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.
- Syntax-driven translation, direct and inverse translation, syntactic translation schemata, 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:

- Modellization 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:

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

Check if they are equal. If they are not give a counterexample.

Solution

It is possible to see that R_1 and R_2 are not equivalent because the character c is in a different position and can be found multiple times in R_2, while in R_1 is found exactly one time. An example can be ab. This string is included in the second language, but not in the first one.

1.2 Regular expression's ambiguity

Given the regular expression:

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

check if it is ambiguous.

Solution

First, we enumerate all the characters in the regular expression, obtaining:

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

and now we try to come up with an ambiguous string to prove that the regular expression is ambiguous. A regular expression is considered ambiguous if there is a string which can be matched by more than one way from the regular expression. For instance, we can have the string a_1 can be generated multiple times selecting the ε $n-1$ times. This proves that the regular expression is ambiguous.

1.3 Operations on languages

Given two regular expressions:

$$R_1 = a((b|bb)a)^+ \qquad R_2 = (ab)^*ba$$

Define the quotient language $L = R_1 - R_2$.

1. Write the three shortest strings of the language L.
2. Write a regular expression that defines the language.

Solution

First, we enumerate all the characters in the regular expressions, obtaining:

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

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

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

For the second regular expression we have that every string generated starts with *ab* and have a single *ba* as a suffix. The shortest strings of this language are: *ba*, *abba* and *abababa*.

We can see that all the strings that have the suffix *aba* or have at least two *bb* are certainly in L.

1. Now, we can see that the three shortest strings are: *aba*, *ababa*, and *abbaba*.
2. The regular expression 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.

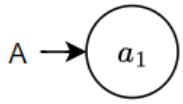
Solution

1. The possible strings are: a, ab, abb, aca, $abba$, $abca$, $acab$, $acca$.

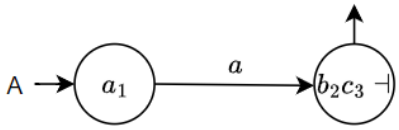
2. First, we enumerate the symbols $R_{\#} = a_1\left(b_2|c_3^{+}a_4\right)^{*} \dashv$. We construct the following support table:

Initials	a_1
Terminals	Followers
a_1	$b_2c_3 \dashv$
b_2	$b_2c_3 \dashv$
c_3	c_3a_4
a_4	$b_2c_3 \dashv$

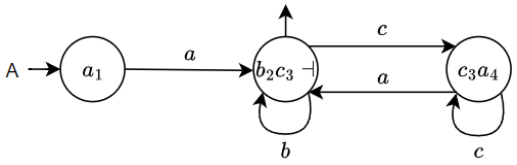
We start wit the initial, and we have:



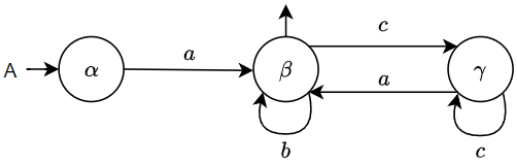
Now we create the state reachable from a_1, and we obtain:



After doing this steps for all the states we obtain the following automaton:



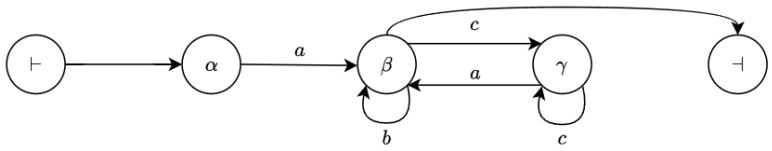
3. We can reduce an automaton if we can reduce the number of states. We have to use some criterions to do the check on automaton A. To simplify the automaton we start by renaming the states:



We can see that α cannot be merged with β because one is not final and the other one is final. Same reasoning holds for β and γ. States

α and γ cannot be merged because they have different transitions. So, the three states are distinguishable, and the automaton is the minimal.

4. We start by creating a virtual initial node and final node connected to the automata:



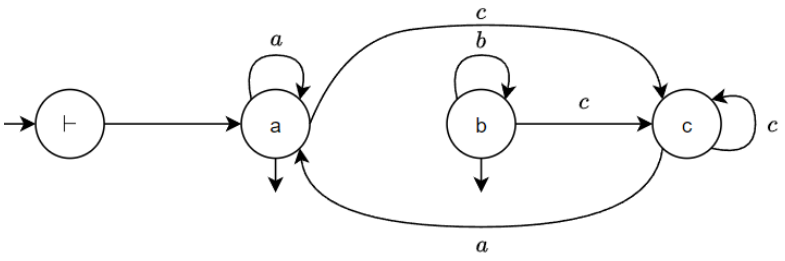
Now we can remove the states one by one until we reach the final state directly from the initial one. So, we remove γ with the loop c^+a. We have two cycles on β that can be substituted by the expression $(b|c^+a)^*$. We have only α that can be easily removed, and we finally have that:

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

5. We have the following sets:

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

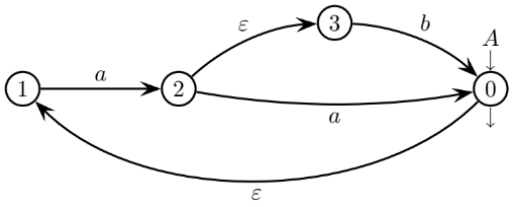
Using these sets we can build a particular automaton $A^{'}$, that have the initial set connected to the set of initials, the final states are the ones in the finals set, and the transitions are the one belonging to the digrams set.



The automaton is local if it recognizes the initial language and if the edge reached by an arc has the same name of the transition. In this case we constructed the nodes such that the second property is satisfied, so we simply need to check the first property. We can not that the states a and b are not distinguishable, so we can reduce the automaton to the A one. So, 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.

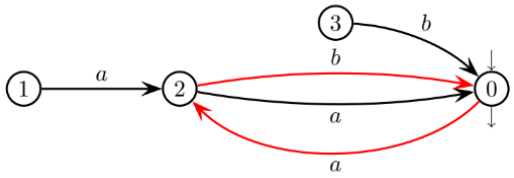
4. Say if language $L(R)$ is local or not, and shortly justify your answer.

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 $y \notin L(A_3)$.

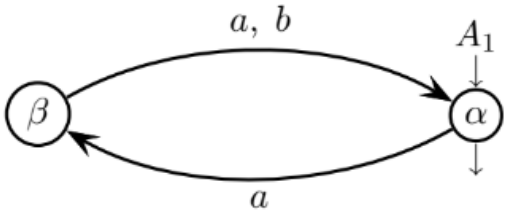
Solution

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

2. The idea of the back propagation method is to start from the arriving state of an ε transition, eliminate the spontaneous transition and back propagate the transition exiting from the state. In this case we obtain the following automaton:



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

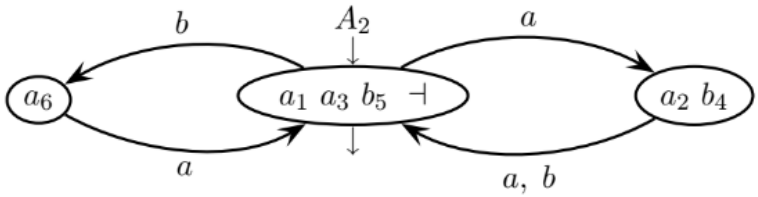


   The states are un-distinguishable, so the automaton is the minimal one, and it is the same as the initial one.

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

Initials	$a_1a_3a_5 \dashv$
Terminals	Followers
a_1	a_2
a_2	$a_1a_3a_5 \dashv$
a_3	b_4
b_4	$a_1a_3a_5 \dashv$
b_5	a_6
a_6	$a_1a_3a_5 \dashv$

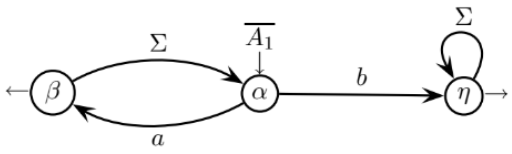
From which we construct the following automaton:



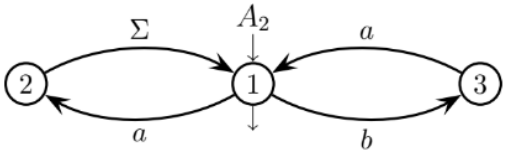
That is already minimal.

4. The language is not local. The minimal automaton of a local language is necessarily local. Automaton A_2 is minimal, but it is not local. Thus, language $L(R)$ is not local.

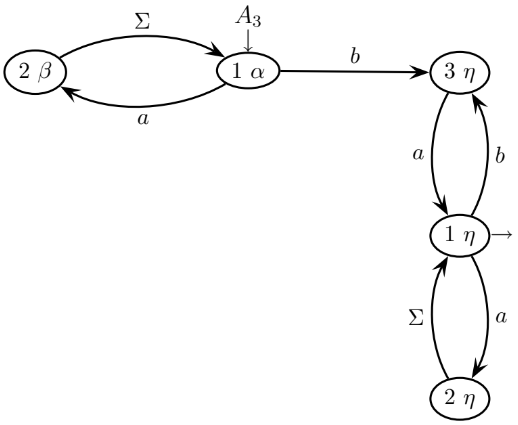
5. The language $L_D = L(R) - L(A) = L(R) \cup \overline{L(A)}$. We need to find the complement of the automaton A, by switching final and non-final states with the complement, and by adding an error state η. In this case we obtain:



We rewrite the automaton A_2



We can write all the states and then try to connect with both transitions, and so we obtain:



Automaton A_3 has five states only. It is deterministic, as both factor automata are so. It is clean, though it might not be minimal. Here is the test of automaton A_3. From question one, alphabetically list the length-4 strings of language $L(R)$ and, similarly, those of language $L(A)$. Then in the first list pick the first string x that is not in the second list, and the first string y that is also in the second list. Result: $x = aab$ a and $y = aaaa$. We soon see that $x \in L(A_3)$ and $y \notin L(A_3)$. Thus, automaton A_3 passes this test successfully.

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 the following:

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

   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:

$$L = (b^n a^m b^n)$$

where $n \geq 0$, and $m \geq 0$ an even number. The corresponding grammar is the following:

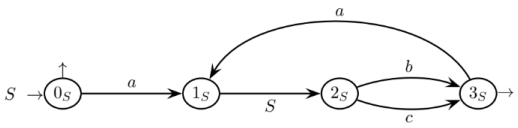
$$\begin{cases} S_1 \to bS_1b|X \\ X \to aXa|\varepsilon \end{cases}$$

3. The intersection of the languages is:

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

3.2 Syntax analysis and parsing

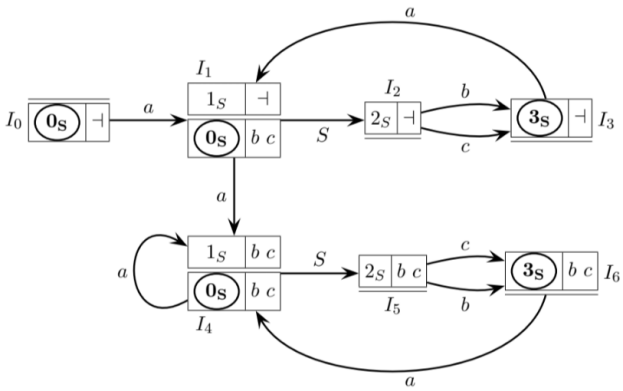
Consider the grammar G below, represented as a machine net, over a three-letter terminal alphabet $\Sigma = \{a, b, c\}$ and a one-letter non-terminal alphabet (axiom S):



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

Solution

The initial state is the same as the one in the given machine. The final state have a circle surrounding the name. The machine is the following:



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.

The convergence conflict arises if:

- We lose the single arc properties (non-determinism).
- We have two arcs incoming in an m-state with the same label.

- There is one shared look ahead in the convergent m-states.

In this case we have no convergence conflicts.

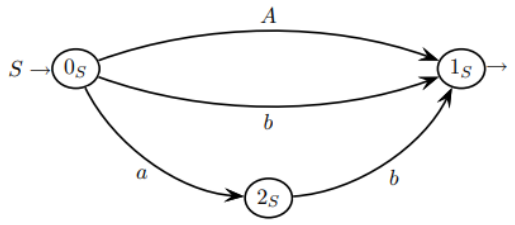
Since we have no conflicts, we are in ELR(1).

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.

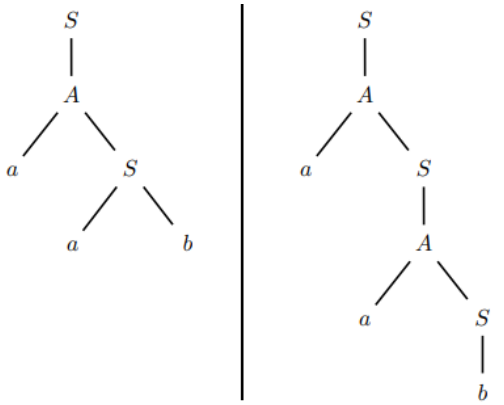
3. Write all the guide sets on the arcs of the machine net (shift and call arcs, and final arrows), say if grammar G is of type ELL(1), based on

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.

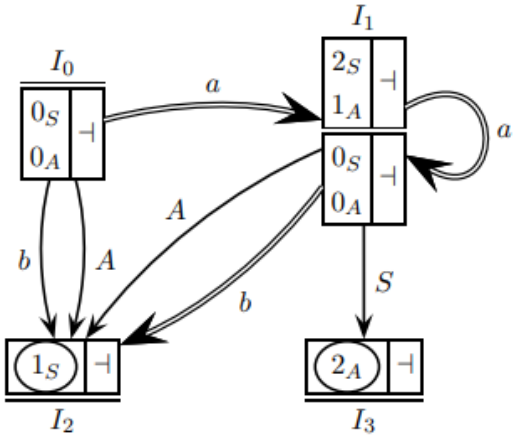
4. Analyze the valid string *aab* using the Earley method; show the item/all the items that gives/give evidence of string acceptance.

Solution

1. The string *aab* is ambiguous and admits these two syntax trees:

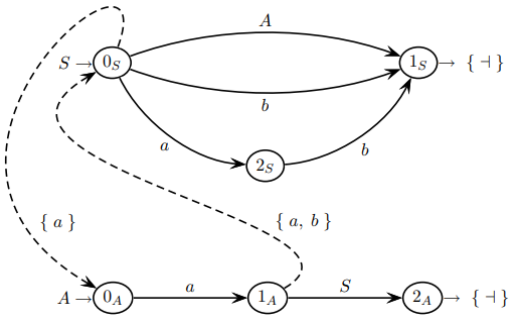


2. The pilot is the following:



No reduce/reduce conflicts, no shift/reduce conflicts. The b transitions comes from the same m-state, converges in the same m-state, and there is an equal symbol in both convergent states, so there is a convergence conflict. So the grammar is not ELR(1).

3. Here are all the guide sets of the machine net, completed with the call arcs to make the Predictive Control Flow Graph (PCFG):



The guide sets on the terminal shift arcs are trivial and not shown. The grammar is not ELL(1), because of the overlapping guide sets at the bifurcation state 0_S (we have non-determinism).

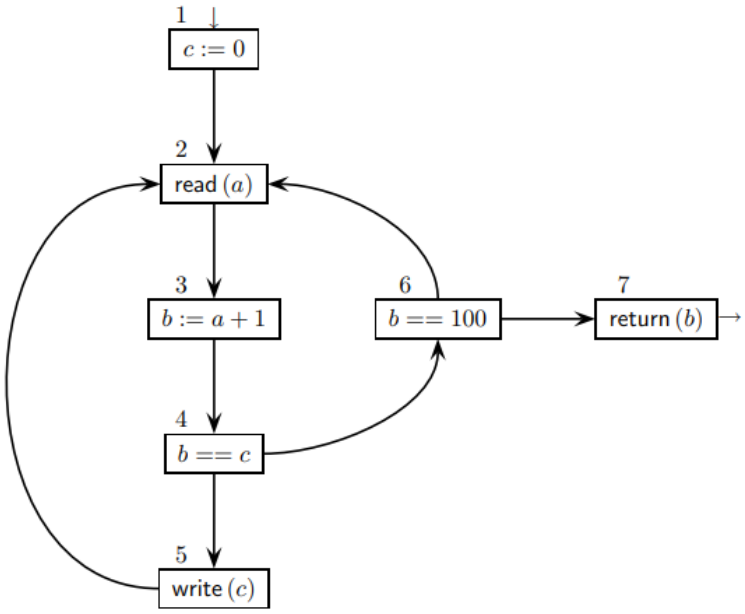
4. Here is the Earley vector for string *aab*:

0		a	1		a	2		b	3	
0_S	0		2_S	0		2_S	1		1_S	1
0_A	0		1_A	0		1_A	1		1_S	2
			0_S	1		0_S	2		2_A	0
			0_A	1		0_A	2		2_A	1
									1_S	0

The final item $\langle 1_S, 0 \rangle$ in the last vector element indicates that string *aab* is accepted and corresponds to the two parsing trees of the string *aab*.

4.2 Static control flow analysis

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



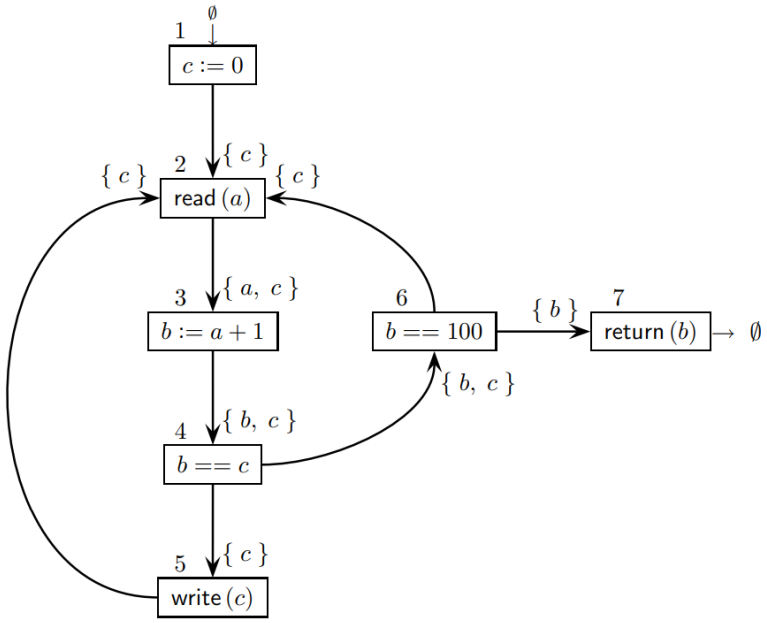
Answer the following questions:

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

And the iterations are:

	initializ.		1		2		3		4	
#	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*
1	–	–	–	–	–	–	c	–	c	
2	–	–	a	–	$a\ c$	c	$a\ c$	c	$a\ c$	
3	–	a	$b\ c$	$a\ c$	$b\ c$	$a\ c$	$b\ c$	$a\ c$	$b\ c$	
4	–	$b\ c$	$b\ c$	$b\ c$	$b\ c$	$b\ c$	$b\ c$	$b\ c$	$b\ c$	
5	–	c	–	c	–	c	c	c	c	
6	–	b	b	b	b	b	$b\ c$	$b\ c$	$b\ c$	
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