2IS15 Generic Language Technology (2014-2015)
Construction of a recursive descent recognizer

Assignment 2 (deadline: September 22$^{\text{nd}}$ 2014, 09:00h)

Introduction

The goal of the first set of assignments of the course Generic Language Technology is to get acquainted with the basic concepts of scanning and parsing. There are several scanner and parser generators, for instance LEX+YACC, that can be used to generate a scanner and parser. In order to get some idea of the underlying technologies, you have to construct a parser "by hand".

The second exercise focuses on the construction of a recursive descent recognizer. Given a lexical and context-free grammar of a simple language, Pico, a recognizer has to be constructed. For the scanning of the input tokens one of the scanner generators ("RegEx" or "Automaton") can be used.

Documentation on the Java "RegEx" library can be found at `http://docs.oracle.com/javase/tutorial/essential/regex/` and `http://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html`. Documentation on the "Automaton" library can be found at: `http://www.brics.dk/automaton/`. It may also be useful to consult the "Frequently Asked Questions" page: `http://www.brics.dk/automaton/faq.html`. The syntax of the regular expressions can be found at: `http://www.brics.dk/automaton/doc/index.html?dk/brics/automaton/RegExp.html`.

Transformation of Grammar to enable a recursive descent recognizer

1. The following grammar contains so-called EBNF constructs, e.g. `(STATEMENT ",")*` and associativity and priority definitions. In order to be able to write a recursive descent recognizer in a straightforward way, these EBNF constructs and the associativity and priority definitions have to be transformed into "regular" production rules.

```
start-symbol PROGRAM

context-free syntax
  PROGRAM ::= "begin" DECLS "|" (STATEMENT ";")* "end"
  DECLS ::= "declare" PICO-ID ("," PICO-ID)*
  DECLS ::= "declare"

context-free syntax
```

```
    STATEMENT ::= PICO-ID ":=" EXP

  context-free syntax
    EXP ::= PICO-ID
    EXP ::= NatCon
    EXP ::= EXP "+" EXP  {left}
    EXP ::= EXP "-" EXP  {left}
    EXP ::= EXP "*" EXP  {left}
    EXP ::= "(" EXP ")"

  lexical syntax
    PICO-ID ::= [a-z][a-z0-9]*

  lexical syntax
    NatCon ::= [0-9]+

  lexical syntax
    LAYOUT ::= [\ \t\n\r]*

  context-free priorities
    EXP ::= EXP "*" EXP >
    EXP ::= EXP "-" EXP >
    EXP ::= EXP "+" EXP
```

2. Is the resulting grammar in question 1 an LL(1) grammar? If yes, motivate your answer; if not, transform it into an equivalent LL(1) grammar and motivate that it is an LL(1) grammar.

3. In the course notes on compilers by Whyley (see webpage of this course) it is shown how to construct a (non-backtracking) recursive descent recognizer from a given LL(1) grammar. The recognizer uses a separate lexical analyzer (also called a scanner) that apart from returning the next token also takes care of recognizing and discarding layout (whitespace in this case). Use this construction to transform the LL(1) grammar resulting in question 2 into a recursive descent recognizer for Pico programs implemented in Java. For the implementation of the lexical analyzer (scanner) you can use one of the libraries described in the introduction (and also used in Assignment 1).

4. Write a number of test programs in Pico, both correct and erroneous. Test the recognizer from question 3 with these test programs.

Submission

Submit via PEACH (only submit text files, pdf-files, and java-files):

1. A description of the grammar resulting from the transformation in question 1.

2. A description of the resulting LL(1) grammar in question 2.

3. `PicoRec.java`, this Java module should contain an implementation of the recursive descent recognizer. The Java code should contain comments and adhere to some standard coding conventions.

4. The Pico test programs and the results of testing the recognizer on these.