Theoretical Computer Science
Exercise Sheet 6

Prof. Dr. Roland Meyer
René Maseli
TU Braunschweig
Winter Semester 2025/26

Release: 2026-01-17 Due: 2026-01-29 23:56

Homework Exercise 6.1: The syntax of programming languages as grammar [4 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) [1 point] 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 $\Sigma := \{\texttt{id}, \texttt{num}, \texttt{;}, \texttt{op}, \texttt{=}, \texttt{(}, \texttt{)}, \texttt{if}, \texttt{else}, \texttt{while}, \texttt{end}, \texttt{break}\}$.
- An **expression** consists of variables, numbers and parenthesized binary operations, e.g. `id`, `(id op num)`, `(id op id)`, `(id op (id op num))`, `(num op (num op num))`.
- A **program** is either
  - empty
  - a variable definition (e.g. `id=(id op num)`)
  - a conditional branching (e.g. `if id id=(id op id) else id=id end`)
  - a loop (e.g. `while (id op num) id=(id op num) end`)
  - a break `break` **but only inside a loop**
  - a `;`-delimited sequence of programs (e.g. `id=num ; id=num`)

b) [1 point] Derive the following program from your grammar in part a) starting from the initial symbol. Additionally to the start symbol and the resulting word, give at least three intermediate words of your derivation sequence.

```
while(id op id) id=(id op num); if(id op id) break else id=(id op num) end end
```

c) [1 point] Modify G into another grammar G', such that the programming language prohibits obvious dead code: `break` jumps out of the current program sequence, so it should not preceed any other program instruction. This also applies to conditional branches, if both subprograms end with `break`. This can even nest indefinetly.

d) [1 point] Modify G to another grammar G', such that its programming language supports functions.

A **function** starts with the keyword `fn`, 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:

```
fn id(id) id=num; return id(id,id) end;
fn id(id,id) if(id op id) return id else return id end end;
id(num);
```

Homework Exercise 6.2: CFG, CNF, CYK [5 points]

Let $G = \langle N, \Sigma, S, P \rangle$ a context-free grammar. You eliminate unit rules by computing the least fixed point of the following transformation $F : \mathcal{P}(N \times (N \cup \Sigma)^*) \to \mathcal{P}(N \times (N \cup \Sigma)^*)$ with $F(X) := P \cup \{ X \to \beta \mid (X \to Y), (Y \to \beta) \in X \}$.

a) [2 points] Let $G = \langle N, \Sigma, S, P \rangle$ context-free. Show that the grammar $G'' = \langle N, \Sigma, S, P'' \rangle$ without unit rules, i.e. with $P'' := \mathrm{lfp}(F) \setminus \{ X \to Y \mid X, Y \in N \}$, fulfills the equation $\mathcal{L}(G) = \mathcal{L}(G'')$.

b) [1 point] Consider the following grammar G_b. Remove **at first** all unreachable non-terminals, **then** all unproductive non-terminals and write down the resulting grammar. Are all remaining non-terminals useful?

$S \to XY \mid bVc$ $\quad T \to cTb \mid ZaZ \mid V$ $\quad U \to a \mid bWb$ $\quad V \to a \mid VVV \mid VXV \mid \varepsilon$

$W \to cTUb \mid ZbZ \mid V$ $\quad X \to aZb \mid cX$ $\quad Y \to b \mid TV$ $\quad Z \to bX \mid XX \mid aZ$

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).

c) [1 point] Use the procedure introduced in the lecture to construct a grammar G_c' in CNF, which satisfies $\mathcal{L}(G_c') = \mathcal{L}(G_c) \setminus \{\varepsilon\}$. The grammar $G_c = \langle \{S, X, Y\}, \{a, b, c\}, P_c, S \rangle$ shall be defined by the following productions:

$S \to XcX$ $\quad X \to a \mid YX \mid YbYb$ $\quad Y \to bc \mid X \mid XX$

d) [1 point] Use the CYK Algorithm to check if the word *bbacbbc* can be produced by the following grammar $G_d := \langle \{S, T, U, A, B, C\}, \{a, b, c\}, P_d, S \rangle$:

$S \to BA \mid BC$ $\quad T \to CT \mid BS$ $\quad U \to TA$

$A \to a \mid AC \mid BB$ $\quad B \to b \mid UU$ $\quad C \to c \mid CT$

Homework Exercise 6.3: Pushdown automata [4 points]

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

Remark: Note that whenever multiple symbols are pushed, the last gets to be the new top.

$$L := \left\{ w \in (bab)^* \sqcup\!\sqcup (ac)^* \;\middle|\; \forall x, y \in \{a, b, c\}^* : w = x.y \Rightarrow |x|_b \leq |x|_c \right\}$$

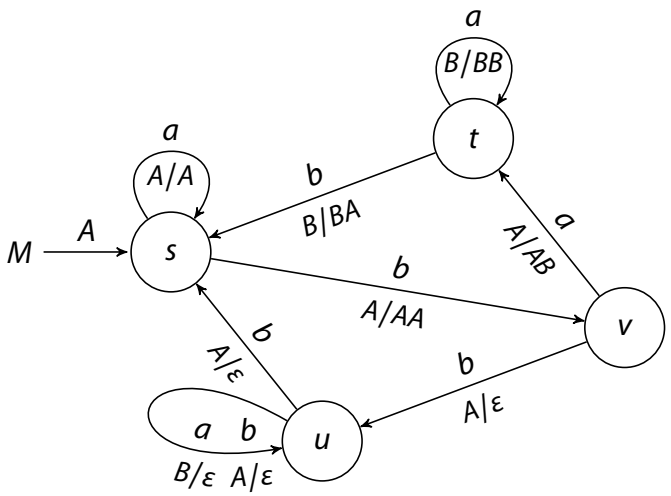
Hereby, $\sqcup\!\sqcup$ denotes the ridge shuffle presented in Tutorium 3. The second constraint describes, that each prefix of an accepted word shall contain at most as many b's as c's.

a) [2 points] Draw the state chart of a pushdown automaton that accepts L exactly.

b) [2 points] Describe your automaton in three sentences.

Homework Exercise 6.4: Triple Construction [3 points]

Consider the Pushdown Automaton $M = \langle \{s, t, u, v\}, \{a, b\}, \{A, B\}, s, A, \delta \rangle$, 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) [2 points] Find a context-free grammar G with $\mathcal{L}(M) = \mathcal{L}(G)$, by using the triple construction from the lecture.

Homework Exercise 6.5: Pumping-Lemma for context-free languages [4 points]

Make use of the Pumping Lemma, to show, that the following languages are not context-free:

a) [2 points] $L_1 = \{ a^n b a^{2n} b a^{3n} b a^{4n} \mid n \in \mathbb{N} \} \subseteq \{a, b\}^*$.

b) [2 points] $L_2 = \{ u\#v\#w \mid u, v, w \in \{a, b\}^* \text{ and } (u = w \text{ or } v = w) \} \subseteq \{a, b, \#\}^*$.

Tutorial Exercise 6.6:

Given the context-free grammar $G = \langle \{S, W, X\}, \{a, b\}, P, S \rangle$ with the following productions

$$S \to \varepsilon \mid bW \qquad W \to a \mid XXb \qquad X \to SS \mid ab \,.$$

a) 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) 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) Use G_b and the CYK algorithm to decide whether the word *babba* is produced by G.

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

Consider the following grammar G_e.

$$S \to UVab \mid bU \qquad U \to aV \mid aUSc \qquad V \to \varepsilon \mid bSc \mid U$$

e) Use the procedure from the lecture to construct a grammar G_e' in CNF, that satisfies $\mathcal{L}(G_e') = \mathcal{L}(G_e) \setminus \{\varepsilon\}$.

f) Use G_e' and the CYK algorithm to decide whether the word *aaabca* is produced by G_e.

g) Use the CYK algorithm to decide, whether the words *babaa* and *baba* are produced by the grammar with the following productions.

$$S \to AB \mid BC \qquad A \to a \mid CC \qquad B \to b \mid BA \qquad C \to a \mid AB$$

Tutorial Exercise 6.7:

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

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

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

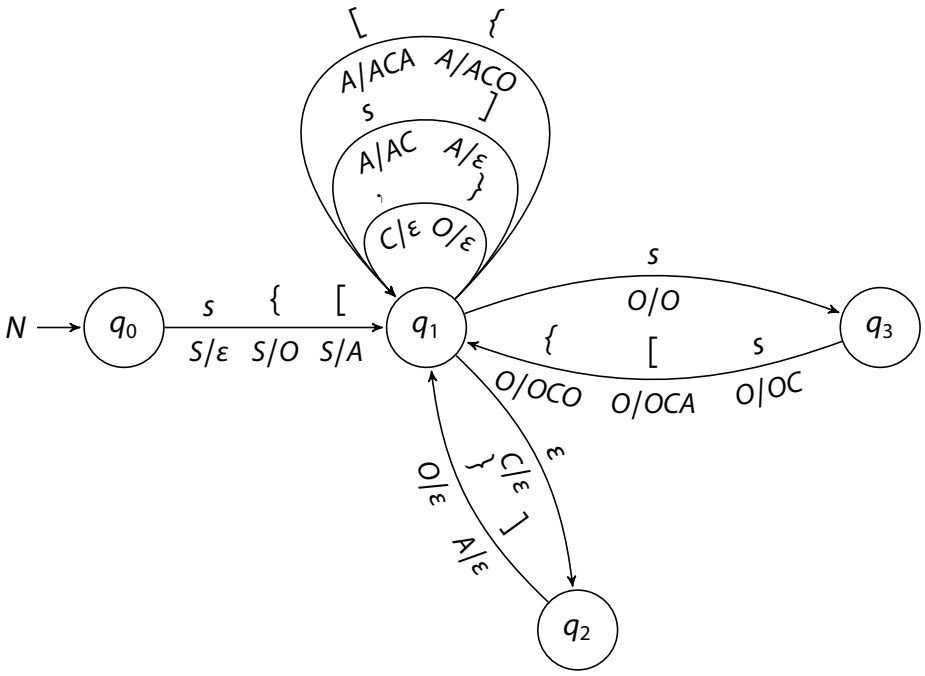
c) Can you construct a PDA, which accepts $L_1 \cap L_2$?
If possible, briefly explain its functionality. If not, what is the intuitive problem here?

$$L_1 \cap L_2 = \left\{ w \in \{a, b, (,)\}^* \;\middle|\; |w|_a \leq 2|w|_b \text{ and } w \text{ is correctly parenthesized } \right\}$$

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).

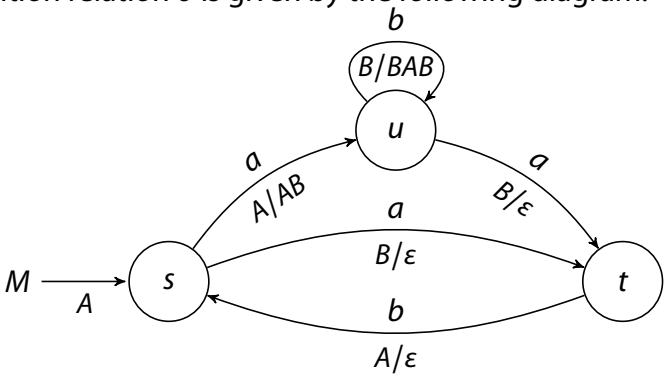
d) Construct pushdown automata M for the following language L and state which acceptance condition (empty stack or final states) you assume. Do not just give context-free grammars. You do not need to prove the correctness of your construction.
Consider a simplified variant of JSON over $\{a, b, \{, \}\}$: ‚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 $\{ab\{\} \notin L$.

e) Describe the behavior of the following pushdown automaton $N = \langle\{q_0, q_1, q_2, q_3\}, \{s, \{, \}, ,, [,]\}, \{C, A, O, S\}, q_0, S, \delta\rangle$, with empty-stack acceptance, by explaining the role of all states and stack symbols with one sentence, each.



Tutorial Exercise 6.8:

Consider the Pushdown Automaton $M = \langle\{s, t, u\}, \{a, b\}, \{A, B\}, s, A, \delta\rangle$, with empty-stack acceptance and whose transition relation δ is given by the following diagram.



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

b) Find a context-free grammar G with $\mathcal{L}(M) = \mathcal{L}(G)$, by using the triple construction from the lecture.

Tutorial Exercise 6.9:

Given a left-linear grammar $G = \langle \Sigma, N, S, P \rangle$ (with at most one non-terminal in the prefix of right sides of each production), let $A_G = \langle Q, q_0 \to, Q_F \rangle$ an NFA with ε-transitions. It consists of 'original' states $N \subset Q$, a new initial state $q_0 \in Q$, sole final state $Q_F = \{S\}$ and transitions

$$Y \xrightarrow{w_1} \cdots \xrightarrow{w_n} X \text{ iff } X \to Yw_1 \ldots w_n \in P$$
$$q_0 \xrightarrow{w_1} \cdots \xrightarrow{w_n} X \text{ iff } X \to w_1 \ldots w_n \in P$$

over additional new states, if needed.

Prove, that $\mathcal{L}(A_G) = \mathcal{L}(G)$ holds.

Tutorial Exercise 6.10:

Transform the following context-free grammar G into GNF.

$$S \to WS \mid UU \qquad U \to b \mid c \mid UW \qquad V \to c \qquad W \to a \mid VW \mid VU$$

a) For each pair of non-terminal $X \in N$ and terminal $s \in \Sigma$, give a regular expression for the language $L_{X,s} \subseteq N^*$ of the suffixes of terms $\alpha \in N^*$ which are produced by the strong left derivation: $X \Rightarrow_{SL}^* s\alpha$.

b) Find for all non-empty languages $L_{X,s}$ a right-linear grammar $G_{X,s}$ over **terminal symbols** N and with initial symbol $T_{X,s}$. Create new non-terminals, if required.

c) Transform the union of G and all your grammars into pseudo-GNF, by letting this new grammar G' guess each next terminal symbol, like has been shown in the lecture. Ensure, that $\mathcal{L}(G') = \mathcal{L}(G)$ holds. A proof is not necessary. Try to avoid useless non-terminals.

d) Eliminate all ε-productions from G', to form a grammar G'' in GNF, which satisfies $\mathcal{L}(G'') = \mathcal{L}(G') \setminus \{\varepsilon\}$.

Now consider the following grammar G in CNF with

$S \to d \mid TU$
$T \to a \mid b \mid UT$
$U \to c \mid d \mid TS \mid US$.

$L_{X,s}$	S	T	U
a	$U \cup S(S \cup TS)^*TU$	$\varepsilon \cup S(S \cup TS)^*T$	$S(S \cup TS)^*$
b	$U \cup S(S \cup TS)^*TU$	$\varepsilon \cup S(S \cup TS)^*T$	$S(S \cup TS)^*$
c	$(S \cup TS)^*TU$	$(S \cup TS)^*T$	$(S \cup TS)^*$
d	$\varepsilon \cup (S \cup TS)^*TU$	$(S \cup TS)^*T$	$(S \cup TS)^*$

e) Use the languages of side-products of the strong left derivation from the table, to construct a grammar G' in GNF with $\mathcal{L}(G') = \mathcal{L}(G)$.