Lecture L&M 6

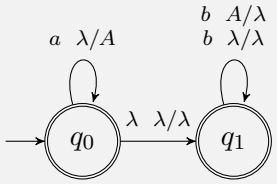
Pushdown Automata

6.1 Pushdown automata (PDA)

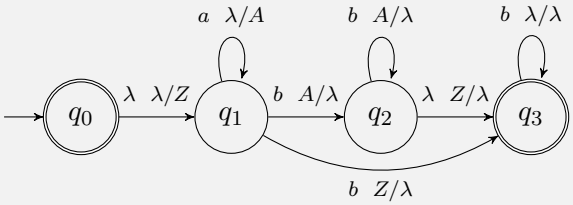
For each of the following languages L, give a PDA that accepts the language.

A. $L = \{a^i b^j \mid 0 \leq i \leq j\}$
(Give a small nondeterministic PDA and a deterministic PDA.)

A straightforward variation on automata in the book and the slides.

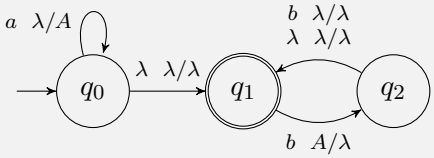


The deterministic variant is more interesting: we only generate extra b if the bottom of the stack is reached. But we must discover it first!



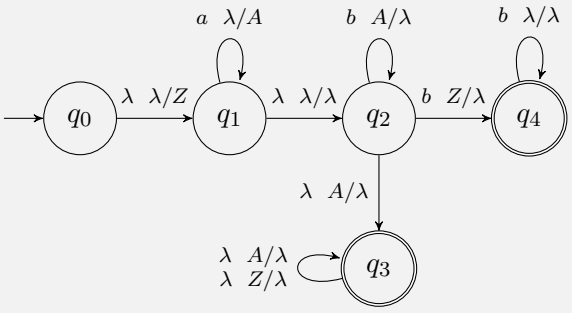
B. $L = \{a^i b^j \mid 0 \leq i \leq j \leq 2i\}$

Somewhat more difficult than a); for example, every a can push two symbols: an optional A and a mandatory B. But there are many more possibilities.



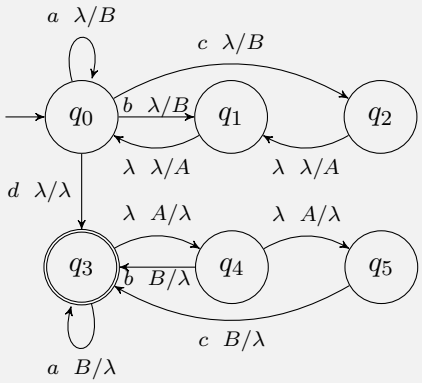
C. $L = \{a^i b^j \mid i \neq j\}$

To see if there are more b than a, we must be able to test for the bottom of the stack; so first add a special symbol. The stack always contains $A^n Z$ (except in the start and final state), where $n = \#a - \#b$.

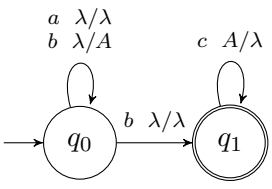


D. Construct a PDA with $|\Gamma| \leq 2$ (i.e. there are only two different stack symbols) that accepts the language $L = \{wdw^R \mid w \in \{a, b, c\}^*\}$ (with alphabet $\Sigma = \{a, b, c, d\}$).

Because there are only 2 stack symbols, a, b, and c must be "coded" in a different way; for example as A^nB where B denotes a "separation symbol" and $n \geq 0$ denotes the symbol (in $\{a, b, c\}$).



E. Consider the following pushdown automaton M:



(a) What are $\mathcal{L}(M)$, $\mathcal{L}_F(M)$ and $\mathcal{L}_E(M)$?

(b) Construct an automaton M_1 such that $\mathcal{L}_F(M_1) = \mathcal{L}_E(M_1) = \mathcal{L}(M)$

(c) Construct an automaton M_2 such that $\mathcal{L}(M_2) = \mathcal{L}_F(M)$

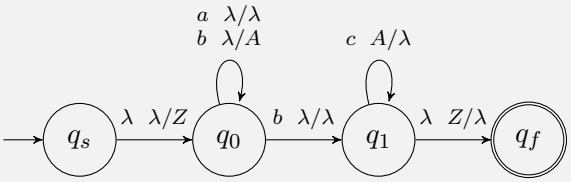
(d) Construct an automaton M_3 such that $\mathcal{L}(M_3) = \mathcal{L}_E(M)$

a. We use $\#_a w$ for the number of a symbols in w.

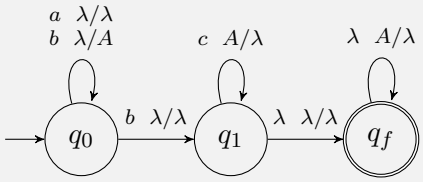
$$\begin{aligned} L(M) &= \{vbc^i \mid v \in \{a,b\}^*, i = \#_b v\} \\ L_F(M) &= \{vbc^i \mid v \in \{a,b\}^*, i \leq \#_b v\} \\ L_E(M) &= L(M) \cup a^+ \end{aligned}$$

(Note that $\lambda \notin L_E(M)$, since the calculations in such an automaton must always take at least one step.)

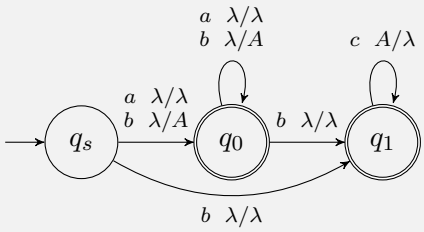
b. A matter of an artificial "bottom" symbol, and new start and final states:



c. From the final state, the stack can be emptied at all times:



d. All states become final states, but we do need a new start state:

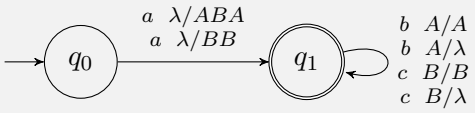


6.2 Conversion from CFG to PDA

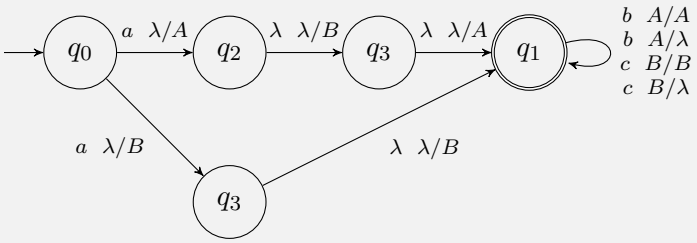
F. Use the technique from the lecture to construct a PDA that accepts the language of the following grammar (which is in Greibach normal form):

$$\begin{aligned} S &\to aABA \mid aBB \\ A &\to bA \mid b \\ B &\to cB \mid c \end{aligned}$$

The (extended) PDA is:



Then the corresponding "normal" PDA is:



6.3 Conversion from CFG to GNF

Using the procedure from the lecture, transform the following grammars given in Chomsky normal form to Greibach normal form.

G.

$$S \to BB$$
$$A \to AA \mid a$$
$$B \to AA \mid BA \mid b$$

Step 1: *Stratification*. We set an order for the variables, say $S < A < B$, and following that order we remove all direct left recursions and unstratified dependencies.

- S needs no change.
- A has left recursion in the rule $A \to AA$, which we deal with via the helper variable Z_A:

$$S \to BB$$
$$A \to aZ_A \mid a$$
$$B \to AA \mid BA \mid b$$
$$Z_A \to AZ_A \mid A$$

- B has the (single) unstratified rule $B \to AA$, since $A < B$ and A is the first variable on the rhs of that rule*. We resolve it inlining the derivations of A, which as per the previous step

will contain no left recursion (and hence no A occurring as first variable):

$$\begin{aligned} S &\to BB \\ A &\to aZ_A \mid a \\ B &\to aZ_AA \mid aA \mid BA \mid b \\ Z_A &\to AZ_A \mid A \end{aligned}$$

Finally we remove the left recursion in B (rule $B \to BA$) via the helper variable Z_B:

$$\begin{aligned} S &\to BB \\ A &\to aZ_A \mid a \\ B &\to aZ_AAZ_B \mid aAZ_B \mid bZ_B \mid aZ_AA \mid aA \mid b \\ Z_A &\to AZ_A \mid A \\ Z_B &\to AZ_B \mid A \end{aligned}$$

Note: Step 1 finalises after revising all *original rules* of the grammar, here for variables S, A, B.

Step 2: *Inline higher-order variables*. After step 1, all rules for the (original) variable with highest order start with a terminal. Moving bottom-up we inline these rules whenever they ocurr as first variables.

- For B into A there is nothing to do, as no rule from A produces a B
- For B into S we get:

$$\begin{aligned} S &\to aZ_AAZ_BB \mid aAZ_BB \mid bZ_BB \mid aZ_AAB \mid aAB \mid bB \\ A &\to aZ_A \mid a \\ B &\to aZ_AAZ_B \mid aAZ_B \mid bZ_B \mid aZ_AA \mid aA \mid b \\ Z_A &\to AZ_A \mid A \\ Z_B &\to AZ_B \mid A \end{aligned}$$

- For A into S there is nothing to do, as no rule from S has an A as first variable.

Now we do the same for the new rules introduced during step 1, which yields:

$$\begin{aligned} S &\to aZ_AAZ_BB \mid aAZ_BB \mid bZ_BB \mid aZ_AAB \mid aAB \mid bB \\ A &\to aZ_A \mid a \\ B &\to aZ_AAZ_B \mid aAZ_B \mid bZ_B \mid aZ_AA \mid aA \mid b \\ Z_A &\to aZ_AZ_A \mid aZ_A \mid a \\ Z_B &\to aZ_AZ_B \mid aZ_B \mid aZ_A \mid a \end{aligned}$$

Step 3: *Special variables for non-initial terminals*. After step 2, all rules start with a terminal. However, some terminals could still appear in the middle of a rule, i.e. not at the start of a rhs. We replace these for special variables which produce only the corresponding terminal.
Notice that in this case, all terminals appear only at the start of all rules *(extra: why?)*. Hence we're done, and the final equivalent grammar in GNF is:

$$\begin{aligned} S &\to aZ_AAZ_BB \mid aAZ_BB \mid bZ_BB \mid aZ_AAB \mid aAB \mid bB \\ A &\to aZ_A \mid a \\ B &\to aZ_AAZ_B \mid aAZ_B \mid bZ_B \mid aZ_AA \mid aA \mid b \\ Z_A &\to aZ_AZ_A \mid aZ_A \mid a \\ Z_B &\to aZ_AZ_B \mid aZ_B \mid aZ_A \mid a \end{aligned}$$

* Recall that only the first occurring variable of each derivation is relevant for stratification, cf. § 4.8 p. 132 in the book "Languages and Machines" by Thomas A. Sudkamp, 3rd edition.

H. *(extra)*

$$\begin{aligned} S &\to AB \mid BC \\ A &\to AB \mid a \\ B &\to AA \mid CB \mid b \\ C &\to a \mid b \end{aligned}$$

Step 1 : *Stratification*. Variables order will be $S < A < B < C$. Removing left recursion in A yields:

$$\begin{aligned} S &\to AB \mid BC \\ A &\to aZ_A \mid a \\ B &\to AA \mid CB \mid b \\ C &\to a \mid b \\ Z_A &\to BZ_A \mid B \end{aligned}$$

Inlining initial A's into the rules of B we get:

$$\begin{aligned} S &\to AB \mid BC \\ A &\to aZ_A \mid a \\ B &\to aZ_AA \mid aA \mid CB \mid b \\ C &\to a \mid b \\ Z_A &\to BZ_A \mid B \end{aligned}$$

Step 2: *Inline higher-order variables*. From C into B we get:

$$\begin{aligned} S &\to AB \mid BC \\ A &\to aZ_A \mid a \\ B &\to aZ_AA \mid aA \mid aB \mid bB \mid b \\ C &\to a \mid b \\ Z_A &\to BZ_A \mid B \end{aligned}$$

From A and B into S (it can be done in any order... why?) we get:

$$\begin{aligned} S &\to aZ_AB \mid aB \mid aZ_AAC \mid aAC \mid aBC \mid bBC \mid bC \\ A &\to aZ_A \mid a \\ B &\to aZ_AA \mid aA \mid aB \mid bB \mid b \\ C &\to a \mid b \\ Z_A &\to BZ_A \mid B \end{aligned}$$

From original rules into new rules of step 1 (i.e. from B into Z_A) we get:

$$\begin{aligned} S &\to aZ_AB \mid aB \mid aZ_AAC \mid aAC \mid aBC \mid bBC \mid bC \\ A &\to aZ_A \mid a \\ B &\to aZ_AA \mid aA \mid aB \mid bB \mid b \\ C &\to a \mid b \\ Z_A &\to aZ_AAZ_A \mid aAZ_A \mid aBZ_A \mid bBZ_A \mid bZ_A \mid aZ_AA \mid aA \mid aB \mid bB \mid b \end{aligned}$$

Step 3: *Special variables for non-initial terminals*. Since all terminals appear only at the start of all rules (why then again?), we're done.

6.4 Operations on context-free languages

I. *(extra)* Prove that the set of context-free languages is closed under reversal.

Call the original language L and let $G = \langle V, \Sigma, P, S\rangle$ be a context-free grammar such that $\mathcal{L}(G) = L$. Now define $G' = \langle V, \Sigma, P', S\rangle$ with $P' = \{A \to w^R \mid (A \to w) \in P\}$. We prove that $\mathcal{L}(G') = \{w^R \mid w \in L\}$. Because this construction is symmetrical, i.e., applying the same construction to G' results in G again, it is sufficient to prove that $\mathcal{L}(G') \supseteq \{w^R \mid w \in L\}$

For the proof, we strengthen the proposition to: $S \Rightarrow_G^* w$ implies $S \Rightarrow_{G'}^* w^R$ for arbitrary sentential forms $w \in (V \cup \Sigma)^*$. This can be proven by induction over the length of the derivation:
Basis: If the length of the derivation is 0, then $w = S = w^R$.
Induction step: Assume that $S \Rightarrow_G^* v \Rightarrow_G w$ and $S \Rightarrow_{G'}^* v^R$. Then $v = v_1Av_2$ with $(A \to v_3) \in P$ and $w = v_1v_3v_2$; so $v^R = v_2^RAv_1^R$ with $(A \to v_3^R) \in P'$. But then also $v_R \Rightarrow_{G'} v_2^Rv_3^Rv_1^R = w^R$.