SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Sebastian Hack
Johannes Doerfert, B.Sc.

Compiler Construction WS15/16

Exercise Sheet 3

Exercise 3.1. 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 3.2. 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.
   - $\{|x| \mid x \in LL(0)\}$, 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 3.3. Checkable LL(k) conditions

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 24 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 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

Implement a parser for C^4:

- For expressions (§6.5) we handle *identifier*, *constant*, *string-literal*, parenthesized expression, `[]`, function call, `.`, `->`, `sizeof`, `&` (unary), `*` (unary), `-` (unary), `!`, `*` (binary), `+` (binary), `-` (binary), <, ==, `!=`, `&&`, `||`, `?:` and =. For all other expressions only the chain productions ($A \to B$) are used.
- At declarations (§6.7) we only consider *init-declarator-list* with at most one *init-declarator* without *initializer*. The only *declaration-specifiers* and *specifier-qualifier-list* is *type-specifier*. *type-specifier* is restricted to `void`, `char`, `int` and *struct-or-union-specifier*. The latter is only `struct` without *type-qualifier*s and bit fields. *declarator* and *direct-declarator* are *pointer* (without *type-qualifier-list*), *identifier*, parenthesized declarator and function declarator with *parameter-type-list*. *parameter-type-list* is only *parameter-list* without ellipses (`...`). All productions for *parameter-declaration* are considered.
- The considered *statement*s (§6.8) are *labeled-statement* with an identifier, *compound-statement*, *expression-statement* (both expression and null statements), *selection-statement* with `if` and `if-else`, *iteration-statement* with `while` and every *jump-statement*.
- The root are external definitions (§6.9), which are handled fully except for *declaration-list* in *function-definition*.

Remarks and hints

- Push your solution to the branch `master` till 17/11/15.
- For parsing your compiler will be invoked with `c4 --parse test.c`.
- The parser must reject all words that are not derivable from the full grammar. The parser must accept all correct programs according to the restricted grammar.
- The grammar as given is not suitable for LL parsing in some places. Adjust the grammar in these places accordingly.
- Don't repeat yourself! Factorise common operations into helper functions.
- For expression parsing see slide 4 of the "Top-Down Parsing (LL)" slides. Additionally, there will be a lecture on the practical aspects of parsing on Friday 13/11/15.
- The grammar is ambiguous for *selection-statement*s. How is this ambiguity resolved in the language standard? How can this be treated in the implemenation of the parser?
- How to implement the k-lookahead capability in your lexer/parser?
- It is not yet required to construct an abstract syntax tree, but will be necessary for the next parts of the project. What classes and class hierarchy for AST nodes do you need, e.g. `Expression`, `BinaryExpression`?
- Keep it simple!