Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2011 Prof. R. Wattenhofer / T. Langner, J. Seidel, J. Smula

Discrete Event Systems

Solution to Exercise Sheet 4

1 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 $|xy| \le p$ and $|y| \ge 1$ (pumping lemma). Because of $|xy| \le p$, xy can only consist of 1s. According to the pumping lemma, we should have $xy^iz \in L$ for all $i \ge 0$. However, by choosing $i = 0$ we delete at least one 1 and get a word $w' = 1^{p-|y|}02^p$ with $|y| \ge 1$. 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 $L = \{0^n 1, n \ge 1\}$ which is clearly regular. A context-free grammar for this language uses only the production $S \to 0S \mid 1$.

2 Context-Free Grammars

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

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

Note: The language is regular!

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

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

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

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

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.

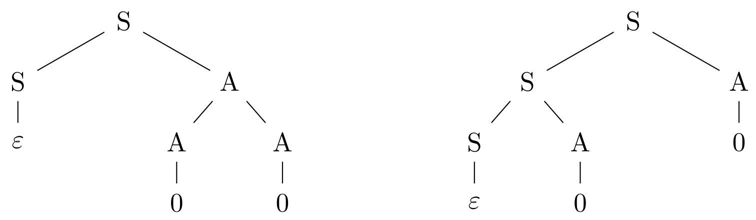
3 Pushdown Automata

a) $\varepsilon, 0, 00, (), (0), 0(), ()0, 000$

b) It is ambiguous, because the word 00 has two different leftmost derivations.

$$\begin{aligned} S &\to SA \\ &\to A \\ &\to AA \\ &\to 0A \\ &\to 00 \end{aligned} \qquad\qquad \begin{aligned} S &\to SA \\ &\to SAA \\ &\to AA \\ &\to 0A \\ &\to 00 \end{aligned}$$

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.

Ambiguity of Grammars

Definition: A string s is derived *ambiguously* in a context-free grammar G if it has two or more different leftmost/rightmost derivations (or two structurewise different derivation trees). Grammar G is *ambiguous* if it generates some string ambiguously.

A *leftmost/rightmost* derivation replaces in every step the leftmost/rightmost variable.

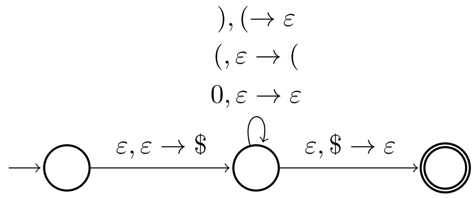
Example: The grammar with the productions '$S \to S \cdot S \mid S + S \mid a$' is ambiguous since the string $s = a \cdot a + a$ has two different leftmost derivations.

$$\begin{aligned} S &\to S \cdot S \\ &\to a \cdot S \\ &\to a \cdot S + S \\ &\to a \cdot a + S \\ &\to a \cdot a + a \end{aligned} \qquad\qquad \begin{aligned} S &\to S + S \\ &\to S \cdot S + S \\ &\to a \cdot S + S \\ &\to a \cdot a + S \\ &\to a \cdot a + a \end{aligned}$$

Intuitively, the derivation on the left corresponds to the arithmetic expression $a \cdot (a+a)$ because we first derive a product and then substitute one factor by a sum while the derivation on the right corresponds to $(a \cdot a) + a$ because we first have a sum and then substitute one summand by a product.

The productions of an equivalent non-ambiguous grammar are $A \to S + a \mid S \cdot a \mid a$.

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



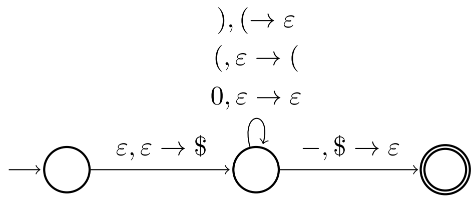
Deterministic PDAs

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 Σ is the string input alphabet and Γ is the stack alphabet. Note that if a state q has only one outgoing transition '$\varepsilon, \varepsilon \to \$$' 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, '$a, \varepsilon \to x$' and '$\varepsilon, b \to y$' leading into different states, it is unclear which transition the system should take if the string input in state q is 'a' and the top element on the stack is 'b'. A PDA containing such ambiguous transisitions 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.

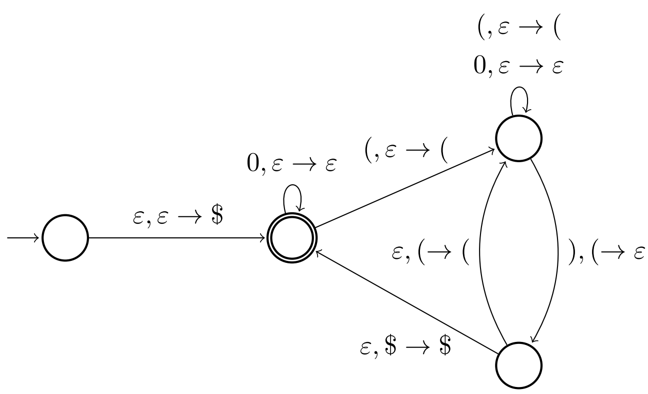
Considering this, the PDA given above is not deterministic: From the middle state, there are two transitions '$(, \varepsilon \to ($' and '$\varepsilon, \$ \to \varepsilon$', such that we do not know which one to take if we read a '(' while the top element on the stack is '\$'. We can overcome this problem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

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

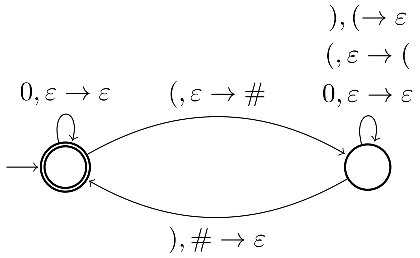


If we assume that the PDA is not able to determine the end of the input, it is not that easy to derive the deterministic PDA from the non-deterministic one.

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



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



4 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 function $f : \mathbb{N} \mapsto \mathbb{N}$ with $f(n) \mapsto \lfloor \sqrt{n} \rfloor$ and observe that $\{f(n) \mid n \in \mathbb{N}\} = \mathbb{N}$. Hence we have

$$L = \{1^{\lfloor \sqrt{n} \rfloor} \mid n \in \mathbb{N}\} = \{1^n \mid n \in \mathbb{N}\} = \{1^+\}$$

which is obviously regular.

c) Consider the alphabet $\Sigma = \{a_1, a_2, ..., a_n\}$ and the language $L = \bigcup_{i-1}^{n} a_i^*$. 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 $n+1$ states to distinguish the n different characters of the alphabet. Thus, for the DFA, we cannot deduce any information from p about the minimum number of states.

The same argument holds for the NFA.