Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Networked Systems Group (NSG)

HS 2024

Prof. L. Vanbever / R. Schmid
based on Prof. R. Wattenhofer's material

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

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

Note: The language is regular!

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

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

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

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

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.

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.

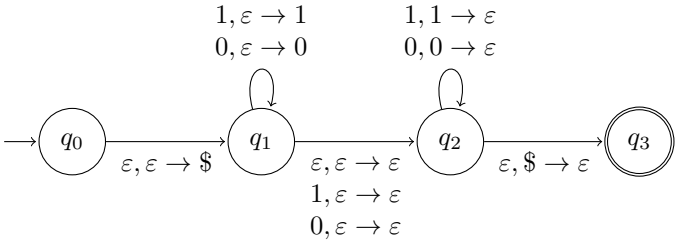
Choose $w = 1^p02^p \in L(G)$. Let $w = xyz$ with $|xy| \leq p$ and $|y| \geq 1$ (pumping lemma). Because of $|xy| \leq p$, xy can only consist of 1s. According to the pumping lemma, we should have $xy^iz \in L$ for all $i \geq 0$. However, by choosing $i = 0$ we delete at least one 1 and get a word $w' = 1^{p-|y|}02^p$ 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 $L = \{0^n1, n \geq 0\}$ which is clearly regular. A context-free grammar for this language uses only the production $S \to 0S \mid 1$.

3 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.

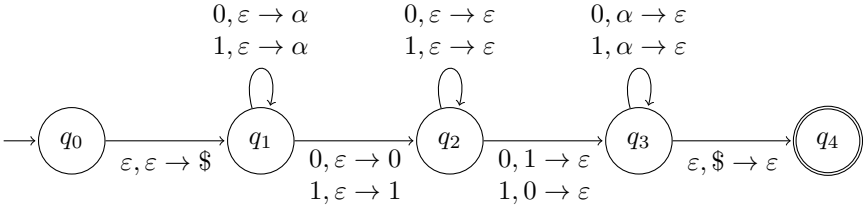
Note that we need to support words of even and odd length. Words of even length have a counterpart for each letter. However, the center letter of an odd word has no counterpart.



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 α 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.



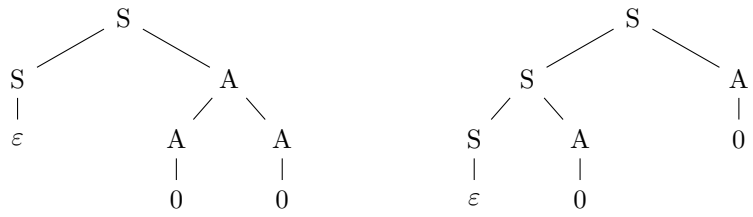
4 Ambiguity

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

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

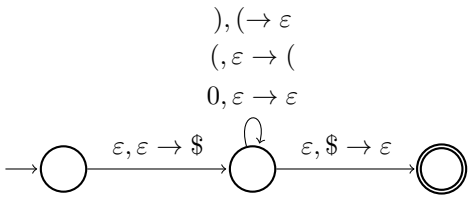
$$
\begin{aligned}
S &\to SA \\
&\to A \\
&\to AA \\
&\to 0A \\
&\to 00
\end{aligned}
\qquad\qquad
\begin{aligned}
S &\to SA \\
&\to SAA \\
&\to AA \\
&\to 0A \\
&\to 00
\end{aligned}
$$

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.

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

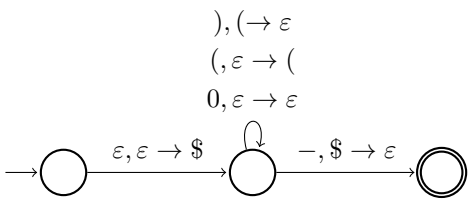


Deterministic PDAs

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.

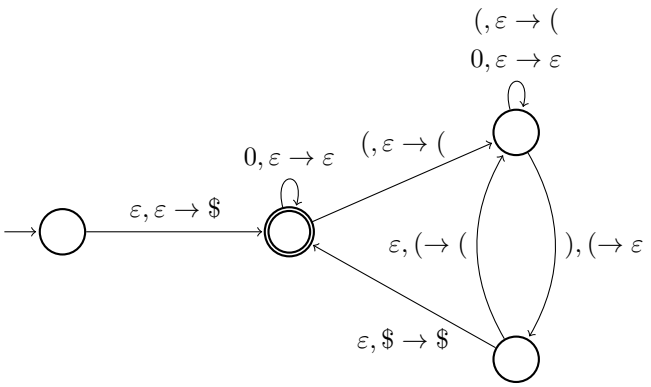
Considering this, the PDA given above is not deterministic: From the middle state, there are two transitions '$(, \varepsilon \to ($' and '$\varepsilon, \$ \to \varepsilon$', such that we do not know which one to take if we read a '(' while the top element on the stack is '$'. We can overcome this problem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

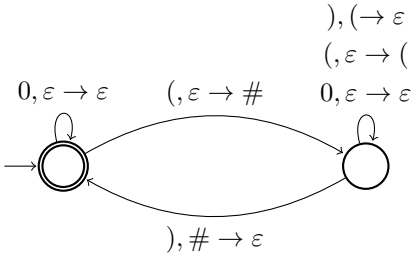


If we assume that the PDA is not able to determine the end of the input, it is not that easy to derive the deterministic PDA from the non-deterministic one.

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



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



5 Counter Automaton

a) A counter automaton is basically a finite automaton augmented by a counter. For every regular language $L \in \mathcal{L}_{reg}$, there is a finite automaton A which recognizes L. We can construct a counter automaton C for recognizing L by simply taking over the states and transitions of A and *not* using the counter at all. Clearly C accepts L. This holds for every regular language and therefore, $\mathcal{L}_{reg} \subseteq \mathcal{L}_{count}$.

b) Consider the language L of all strings over the alphabet $\Sigma = \{0, 1\}$ with an equal number of 0s and 1s. We can construct a counter automaton with a single state q that increments/decrements its counter whenever the input is a 0/1. If the value of the counter is equal to 0, it accepts the string. Hence, L is in $\mathcal{L}_{count}$. On the other hand, it can be proven (using the pumping lemma) that L is not in $\mathcal{L}_{reg}$ and it therefore follows $\mathcal{L}_{count} \nsubseteq \mathcal{L}_{reg}$.

Some languages where the (non-finite) frequency of one or several symbols depends on the frequency of other symbols can be recognized by counter automata. Such languages cannot be recognized by finite automata.

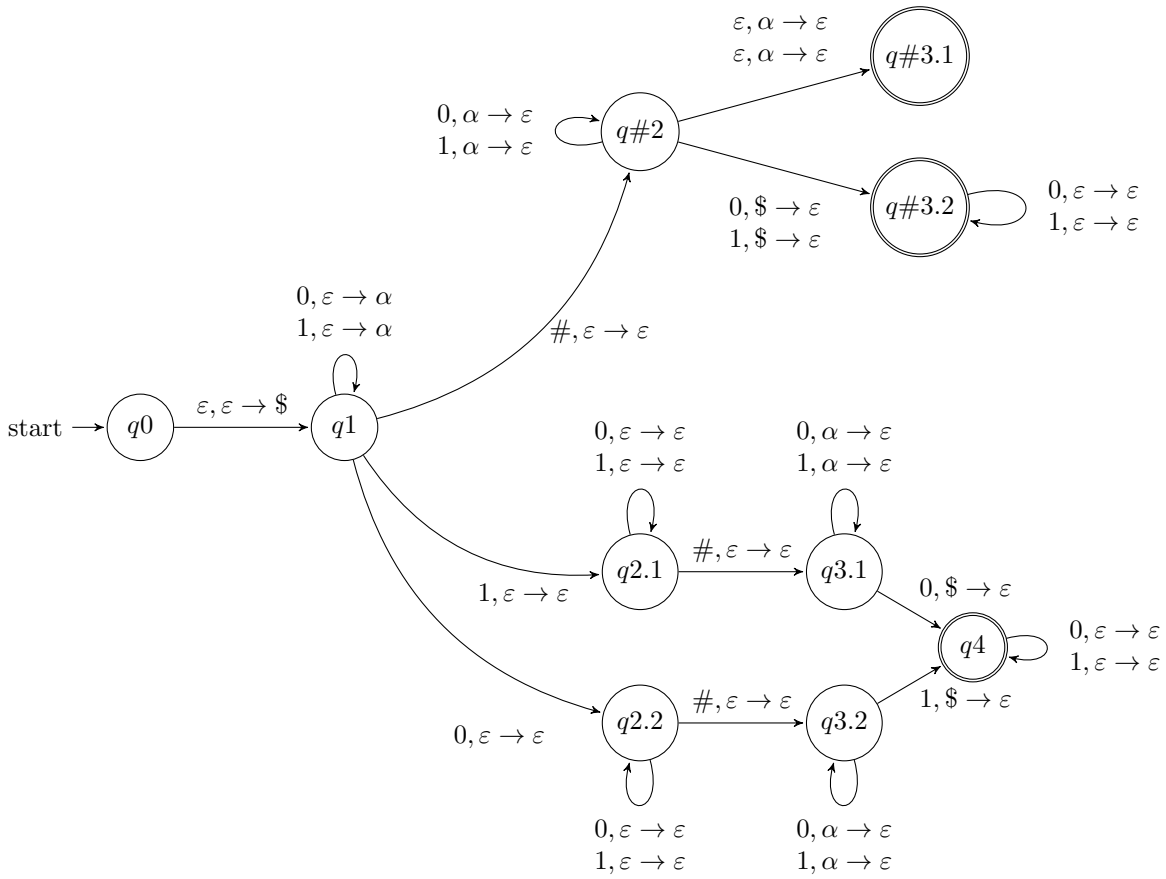
c) First, we show that a pushdown automaton can simulate a counter automaton. Hence, PDAs are at least as powerful as CAs! The simulation of a given CA works as follows. We construct a PDA which has exactly the same states as the CA. The transitions also remain between the same pairs of states, but instead of operating on an INC/DEC register, we have to use a stack. Concretely, we store the state of the counter on the stack by pushing '+' and '−' on the stack. For instance, a counter value '3' is represented by three '+' on the stack, and similarly a value '−5' by five '−'. Therefore, when the CA checks whether the counter equals 0, the PDA can check whether its stack is empty.

In the following, we give just one example of how the transitions have to be transformed. Assume a transition of the counter automaton which, on reading a symbol s, increments the counter—independently of the counter value. For the PDA, we can simulate this behavior with three transitions: On reading s and if the top element of the stack is '−', a minus is popped; if the top element is a '+', another '+' is pushed; and if the stack is empty, also a '+' is pushed.

Hence, we have shown that the PDA is at least as powerful as the CA, and it remains to investigate whether both CA and PDA are equivalent, or whether a PDA is stronger. Although it is known that the PDA is actually more powerful, the proof is difficult: There is no pumping lemma for CAs for example such that we can prove that a given context-free language cannot be accepted by a CA. However, of course, if you have tackled this issue, we are eager to know your solution... :-)

6 Inequality-checking with PDAs [Exam HS20]

A PDA recognizing $L = \{x\#y \mid x, y \in \{0,1\}^+,\ x \neq y\}$ could look like this:



We make use of non-determinism for two decisions: First, we decide whether to check that the lengths $|x|$ and $|y|$ differ, i.e. whether $|x| > |y|$ ($\rightarrow q\#3.1$) or $|x| < |y|$ ($\rightarrow q\#3.2$). If they do not differ by length, we use non-determinism to fix some position in x that is a 1 (or a 0), and then we make sure that the same digit in y is a 0 (or a 1; $\rightarrow q4$). As both $x, y \in \{0,1\}^+$ and have the same length, there must always exist such a position.

Note that the presented automaton is not minimal, as states $q4$ and $q\#3.2$ are indistinguishable. However, we leave the presentation "as is" for clarity of the construction idea.