Master of Science in Computer Science - University of Camerino
Formal Languages and Compilers A. Y. 2017/2018
Written Test of 12th July 2018 (Appello II)
Teacher: Luca Tesei

NOTE: Regular expressions should be written using the usual rules of precedence: the $*$ operator has precedence on concatenation, which has precedence on the | operator. The notation $(r)^+$ can be used with the usual meaning.

EXERCISE 1 (8 points)

Write a *regular definition* defining, among others, the lexical category **commands** denoting the language informally defined as follows:

- Every command starts with the character \ or with the character !
- After the starting character the name of the command follows, without spaces in between. The name must be a non-empty string of letters and digits starting with a letter.
- The name of the command may be followed, without spaces in between, by an optional parameter that must be inserted between curly braces. The parameter can be any integer number that does not start with zero and can not be zero.

Examples of valid **commands**: `\com1`, `!c`, `\fun{67}`. Hint: remember that in regular definition some lexical categories can be defined and used in subsequent definitions (e.g. **letter**, **digit** and so on).

EXERCISE 3 (12 points)

Consider the following grammar:

$$\begin{array}{lcl} S & \rightarrow & Ad \mid Bbc \\ A & \rightarrow & aB \\ B & \rightarrow & aAb \mid c \mid d \end{array}$$

1. Write formally the language generated by the grammar as a set of strings.
2. Is the grammar $\mathrm{LR}(1)$? Justify your answer and, if the answer is yes, provide the table for a bottom-up shift-reduce parser for the language.
3. Enumerate all the valid items for the viable prefix $aaaaa$.

EXERCISE 4 (12 points)

Consider a language of expressions (at least one) separated by semicolons. Each expression is formed by using operators $\oplus$ and $\otimes$ on operands that can be integer numbers or identifiers. Both the operators must be right associative and $\oplus$ has precedence on $\otimes$. Parentheses can be used to override the precedence and associativity rules.

1. Define a Syntax Directed Definition suitable to be implemented during bottom-up parsing for this language. The SDD has to have, among others, a boolean attribute of the top level symbol saying if the expressions generated by the symbol are of strictly increasing size from left to right. The size of an expression is defined as the length of its operands (which is the length of the sequence of digits for the number or the length of the name of the identifier; they can both be determined as an attribute of the lexical token) plus the number of used operators plus the number of used parentheses.

As an example, the sequence of expressions $15 \oplus y; 35 \oplus (i3 \otimes 87) \otimes x$ is a sequence in which the first expression has size 4 and the second one has size 12. The sizes increase from left to write, thus the boolean attribute at the top level must be true.