SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Reinhard Wilhelm
Prof. Dr. Sebastian Hack
Dipl.-Inform. Daniel Grund
Michael Jacobs, M.Sc.

Compiler Construction WS11/12

Exercise Sheet 3

Please hand in the solutions to the theoretical exercises until the beginning of the lecture next Friday 2011-11-11, 12:00. Please write the number of your tutorial group or the name of your tutor on the first sheet of your solution.

Exercise 3.1. Item-PDAs Revisited (Points: 4+2)

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 3.2. LL(k) (Points: 2+2+2+2)

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.
   - $\{|x| \mid x \in LL(0)\}$, 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 $A \rightarrow A\mu$. Show that a left-recursive grammar is not $LL(k)$ for any k.

Exercise 3.3. Checkable LL(k) conditions (Points: 3+4+3)

The formal definition of an $LL(k)$-grammar as given in the previous exercise 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 33 and 34).

- 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 33 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.

Project task C. Parser and AST construction

Implement a recursive descent parser for MiniJava:

- The parser must accept exactly the words of the MiniJava language, i.e. those words derivable with the grammar, G, given in the language specification.
- Also, construct a syntax tree for syntactically correct inputs.
- Check your implementation against the provided test cases and write additional test cases on your own.
- The next project task will be to pretty-print source code from the AST in two different flavors.

Before you start hacking the parser, plan ahead:

- Find the ambiguities in G and its left-recursive productions and resolve them as you deem it fit. For your revised grammar, G', determine the FiFo-sets and a k such that G' is $SLL(k)$.
- How to implement the k-lookahead capability in your lexer/parser?
- How to represent the AST? What classes and class hierarchy for AST nodes do you need, e.g. `Expression`, `BinaryExpression`?

Additional technical requirements and restrictions:

- `mjavac --parse [file]` must perform the syntactical analysis and accept (reject) the syntactically correct (incorrect) programs and terminate with return code 0 (1).

Please check in your solution into your repository until 2011-11-17, 12:00.