Pushdown Automata Exercises

We start with standard problems on building pda for a given language, ending with more challenging problems. . . . to be continued . . .

1. Construct pushdown automata for the following languages. Acceptance either by empty stack or by final state.

   ♣ (a) $\{ a^n b^m a^n \mid m, n \in \mathbb{N} \}$

   (b) $\{ a^n b^m c^m \mid m, n \in \mathbb{N} \}$

   (c) $\{a^i b^j c^k \mid i, j, k \in \mathbb{N}, i > j\}$

   (d) $\{a^i b^j c^k \mid i, j, k \in \mathbb{N}, i + j = k\}$

   ♣ (e) $\{a^i b^j c^k \mid i, j, k \in \mathbb{N}, i + k = j\}$

   ♣ (f) $\{ a^n b^m \mid n \le m \le 2n \}$

   (g) $\mathrm{PAL} = \{w \in \{a, b\}^* \mid \mathrm{mir}(w) = w\}$

   (h) $\{w_1 c w_2 c \cdots c w_k c x \mid x, w_1, \ldots, w_k \in \{a, b\}^*,\ k \in \mathbb{N},\ x = \mathrm{mir}(w_j) \text{ for some } j\ \}$

   ♣ (i) $\{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$, $\#_a(w)$ represents the number of a's in w

   (j) $\{w \in \{a, b\}^* \mid \#_a(w) = 2 \cdot \#_b(w)\}$

2. More languages.

   (a) $\{ w \in \{a, b\}^* \mid w \text{ equals } x\mathrm{mir}(x) \text{ met } x \in \{a, b\}^* \}$

   ♣ (b) $\{ w \in \{a, b\}^* \mid w$ does *not* equal $x\mathrm{mir}(x)$ for some $x \in \{a, b\}^* \}$

   ♣ (c) $\{ w \in \{a, b, c\}^* \mid w$ does not equal xcx for some $x \in \{a, b\}^* \}$

   ♣ (d) $\{ w \in \{a, b\}^* \mid w$ does not equal xx for some $x \in \{a, b\}^* \}$

3. Construct a pda with final state acceptance for the language

   $B = \{ \mathrm{bin}(i)\$\mathrm{mir}(\mathrm{bin}(i + 1)) \mid i \ge 0 \} \subseteq \{0, 1, \$\}^*$

   ♣ Here is $\mathrm{bin}(i) \in \{0, 1\}^*$ the binary representation (without leading zero's) of the number i. Eg. $\mathrm{bin}(11) = 1011$ and $\mathrm{mir}(\mathrm{bin}(12)) = 0011$.

4. Give a one-state pda for the 'coin-change' language from the chapter:

$$L_{ex} = \{\ x \mathtt{=} y \mid x \in \{\mathtt{1}, \mathtt{2}\}^*, y \in \{\mathtt{5}\}^*, |x|_{\mathtt{1}} + 2 \cdot |x|_{\mathtt{2}} = 5 \cdot |y|_{\mathtt{5}}\ \},$$

Pda and cfg.

5. pushdown automaton $\mathcal{A}$ is specified by

   $\mathcal{A} = (\{q_0, q_1\}, \{a, b\}, \{Z, X\}, \delta, q_{in}, Z, \emptyset)$, where δ contains the following transitions:

   $(q_0, a, Z) \mapsto (q_0, \lambda)$, $(q_0, a, Z) \mapsto (q_0, XZ_{in})$,

   $(q_0, a, X) \mapsto (q_0, XX)$, $(q_0, b, X) \mapsto (q_1, \lambda)$,

   $(q_1, b, X) \mapsto (q_1, \lambda)$, $(q_1, a, Z) \mapsto (q_0, Z)$.

   ♣ Determine a (reduced) context-free grammar G for the empty stack language of $\mathcal{A}$, i.e., $L(G) = L_e(\mathcal{A})$.

6. Given two cf languages K and L, there is a pda $\mathcal{A}$ such that $L_f(\mathcal{A}) = K$ and $L_e(\mathcal{A}) = L$ (where the subscripts f and e refer to the final state and empty stack acceptance respectively).

Deterministic automata.

7. Consider the languages of Exercise 1. Which of these are accepted by deterministic automata? Give an automaton where possible. What is the acceptance type?

8. Show formally that the language $\{ a^n b^m \mid n \leq m \leq 2n \}$ is not deterministically context-free.

   > Unfortunately, when applying the operation pre from the chapter, we obtain $\{ a^n b^m \mid n < m \leq 2n \}$ which still is context-free. We need a different closure property of the deterministic context-free languages to tackle this problem.

9. Show formally that the language $\{ w\text{mir}(w) \mid w \in \{a, b\}^* \}$ is not deterministically context-free.

10. Show that $\{ a^m b^n c^p \mid \ m < n \text{ or } n < p\}$ is not deterministically context-free.

11. Let $w_1, \dots, w_n$ be a sequence of words over the alphabet Δ en let $\Sigma = \{1, 2, \dots, n\}$ be an alphabet reprsenting numbers. A context-free language related to the *post correspondence problem* is generated by the productions $S \to w_i S i$ $(i = 1, \dots, n)$, $S \to \lambda$.

    Argue that also the complement of this language is context-free. (Hint: this is a question on deterministic languages.)

    From this infer that '$L(G) = \Sigma^*$' is undecidable for a cfg G over alphabet Σ.

12. Let # be a symbol not in the alphabet Σ.
    Give a proof that $K \in \mathsf{DPD}_f \iff K \cdot \# \in \mathsf{DPD}_e$

    Also $K \in \mathsf{DPD}_f \iff K \cdot \# \in \mathsf{DPD}_f$ $\quad$ (DPD_f is denoted DCF in the chapter)

Closure Properties.

13. ♣ For each of the following operations, construct a finite state transducer that performs that operation (for a fixed regular R).

    (a) intersection with regular language R

    (b) right quotient with regular language R:
    for $K, R \in \Sigma^*$, $K/R = \{ x \in \Sigma^* \mid xy \in K \text{ for some } y \in R \}$

    (c) concatenation with regular language R

14. ♣ Write each of the operations of the previous exercise as a combination of *full trio* operations, i.e., as a composition of morphism, inverse morphism, and intersection with regular languages.

15. The family CF is closed under inverse morpisms. This can be shown using pda, either directly as in the chapter (Lemma 7) or via finite state transductions, as in the notes (Lemma 11). Verify that this closure property also holds in the deterministic case.

Extending the model.

16. A two-way pushdown automaton may move on its input tape in two directions. As usual for two-way automata we assume that the begin and end of the input tape is marked by special symbols. In this way the automaton can recognize those positions.

    Describe a two-way pda for each of the following languages.

    (a) $\{ a^n b^n c^n \mid n \in \mathbb{N} \}$ (easy)

    (b) $\{ ww \mid w \in \{a,b\}^* \}$ (nice puzzle)

    (c) $\{ v\,c\,uvw \mid u,v,w \in \{a,b\}^* \}$ (pattern matching; clever trick)

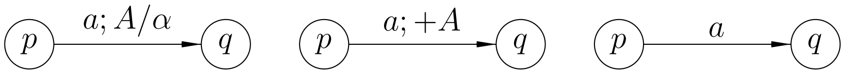
17. In the case of finite state automata, the two-way model is equivalent to the usual one-way automaton. Find a proof of this result.

18. Stack automata are pda that may inspect their stack. The chapter states: “stack automata that do not read input during inspection of the stack are equivalent to pda’s”. Verify this fact.

Research.

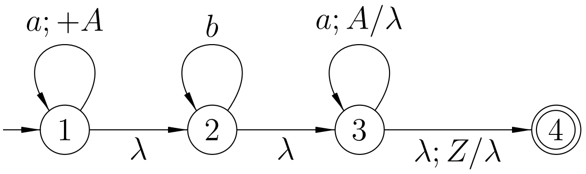
19. Investigate the notion of ‘recursive finita state automata’. How do you formalize this notion. How do you define determinism here? What is the relation to (deterministic) pushdown automata? ♣

Convention. When we use a diagram to depict a pda, we use our own private conventions. A single instruction (p, a, A, q, α) is given as an arc from state p to state q labelled by $a; A/\alpha$. Label $a; +A$ represents a set of instructions pushing A on top of the stack (p, a, X, q, AX) for all $X \in \Gamma$, while label a represents the set (p, a, X, q, X) of instructions that 'ignore' the stack.



Solutions

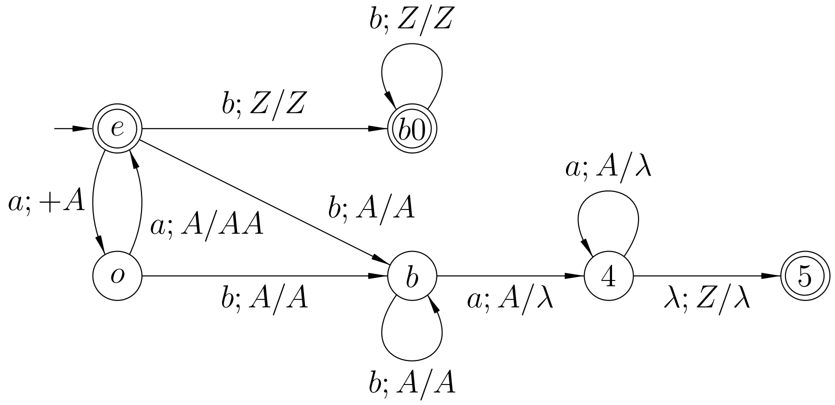
1a The pda is depicted by the following diagram. Formally, it consists of the following components: state set $Q = \{1, 2, 3, 4\}$, alphabet $\Sigma = \{a, b\}$, stack alphabet $\Gamma = \{Z, A\}$, initial state 1, final state 4, initial stack symbol Z, and instructions $(1, a, X, 1, AX)$, $(1, \lambda, X, 2, X)$, $(2, b, X, 2, X)$, $(2, \lambda, X, 3, X)$ $(3, a, A, 3, \lambda)$, $(3, \lambda, Z, 3, \lambda)$, for all $X \in \Gamma$. Acceptance either by final state or by empty stack.



Alternatively, we build a cfg: $S \to aSA$, $S \to T$, $T \to bT$, $T \to \lambda$, $A \to a$.

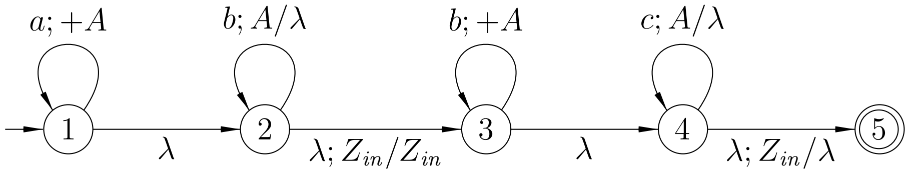
This translates into the single state pda with instructions (p, a, S, p, SA), (p, λ, S, p, T), (p, b, T, p, T), $(p, \lambda, T, p, \lambda)$, and (p, a, A, p, λ). Initial stack symbol S, acceptance by empty stack.

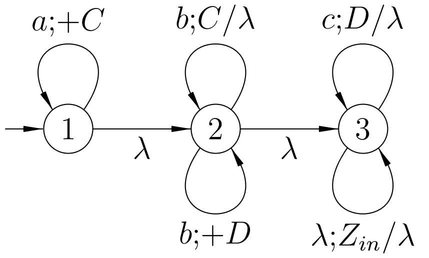
Determinism. The automata above are not deterministic. To accept deterministically we have to realize strings without b's (including λ) are to be accepted too, without recognizing (guessing) the middle of the string. However, strings without b from the regular subset $(aa)^*$. Likewise there is a 'special' subset b^*.

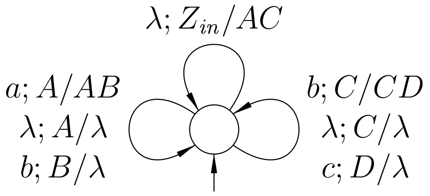


Study such a diagram with care. Is it deterministic? Are the special cases (like the empty string) handled properly? How are the good strings accepted, but also how are the bad ones rejected?

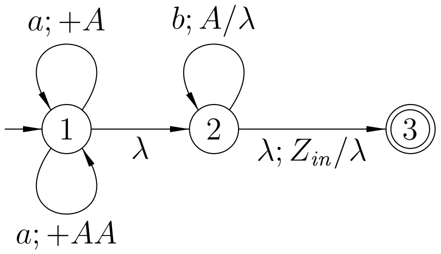
1e We give three different pda accepting this language. All variants use the fact that the language can be writen as $\{ a^i b^i b^j c^j \mid i, j \in \mathbb{N} \}$.







1f For each a we put nondeterministically one or two symbols onto the stack. Each b removes one symbol.



1i Push for a's and pop for b's, or more precisely put the number $\#_a(v) - \#_b(v)$ onto the stack, where v is the prefix of the input read. Note that this number can become negative. We can either use two different pushdown symbols, or we can use the states to store the sign.

Determinism. Before each letter we test whether the contents of the stack represents zero. In that case we pass through a final state to signal possible acceptance (in case the end of the input was reached).

Initial state 0, final state f, initial pushdown symbol Z (for 'zero'). Instructions: $(0, \lambda, Z, f, Z)$, $(0, \lambda, P, 1, P)$, $(0, \lambda, N, 1, N)$, $(f, \lambda, Z, 1, Z)$, (testing zero) $(1, a, P, 0, PP)$, $(1, a, N, 0, \lambda)$, $(1, b, P, 0, \lambda)$, $(1, b, N, 0, NN)$ (counting).

2b When w is not of the form $x\text{mir}(x)$ then either its length is odd, or there is a k such that the k-th symbol from the word differs from the k-th symbol from the rear. De language of strings of odd length is regular, and hence accepted by a pda. Since pda languages are closed under union it suffices to construct a pda for the language

$$\{ x\sigma_1 y\sigma_2 z \mid x, y, z \in \{a, b\}^*, |x| = |z|, \sigma_1, \sigma_2 \in \{a, b\}, \sigma_1 \neq \sigma_2 \}.$$

This automaton goes through the following steps: (1) read the input, stacking for each letter read the symbol X under the initial symbol Z. (2) read a letter, storing its value on top of the stack, (3) ignore a random number of symbols from the input, (4) read a letter that is different from the one stored on the stack, otherwise block, (5) read input symbols, taking an X from the stack for each letter. This can be achieved using a single state. Acceptance by empty stack (of course).

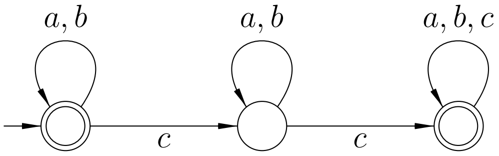
$(p,\sigma,Z) \mapsto (p,ZX) \quad (\sigma \in \{a,b\})$,
$(p,a,Z) \mapsto (p,A), \quad (p,b,Z) \mapsto (p,B)$,
$(p,\sigma,A) \mapsto (p,A), \quad (p,\sigma,B) \mapsto (p,B) \quad (\sigma \in \{a,b\})$,
$(p,a,B) \mapsto (p,\lambda), \quad (p,b,A) \mapsto (p,\lambda)$,
$(p,\sigma,X) \mapsto (p,\lambda) \quad (\sigma \in \{a,b\})$.

In my philosophy this is a context-free grammar in disguise:

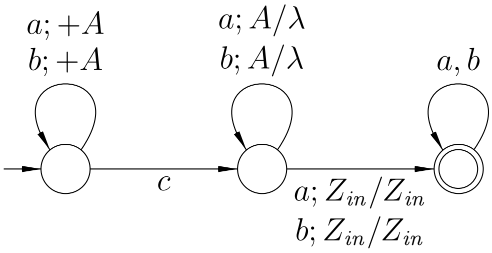
$Z \to \sigma XZ$, $Z \to aA$, $Z \to bB$, $A \to \sigma A$, $B \to \sigma B$, $B \to a$, $A \to b$, $X \to \sigma$. $(\sigma \in \{a,b\})$

2c The language can be written as $L_1 \cup L_2 \cup L_3 \cup L_4 \cup L_5$ where

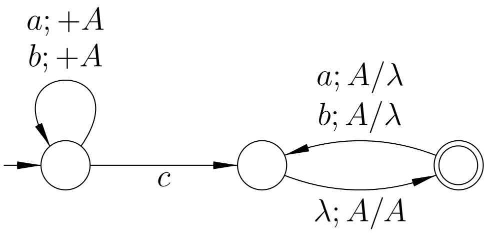
$L_1 = \{\ w \in \{a,b,c\}^* \mid \#_c(w) \neq 1\ \}$, a regular language



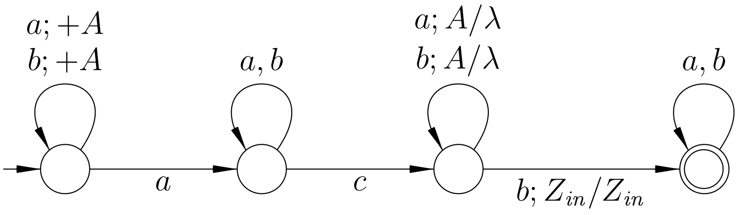
$L_2 = \{\ x_1 c x_2 \mid x_1, x_2 \in \{a,b\}^*, |x_1| < |x_2|\ \}$,



$L_3 = \{\ x_1 c x_2 \mid x_1, x_2 \in \{a,b\}^*, |x_1| > |x_2|\ \}$,



$L_4 = \{\ x_1 a y_1 c x_2 b y_2 \mid x_1, x_2, y_1, y_2 \in \{a,b\}^*, |x_1| = |x_2|\ \}$,



and finally L_5 equals L_4 but with strings of the form $x_1by_1cx_2ay_2$.

The pda that we have constructed can be joined with instructions $(q_{in}, \lambda, Z_{in}) \mapsto (q, Z_{in})$ starting from a new initial state q_{in}.

2d When w does not consist of two copies of the same word, then either w has odd length, or the first half of w does not equal the second half, so there is a k such that the letter on position k differs from the letter at position k after the middle. Unlike the previous exercise we cannot recognize the middle, and we have to guess that point.

Thus we find the language

$$K_e = \{\ x_1\sigma_1y_1x_2\sigma_2y_2 \mid x_1, y_1, x_2, y_2 \in \{a,b\}^*,\\ |x_1| = |x_2|, |y_1| = |y_2|, \sigma_1, \sigma_2 \in \{a,b\}, \sigma_1 \neq \sigma_2\ \}.$$

In this way we cannot recognize the language. We cán if we realize that that it is not necessary to determibe middle, but that it suffices to check that between the σ's are as many letters as before and after: $|y_1x_2| = |x_1| + |y_2|$ replaces the requirement $|x_1| = |x_2|, |y_1| = |y_2|$.

3 $B = B_1 \cup B_2 \cup B_3$ met

$B_1 = \{\ 1x01^n\$0^n1\mathrm{mir}(x)1 \mid x \in \{0,1\}^*, n \geq 0\ \}$,
$B_2 = \{\ 1^n\$0^n1 \mid n \geq 1\ \}$, en
$B_3 = \{\ 0\$1\ \}$.

From this a pda for B can be constructed in a straightforward way.

5 Using the standard construction for obtaining a cfg for a given pda, we find 15 productions: (in all cases: $p, r \in Q$, where $Q = \{q_0, q_1\}$).

$S \to [q_0Zp]$,
$[q_0Zq_0] \to a$, $\quad [q_0Zr] \to a[q_0Xp][pZr]$,
$[q_0Xr] \to a[q_0Xp][pXr]$, $\quad [q_0Xq_1] \to b$,
$[q_1Xq_1] \to b$, $\quad [q_1Zr] \to a[q_0Zr]$.

In a bottom-up way we determine the *productive* nonterminals, those nonterminals that can derive a (terminal) string:
(1) $[q_0Zq_0]$, $[q_0Xq_1]$, $[q_1Xq_1]$, (because of productions $[q_0Zq_0] \to a$, etc.)
(2) S, $[q_1Zq_0]$ (because of $[q_1Zq_0] \to a[q_0Zq_0]$).

Thus, e.g., $[q_0, Z, q_1]$ is not productive. This means that there is no computation of the form $(w, q_0, Z) \vdash_{\mathcal{A}}^* (\lambda, q_1, \lambda)$, from q_0 to q_1 the pda removes Z from the stack, reading some string $w \in \{a,b\}^*$.

Deleting productions containing the remaining nonterminals, we obtain an equivalent grammar:
$S \to [q_0Zq_0]$,
$[q_0Zq_0] \to a$, $\quad [q_0Zq_0] \to a[q_0Xq_1][q_1Zq_0]$,
$[q_0Xq_1] \to a[q_0Xq_1][q_1Xq_1]$, $\quad [q_0Xq_1] \to b$,
$[q_1Xq_1] \to b$, $\quad [q_1Zq_0] \to a[q_0Zq_0]$.

Now we compute the *reachable* nonterminals, i.e., those that are generated starting from the axiom S:
(0) S (1) $[q_{in}Zq_{in}]$ (2) $[q_{in}Xq_1]$, $[q_1Zq_{in}]$ (3) $[q_1Xq_1]$.
In fact, these are all the nonterminals found in the previous step. The grammar we had in the last step is reduced. The (single) production for S can be removed when taking $[q_0Zq_0]$ as axiom.

We can slightly 'optimize' this. Write Z instead of $[q_0Zq_0]$, X instead of $[q_0Xq_1]$, and substitute $[q_1Zq_0] \to a[q_0Zq_0]$ in $[q_0Zq_0] \to a[q_0Xq_1][q_1Zq_0]$, and $[q_1Xq_1] \to b$ in $[q_{in}Xq_1] \to a[q_{in}Xq_1][q_1Xq_1]$. We thus obtain the equivalent grammar:

$Z \to a$, $\quad Z \to aXaZ$,
$X \to aXb$, $\quad X \to b$

Then, it is not difficult to verify that $L(G) = L_e(\mathcal{A}) = (\{\, a^nb^n \mid n \geq 1\}a)^* \cdot a$.

13a Given a finite state automaton for R, we turn it into a finite state transducer for the intersection with R by changing each transition reading a letter into one reading ánd writing it. This fst copies a string to the output and accepts provided that string belongs to R: intersection.

13b Given a finite state automaton for R, we turn it into a finite state transducer for the right quotient with R as follows. First we copy a prefix of the input, using a single state with loops. Then nondeterministically we move to the automaton for R, which tests the suffix of the string, while outputting nothing.

14 (13b right quotient with R) Assume the regular language R is subset of $\{a, b\}$. The inverse of the morphism $g : a \mapsto a, b \mapsto b, \bar{a} \mapsto a, \bar{b} \mapsto b$ marks a random part of strings in a language. After intersecting with $\{\bar{a}, \bar{b}\}^* \cdot R$ we keep strings that start with a marked prefix followed by a suffix in R. Applying the morphism $f : a \mapsto \lambda, b \mapsto \lambda, \bar{a} \mapsto a, \bar{b} \mapsto b$ we delete the R-suffix, while unmarking the prefix.

Thus, $K/R = h(g^{-1}(K) \cap R)$, the composition we are looking for.

17 M.O. Rabin, D. Scott. Finite automata and their decision problems, IBM Journal of Research and Development 3, 114–125, 1959.
also in: Sequential Machines: Selected Papers (E.F. Moore, ed.), Addison-Wesley, Reading, MA, 1964, pp. 63–91.
http://www.research.ibm.com/journal/rd/032/ibmrd0302C.pdf

J.C. Shepherdson. The reduction of two-way automata to one-way automata, IBM Journal of Research and Development 3, 198–200, 1959.
also in: Sequential Machines: Selected Papers (E.F. Moore, ed.), Addison-Wesley,

Reading, MA, 1964, pp. 92–97.
http://www.research.ibm.com/journal/rd/032/ibmrd0302P.pdf

19 Jean H. Gallier, Salvatore La Torre, and Supratik Mukhopadhyay, Deterministic finite automata with recursive calls and DPDAs, Information Processing Letters, Volume 87, 187 – 193, 2003. `http://dx.doi.org/10.1016/S0020-0190(03)00281-3`

> **Abstract.** We study deterministic finite automata (DFA) with recursive calls, that is, finite sequences of component DFAs that can call each other recursively. DFAs with recursive calls are akin to recursive state machines and unrestricted hierarchic state machines. We show that they are language equivalent to deterministic pushdown automata (DPDA).