Formal Languages and Compilers
(Linguaggi Formali e Compilatori)

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

Written exam - 27 june 2009 - 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 answer the mandatory (not optional) questions. Notice the full grade is achieved by answering the optional questions.
- 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. Consider the following program:

```
a: read (i)
b: while (i ≠ 0)
c:     read (i)
d:     if (i > 0)
e:         then
               print (i)
f:         else
               print (−i)
               break
       end if
   end while
```

Suppose the instruction labels are the characters of the alphabet $\Sigma = \{a, b, c, d, e, f\}$. The instruction break causes the control flow to immediately exit the while loop.

Answer the following questions:

(a) Formalise the program control flow as a finite state automaton A recognising the regular language $L \subseteq \Sigma^+$ of the label sequences (e.g. $a\,b\,c\,d\,f$) corresponding to possible program executions. Draw the state-transition graph of automaton A.

(b) Find algorithmically a regular expression R that generates language L.

---

2. For the ternary alphabet $\{a, b, c\}$, consider the regular language L of the strings x defined as follows:

$$x = u\ a\ v\ b\ w$$

where the following conditions hold:

$$u \in \{b, c\}^+ \qquad v \in \{a, b, c\}^* \qquad w \in \{a, c\}^+ \qquad |u| \text{ and } |w| \text{ are both odd}$$

Notice that the string u does not contain any letter a and that the string w does not contain any letter b.

Examples:

$$b\,a\,c\,b\,a\,c\,a = \underbrace{b}_{u}\ a\ \underbrace{c}_{v}\ b\ \underbrace{a\,c\,a}_{w}$$

$$b\,a\,c\,b\,a\,c\,a = \underbrace{b\,c\,c}_{u}\ a\ \underbrace{b\,b\,a}_{v}\ b\ \underbrace{c}_{w}$$

Counterexamples:

$$a\,b \qquad\qquad b\,b\,a\,b\,a\,a$$

Answer the following questions:

(a) Design a finite state automaton A that recognises the language L, deterministic or not. Explain briefly how the automaton A works.

(b) (optional) Check if the automaton A is deterministic and minimal. If not so, design the deterministic and minimal equivalent automaton A'.

---

2 Free Grammars and Pushdown Automata 20%

1. The strings of a free language L over the binary alphabet $\{a, b\}$ consist of a number multiple of four of groups of consecutive letters a, that is there are $4\,n$ groups for some $n > 0$. Every group of letters a is separated from the next one by a letter b.

   In the left half of the groups (that is in the initial $2\,n$ groups), every group in odd position (i.e. the first, the third, etc) is strictly shorter (i.e. consists of a smaller number of letters a) than the next group (which is in even position).

   In the right half of the groups (that is in the final $2\,n$ groups), every group in odd position (i.e. the first, the third, etc) is strictly longer (i.e. consists of a larger number of letters a) than the next group (which is in even position).

   Here is an example of string belonging to language L (with $n = 2$ that is 8 groups):

   $$\underbrace{\overbrace{a^2}^{\text{group 1}}\ b\ \overbrace{a^5}^{\text{group 2}}\ b\,a^7\,b\,a^9}_{\text{left half}}\ \underbrace{b\,a^6\,b\,a^4\,b\,a^3\,b\,a^2}_{\text{right half}}$$

   Answer the following questions:

   (a) Write a grammar G, preferably not ambiguous, that generates the language L described above.

   (b) (optional) From grammar G, obtain in a systematic way a pushdown automaton A that recognises the language L. Say whether the automaton A is deterministic or not and explain why.

---

2. Write a grammar G, of type EBNF and not ambiguous, that generates the following script language for numerical computation:

   - the variables are not declared explicitly, but are implicitly of numerical type, and may be scalar or vectorial, with one or more indices separated by comma "," and included in square brackets
   - the variable identifiers are denoted by the terminal symbol id, as follows:

     ```
     id    id [1]    id [2, 3, 4]
     ```

   - there are two expression types:
     - arithmetic, consisting of constants, variables and arithmetic operators
     - logical (condition), consisting of arithmetic expressions with relational and logical operators
   - the numerical constants are indicated by the terminal symbol const
   - the arithmetic, relational and logical operators are listed as follows by decreasing priority; the grammar G must model syntactically the operator priority
     - $\times$ and $/$, multiplication and division
     - $+$ and $-$, addition and subtraction
     - $>$ and $\geq$, greater than and greater than or equal to
     - not, logical negation
     - and, logical product
     - or, logical sum
   - the expressions may contain round brackets, with the usual meaning
   - there are the following instruction types:
     - assignment, like for instance

       ```
       a := b + c ;
       ```

       the assignment instructions are followed by semicolon ";" and contain only arithmetic expressions (the other instructions do not have a final semicolon)
     - for loop
     - while loop
     - if-then-else conditional
   - the structured instructions may be freely nested into one another, except that the for loop may not contain the while loop, at whatever nesting depth level
   - structures of for and while loops, and of if-then-else (the else branch is optional):

     ```
     for var in start : step : end do
         instruction_list
     end
     ```

     ```
     while condition do
         instruction_list
     end
     ```

     ```
     if condition
         then instruction_list
         else instruction_list
     end
     ```

     where start, step and end are the initial value, the stride and the final value of the counter variable var, respectively, and are arithmetic expressions

3 Syntax Analysis and Parsing 20%

1. Consider the following grammar G over the terminal alphabet $\{a, b, c, d\}$ (axiom S):

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

Answer the following questions:

(a) Draw the network of finite state automata, over the total alphabet $\{a, b, S, X, Y\}$, that represents the grammar G.

(b) Find the smallest value of $k \geq 1$ such that the grammar G is of type $LL(k)$, by computing the lookahead sets on all the arcs of the automata for $k = 1$, and only if and where it helps for $k \geq 1$.

(c) (optional) For the value of k found, write the recursive descent syntax analyser of grammar G (or at least sketch a significant procedure thereof).

---

2. Consider the following grammar G over the terminal alphabet $\{a, b, c\}$ (axiom S):

$$G \begin{cases} S & \rightarrow & a\ S\ B \ \mid \ a\ S\ C \ \mid \ a \\ B & \rightarrow & b \\ C & \rightarrow & C\ c \ \mid \ b\ c \end{cases}$$

Answer the following questions:

(a) Explain as briefly as possible (but clearly) why the grammar G is not $LR(0)$.

(b) Prove that the grammar G is of type $LR(1)$ by drawing the related driver graph.

---

4 Translation and Semantic Analysis 20%

1. Consider the source language L_s that consists of the lists of arithmetic expressions (with only addition and without brackets). A sample phrase of L_s is as follows:

   $$a + a + a \,;\, a \,;\, a + a + a \,;\, a + a$$

   Notice that the separator between the expressions is the semicolon ";".

   The source language L_s is generated by the following source grammar G_s (axiom S):

   $$G_s \begin{cases} S & \rightarrow & \langle \mathsf{expr} \rangle \mid \langle \mathsf{expr} \rangle \text{ ';' } S \\ \langle \mathsf{expr} \rangle & \rightarrow & a \mid a \text{ '+' } \langle \mathsf{expr} \rangle \end{cases}$$

   One need define the transductions that compute the following translations:

   c1 the translation lists the expressions from right to left

   c2 every expression is translated into postfix form

   c1 and c2 the combination of the two conditions above

   For instance, the translated image of the previous sample phrase, with condition **c1**, is as follows:

   $$a \; + \; a \,;\, a \; + \; a \; + \; a \,;\, a \,;\, a \; + \; a \; + \; a$$

   and with condition **c2**, is as follows:

   $$a\, a \; + \; a + \,;\;\; a \,;\, a\, a \; + \; a + \,;\, a\, a +$$

   and finally, with condition **c1 and c2**, is as follows:

   $$a\, a + \,;\, a\, a \; + \; a + \,;\, a \,;\, a\, a \; + \; a +$$

   Notice that the source grammar G_s may be modified to write the transduction schemes, if necessary.

   Answer the following questions:

   (a) Write the transduction scheme (purely syntactic) that computes the translation specified by condition **c1**. Also draw the syntax trees of the source and destination phrases of the example above.

   (b) Similarly, write the transduction scheme (purely syntactic) that computes the translation specified by condition **c2** (no need of drawing the syntax trees here).

   (c) (optional) Write a transduction scheme (purely syntactic) that computes the translation specified by condition **c1 and c2** (again, no trees).

---

2. The so-called"rule of nine" is a simple criterion for checking the correctness of the multiplication of decimal integer numbers.

   Here is an example of the application of such a rule. Consider the multiplication $143 \times 283 = 40469$. Compute the two sums modulus 9 of the digits of each of the two factors and the sum modulus 9 of the digits of the product, as follows:

   $$(1 + 4 + 3) \mod 9 = 8 \mod 9 = 8$$

   $$(2 + 8 + 3) \mod 9 = 13 \mod 9 = 4$$

   and

   $$(4 + 0 + 4 + 6 + 9) \mod 9 = 23 \mod 9 = 5$$

   Now compute the product modulus 9 of the two sums obtained for the two factors. It yields:

   $$(8 \times 4) \mod 9 = 32 \mod 9 = 5$$

   The last value should coincide with the sum modulus 9 of the digits of the product, previously computed (which is precisely what happens here).

   Consider the following grammar G, that generates the equality between the multiplication of two integer factors and the product (axiom E).

   $$G \begin{cases} E \rightarrow T \text{ `=' } C \\ T \rightarrow C \text{ `}\times\text{' } C \\ C \rightarrow D\ C \\ C \rightarrow D \\ D \rightarrow \text{`0'} \mid \text{`1'} \mid \ldots \mid \text{`9'} \end{cases}$$

   Answer the following questions:

   (a) Basing on such a syntactic scheme G, define a simple attribute grammar that verifies the rule of nine. Preferably use only two synthesised attributes.

   Hint: use an attribute *eq* of logical type, true if and only if the rule of nine matches, and an attribute *val* of type integer modulus 9 (i.e. from 0 to 8), which stores the intermediate values during the rule verification.

   (b) Check if the attribute grammar is of type one-sweep and explain why.

---

attributes to be used for the grammar

type	name	(non)terminals	domain	meaning
left	*eq*		logical	true if the rule matches, false otherwise
left	*val*		integer modulus 9	intermediate values during the rule verification

	syntax	semantic functions
0:	$D_0 \rightarrow$ '0'	
1:	$D_0 \rightarrow$ '1'	
...	...	
8:	$D_0 \rightarrow$ '8'	
9:	$D_0 \rightarrow$ '9'	
10:	$C_0 \rightarrow D_1$	
11:	$C_0 \rightarrow D_1\ C_2$	
12:	$C_0 \rightarrow C_1$ '$\times$' C_2	
13:	$E_0 \rightarrow T_1$ '=' C_2	

extra sheet for notes and extension

extra sheet for notes and extension