Compiler Construction (WS 2017/18)

Exercise Sheet 2

Compiler Design Lab
Prof. Dr. Sebastian Hack
Fabian Ritter, B.Sc.

Exercise 2.1 Push-Down Automata

Let $(\{S, A, B, C, D, H, K\}, \{a, b, c, d, e\}, P, S)$ be a context-free grammar with the following productions P:

$$\begin{array}{lcl} S & \rightarrow & KA \mid BK \\ A & \rightarrow & abA \mid BcH \mid \varepsilon \\ B & \rightarrow & eBd \mid c \\ C & \rightarrow & dAb \mid aa \\ D & \rightarrow & S \mid \varepsilon \\ H & \rightarrow & CD \\ K & \rightarrow & cd \end{array}$$

Write down a successful run of the push-down automaton constructed for this grammar (using the algorithms presented in the lecture) on the input word $cdeecddcaaccd$.

Exercise 2.2 Grammar Ambiguity, Theory and Practice

Consider a grammar that captures a subset of the statements of C with starting non-terminal *stmt* and the following productions (expression rules are omitted for brevity):

$$\begin{array}{rcl} \textit{stmt} & \rightarrow & \textit{expr}\,\mathbf{;} \\ & | & \textbf{return}\ \textit{expr}\,\mathbf{;} \\ & | & \textbf{if}\ \mathbf{(}\textit{expr}\mathbf{)}\ \textit{stmt} \\ & | & \textbf{if}\ \mathbf{(}\textit{expr}\mathbf{)}\ \textit{stmt}\ \textbf{else}\ \textit{stmt} \\ & | & \textbf{while}\ \mathbf{(}\textit{expr}\mathbf{)}\ \textit{stmt} \\ & | & \mathbf{\{}\ \textit{stmt-seq}\ \mathbf{\}} \\ \textit{stmt-seq} & \rightarrow & \textit{stmt-seq}\ \textit{stmt} \mid \textit{stmt} \\ \textit{expr} & \rightarrow & \ldots \end{array}$$

1. Give an input token string which demonstrates that this grammar is ambiguous.
2. Adjust the grammar such that it becomes unambiguous (but still describes the same language as before). *Hint: Introduce new non-terminal symbols to group the different statements into categories and add slightly modified duplicates of some of the productions.*
3. Check how this ambiguity issue is treated in the C standard.

Exercise 2.3 LL(k)

A grammar is an LL(k)-grammar for some $k \in \mathbb{N}$ if whenever there exist $u, x, y \in V_{T^*}$ with $k : x = k : y, Y \in V_N$ and $\alpha, \beta, \gamma \in (V_T \cup V_N)^*$ such that

$$\begin{array}{ccccccc} S & \underset{lm}{\overset{*}{\Longrightarrow}} & uY\alpha & \underset{lm}{\Longrightarrow} & u\beta\alpha & \underset{lm}{\overset{*}{\Longrightarrow}} & ux \\ S & \underset{lm}{\overset{*}{\Longrightarrow}} & uY\alpha & \underset{lm}{\Longrightarrow} & u\gamma\alpha & \underset{lm}{\overset{*}{\Longrightarrow}} & uy \end{array}$$

then $\beta = \gamma$
A language L is an LL(k)-language if there exists an LL(k)-grammar that generates L.

1. Prove that for each $k \in \mathbb{N}$ there exists a grammar which is $LL(k+1)$ but not LL(k).

2. Prove that for each $k \in \mathbb{N}$ an LL(k)-grammar is an LL($k+1$)-grammar.

3. Investigate the relationship between LL(0)-languages and regular languages. In particular provide the following information.

   - $\{|x| \mid x \in \text{LL}(0)\}$, where LL(0) is the set of all LL(0)-languages.
   - $\{|x| \mid x \in L_{reg}\}$, where L_{reg} is the set of all regular language.
   - Which set relation holds between LL(0) and L_{reg}?

4. A grammar is left-recursive if it has a production of the form $A \to A\mu$. Show that a left-recursive grammar is not LL(k) for any k.

Exercise 2.4 Checkable LL(k) conditions

The formal definition of an LL(k)-grammar as given in the lecture is not very handy for checking if a given grammar is an LL(k)-grammar. Therefore the lecture about LL-parsing introduced some checkable LL(k) conditions (slides 31 and 32).

- Show that an LL(k)-grammar does in general not have to be a strong LL(k)-grammar for $k > 1$.
- Show that an LL(1)-grammar is always also a strong LL(1)-grammar. (Prove one direction of the theorem on slide 31 of the lecture about LL-parsing.)
- Provide a sufficient condition to find out if a given context-free grammar is an LL(k)-grammar. This condition should be weaker than the check if a grammar is a strong LL(k)-grammar. Give an example where your condition classifies a grammar as LL(k)-grammar even if it is no strong LL(k)-grammar. Remember that the definition of an LL(k)-grammar itself is of course also a sufficient condition, but for grammars that define infinite languages it cannot be checked.

The following exercises provide further opportunities for practicing with finite automata, item PDAs and regular expressions. They might not be discussed in full detail in the tutorials.

Exercise 2.5 Item-PDAs Revisited

Let the pushdown automaton $P = (\{a, b\}, \{q_0, q_1, q_2, q_3\}, \Delta, q_0, \{q_3\})$, where

$$\Delta = \{(q_0, a, q_0q_1), (q_0, b, q_0q_2), (q_0, \#, q_3), (q_1, a, q_1q_1), (q_1, b, \epsilon), (q_2, a, \epsilon), (q_2, b, q_2q_2)\}$$

and $\# \notin \Sigma$ symbolizes the end of the input word, be given.

Provide a context-free grammar that generates the language L accepted by P. If possible, provide also a regular expression for L. Otherwise provide sufficient arguments why this is not possible.

Exercise 2.6 Regular Expressions and Languages

The lecture defined regular expressions using the metacharacters $\underline{\emptyset}$ and $\underline{\varepsilon}$. Show that they are the neutral elements with respect to the alternative and concatenation operations in regular expressions. This means show that:

- $\underline{(}r_1\underline{|}\underline{\emptyset}\underline{)}$ describes the same language as r_1
- $\underline{(}r_1\underline{\varepsilon}\underline{)}$ describes the same language as r_1

only by reasoning about the described languages as shown in the lecture. Assume the regular expression r_1 to denote the language R_1.

Exercise 2.7 Finite Automata Reloaded

In this exercise we take a closer look at recognising common language structures like comments. Consider comments in XML which start with <!-- and end with the first occurrence of -->. However, XML comments are not nestable. So the first --> ends the comment no matter how many <!-- it contained. We can define the construct <!-- $until$ --> to describe such comments.

- Create a minimal deterministic finite automaton that accepts XML comments over an alphabet Σ, where $\{<, >, -, !\} \subseteq \Sigma$. You may label an automaton edge with $\Sigma \setminus \{x, y\}$ to express that there are in fact edges for all of the alphabet's symbols except $\{x, y\}$.