Formal Languages And Compilers
Exercises

Christian Rossi

Academic Year 2023-2024

Abstract

The lectures are about those topics:

- Definition of language, theory of formal languages, language operations, regular expressions, regular languages, finite deterministic and non-deterministic automata, BMC and Berry-Sethi algorithms, properties of the families of regular languages, nested lists and regular languages.
- Context-free grammars, context-free languages, syntax trees, grammar ambiguity, grammars of regular languages, properties of the families of context-free languages, main syntactic structures and limitations of the context-free languages.
- Analysis and recognition (parsing) of phrases, parsing algorithms and automata, push down automata, deterministic languages, bottom-up and recursive top-down syntactic analysis, complexity of recognition.
- Translations: syntax-driven, direct, inverse, syntactic. Transducer automata, and syntactic analysis and translation. Definition of semantics and semantic properties. Static flow analysis of programs. Semantic translation driven by syntax, semantic functions and attribute grammars, one-pass and multiple-pass computation of the attributes.

The laboratory sessions are about those topics:

- Modelisation of the lexicon and the syntax of a simple programming language (C-like).
- Design of a compiler for translation into an intermediate executable machine language (for a register-based processor).
- Use of the automated programming tools Flex and Bison for the construction of syntax-driven lexical and syntactic analyzers and translators.

Contents

CHAPTER 1

Exercise session I

1.1 Regular expression's equality

Given two regular expression:

$$R_1 = ((2b)^*c)^*$$
$$R_2 = (c^*(2b)^*)^*$$

Determine if they are equivalent. If not, provide a counterexample.

Solution Upon examination, it is evident that R_1 and R_2 are not equivalent due to the differing positions of the character c and its potential occurrence multiple times in R_2, while in R_1, it occurs exactly once. An illustrative counterexample is the string ab. This string is encompassed by the second language (R_2) but is not part of the first one (R_1).

1.2 Regular expression's ambiguity

Given the regular expression:

$$R_1 = (a|\varepsilon)^+(ba|bab)^*$$

Determine whether it is ambiguous.

Solution Firstly, we assign indices to each character in the regular expression, resulting in:

$$R_1 = (a_1|\varepsilon)^+(b_2a_3|b_4a_5b_6)^*$$

Next, we aim to find an ambiguous string to demonstrate the ambiguity of the regular expression. A regular expression is deemed ambiguous if there exists a string that can be matched in more than one way according to the regular expression. For instance, consider the string a_1, which can be generated multiple times by selecting ε $n-1$ times This observation establishes that the regular expression is indeed ambiguous.

1.3 Operations on languages

Given two regular expressions:

$$R_1 = a((b|bb)a)^+$$

$$R_2 = (ab)^*ba$$

- Define the quotient language $L = R_1 - R_2$.
- Write the three shortest strings of the language L.
- Write a regular expression that defines the language.

Solution Initially, we enumerate all characters in the given regular expressions as follows:

$$R_1 = a_1((b_2|b_3b_4)a_5)^+$$

$$R_2 = (a_1b_2)^*b_3a_4$$

The a_1 is surely a prefix for every string in the language, and all the strings have a_5 as a suffix. The three shortest strings in this language are *aba*, *ababa*, and *abababa*.

In the case of the second regular expression, every generated string starts with *ab* and concludes with a single *ba*. The three shortest strings in this language are *ba*, *abba* and *abababa*.

It becomes apparent that all strings with the suffix *aba* or containing at least two instances of *bb* are certainly part of L. Consequently, the three shortest strings in L are *aba*, *ababa*, and *abbaba*. The regular expression defining the language is:

$$L = \{(a(b|bb))^*aba\} \cup \{(a(b|bb))^*abba(a(b|bb))^+abba\}$$

CHAPTER 2

Exercise session II

2.1 Regular expressions and FSA

Consider the regular expression R below, over the three-letter alphabet $\Sigma = \{a, b, c\}$

$$R = a\left(b|c^{+}a\right)^{*}$$

Answer the following questions:

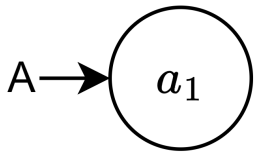
1. Write all the strings x of the language of R that have a length less than or equal to four, i.e., $x \in L(R)$ with $|x| \leq 4$, in lexicographic order (with $a < b < c$).
2. By means of the Berry-Sethi method, find a deterministic automaton A equivalent to the regular expression R.
3. Is the deterministic automaton A found before minimal? Justify your answer.
4. By means of the Brzozowski method (node elimination), starting from the automaton A found before, obtain a regular expression $R^{'}$ equivalent to A.
5. Is the language $L(R)$ locally testable? Formally prove your answer.

Solution

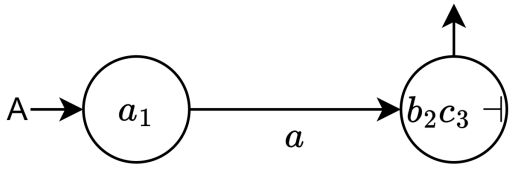
1. The possible strings include: *a*, *ab*, *abb*, *aca*, *abba*, *abca*, *acab*, *acca*.
2. Firstly, we enumerate the symbols for $R_{\#} = a_1\left(b_2|c_3^{+}a_4\right)^{*} \dashv$. We construct the following support table:

Initials	a_1
Terminals	Followers
a_1	$b_2 c_3 \dashv$
b_2	$b_2 c_3 \dashv$
c_3	$c_3 a_4$
a_4	$b_2 c_3 \dashv$

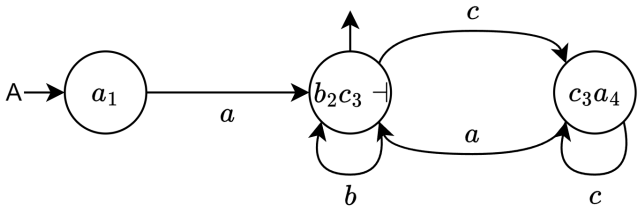
We begin with the initial state and proceed as follows:



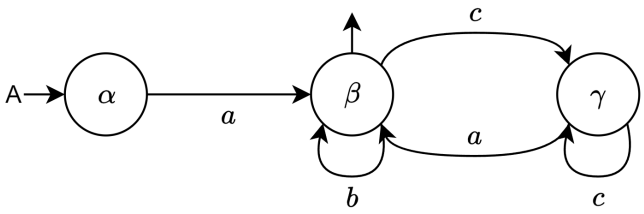
Next, we create a state reachable from a_1, resulting in:



After performing these steps for all states, we obtain the following automaton:

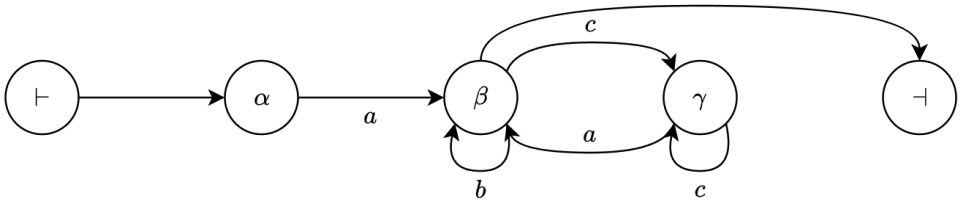


3. Automaton reduction involves minimizing the number of states. To simplify the automaton, we start by renaming the states:



We observe that α cannot be merged with β due to one being final and the other not. The same holds for β and γ. States α and γ cannot be merged due to differing transitions. Thus, the three states are distinguishable, and the automaton is minimal.

4. We begin by creating a virtual initial and final node connected to the automaton:



We then remove states one by one until reaching the final state directly from the initial one. We start by removing γ with the loop c^+a. We have two cycles on β that can be

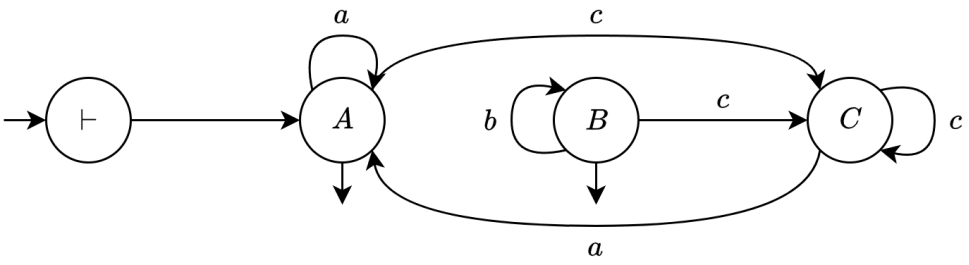
substituted with the expression $(b|c^{+}a)^{*}$. The only state remaining is α, and we finally have:

$$R^{'} = a(b|c^{+}a)^{*}$$

5. The sets are as follows:

   - Initials: $\{a\}$
   - Finals: $\{a, b\}$
   - Digrams: $\{aa, ac, bb, bc, ca, cc\}$

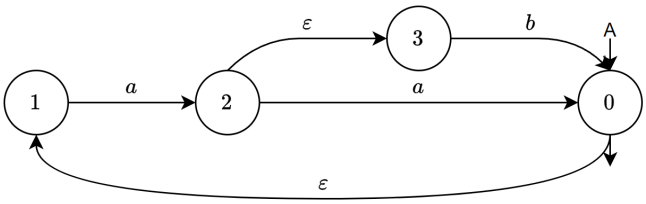
   Using these sets, we construct a specific automaton $A^{'}$. The initial set is connected to the set of initials, the final states are those in the finals set, and the transitions are based on the digrams set.



   The automaton is locally testable if it recognizes the initial language, and the edge reached by an arc has the same name as the transition. In this case, we constructed the nodes such that the second property is satisfied. We observe that states a and b are not distinguishable, allowing us to reduce the automaton to A. Therefore, the language is locally testable.

2.2 Regular expressions and FSA

Take a two-letter alphabet $\Sigma = \{a, b\}$. Consider the nondeterministic automaton A over Σ below, which has spontaneous transitions (ε-transitions):



And consider also the regular expression R over Σ below:

$$R = (aa|ab|ba)^{*}$$

Answer the following questions:

1. Find all the valid strings of language $L(A)$ that have a length less or equal than four. Do the same for language $L(R)$, too.

2. By using the back propagation method, eliminate the spontaneous transitions of automaton A and obtain an equivalent deterministic automaton A_1 without spontaneous transitions. Clean and minimize automaton A_1, if necessary. Test automaton A_1 with the valid short strings found at point one for automaton A.

3. By using the Berry-Sethi method, obtain a deterministic automaton A_2 equivalent to the regular expression R. Minimize automaton A_2, if necessary. Test automaton A_2 with the valid short strings found at point one for expression R.

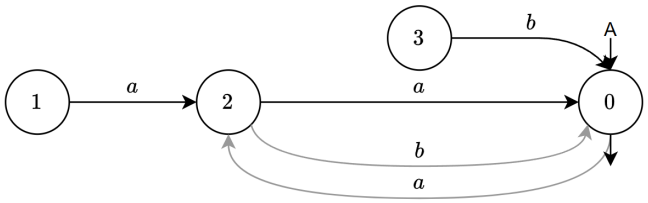
4. Say if language $L(R)$ is local or not, and shortly justify your answer.

5. By using a systematic method of your choice, obtain a deterministic automaton A_3 for the difference language $L_D = L(R) - L(A)$, i.e., $L(R) \cap \overline{L(A)}$, which contains the strings generated by R that are not recognized by A. Then test automaton A_3: by examining (only) the strings of length 4 found at point one, find in alphabetical order the first two strings x and y such that $x \in L_D$ and $y \notin L_D$, and show that $x \in L(A_3)$ and $y \notin L(A_3)$.

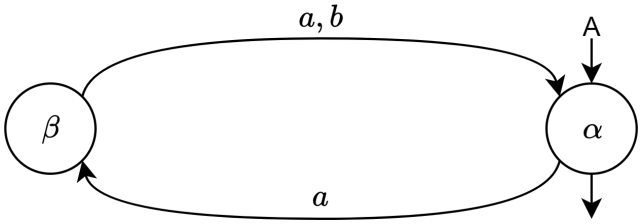
Solution

1. The strings in the language $L(A)$ are ε, *aa*, *ab*, *aaaa*, *aaab*, *abaa*, and *abab*. The strings in the language $L(R)$ are ε, *aa*, *ab*, *ba*, *aaaa*, *aaab*, *aaba*, *abaa*, *abab*, *abba*, *baaa*, *baab*, and *baba*.

2. The backpropagation method involves starting from the arriving state of an ε transition, eliminating the spontaneous transition, and backpropagating the transition exiting from the state. In this case, the resulting automaton is:



After that we can remove the useless states and rename the remaining ones.

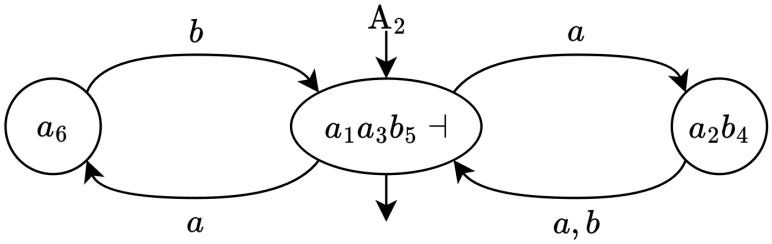


The states are undistinguishable, so the automaton is the minimal one, and it is the same as the initial one.

3. Firstly, we enumerate the symbols $R_{\#} = (a_1a_2|a_3b_4|b_5a_6)^* \dashv$. We construct the following support table:

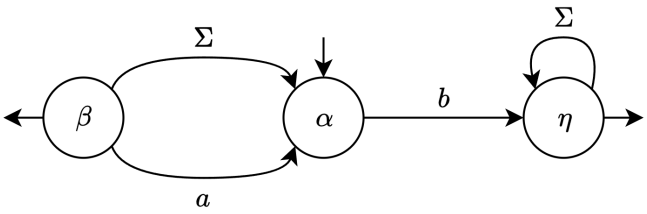
Initials	$a_1 a_3 a_5 \dashv$
Terminals	Followers
a_1	a_2
a_2	$a_1 a_3 a_5 \dashv$
a_3	b_4
b_4	$a_1 a_3 a_5 \dashv$
b_5	a_6
a_6	$a_1 a_3 a_5 \dashv$

The resulting automaton is already minimal:

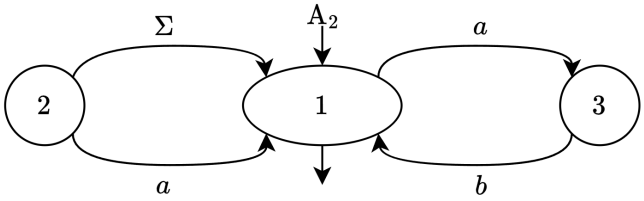


4. The language is not local. Although automaton A_2 is minimal, it is not local. The minimal automaton of a local language is necessarily local.

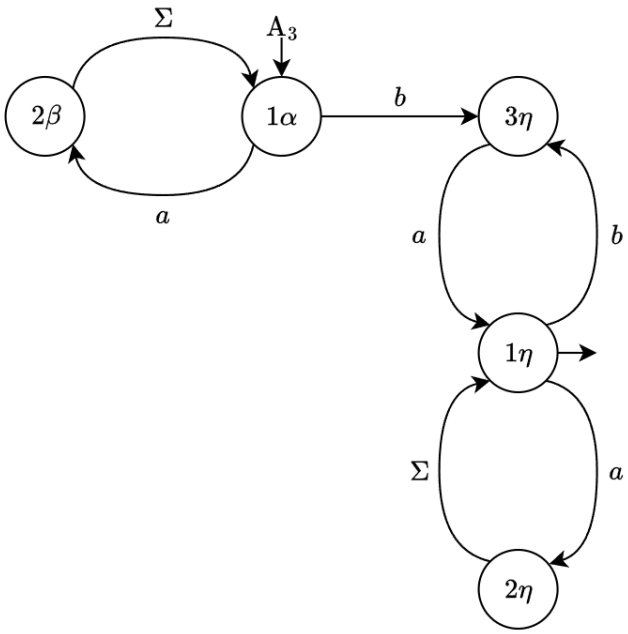
5. The language $L_D = L(R) - L(A) = L(R) \cup \overline{L(A)}$. To find the complement of automaton A, we switch final and non-final states and add an error state η. This results in:



Rewriting automaton A_2:



By connecting states with both transitions, we obtain:



Automaton A_3 has only five states, is deterministic, and is clean. While it may not be minimal, it passes a test where alphabetically listing the length-4 strings of language $L(R)$ and $L(A)$ reveals that A_3 correctly recognizes certain strings.

CHAPTER 3

Exercise session III

3.1 Free Grammars and PDA

Given the alphabet $\Sigma = \{a, b\}$ and the languages $L_1 = \{ww^R | w \in \Sigma^+\}$ and $L_2 = (b^*a^*b^*)$.

1. Define a grammar for L_1 and L_2.
2. Find the intersection of the two grammars.
3. Find the intersection $L_1 \cap (b^*a^*b^*a^*)$

Solution

1. The grammar for the first language is as follows:

$$\begin{cases} S_1 \to aS_1a \\ S_1 \to bS_1b \\ S_1 \to \varepsilon \end{cases}$$

   The grammar for the second language is:

$$\begin{cases} S_1 \to bS_1 | X \\ X \to aX | Y \\ Y \to bY | \varepsilon \end{cases}$$

2. The intersection between the two grammars is the language $L = (b^n a^m b^n)$ where $n \geq 0$ and $m \geq 0$ is an even number. The corresponding grammar is:

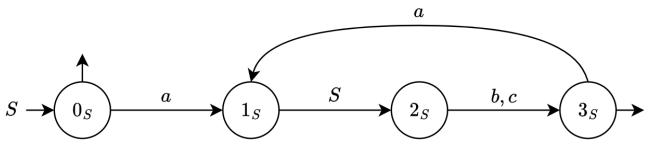
$$\begin{cases} S_1 \to bS_1b | X \\ X \to aXa | \varepsilon \end{cases}$$

3. The intersection of the languages is given by the grammar:

$$\begin{cases} S_1 \to X | U \\ X \to bXb | Y \\ Y \to aYa | \varepsilon \\ U \to aUa \end{cases}$$

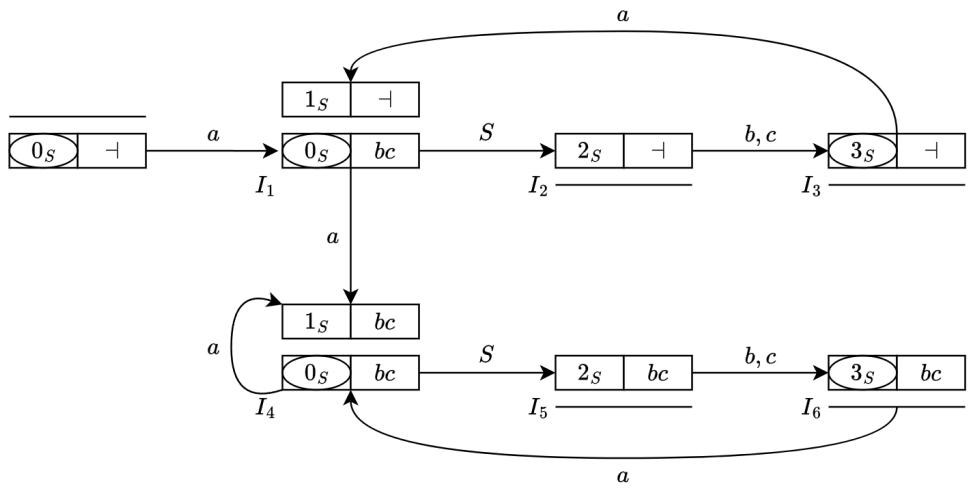
3.2 Syntax analysis and parsing

Consider the grammar G below, represented as a machine net, over a three-letter terminal alphabet $\Sigma = \{a, b, c\}$ and a one-letter non-terminal alphabet (axiom S):



Draw the complete pilot of grammar G, check for the conflicts, and show that grammar G is ELR(1).

Solution The initial state mirrors the one depicted in the provided machine. The final states are denoted by a circled name. The machine is illustrated below:



The shift/reduce conflicts are present if an outgoing arc from the terminal states has a label with a symbol in the look ahead of the terminal states. In this case we have no shift/reduce conflicts.

The reduce/reduce conflicts are present if an m-state have two final states and at least one shared symbol in the two look ahead of the final state. In this case we have no reduce/reduce conflicts.

Convergence conflicts manifest under the following conditions:

- Loss of the single arc properties (non-determinism).
- Presence of two incoming arcs in an m-state with identical labels.
- Existence of a shared look ahead in the convergent m-states.

In this case we have no convergence conflicts.

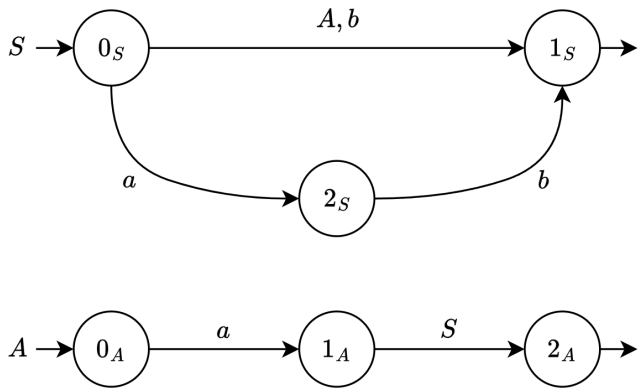
Given the absence of conflicts, it is evident that the grammar adheres to the ELR(1) property.

CHAPTER 4

Exercise session IV

4.1 Syntax analysis and parsing

Consider the following grammar G, represented as a machine net over the two-letter terminal alphabet $\Sigma = \{a, b\}$ and the two-letter non-terminal alphabet $V = \{S, A\}$ (axiom S).

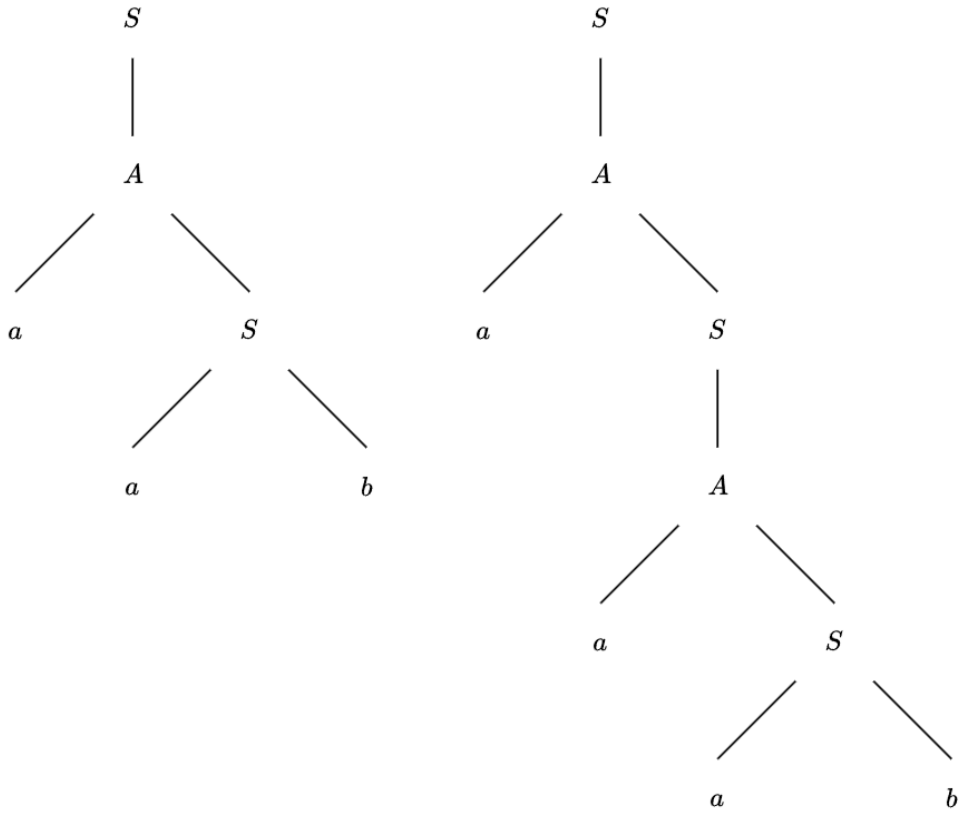


Answer the following questions:

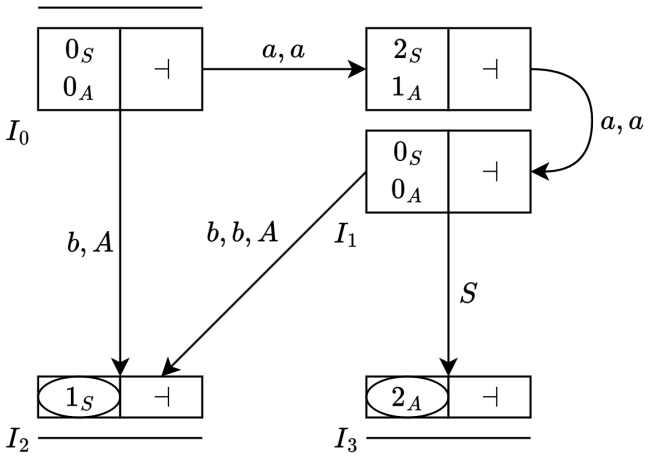
1. Draw the syntax tree (or trees if there are two or more) of the valid string *aab*.
2. Draw the complete pilot of grammar G, say if grammar G is of type ELR (1), and shortly justify your answer. If the grammar is not ELR(1) then highlight all the conflicts in the pilot.
3. Write all the guide sets on the arcs of the machine net (shift and call arcs, and final arrows), say if grammar G is of type ELL(1), based on the guide sets, and shortly justify your answer. If you wish, you can use the figure above to add the call arcs and annotate the guide sets.
4. Analyze the valid string *aab* using the Earley method; show the item/all the items that gives/give evidence of string acceptance.

Solution

1. The string *aab* exhibits ambiguity, leading to two distinct syntax trees:

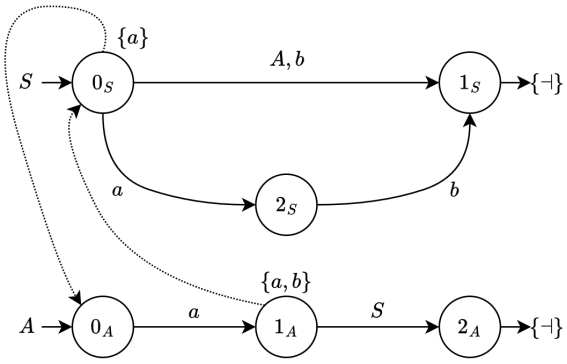


2. The pilot for the given grammar is as follows:



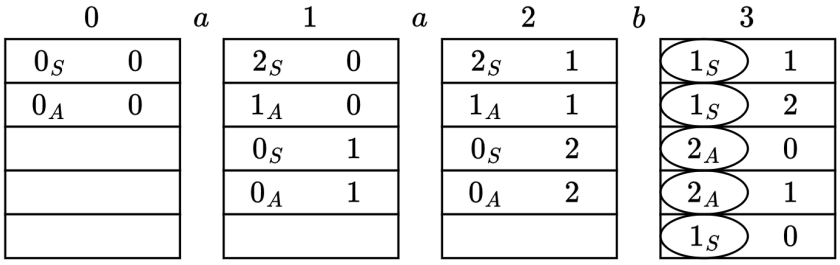
Notably, there are no reduce/reduce conflicts or shift/reduce conflicts. However, a convergence conflict arises from the b transitions originating from the same m-state, converging in the same m-state, and sharing an identical symbol in both convergent states. Consequently, the grammar is not ELR(1).

3. The guide sets of the machine net, completed with call arcs to construct the Predictive Control Flow Graph (PCFG), are outlined below:



The guide sets on the terminal shift arcs, although trivial, are not displayed. The grammar is not ELL(1) due to overlapping guide sets at the bifurcation state 0_S, indicating non-determinism.

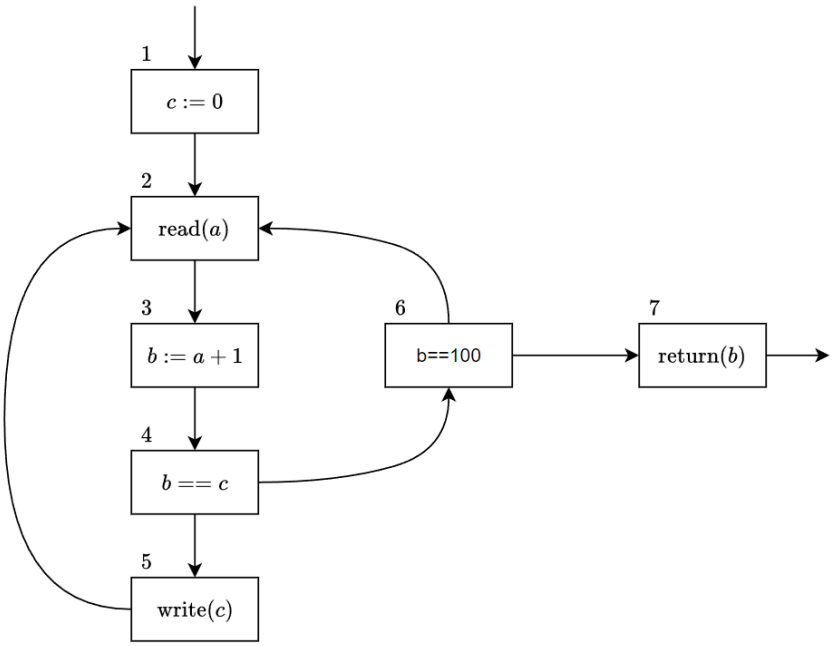
4. The Earley vector for the string *aab* is presented below:



The final item $\langle 1_S, 0\rangle$ in the last vector element signifies the acceptance of the string *aab* and corresponds to the two parsing trees associated with the string.

4.2 Static control flow analysis

Consider the program Control Flow Graph (CFG) below, with seven nodes:

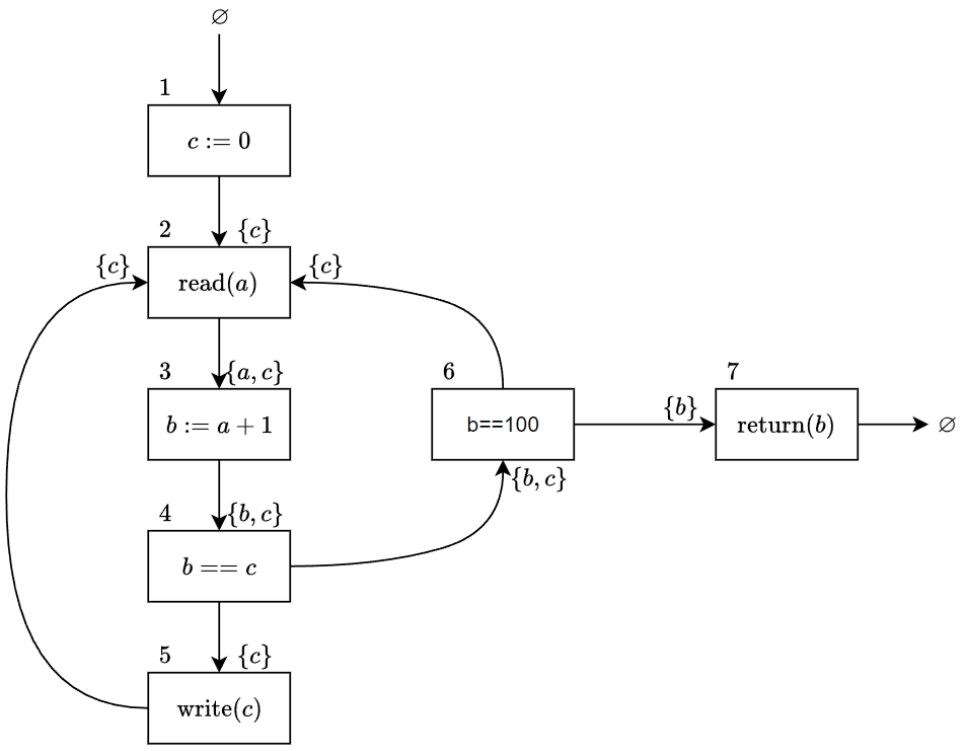


Answer the following questions:

1. Informally find the live variables at the input of each node of the CFG.
2. Can variables a and b share the same memory cell?
3. Find again the live variables at the input of each CFG node, through the data-flow equation method. Verify that the result is coherent with point one.

Solution

1. At the input of node 1 (initial) no variable is live, since every variable is assigned in the initial node itself or in some successor thereof before using. At the output of node 7 (final), by definition no variable is live. The other liveness intervals are just an application of the liveness definition. For instance, variable c is live at the inputs of all the nodes in the two loops (2-3-4-5 and 2-3-4-6), since it is used inside both loops (in the node 4) and is never reassigned after being initialized in the node 1 outside the loops. Variables a and b are even simpler: a is assigned in 2 and is used in 3, thus it is live only at 3; and b is assigned in 3 and is used in 4, 6 and 7, respectively first, second and third successor of 3, thus it is live at 4, 6 and 7 themselves.



2. Yes, variables a and b can share the same location. In fact, they are not simultaneously live at any program node, thus they can share the same memory cell or processor register.
3. We can construct the following table:

	Defined	**Used**
1	c	
2	a	
3	b	a
4		b, c
5		c
6		b
7		b

We have that:

Node i	**In equations** $\text{in}(i) = (\text{out}(i) - \text{def}) \cup \text{used}$	**Out equations** $\text{out}(i) = \bigcup_{j \in \text{succ}(i)} \text{in}(j)$
1	$\text{in}(1) = \text{out}(1) - \{c\}$	$\text{out}(1) = \text{in}(2)$
2	$\text{in}(2) = \text{out}(2) - \{a\}$	$\text{out}(2) = \text{in}(3)$
3	$\text{in}(3) = (\text{out}(3) - \{b\}) \cup \{a\}$	$\text{out}(3) = \text{in}(4)$
4	$\text{in}(4) = \text{out}(4) \cup \{b, c\}$	$\text{out}(4) = \text{in}(5) \cup \text{in}(6)$
5	$\text{in}(5) = \text{out}(5) \cup \{c\}$	$\text{out}(5) = \text{in}(2)$
6	$\text{in}(6) = \text{out}(6) \cup \{b\}$	$\text{out}(6) = \text{in}(7) \cup \text{in}(2)$
7	$\text{in}(7) = \text{out}(7) \cup \{b\}$	$\text{out}(7) = \varnothing$

And the iterations are:

	Initialization		**1**		**2**		**3**		**4**	
#	out	in	out	in	out	in	out	in	out	in
1	-	-	-	-	-	-	c	-	c	-
2	-	-	a	-	ac	c	ac	c	ac	-
3	-	a	bc	ac	bc	ac	bc	ac	bc	-
4	-	bc	bc	bc	bc	bc	bc	bc	bc	-
5	-	c	-	c	-	c	c	c	c	-
6	-	b	b	b	b	b	bc	bc	bc	-
7	-	b	-	b	-	b	-	b	-	-

The out columns at steps 3 and 4 coincide, thus the iteration process converges in three steps. The in column at step 3 lists all the variables live at the program node inputs. The live variable latest to reach a node input is c at node 6 (compare the in columns at steps 2 and 3). In fact, variable c propagates backwards from node 4, where it is used, to node 6, and to do so it has to cross nodes 3 and 2 in succession, which just takes three steps in total. All the other propagation cases are fast. The systematic solution obtained through the data-flow method is identical to the informal one of point one.

CHAPTER 5

Exercise session V

5.1 Language translation, semantic analysis

Consider the source language L_S generated by the grammar G_S, over the two-letter alphabet $\Sigma = 0, 1$:

$$G_S = \begin{cases} S \to 0Q|1R \\ Q \to 0S|1S|0|1 \\ R \to 0S|1S|0|1 \end{cases}$$

1. Write a syntax scheme G_τ (or a translation grammar) for the translation τ over the source language L_S defined as follows: all couples of binary number must be translated into base four numbers. For example:

   $$100111000 \to 2130$$

   The grammar generates strings of even length greater or equal than two.

2. Define the regular translation expression.

Solution

1. The translation grammar is defined as follows:

$$G_\tau = \begin{cases} S \to \dfrac{0}{\varepsilon}Q|\dfrac{1}{\varepsilon}R \\ Q \to \dfrac{0}{0}S|\dfrac{1}{1}S|\dfrac{0}{0}|\dfrac{1}{1} \\ R \to \dfrac{0}{2}S|\dfrac{1}{3}S|\dfrac{0}{2}|\dfrac{1}{3} \end{cases}$$

2. The corresponding regular translation expression is given by:

$$T_\tau = \left(\frac{00}{0}|\frac{01}{1}|\frac{10}{2}|\frac{11}{3}\right)^+$$

The associated translation automaton is depicted below: