FORMAL LANGUAGES AND COMPILERS

prof.s Luca Breveglieri and Angelo Morzenti

Exam of Fri 4 September 2015 - Part Theory

WITH SOLUTIONS - FOR TEACHING PURPOSES HERE THE SOLUTIONS ARE WIDELY COMMENTED

NAME:

MATRICOLA: SIGNATURE:

INSTRUCTIONS - READ CAREFULLY:

- The exam is in written form and consists of two parts:
  1. Theory (80%): Syntax and Semantics of Languages
     - regular expressions and finite automata
     - free grammars and pushdown automata
     - syntax analysis and parsing methodologies
     - language translation and semantic analysis
  2. Lab (20%): Compiler Design by Flex and Bison
- To pass the exam, the candidate must succeed in both parts (theory and lab), in one call or more calls separately, but within one year (12 months) between the two parts.
- To pass part theory, the candidate must answer the mandatory (not optional) questions; notice that the full grade is achieved by answering the optional questions.
- The exam is open book: textbooks and personal notes are permitted.
- Please write in the free space left and if necessary continue on the back side of the sheet; do not attach new sheets and do not replace the existing ones.
- Time: part lab 60m - part theory 2h.15m

1 Regular Expressions and Finite Automata 20%

1. Given the alphabet $\Sigma = \{\ a,\ b,\ c\ \}$, consider the following regular expression R:

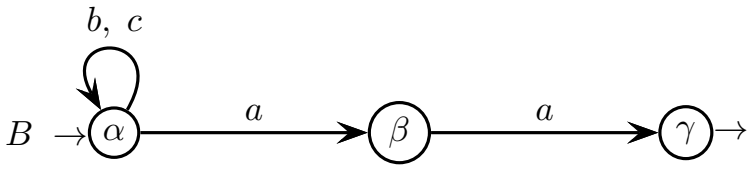
$$R = c\ (\ b\ \mid\ b\ a\ )^*\ a$$

Answer the following questions:

(a) Derive the deterministic automaton A that recognizes the language $L(R)$, by using the Berry-Sethi method.

(b) Determine, in a systematic way, if the automaton A is minimal.

(c) Consider the automaton B drawn below:



Design the automaton C that accepts the intersection of languages $L(R)$ and $L(B)$, by using the automaton cross product (Cartesian product) construction.

(d) From the automaton C derived at the previous point, derive the regular expression R' for language $L(R) \cap L(B)$, by using the BMC method.

(e) (optional) Determine if the language $L(R)$ is local and provide an adequate justification to your answer.

Solution

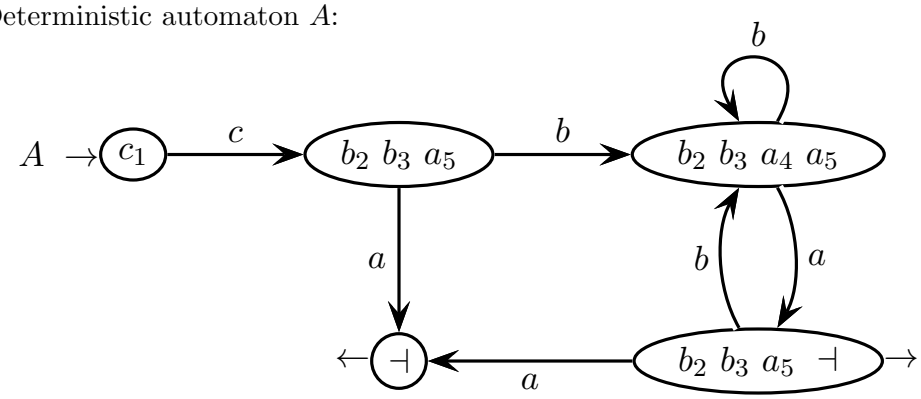
(a) Berry-Sethi. Numbered regular expression $R_{\#}$:

$$R_{\#} = c_1 \ (\ b_2 \ | \ b_3 \ a_4 \ )^* \ a_5 \ \dashv$$

Initials and followers:

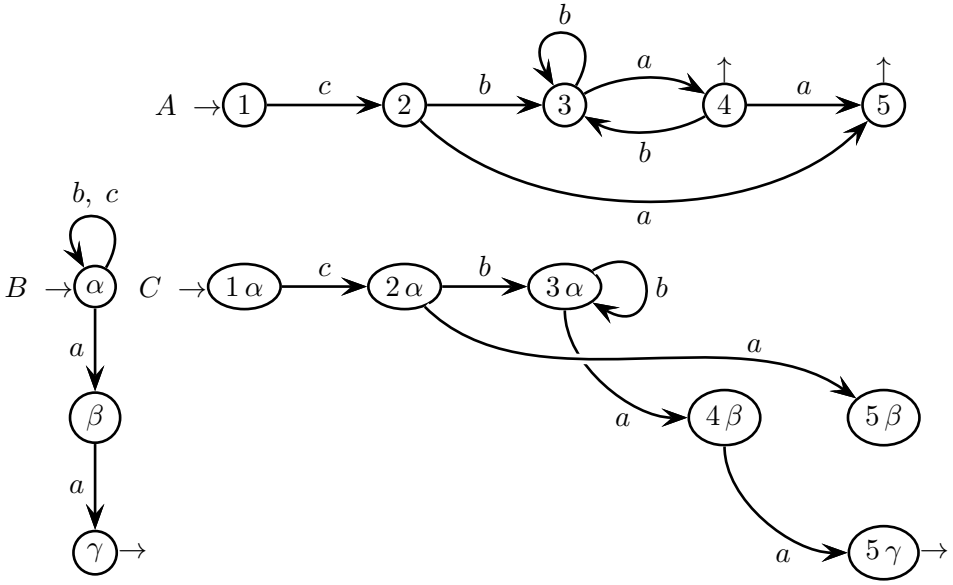
initials	c_1
generators	*followers*
c_1	$b_2 \ b_3 \ a_5$
b_2	$b_2 \ b_3 \ a_5$
b_3	a_4
a_4	$b_2 \ b_3 \ a_5$
a_5	$\dashv$

Deterministic automaton A:

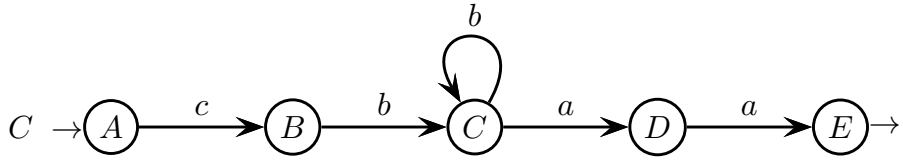


(b) Automaton A is minimal. First notice that it is clean (all the states are useful). Second: the two final states have outgoing arcs with different labels, therefore they are distinguishable; the initial state has an outgoing arc with a label different from those of the outgoing arcs of the other two non-final states, therefore it is distinguishable from such states; and the two non-initial non-final states have outgoing arcs with label a that go to two states already known to be distinguishable, therefore they are distinguishable as well. Thus all the states of A are distinguishable and in conclusion the automaton is minimal.

(c) Product construction (the state names of automaton A are shortened and the product states that are inaccessible from the initial one are not drawn):



The product automaton C is not clean, as state $5\,\beta$ is not post-accessible (it does not reach any final state) and thus can be canceled. Here is the clean product automaton C, after renaming the states again:



The clean automaton C is the product of two deterministic automata, hence it is deterministic. It would be easy to verify that it is minimal as well.

(d) We skip the stepwise application of the *BMC* method to the (clean) automaton C, as it is quite simple, and we directly go to the fairly intuitive result:

$$R' = c\ b\ b^*\ a\ a = c\ b^+\ a^2$$

(e) The language $L\,(R)$ is not local. In fact the local sets of $L\,(R)$ are $Ini\big(L\,(R)\big) = \{\ c\ \}$, $Fin\big(L\,(R)\big) = \{\ a\ \}$ and $Dig\big(L\,(R)\big) = \{\ a\,a,\ b\,a,\ b\,b,\ c\,b,\ c\,a\ \}$. They allow the string $c\,a\,a$, yet such a string does not belong to $L\,(R)$.

2 Free Grammars and Pushdown Automata 20%

1. Consider the free language L, over the three-letter alphabet $\Sigma = \{\ a,\ b,\ c\ \}$, defined as follows:

$$L = \left\{\ x\ c\ w\ c\ y\ \middle|\quad x,\ y \in \{\ a,\ b\ \}^*\ \wedge\ w^R = y\ x\ \right\}$$

Answer the following questions:

(a) Write a non-extended (*BNF*) grammar G that generates the language L.

(b) Consider the following language L', which is the intersection of the language L with a regular language:

$$L' = L\ \cap\ (\ a\ b\ )^*\ \ c\ \Sigma^*\ c\ \ (\ a\ b\ )^*$$

Write a non-extended (*BNF*) grammar G' that generates the language L'.

(c) (optional) Consider the language $L'' = \overline{L'}$, which is the complement of the language L' found at the previous point. Say if language L'' is regular or not, and reasonably justify your answer.

Solution

(a) Since from $w^R = y\ x$ it follows that $w = x^R\ y^R$, language L is the concatenation of two palindromes with centre. Grammar G easy (axiom S):

$$G \begin{cases} S & \to & P\ P \\ P & \to & a\ P\ a \\ P & \to & b\ P\ b \\ P & \to & c \end{cases}$$

Grammar G is *BNF* (and not ambiguous).

(b) Clearly language L' is free as well. The effect of the regular language is to force the left and right palindromes to alternate $a\,b$ and $b\,a$, respectively. Here is grammar G' (axiom S):

$$G' \begin{cases} S & \to & P_1\ Q_1 \\ P_1 & \to & a\ P_2\ a \\ P_2 & \to & b\ P_1\ b \\ P_1 & \to & c \\ Q_1 & \to & b\ Q_2\ b \\ Q_2 & \to & a\ Q_1\ a \\ Q_1 & \to & c \end{cases}$$

or also:

$$G' \begin{cases} S & \to & P\ Q \\ P & \to & a\ b\ P\ b\ a \\ P & \to & c \\ Q & \to & b\ a\ Q\ a\ b \\ Q & \to & c \end{cases}$$

Both versions are not ambiguous.

(c) Language L'' is not regular. If it were, then language L' itself would be regular, yet clearly L' is not. In fact, it can be formulated as:

$$L' = (\,a\,b\,)^h\ c\ (\,b\,a\,)^{h+k}\ c\ (\,a\,b\,)^k$$

and clearly it is not regular because of the relationships between the exponents, which cannot be obtained by purely iterative operators like star or cross.

2. A marked language, inspired to *XML*, is used to encode trees in textual form. Such textual representations are subject to the following syntax constraints.

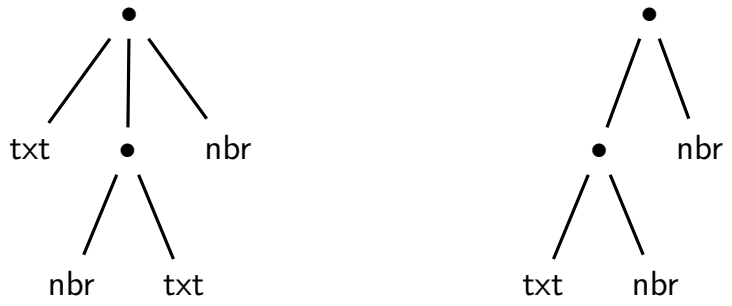
   - All tree nodes are either ternary or binary inner nodes, or leaves. In particular:
     - ternary nodes have three child nodes, and are marked with the ⟨3TRSTRT⟩ and ⟨3TREND⟩ tags
     - binary nodes have two child nodes, and are marked with the ⟨2TRSTRT⟩ and ⟨2TREND⟩ tags
   - Ternary nodes can have any type of child node, while the child nodes of binary nodes can only be binary nodes or leaves.
   - Leaf nodes are either text elements (represented by the **txt** terminal) or number elements (represented by the **nbr** terminal).
   - A phrase of the language consists of a non-empty, comma-separated, list of trees (that is, a so-called "tree forest").

   The sample text below encodes a list of two trees, the first with a ternary root and a binary inner node, and the second with a binary root and a binary inner node.

   ⟨3TRSTRT⟩ **txt** ⟨2TRSTRT⟩ **nbr txt** ⟨2TREND⟩ **nbr** ⟨3TREND⟩ ,

   ⟨2TRSTRT⟩ ⟨2TRSTRT⟩ **txt nbr** ⟨2TREND⟩ **nbr** ⟨2TREND⟩

   The figure below shows the two trees (forest) encoded by the text sample above.



   Answer the following questions:

   (a) Write an extended grammar (*EBNF*) for this marked language and verify that the grammar is not ambiguous.

   (b) (optional) Without changing the types and specifications of the ternary, binary and leaf nodes, define a new node type with an arity (number of child nodes) strictly greater than three. This new node type has tags ⟨>3TRSTART⟩ and ⟨>3TREND⟩. Its child nodes can be of any type: leaf, binary, ternary, and the new node type. Show only the rules of the previous grammar that need to be modified, and the new rules introduced (if any).

Solution

(a) Here is grammar G (axiom FOREST):

$$G \begin{cases} \langle \text{FOREST} \rangle \to \langle \text{NODE} \rangle \ (\text{`,'} \ \langle \text{NODE} \rangle \ )^* \\ \langle \text{NODE} \rangle \to \langle \text{TERNARY} \rangle \mid \langle \text{BINARY} \rangle \mid \langle \text{LEAF} \rangle \\ \hline \langle \text{TERNARY} \rangle \to \text{`}\langle \text{3TRSTRT} \rangle\text{'} \ \langle \text{3CHILD} \rangle \ \langle \text{3CHILD} \rangle \\ \qquad \langle \text{3CHILD} \rangle \ \text{`}\langle \text{3TREND} \rangle\text{'} \\ \langle \text{BINARY} \rangle \to \text{`}\langle \text{2TRSTRT} \rangle\text{'} \ \langle \text{2CHILD} \rangle \ \langle \text{2CHILD} \rangle \ \text{`}\langle \text{2TREND} \rangle\text{'} \\ \langle \text{LEAF} \rangle \to \text{txt} \mid \text{nbr} \\ \hline \langle \text{3CHILD} \rangle \to \langle \text{TERNARY} \rangle \mid \langle \text{BINARY} \rangle \mid \langle \text{LEAF} \rangle \\ \langle \text{2CHILD} \rangle \to \langle \text{BINARY} \rangle \mid \langle \text{LEAF} \rangle \end{cases}$$

Notice the distinction between terminals enclosed in angle brackets, and non-terminals, which is essential. Notice there are two rules with the same right part. Such rules could be unified, though keeping them helps us understand the grammar.

(b) Quite simple. Modified rule:

$$\left\{ \langle \text{NODE} \rangle \to \langle \text{N\_ARY} \rangle \mid \langle \text{TERNARY} \rangle \mid \langle \text{BINARY} \rangle \mid \langle \text{LEAF} \rangle \right.$$

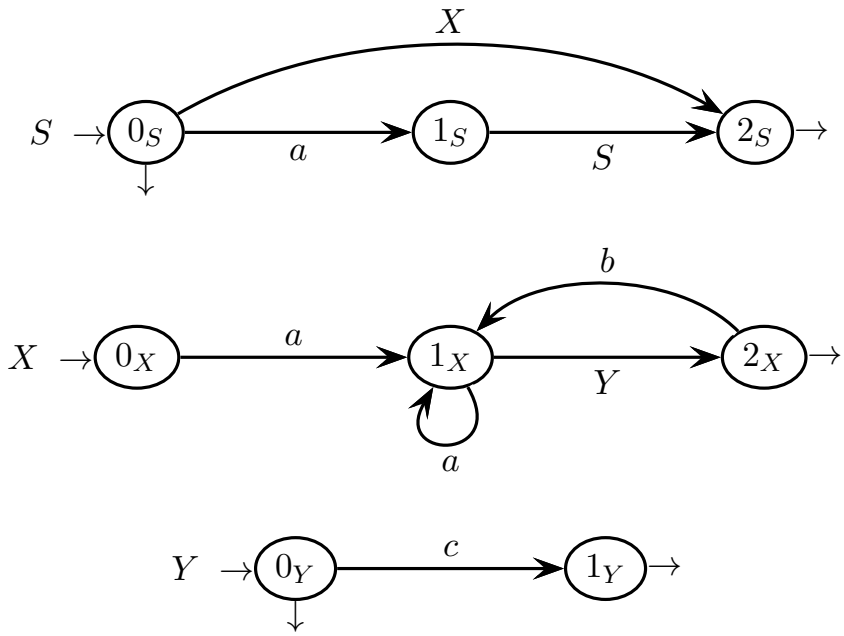
Added rules:

$$\begin{cases} \langle \text{NODE} \rangle \to \langle \text{N\_ARY} \rangle \mid \langle \text{TERNARY} \rangle \mid \langle \text{BINARY} \rangle \mid \langle \text{LEAF} \rangle \\ \langle \text{N\_ARY} \rangle \to \text{`}\langle > \text{3TRSTRT} \rangle\text{'} \ \langle \text{N\_CHILD} \rangle \ \langle \text{N\_CHILD} \rangle \\ \qquad \langle \text{N\_CHILD} \rangle \ ( \ \langle \text{N\_CHILD} \rangle \ )^+ \ \text{`}\langle > \text{3TREND} \rangle\text{'} \\ \langle \text{N\_CHILD} \rangle \to \langle \text{N\_ARY} \rangle \mid \langle \text{TERNARY} \rangle \mid \langle \text{BINARY} \rangle \mid \langle \text{LEAF} \rangle \end{cases}$$

A digression on semantic ambiguity. Despite the apparent clarity of the clause "Without changing the types and specifications of the ternary, binary and leaf nodes ...", it is questionable whether now a ternary node may also have n_ary child nodes. We chose to change grammar G as little as possible, so we opted in the negative sense. One might instead argue that (going back to the specifications), if a ternary node can have "any type of child node", then this will comprehend also the new node type, after allowing it into the language. Both viewpoints are acceptable, as it depends on the semantic ambiguity of the indefinite adjective "any" ... We preferred to avoid any addition that was not explicitly mentioned, as hinted by the parenthesis of the "any" clause.

3 Syntax Analysis and Parsing Methodologies 20%

1. Consider the following *EBNF* grammar G, represented as a machine net over the three-letter terminal alphabet $\Sigma = \{\ a,\ b,\ c\ \}$ (axiom S):



Answer the following questions:

(a) Draw the complete *ELR* pilot of grammar G, say if grammar G is of type $ELR\,(1)$ or not, and justify your answer.

(b) Find all the guide sets on all the arcs (terminal arcs, call arcs and exit arrows) of the machine net of grammar G, say if grammar G is of type $ELL\,(1)$ or not, and justify your answer.

(c) Say if grammar G is of type $ELL\,(k)$ for some $k \geq 2$, and reasonably justify your answer.

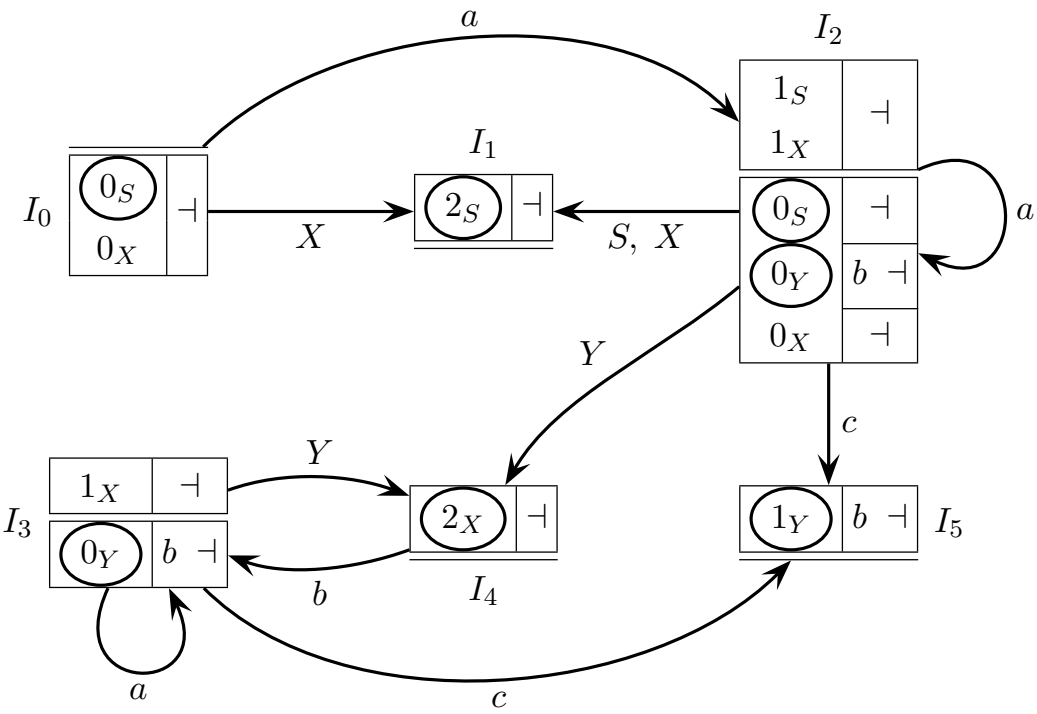
(d) (optional) Say if language $L\,(G)$ is of type $ELL\,(k)$ for some $k \geq 2$, and reasonably justify your answer.

Solution

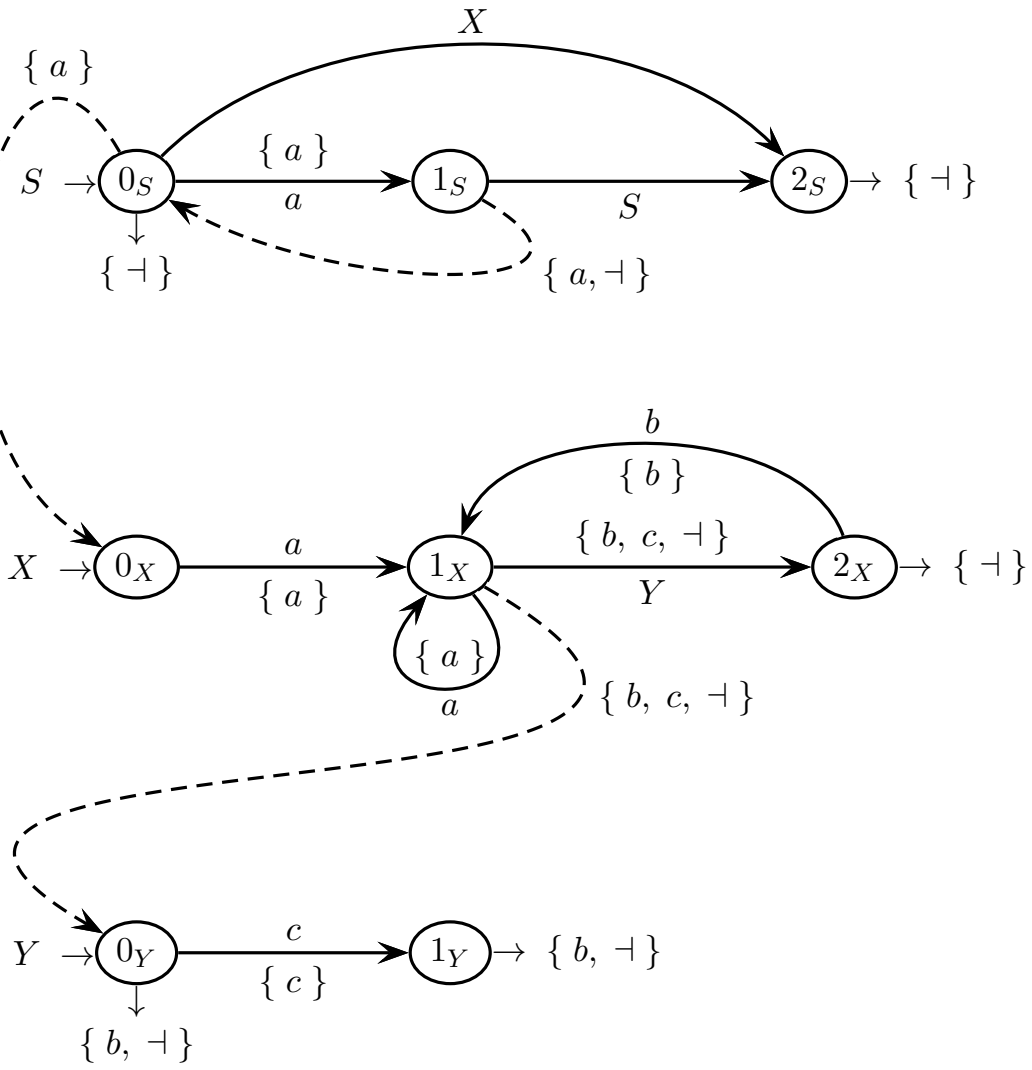
(a) The pilot is easy: it has 6 m-states and it has an *RR* conflict in the m-state I_2 (by the way, it does not have the *STP* property). Thus grammar G is not $ELR\,(1)$. Notice also that the string a is ambiguous, as it has these two derivations:

$$S \overset{S \to a\ S}{\Longrightarrow} a\ S \overset{S \to \varepsilon}{\Longrightarrow} a\ \varepsilon = a$$

$$S \overset{S \to X}{\Longrightarrow} X \overset{X \to a\ Y}{\Longrightarrow} a\ Y \overset{Y \to \varepsilon}{\Longrightarrow} a\ \varepsilon = a$$



(b) Here are all the call arcs and guide sets:



There are two overlapping guide sets on state 0_S with the same element a, thus grammar G is not $ELL(1)$ (and also because the pilot does not have the STP).

Caveat ! How is the guide set on the call arc $1_X \dashrightarrow 0_Y$ computed?

Correct answer: letter c is in, as c is an initial of nonterminal Y; letter b is in, as Y is nullable and b follows Y in the machine M_X; and the end-of-text $\dashv$ is in, as Y is at the end of M_X, whose prospect set is $\dashv$ (the shift arc $1_X \xrightarrow{Y} 2_X$ enters directly the final state 2_X and the prospect set of 2_X is $\dashv$).

Typical wrong answer: b and $\dashv$ are in, as they figure as the guide set on the exit arrow of the destination state 0_Y of the call arc $1_X \dashrightarrow 0_Y$, and therefore they back-propagate onto the call arc.

Remember: the guide set on an exit arrow of a final state does not back-propagate to the guide set of a call arc entering that final state.

A similar situation occurs for the call arc $1_S \dashrightarrow 0_S$.

(c) Both overlapping guide sets of length k contain elements of type a^k for any $k \geq 1$, thus grammar G is not $ELL(k)$ for any $k \geq 2$ (all the other guide sets on bifurcation states are not overlapping).

(d) Notice: language $L(Y)$ is finite, hence regular; language $L(X)$ is regular, as the only nonterminal Y that appears in the machine M_X generates a regular language; language $L(S)$ is generated by the extended rule $S \to a\ S \mid X \mid \varepsilon$ (obtained by finding a regular expression for machine M_S), which is right-linear. Therefore such a rule can be solved by the Arden identity and yields $L(S) = a^*\ (L(X) \cup \varepsilon)$. Thus language $L(S)$ is regular since language $L(X)$ is regular, as argued before. Since by definition it holds $L(G) = L(S)$, the language generated by grammar G is regular, whence it is $ELL(1)$.

4 Language Translation and Semantic Analysis 20%

1. Given a binary string w, we denote as $\overline{w}$ the string with the bits complemented, and as w_2 the string where every bit is repeated. For instance, if $w = 0\,1\,0\,0$ then $\overline{w} = 1\,0\,1\,1$ and $w_2 = 0\,0\,1\,1\,0\,0\,0\,0$.

   Consider now the following translation τ:

   $$\tau\left(\; w \;\; (\overline{w})^R \;\right) = \left((\overline{w})^R \right)_2$$

   For instance, if $w = 0\,1\,0\,0$ then:

   $$\tau\left(\; w \;\; (\overline{w})^R \;\right) = \tau\,(0\,1\,0\,0\,1\,1\,0\,1) = (1\,1\,0\,1)_2 = 1\,1\,1\,1\,0\,0\,1\,1$$

   Notice that the strings of the source language are composed of a sequence of bits followed by the same sequence complemented and reversed.

   Answer the following questions:

   (a) Design the syntactic translation scheme or grammar that defines translation τ.

   (b) Draw the syntax tree (for the translation grammar) or trees (for the translation scheme) for the example $\tau\,(0\,1\,0\,0\,1\,1\,0\,1) = (1\,1\,0\,1)_2 = 1\,1\,1\,1\,0\,0\,1\,1$.

   (c) (optional) Determine if the translation scheme or grammar is deterministic, and reasonably justify your answer.

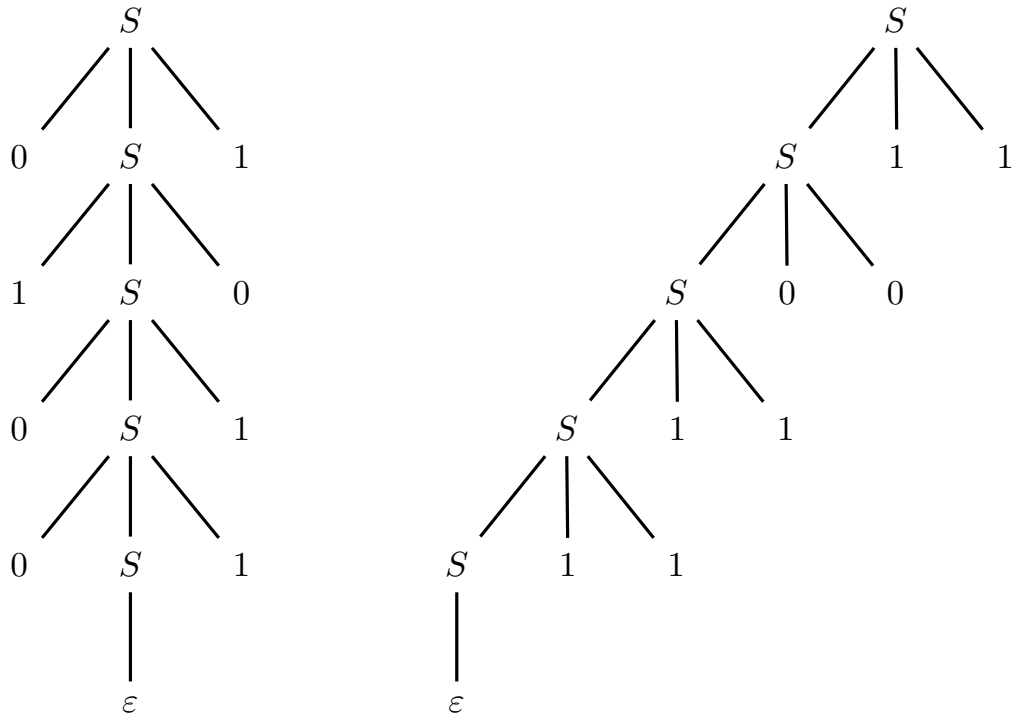
Solution

(a) Here is the translation scheme (axiom S):

$S \to 0\,S\,1$	$S \to S\,1\,1$
$S \to 1\,S\,0$	$S \to S\,0\,0$
$S \to \varepsilon$	$S \to \varepsilon$

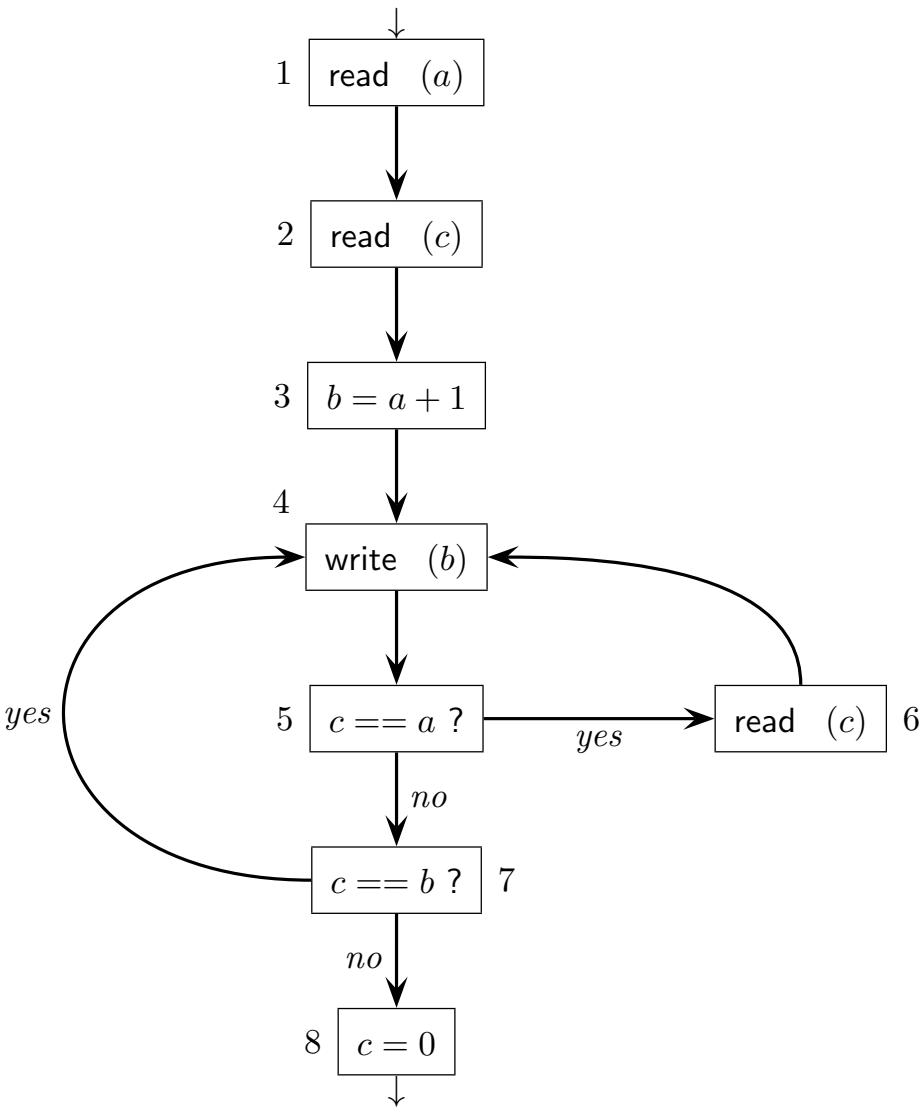
This scheme is clearly not ambiguous, so the translation associated is a function.

(b) Here are the trees for the source and target grammar.



(c) The source language is clearly nondeterministic, as it is a sort of "complemented palindrome" without a centre. As a consequence, the source grammar is also nondeterministic.

2. Consider the following control flow graph of a program with three integer variables a, b and c:



Answer the following questions:

(a) Write the regular expression R, over the node alphabet $\{1, \ldots, 8\}$, that models all the program execution traces, independently of the program semantics.

(b) Write the system of flow equations for the live variables of the program, solve such a system iteratively and write the variables live at each node (use the tables and graph prepared on the next pages).

(c) (optional) By considering the program semantics, write the regular expression R' of the effective execution traces modeled by the finite automaton represented by the graph, and say what changes to the distribution of the live variables are induced by the semantics.

---

please here write the **REGULAR EXPRESSION R**

node	*defined*
1	
2	
3	
4	
5	
6	
7	
8	

node	*used*
1	
2	
3	
4	
5	
6	
7	
8	

system of data-flow equations for **LIVE VARIABLES**

node	*in equations*	*out equations*
1		
2		
3		
4		
5		
6		
7		
8		

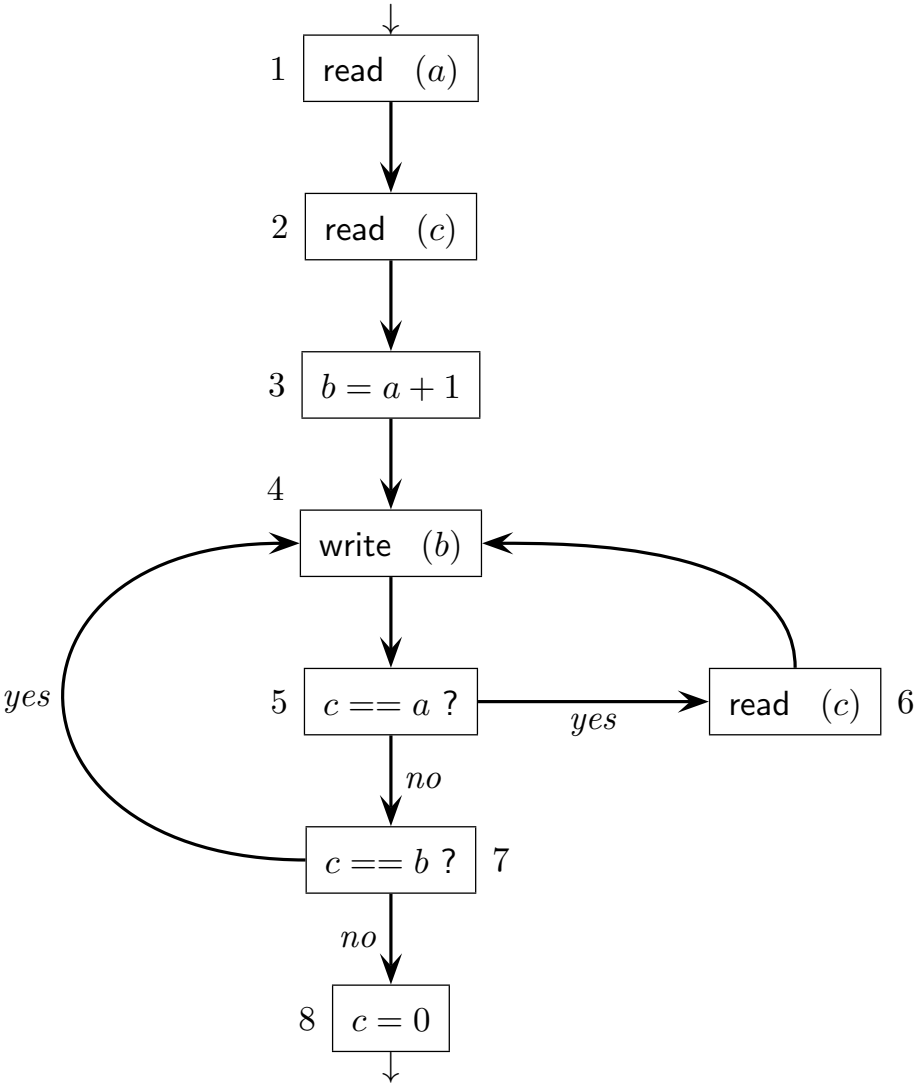
iterative solution table of the system of data-flow equations (**LIVE VAR.S**)

(the number of tables and columns is not significant)

	initialization		1		2		3		4		5		6	
#	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*
1														
2														
3														
4														
5														
6														
7														
8														

	7		8		9		10		11		12		13	
#	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*
1														
2														
3														
4														
5														
6														
7														
8														

please here write the **LIVE VARIABLES**



Solution

(a) Regular expression R:

$$R = 1\,2\,3\,\big(\,4\,5\,(\,6\;\mid\;7\,)\,\big)^{*}\,4\,5\,7\,8$$

(b) Variable tables:

node	*defined*
1	a
2	c
3	b
4	–
5	–
6	c
7	–
8	–

node	*used*
1	–
2	–
3	a
4	b
5	$a\ c$
6	–
7	$b\ c$
8	–

System of data-flow equations:

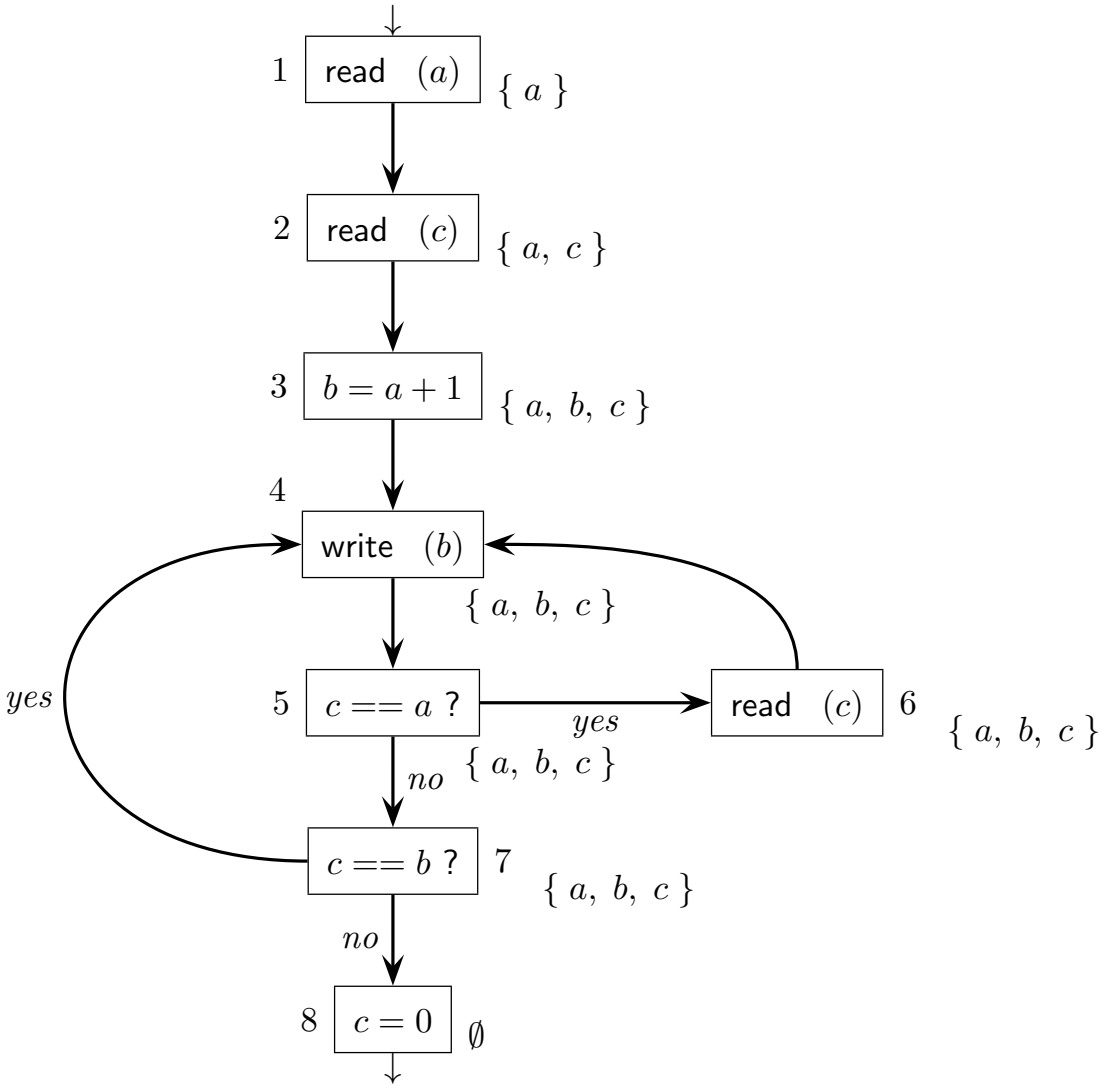
node	*in equations*	*out equations*
1	$\emptyset\ \cup\ \big(\ out(1) - \{\,a\,\}\ \big)$	$in(2)$
2	$\emptyset\ \cup\ \big(\ out(2) - \{\,c\,\}\ \big)$	$in(3)$
3	$\{\,a\,\}\ \cup\ \big(\ out(3) - \{\,b\,\}\ \big)$	$in(4)$
4	$\{\,b\,\}\ \cup\ \big(\ out(4) - \emptyset\ \big)$	$in(5)$
5	$\{\,a,\,c\,\}\ \cup\ \big(\ out(5) - \emptyset\ \big)$	$in(6)\ \cup\ in(7)$
6	$\emptyset\ \cup\ \big(\ out(6) - \{\,c\,\}\ \big)$	$in(4)$
7	$\{\,b,\,c\,\}\ \cup\ \big(\ out(7) - \emptyset\ \big)$	$in(4)\ \cup\ in(8)$
8	$\emptyset\ \cup\ \big(\ out(8) - \{\,c\,\}\ \big)$	$\emptyset$

Iterative solution table of the system of data-flow equations:

	initialization		1		2		3	
#	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*
1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	a	$\emptyset$	a	$\emptyset$
2	$\emptyset$	$\emptyset$	a	a	a	a	$a\ c$	a
3	$\emptyset$	a	b	a	$a\ b\ c$	$a\ c$	$a\ b\ c$	$a\ c$
4	$\emptyset$	b	$a\ c$	$a\ b\ c$	$a\ b\ c$	$a\ b\ c$	$a\ b\ c$	$a\ b\ c$
5	$\emptyset$	$a\ c$	$b\ c$	$a\ b\ c$	$b\ c$	$a\ b\ c$	$a\ b\ c$	$a\ b\ c$
6	$\emptyset$	$\emptyset$	b	b	$a\ b\ c$	$a\ b$	$a\ b\ c$	$a\ b$
7	$\emptyset$	$b\ c$	b	$b\ c$	$a\ b\ c$	$a\ b\ c$	$a\ b\ c$	$a\ b\ c$
8	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

The *in* columns of iteration 2 and 3 coincide, therefore the solution converges in three iterations.

Live variables (at the node outputs):



(c) Clearly the instruction sequence 4 5 7 cannot be executed more than once, else the loop 4 5 7 is endless. This is equivalent to saying that arc $7 \to 4$ can be deleted. Therefore:

$$R' = 1\,2\,3\ (4\,5\,6)^*\ 4\,5\,7\,8$$

Obviously variables a and b are no more live at (the output of) node 7.

More radically, node 7 becomes useless and can be canceled, by connecting node 5 directly to node 8. Therefore variables a and b are no more live at (the output of) node 5 either.