Formal Languages and Compilers
(Linguaggi Formali e Compilatori)

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

Exam - 7 february 2007 - 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 whatever order in a single call or in different calls, but to conclude in one year.
- To pass part I (theory) the candidate is required to demonstrate a sufficient knowledge of the topics of each one 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 sheets; do not attach or replace sheets
- Time: Part I (theory): 2h.30m - Part II (practice): 45m

1 Regular expressions and finite automata 20%

1. Consider the two following regular expressions:

$$R_1 = a^* b^* a^* \qquad R_2 = (ab \mid bb \mid a)^*$$

which generate the regular languages L_1 and L_2, respectively.

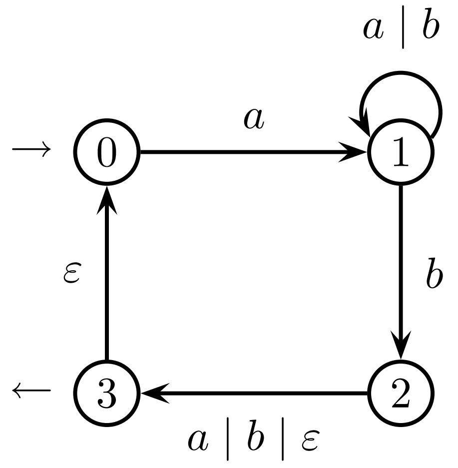
Do the following points:

(a) List all the strings of length ≤ 4 belonging to the set differences $L_1 \setminus L_2$ and $L_2 \setminus L_1$ (fill the space below).

$L_1 \setminus L_2$	$L_2 \setminus L_1$

(b) Write the regular expression R_3 generating the language $L_3 = \neg L_1$ (the set complement of L_1) and use only the union, concatenation, star and cross operators.

2. Consider the following finite state automaton:



Do the following points:

(a) Design the equivalent deterministic automaton (use a method of choice).

(b) Minimize the number of states of the deterministic automaton obtained before.

2 Context-free grammars and pushdown automata 20%

1. Consider the Dyck language over the alphabet $\Sigma = \{a, b, c, d\}$, where a, c are open parentheses and b, d are the corresponding closed ones, respectively.

   Do the following points:

   (a) Write the grammars G_1 and G_2 of the languages L_1 and L_2, respectively, both of type BNF (non-extended). The languages L_1 and L_2 are defined as follows:

   - $L_1 =$ the Dyck language with parentheses structures of depth at most 2

     Examples: $\varepsilon \quad a\,b \quad c\,d \quad a\,b\,c\,d \quad a\,c\,d\,b \quad a\,a\,b\,c\,d\,b$

     Counterexamples: $a\,a\,c\,d\,b\,b \quad a\,a\,b\,c\,a\,b\,d\,b$

   - $L_2 =$ the Dyck language with parentheses structures of arbitrary depth, but not containing concatenated structures at any level

     Examples: $\varepsilon \quad a\,b \quad c\,d \quad a\,c\,d\,b \quad a\,a\,c\,d\,b\,b$

     Counterexamples: $a\,a\,b\,c\,d\,b \quad c\,d\,a\,b$

   (b) Write a non-ambiguous grammar G of the language L, of type BNF (non-extended), where the language L is defined as follows:

   $$L = L_1 \cup L_2$$

   that is, the union of L_1 and L_2.

---

2. Consider a simplified version of the query language SQL, defined as follows:

   - There are attribute names and relation names (neither one is to be defined here). An attribute may be denoted by a simple name (an identifier) or by a compound name (a pathname) with "`.`":

     `relation-name.attribute-name`

   - There are logical expressions with parentheses "`(`" and "`)`", consisting of the logical operators `and` and `not`, and of relational predicates; `not` precedes `and`.
   - There are relational predicates, consisting of the comparison operator "equal-to" or "less-than", i.e. "`==`" or "`<`", and of two operands, namely:

     `term1  rel-op  term2`

     Operand `term1` is an attribute, while operand `term2` may be an attribute or a number (not to be defined here). Relational operators precede the logical ones.
   - A single SQL query block must contain all the following clauses (orderly listed):
     - `select`, followed by:
       - either a list of attributes with separator "`,`"
       - or the terminal "`*`", which indicates all the attributes
     - `from`, followed by a list of relations with separator "`,`"
     - `where`, followed by a logical expression

     The query block may end with the `order-by` clause, followed by one of the attributes listed in `select` and a flag to sort in ascending or descending order.
   - A SQL query expression consists either of one query block or of two blocks connected by the set operator `union` or `intersect`.

   Examples:

```
SELECT   CCB.NAME, TRANSFERS.PRICE, TRANSFERS.MOTIVATION
FROM     CCB, TRANSFERS
WHERE    CCB.NUMCCB == TRANSFERS.NUMCCB AND NOT (TRANSFERS.PRICE < 1000)
   UNION
SELECT   CCB.NOME, TRANSFERS.PRICE, TRANSFERS.MOTIVATION
FROM     CCB, MOVIMENTI
WHERE    CCB.NUMCCB == TRANSFERS.NUMCCB AND CCB.NUMCCB < 1356789
```

   and also

```
SELECT   *
FROM     CCB
WHERE    BALANCE < 200000
ORDER-BY CCB.NUMCCB ASC
```

   Do the following points:

   (a) Design a non-ambiguous grammar, of type EBNF (extended), to model the language of the SQL query expressions defined before.

   (b) List the aspects of the proposed problem that could not be modeled by means of a purely syntactic tool as an EBNF grammar.

---

3 Syntax analysis and parsers 20%

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

$$
\begin{aligned}
S &\rightarrow \left(E \text{ `} = \text{'`} = \text{' } E \mid \text{`}i\text{' `} = \text{' } E \right)^{+} \\
E &\rightarrow \text{`}i\text{' } \left( \text{`} + \text{' `}i\text{'} \right)^{*}
\end{aligned}
$$

Do the following points:

(a) Represent grammar G as a network of recursive finite state machines.

(b) Check whether the grammar (in network form) is of type $LL(k)$, for some $k \geq 1$.

(c) If necessary, modify grammar G so as to make it of type $LL(1)$.

---

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

$$
\begin{aligned}
S &\rightarrow E \text{ `} \dashv \text{'} \\
E &\rightarrow T \text{ `} + \text{'} E \mid T \\
T &\rightarrow \text{`}i\text{'} \mid \text{`}i\text{'} \text{ `} + \text{'} T \mid \text{`(' } E \text{ `)'}
\end{aligned}
$$

Do the following points:

(a) Check whether grammar G is of type $LR(1)$ (use a method of choice).

(b) If necessary, modify grammar G so as to make it of type $LR(1)$.

(c) (optional) Discuss briefly whether language L has a grammar of type $LR(0)$.

---

4 Transduction and semantic analysis 20%

1. A series of multiplications and additions of integers, without parentheses, should be translated from the infix to the prefix (polish) notation. Multiplication precedes addition and the operands are one-digit (decimal) numbers (from 0 to 9). The source language L_s is defined by the following (source) grammar G_s:

G_s	G_p
$E \to T$ ‘ + ’ E	
$E \to T$	
$T \to$ ‘0’ ‘ × ’ T	
$T \to$ ‘1’ ‘ × ’ T	
...	
$T \to$ ‘9’ ‘ × ’ T	
$T \to$ ‘0’	
$T \to$ ‘1’	
...	
$T \to$ ‘9’	

Here follows a transduction example:

$$3 + 1 \times 5 \times 4 + 2 \quad \xrightarrow{\tau} \quad \text{add 3 add mul 1 mul 5 4 2}$$

Do the following points:

(a) Complete the transduction grammar drafted before and write the destination grammar G_p that generates the transduction τ (fill the free space above).

(b) Without changing the source language L_s, one wishes one improved the transduction τ as much as possible, not to emit dummy operations such as:

- multiplication by 0
- multiplication by 1

Here follow two examples of the optimised transduction τ':

$$3 + 1 \times 5 + 2 \quad \xrightarrow{\tau'} \quad \text{add 3 add 5 2}$$

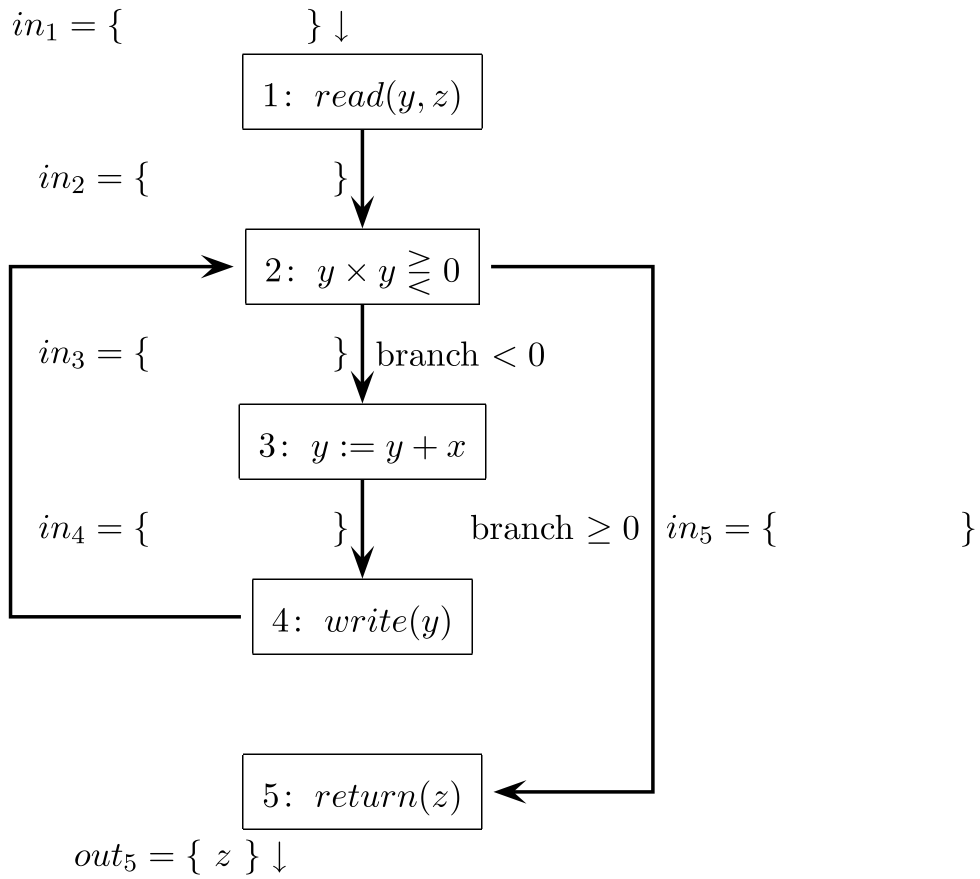
$$3 \times 5 \times 0 + 4 + 0 \quad \xrightarrow{\tau'} \quad \text{add 0 add 4 0}$$

Design and write (on the next page, fill the free space) a new transduction grammar that computes the optimised transduction τ' (write both the new source and the new destination components), and draw the new source and destination syntax trees (or both superposed) of the latter example.

(c) (optional) Discuss briefly how to improve more the transduction τ', in order not to emit the addition operation in the case either summand is null.

G'_s	G'_p

2. A subprogram (routine) with an input parameter x and an output parameter z, is modeled by means of the following control graph:



Node 2 is a two-way conditional instruction. It is specified that the output parameter z is live at the output of node 5.

Do the following points:

(a) Write the control flow equations to compute the *live variables* at each node in the program.

(b) Compute the solution of the flow equations and list the live variables in the figure above (fill the free sets at each point).

(c) (optional) Suppose that the conditional node 2 executes the following test:

$$\text{if} \quad (y \times y \geq 0) \quad \text{go to } 5 \text{ else go to } 3$$

Indicate which sets of live variables change as a consequence, and how.
