René Maseli
Prof. Dr. Roland Meyer

Theoretical Computer Science 1
Exercise Sheet 6

TU Braunschweig
Winter semester 2022/23

Release: 24.01.2023 Due: 03.02.2023, 09:45

Hand in your solutions to the Vips directory of the StudIP course until Friday, 03.02.2023 09:45 pm. You should provide your solutions either directly as .pdf file or as a readable scan/photo of your handwritten notes. Submit your results as a group of four.

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

Exercise 1: 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 $\Sigma := \{$`id`, `num`, `var`, `if`, `then`, `else`, `end`, `;`, `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 parenthesized binary operations such as i.e. `(x+2)`, `(z<500)`, `(x*(y/3))`, `(x==(y+1))`.

- A **program** is either
  - empty
  - a variable definition (i.e. `let x = (y + 1)`)
  - a conditional statement (i.e. `if x then let y=(z/x) end`)
  - a case distinction (i.e. `if x then let y=(z/x) else let y=z end`)
  - a `;`-delimited sequence of programs (i.e. `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.
`let x=10; let y=0; if (y < x) then let y=((y*2)+1) end`
(First you have to replace each variable with `id` and each number with `num` and the operations by op.)

c) [2 points] Use the pumping lemma to prove that $\mathcal{L}(G)$ is not regular.

d) [2 points] Modify G to another grammar 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 (i.e. `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);
```

Exercise 2: CFG, CNF, CYK [10 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 $G = \langle \{S, W, X\}, \{a, b\}, P_G, S \rangle$ and $H = \langle \{S, U, V\}, \{a, b, c\}, P_H, S \rangle$.

$$\begin{aligned} P_G : S &\to \varepsilon \mid bW \\ W &\to a \mid XXb \\ X &\to SS \mid ab \end{aligned}$$

a) [1 point] Use the procedure introduced in the lecture to construct a grammar G_a without ε productions, which satisfies $\mathcal{L}(G_a) = \mathcal{L}(G) \setminus \{\varepsilon\}$.

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

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

d) [3 points] Use G_b and the CYK algorithm to decide whether $bbababb \in \mathcal{L}(G)$ is true.

$$\begin{aligned} P_H : S &\to UVab \mid bU \\ U &\to aV \mid aUSc \\ V &\to \varepsilon \mid bSc \mid U \end{aligned}$$

e) [1 point] Use the procedure from the lecture to construct a grammar H_e in CNF, that satisfies $\mathcal{L}(H_e) = \mathcal{L}(H) \setminus \{\varepsilon\}$.

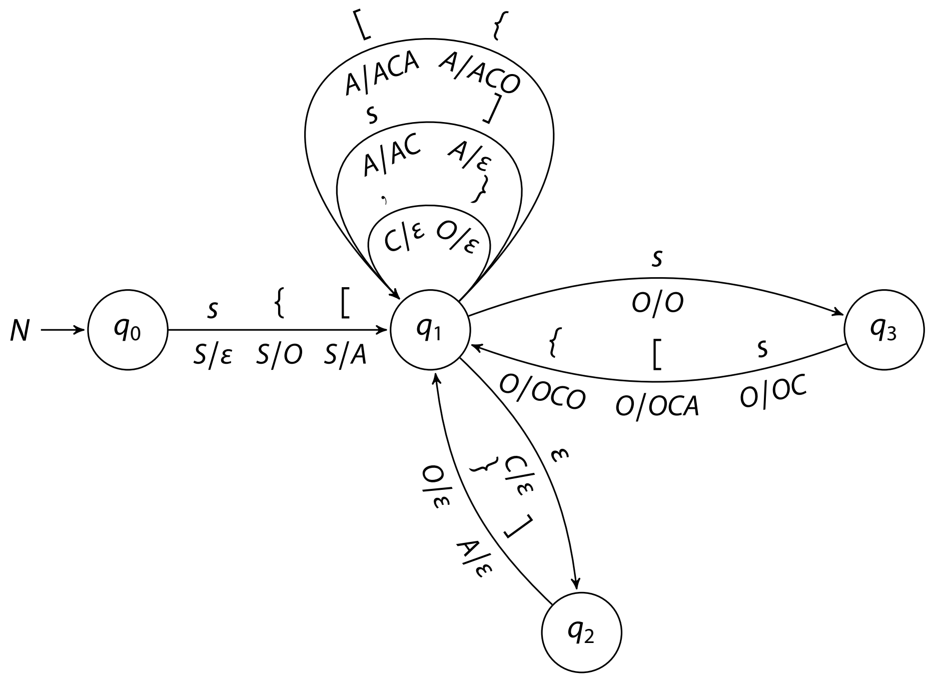
f) [3 points] Use H_e and the CYK algorithm to decide whether the word $aaabca$ is produced by H.

Exercise 3: Pushdown automata [10 points]

JavaScript Object Notation (JSON) is a description language for structured collections of serializable data, which is applied in numerous web technologies. Alongside some primitive datatypes, they can also express lists (arrays) and associative containers (objects).

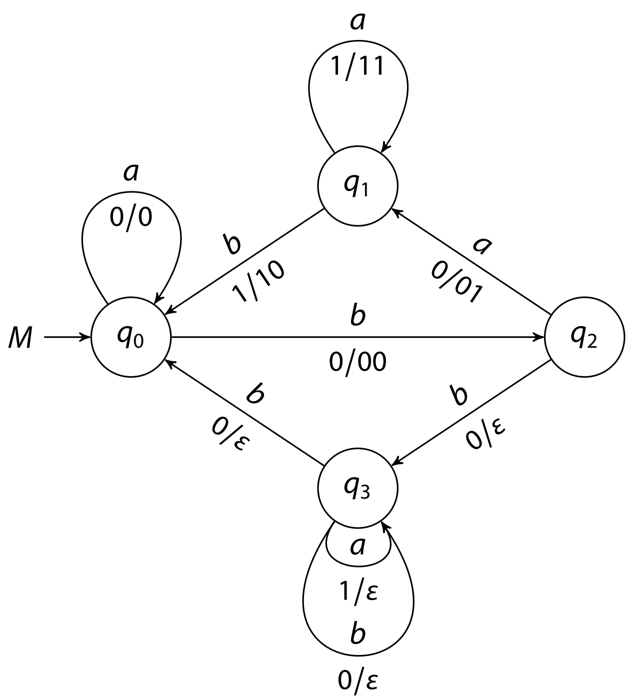
a) [5 points] Construct pushdown automata M for the following language L and state which acceptance condition (empty stack or final states) you assume. Do not just give context-free grammars. You do not need to prove the correctness of your construction.
Consider a simplified variant of JSON over $\{a, b, \{, \}\}$: ,Objects' start and end with fitting curly braces ,{' and ,}'. Inside, there is an arbitrary number of key-value pairs. Keys are words of $a.b^*$ and may not be unique inside the same object. Values are either words of $a.b^*$, or again objects. The automaton M shall accept exactly the well-formed objects.
For example, $\{abbababb\{\}\} \in L$ and $\{abb\{ab\{aba\}a\{abbab\}\}\} \in L$ have to be accepted, but neither $\{ababab\} \notin L$, $abb\{aa\} \notin L$ nor $\{ab\{\} \notin L$.

b) [5 points] Describe the behavior of the following pushdown automaton N, with empty-stack acceptance, by explaining the role of all states and stack symbols with one sentence, each.



Exercise 4: Triple Construction [7 points]

Consider the Pushdown Automaton $M = \langle\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{0, 1\}, q_0, 0, \delta\rangle$, with empty-stack acceptance and whose transition relation δ is given by the following diagram.



a) [1 point] Consider just the states on their own, to answer those two questions. Which two states are befitting destinations $q \in Q$ in triples like $\langle p, s, q\rangle$? Which five pairs of $p \in Q$ and $s \in \Gamma$ are enabled?

b) [6 points] Find a contextfree grammar G with $\mathcal{L}(M) = \mathcal{L}(G)$, by using the triple construction from the lecture.