrow & b(q_0Aq_1)(q_1Aq_1).\end{array}$$

with (q_0zq_2) as the starting variable.

If we did one more step and also eliminated λ-productions and rearranged the productions we are left with the following much shorter grammar

$$\begin{array}{lcl}(q_0zq_2) & \rightarrow & a(q_0Aq_1), \\ (q_0Aq_1) & \rightarrow & b(q_0Aq_1)|a.\end{array}$$