René Maseli
Prof. Dr. Roland Meyer

Theoretical Computer Science 1
Exercise Sheet 6

TU Braunschweig
Winter semester 2021/22

Release: 01.02.2022 Due: 10.02.2022, 23:59

Hand in your solutions per e-mail or StudIP to your tutor until Thursday, 10.02.2022 23:59 pm. You should provide your solutions either directly as .pdf file or as a readable scan/photo of your handwritten notes.

This is the last exercise sheet of the semester. However, all coming lecture content is still relevant for the written exam.

Aufgabe 1: Closure properties [9 points]

a) [4 points] It is known, that the language $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ is not context-free. Show that its complement $\overline{L}$, however, is indeed context-free. With this, $\overline{L}$ is an example of a context-free language whose complement is not context-free.

   Hint: There can be multiple reasons why a word w is not contained in L. Since context-free languages are closed under unions, it is sufficient to consider each of these reasons separately and show that they are context-free.

b) For any word $w = w_1 w_2 \dots w_{n-1} w_n$ we define $w^R = w_n w_{n-1} \dots w_2 w_1$. For any language L let $L^R = \{w^R \mid w \in L\}$.

   - [2 points] Show how to construct from an NFA A a new NFA B satisfying $\mathcal{L}(B) = \mathcal{L}(A)^R$.
   - [2 points] Now show how to construct a context-free grammar H from a context-free grammar G such that $\mathcal{L}(H) = \mathcal{L}(G)^R$ holds.
   - [1 point] From the above two points and the fact that right-linear grammars produce regular languages (see previous exercise sheet), deduce that left-linear languages produce regular languages as well.

Aufgabe 2: CFG, CNF, CYK [8 points]

The Cocke-Younger-Kasami-algorithm (CYK algorithm) assumes as input a context-free grammar (CFG) in Chomsky normal form (CNF). This means that all production rules are of the form $X \to YZ$ (for non-terminals Y and Z) or of the form $X \to a$ (for a terminal a).

Given the two CFG $G = \langle \{S, X, Y\}, \{a, b\}, P_G, S \rangle$ and $H = \langle \{S, X, Y\}, \{a, b, c\}, P_H, S \rangle$.

P_G :	$S \to XaY \mid \varepsilon$	P_H :	$S \to X \mid aba$
	$X \to bXYb \mid Y$		$X \to YYa \mid ab$
	$Y \to aXYb \mid \varepsilon$		$Y \to S \mid c$

a) [1 point] Use the procedure introduced in the lecture to construct a grammar G_a without ε productions that is language-equivalent to G.

b) [1 point] Use G_a and the procedure from the lecture to construct a grammar G_b in CNF with $\mathcal{L}(G_b) = \mathcal{L}(G)$.

c) [1 point] Use G_b and the CYK algorithm to decide whether the word *bbaab* is produced by G.

d) [2 points] Use G_b and the CYK algorithm to decide whether $ababbaa \in \mathcal{L}(G)$ is true.

e) [1 point] Use the procedure from the lecture to construct an H-language-equivalent grammar H_e in CNF.

f) [1 point] Use H_e and the CYK algorithm to decide whether the word *ccaba* is produced by H.

g) [1 point] Prove or disprove $cabaa \in \mathcal{L}(H)$ by using H_e and the CYK algorithm.

Aufgabe 3: The syntax of programming languages as grammar [8 points]

The syntax of a programming language is usually formulated with a context-free grammar (oftentimes expressed in EBNF or a syntax diagram). In this exercise you will construct a grammar which describes the syntax of a simple programming language.

a) [2 points] Give a context-free grammar G such that its language $\mathcal{L}(G)$ consists of the set of syntactically correct programs as described below.

- Use the terminals Σ := {`id`, `num`, `var`, `if`, `then`, `else`, `end`, `while`, do, `;`, `op`, =, `(`, `)`}.

  `id`, `num` and op are placeholders for possible variable names, natural numbers and binary operators (including ==). The other symbols represent single keywords and symbols.

- An **expression** consists of variables, numbers and operations that combine them such as z.B. `(x+2)`, `(z<500)`, `(x*(y/3))`, `(x==(y+1))`. Binary operations are always parenthesized.
- A **program** is either
  - empty
  - a variable declaration (z.B. `var x`)
  - an assignment of an expression to a variable (z.B. `x=(x+5)`)
  - a conditional statement (z.B. `if x then y=(z/x) end`)
  - a case distinction (z.B. `if x then y=(z/x) else y=z end`)
  - a loop (z.B. `while x do x=(x-1) end`)
  - a `;`-delimited sequence of programs (z.B. `var x; x=500`)

b) [2 points] Derive the following program from your grammar in part a) starting from the initial symbol. Give the complete derivation sequence.

```
var x; x=10; var y; y=0; while x do x=(x-1); y=((y*2)+1); end
```

(First you have to replace each variable with id and each number with num.)

c) [2 points] Prove that $\mathcal{L}(G)$ is not regular.

d) [2 points] Modify G such that its programming language supports functions.

A **function** starts with the keyword `function`, a function name and a `,`-delimited list of parameters (potentially empty), followed by the function body (a program) and the keyword end. Inside the function body, the return statement (z.B. `return (x*x)`) may appear.

Function calls may be used as statements and as expressions in the program.

For example, the following word should be a valid program:

```
function f (x) var y; y=2; return g(x,y) end;

function g (x,y) if (x<y) return x else return y end end;

f(4);
```

Aufgabe 4: Pushdown automata [10 points]

Construct pushdown automata for the following languages and state which acceptance condition (empty stack or final states) you assume.

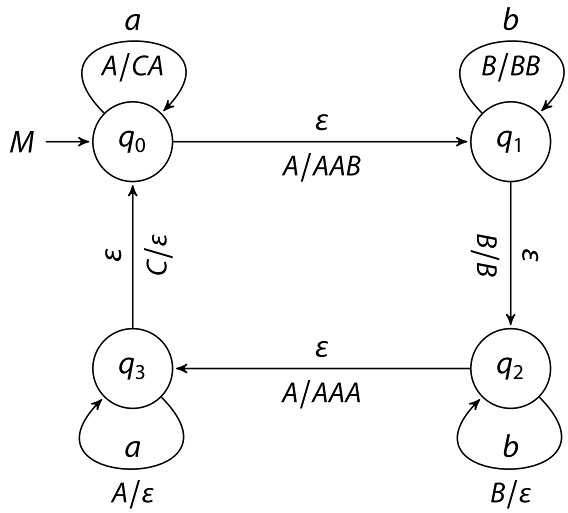
1. [2 points] $L_1 = \{w \in \{a, b, (,)\}^* \mid w \text{ is correctly parenthesized }\}$.

2. [2 points] $L_2 = \{w \in \{a, b, (,)\}^* \mid |w|_a = 2|w|_b\}$.

3. [1 point] $L_1 \cap L_2 = \{w \in \{a, b, (,)\}^* \mid |w|_a = 2|w|_b \text{ and } w \text{ is correctly parenthesized }\}$

   Can you construct a PDA, which accepts $L_1 \cap L_2$? If not, what is the intuitive problem here?

4. [5 points] Consider the following PDA $M = \langle \{q_0, q_1, q_2, q_3\}, \{a, b\}, \{A, B, C\}, A, q_0, \delta \rangle$, which accepts on empty stack. Using the triple construction, give a context-free grammar for $\mathcal{L}(M)$. Give only the useful non-terminals.



Hint: There are 8 useful non-terminals with 10 productions.