Formal Languages and Compilers
(Linguaggi Formali e Compilatori)

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

Written exam - 6 march 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. Give the binary alphabet $\Sigma = \{a, b\}$. Consider the regular language L, over alphabet Σ, defined as follows:

$$L = \{\ w \mid \quad w \text{ does not contain three consecutive letters } b\ \}$$

Example strings belonging to language L:

$$\varepsilon \qquad a \qquad b \qquad a\,b \qquad b\,b \qquad a\,b\,b\,a \qquad a\,b\,a\,b\,b \qquad \ldots$$

Counterexample strings <u>not</u> belonging to language L:

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

Answer the following questions:

(a) Draw the state-transition graph of the deterministic minimal automaton A that recognises language L.

(b) (optional) Write a regular expression R equivalent to automaton A.

---

2. Give the binary alphabet $\Sigma = \{a, b\}$. Consider the regular languages L_1 and L_2, over alphabet Σ, defined as follows:

$$L_1 = \{ \; w \mid \quad |w| \geq 2 \; \wedge \text{ the penultimate character of } w \text{ is } b \; \}$$

$$L_2 = \{ \; w \mid \quad \text{in every odd position of } w \text{ the character is } b \; \}$$

Here are a few sample strings of language L_2:

$$\varepsilon \qquad b \qquad b\,a \qquad b\,b \qquad \ldots$$

Answer the following questions:

(a) Draw in a systematic way an automaton A, deterministic or not, that recognises the intersection language $L = L_1 \cap L_2$.

(b) (optional) Draw in a systematic way an automaton A' that recognises the difference language $L' = L_1 \setminus L_2$ (also written as $L' = L_1 - L_2$).

2 Free Grammars and Pushdown Automata 20%

1. Refer to the Dyck language with round and square brackets. Consider the two languages L_1 and L_2, which are both subsets of the Dyck language, defined as follows:

   - The strings of language L_1 contain only round brackets and the innermost bracket pairs are at even depth.
   - In the strings of language L_2 the <u>round</u> brackets are nested <u>into one another</u> down to even depth, independently of the possible presence of of square brackets. Notice: by canceling the square brackets from the strings of language L_2, language L_1 is obtained.

   Here are a few sample strings of language L_1:

   $$\varepsilon \qquad (\,(\,)\,) \qquad (\,(\,)\,(\,)\,) \qquad (\,(\,)\,)\,(\,(\,)\,) \qquad (\,(\,(\,(\,)\,)\,)\,)$$

   And here are a few sample strings of language L_2:

   $$\varepsilon \qquad [\,] \qquad (\,(\,)\,) \qquad (\,[\,(\,)\,]\,) \qquad [\,(\,[\,[\,(\,)\,]\,(\,[\,]\,)\,]\,)\,]$$

   Answer the following questions:

   (a) Define a grammar G_1, preferably not ambiguous, that recognises language L_1.

   (b) (optional) Define a grammar G_2, preferably not ambiguous, that recognises language L_2.

---

2. Consider the language of the arithmetic integer expressions with the following lexical and syntactic aspects:

   - the language has alphanumeric identifiers, similarly to the C language, like

     `a     alpha     alpha12     beta_1     alpha_omega`

   - the language has integer decimal constants, similarly to the C language, like

     `0     1     1234     98012`

   - the expression contains variables, whose names are generic identifiers, and integer decimal constants
   - the expression contains functions, with the following syntax:

     `function_name ( parameter_list )`

     and function names are generic identifiers
   - the parameter list is not empty, the parameters are separated by ',' (comma) and in general the parameter is an expression
   - the expression contains the arithmetic operators '+' and '$*$' (addition and multiplication), of <u>infix</u> type, and multiplication has higher precedence than addition
   - the expression contains the <u>binary</u> operator 'max', of <u>prefix</u> type, whose precedence is <u>lower</u> than that of both addition and multiplication, like

     `max a b + c`

     where addition `b + c` is computed first, then the maximum of `a` and the sum
   - the expression contains round brackets '(' and ')'

   Here is a sample expression:

   ```
   12 + (max b1 + c_2 alpha * (d + fun (a, omega))) * beta
   ```

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

---

3 Syntax Analysis and Parsing 20%

1. The ternary alphabet $c = call$, $r = return$ and $s = statement$ models the instructions `call`, `return` and the generic `statement` of a programming language L. Language L is generated by the following grammar G, in extended form (EBNF) with axiom S:

$$G \begin{cases} S & \rightarrow & (\ s \ \mid \ X\ )^* (\ s \ \mid \ X\ ) \\ X & \rightarrow & c\ (\ s \ \mid \ X\ )\ r \end{cases}$$

   The character c is always followed (though not consecutively) by a matching character r, and the characters s may be placed anywhere and in any number.

   Answer the following questions:

   (a) Draw a network of two recursive machines equivalent to grammar G.

   (b) On such a network write the lookahead sets, in the points where it is necessary, and say whether grammar G is of type $LL(1)$ or $LL(k)$, for some $k > 1$.

   (c) (optional) Extend language L and admit instructions `call` unmatched with any instruction `return`, like for instance $s\,c\,s\,c\,s\,r$. Examine whether the extended language is of type $LL(k)$, for some $k \geq 1$.

---

2. Give the following grammar G (axiom S):

$$G \begin{cases} S & \rightarrow & A\,S \mid A \\ A & \rightarrow & a\,b\,A\,a \mid b\,A\,a\,b \mid a\,b \end{cases}$$

Answer the following questions:

(a) Prove that grammar G is ambiguous, by finding the shortest string that has two syntax trees.

(b) Prove that grammar G is not of type $LR(1)$, by drawing the driver graph as completely as to obtain at least one inadequate state.

(c) (optional) Analyse the following string by means of the Earley algorithm:

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

Show which prefixes of such a string are accepted because they belong to the language generated by G as well. To this purpose please fill the following table, at whose bottom for convenience the rules are rewritten.

Simulation scheme of the Earley algorithm											
state 0	pos. a	state 1	pos. b	state 2	pos. a	state 3	pos. b	state 4	pos. a	state 5	

$S \rightarrow A\ S$ $\qquad$ $S \rightarrow A$ $\qquad$ $A \rightarrow a\ b\ A\ a$ $\qquad$ $A \rightarrow b\ A\ a\ b$ $\qquad$ $A \rightarrow a\ b$

4 Translation and Semantic Analysis 20%

1. Give the quaternary alphabet $\Sigma = \{a, b, c, d\}$, for both source and destination. Consider the source and destination languages L_s and L_d, defined as follows:

$$L_s = a^* \; b^+ \; c^+ \; d \qquad L_d = a^* \; c^+ \; b^+ \; d$$

On such languages define the syntax transduction τ as follows:

$$\tau \colon L_s \rightarrow L_d$$

$$\tau \left(a^p \; b^q \; c^r \; d \right) \mapsto a^p \; c^r \; b^q \; d \qquad p \geq 0 \qquad q, r \geq 1$$

Example:

$$\tau \left(a \; a \; b \; b \; b \; b \; c \; c \; d \; \right) = a \; a \; c \; c \; b \; b \; b \; b \; d$$

Answer the following questions:

(a) Draw the state-transition graph of a pushdown transducer T that computes transduction τ. Freely choose whether to make it deterministic or indeterministic and to recognise by empty stack or final state.

(b) (optional) Write the syntax transduction scheme G_τ (or the transduction grammar) that computes transduction τ. Suggestion: consider the source scheme shown in the following and write the corresponding destination scheme.

Scheme to be completed for the optional question (b):

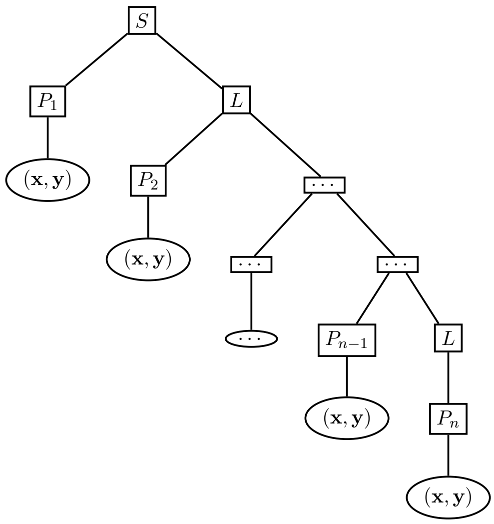
$$G_s \begin{cases} S \rightarrow A \\ A \rightarrow a \; A \\ A \rightarrow b \; X \; c \; d \\ X \rightarrow b \; X \; c \\ X \rightarrow \varepsilon \\ X \rightarrow b \; B \\ X \rightarrow C \; c \\ B \rightarrow b \; B \\ B \rightarrow \varepsilon \\ C \rightarrow C \; c \\ C \rightarrow \varepsilon \end{cases} \qquad G_d \begin{cases} S \rightarrow \\ A \rightarrow \\ A \rightarrow \\ X \rightarrow \\ X \rightarrow \\ X \rightarrow \\ X \rightarrow \\ B \rightarrow \\ B \rightarrow \\ C \rightarrow \\ C \rightarrow \end{cases}$$

2. A piecewise straight line is described by listing its points (at least two), as follows:

$$P_1 = (\ x_1,\ y_1) \quad P_2 = (\ x_2,\ y_2\ ) \quad \ldots \quad P_n = (\ x_n,\ y_n\ ) \quad n \geq 2$$

The list of points is generated by the following syntactic support G (axiom S):

$$G \begin{cases} 1: & S \rightarrow P\ L \\ 2: & L \rightarrow P\ L \\ 3: & L \rightarrow P \\ 4: & P \rightarrow (\ \mathbf{x},\ \mathbf{y}\ ) \end{cases}$$



Suppose the generated points are numbered as follows: $S \overset{*}{\Rightarrow} P_1\ P_2\ \ldots\ P_{n-1}\ P_n$; as the syntax tree on the right shows.

Suppose to have two lexical attributes α and ω (abscissa and ordinate), of integer and left type, associated with the nonterminal P, precomputed (i.e. immediately available), which contain the values of the terminals $\mathbf{x}$ and $\mathbf{y}$, respectively.

Answer the following questions:

(a) Write the semantic functions of an attribute grammar that computes the length λ (a real number) of the piecewise straight line $P_1,\ P_2,\ \ldots,\ P_i$ and associates it with the terminal P representing point P_i ($1 \leq i \leq n$); at point P_1 pose $\lambda = 0$. If necessary use more attributes. Fill the table on the next page.

(b) (optional) Check if the attribute grammar designed above is of type one-sweep.

syntax	semantic functions
1: $S_0 \rightarrow P_1\ L_2$	
2: $L_0 \rightarrow P_1\ L_2$	
3: $L_0 \rightarrow P_1$	
4: $P_0 \rightarrow (\ \mathbf{x},\ \mathbf{y}\ )$	