

# Discrete Event Systems

## Solution to Exercise Sheet 4

### 1 Context-Free Grammars

- a) An example for a grammar  $G$  producing the language  $L_1$  is  $G = (V, \Sigma, R, S)$  with

$$\begin{aligned}V &= \{X, A\}, \\ \Sigma &= \{0, 1\}, \\ R &= \left\{ \begin{array}{l} X \rightarrow XAX \mid A, \\ A \rightarrow 0 \mid 1 \end{array} \right\}, \\ S &= X\end{aligned}$$

*Note:* The language is regular!

- b) A rather natural grammar generating  $L_2$  uses the following productions:

$$\begin{aligned}S &\rightarrow A1A \\ A &\rightarrow A1 \mid 1A \mid A01 \mid 0A1 \mid 01A \mid A10 \mid 1A0 \mid 10A \mid \varepsilon\end{aligned}$$

Another slightly more complicated solution yielding simpler productions looks as follows:

$$\begin{aligned}S &\rightarrow A1A \\ A &\rightarrow AA \mid 1A0 \mid 0A1 \mid 1 \mid \varepsilon\end{aligned}$$

The idea of both grammars is to first ensure that there is at least one 1 more and then have a production that generates all possible strings with the same number of 0s and 1s or further 1s at arbitrary places.

### 2 Regular and Context-Free Languages

- a) Sometimes, even simple grammars can produce tricky languages. We can interpret the 1s and 2s of the second production rule as opening and closing brackets. Hence,  $L(G)$  consists of all correct bracket terms where at least one 0 must be in each bracket.

Choose  $w = 1^p 0 2^p \in L(G)$ . Let  $w = xyz$  with  $|xy| \leq p$  and  $|y| \geq 1$  (pumping lemma). Because of  $|xy| \leq p$ ,  $xy$  can only consist of 1s. According to the pumping lemma, we should have  $xy^i z \in L$  for all  $i \geq 0$ . However, by choosing  $i = 0$  we delete at least one 1 and get a word  $w' = 1^{p-|y|} 0 2^p$  with  $|y| \geq 1$ .  $w'$  is not in  $L$  since it has fewer 1s than 2s. This means that  $w$  is not pumpable and hence,  $L(G)$  is not regular.

- b) Since *every* regular language is also context-free, we can choose an arbitrary regular language. For example, we can choose the language  $L = \{0^n 1, n \geq 1\}$  which is clearly regular. A context-free grammar for this language uses only the production  $S \rightarrow 0S \mid 1$ .

### 3 Pumping Lemma Revisited

- a) Let us assume that  $L$  is regular and show that this results in a contradiction.

We have seen that any regular language fulfills the pumping lemma. This means, there exists a number  $p$ , such that every word  $w \in L$  with  $|w| \geq p$  can be written as  $w = xyz$  with  $|xy| \leq p$  and  $|y| \geq 1$ , such that  $xy^iz \in L$  for all  $i \geq 0$ .

In order to obtain the contradiction, we need to find at least one word  $w \in L$  with  $|w| \geq p$  that does not adhere to the above proposition. We choose  $w = xyz = 1^{p^2}$  and consider the case  $i = 2$  for which the Pumping Lemma claims  $w' = xy^2z \in L$ .

We can relate the lengths of  $w = xyz$  and  $w' = xy^2z$  as follows.

$$p^2 = |w| = |xyz| < |w'| = |xy^2z| \leq p^2 + p < p^2 + 2p + 1 = (p + 1)^2$$

So we have  $p^2 < |w'| < (p + 1)^2$  which implies that  $|w'|$  cannot be a square number since it lies between two consecutive square numbers. Therefore,  $w' \notin L$  and hence,  $L$  cannot be regular.

- b) Consider the alphabet  $\Sigma = \{a_1, a_2, \dots, a_n\}$  and the language  $L = \bigcup_{i=1}^n a_i^* = a_1^* \cup a_2^* \cup \dots \cup a_n^*$ . In other words, each word of the language  $L$  contains an arbitrary number of just **one** symbol  $a_i$ . The language is regular, as it is the union of regular languages, and the smallest possible pumping number  $p$  for  $L$  is 1. But any DFA needs at least  $n + 2$  states to accept the empty word, distinguish the  $n$  different characters of the alphabet, and for a failing state. Thus, for the DFA, we cannot deduce any information from  $p$  about the minimum number of states.

The same argument holds for the NFA.

### 4 Context Free or Not?

- a) For reasons of brevity, we only give the productions of the grammar.

First, we create an equal number of symbols for  $w$  and  $z$  using rule (2), and then an equal number of symbols for  $x$  and  $y$  using rule (3).

$$S \rightarrow A \tag{1}$$

$$A \rightarrow YAY \mid \#B\# \tag{2}$$

$$B \rightarrow YBY \mid \# \tag{3}$$

$$Y \rightarrow a \mid b \tag{4}$$

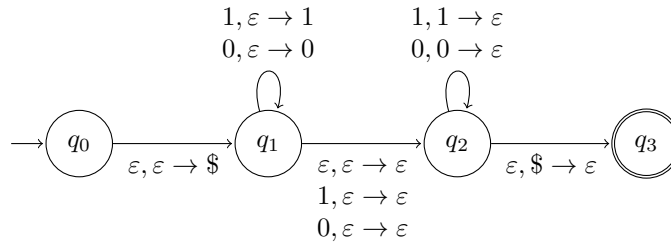
- b) If  $|w| = |y|$  and  $|x| = |z|$ , the resulting language is not context free, thus a CFG does not exist. This can be seen using the tandem pumping lemma as follows.

Let the word considered be  $s = a^p \# a^p \# a^p \# a^p \in L$  with  $|s| = 4p + 3 \geq p$ . For any division  $s = defgh$  with  $|eg| \geq 1$  and  $|efg| \leq p$ , the pumpable regions  $e$  and  $g$  can never consist of both  $a$ s from  $w$  and  $y$  or both  $x$  and  $z$  because of the condition  $|efg| \leq p$ . Hence, any pumping would inevitably only modify the number of  $a$ s in one part thereby creating a word  $s' \notin L$ . Therefore,  $L$  cannot be context free.

### 5 Push Down Automata

- a) This PDA should recognize all palindromes. However, we don't know where the middle of the word to recognize is. Therefore, we have to construct a non-deterministic automaton that decides itself when the middle has been reached.

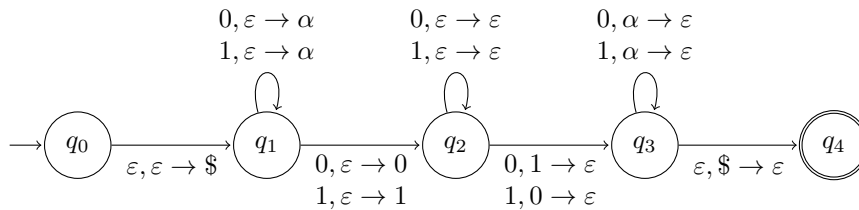
Note that we need to support words of even and odd length. Words of even length have a counterpart for each letter. However, the center letter of an odd word has no counterpart.



- b) Consider the word  $w$  to be an array of symbols. If  $w \in L$ , there is at least one offset  $c$ , such that  $w[c] \neq w[|w| - c]$ . That is, there are two symbols  $x$  and  $y$  in  $w$  s.t.  $x \neq y$  and the distance of  $x$  from the start of  $w$  equals the distance of  $y$  from the end of  $w$ .

The PDA reads  $c - 1$  symbols, and stores a token  $\alpha$  on the stack for each read symbol. Then, it reads the  $c$ -th symbol, and puts the symbol onto the stack. Afterwards, the PDA allows to read arbitrarily many symbols from the input, and does not modify the stack. Then, when only  $c$  symbols are left on the input stream, the PDA requires that the symbol on the stack must differ from the one on the input. Finally, the PDA reads the remaining  $c - 1$  symbols and accepts if the stack is empty.

Note that this is again a non-deterministic PDA, as we do not know the value of  $c$ .

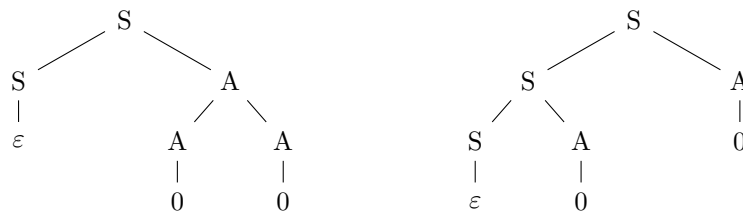


## 6 Ambiguity

- a)  $\varepsilon, 0, 00, (), (0), 0(), ()0, 000$   
 b) It is ambiguous, because the word  $00$  has two different leftmost derivations.

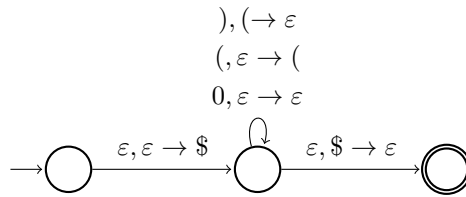
$$\begin{array}{ll}
 S \rightarrow SA & S \rightarrow SA \\
 \rightarrow A & \rightarrow SAA \\
 \rightarrow AA & \rightarrow AA \\
 \rightarrow 0A & \rightarrow 0A \\
 \rightarrow 00 & \rightarrow 00
 \end{array}$$

It can also be seen by taking a look at these two derivation trees that both belong to the word  $00$ :



Because the two derivation trees are structurewise different, the word  $00$  can be derived ambiguously from  $G$ .

c) A simple non-deterministic PDA for  $L(G)$  looks as follows:



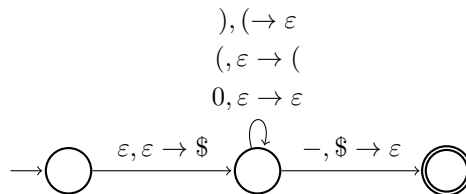
**Deterministic PDAs**

A push-down automaton  $M$  is *deterministic* iff in each state, there is exactly one successor state for every combination  $(a, b) \in \Sigma \times \Gamma$  where  $\Sigma$  is the string input alphabet and  $\Gamma$  is the stack alphabet. Note that if a state  $q$  has only one outgoing transition  $\langle \varepsilon, \varepsilon \rightarrow \$ \rangle$  the PDA is still deterministic since there is no ambiguity of what the successor state of  $q$  will be. If a state  $q$ , however, has two outgoing transitions,  $\langle a, \varepsilon \rightarrow x \rangle$  and  $\langle \varepsilon, b \rightarrow y \rangle$  leading into different states, it is unclear which transition the system should take if the string input in state  $q$  is  $\langle a \rangle$  and the top element on the stack is  $\langle b \rangle$ . A PDA containing such ambiguous transitions is *not* deterministic.

Unlike in deterministic finite automata, we take the liberty of omitting transitions leading to an (imaginary) fail state as well as the fail state itself when drawing deterministic PDAs.

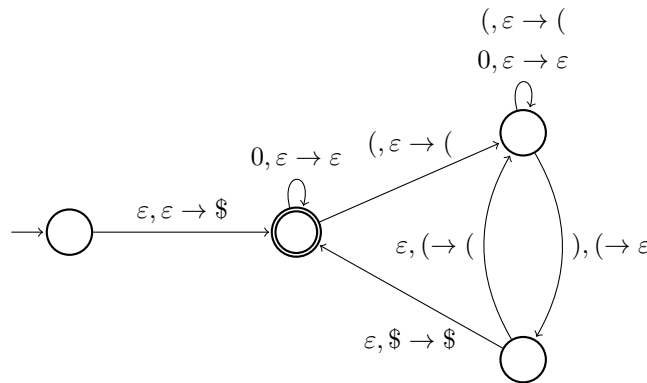
Considering this, the PDA given above is not deterministic: From the middle state, there are two transitions  $\langle (, \varepsilon \rightarrow ( \rangle$  and  $\langle \varepsilon, \$ \rightarrow \varepsilon \rangle$ , such that we do not know which one to take if we read a  $\langle ( \rangle$  while the top element on the stack is  $\langle \$ \rangle$ . We can overcome this problem in different ways.

If we assume that our PDA recognizes the end of the input string (denoted by  $\langle - \rangle$ ), it is easy to transform the non-deterministic PDA above into a deterministic one:



If we assume that the PDA is not able to determine the end of the input, it is not that easy to derive the deterministic PDA from the non-deterministic one.

An example of a deterministic PDA accepting  $L(G)$  is the following:



The deterministic PDA using as few states as possible is the following:

