Fundamentele Informatica II

Answer to selected exercises 5

John C Martin: Introduction to Languages and the Theory of Computation

M.M. Bonsangue (and J. Kleijn)

Fall 2011

5.1.a $(q_0, ab, Z_0) \vdash (q_1, b, aZ_0) \vdash (q_2, \Lambda, Z_0) \vdash$, $(q_3, \Lambda, Z_0) \vdash$, acceptance.
$(q_0, aab, Z_0) \vdash (q_1, ab, aZ_0) \vdash (q_1, b, aaZ_0) \vdash (q_2, \Lambda, aZ_0) \vdash$, crash.
$(q_0, abb, Z_0) \vdash (q_1, bb, aZ_0) \vdash (q_2, b, Z_0) \vdash$, crash.
5.1.b $(q_0, bbcbb, Z_0) \vdash (q_0, bcbb, bZ_0) \vdash (q_0, cbb, bbZ_0) \vdash (q_1, bb, bbZ_0) \vdash (q_1, b, bZ_0)$
$\vdash (q_1, \Lambda, Z_0) \vdash (q_2, \Lambda, Z_0)$, acceptance.
$(q_0, baca, Z_0) \vdash (q_0, aca, bZ_0) \vdash (q_0, ca, abZ_0) \vdash (q_1, a, abZ_0) \vdash (q_1, \Lambda, bZ_0) \nvdash$, crash.

5.2 We give here all computations of the PDA on aba. Note that the non-determinism is present only in q_0. In that state one can always, whatever the top of the stack, with a Λ-transition go to state q_1. This corresponds to guessing that the input is of even length and the middle of the string has been found. Moreover one may switch also to q_1 when reading an a or a b, which correspond to guessing that the input is of odd length and the current symbol is the middle one. Finally, one may simply read and stack the input symbol.
$(q_0, aba, Z_0) \vdash (q_1, aba, Z_0) \vdash (q_2, aba, Z_0) \nvdash$, crash
$(q_0, aba, Z_0) \vdash (q_1, ba, Z_0) \vdash (q_2, ba, Z_0) \nvdash$, crash
$(q_0, aba, Z_0) \vdash (q_0, ba, aZ_0) \vdash (q_1, ba, aZ_0) \nvdash$, crash
$(q_0, aba, Z_0) \vdash (q_0, ba, aZ_0) \vdash (q_1, a, aZ_0) \vdash (q_1, \Lambda, Z_0) \vdash (q_2, \Lambda, Z_0)$, acceptance
$(q_0, aba, Z_0) \vdash (q_0, ba, aZ_0) \vdash (q_0, a, baZ_0) \vdash (q_1, a, baZ_0) \nvdash$, crash
$(q_0, aba, Z_0) \vdash (q_0, ba, aZ_0) \vdash (q_0, a, baZ_0) \vdash (q_1, \Lambda, baZ_0) \nvdash$, crash
$(q_0, aba, Z_0) \vdash (q_0, ba, aZ_0) \vdash (q_0, a, baZ_0) \vdash (q_0, \Lambda, abaZ_0) \vdash (q_1, \Lambda, abaZ_0) \nvdash$
crash

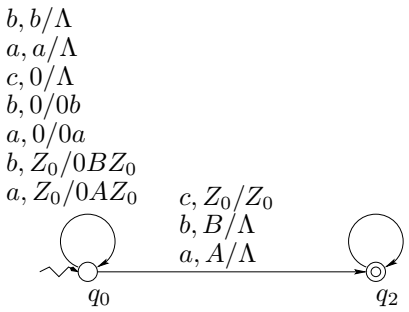
5.3 As argued in exercise 5.2, the PDA has two options in state q_0: to consider the current symbol as the middle one in an odd length palindrome,

or to make a Λ-move to q_1. In addition, there is the option never to leave q_0. Consequently, for an input string of length n the PDA has $2n + 1$ different (complete) computations.

5.4 a. The PDA accepts only even length palindromes once the possibilities (in moves 1-6) to go from q_0 to q_1 while reading an a or a b have been removed.
b. The PDA accepts only odd length palindromes once the possibilities (moves 7-9) to go from q_0 to q_1 with a Λ-move have been removed.

5.6 a. This PDA accepts $\{axa, bxb \mid x \in \{a, b\}^*\}$.
b. The PDA accepts $\{xcy \mid x, y \in \{a, b\}^* \text{ and } |x| = |y|\}$.

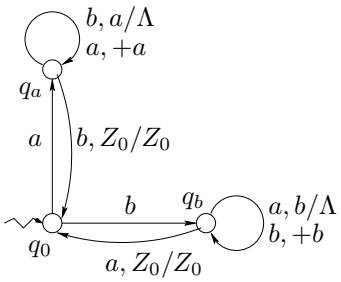
5.9 In the figure below we have drawn the transition diagram of a PDA for the language $\{xcx^r \mid x \in \{a, b\}^*\}$ of simple palindromes with two states (compare with Table 5.6). It uses the stack symbol 0 to indicate that it is reading the "first half" of the word (before the c) and that the a's and b's have to be pushed onto the stack. Reading c pops the 0. Then for each newly read a and b, the corresponding symbol has to be popped. The stack symbols A and B are used only for the first a and b, respectively, and when they are on top they indicate that the PDA should go to the accepting state q_2 while reading the last input a or b.



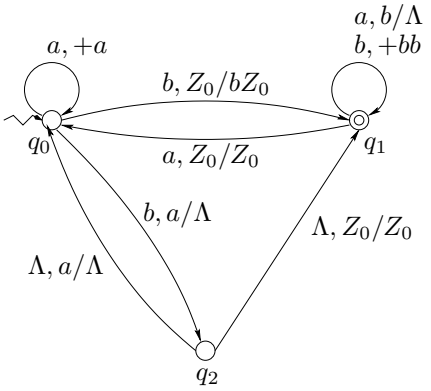
5.10 Let $M_0 = (Q, \Sigma, q_0, A, \delta)$ be an FA. From M_0 we construct a DPDA M with two states: p_0 is the initial state; if $q_0 \in A$, that is $\Lambda \in L(M_0)$, then p_0 is also the accepting state, otherwise p_1 is the accepting state. The stack alphabet of M is Q with q_0 as the initial stack symbol.
The DPDA simulates M_0 as follows: the state on top of the stack corresponds to the current state of M_0. If M_0 moves to a state r while reading an input symbol a, also M reads a and it pushes r onto the stack while moving to

the accepting state if $r \in A$ and to the non-accepting state if $r \notin A$.
Note that M never removes a symbol from the stack, has no Λ-transitions, and is deterministic (because M_0 is deterministic).

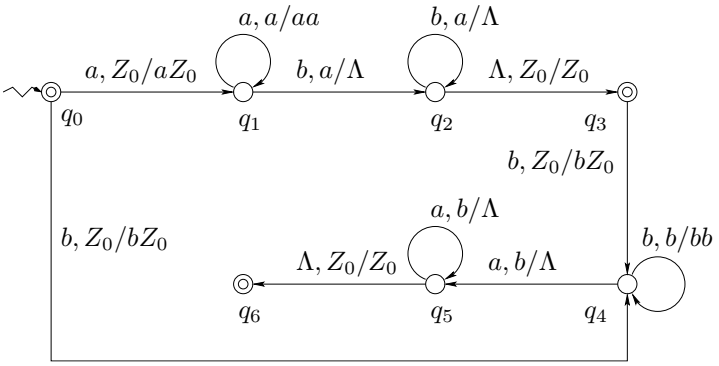
5.18 For **a.** modified for the language $\{x | n_a(x) = n_b(x)\}$ and **b.** see the diagram below. This DPDA has state q_0 corresponding to the situation that the same numbers of a's and b's have been read. In q_a more a's than b's have been read and, similarly, in q_b, more b's than a's have been read. Thus for the language $\{x \in \{a,b\}^* \mid n_a(x) = n_b(x)\}$ choose q_0 as the only accepting state. For $\{x \in \{a,b\}^* \mid n_a(x) \neq n_b(x)\}$ let q_a and q_b be the accepting states.



c. A DPDA for c is given in the book on page 406. Here we give the modified version for $\{x \in \{a,b\}^* \mid n_a(x) < 2n_b(x)\}$.
Essentially every b that is read is treated as if it were two b's, because the number of a's has to be compared to twice the number of b's.
In q_0, the stack contains only Z_0 with zero or more a's on top. For every a read, an a is pushed; for every b first one a is popped and then in state q_2 it is checked whether still another a can be popped. If not, the state is changed to q_1. The DPDA goes from state q_0 to state q_1, if a b is read and there are no a's on the stack. Note that only one b is pushed onto the stack now. In q_1 for every b read, two b's are pushed onto the stack and for every a one b is popped or if there are no b's anymore, the state changes to q_0 and the stack is not affected. Thus if the DPDA is in state q_0 after having read an input string w, the number of a's on the stack is $n_a(w) - 2n_b(w) \geq 0$.
When the DPDA is in state q_1 after reading a string w, with Z_0 at the top of the stack $n_a(w) = 2n_b(w) - 1$ and if b is at the top, then $n_a(w) \leq 2n_b(w) - 2$.



d. A DPDA for $\{a^n b^{n+m} a^m \mid n, m \geq 0\}$.



The empty word is accepted in q_0; in q_1 a's are pushed onto the stack; q_2 is used to match b's with the first series of a's. If these are exhausted, the DPDA can accept the input in q_3. However more b's may follow which are pushed onto the stack and finally matched with a new series of a's in q_5. When the stack thus has been emptied the input can be accepted in q_6.

5.19 M_1 and M_2 are two PDAs specified by $M_1 = (Q_1, \Sigma_1, \Gamma_1, q_1, Z_1, A_1, \delta_1)$ and $M_2 = (Q_2, \Sigma_2, \Gamma_2, q_2, Z_2, A_2, \delta_2)$. Without loss of generality we may assume that $Q_1 \cap Q_2 = \emptyset$ and $\Gamma_1 \cap \Gamma_2 = \emptyset$.
a. $L(M_1) \cup L(M_2)$ is accepted by the new PDA $M = (Q_1 \cup Q_2 \cup \{q_0\}, \Sigma_1 \cup \Sigma_2, \Gamma_1 \cup \Gamma_2 \cup \{Z_0\}, q_0, Z_0, A_1 \cup A_2, \delta_0)$, where q_0 is a new state, the inital state of M, and Z_0 is the new initial stack symbol. The transition function δ_0 of M is defined by
$\delta_0(q, d, X) = \delta_1(q, d, X)$ if $q \in Q_1$, $d \in \Sigma_1 \cup \{\Lambda\}$, and $X \in \Gamma_1$;
$\delta_0(q, d, X) = \delta_2(q, d, X)$ if $q \in Q_2$, $d \in \Sigma_2 \cup \{\Lambda\}$, and $X \in \Gamma_2$;
and $\delta_0(q_0, \Lambda, Z_0) = \{(q_1, Z_1), (q_2, Z_2)\}$.

Thus in M first a nondeterministic choice is made between M_1 and M_2 after which the automaton proceeds as the chosen one. Hence $L(M) = L(M_1) \cup L(M_2)$.
b. $L(M_1)L(M_2)$ is accepted by the new PDA $M = (Q_1 \cup Q_2 \cup \{q_0\}, \Sigma_1 \cup \Sigma_2, \Gamma_1 \cup \Gamma_2 \cup \{Z_0\}, q_0, Z_0, A_2, \delta_0)$, where q_0 is a new state, the inital state of M, and Z_0 is the new initial stack symbol. The transition function δ_0 of M is defined by
$\delta_0(q, d, X) = \delta_1(q, d, X)$ if $q \in Q_1 - A_1$, $d \in \Sigma_1 \cup \{\Lambda\}$, and $X \in \Gamma_1$;
$\delta_0(q, a, X) = \delta_1(q, a, X)$ if $q \in A_1$, $a \in \Sigma_1$, and $X \in \Gamma_1$;
$\delta_0(q, \Lambda, X) = \delta_1(q, \Lambda, X) \cup \{(q_2, Z_2X)\}$ if $q \in A_1$ and $X \in \Gamma_1$;
$\delta_0(q, d, X) = \delta_2(q, d, X)$ if $q \in Q_2$, $d \in \Sigma_2 \cup \{\Lambda\}$, and $X \in \Gamma_2$;
and $\delta_0(q_0, \Lambda, Z_0) = \{(q_1, Z_1Z_0)\}$.
Thus in M first Z_1 is pushed on top of Z_0 after which the automaton proceeds as M_1 until an original accepting state of M_1 is reached. Then there is a nondeterministic choice: either the computation proceeds as in M_1 until (again) an accepting state is reached, or the initial stack symbol of M_2 is pushed onto the current stack, the state becomes the initial state of M_2 and the computation proceeds as in M_2. Note that when an accepting state of M_1 is reached, the stack will not be empty because M_1 has no instructions for Z_0. Hence the computation will not crash at this point because of an empty stack. It may be the case that the stack still has some stack symbols from M_1 when the computation has been switched to M_2, but seeing such symbol corresponds to an empty stack in the original M_2 leading to a crash unless an accepting state has been reached. Hence $L(M) = L(M_1)L(M_2)$.
c. $L(M_1)^*$ is accepted by $M = (Q_1 \cup \{q_f\}, \Sigma_1, \Gamma_1 \cup \{Z_0\}, q_f, Z_0, \{q_f\}, \delta_0)$, where q_f is a new state, the inital and accepting state of M, and Z_0 is the new initial stack symbol. The transition function δ_0 of M is defined by
$\delta_0(q, d, X) = \delta_1(q, d, X)$ if $q \in Q_1 - A_1$, $d \in \Sigma_1 \cup \{\Lambda\}$, and $X \in \Gamma_1$;
$\delta_0(q, a, X) = \delta_1(q, a, X)$ if $q \in A_1$, $a \in \Sigma_1$, and $X \in \Gamma_1$;
$\delta_0(q, \Lambda, X) = \delta_1(q, \Lambda, X) \cup \{(q_f, Z_1X)\}$ if $q \in A_1$ and $X \in \Gamma_1$;
and $\delta_0(q_f, \Lambda, X) = \{(q_1, Z_1X)\}$ for all $X \in \Gamma_1 \cup \{Z_0\}$.
Thus M begins its computations in q_f. Since q_f is also an accepting state we have $\Lambda \in L(M)$. In q_f the stack symbol Z_1 is pushed on top of the stack and the computation proceeds as in M_1 until an accepting state of M_1 is reached. Then a nondeterministic choice is made: either the computation proceeds as in M_1 or the state is changed in the accepting q_f after which a new computation of M_1 may begin. Thus $L(M) = \{\Lambda\} \cup L(M_1)^+ = L(M)^*$.

5.20 Let L be the language accepted by the DPDA M. Then we can construct a DPDA M' which accepts the language $K = \{x\#y \mid x \in L \text{ and } xy \in$

$L\}$ consisting of all words in L which have a prefix in L as marked now by the new symbol #.
Without loss of generality we may assume that M has no Λ-transitions pointing out from an accepting state (see the intermezzo at 7.16). M' consists essentially of two distinct copies M'' and M''' of M. The initial state of M' is the state in the first copy M'' corresponding to the initial state of M and the accepting states of M' are the states in the second copy M''' corresponding to the accepting states of M. Computations start in M'' and proceed as in M until # is encountered. If the current state corresponds to an accepting state of M, then M' switches (deterministically, because M and hence M'', has no Λ-transitions from final states) from M'' to M''' where the computation is continued. Thus $\delta'(q'', \#, X) = (q''', X)$ whenever q is an accepting state of M with copy q'' in M'' and copy q''' in M'''.
Thus M' is a DPDA and a word is accepted by M' if and only if it is of the form $x\#y$ with $x \in L$ and $xy \in L$.

Note however that to prove this latter statement formally, one actually needs the assumption that M is deterministic. Otherwise, there may be words x, y such that x and xy belong to $L(M)$, but the computation leading to the acceptance of x cannot be prolonged to an accepting computation for xy (which may have an accepting computation which doesn't encounter an accepting state after the prefix x). This is the answer to exercise **8.21**: if M is not deterministic, then we do have $L(M') \subseteq K$, but this inclusion may be strict.

This result doesn't hold for context-free languages in general: `pal` is a context-free language, but $K = \{x\#y \mid x, xy \in \texttt{pal}\}$ is not a CFL.
This follows from the observation that if it were a CFL, then also $L = K \cap \{0\}^*\{1\}^*\{0\}^*\#\{1\}^*\{0\}^* = \{0^i1^j0^i\#1^j0^i \mid i, j \geq 0\}$. (Because the family of CFLs is closed under intersection with regular languages, Theorem 8.4.)
However it follows from the pumping lemma that L is not context-free.
Moreover: since L is not context-free, it is certainly not a DCFL. Thus K is not a DCFL and hence `pal` is not a DCFL according to what we have shown here in exercise **5.20**. This is an alternative proof for Theorem 5.16.

5.21 First we establish some **normal forms** for (D)PDAs:
For every PDA M, there exists (effectively) a PDA M' such that
$L(M) = L(M')$ and
1. M' never empties its stack
and/or
2. M' has no Λ-transitions from an accepting state.
Moreover M' is deterministic whenever M is.

Let $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$.
To guarantee property 1, we create a new initial state ${q_0}' \notin Q$ and we add a new (dummy) bottom symbol: $F \notin \Gamma$ together with a new move: $\delta'({q_0}', \Lambda, Z_0) = \{(q_0, Z_0F)\}$. All other parameters of M remain unchanged. Note that the resulting PDA never pops or pushes F after leaving the new initial state (to which it never returns). It crashes due to lack of instructions when F is the top of the stack if and only if M has an empty stack (and the two automata are in the same state after having processed the same prefix of the current input word). It follows that the newly constructed PDA accepts the same language as M and that it is deterministic whenever M is.
To enforce property 2, we modify M in the following way. We introduce for every $a \in \Sigma$, an a-indexed copy of each state: $Q_a = \{r_a \mid r \in Q\}$. Whenever M has a Λ-transition from an accepting state we replace this by transitions which guess what symbol ($a \in \Sigma$) will be read next:
for all $p \in A$, $Z \in \Gamma$, and $a \in \Sigma$, replace every $(r, \alpha) \in \delta(p, \Lambda, Z)$ by (r_a, α);
and move between (a-indexed) new states just like M (until an input symbol, which has to be a, is read):
for all $p \in Q$, $Z \in \Gamma$, and $a \in \Sigma$,
add (q_a, α) as a move from (r_a, Λ, Z) whenever $(q, \alpha) \in \delta(r, \Lambda, Z)$ and
add (q, α) as a move from (r_a, Λ, Z) whenever $(q, \alpha) \in \delta(r, a, Z)$.
This defines a new PDA M'. By construction, M' satisfies property 2. That $L(M') = L(M)$ follows from the correspondence between the successful computations of M and M' (without loss of generality we do not consider successful computations which move from accepting state to accepting state without ever reading a new input symbol). Again M' is deterministic whenever M is.
We conclude with the observation that the two constructions given above are independent. Hence we can apply them one after the other to obtain a (D)PDA satisfying both properties 1 and 2.

5.22 Let $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ be a DPDA accepting the language L. Let \$ be a new symbol (not in the alphabet of L). Then we can construct from M a new DPDA M' which accepts $L\{\$\}$ by empty stack as follows.
First, observe that, according to the intermezzo above, we may assume that no computation of M ever leads to the stack being empty. Moreover, M has no Λ-transitions leaving an accepting state. M' proceeds as M until it reaches an accepting state $q \in A$. Then M' moves to the new state p_e provided it reads \$: $\delta'(q, \$, X) = (p_e, \Lambda)$ for all $q \in A$ and $X \in \Gamma$.
In p_e the stack is emptied without reading any symbols anymore: $\delta'(p_e, \Lambda, X) = (p_e, \Lambda)$ for all $X \in \Gamma$.

Note that M' is deterministic because M satisfies property 2. Furthermore, the stack can only be emptied at p_e which state can only be reached after reading a \$ following a word accepted by M. Conversely, for every $w \in L$, we have $w\$ \in L_e(M')$. Thus $L\{\$\} = L(M')$ is accepted by the DPDA M' by empty stack.

5.23 Let L be a language accepted by a DPDA M by empty stack. Thus for every $x \in L$, the (unique) sequence of moves by M to accept x leaves the stack empty. Consequently whenever $x \in L$ there doesn't exist any $z \neq \Lambda$ such that $xz \in L$, because M is deterministic and cannot move after accepting x.
This implies that no language L with $x, y \in L$ such that x is a proper prefix of y can be accepted by a DPDA by empty stack.

5.24 The property of `pal` used in the proof of Theorem 5.16 is that all distinct words in $\{a, b\}^*$ are distinguishable with respect to `pal`:
For all $x, y \in \{a, b\}^*$, whenever $x \neq y$ there exists a z such that $xz \in$ `pal` and $yz \notin$ `pal` or vv.
To prove that the given languages cannot be accepted it is sufficient to establish that they also have this property and apply the reasoning from the proof of Theorem 5.16.
a. and **b.** Similar to the argument in Example 2.27.
c. $L = \{uv \in \{0,1\}^* \mid v = u^!\}$. Here $u^!$ is the word obtained from u by replacing 0's by 1's and 1's by 0's.
See **d** and try to adapt the proof given there.
d. $L = \{uv \in \{0,1\}^* \mid v = u \text{ or } v = u^!\}$.
Let $x, y \in \{0,1\}^*$ with $x \neq y$. If $|x| = |y|$, then consider $z = 0x0$. Clearly $x0x0 = xz \in L$. However $yz = y0x0$ cannot be in L: since $|x| = |y|$ we know that $x0 \neq (y0)^!$ and since in addition $x \neq y$ it is also not the case that $y0 = x0$.
Now assume that $|x| < |y|$ (the case $|y| < |x|$ can be dealt with analogously). First assume that $|y| - |x|$ is odd. Let $z = x$. Then $xz = xx \in L$, but $yz = yx \notin L$, because $|yx| = |y| + |x| = |y| - |x| + 2|x|$ is odd. Next assume that $|y| - |x| = 2k$ for some $k \geq 1$. Thus $y = wu_1u_2$ with $|w| = |x|$ and $|u_1| = |u_2| = k$. Let $z = u_2u_2^!xu_2u_2^!$. Then $xz = xu_2u_2^!xu_2u_2^! \in L$, but $yz = wu_1u_2u_2u_2^!xu_2u_2^!$ is not in L: because yz consists of two halves $|wu_1u_2u_2| = |u_2^!xu_2u_2^!|$ ending differently with u_2 and $u_2^!$ respectively; and it cannot be of the form $ww^!$ because the two halves end with u_2u_2 and $u_2u_2^!$ respectively.

5.25 A counter automaton is a PDA with next to the initial stack symbol

Z_0 only one other stack symbol A. During a computation the stack contents is always of the form $A^n Z_0$ where $n \geq 0$. Thus, the automaton can only "count": push and pop A's.

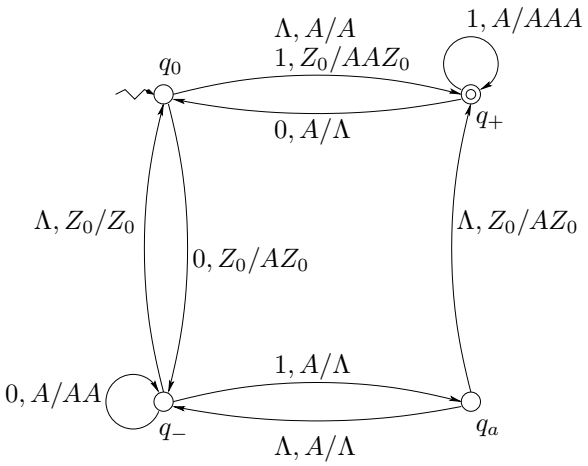
b. A counter automaton for $L = \{x\{0,1\}^* \mid n_0(x) < 2n_1(x)\}$ compares the number of 0's read with the number of 1's read under the assumption that every 1 counts for two 0's (see also exercise 7.13c).

For $x \in \{0,1\}^*$ we use $d(x) = 2n_1(x) - n_0(x)$ to indicate the difference between two times the number of 1's in x and the number of 0's in x. The automaton has 3 states: q_0 corresponding to $d(x) = 0$, the initial state; q_+ corresponding to $d(x) > 0$, the accepting state; and q_- corresponding to $d(x) < 0$. The number of A's on the stack indicates the absolute value of $d(x)$. The state q_a is an auxiliary state.

In q_0 reading initially a 1 leads to pushing two A's and a move to q_+. Also in q_+ reading a 1 leads to pushing two A's. When in q_+ a 0 is read one A is popped and the automaton moves temporarily to q_0 where it is checked whether or not there are still A's on the stack (to avoid acceptance in case $d(x) = 0$). If there are still A's the automaton moves back to q_+ with a Λ-transition. Otherwise we are back in the initial situation.

In q_0 reading initially a 0 leads to pushing an A and a move to q_-. Also in q_- reading a 0 leads to pushing an A. When in q_- a 1 is read one A is popped (if possible) and the automaton moves temporarily to q_a where it is checked whether or not another A can be popped. If yes, the automaton moves back to q_- with a Λ-transition while popping the second A. If no, then the automaton moves to q_+ with a Λ-transition while pushing an A (note that $d(x)$ is now 1).

When in q_- the stack no longer contains A's the automaton moves with a Λ-transition back to the inital situation ($d(x) = 0$).



Note that this PDA (counter automaton) is deterministic.

5.26 Let M be a DPDA over some input alphabet Σ. Consider a word $y \in \Sigma^* - L(M)$ and suppose that M has not read the last letter of y (note that since M is deterministic, it has only one computation for y). Then, before it can read this last letter M either crashes (case 1) or it enters an infinite loop of Λ-transitions (case 2). Note that case 2 can be easily translated into case 1 by adding a "garbage" state g with the "missing" instructions now added and leading to g. State g has only Λ-transitions pointing back to it (one for each stack symbol).

Apply this construction to the DPDA of Example 5.11 (accepting balanced strings of brackets), see Table 5.12. One of the strings on which it crashes is }.

5.28 b. Given is the CFG with productions $S \to S + S \,|\, S * S \,|\, (S) \,|\, a$. This grammar generates the string $x = (a * a + a)$. We consider the top-down PDA constructed from the grammar as in Definition 7.4 and trace a sequence of steps by which x is accepted together with the corresponding leftmost derivation.

state	string	stack		step
q_0	$(a*a+a)$	Z_0		
q_1	$(a*a+a)$	SZ_0		S
q_1	$(a*a+a)$	$(S)Z_0$		$\Rightarrow (S)$
q_1	$a*a+a)$	$S)Z_0$		
q_1	$a*a+a)$	$S+S)Z_0$		$\Rightarrow (S+S)$
q_1	$a*a+a)$	$S*S+S)Z_0$		$\Rightarrow (S*S+S)$
q_1	$a*a+a)$	$a*S+S)Z_0$		$\Rightarrow (a*S+S)$
q_1	$*a+a)$	$*S+S)Z_0$		
q_1	$a+a)$	$S+S)Z_0$		
q_1	$a+a)$	$a+S)Z_0$		$\Rightarrow (a*a+S)$
q_1	$+a)$	$,+S)Z_0$		
q_1	$a)$	$S)Z_0$		
q_1	$a)$	$a)Z_0$		$\Rightarrow (a*a+a)$
q_1	$)$	$)Z_0$		
q_1	Λ	Z_0		
q_2	Λ	Z_0		

5.32 Consider the PDA M from Example 5.7 with move 12 $\delta(q_1, \Lambda, Z_0) = \{(q_2, Z_0)\}$ changed into $\delta(q_1, \Lambda, Z_0) = \{(q_2, \Lambda)\}$.

state	string	stack		step
q_0	$ababa$	Z_0		$S \Rightarrow [q_0, Z_0, q_2]$
q_0	$baba$	aZ_0		$\Rightarrow a[q_0, a, q_1][q_1, Z_0, q_2]$
q_0	aba	baZ_0		$\Rightarrow ab[q_0, b, q_1][q_1, a, q_1][q_1, Z_0, q_2]$
q_1	ba	baZ_0		$\Rightarrow aba[q_1, b, q_1][q_1, a, q_1][q_1, Z_0, q_2]$
q_1	a	aZ_0		$\Rightarrow abab[q_1, a, q_1][q_1, Z_0, q_2]$
q_1	Λ	Z_0		$\Rightarrow ababa[q_1, Z_0, q_2]$
q_2	Λ	Λ		$\Rightarrow ababa$

This PDA accepts the same language $L = L(M)$ but now by empty stack and so we can apply Theorem 5.29: we consider the CFG as constructed there with $L(G) = L$.
For $x = ababa$ the table shows a sequence of steps in the new PDA by which x is accepted together with the corresponding leftmost derivation in G.

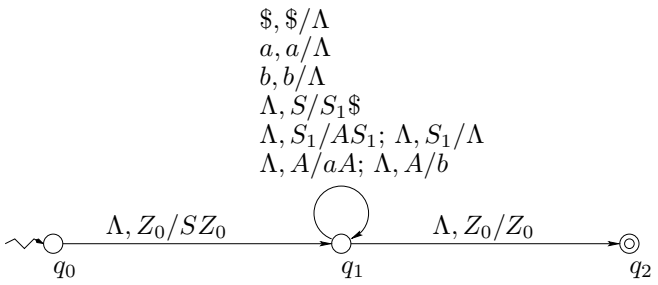
5.34 a. Given is the CFG with productions $S \to [S]S \,|\, \Lambda$. This grammar generates the string $x = [][[]]$. We consider the bottom-up PDA constructed

from the grammar as in Example 5.24 and trace a sequence of steps by which x is accepted together with the corresponding rightmost derivation of x in the grammar in reverse order.

move	state	string	stack		step
	q	$[][]$	Z_0		
reduce	q	$[][]$	SZ_0		$\Rightarrow [][]$
shift	q	$][]$	$[SZ_0$		
reduce	q	$][]$	$S[SZ_0$		$\Rightarrow S[][]$
shift	q	$[]$	$]S[SZ_0$		
reduce	$q_{1,1}$	$[]$	$S[SZ_0$		$\Rightarrow S[S][]$
	$q_{1,2}$	$[]$	$[SZ_0$		
	$q_{1,3}$	$[]$	SZ_0		
	q	$[]$	SZ_0		
shift	q	$]$	$[SZ_0$		
reduce	q	$]$	$S[SZ_0$		$\Rightarrow S[S]$
shift	q	$]]$	$[S[SZ_0$		
reduce	q	$]]$	$S[S[SZ_0$		$\Rightarrow S[S[]]$
shift	q	$]$	$]S[S[SZ_0$		
reduce	$q_{1,1}$	$]$	$S[S[SZ_0$		$\Rightarrow S[S[S]]$
	$q_{1,2}$	$]$	$[S[SZ_0$		
	$q_{1,3}$	$]$	$S[SZ_0$		
	q	$]$	$S[SZ_0$		
shift	q	Λ	$]S[SZ_0$		
reduce	$q_{1,1}$	Λ	$S[SZ_0$		$S \Rightarrow S[S]$
	$q_{1,2}$	Λ	$[SZ_0$		
	$q_{1,3}$	Λ	SZ_0		
	q	Λ	SZ_0		
(pop S)	q_1	Λ	Z_0		
(accept)	q_2	Λ	Z_0		

5.36 Let M be a PDA (accepting with empty stack) and consider the CFG G obtained from M as in the proof of Theorem 5.29. Since every accepting computation of M determines a unique leftmost derivation of a word in $L(G)$, it follows that G is unambiguous if M is deterministic.
However if G is unambiguous it is not necessarily the case that M is deterministic. It is sufficient if M never has more than one *accepting* computation per word.

5.38 Each of the given grammars satisfies the LL(1) property: looking one symbol ahead in the input is sufficient to choose (deterministically) the next move of its associated push-down parser. Thus we can obtain for each a deterministic PDA:
a. Consider the CFG G given by the productions:
$S \to S_1\$, \quad S_1 \to AS_1 \,|\, \Lambda, \quad A \to aA \,|\, b.$
The top-down PDA associated with G (Definition 7.4) is given below through its transition diagram.



This PDA is non-deterministic as a consequence of a choice in productions when rewriting S_1 or A.
G is LL(1): The lookahead for $S_1 \to AS_1$ is $\{a, b\}$ and for $S_1 \to \Lambda$ it is $\{\$\}$; thus the two productions have disjoint lookaheads;
also, the lookahead for $A \to aA$ is $\{a\}$ and for $A \to b$ it is $\{b\}$.
Note that the lookahead for $S \to S_1\$$ is $\{a, b, \$\}$.
From this PDA (or rather from the grammar) we can now construct a deterministic PDA by incorporating the lookahead.

The most systematic (preferred) way is to introduce a separate state for each terminal symbol (input symbol to the PDA). As long as the top of the stack is not Z_0, the PDA has to read in q_1 first a new input symbol; it then moves to the corresponding state where it applies productions which have that symbol in their lookahead until that terminal symbol is produced (on top of the stack); then it moves back to q_1 while popping the terminal. See Figure 1, below. The thus obtained PDA is clearly deterministic.
Note that this parser is actually directly constructed from the grammar!

Alternatively, one could also (more opportunistically) repair only the non-determinism where it actually occurs (see Figure 2):
Again separate states for terminal symbols are introduced. Whenever the top of the stack is a non-terminal with more than one production, the PDA has to read an input symbol which then determines what production is to be applied. The PDA moves to the corresponding state where it applies

productions which have that symbol in their lookahead until it appears on top of the stack; then it moves back to q_1 while popping the terminal.

Figure 1

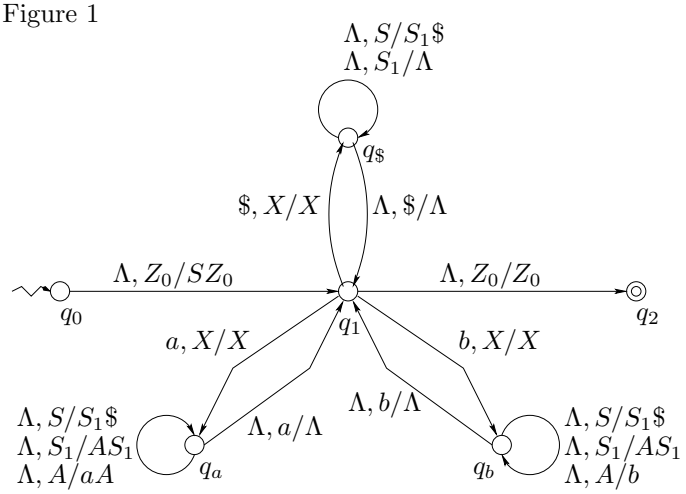
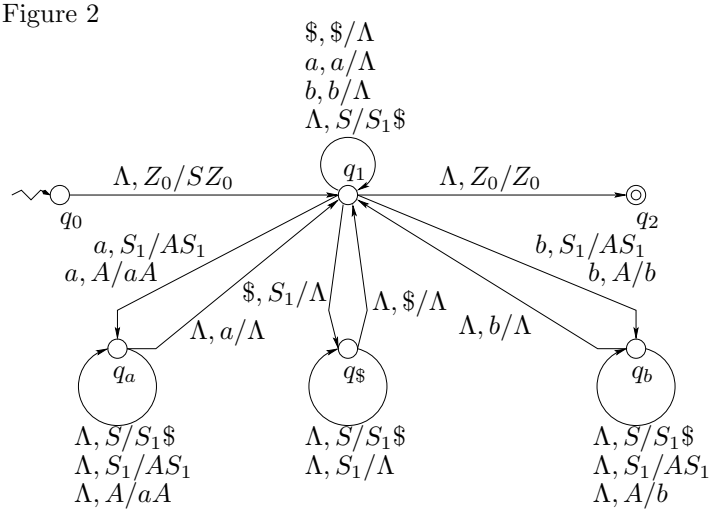


Figure 2



It is possible that the DPDAs have superfluous transitions which could be removed.

b. Consider the CFG G given by the productions:
$S \to S_1\$, \quad S_1 \to aA, \quad A \to aA \,|\, bA \,|\, \Lambda.$
Lookaheads:
$\mathtt{LA}_1(S \to S_1\$) = \{a\}$

$\texttt{LA}_1(S_1 \to aA\$) = \{a\}$
$\texttt{LA}_1(A \to aA) = \{a\}$ $\texttt{LA}_1(A \to bA) = \{b\}$ $\texttt{LA}_1(A \to \Lambda) = \{\$\}$.
(Note that G is LL(1).)
The deterministic pushdown parser we give below has initial state q_0, "intermediate" state q_1, accepting state q_2, and three "input lookahead" states q_a, q_b, and $q_\$$. The transition table, derived from the productions and their lookahead, follows next. Symbol X denotes an arbitrary stack symbol which is not Z_0.

state	input	stack symbol	move
q_0	Λ	Z_0	(q_1, SZ_0)
q_1	Λ	Z_0	(q_2, Z_0)
q_1	a	X	(q_a, X)
q_1	b	X	(q_b, X)
q_1	\$	X	$(q_\$, X)$
q_a	Λ	a	(q_1, Λ)
q_b	Λ	b	(q_1, Λ)
$q_\$$	Λ	\$	(q_1, Λ)
q_a	Λ	S	$(q_1, S_1\$)$
q_a	Λ	S_1	(q_1, aA)
q_a	Λ	A	(q_1, aA)
q_b	Λ	A	(q_1, bA)
$q_\$$	Λ	A	(q_1, Λ)

c. Consider the CFG G given by the productions:
$S \to S_1\$, \quad S_1 \to aAb \,|\, bBa, \quad A \to bS_1 \,|\, a, \quad B \to aS_1 \,|\, b.$
Lookaheads:
$\texttt{LA}_1(S \to S_1\$) = \{a, b\}$
$\texttt{LA}_1(S_1 \to aAb\$) = \{a\}$ $\texttt{LA}_1(S_1 \to bBa\$) = \{b\}$
$\texttt{LA}_1(A \to bS_1) = \{b\}$ $\texttt{LA}_1(A \to a) = \{a\}$
$\texttt{LA}_1(B \to aS_1) = \{a\}$ $\texttt{LA}_1(B \to b) = \{b\}$.
(Note that G is LL(1).)
The deterministic pushdown parser we give below has initial state q_0, "intermediate" state q_1, accepting state q_2, and three "input lookahead" states q_a, q_b, and \$. (Note that \$ is not a lookahead symbol of the grammar.) The transition table, derived from the productions and their lookahead, follows next. Symbol X denotes an arbitrary stack symbol which is not Z_0.

state	input	stack symbol		move
q_0	Λ	Z_0		(q_1, SZ_0)
q_1	Λ	Z_0		(q_2, Z_0)
q_1	a	X		(q_a, X)
q_1	b	X		(q_b, X)
q_1	\$	X		$(q_\$, X)$
q_a	Λ	a		(q_1, Λ)
q_b	Λ	b		(q_1, Λ)
$q_\$$	Λ	\$		(q_1, Λ)
q_a	Λ	S		$(q_1, S_1\$)$
q_a	Λ	S_1		(q_1, aAb)
q_a	Λ	A		(q_1, a)
q_a	Λ	B		(q_1, aS_1)
q_b	Λ	S		$(q_1, S_1\$)$
q_b	Λ	S_1		(q_1, bBa)
q_b	Λ	A		(q_1, bS_1)
q_b	Λ	B		(q_1, b)

5.41 a. Consider the CFG given by: $S \to S_1\$, \quad S_1 \to aaS_1b \,|\, ab \,|\, bb$.
This grammar is not LL(1): a is a lookahead symbol both for $S_1 \to aaS_1b$ and $S_1 \to ab$. We factor the righthand sides of these two productions:
$S_1 \to aX, \quad X \to aS_1b \,|\, b$.
Thus the modified grammar is given by the productions:
$S \to S_1\$, \quad S_1 \to aX \,|\, bb \quad X \to aS_1b \,|\, b$.
This grammar is LL(1): different productions of the same nonterminal have different lookahead symbols.
b. Consider the CFG given by: $S \to S_1\$, \quad S_1 \to S_1A \,|\, \Lambda, \quad A \to Aa \,|\, b$.
This grammar is not LL(1) because of the left-recursion in the productions $S_1 \to S_1A$ and $A \to Aa$.
Replace $S_1 \to S_1A \,|\, \Lambda$ by $S_1 \to \Lambda X$ and $X \to AX \,|\, \Lambda$ and note that S_1 and X can be identified; thus we obtain $S_1 \to AS_1 \,|\, \Lambda$.
Replace $A \to Aa \,|\, b$ by $A \to bY$ and $Y \to aY \,|\, \Lambda$;
Thus the modified grammar is given by the productions:
$S \to S_1\$, \quad S_1 \to AS_1 \,|\, \Lambda, \quad A \to bY, \quad Y \to aY \,|\, \Lambda$.
It is easy to see that this grammar is LL(1):
Lookahead for $S_1 \to AS_1$ is $\{b\}$; for $S_1 \to \Lambda$ it is $\{\$\}$, thus no common symbols; also the lookahead for $Y \to aY$ which is $\{a\}$ and the lookahead for

$Y \to \Lambda$ which is $\{\$, b\}$ have no symbols in common.
Variant of 5.41b by Rudy van Vliet
Consider the context-free grammar with productions:
$S \to S_1\$ \quad S_1 \to S_1A \,|\, a \,|\, \Lambda \quad A \to Aa \,|\, b$
This context-free grammar does not satisfy the LL(1) property due to left recursion in the productions for S_1 leading to common lookahead symbols:
$\texttt{LA}_1(S_1 \to a) = \{a\} \quad \texttt{LA}_1(S_1 \to S_1A) = \{a, b\} \quad \texttt{LA}_1(S_1 \to \Lambda) = \{\$, b\}$.
And similarly for A: $\texttt{LA}_1(A \to b) = \{b\} \quad \texttt{LA}_1(A \to Aa) = \{b\}$.
We eliminate the left recursion using new non-terminal symbols U and W:
$S_1 \to aU \,|\, U \quad U \to \Lambda \,|\, AU \quad$ and $\quad A \to bW \quad W \to \Lambda \,|\, aW$
The productions of the resulting context-free grammar are
$S \to S_1\$ \quad S_1 \to aU \,|\, U \quad U \to \Lambda \,|\, AU \quad A \to bW \quad W \to \Lambda \,|\, aW$
This grammar is LL(1):
$\texttt{LA}_1(S_1 \to aU) = \{a\} \quad \texttt{LA}_1(S_1 \to U) = \{b, \$\}$
$\texttt{LA}_1(U \to AU) = \{b\} \quad \texttt{LA}_1(U \to \Lambda) = \{\$\}$
$\texttt{LA}_1(W \to aW) = \{a\} \quad \texttt{LA}_1(W \to \Lambda) = \{b, \$\}$
c. Consider the CFG given by: $S \to S_1\$, \quad S_1 \to S_1T \,|\, ab, \quad T \to aTbb \,|\, ab.$
After removing the left-recursion in the production $S_1 \to S_1T$ and the factoring of $T \to aTbb$ and $T \to ab$, the following grammar results:
$S \to S_1\$, \quad S_1 \to abX, \quad X \to TX \,|\, \Lambda, \quad T \to aY, \quad Y \to Tbb \,|\, b.$
This grammar is LL(1):
Lookahead for $X \to TX$ is $\{a\}$; for $X \to \Lambda$ it is $\{\$\}$.
Lookahead for $Y \to Tbb$ is $\{a\}$; for $Y \to b$ it is $\{b\}$.
d. Consider the CFG given by:
$S \to S_1\$, \quad S_1 \to aAb \,|\, aAA \,|\, aB \,|\, bbA, \quad A \to aAb \,|\, ab, \quad B \to bBa \,|\, ba.$
After factoring the following CFG results:
$S \to S_1\$, \quad S_1 \to aX \,|\, bbA, \quad X \to AY \,|\, B, \quad Y \to b \,|\, A,$
$A \to aZ, \quad Z \to Ab \,|\, b, \quad B \to bU, \quad U \to Ba \,|\, a.$
This grammar is LL(1):
Lookahead for $S_1 \to aX$ is $\{a\}$; for $S_1 \to bbA$ it is $\{b\}$.
Lookahead for $X \to AY$ is $\{a\}$; for $X \to B$ it is $\{b\}$.
etc. DIY

5.42 Consider the following modification of the CFG from exercise 5.41c:
$S \to S_1\$, \quad S_1 \to S_1T \,|\, ab, \quad T \to aTbb \,|\, a.$
After factoring and eliminating left-recursion we obtain:
$S \to S_1\$, \quad S_1 \to abX, \quad X \to TX \,|\, \Lambda, \quad T \to aY, \quad Y \to Tbb \,|\, \Lambda.$
This grammar however is not LL(1): Lookahead for $Y \to Tbb$ is $\{a\}$ while the lookahead for $Y \to \Lambda$ is $\{a, b, \$\}$. Thus these two productions of Y have a as a common lookahead symbol. Note that the lookahead for $X \to TX$ is

$\{a\}$ and for $X \to \Lambda$ it is $\{b, \$\}$.
The string $abaabbaa\$$ is an example where indeed parsing with a lookahead of one symbol cannot be done deterministically as demonstrated by the following leftmost derivation:
$S \Rightarrow S_1\$ \Rightarrow abX\$ \Rightarrow abTX\$$ lookahead a: $X \to TX$
$\Rightarrow abaYX\$ \Rightarrow abaTbbX$ lookahead a, **we choose** $Y \to Tbb$
$\Rightarrow abaaYbbX\$ \Rightarrow abaabbX\$ \Rightarrow abaabbTX\$$
. lookahead b: $Y \to \Lambda$; lookahead a: $X \to TX$
$\Rightarrow abaabbaYX\$ \Rightarrow abaabbaX\$ \Rightarrow abaabbaTX\$$
. lookahead a, **now we choose** $Y \to \Lambda$; lookahead a: $X \to TX$
$\Rightarrow abaabbaaYX\$ \Rightarrow abaabbaaX\$ \Rightarrow abaabbaa\$$
. lookahead \$: $Y \to \Lambda$; lookahead \$: $X \to \Lambda$.
Note that the points where we are forced to choose in the above derivation correspond to non-determinism in a "parser".

5.43 Consider the CFG given by: $S \to S_1\$$,
$S_1 \to S_1 + T \,|\, T, \quad T \to T * F \,|\, F, \quad F \to (S_1) \,|\, a.$
a. After eliminating left-recursion we obtain: $S \to S_1\$$,
$S_1 \to TX$, $X \to +TX \,|\, \Lambda, \quad T \to FY$, $Y \to *FY \,|\, \Lambda, \quad F \to (S_1) \,|\, a.$
b. The new CFG in **a** is LL(1):
lookahead for $X \to +TX$ is $\{+\}$ and for $X \to \Lambda$ it is $\{), \$\}$;
lookahead for $Y \to *FY$ is $\{*\}$ and for $Y \to \Lambda$ it is $\{+,), \$\}$;
lookahead for $F \to (S_1)$ is $\{(\}$ and for $F \to a$ it is $\{a\}$.
See exercise 5.38 for an explanation on how to obtain a DPDA (top-down parser) from this grammar.
