Formal Languages and Compilers
(A.Y. 2015/2016)
2h30m

September 27^{th}, 2016

First name: **Last name:**

Matriculation n.: **e-mail:**

Lexical Analysis

Exercise 1 – 6pt

In the definition of a regular expression real languages generally includes the not operator (in addition to the traditional symbols defining regular expressions). The operator permits to identify the set of strings that do not match the regular expression pattern, given an alphabet Σ. So, given a regular expression r on Σ, the strings matching $\neg r$ on Σ are those not matching r. If useful the operator can be included in the definition of the regular expressions for the following languages:

- The language $\mathcal{L}$ including strings over the alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, /\}$ which permits to represents dates between January 1^{st}, 1970 to December 31^{st}, 2029 using the format $dd/mm/yyyy$ (**in defininig the RegExp you can assume that leap years do not exist**)
- The language $\mathcal{L}$ including strings over the alphabet $\Sigma = \{a, b, c, d\}$ containing at least one occurrence of three different letters.
  (e.g. *abcd*, *bbabac*, *cadc*, *dbaaddcc* $\in \mathcal{L}$, *aabba*, *cccaacaa*, *dddbbdb* $\notin \mathcal{L}$)

Syntax Analysis

Exercise 2 – 13pts

Let's $\mathcal{G}$ the grammar defined by the following productions:

$$S \longrightarrow aAS \mid bS \mid A \quad A \longrightarrow aA \mid b \tag{1}$$

1. Discuss the applicability of the SLR parsing strategy
2. Discuss the applicability of the LR(1) parsing strategy
3. Show the steps of a parser LR(1) in the acceptance or rejection of the string *babaab*

Semantic Analysis

Exercise 3 – 14pts

Consider the following excerpt from a grammar for a complex programming language:

$$S \quad \rightarrow \quad \textbf{Do } S \textbf{ till } (B)$$

When executed the command perform S until the condition B becomes $true$. In any case the cycle is entered once and S executed, and only after the condition is checked.

- Provide an L-attributed SDD for the command that permits to translate it in a three-address code program that behaves as expected[1]
- Show the parse tree and derive the three address code program for the code snippet below. In doing this refer to the translation schemes for expressions and commands which have been introduced during the course. It is not necessary to check the type of the expressions, and it can be assumed that variables have been declared somewhere before reaching the statement.

```
. . .
i = 0;
Do
i = i + 1;
i = i + 2;
till (i > 5)
. . .
```

[1]For your convenience I recall that the function *top.get(id.lexeme)* permits to retrieve the address of the specified id, while the function *gen(. . .)* is used to generate three-address code in the right format for the different istructions, and finally *new Temp()* and *new Label()* permit to generate a new temporary address and a new label, respectively.