FORMAL LANGUAGES AND COMPILERS

prof.s Luca Breveglieri and Angelo Morzenti

Exam of Tue 21 February 2017 - Part Theory

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

LAST + FIRST NAME:

(capital letters please)

MATRICOLA: SIGNATURE:

(or PERSON CODE)

INSTRUCTIONS - READ CAREFULLY:

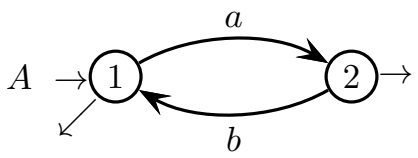
- The exam is in written form and consists of two parts:
  1. Theory (80%): Syntax and Semantics of Languages
     - regular expressions and finite automata
     - free grammars and pushdown automata
     - syntax analysis and parsing methodologies
     - language translation and semantic analysis
  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. Consider the following regular expression e, over the two-letter alphabet $\Sigma = \{\ a,\ b\ \}$:

$$e = (\ a\ b\ \mid\ a\ )^{+}\ (\ b\ a\ )^{*}$$

Consider also the finite-state automaton A below, over the same alphabet Σ, with initial state 1 and final states 1, 2:



Answer the following questions:

(a) Find the shortest ambiguous string of the regular language $L\,(e)$ and suitably justify your answer. Provide evidence that the string is ambiguous and that it is the shortest ambiguous one.

(b) By using the Berry-Sethi method, design a finite-state deterministic automaton B that accepts the language $L\,(e)$, and if necessary minimize the automaton B by applying the suitable minimization algorithm.

(c) By using a systematic method, design a finite-state automaton C that accepts the intersection language $L\,(e)\ \cap\ L\,(A)$.

(d) By using the node elimination method (Brzozowsky) applied to automaton A, find an equivalent regular expression r.

Solution

(a) Here is the numbered version $e_{\#}$ of the regular expression e:

$$e_{\#} = (\, a_1\ b_2 \ \mid\ a_3\,)^+\ (\, b_4\ a_5\,)^*$$

The two numbered strings $a_1\, b_2\, a_3$ and $a_3\, b_4\, a_5$ can be derived from $e_{\#}$, and they correspond to the same unnumbered string $a\, b\, a \in L\,(e)$. This proves that the regular expression e is ambiguous. It is not difficult to see that all the (infinitely many) strings $(\, a\ b\,)^+\ a$ generated by e are ambiguous as well.

Here are the strings of the language $L\,(e)$ that have length 1: only a; and those that have length 2: $a\, a$, $a\, b$ and $b\, a$. All such strings are non-ambiguous, as they do not have alternative derivation choices. In fact the following:

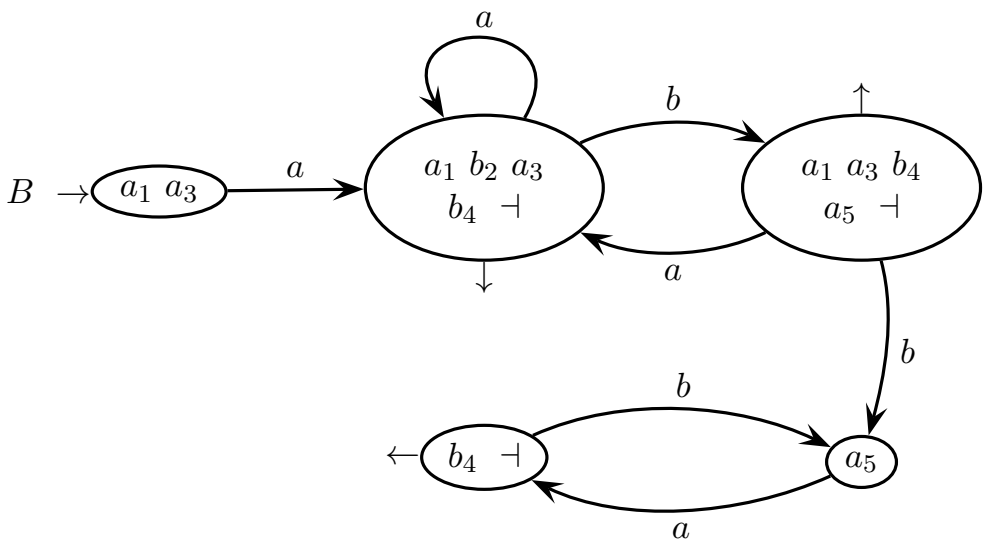
$$a\colon a_3 \qquad a\ a\colon a_3\ a_3 \qquad a\ b\colon a_1\ b_2 \qquad b\ a\colon b_4\ a_5$$

are the only derivation choices possible for each of these strings. Thus string $a\, b\, a$ is the shortest ambiguous one of the regular expression e. Furthermore, it is the only ambiguous string with length 3 generated by e, since the only other string of $L\,(e)$ that has length 3 is $a\, a\, a\colon a_3\, a_3\, a_3$, which clearly is not ambiguous.

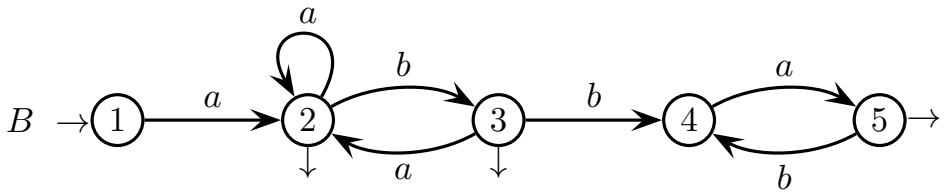
(b) Here is the Berry-Sethi automaton B obtained from the regular expression e. First, these are the initials and followers of e:

initials	$a_1\ a_3$
terminals	followers
a_1	b_2
b_2	$a_1\ a_3\ b_4\ \dashv$
a_3	$a_1\ a_3\ b_4\ \dashv$
b_4	a_5
a_5	$b_4\ \dashv$

Then, here is the state-transition graph of automaton B produced by the *BS* algorithm, with five states:

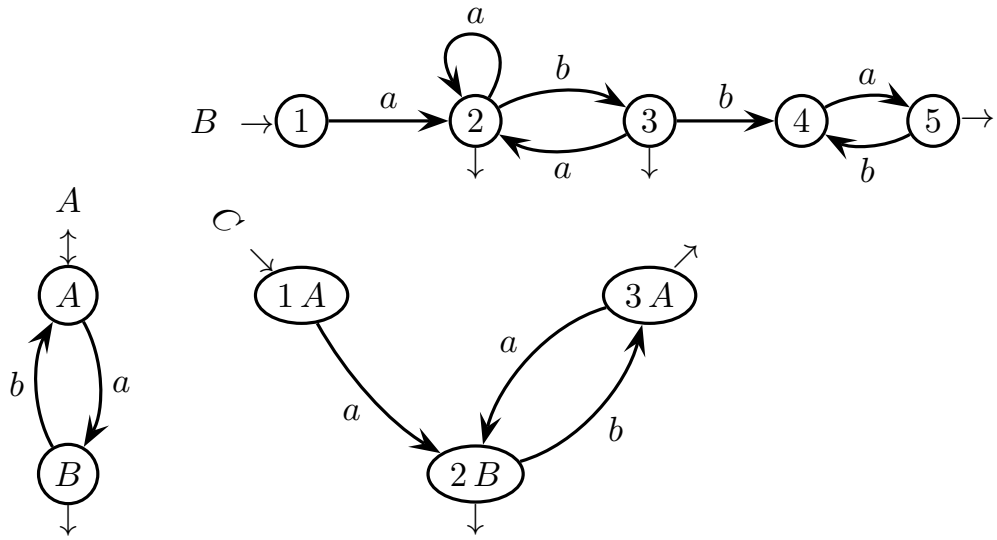


Concerning the minimality of automaton B (after a change of state names):



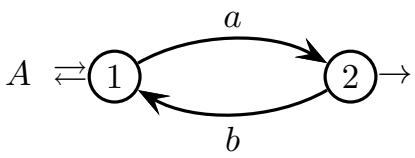
Automaton B is clean by construction (in fact all its states are accessible and post-accessible). Final states: state 5 is distinguishable from both states 2 and 3, as it does not have an outgoing a-arc; states 2 and 3 are distinguishable from each other as their outgoing b-arcs enter states 3 and 4, respectively, which are final and non-final hence distinguishable from each other. Non-final states: states 1 and 4 are distinguishable from each other, as their outgoing a-arcs enter states 2 and 5, respectively, which are already known to be distinguishable. In conclusion automaton B is in the minimal form.

(c) From before, it holds $L(e) = L(B)$. Here is the Cartesian product of the automaton B equivalent to the regular expression e, and of the automaton A:

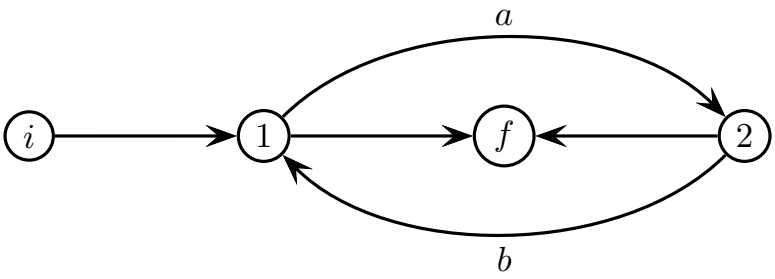


Clearly only three states of the product automaton C are both accessible (from state $1\,A$) and post-accessible (to states $2\,B$ or $3\,A$), whereas all the others (in principle automaton C has $2 \times 5 = 10$ states) are either inaccessible, or post-inaccessible, or both, and thus can be canceled, thus leaving a state-transition graph in the clean form. Furthermore, being the product of two deterministic automata, also automaton C is deterministic, and clearly it is already minimal, as the two final states are distinguishable since they have different outgoing arcs. Intuitively, a regular expression equivalent to the Cartesian product automaton $C = A \times B$ is $a\,(\,b\,a\,)^*\,[\,b\,] = (\,a\,b\,)^*\,a\,[\,b\,] = (\,a\,b\,)^+ \mid (\,a\,b\,)^*\,a$ (since it holds $a\,(\,b\,a\,)^* = (\,a\,b\,)^*\,a$). This regular expression comes from modifying the original regular expression $e = (\,a\,b \mid a\,)^+\,(\,b\,a\,)^*$, which is equivalent to automaton B, to make it satisfy the constraints imposed by intersecting with automaton A, namely, to have alternating letters a and b, and to have initial letter a.

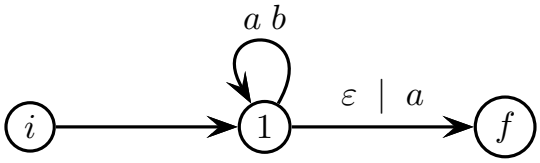
(d) Here is the node elimination (Brzozowsky) method applied to automaton A:



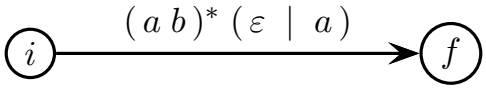
Unique initial and final states without ingoing and outgoing arcs, respectively:



Eliminate node 2:



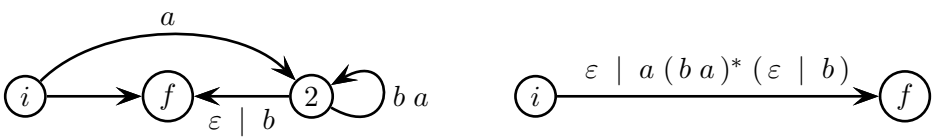
Eliminate node 1:



In conclusion, here is a (compact formulation of the) regular expression R:

$$R = (a\ b)^* \ (\varepsilon \ \mid \ a) = (a\ b)^* \ [a]$$

which is visibly equivalent to automaton A. Of course, the expression can be written in different equivalent forms, e.g., $R = (a\ b)^* \ \mid \ (a\ b)^*\ a = (a\ b)^* \ \mid \ a\ (b\ a)^*$, since it holds $(a\ b)^*\ a = a\ (b\ a)^*$.
If we had eliminated first node 1 then node 2, we would have obtained:



and consequently we would have had the following regular expression R':

$$R' = \varepsilon \ \mid \ a\ (b\ a)^* \ (\varepsilon \ \mid \ b\ ) = \big[a\ (b\ a)^* \ [b] \big]$$

equivalent to the regular expression R, though a little more complicated.

2 Free Grammars and Pushdown Automata 20%

1. Consider a two-letter alphabet $\Sigma = \{ a, b \}$ and the classical free language L of the palindromes without centre, below:

$$L = \left\{ x\, x^R \mid \quad x \in \Sigma^* \right\}$$

Answer the following questions:

(a) Write a *BNF* grammar G_1 (no matter if ambiguous) that generates the language L_1 of the (possibly empty) prefixes of language L, namely:

$$L_1 = \{ y \in \Sigma^* \mid \quad \exists\, z \in \Sigma^* \text{ such that } y\, z \in L \}$$

(b) Write a *BNF* grammar G_2 (no matter if ambiguous) that generates the language L_2 of the (possibly empty) substrings of language L, namely:

$$L_2 = \{ w \in \Sigma^* \mid \quad \exists\, u, v \in \Sigma^* \text{ such that } u\, w\, v \in L \}$$

(c) (optional) Determine if grammars G_1 and G_2 are ambiguous and justify your answer: if a grammar is ambiguous then provide a string with two or more syntax trees, else argue that it does not generate any ambiguous string.

Solution

(a) The question has two simple solutions, depending of how closely the structure of language L_1 is analyzed. By definition language L_1 contains the palindrome language itself, and in addition to it all the proper palindrome prefixes. Thus:

i. A (possibly empty) prefix of a (non-centred) palindrome is a generic (possibly empty) string followed by a (possibly empty) palindrome. Basically, there are the two cases below (where ε marks the palindrome midpoint):

Thus a possible solution is the following grammar G_1:

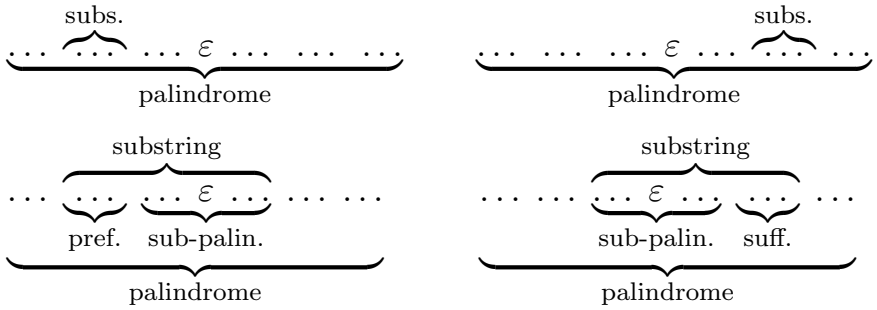
$$G_1 \begin{cases} S & \rightarrow & X\ P \\ X & \rightarrow & a\ X\ \mid\ b\ X\ \mid\ \varepsilon \\ P & \rightarrow & a\ P\ a\ \mid\ b\ P\ b\ \mid\ \varepsilon \end{cases}$$

which works in all cases, since both nonterminals X and P are nullable.

ii. By extending the previous analysis, one can notice that a generic prefix followed by a (possibly empty) palindrome is a generic string (possibly empty), so that actually the language L_1 is the universal language Σ^*. To generate language L_1, it suffices a simplified grammar G_{univ} that only keeps the rules $S \rightarrow X$ and $X \rightarrow a\ X\ \mid\ b\ X\ \mid\ \varepsilon$ of grammar G_1 (or just the second rule with axiom X if a recursive axiom is admitted).

(b) The situation is similar to that of question (a), namely:

i. A (possibly empty) substring of a (non-centred) palindrome is a (possibly empty) palindrome with either a generic (possibly empty) prefix or a generic (possibly empty) suffix, but not with both a prefix and a suffix. Basically, there are the four cases below:



Thus a possible solution is the following grammar G_2:

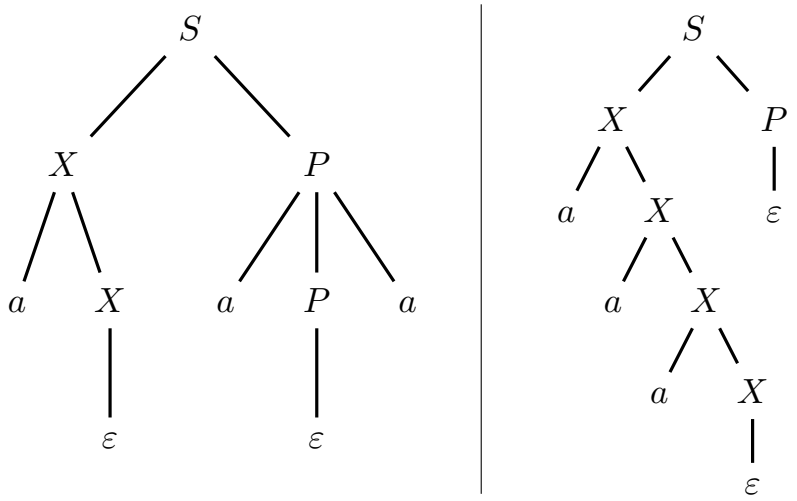
$$G_2 \begin{cases} S & \rightarrow & X\ P\ \mid\ P\ X \\ X & \rightarrow & a\ X\ \mid\ b\ X\ \mid\ \varepsilon \\ P & \rightarrow & a\ P\ a\ \mid\ b\ P\ b\ \mid\ \varepsilon \end{cases}$$

which works in all cases, since both nonterminals X and P are nullable.

ii. Again, a generic (possibly) string is obtained, so that the language L_2 is the universal language Σ^* as well. To generate language L_2, it suffices the same simplified grammar G_{univ} shown before.

(c) Concerning ambiguity however, the solutions discussed above behave differently:

i. Both grammars G_1 and G_2 are ambiguous. In fact string $a\,a\,a \in L_1$, L_2 is ambiguous for both grammars ! The reason is the concatenation ambiguity between the strings generated by nonterminals X and P. Here are the trees:



These trees are generated by both grammars. They are the only trees of grammar G_1 for string $a\,a\,a$, whereas grammar G_2 admits, besides the two trees above, two more trees, which are obtained by using rule $S \to P\ X$ instead of rule $S \to X\ P$.

ii. It is well known that the universal language is non-ambiguous, and that the simplified *BNF* grammar G_{univ} shown before, being of type $LL\,(1)$, is clearly non-ambiguous.

Notice that the shorter string $a\,a$ is ambiguous as well (for both grammars), because both nonterminals X and P are nullable, both are able to generate $a\,a$, and consequently the axiomatic rule $S \to X\ P$ is affected by concatenation ambiguity. This concludes the ambiguity exam for grammars G_1 and G_2.

The reader may be interested in modifying this exercise and let the palindrome have a centre c instead. Try to answer questions (a), (b) and (c) under such a new hypothesis. We encourage the reader to proceed by himself and take inspiration from the previous answers, independently of whether they still hold true or do not.

However, here is a possible solution with grammars G'_1 and G'_2 (axiom S), over the three-letter alphabet $\Sigma = \{\ a,\ b,\ c\ \}$ (letter c is the palindrome centre):

$$G'_1 \begin{cases} S \to X \mid X\,P \\ X \to a\,X \mid b\,X \mid \varepsilon \\ P \to a\,P\,a \mid b\,P\,b \mid c \end{cases} \qquad G'_2 \begin{cases} S \to X \mid X\,P \mid P\,X \\ X \to a\,X \mid b\,X \mid \varepsilon \\ P \to a\,P\,a \mid b\,P\,b \mid c \end{cases}$$

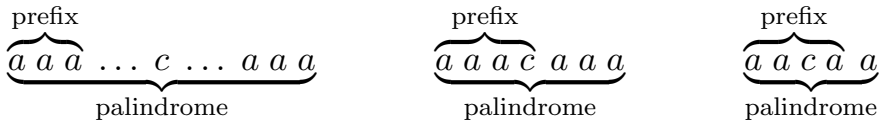
The prefix language of the centred palindromes consists of arbitrary strings over the sub-alphabet $\Sigma' = \{\ a,\ b\ \}$, and of arbitrary strings over Σ' followed by centred palindromes. Grammar G'_1 generates these two string models.

The substring language of the centred palindromes consists of arbitrary strings over Σ', and of centred palindromes either preceded or followed (but not both cases together) by arbitrary strings over Σ'. Grammar G'_2 generates these three string models.

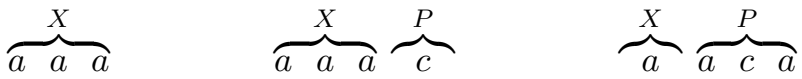
Grammars G'_1 and G'_2 are inspired to grammars G_1 and G_2, respectively: in both G'_1 and G'_2, the palindrome rules generate the centre c and a new axiomatic alternative $S \to X$ is added, to consider the case of a prefix or of a substring that does not contain the centre c; the remaining rules are unchanged.

Concerning ambiguity, first consider grammar G'_1. Observe that the sub-grammars of nonterminals X and P are unambiguous, that the sub-language $L(X\,P)$ generated by the concatenation $X\,P$ is unambiguous (because the condition of concatenation ambiguity is not satisfied by the two sub-languages $L(X)$ and $L(P)$), and that the two sub-languages generated by X and by $X\,P$ are disjoint, i.e., $L(X) \cap L(X\,P) = \emptyset$. Therefore grammar G'_1 is unambiguous, as it is the union of two disjoint sub-languages that are generated unambiguously (see the axiomatic rule of G'_1).

For instance, the ambiguous string $a\,a\,a$ discussed before for the prefix language of the non-centred palindromes, now splits into three cases: $a\,a\,a$, $a\,a\,a\,c$ and $a\,a\,c\,a$. The first is a palindrome truncated before the centre c, the second is truncated immediately after the centre, and the third after the centre but before the end. This way:



None of these three cases is affected by concatenation ambiguity any more. In fact, grammar G'_1 generates each of them in these three unique ways, respectively:



Of course, grammar G'_1 can still generate all the centred palindromes, as it is permitted by the definition of language L_1, since the nonterminal X is nullable and consequently all these derivations are possible: $S \Rightarrow X\,P \Rightarrow P \stackrel{*}{\Rightarrow}$ any centred palindrome.

Then, considering grammar G'_2, the same observations as for grammar G_1 apply, with only one difference: the two sub-languages $L(X\;P)$ and $L(P\;X)$ overlap, i.e., $L(X\,P) \;\cap\; L(P\,X) \neq \emptyset$. In fact, since nonterminal X is nullable, it holds $L(P) \subset L(X\;P)$, $L(P\;X)$, that is, the two sub-languages share all the centred palindromes (and only these). For instance, string c (empty palindrome) is generated by both concatenations $X\,P$ and $P\,X$. Therefore grammar G'_2 is affected by union ambiguity.

Anyway, if one needs or wishes, it is not difficult to make grammar G'_2 unambiguous, with a little more effort: since the sub-language overlap is caused only by the nullability of nonterminal X, it suffices to make nonterminal X non-nullable (of course without changing the language). Here is a possible solution G''_2 (axiom S):

$$G''_2 \left\{ \begin{array}{lcl} S & \to & \varepsilon \mid X \mid P \mid X\,P \mid P\,X \\ X & \to & a\,X \mid b\,X \mid a \mid b \\ P & \to & a\,P\,a \mid b\,P\,b \mid c \end{array} \right.$$

This grammar variant G''_2 is obtained from G'_2 by means of the standard transformation into non-nullable form (see the textbook) and is equivalent to grammar G'_2. For instance, string c (empty palindrome) may not be generated by either concatenation $X\,P$ or $P\,X$, whereas it can be generated only by nonterminal P.

2. Consider a description language that models the organizational chart of a generic institution, which is structured according to the following rules:

   - At the top level, the organization has a name (a non-empty alphanumerical string), a president, a secretary and a staff (a non-empty group of participating persons), all mandatory.
   - The organization also includes a non-empty list of committees. A committee may include zero or more subcommittees, and a subcommittee may further include zero or more subcommittees, at any level of depth.
   - A committee has a name (a non-empty alphanumerical string), a chairperson, a secretary, a staff (a non-empty group of participating persons), a date of establishment and one of closure; all are mandatory, except for the date of closure.
   - A subcommittee has the same structure as a committee, except that the secretary and the date of establishment are optional, and that the date of closure may be present only if a date of establishment is also included.
   - The order of the various elements listed above in the organization, in the committees and in the subcommittees, is fixed and is shown in the example below.
   - A person is described by his first name and his last name (both are alphabetical strings), and by his pid (personal identifier), which is a string of exactly six decimal digits. A date is composed of a day/month/year in the customary format xx/xx/xxxx, where x is a decimal digit.

   Further detailed information about the lexical and syntactic structure of the description can be inferred from the example below:

```
organization CompSciConference2017
   president firstname Lilli lastname Verdi pid 839257;
   secretary firstname Paolo lastname Bianchi pid 325476;
   staff
      firstname ... lastname ... pid ...;
      firstname ... lastname ... pid ...;
   end staff
   committee communication
      chairperson firstname Mario lastname Sarti pid 937294;
      secretary firstname Lisa lastname Volo pid 739473;
      staff ... end staff
      established 23/07/2015;
      subcommittee invitations
         chairperson firstname Sandra lastname Piano pid 938402;
         staff ... end staff
         established 14/08/2015;
         closed 21/01/2017;
      end subcommittee invitations
   end committee communication
   ...
end organization
```

   Write a grammar, *EBNF* and unambiguous, that models the sketched language.

Solution

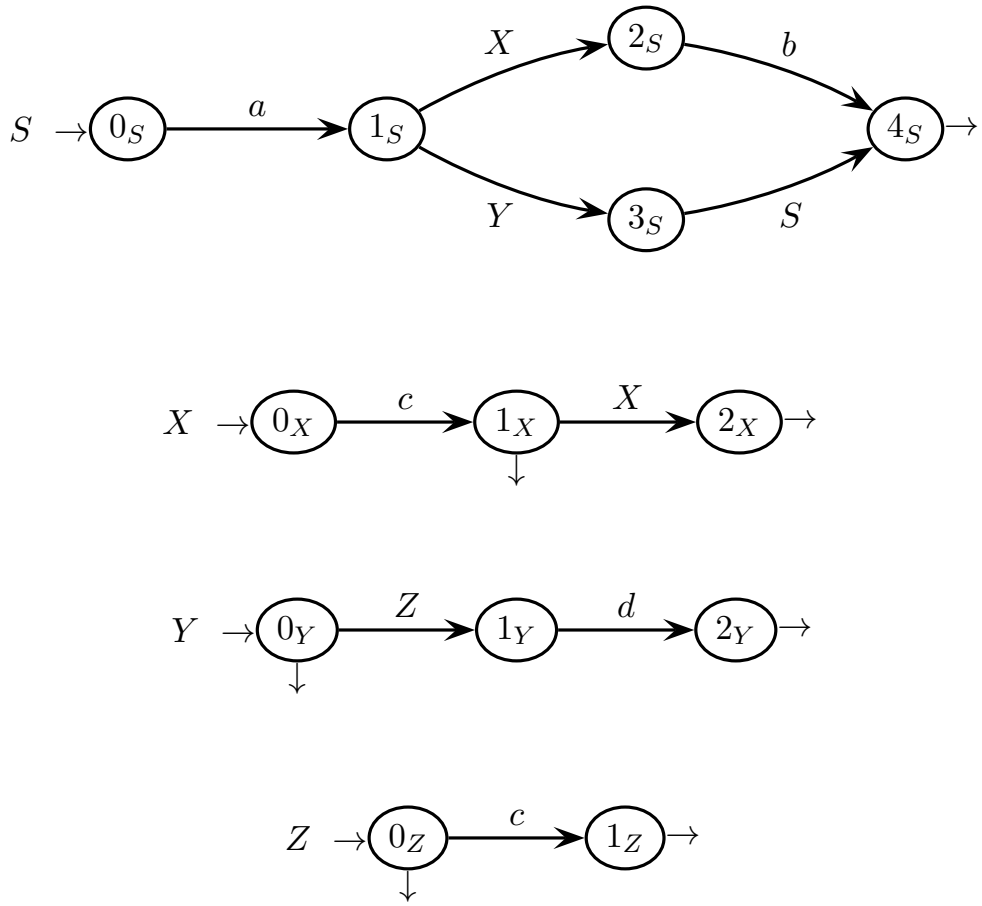
(a) Here is a reasonable grammar that generates the sketched language (axiom ORG):

$$
\left\{
\begin{array}{rcl}
\langle \mathsf{ORG} \rangle & \to & \mathsf{organization}\ \langle \mathsf{STR} \rangle \\
& & \quad \mathsf{president}\ \langle \mathsf{PERS} \rangle\ \text{`;'} \\
& & \quad \mathsf{secretary}\ \langle \mathsf{PERS} \rangle\ \text{`;'} \\
& & \quad \langle \mathsf{STAFF} \rangle \\
& & \quad \langle \mathsf{COMM} \rangle^{+} \\
& & \mathsf{end\ organization} \\
\hline
\langle \mathsf{COMM} \rangle & \to & \mathsf{committee}\ \langle \mathsf{STR} \rangle \\
& & \quad \mathsf{chairperson}\ \langle \mathsf{PERS} \rangle\ \text{`;'} \\
& & \quad \mathsf{secretary}\ \langle \mathsf{PERS} \rangle\ \text{`;'} \\
& & \quad \langle \mathsf{STAFF} \rangle \\
& & \quad \mathsf{established}\ \langle \mathsf{DATE} \rangle\ \text{`;'} \\
& & \quad \big[\, \mathsf{closed}\ \langle \mathsf{DATE} \rangle\ \text{`;'} \big] \\
& & \quad \langle \mathsf{SCOMM} \rangle^{*} \\
& & \mathsf{end\ committee}\ \langle \mathsf{STR} \rangle \\
\hline
\langle \mathsf{SCOMM} \rangle & \to & \mathsf{subcommittee}\ \langle \mathsf{STR} \rangle \\
& & \quad \mathsf{chairperson}\ \langle \mathsf{PERS} \rangle\ \text{`;'} \\
& & \quad \big[\, \mathsf{secretary}\ \langle \mathsf{PERS} \rangle\ \text{`;'} \big] \\
& & \quad \langle \mathsf{STAFF} \rangle \\
& & \quad \Big[\, \mathsf{established}\ \langle \mathsf{DATE} \rangle\ \text{`;'}\ \big[\, \mathsf{closed}\ \langle \mathsf{DATE} \rangle\ \text{`;'} \big] \Big] \\
& & \quad \langle \mathsf{SCOMM} \rangle^{*} \\
& & \mathsf{end\ subcommittee}\ \langle \mathsf{STR} \rangle \\
\hline
\langle \mathsf{STAFF} \rangle & \to & \mathsf{staff}\ \big(\, \langle \mathsf{PERS} \rangle\ \text{`;'} \,\big)^{+}\ \mathsf{end\ staff} \\
\langle \mathsf{PERS} \rangle & \to & \mathsf{firstname}\ \langle \mathsf{NAME} \rangle\ \mathsf{lastname}\ \langle \mathsf{NAME} \rangle\ \mathsf{pid}\ \langle \mathsf{PID} \rangle \\
\hline
\langle \mathsf{STR} \rangle & \to & \big(\, \langle \mathsf{ALPH} \rangle \mid \langle \mathsf{NUM} \rangle \,\big)^{+} \\
\langle \mathsf{NAME} \rangle & \to & \langle \mathsf{ALPH} \rangle^{+} \\
\langle \mathsf{PID} \rangle & \to & \langle \mathsf{NUM} \rangle^{6} \\
\langle \mathsf{DATE} \rangle & \to & \langle \mathsf{NUM} \rangle^{2}\ \text{`/'}\ \langle \mathsf{NUM} \rangle^{2}\ \text{`/'}\ \langle \mathsf{NUM} \rangle^{4} \\
\hline
\langle \mathsf{ALPH} \rangle & \to & [\, A \ldots Z \,] \mid [\, a \ldots z \,] \\
\langle \mathsf{NUM} \rangle & \to & [\, 0 \ldots 9 \,]
\end{array}
\right.
$$

The grammar is *EBNF* and non-ambiguous, as it is a made of non-ambiguous substructures. It is presented in a stratified layout, to isolate the groups of rules that have similar roles and to put in evidence the dependences between the rules. Recursion is used only for nesting subcommittees, whereas all the other syntactic details are modeled by means of regular constructs. Identifiers and numbers are modeled as customary. Minor adjustments are possible, for instance to better model the alphanumerical strings or to differently place the semicolon separator in the rules, etc. Other solutions are also possible, more or less different.

3 Syntax Analysis and Parsing Methodologies 20%

1. Consider the following grammar G, represented as a machine net over the four-letter terminal alphabet $\Sigma = \{ a, b, c, d \}$ and the four-letter nonterminal alphabet $V = \{ S, X, Y, Z \}$ (axiom S):



Answer the following questions:

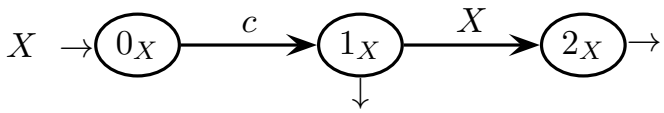
(a) Draw the complete pilot of grammar G, say if grammar G is of type $ELR(1)$, and shortly justify your answer. If grammar G is not $ELR(1)$ then highlight all the conflicts in the pilot.

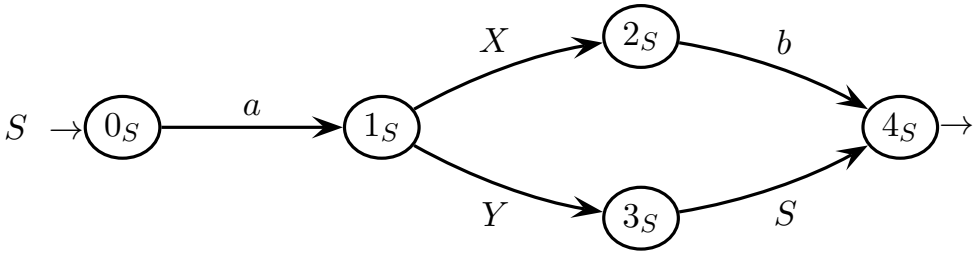
(b) Write the necessary guide sets on the arcs of the machine net, show that grammar G is not of type $ELL(1)$, based on the guide sets, and shortly justify your answer (for the guide sets please use the figure prepared on the next page).

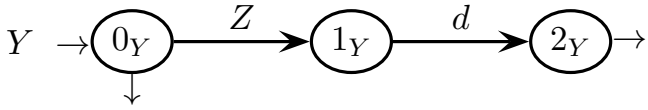
(c) Say if grammar G is of type $ELL(k)$ for some $k > 1$, and provide an adequate justification to you answer.

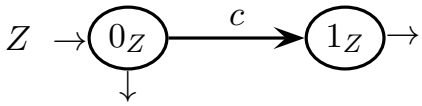
(d) (optional) Identify the four languages $L(Z)$, $L(Y)$, $L(X)$ and $L(S)$ by resorting, for representing each of them, to a formalism as simple as possible. Determine if there is a grammar G' equivalent to grammar G that is of type $ELL(1)$, and adequately justify your answer.

please here draw the call arcs and write the guide sets with $k = 1$



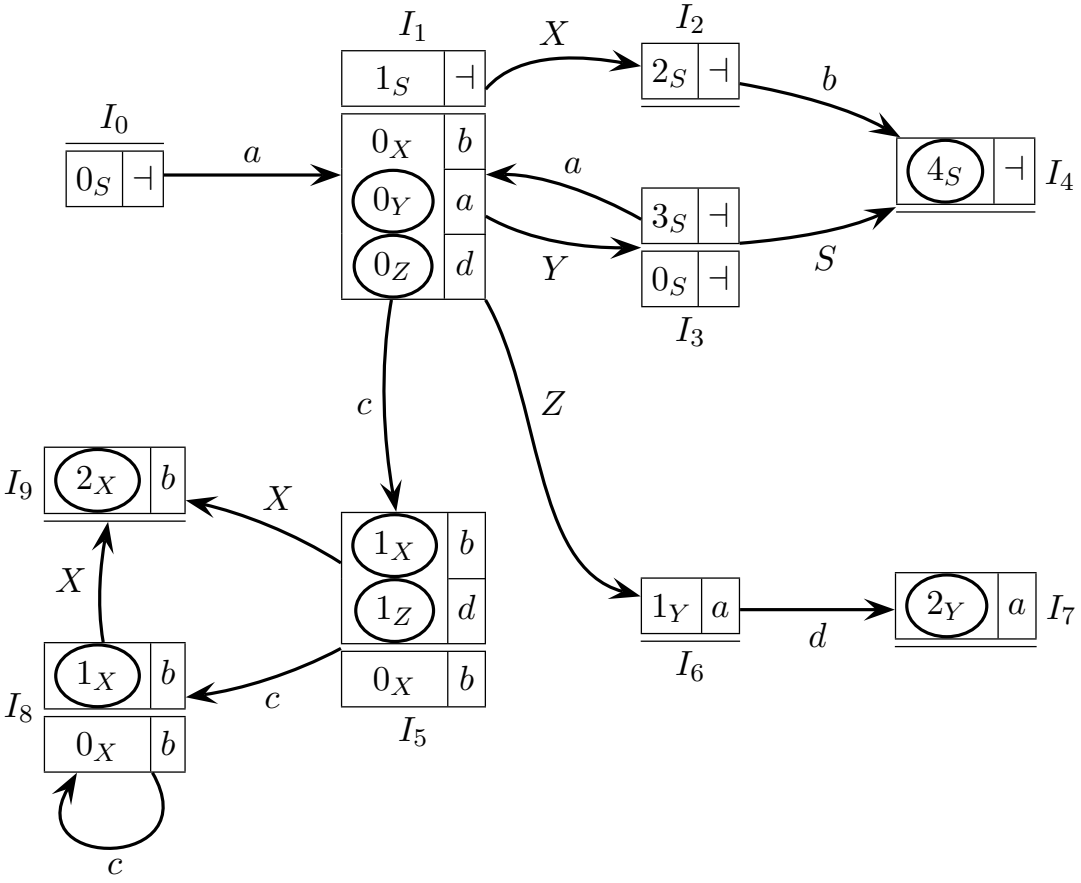






Solution

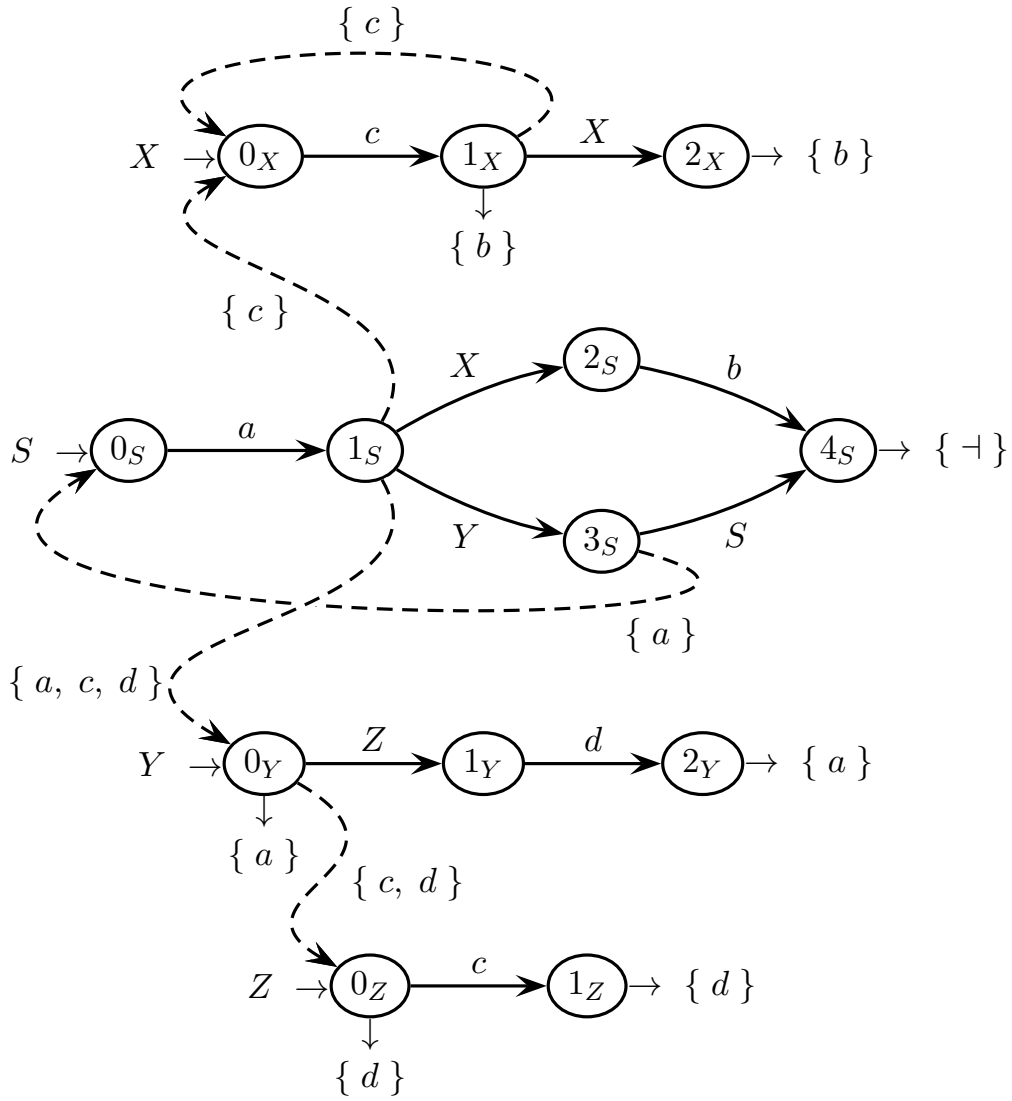
(a) Here is the pilot of grammar G, with ten m-states (I_0 is the initial m-state):



The pilot contains a few reduction items, and each of the m-states I_1 and I_5 contains two reduction items, yet there are not any shift-reduce or reduce-reduce conflicts. Furthermore, the only multiple (precisely double) transition is $I_1 \xrightarrow{c} I_5$, yet it is not convergent, so there are not any convergence conflicts either. Thus the pilot is conflict-free and consequently grammar G is of type $ELR\,(1)$.

Notice that the pilot does not have the *STP*, since transition is $I_1 \xrightarrow{c} I_5$ is multiple or, equivalently, the base of m-state I_5 contains two items. This is the only aspect of the pilot that causes it to lose the *STP*. Of course, failing to have the *STP* immediately makes grammar G fail to be of type $ELL\,(1)$. Anyway, the $ELL\,(1)$ condition does not give more information, and for better understanding why and where the $ELL\,(1)$ property fails in the grammar G, see point (b).

(b) Here are the guide sets of grammar G, with $k = 1$ (those on the terminal shift arcs are understood):



Computing the guide sets is not difficult. Only notice that the guide set on the call arc $1_S \overset{\{\, a,\, c,\, d \,\}}{- - - \rightarrow} 0_Y$ contains terminal a because nonterminal Y is nullable and terminal a is an initial of nonterminal S, which follows nonterminal Y. The reason is *not* that terminal a is contained in the guide set of the exit arrow $0_Y \rightarrow$. A similar argument applies to the call arc $0_Y \overset{\{\, c,\, d \,\}}{- - \rightarrow} 0_Z$ as for terminal d.
On the bifurcation state 1_S, the guide sets of the two call arcs are not disjoint, thus grammar G is not of type $ELL\,(1)$. All the other guide sets on bifurcation states, namely on states 1_X, 0_Y and 0_Z, are disjoint.

(c) The only conflicting state is 1_S, and the only overlapping element is letter c. Moving to $k = 2$ and considering only such crucial element, the guide set on the call arc $1_S \dashrightarrow 0_X$ becomes $\{\, c\,b,\ c\,c \,\}$, whereas that on $1_S \dashrightarrow 0_Y$ becomes $\{\, a,\ c\,d,\ d \,\}$; the other elements a and d do not require to be extended to $k = 2$. Thus now the two guide sets are disjoint, therefore grammar G is of type $ELL\,(2)$.

(d) It turns out that all the four languages are finite or at worst regular. In fact, the following language representations hold (using the optionality operator [ ]):

$$L(Z) = \varepsilon \mid c = [\,c\,]$$

due to direct evidence. Then:

$$L(Y) = \varepsilon \mid L(Z)\,d = \varepsilon \mid (\,\varepsilon \mid c\,)\,d = \big[\,[\,c\,]\,d\,\big]$$

due to substitution. Then:

$$L(X) = c \mid c\,L(X) = c^{+}$$

due to the Arden identity. Then:

$$L(S) = a\,L(X)\,b \mid a\,L(Y)\,L(S) = \big(\,a\,L(Y)\,\big)^{*}\,a\,L(X)\,b$$

due to the Arden identity again. Eventually:

$$L(S) = \Big(\,a\,\big(\,\underbrace{\varepsilon \mid (\,\varepsilon \mid c\,)\,d}_{L(Y)}\,\big)\Big)^{*}\,a\,\underbrace{c^{+}}_{L(X)}\,b$$

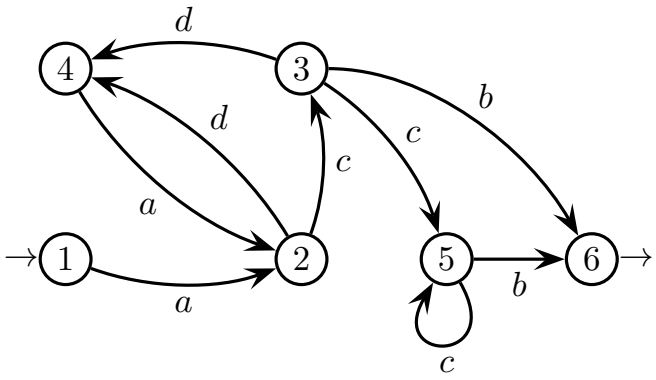
due to substitution again. This can be written in a slightly more compact form:

$$L(S) = \Big(\,a\,\big[\,[\,c\,]\,d\,\big]\,\Big)^{*}\,a\,c^{+}\,b$$

that also uses the optionality operator [ ]. In all the four cases the simplest formalism to represent each language is a regular expression. Actually languages $L(Z)$ and $L(Y)$ are even finite, and we could just list their strings.

Since by definition $L(G) = L(S)$, grammar G generates a regular language. Consequently, it is well known that there is a finite-state deterministic automaton equivalent to G, and that the right-unilinear grammar associated to such an automaton is of type $ELL(1)$, precisely due to determinism. Thus language $L(G)$ is of type $ELL(1)$, and a viable $ELL(1)$ (actually $LL(1)$) grammar for $L(G)$ is this (*BNF*) right-unilinear one (the reader may wish to find it for practice).

For instance, the minimal deterministic automaton equivalent to grammar G is the following (it can be obtained by means of the Berry-Sethi method):



$$\begin{cases} 1 \;\to\; a\,2 \\ 2 \;\to\; c\,3 \\ 2 \;\to\; d\,4 \\ \ldots \;\to\; \ldots \\ 6 \;\to\; \varepsilon \end{cases}$$

On the right there is (part of) the corresponding right-unilinear *BNF* grammar (axiom 1), which is of type $LL(1)$ (the reader may complete it by himself).

4 Language Translation and Semantic Analysis 20%

1. The grammar G below (axiom S) generates bit strings that represent positive natural numbers according to the usual positional encoding:

$$G \begin{cases} S & \rightarrow & S\,0 \mid S\,1 \mid N\,1 \\ N & \rightarrow & N\,0 \mid N\,1 \mid \varepsilon \end{cases}$$

   For instance, string $0\,1\,0\,1\,0\,0$ represents the (decimal) number 20, string $1\,0\,1\,0$ represents number 10 and string $0\,1\,1$ number 3.

   Please answer the following questions:

   (a) Define a translation scheme that concatenates to the source string a prefix consisting of a number of symbols 'e' equal to the exponent of the largest power of 2 of which the encoded number is a multiple.

   For instance, the string $0\,1\,0\,1\,0\,0$ encodes number 20, which is a multiple of 4, thus it is translated into string $e\,e\,0\,1\,0\,1\,0\,0$. Other examples are $1\,0\,1\,0 \mapsto e\,1\,0\,1\,0$ and $0\,1\,1 \mapsto 0\,1\,1$. If necessary, modify the source grammar as little as possible.

   Can this translation be modeled by means of a rational (regular) translation scheme ? In the negative case, please justify your answer; in the positive case, provide such a regular translation scheme.

   (b) Define a translation scheme that cancels all the leading zeroes from the source string. For instance: $0\,1\,0\,1\,0\,0 \mapsto 1\,0\,1\,0\,0$, $1\,0\,1\,0 \mapsto 1\,0\,1\,0$ and $0\,1\,1 \mapsto 1\,1$. If necessary, modify the source grammar as little as possible.

   Can this translation be modeled by means of a rational (regular) translation scheme ? In the negative case, please justify your answer; in the positive case, provide such a regular translation scheme.

   (c) Is the translation of the previous point (b) surjective ? Is it injective ? To answer these questions, consider the source and destination languages as domain and image of the translation function, respectively. Provide an adequate justification to your answer.

Solution

(a) The translation counts how many trailing zeroes the source string has, and it prepends as many characters e to the destination string. We have to separate the generation of the trailing zeroes from the non-trailing ones, thus grammar G has to be modified. Here is a modified source grammar G'_s (axiom S):

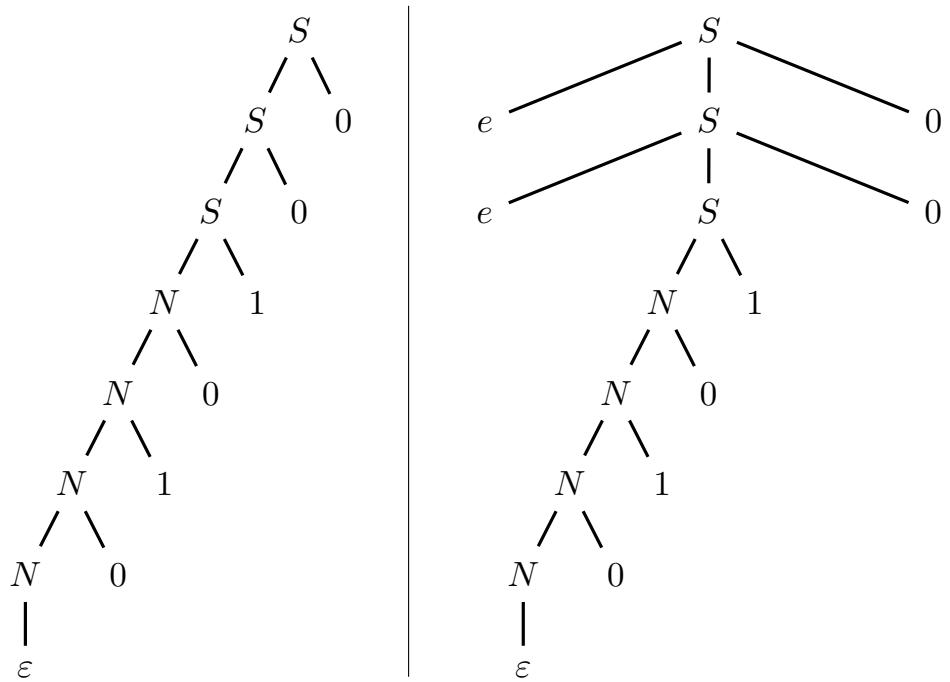
$$G'_s \begin{cases} S & \rightarrow & S\ 0 \ \mid\ N\ 1 \\ N & \rightarrow & N\ 0 \ \mid\ N\ 1 \ \mid\ \varepsilon \end{cases}$$

where the rule $S \rightarrow S\ 1$ has been removed to let nonterminal S generate all and only the trailing zeroes. Here is the corresponding destination grammar G'_d:

$$G'_d \begin{cases} S & \rightarrow & e\ S\ 0 \ \mid\ N\ 1 \\ N & \rightarrow & N\ 0 \ \mid\ N\ 1 \ \mid\ \varepsilon \end{cases}$$

which prepends a character e per each trailing zero.

The source and destination syntax trees for the longest example before should convince anybody of the correctness of the translation scheme G':



The translation cannot be modeled by means of a regular translation scheme, because it requires to ***count*** the number of trailing zeroes. Said differently, the destination language is $e^n\ (\ 0\ \mid\ 1\ )^*\ 1\ 0^n$, with $n \geq 0$, and it is not regular.

Incidentally, notice that the original source grammar G is ambiguous, as a string of bits 1 can be generated by repeatedly using either rule $S \rightarrow S\ 1$ or rule $N \rightarrow N\ 1$, or a mix thereof. Instead, the modified source grammar G'_s is not ambiguous any longer, because rule $S \rightarrow S\ 1$ has been removed.

(b) For this second translation, we can take as basis the scheme G', which already generates the trailing zeroes only by nonterminal S, and just switch the trailing zeroes into leading zeroes. For this purpose, it suffices to turn the alternative rules that expand nonterminal S from the left-linear form into the right-linear form (those of N can stay). Here is a modified source grammar G''_s (axiom S):

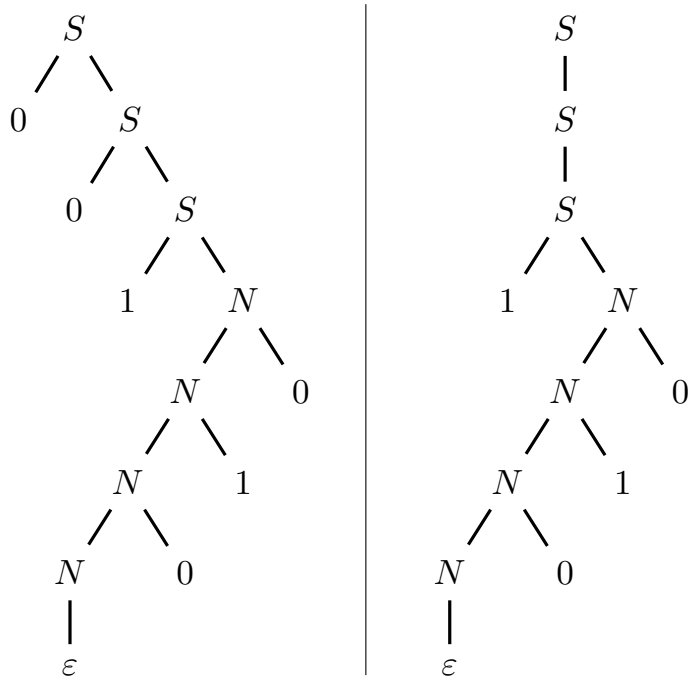
$$G''_s \begin{cases} S & \to & 0\ S \ \mid\ 1\ N \\ N & \to & N\ 0 \ \mid\ N\ 1 \ \mid\ \varepsilon \end{cases}$$

where nonterminal S generates all and only the leading zeroes plus the leftmost 1 digit. Here is the corresponding destination grammar G''_d:

$$G''_d \begin{cases} S & \to & S \ \mid\ 1\ N \\ N & \to & N\ 0 \ \mid\ N\ 1 \ \mid\ \varepsilon \end{cases}$$

which just cancels the leading zeroes.

Again, the source and destination syntax trees for the longest example before should convince anybody of the correctness of the translation scheme G'':



The source grammar G''_s is not ambiguous either. It is evident that also the alternative rules that expand nonterminal N could be turned into right-linear form. This however would change the grammar more than necessary.

The translation can be modeled by the following regular translation scheme:

$$\left(\begin{smallmatrix} 0 \\ \varepsilon \end{smallmatrix}\right)^*\ \begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\ \left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\ \mid\ \begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)^*$$

and the destination language is $1\ (\,0\ \mid\ 1\,)^*$, regular as expected.

(c) First (as suggested) we had better make explicit the source and destination languages L_s and L_d of translation (b), since the surjective and injective properties depend on them. These languages, both regular as said at point (b), can be quickly obtained from the regular translation scheme of point (b):

$$L_s = 0^*\ 1\ (\ 0\ \mid\ 1\ )^* \qquad\qquad L_d = 1\ (\ 0\ \mid\ 1\ )^*$$

Both languages L_s and L_d are a proper subset of the universal language, i.e., $L_s,\ L_d \subsetneq (\ 0\ \mid\ 1\ )^*$, and language L_d is a proper subset of L_s, i.e., $L_d \subsetneq L_s$.

Translation (b) is surely surjective: each destination bit string has at least one (actually more than one) source bit strings as its counterimages; more precisely, it has all the source strings equal to itself plus a (possibly empty) prefix composed only of zeroes. In symbols: each destination string $1\ (\ 0\ \mid\ 1\ )^* = s$ is the image of all the (infinitely many) source strings $0^*\ s$.

For the same reason, translation (b) is certainly not injective: all the source bit strings that differ only by a prefix composed of zeroes, are translated into the same destination bit string. In symbols: all the (infinitely many) source strings $0^*\ \underbrace{1\ (\ 0\ \mid\ 1\ )^*}_{s} = 0^*\ s$ are translated into the same destination string s.

In summary, the translation of point (b) is surjective but not injective, and consequently, it is not bijective (one-to-one) either.

For completeness, here are the two translations (a) and (b) written in fractional form:

$$G' \begin{cases} S & \rightarrow & \frac{\varepsilon}{e}\ S\ \frac{0}{0}\ \mid\ N\ \frac{1}{1} \\ N & \rightarrow & N\ \frac{0}{0}\ \mid\ N\ \frac{1}{1}\ \mid\ \frac{\varepsilon}{\varepsilon} \end{cases}$$

$$G'' \begin{cases} S & \rightarrow & \frac{0}{\varepsilon}\ S\ \mid\ \frac{1}{1}\ N \\ N & \rightarrow & N\ \frac{0}{0}\ \mid\ N\ \frac{1}{1}\ \mid\ \frac{\varepsilon}{\varepsilon} \end{cases}$$

These forms are just a notational change of the above representations, of course.

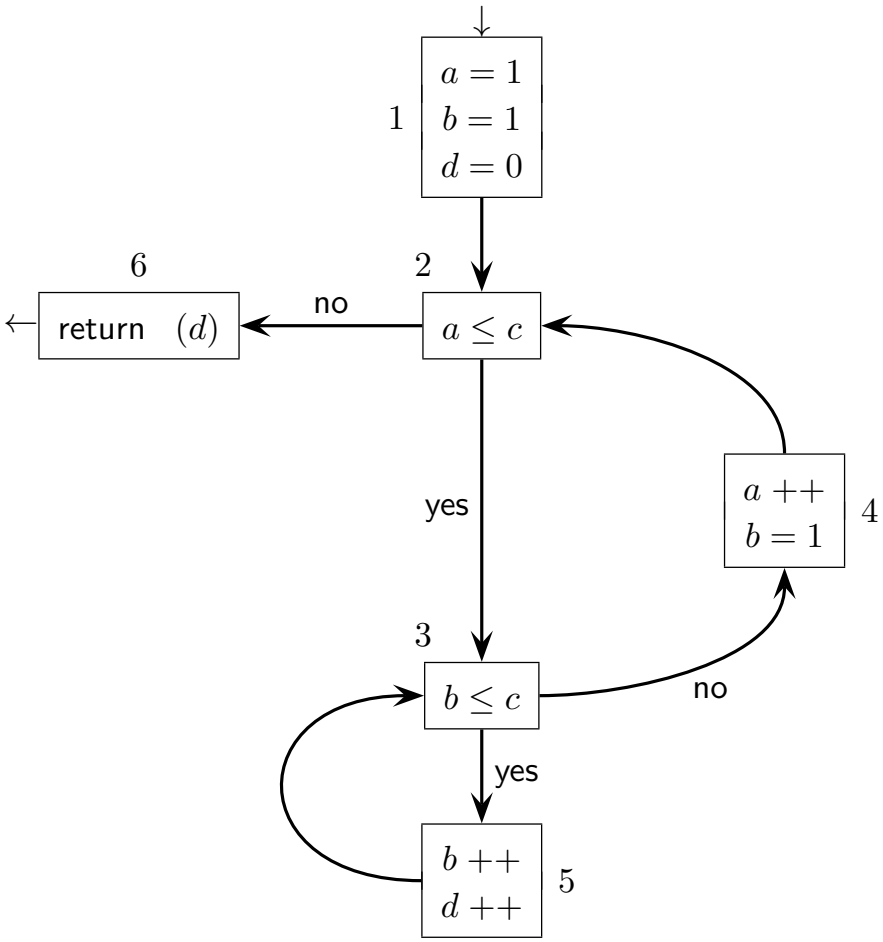
Similarly, the two translation grammars could be rewritten in the equivalent brace notation, as follows:

$$G' \begin{cases} S & \rightarrow & \{\ e\ \}\ S\ 0\ \{\ 0\ \}\ \mid\ N\ 1\ \{\ 1\ \} \\ N & \rightarrow & N\ 0\ \{\ 0\ \}\ \mid\ N\ 1\ \{\ 1\ \}\ \mid\ \varepsilon \end{cases}$$

$$G'' \begin{cases} S & \rightarrow & 0\ S\ \mid\ 1\ \{\ 1\ \}\ N \\ N & \rightarrow & N\ 0\ \{\ 0\ \}\ \mid\ N\ 1\ \{\ 1\ \}\ \mid\ \varepsilon \end{cases}$$

Notice that when the destination grammar contains the empty string $\{\ \varepsilon\ \}$, for simplicity we drop it, e.g., writing $\frac{0}{\varepsilon} = 0\ \{\ \varepsilon\ \}$ and $\frac{\varepsilon}{\varepsilon} = \varepsilon\ \{\ \varepsilon\ \}$ is shortened to 0 and ε, respectively. Similarly, instead of writing $\frac{\varepsilon}{e} = \varepsilon\ \{\ e\ \}$, we shorten to $\{\ e\ \}$.

2. Consider the following Control Flow Graph (*CFG*) of a program, with some nodes that contain multiple assignments (input node 1 and output node 6):



This program has three variables a, b and d, and an input parameter c. All of them are integers ≥ 0.

Answer the following questions:

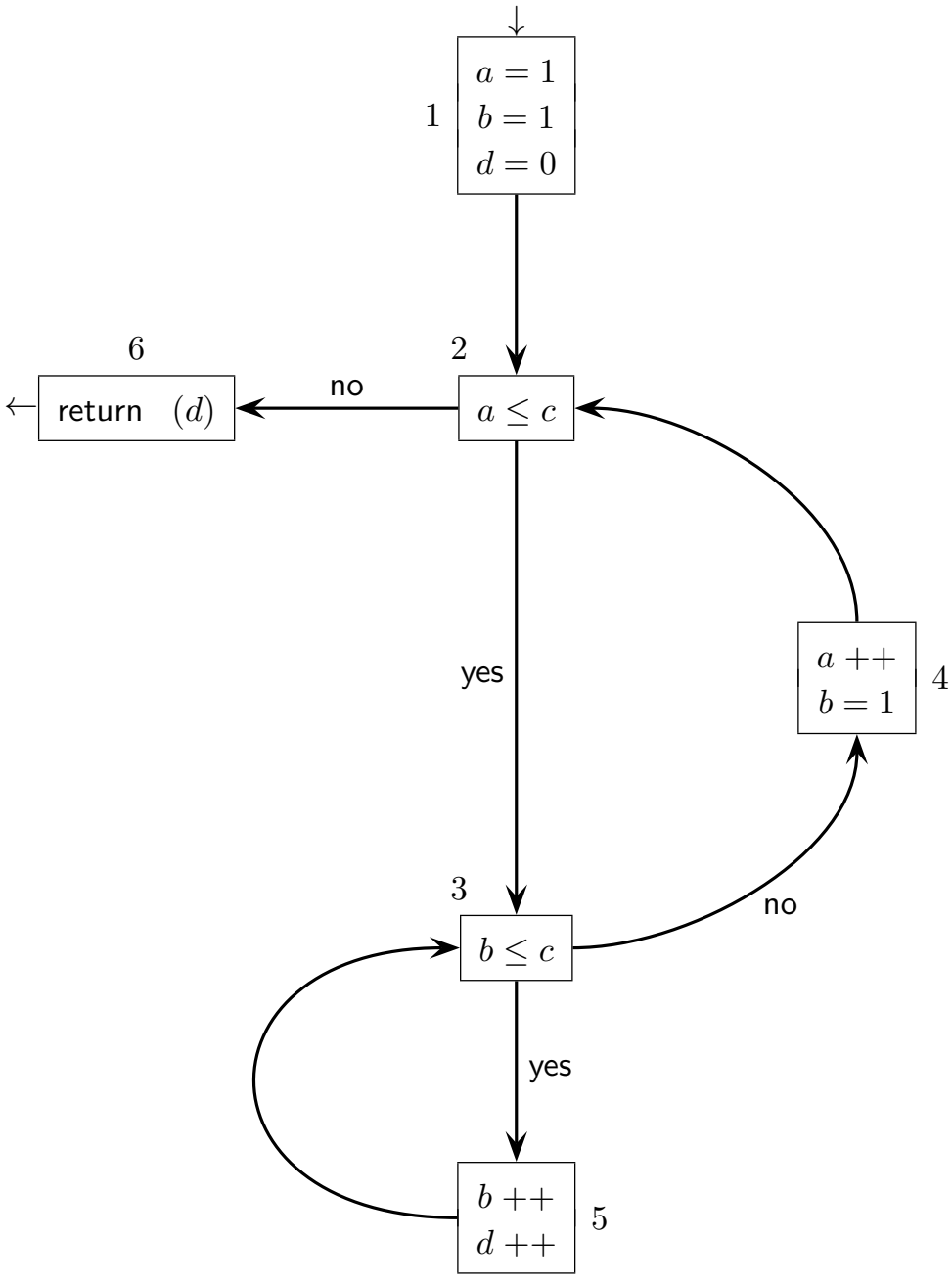
(a) Find a regular expression R, over the node name alphabet, that describes the execution traces of the program, ignoring any semantic restriction.

(b) Directly identify the variables live at the node inputs of the *CFG* and write them on the *CFG* (use the figure prepared on the next page).

(c) Write the system of flow equations for the live variables of the *CFG*, and iteratively solve the system (use the tables prepared on the next pages). Verify that the obtained solution coincides with that found at point (b).

(d) (optional) Determine the value d returned by the program as a function of the input parameter c. Based on this function, refine the regular expression R found at point (a) into a language expression over the node name alphabet that exactly defines the program execution traces. Is the language of the execution traces still regular ? Adequately justify your answer.

please write here the variables live at the node inputs



tables of definitions and usages at the nodes

node	*defined*
1	
2	
3	
4	
5	
6	

node	*used*
1	
2	
3	
4	
5	
6	

system of data-flow equations

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

iterative solution table of the system of data-flow equations
(the number of columns is not significant and is more than sufficient to solve the system)

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

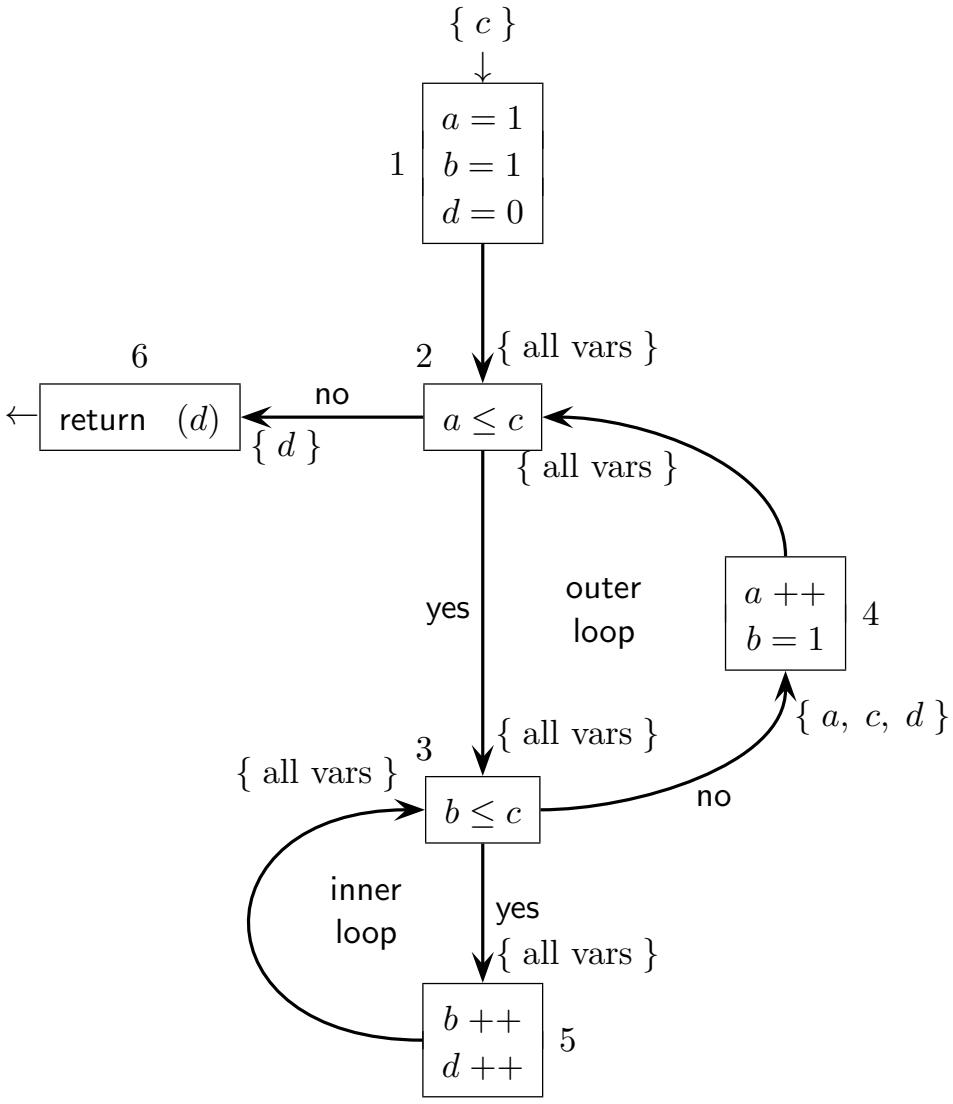
Solution

(a) Here is a regular expression R, obtained intuitively, for the program traces:

$$R = 1\ 2\ \big(\ 3\ (\ 5\ 3\ )^*\ 4\ 2\ \big)^*\ 6$$

The *CFG* consists of two nested loops, namely the inner loop (5, 3) and the outer loop (3, (inner loop), 4, 2), with some prologue and epilogue, so finding the regular expression R is simple. One could quickly obtain a regular expression R by node elimination as well, the same as above or an equivalent one.

(b) Here are the live variables of the *CFG*, obtained directly from their definition:



All the variables are live on most nodes. Variable b is not live on node 4, as it is redefined yet unused therein. Only variable d is live on the final node 6, as it is the only one used therein. The input parameter c is live on the input of the initial node. Notice that the auto-increment operator ++ causes the variable to be both used and defined, since e.g. a ++ is equivalent to $a = a + 1$.

(c) Here is the flow equation method:

tables of definitions and usages at the nodes

node	*defined*
1	$a\ b\ d$
2	–
3	–
4	$a\ b$
5	$b\ d$
6	–

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

system of data-flow equations

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

Some flow equations can be more or less simplified, namely:

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

Notice that c is an input parameter and is used in the program, thus it is the only variable that can be live at the input of the initial node.

iterative solution table of the system of data-flow equations using the non-simplified version
(the number of columns is not significant)

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

The solution converges in two steps only. Column *in* at step 2 provides the live variables at the node inputs.
The solution is identical to the direct one.

(d) The input-output relation of this program is $d = c^2$ (i.e., the square of c), as:

$$d = \overset{\text{initial value}}{0} + \underbrace{\overbrace{\underbrace{1}_{b=1} + \underbrace{1}_{b=2} + \ldots + \underbrace{1}_{b=c}}^{a=1}}_{c \text{ times}} + \overbrace{\text{idem}}^{a=2} + \ldots + \overbrace{\text{idem}}^{a=c} = c \times c = c^2$$

c times

Said synthetically, the program consists of two nested independent loops, each of which iterates exactly c times, thus the total number of increments d ++ of the variable d is the product of the numbers of iterations, i.e., $c \times c = c^2$.

A language formulation that keeps semantics into consideration is a sort of refinement R' of the regular expression R, namely:

$$R' = 1\ 2\ \big(\,3\ (\,5\ 3\,)^h\ 4\ 2\,\big)^h\ 6$$

where $h = c \geq 0$, the input parameter. Said differently, one can see that semantics imposes to derive the two star operators '$*$' (both the inner and the outer one) by choosing for both an equal exponent $h = c$.

Clearly the formulation R' is not a regular expression. The refined language of the execution traces of the *CFG* is not regular either, as it has two equal exponents that cannot be granted by two independent iteration operators.