FORMAL LANGUAGES AND COMPILERS

prof Angelo Morzenti

Exam of Thu 21 September 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 two-letter alphabet $\Sigma = \{\ a,\ b\ \}$ and answer the following questions:

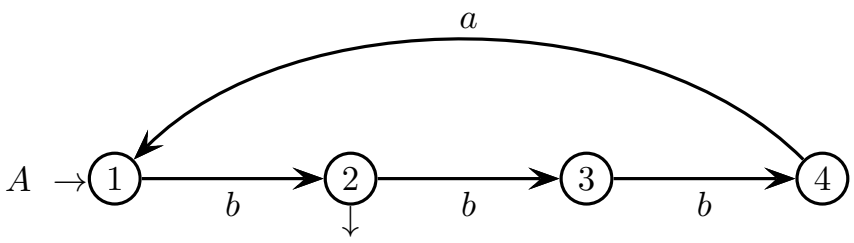
   (a) By using the Berry-Sethi method, design a deterministic finite-state automaton A_e that accepts the language generated by the regular expression e below:

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

   (b) Check if the automaton A_e designed at point (a) is minimal, and minimize it if necessary.

   (c) Is the language $L = L\,(e)$ local? Provide a justification for your answer.

   (d) Consider the finite-state automaton A below, over the two-letter alphabet $\{\ a,\ b\ \}$:



   By using a systematic method, derive an automaton $A_\cap$ that accepts the intersection language $L_\cap = L\,(e)\ \cap\ L\,(A)$.

   (e) (optional) Identify the regular expression $e_\cap$ equivalent to the automaton $A_\cap$ derived at point (d), and provide a suitable justification for your answer.

Solution

(a) Deterministic finite-state automaton A_e:

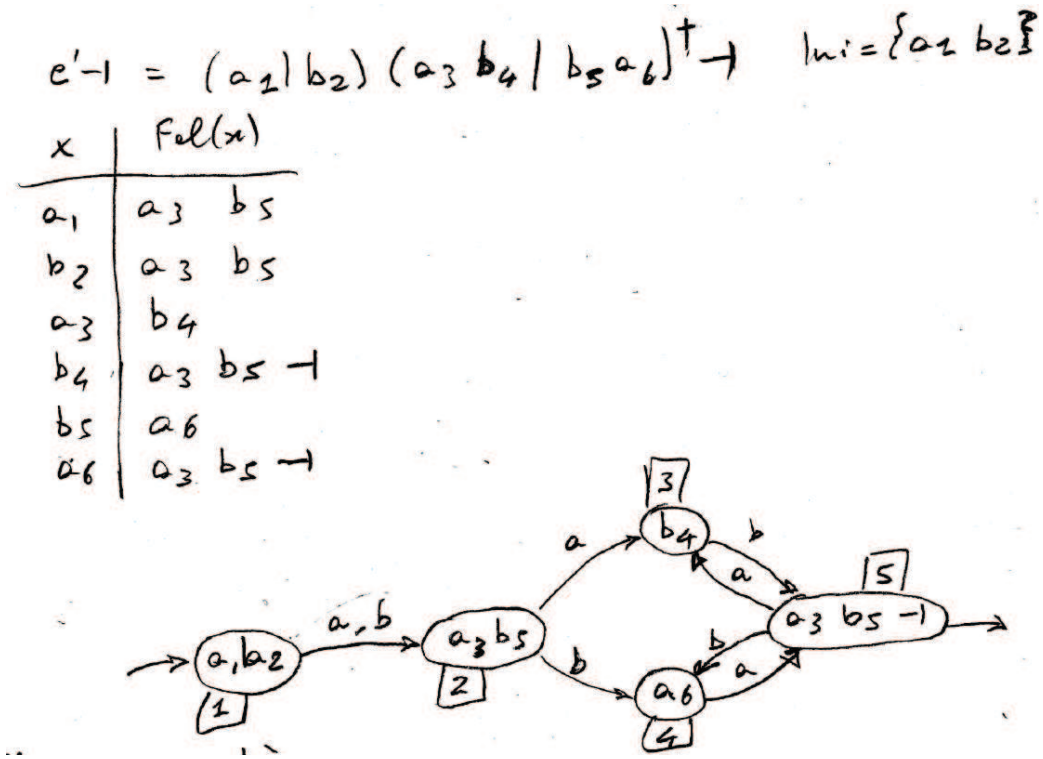


Figure 1: Berry-Sethi construction

(b) The automaton is indeed minimal. Final and non-final states are distinguishable. State pairs $(2-3)$, $(2-4)$, $(1-3)$, $(1-4)$, $(5-3)$, $(5-4)$, and $(3-4)$ are distinguishable because the sets of labels of outgoing arcs are different. States $(1-2)$ are distinguishable because so are $(2-3)$ and $(2-4)$.

(c) The language $L = L(e)$ is not local. Its local sets are $Ini(L) = \{a, b\}$, $Fin(L) = \{a, b\}$, $Dig(L) = \{aa, ab, ba, bb\}$, and for string $x = bbb$ it holds $x \neq \varepsilon$, $x \notin L(e)$, $Ini(x) \in Ini(L)$, $Fin(x) \in Fin(L)$, and $Dig(x) \subseteq Dig(L)$.

(d) See Figure 2b.

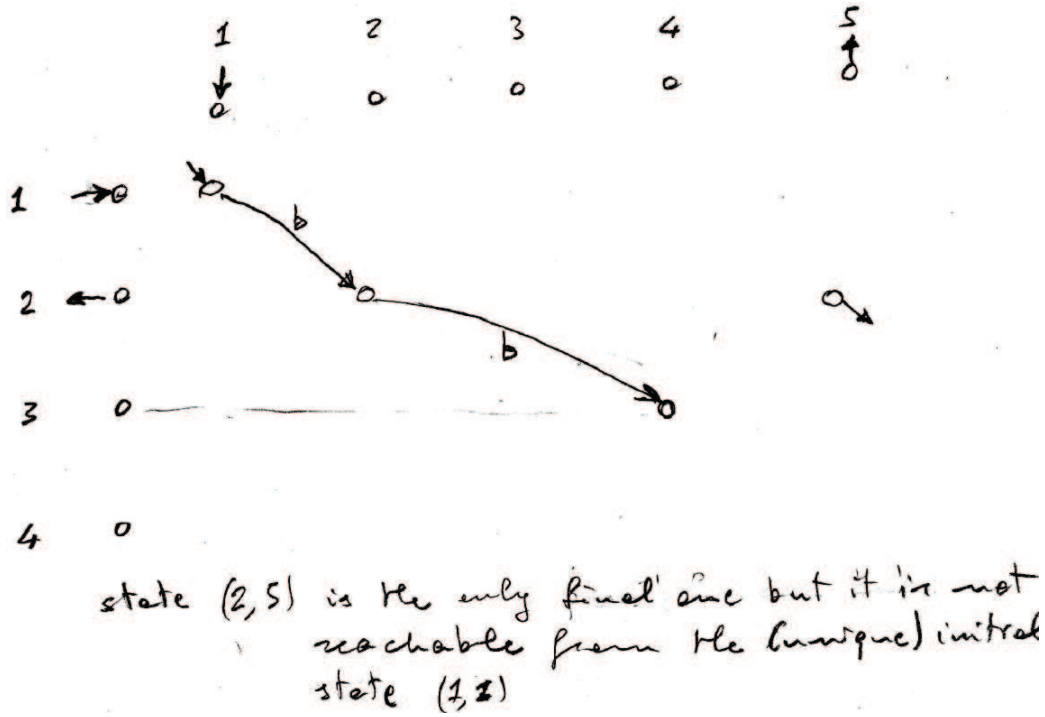


Figure 2: Product automaton

(e) $e_\cap = \emptyset$ because $L(A_\cap) = \emptyset$.

2 Free Grammars and Pushdown Automata 20%

1. Consider the following context-free language L, over the two-letter terminal alphabet $\{\ a,\ b\ \}$:

$$L = \left\{\ a^h\ b^k \mid \quad h, k \geq 0 \ \text{ and } \ k \neq 2h\ \right\}$$

Answer the following questions:

(a) Write a *BNF* grammar G that generates the language L, no matter if ambiguous, and draw the syntax trees for the four valid strings: a, b, $a\,b$ and $a\,b\,b\,b$.

(b) The language $\overline{L} \subseteq \Sigma^*$, complement of language L, is also context-free: write a *BNF* grammar $\overline{G}$ that generates it.

(c) (optional) Argue if the complement grammar $\overline{G}$ found at point (b) is ambiguous or not, and in the case it is, write a non-ambiguous equivalent grammar $\overline{G}'$, of type *BNF* or also *EBNF*.

Solution

(a) A simple way to obtain a grammar G for the language L consists of splitting L in the following union ($h, k \geq 0$):

$$L = \left\{ a^h\ b^k \mid \quad k > 2h \right\} \cup \left\{ a^h\ b^k \mid \quad k < 2h \right\}$$

The two sub-languages are similar to the well known languages $a^h\ b^k$ with $k > h$ or with $k < h$, and grammars are easy to obtain for them. Thus (axiom S):

$$G \begin{cases} S & \rightarrow & S_1 \mid S_2 \\ S_1 & \rightarrow & a\ S_1\ b\ b \mid B \\ B & \rightarrow & B\ b \mid b \\ S_2 & \rightarrow & a\ S_2\ b\ b \mid A\ b \mid A \\ A & \rightarrow & a\ A \mid a \end{cases}$$

Notice the presence of the rule $S_2 \rightarrow A\ b$ that accounts for an odd number of letters b. Valid strings have simple trees (to be drawn). Since the union is disjoint, grammar G is not ambiguous and the trees are unique.

(b) A way to formulate the complement language $\overline{L}$ is the following:

$$\overline{L} = \left\{ a^h\ b^k \mid \quad h, k \geq 0 \ \text{ and } \ k = 2h \right\} \cup \Sigma^*\ b\ a\ \Sigma^*$$

as the (disjoint) union of all the strings of type $a^*\ b^*$ that do not belong to language L and of all the strings with a letter b that immediately precedes a letter a (or equivalently of all the strings that contain a digram $b\,a$).
More compactly:

$$\overline{L} = \left\{ a^h\ b^{2h} \mid \quad h \geq 0 \right\} \cup \Sigma^*\ b\ a\ \Sigma^*$$

Here is a grammar $\overline{G}$ (axiom S):

$$\overline{G} \begin{cases} S & \rightarrow & S_1 \mid S_2 \\ S_1 & \rightarrow & a\ S_1\ b\ b \mid \varepsilon \\ S_2 & \rightarrow & X\ b\ a\ X \\ X & \rightarrow & a\ X \mid b\ X \mid \varepsilon \end{cases}$$

(c) Despite the union above is disjoint, grammar $\overline{G}$ is ambiguous as nonterminal S_2 generates its (regular) sub-language ambiguously, e.g., string $b\,a\,b\,a$ is ambiguous. Of course, it could be disambiguated, for instance as follows (axiom S):

$$\overline{G}' \begin{cases} S & \rightarrow & S_1 \mid S_2 \\ S_1 & \rightarrow & a\ S_1\ b\ b \mid \varepsilon \\ S_2 & \rightarrow & a^*\ b^+\ a\ \Sigma^* \end{cases}$$

of type *EBNF*, for simplicity. For grammar $\overline{G}'$ the string $b\,a\,b\,a$ is not ambiguous, as the regular expression $a^*\ b^+\ a\ \Sigma^*$ is not ambiguous.

2. An app for the management of personal fitness maintains data concerning itineraries for hikes. Such itineraries are composed according to the following rules:

   - An itinerary has a name (a string of letters and digits starting with a letter), and an estimated duration expressed in hours and minutes by using integer constants, with the keywords **estTime**, **hrs** and **min** as shown in the example below.
   - An itinerary includes a number ≥ 3 of elements, each of them being either a point or a subitinerary; the first and the last elements of an itinerary may not be a subitinerary (they may only be a point); each element is separated from the next by a semicolon " ; " and the last is terminated by a semicolon as well.
   - A point description consists of a name, an optional integer value denoting the *altitude*, and two coordinates: the *latitude* and the *longitude*, which are numbers with a non-empty integer part and a fractional part.
   - A subitinerary description includes a name and at least two elements; each element can be a point or a subitinerary, but without any other restriction on their type and order.

   A phrase of the language consists of one itinerary description according to the above outlined rules.

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

```
itinerary mountainHike94 estTime 2 hrs 43 min includes

   point NoraVillage altitude 1024 latitude 40.446 longitude 70.982;

   point MidWood latitude 40.485 longitude 70.963;

   subitinerary LeftValley includes

      point WatsonCreek latitude 40.513 longitude 71.026;

      point WatsonSpring latitude 40.529 longitude 71.053;

   end subitinerary;

   point JacksonShelter altitude 2069 latitude 40.537 longitude 71.032;

end itinerary mountainHike94
```

   Write an *EBNF* grammar, not ambiguous, that models the description language outlined above. Draw the syntax tree of the sample phrase above.

Solution

Here is a possible grammar G (axiom ITNR) for the sketched description language:

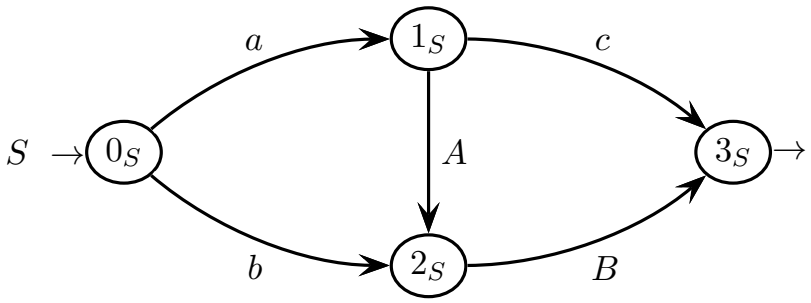
$$
\begin{cases}
\langle \text{ITNR} \rangle \to \text{itinerary } \langle \text{NAME} \rangle \ \langle \text{ETIME} \rangle \text{ includes } \langle \text{BODY} \rangle \text{ end } \langle \text{NAME} \rangle \\
\hline
\langle \text{ETIME} \rangle \to \text{estTime } \langle \text{NUM} \rangle \text{ hrs } \langle \text{NUM} \rangle \text{ min} \\
\langle \text{BODY} \rangle \to \langle \text{PNT} \rangle \text{ ';' } \big(\langle \text{ELM} \rangle \text{ ';' } \big)^{+} \langle \text{PNT} \rangle \text{ ';'} \\
\hline
\langle \text{PNT} \rangle \to \text{point } \langle \text{NAME} \rangle \ [\ \langle \text{ALT} \rangle \ ] \ \langle \text{LAT} \rangle \ \langle \text{LON} \rangle \\
\langle \text{ELM} \rangle \to \langle \text{PNT} \rangle \mid \langle \text{SITN} \rangle \\
\hline
\langle \text{SITN} \rangle \to \text{subitinerary } \langle \text{NAME} \rangle \text{ includes } \big(\langle \text{ELM} \rangle \text{ ';' } \big)^{2} \big(\langle \text{ELM} \rangle \text{ ';' } \big)^{*} \text{ end } \langle \text{NAME} \rangle \\
\hline
\langle \text{ALT} \rangle \to \text{altitude } \langle \text{NUM} \rangle \\
\langle \text{LAT} \rangle \to \text{latitude } \langle \text{NUM} \rangle \\
\langle \text{LON} \rangle \to \text{longitude } \langle \text{NUM} \rangle \\
\hline
\langle \text{NAME} \rangle \to \langle \text{ALPHA} \rangle \ \Big(\ (\text{'\_'})^{*} \ \big(\ \langle \text{ALPHA} \rangle \mid \langle \text{DIGIT} \rangle \ \big) \Big)^{*} \\
\langle \text{NUM} \rangle \to \langle \text{DIGIT} \rangle^{+} \ [\text{'.'} \ \langle \text{DIGIT} \rangle^{+} \ ] \\
\hline
\langle \text{ALPHA} \rangle \to [\text{'A'} - \text{'Z'}] \mid [\text{'a'} - \text{'z'}] \\
\langle \text{DIGIT} \rangle \to [\text{'0'} - \text{'9'}]
\end{cases}
$$

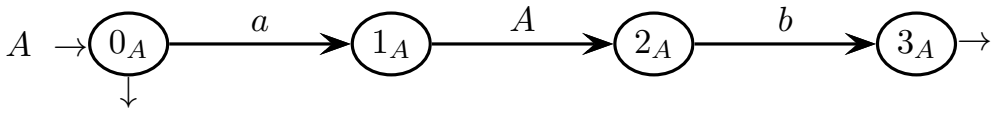
The grammar is not ambiguous. Identifiers are modeled as usual. Numbers may be natural or also have a fractional part, and leading and trailing zeroes are permitted. All the elements of a (sub)itinerary have a separator or a terminator, but the top-level itinerary does not have a terminator (as the example suggests). Minor details might be modeled more thoroughly, e.g., leading and trailing zeroes, or a slightly different placement of separators and terminators, etc. The choice of the syntax classes and of their names highlights the relevant substructures of the language.

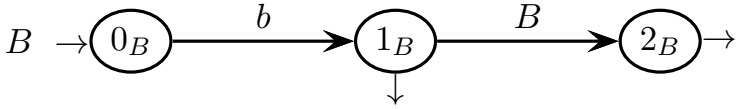
Syntax tree is left as an (easy) exercise.

3 Syntax Analysis and Parsing Methodologies 20%

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







Answer the following questions:

(a) Draw the complete pilot of grammar G and prove that the grammar is of type $ELR(1)$ (shortly justify your reasoning).

(b) Write the necessary guide sets on all the arcs of the machine net (call arcs and exit arrows) and prove that grammar G is of type $ELL(1)$, based on the guide sets (shortly justify your reasoning).

(c) Consider the sample valid string below:

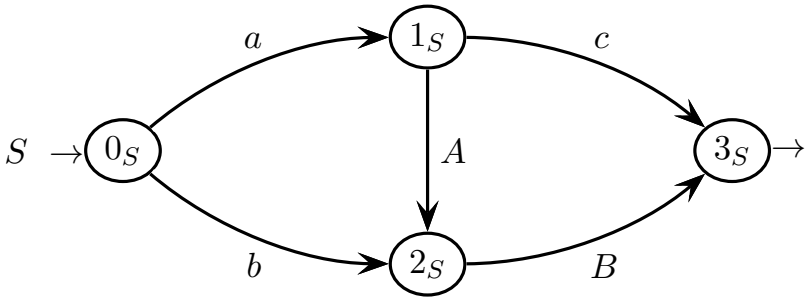
$$a\,a\,b\,b \in L(G)$$

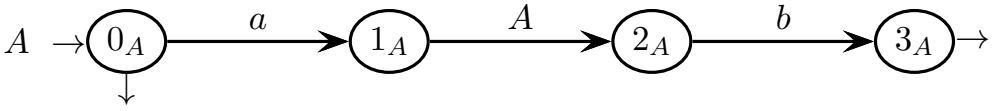
Simulate the recognition process of this string by the ELL syntax analyzer of grammar G (use the simulation table prepared on the next page).

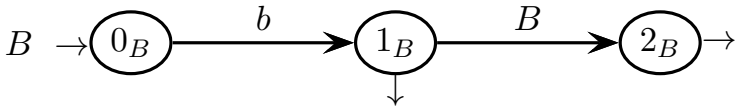
(d) (optional) Examine the language $L(G)$ and characterize its structure as a set of strings. Is language $L(G)$ regular ? (shortly justify your answer)

please here draw the pilot - question (a)

please here draw the call arcs and write all the guide sets - question (b)







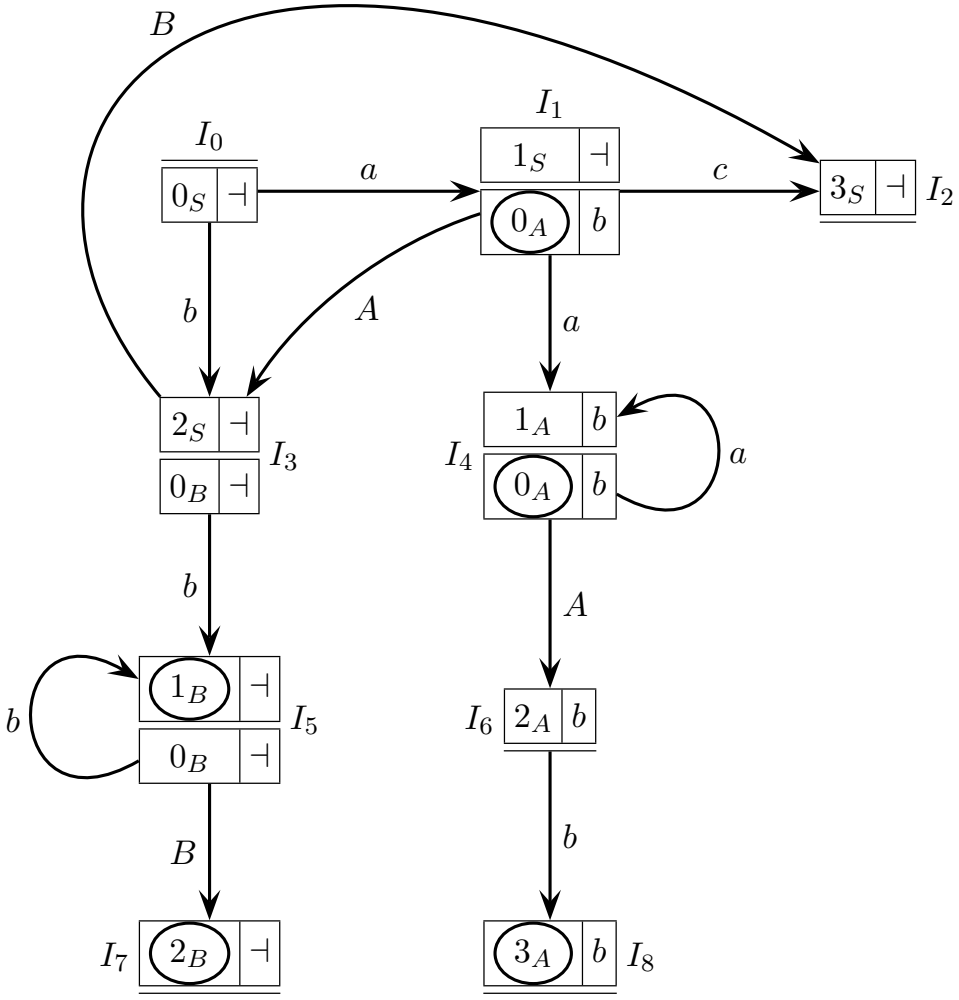
simulation table for the *ELL* analysis (num. of rows is not significant) - question (c)

stack of machine net states	*input left to be scanned*	*move executed*
0_S	$a\ a\ b\ b\ \dashv$	initial conf.: **call** machine M_S
1_S	$a\ b\ b\ \dashv$	shift: $0_S \xrightarrow{a} 1_S$

please here write the language characterization as a set of strings - question (d)

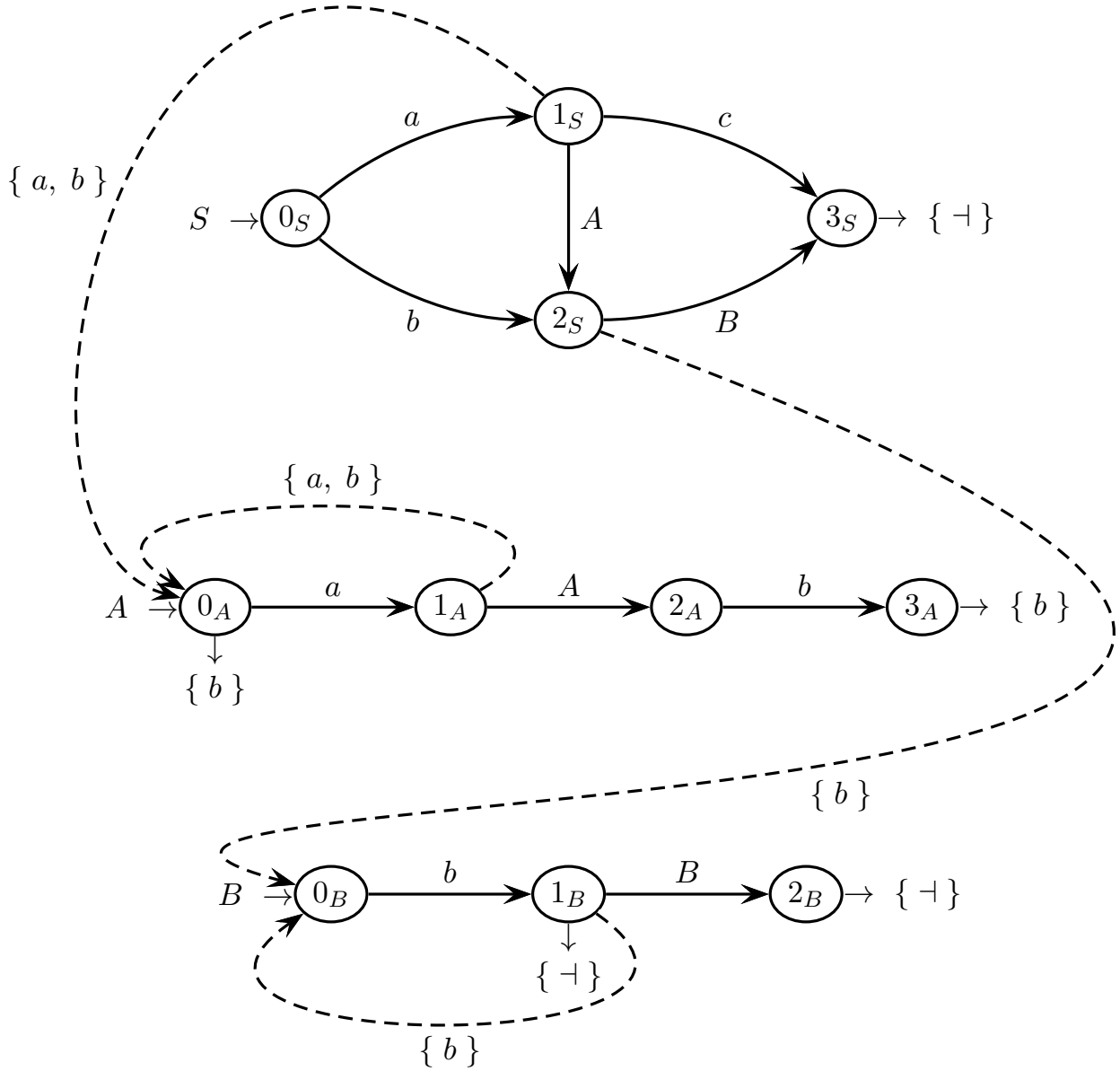
Solution

(a) Here is the complete pilot:



There are not any conflicts (SR, RR or C). Grammar G is $ELR\,(1)$.

(b) Here is the *PCFG*:



There are not any conflicts between the guide sets on the bifurcation states: 1_S, 0_A and 1_B. Grammar G is $ELL(1)$.

(c) Here is the *ELL* simulation of the valid sample string $a\,a\,b\,b \in L\,(G)$:

stack of machine net states	*input left to be scanned*	*move executed*
0_S	$a\ a\ b\ b\ \dashv$	initial conf.: **call** machine M_S
1_S	$a\ b\ b\ \dashv$	shift $0_S \xrightarrow{a} 1_S$
$2_S\ 0_A$	$a\ b\ b\ \dashv$	call M_A from 1_S on guide a
$2_S\ 1_A$	$b\ b\ \dashv$	shift $0_A \xrightarrow{a} 1_A$
$2_S\ 2_A\ 0_A$	$b\ b\ \dashv$	call M_A from 1_A on guide b
$2_S\ 2_A$	$b\ b\ \dashv$	return from M_A to 2_A on guide b
$2_S\ 3_A$	$b\ \dashv$	shift $2_A \xrightarrow{b} 3_A$
2_S	$b\ \dashv$	return from M_A to 2_S on guide b
$3_S\ 0_B$	$b\ \dashv$	call M_B from 2_S on guide b
$3_S\ 1_B$	$\dashv$	shift $0_B \xrightarrow{b} 1_B$
3_S	$\dashv$	return from M_B to 3_S on guide $\dashv$
the stack contents consist of one final axiomatic state exactly	empty tape	
acceptance condition verified — stop and accept		

(d) Here is a characterization as a set of strings:

$$L\,(B) = b^+$$

$$L\,(A) = a^h\ b^h \qquad h \geq 0$$

$$L\,(S) = a\ c\ \mid\ b\ L\,(B)\ \mid\ a\ L\,(A)\ L\,(B)$$

that is

$$L\,(S) = a\ c\ \mid\ b\ b^+ \mid\ a\ a^h\ b^h\ b^+ \qquad h \geq 0$$

Thus:

$$L\,(S) = \{\ a\ c,\ b^n,\ a^p\ b^q \mid\ \ n \geq 2\ \wedge\ q \geq p \geq 1\ \}$$

The language is not regular because of the strings $a^p\ b^q$ with $q \geq p$, essentially due to the auto-inclusive machine of nonterminal A.

4 Language Translation and Semantic Analysis 20%

1. Consider the *BNF* source grammar G_s below, over the two-letter terminal alphabet $\{ a, b \}$ and the two-letter nonterminal alphabet $V = \{ A, S \}$ (axiom S).

$$G_s \begin{cases} S & \rightarrow & a\ S\ b\ b \\ S & \rightarrow & A \\ A & \rightarrow & a\ A\ b\ b\ b \\ A & \rightarrow & \varepsilon \end{cases}$$

The language L_s generated by G_s includes, for instance, the strings ε, $a^3\,b^6$, $a^3\,b^7$, $a^3\,b^8$ and $a^3\,b^9$, but it does not include the strings $a^3\,b^5$ and $a^3\,b^{10}$.

Please answer the following questions:

(a) Write a target (or destination) grammar G_t associated to the source grammar G_s, without changing G_s, that defines the following translation τ:

$$\tau\,(a^m\,b^n) = a^n\,b^m$$

(b) Draw the syntax trees of the source and target strings for the translation example: $\tau\,(a^3\,b^8) = a^8\,b^3$.

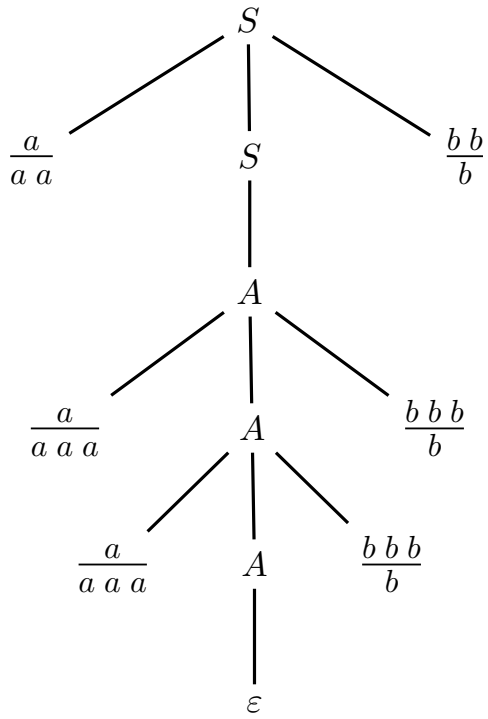
(c) Say if the translation function τ can be computed deterministically or not, and adequately justify your answer.

Solution

(a) Here is the target grammar G_t (axiom S):

$$G_t \begin{cases} S & \to & a\ a\ S\ b \\ S & \to & A \\ A & \to & a\ a\ a\ A\ b \\ A & \to & \varepsilon \end{cases}$$

(b) Here is the syntax tree (unified) for the translation example $\tau\,(a^3\,b^8) = a^8\,b^3$:



(c) The translation function τ can be computed deterministically, by a very simple pushdown transducer that stores the source string in the stack and then outputs it, in the reverse order, changing letters a into letters b and viceversa.

2. Consider the two-level lists defined by the following abstract syntax (axiom A):

$$\begin{cases} 1\colon\ A \rightarrow \varepsilon & \text{— 1-st level list} \\ 2\colon\ A \rightarrow a\ A & \\ 3\colon\ A \rightarrow a\ B\ A & \\ 4\colon\ B \rightarrow b & \text{— 2-nd level list} \\ 5\colon\ B \rightarrow B\ b & \end{cases}$$

Notice that the 2-nd level lists, i.e., the substrings of letters b, are optional.

We want to verify that the two-level list is ordered from left to right by strictly increasing length of the 2-nd level lists of elements b, if any. For instance, both the two-level lists below are syntactically valid, but:

- the two-level list $a\,b\,a\,a\,b\,b$ is semantically *correct*
- the two-level list $a\,b\,b\,a\,a\,b$ is semantically *incorrect*

Attributes permitted to use (no other attributes are allowed):

name	*type*	*domain*	*symbol*	*meaning*
len	left	integer	B	length of a 2-nd level list, i.e., a substring of letters b
lim	left	integer	A	length of the *shortest* 2-nd level list in the tree rooted at A (conventionally = 0 if the tree rooted at A does not include any 2-nd level list)
corr	left	boolean	A	true if the two-level list is semantically correct as explained, else false
err	right	boolean	B	true if a 2-nd level list, i.e., a substring of letters b, violates the semantic correctness of the two-level list, else false — this attribute is for question (c)

Answer the following questions:

(a) Write an attribute grammar (in the table prepared on the next page), based on the above syntax and using only the first three attributes listed above (*len*, *lim* and *corr*), that computes the value of the attribute *corr* in the tree root.

(b) Test the attribute grammar of point (a) by decorating with the attribute values the syntax tree of the correct two-level list $a\,b\,a\,a\,b\,b$.

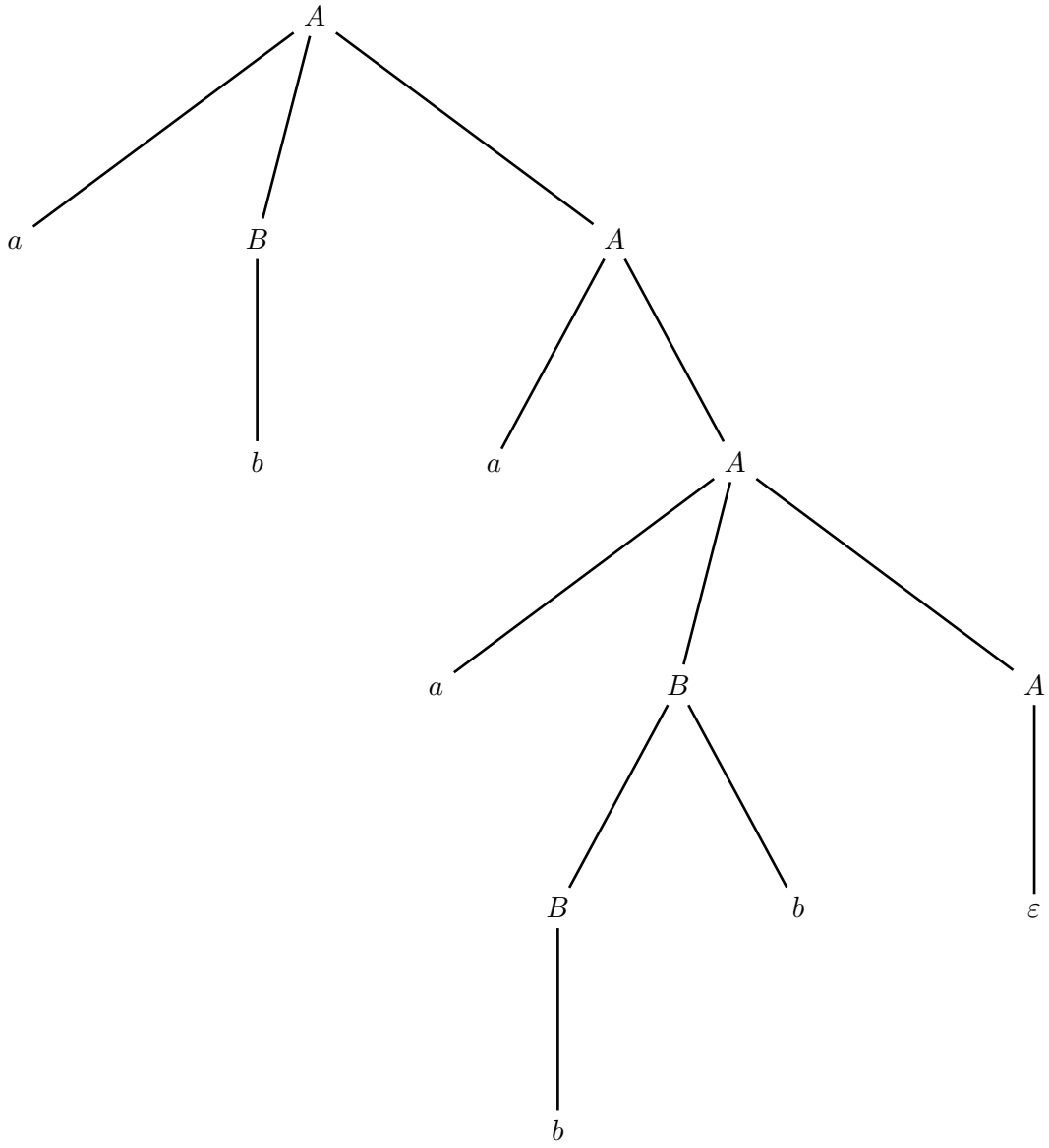
(c) (optional) Extend the attribute grammar of point (a) to define the value of the attribute *err* of B, so to allow for printing a diagnostic at the left end of each 2-nd level list that violates the correctness of the whole two-level list. Is the extended attribute grammar of type one-sweep ? (justify your answer)

attribute grammar to write - question (a)

name	*type*	*domain*	*symbol*	*meaning*
len	left	integer	B	length of a 2-nd level list
lim	left	integer	A	length of the shortest 2-nd level list
corr	left	boolean	A	true if the two-level list is correct

#	*syntax*	*semantics*
1:	$A_0 \rightarrow \varepsilon$	
2:	$A_0 \rightarrow a\ A_1$	
3:	$A_0 \rightarrow a\ B_1\ A_2$	
4:	$B_0 \rightarrow b$	
5:	$B_0 \rightarrow B_1\ b$	

syntax tree to be decorated - question (b)



attribute grammar to extend - question (c)

name	*type*	*domain*	*symbol*	*meaning*
the same attributes as for question (a) plus the attribute *err* below				
err	right	boolean	B	true if a 2-nd level list violates the correctness of the two-level list — for (c)

#	*syntax*	*semantics*
1:	$A_0 \rightarrow \varepsilon$	
2:	$A_0 \rightarrow a\ A_1$	
3:	$A_0 \rightarrow a\ B_1\ A_2$	
4:	$B_0 \rightarrow b$	
5:	$B_0 \rightarrow B_1\ b$	

Solution

(a) Here is the attribute grammar:

#	*syntax*	*semantics*
1:	$A_0 \rightarrow \varepsilon$	$lim_0 = 0$ $corr_0 = true$
2:	$A_0 \rightarrow a\ A_1$	$lim_0 = lim_1$ $corr_0 = corr_1$

3:	$A_0 \rightarrow a\ B_1\ A_2$	**if** $(\ lim_2 \neq 0\ )$ **then**
		$lim_0 = \min(\ len_1,\ lim_2\ )$
		$corr_0 = (\ len_1 < lim_2\ )\ \wedge\ corr_2$
		else
		$lim_0 = len_1$
		$corr_0 = corr_2$
		endif

4:	$B_0 \rightarrow b$	$len_0 = 1$
5:	$B_0 \rightarrow B_1\ b$	$len_0 = len_1 + 1$

A different approach might extend the domain of the attribute *lim* and include a special value ∞, so the initialization in the rule 1 could be changed into $lim_0 = \infty$, the test on *lim* in the rule 2 could be removed and the assignments therein simplified into $lim_0 = \min(\ len_1,\ lim_2\ )$ and $corr_0 = (\ len_1 < lim_2\ )\ \wedge\ corr_2$, that is, the **else** branch of the conditional would become useless. This way:

#	*syntax*	*semantics*
1:	$A_0 \rightarrow \varepsilon$	$lim_0 = \infty$ $corr_0 = true$
2:	$A_0 \rightarrow a\ A_1$	$lim_0 = lim_1$ $corr_0 = corr_1$
3:	$A_0 \rightarrow a\ B_1\ A_2$	$lim_0 = \min(\ len_1,\ lim_2\ )$ $corr_0 = (\ len_1 < lim_2\ )\ \wedge\ corr_2$
4:	$B_0 \rightarrow b$	$len_0 = 1$
5:	$B_0 \rightarrow B_1\ b$	$len_0 = len_1 + 1$

Of course, in this way the domain of attribute *lim* becomes: integer $\cup\ \{\ \infty\ \}$.

(b) See Figure 3.

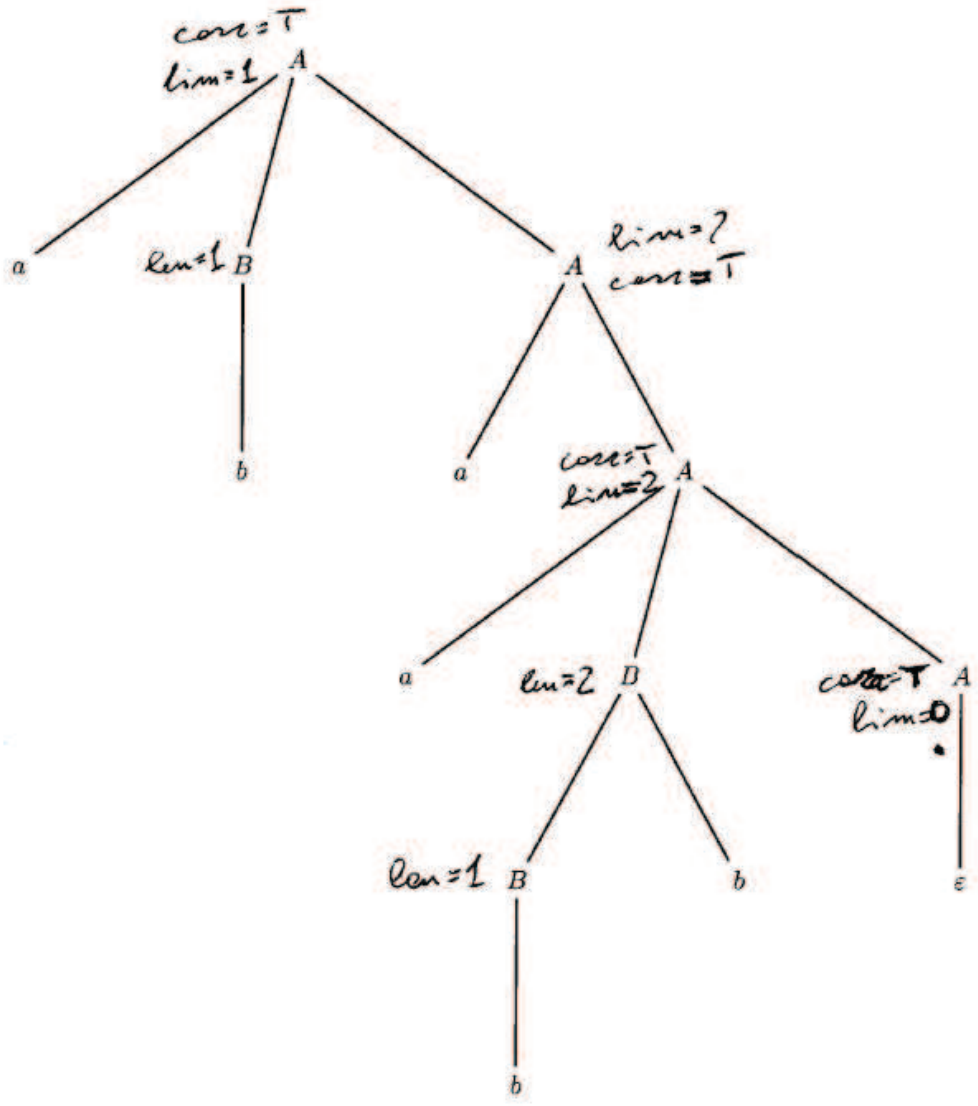


Figure 3: Decorated tree.

(c) Here is the extended attribute grammar:

#	*syntax*	*semantics*
1:	$A_0 \rightarrow \varepsilon$	$lim_0 = 0$ $corr_0 = true$
2:	$A_0 \rightarrow a\ A_1$	$lim_0 = lim_1$ $corr_0 = corr_1$

3:	$A_0 \rightarrow a\ B_1\ A_2$	**if** $(\ lim_2 \neq 0\ )$ **then**
		$err_1 = (\ len_1 \geq lim_2\ )$
		$lim_0 = \min(\ len_1,\ lim_2\ )$
		$corr_0 = (\ len_1 < lim_2\ )\ \wedge\ corr_2$
		— or $corr_0 = \neg err_1\ \wedge\ corr_2$
		else
		$err_1 = false$
		$lim_0 = len_1$
		$corr_0 = corr_2$
		endif

4:	$B_0 \rightarrow b$	$len_0 = 1$
5:	$B_0 \rightarrow B_1\ b$	$err_1 = err_0$ $len_0 = len_1 + 1$

It might be changed as explained before.

The extended grammar is not of type one-sweep, as the inherited attribute *err* depends on a synthesized attribute, namely *len*, of the same node, namely the node B in the rule 3.