Formal Languages and Compilers
(Linguaggi Formali e Compilatori)

prof. Luca Breveglieri
(prof. S. Crespi Reghizzi, prof. A. Morzenti)

Written exam - 5 february 2008 - Part I: Theory

NAME: ______________________________

SURNAME: ______________________________

ID: ______________ SIGNATURE: ______________

INSTRUCTIONS - READ CAREFULLY:

- The exam consists of two parts:
  - I (80%) Theory:
    1. regular expressions and finite automata
    2. free grammars and pushdown automata
    3. syntax analysis and parsing
    4. translation and semantic analysis
  - II (20%) Practice on Flex and Bison
- To pass the exam, the candidate must succeed in both parts (I and II), in one call or more calls separately, but within one year.
- To pass part I (theory) one must be sufficient in all the four sections (1-4).
- The exam is open book (texts and personal notes are admitted).
- Please write in the free space left and if necessary continue on the back side of the sheet; do not attach new sheets nor replace the existing ones.
- Time: Part I (theory): 2h.30m - Part II (practice): 45m

1 Regular Expressions and Finite Automata 20%

1. The two following regular expressions R_1 and R_2 are given:

$$R_1 = (a \mid b)^* \; b \; (a \mid b \mid c)^*$$

$$R_2 = (a \mid c)^* \; b \; (a \mid b)^*$$

Both expressions are defined over alphabet $\{a, b, c\}$ and generate regular languages L_1 and L_2, respectively.

Answer the following questions:

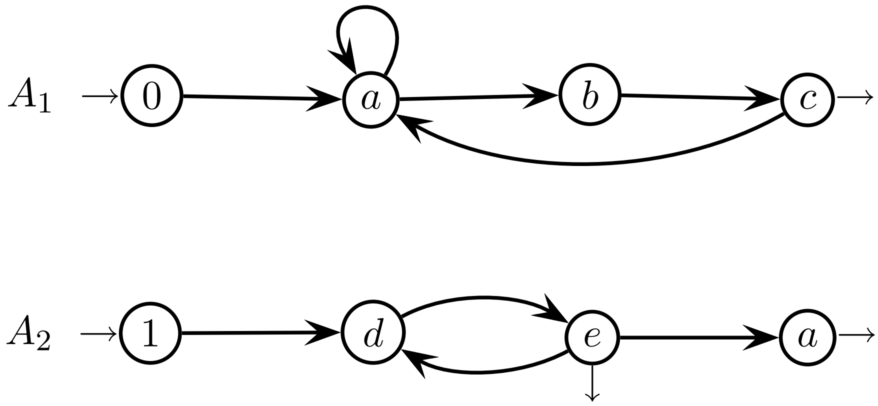
(a) List the four shortest strings that belong to the difference languages $L_1 \setminus L_2$ and $L_2 \setminus L_1$. Please fill the table below.

(b) Design the recogniser automaton of the difference language $L_1 \setminus L_2$ (either deterministic or indeterministic).

(c) If necessary, determinise and minimise the automaton previously designed.

#	$L_1 \setminus L_2$	$L_2 \setminus L_1$
1		
2		
3		
4		

2. The two following finite state automata A_1 and A_2 are given, which are of local type:



Such local automata A_1 and A_2 are defined over alphabets $\Sigma_1 = \{a, b, c\}$ and $\Sigma_2 = \{a, d, e\}$, respectively.

Answer the following questions:

(a) Write two regular expressions R_1 and R_2 that generate languages $L(A_1)$ e $L(A_2)$, respectively.

(b) Check whether the concatenation language L_C:

$$L_C = L(A_2) \cdot L(A_1)$$

and the union language L_U

$$L_U = L(A_1) \cup L(A_2)$$

are languages of local type as well, or not (notice that language L_C concatenates first $L(A_2)$ and then $L(A_1)$).

---

2 Free Grammars and Pushdown Automata 20%

1. The following free language L is given, which is the union of two component languages L_1 and L_2, both over alphabet $\{a, b\}$:

$$L = \underbrace{\{(a^n\, b^n)^* \mid \quad n \geq 1\}}_{L_1} \cup \underbrace{\left\{\left(a\, b^+\right)^*\right\}}_{L_2}$$

Here are a few sample strings of language L:

$$\varepsilon \qquad a\, b \qquad a\, b^+ \qquad a^2\, b^2 \qquad a^2\, b^2\, a\, b\, a\, b \qquad a\, b^2 \qquad a\, b^2\, a\, b$$

Answer the following questions:

(a) Write a grammar G, not in extended form (BNF), that generates language L (no matter whether G is ambiguous).

(b) If necessary, write a grammar G', not ambiguous, that generates language L and explain briefly why G' is not ambiguous.

---

2. Consider a free language that models simplified programs, which consist of lists of assignments. Such a language must include the following syntactic features:

   - The program is a list (not empty) of assignment statements, separated by ";" (semicolon); the bottom statement has a separator as well.
   - The assignment statement is denoted in the prefix form, according to the following model:

     ```
     :=  obj  expression
     ```

     where ":=" (colon equal) is the (prefix) assignment operator and "expression" is the value to be assigned to objct"obj" (see below).
   - The expression is a two-level sum-of-products and uses the (prefix) operators "add" and "mul" (addition and multiplication), is left-associative and is denoted in the prefix form as well, according to the following models (the equivalent infix form is displayed on the right as a comment):

     ```
     Expression in prefix form:      Comment (infix form):
       add obj obj                     obj + obj
       add add obj obj obj             obj + obj + obj
       add obj mul obj obj             obj + obj * obj
     ```

     Here "obj" repesents a generic object (see below).
   - The objects "obj" that can appear in the assignment statement (where it makes sense) and in the expression, are the following:
     - constant value, schematised by terminal symbol "c"
     - nominal variable, schematised by terminal symbol "v"
     - array element (one or more dimensioned), with the following syntax:

       ```
       a ( list_of_expessions )
       ```

       where the expressions in the list (not empty) are separated by "," (comma), and the array name is schematised by terminal symbol "a"

   Here is a short sample program (to help the reader on the right the usual infix notation is displayed as a comment):

   ```
   Program:                                    Comment (infix form):
     :=  v   c ;                                 v := c ;
     :=  v   add v c ;                           v := v + c ;
     :=  v   add add c v v ;                     v := c + v + v ;
     :=  v   add mul v v v                       v := v * v + v ;
     :=  v   a(mul v v) ;                        v := a(v * v) ;
     :=  a(c, add v v)  c ;                      a(c, v + v) := c ;
   ```

   Answer the following questions:

   (a) Write a grammar G, not ambiguous and in extended form (EBNF), that generates the above described language.

(b) Briefly say what semantic aspects of the above descrived simplified programming language cannot be modeled in a purely syntactic way.

---

3 Syntax Analysis and Parsing 20%

1. The following grammar G is given (with axiom S), in the extended form (EBNF):

$$G\left\{\ S \ \rightarrow \ (a \ \mid \ c) \ (b \ \mid \ S^{+}) \ c \, d\right.$$

   Such a grammar G is defined over terminal alphabet $\{a, b, c, d\}$.

   Answer the following questions:

   (a) Represent grammar G as a network of recursive finite state automata over total alphabet (union of terminals and nonterminals).

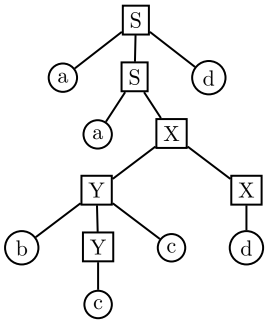
   (b) Check whether grammar G is of type $LL(k)$, for some $k \geq 1$.

   (c) (optional) Write the syntactic procedure of axiom S of the syntax analyser of type LL of grammar G (for the above determined value of k).

---

2. Consider the following grammar G (with axiom S), not in extended form (BNF), over terminal alphabet $\{a, b, c, d\}$:

$$G \begin{cases} S \rightarrow a\,S\,d \mid a\,X \\ X \rightarrow Y\,X \mid d \\ Y \rightarrow b\,Y\,c \mid c \end{cases}$$



sample string $a\,a\,b\,c\,c\,d\,d$

Answer the following questions:

(a) Design the driver graph (the prefix recogniser) of type $LR(0)$ of grammar G and prove that G is of type $LR(0)$.

(b) Compute the simulation of the recognition of the sample string $a\,a\,b\,c\,c\,d\,d \in L(G)$ (the syntax tree is shown above) by the syntax analyser of type $LR(0)$ of grammar G; please fill the table below (the number of rows is not significant); the two top rows are already filled.

#	transition	stack operation	input operation	stack contents	input tape contents	comment
0		push 1		1	$a\,a\,b\,c\,c\,d\,d\,\dashv$	start at macrostate 1
1	$1 \xrightarrow{a} 2$	push 2	shift a	1 2	$a\,b\,c\,c\,d\,d\,\dashv$	
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						
29						

4 Translation and Semantic Analysis 20%

1. One wishes one transformed an arithmetic expression of type sum-of-products, with possible subexpressions enclosed in round brackets, as for instance:

$$a + a \times a \qquad\qquad a + a \times a \times a \qquad\qquad a + a \times (a + a)$$

in the following way:

(a) the infix sign "+" becomes the sign "add", still of infix type

(b) the infix sign "×" becomes the sign "div" and the right factor "fact_2" of "×" becomes the inverse "($1\ \mathsf{div}\ \mathsf{fact}_2$)", that is:

$$\mathsf{fact}_1 \times \mathsf{fact}_2 \quad \Rightarrow \quad \mathsf{fact}_1\ \mathsf{div}\ (1\ \mathsf{div}\ \mathsf{fact}_2)$$

Notice that round brackets ensure that operator "div" is right-associative. For instance, the three expressions given before must be translated as follows (orderly):

$$a\ \mathsf{add}\ a\ \mathsf{div}\ (1\ \mathsf{div}\ a)$$

$$a\ \mathsf{add}\ a\ \mathsf{div}\ (1\ \mathsf{div}\ a\ \mathsf{div}\ (1\ \mathsf{div}\ a))$$

$$a\ \mathsf{add}\ a\ \mathsf{div}\ (1\ \mathsf{div}\ (a\ \mathsf{add}\ a))$$

The source language is defined by the following source grammar G:

$$G \begin{cases} E \rightarrow E\ \text{'+'}\ T \mid T \\ T \rightarrow T\ \text{'}\times\text{'}\ F \mid F \\ F \rightarrow \text{'('}\ E\ \text{')'} \mid \text{'}a\text{'} \end{cases}$$

Answer the following questions:

(a) Design a syntax transduction scheme (or grammar) G_τ that computes the above described translation. If one wishes one may modify the source grammar G, provided the modified grammar is equivalent to G.

(b) Design both the source and destination syntax trees of the third sample expression shown above.

---

2. The following production rule is given, of an attribute grammar G (the other rules of such a grammar do not matter here):

$$G \begin{cases} \dots \\ A \quad \rightarrow \quad X\ A\ Z \\ \dots \end{cases}$$

Grammar G has the following three attributes: α, β and γ. Suppose the rule is numbered as customary, that is $A_0 \rightarrow X_1\ A_2\ Z_3$, then the following semantic functions are given:

$$\begin{aligned} \alpha_0 &= f_1\,(\beta_0, \alpha_1, \alpha_2) \\ \beta_1 &= f_2\,(\alpha_3, \beta_3) \\ \beta_2 &= f_3\,(\beta_0, \alpha_1) \\ \gamma_2 &= f_4\,(\beta_2) \\ \gamma_3 &= f_5\,(\gamma_0) \\ \beta_3 &= f_6\,(\gamma_3) \end{aligned}$$

Here the detailed functional forms of $f_1, \dots, f_6$ do not matter.

Answer the following questions:

(a) Say which attributes are synthesised or inherited, respectively, and then draw the function dependence graph of the given grammar rule; please decorate the syntax support of the rule displayed below.

(b) Check whether the given rule satisfies the one-sweep condition or not, and, if it is the case, whether the rule is of type L as well; in both cases explain why this happens.

(c) (optional) Write the procedure of the semantic analyser for the above given rule.

---

rule syntactic support to be decorated with attribute function dependences