# Compiler Construction (List 5)

Hans de Nivelle

6 November 2013

1. Consider the language $(\Sigma, R, S)$, defined by $\Sigma = \{\ '(', ')'\ \}$, $R = \{S \to SS,\ \ S \to (S),\ \ S \to \epsilon$.

   (a) Give a DFA-based grammar for this language. (Use the non-ambiguous grammar that was constructed in List 4 as a starting point.)

   (b) Does the DFA-based grammar have any nullable symbols? Give the FIRST and FOLLOW sets for each of the non-terminals of the DFA-based grammar. (This is a trivial task, but it has to be given anyway.)

2. (a) Give a DFA-based grammar for Lisp. The elements of a list can be atoms, numbers, or lists by themselves. Compute the FIRST and FOLLOW sets for every non-terminal occurring in the grammar.

   (b) Using the top down parser obtained from the DFA-based grammar, parse some expressions, e.g.

```
( a b c )
(( a ) ( b ) c )
( set a ( quote b ))
( define ( abs x ) ( if ( < x 0 ) ( - x ) x ))
```

   (c) Are there nullable symbols? Give the FIRST, and FOLLOW sets for every non-terminal that occurs in the grammar.

   (d) Sketch a recursive descent parser for LISP. (I am especially interested in the attribute computations.)

3. (a) Give a DFA-based grammar for Prolog expressions. You may ignore the presence of user defined operators, so it is sufficient to define a grammar for functional expressions of form

$$f(t_1, \ldots, f_n),$$

   and for lists, which have form

$$[\,],\ [L],\ [L_1, L_2],\ [L_1, L_2, L_3],\ \text{etc.}$$

(b) Are there nullable symbols? Give the FIRST and FOLLOW sets for every non-terminal of the grammar.

(c) Use the top-down parser derived from the DFA-based grammar to parse some realistic expressions.

4. (a) Give a DFA-based grammar for the 'Realistic Grammar' occuring on slide 11 of parsing.pdf.

(b) Give the FIRST and FOLLOW sets for every non-terminal of the grammar.

(c) Using the top-down parser derived from the DFA-based grammar, parse some realistic expressions, e.g.

```
while ( num < ident )
begin
   ident := ident + num
end

if( ident < ident )
   ident := ident
else
   ident := ident
```