Formal Languages and Compilers
(Linguaggi Formali e Compilatori)

prof. L. Breveglieri
(prof. S. Crespi Reghizzi, L. Sbattella)

Exam - 07.02.2006 - Part I: Theory

NAME & SURNAME:

ID: SIGNATURE:

INSTRUCTIONS - PLEASE READ CAREFULLY:

- The exam consists of two parts:
  - I (80%) Theory:
    1. regular expressions and finite state automata
    2. context-free grammars and pushdown automata
    3. syntax analysis and parsers
    4. transduction and semantic analysis
  - II (20%) Practice on Flex and Bison
- To pass the complete exam the candidate is required to pass both parts (I and II), in a single call or in different calls in the same exam session.
- To pass part I (theory) the candidate is required to demonstrate a sufficient knowledge of the topics of all the four sections (1-4).
- The exam is open-book: textbooks and personal notes are permitted.
- Please write clearly in the free space left on the sheet and, if necessary, continue on the back side; attached to each section (1-4) one can find an additional white sheet.
- Time: Part I (theory): 2h.30m - Part II (practice): 30m

1 Regular expressions and finite automata 20%

1. Consider the following regular expression R:

$$R = a\,(ab^* \mid b^* a^+ c)^*$$

Do the following points:

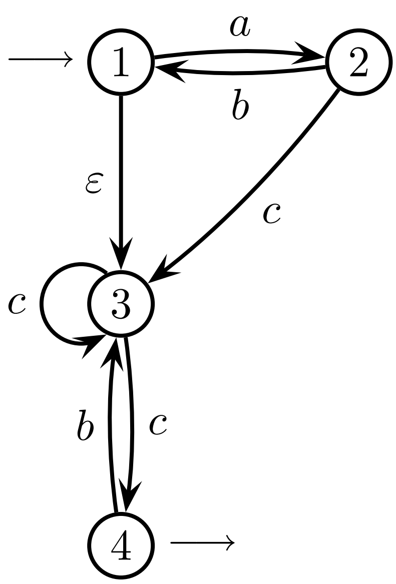
(a) Determine whether the regular expression R is ambiguous or not and explain in short why.

(b) Design the deterministic automaton (at one's choice whether in minimal form or not) recognizing the language $L(R)$.

(c) (optional) Design a non-deterministic automaton recognizing $L(R)$ and try to bound its number of states as much as possible.

---

2. Consider the following finite state automaton M:



Do the following points:

(a) Design the deterministic automaton (not necessarily in minimal form) equivalent to the automaton M.

(b) Minimize the number of states of the deterministic automaton (or justify whether it is already in minimal form).

(c) (optional) Determine a regular expression (at one's choice whether ambiguous or not) equivalent to the automaton M.

---

2 Context-free grammars 20%

1. Consider the following context-free language, of alphabet $\Sigma = \{a, b\}$:

$$L = \left\{ a^{2k} b^{2k} \mid \ k \geq 0 \right\} \cup \left\{ a^{3k} b^{3k} \mid \ k \geq 0 \right\}$$

   Examples: ε $aabb$ $aaabbb$

   Counterexamples: $aabbb$ $aaabb$

   Do the following points:

   (a) Design a context-free grammar G generating the language L and make sure such a grammar is not ambiguous.

   (b) Draw the syntax tree of the phrase: $a^6 b^6$

---

2. Design the context-free grammar, of EBNF type and not ambiguous, generating a subset of the C language, to model the selection statement `switch` of C. Such a grammar has to include the following concepts:

   - `switch` can have one or more branches, and possibly also the `default` branch (at the bottom)
   - the `switch` condition is an algebraic expression, of integer type
   - the labels of each branch of `switch` are constants of integer type, enclosed between apices, separated from each other with ‘,’ and from the execution part of the branch with ‘:’
   - the branches of `switch` consist of instruction lists, compliant with the usual C notation, and may be ended by the clause `break`
   - every instruction is either a procedure invocation, possibly with parameters, or an assignment statement, or an inner `switch`.
   - there are variables and constants of integer type
   - there are algebraic expressions of integer type, as follows
     - containing variables and constants
     - with the binary infix operators $+$, $-$ and $\times$
     - the operator $\times$ takes precedence over $+$ and $-$
     - parentheses are not used
   - identifiers and constants are simply modeled by *id* and *const*, respectively

   Example:

   ```
   switch (a + b - 1) {
       ‘1’ : {
            c = c + 2;
            proc1 (12, b + 1);
        }
        ‘2’, ‘3’ : d = d * e + 22;
        ‘5’ : {
            proc2 ();
            break;
        }
        default: { ... }
   }
   ```

   Do the following points:

   (a) Design the required context-free grammar (EBNF and not ambiguous).

   (b) Draw the syntax tree of the sample phrase above given.

---

3 Syntax analysis and parsers 20%

1. Consider the following context-free grammar G (axiom S):

production	lookahead set
$S \to A\ c\ S$	
$S \to \varepsilon$	
$A \to a\ A\ b$	
$A \to a$	

Do the following points:

(a) Compute the lookahead sets and check whether G is $LL(k)$ for some k.

(b) Design the LL syntactic procedure analyzing the non-terminal A.

2. Consider the following context-free grammar G (axiom S_0):

$$S_0 \rightarrow S \ \dashv$$

$$S \rightarrow A\ S$$

$$S \rightarrow \varepsilon$$

$$A \rightarrow a\ A\ b$$

$$A \rightarrow a$$

Do at your choice one of the following points (and optionally also the other one, if spare time is left):

(a) Draw the complete $LR(1)$ driver graph of the grammar G and determine whether such grammar is $LR\,(0)$, $LALR\,(1)$ or $LR\,(1)$.

(b) Execute the simulation of the recognition process for the string $aab \dashv \in L(G)$ by means of the Earley algorithm (fill the draft frame shown in the next page), then reconstruct the syntax tree of the string.

Draft frame for the simulation of the Earley algorithm

state 0	pos. a	state 1	pos. a	state 2	pos. b	state 3	pos. $\dashv$	state 4

4 Transduction and semantic analysis 20%

1. Consider the following transduction function τ:

$$\tau(x) = h_a(x) \qquad \text{if } |x|_a \text{ is even}$$
$$\tau(x) = h_b(x) \qquad \text{is } |x|_a \text{ is odd}$$

   where h_a and h_b are the natural projections canceling b and a, respectively, and x is any string over the alphabet $\{a, b\}$; for instance $h_a(abaa) = aaa$ and $h_b(bbabb) = bbbb$.

   Do the following points:

   (a) Design a finite state transducer (I/O automaton), possibly non-deterministic, computing the transduction τ.

   (b) Design a deterministic pushdown transducer computing the same transduction (one may assume the source string x has a terminator).

---

2. Consider a text, over the italian alphabet a, ..., z, to which a numeric key is appended; the key is an integer denoted in unary base (e.g., key 111_{unary} means $3_{decimal}$).

   The text has to be translated by alphabetic shifting: each character is replaced with the one that follows in the alphabetic ordering at the key distance. For instance:

source text with key	*translated (shifted) text (key removed)*
`vita111`	`bnzd`

   The syntactic support is the following

$$
\begin{array}{lcl}
S & \rightarrow & T\ C \\
T & \rightarrow & L\ T \\
T & \rightarrow & \varepsilon \\
L & \rightarrow & a \mid b \mid \ldots \mid z \\
C & \rightarrow & 1\ C \\
C & \rightarrow & 1
\end{array}
$$

   It is requested to design an attribute grammar that can extract the value of the key and can use it to compute an attribute τ associated with the non-terminal L; the attribute τ expresses the shifted version of the child terminal character of L (one may define and use more attributes if necessary or useful).

   Do the following points:

   (a) Define and list the necessary attributes, along with their type and meaning.

   (b) Write the semantic functions computing the attributes (in the next page one can find the draft frame to fill).

   (c) Draw the graphs of the function dependencies among the attributes, for each production rule.

   (d) Determine whether the attribute grammar is of type one-sweep.

   (e) Determine whether the attribute grammar is of type L.

---

syntax	*semantic functions*
$S_0 \rightarrow T_1\ C_2$	
$T_0 \rightarrow L_1\ T_2$	
$T_0 \rightarrow \varepsilon$	
$L_0 \rightarrow a$	
$\ldots$	
$L_0 \rightarrow z$	
$C_0 \rightarrow 1\ C_1$	
$C_0 \rightarrow 1$	