Laboratory 13

Recursive-descent parsing – Lists in Haskell

In laboratory 13 we will adapt the subject of laboratory 9, which consisted of operations on lists in Haskell, now for recursive-descent parsing.

The grammar for the analyzer we must build is the following:

```
T = { concat, take, drop, i, (, ), [, ], : }
N = { E, L, Enum }
P = { E -> L concat E | L,
      L -> take E i | drop E i | (E) | [Enum],
      Enum -> i , Enum | i }
```

The set T contains terminals used in grammar and the set N contains nonterminals that we can also find in the set of productions P.

The nonterminal E stands for an expression from the input that could be formed by concatenating some expressions that return lists. The nonterminal L describes such expressions (that return lists), out of which we can remember *take* and *drop* commands. The nonterminal Enum stands for the construction of a list. The rule is right recursive because a left recursion would not be good for recursive-descent parsing.

In the construction of this analyzer, we will implement homonymous functions for each nonterminal from the grammar. Functions *main(), next_symbol()*, and *error()* can be reused from the previous labs, while functions for processing lists can be reused from laboratory 9.

If we want to add the functionality for *cons* to the analyzer, we must adapt the grammar:

```
P' = { E -> L concat E | L,
       L -> i : L | L1,
       L1 -> take E i | drop E i | (E) | [Enum],
       Enum -> i , Enum | i }
```

The operator "**:**" for *cons* is infix, like the *concat* operator, but has a higher precedence than *concat*. Therefore, we rank the E, L, and L1 nonterminals, in order to impose a hierarchy of precedences for the infix operators *concat* and '**:**'. Similarly, we imposed a hierarchy of precedences for the arithmetic operators through the hierarchy of nonterminals E (expression), T (term), and F (factor) in grammar G1. In grammar G1 we could introduce *sin* and *cos* functions at the lowest level (having the highest precedence), more exactly at the level of nonterminal F.

Similarly, we introduced functions *take* and *drop* in the current analyzer at the lowest hierarchical level (with a maximum precedence), that is nonterminal L1. We also introduced the possibility of grouping an entire expression E between parentheses at the lowest level (nonterminal L1), therefore with the highest precedence, similarly to G1 grammar where parentheses are introduced at the lowest level (nonterminal F).

Commands for generating the lexico-syntactic analyzer:

```
lex lists.l
gcc -o TREE lex.yy.c lists.c -ll
./TREE
```

Examples of input:

take [1, 2, 3, 4, 5] 3
> [1, 2, 3]

drop [1, 2, 3, 4, 5] 3
> [4, 5]

take ([1, 2, 13, 14] ++ [1, 2]) 3 ++ drop [22, 23] 1 ++ take (drop [1, 25, 2, 4, 18, 20] 3) 2
> [1, 2, 13, 23, 4, 18]

drop (take [15, 14, 13, 12, 11, 10] 5) 4 ++ [11, 12]
> [11, 11, 12]

Pentru cons:
1 : [2, 3, 4]
> [1, 2, 3, 4]