30 AUGUST 2021

1 Regular Expressions and Finite Automata 20%

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

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

Answer the following questions:

(a) 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 $a < b < c$).

(b) By means of the Berry-Sethi (BS) method, find a deterministic automaton A equivalent to the regular expression R.

(c) Is the deterministic automaton A found before minimal ? Justify your answer.

(d) By means of the Brzozowski method (node elimination), starting from the automaton A found before, obtain a regular expression R' equivalent to A.

~~(e) (optional) Is the language $L\,(R)$ locally testable ? Formally prove your answer.~~

TOPIC NOT COVERED IN A.A. 23/24

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

a)

1	a
2	ab
3	abb, acca
4	abbb, abcca, acab, accc a

b) BERRY-SETHI

$$R' = a_1(b_2|c_3^+a_4)^*$$

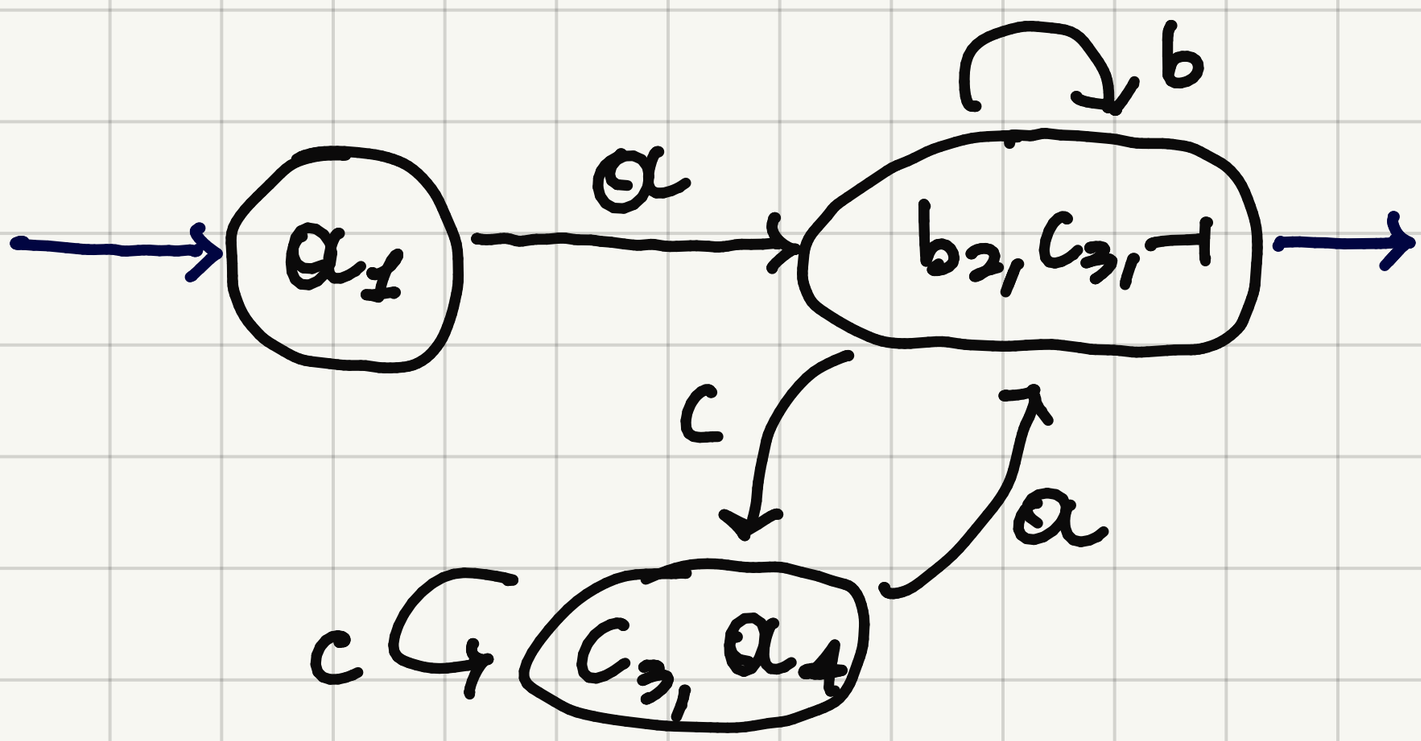
initials = $\{a_1\}$

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

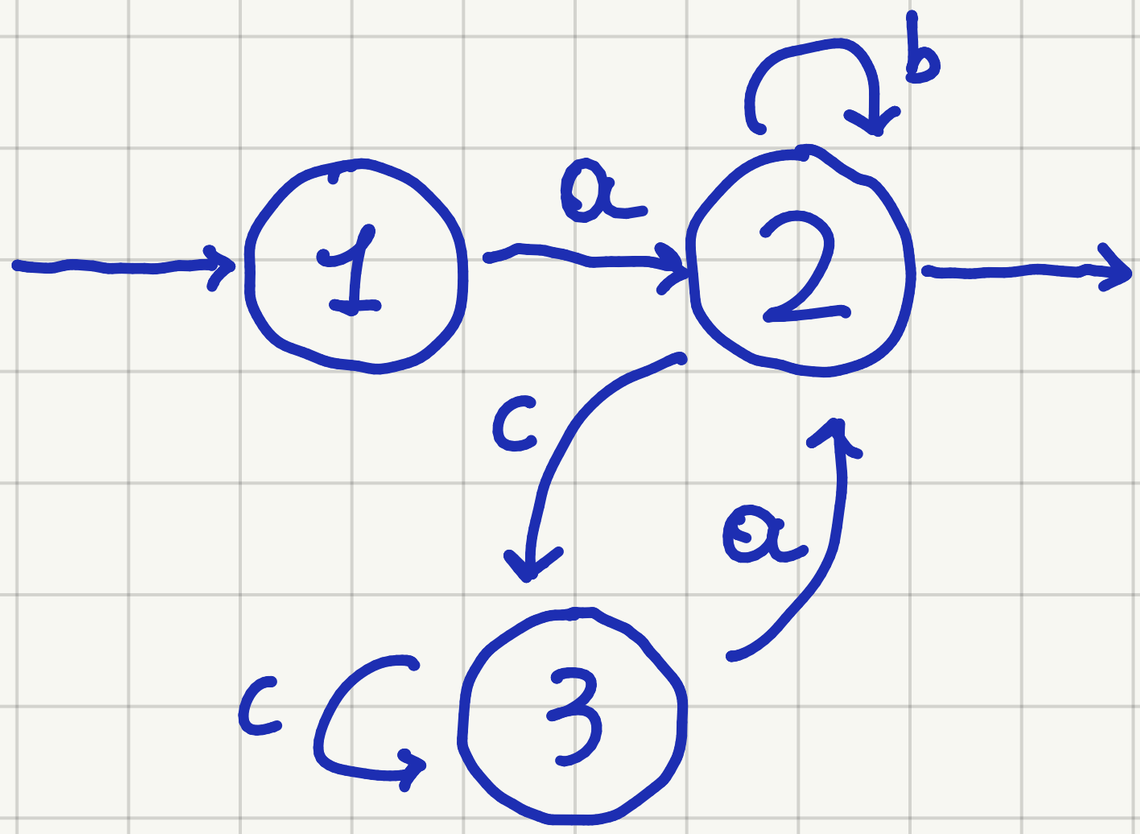
"AFTER a_1 WE CAN FIND EITHER b_2 OR c_3 OR NOTHING"

SINCE a_1, b_2 AND a_4 HAVE THE SAME FOLLOWERS, WE CAN SIMPLIFY THE TABLE

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



THE FINAL RESULT WILL BE



c)

	1	2	3
1			
2	X		
3	X	X	

DETERMINE, FOR EACH PAIR, IF THEY ARE UNDISTINGUISHABLE

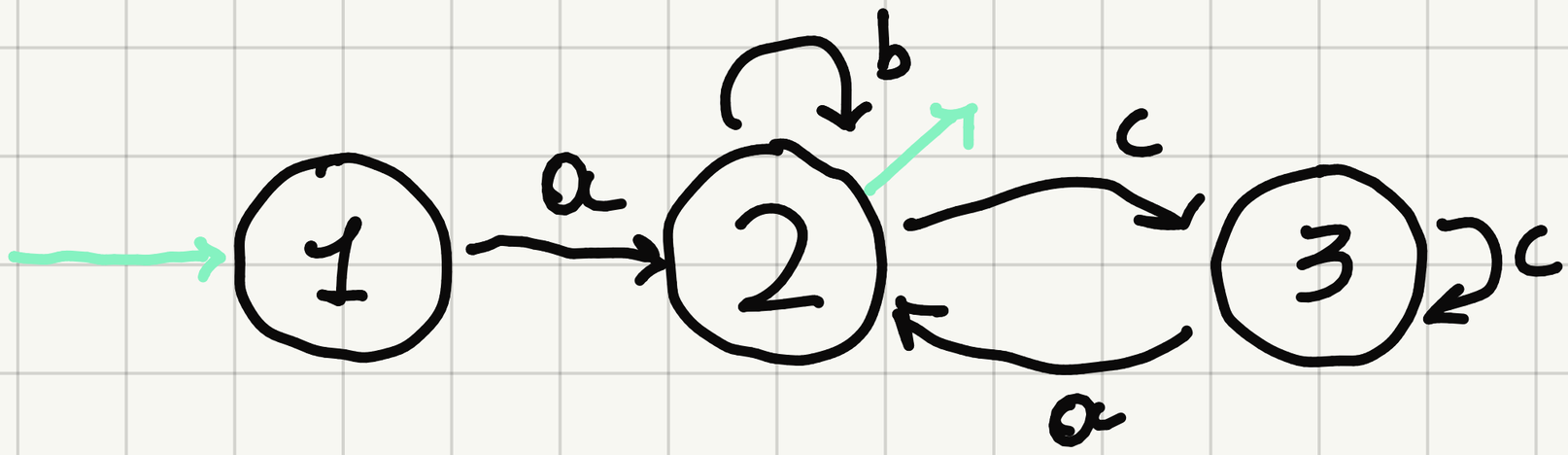
2 IS FINAL, 1 AND 3 AREN'T ⇒ THEY ARE DISTINGUISHABLE

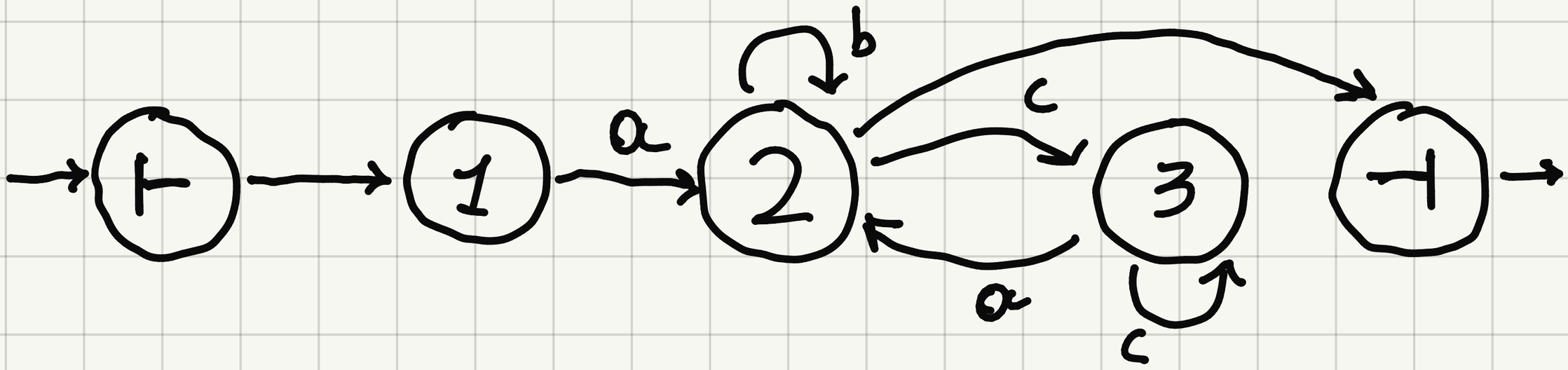
ARE 1 AND 3 DISTINGUISHABLE ω = NOTHING

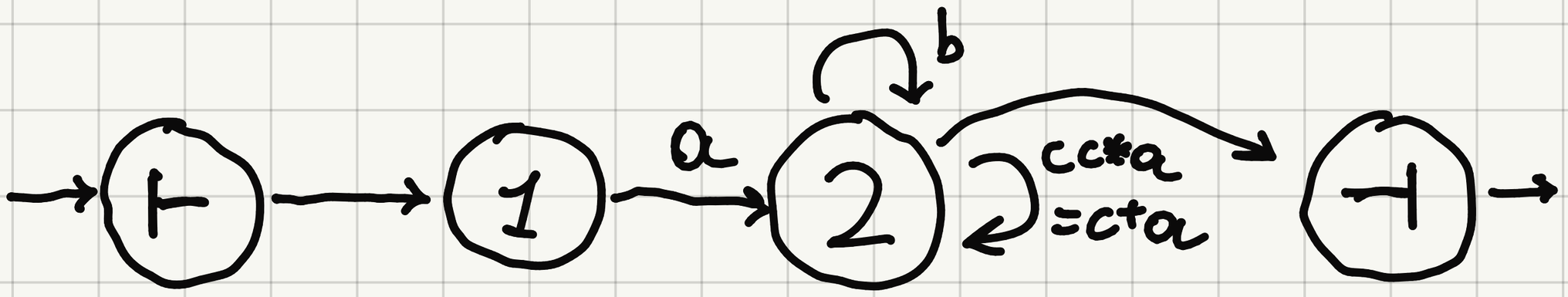
$$\begin{cases} \delta(1,a)=2 \\ \delta(1,b)=\omega \\ \delta(1,c)=\omega \end{cases} \qquad \begin{cases} \delta(3,a)=2 \\ \delta(3,b)=\omega \\ \delta(3,c)=3 \end{cases}$$

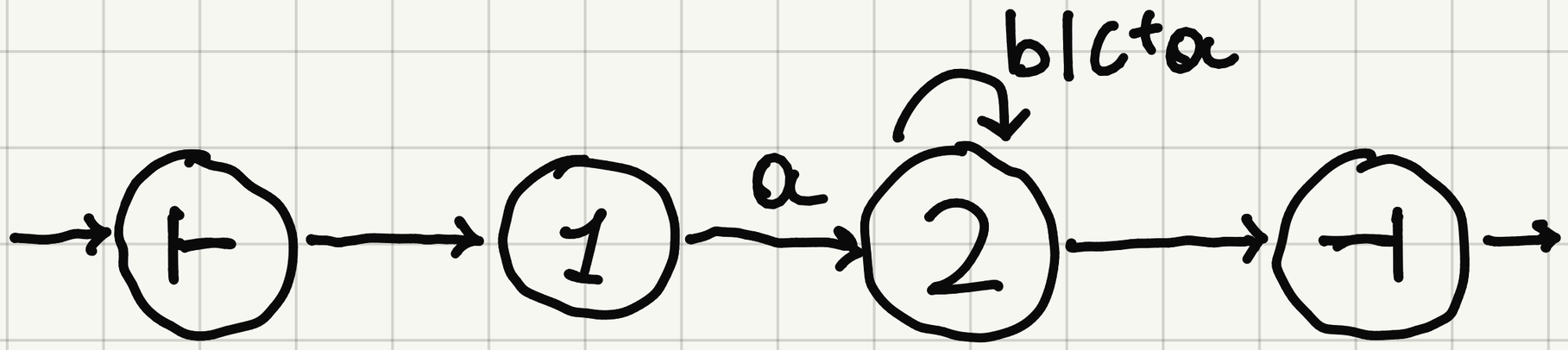
1 AND 3 ARE DISTINGUISHABLE ⇒ THE AUTOMATON IS ALREADY MINIMAL

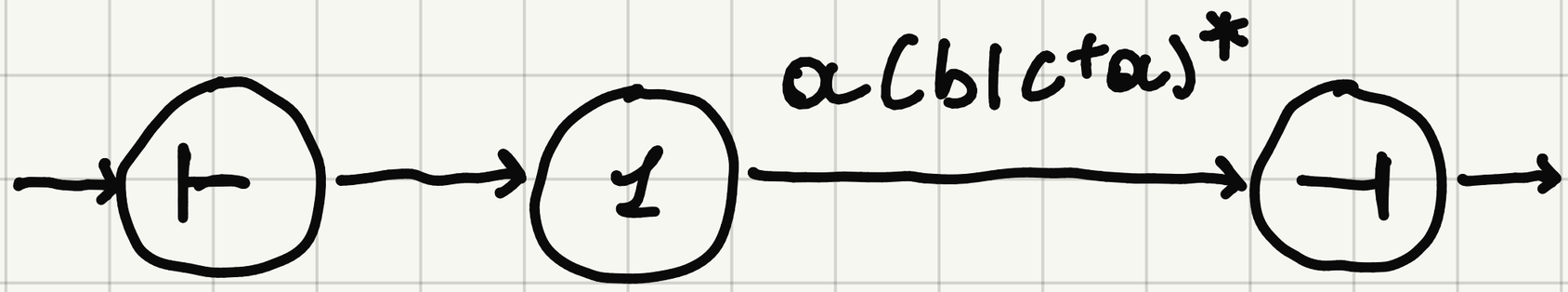
d)

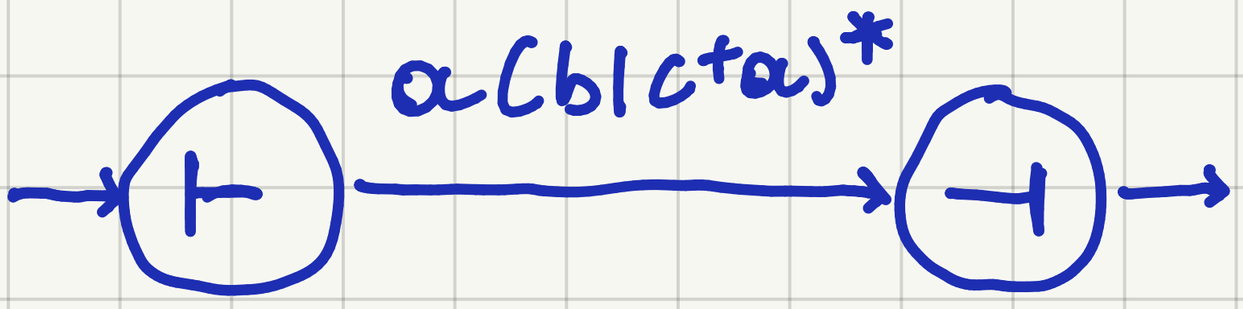










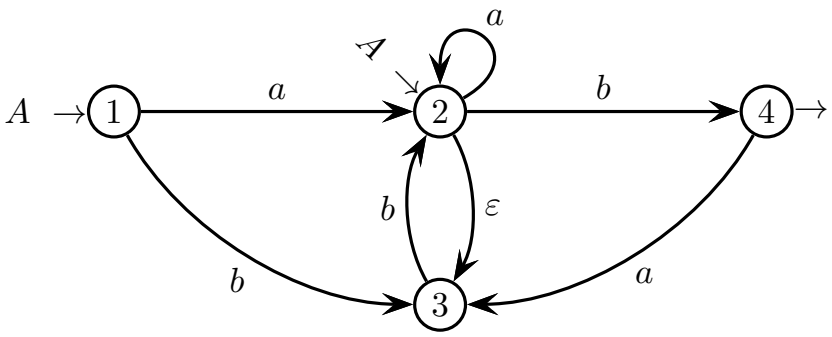


$R' = a(b|c+a)^*$ MA È LETTERALMENTE UGUALE AD R...

5 JULY 2017

1 Regular Expressions and Finite Automata 20%

1. Consider the nondeterministic finite-state automaton A below, over the two-letter alphabet $\Sigma = \{\ a,\ b\ \}$, with two initial states 1 and 2, and with one final state 4:



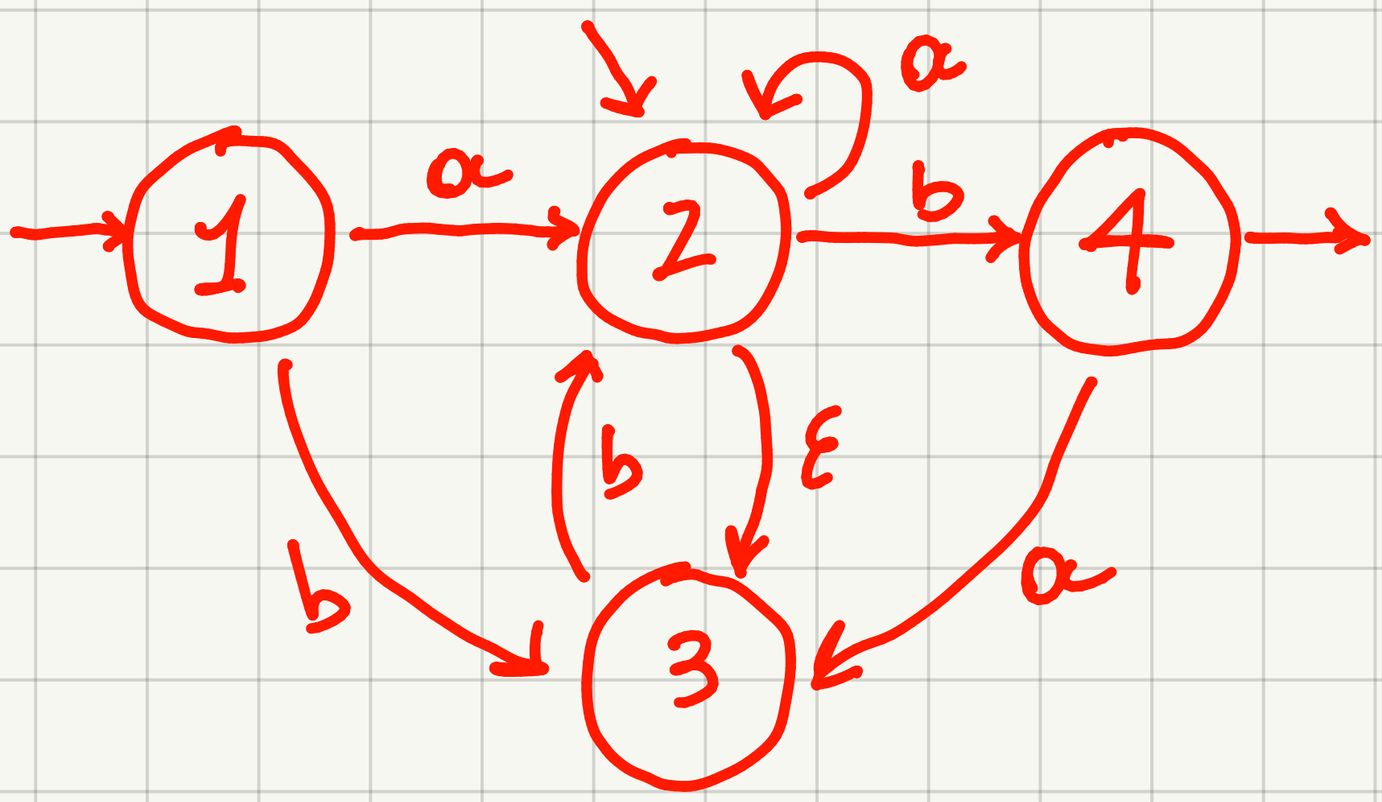
Answer the following questions:

(a) Find the shortest string of the regular language $L(A)$ that is recognized by automaton A with at least two different computations and that therefore is ambiguous, and suitably justify your answer. Provide evidence that the string is the shortest one with such a property.

(b) Transform automaton A and obtain an automaton A', possibly still nondeterministic, that has only one initial state and does not have any spontaneous transitions (ε-transitions), by properly cutting such transitions.

(c) By using the Berry-Sethi method, obtain a deterministic automaton A'' equivalent to the original nondeterministic automaton A.

(d) (optional) Minimize the deterministic automaton A'' found at point (c). Then write a regular expression R equivalent to the automaton A'' found. Notice: for this whole question, you may proceed systematically or informally, but in the latter case provide some evidence of correctness.

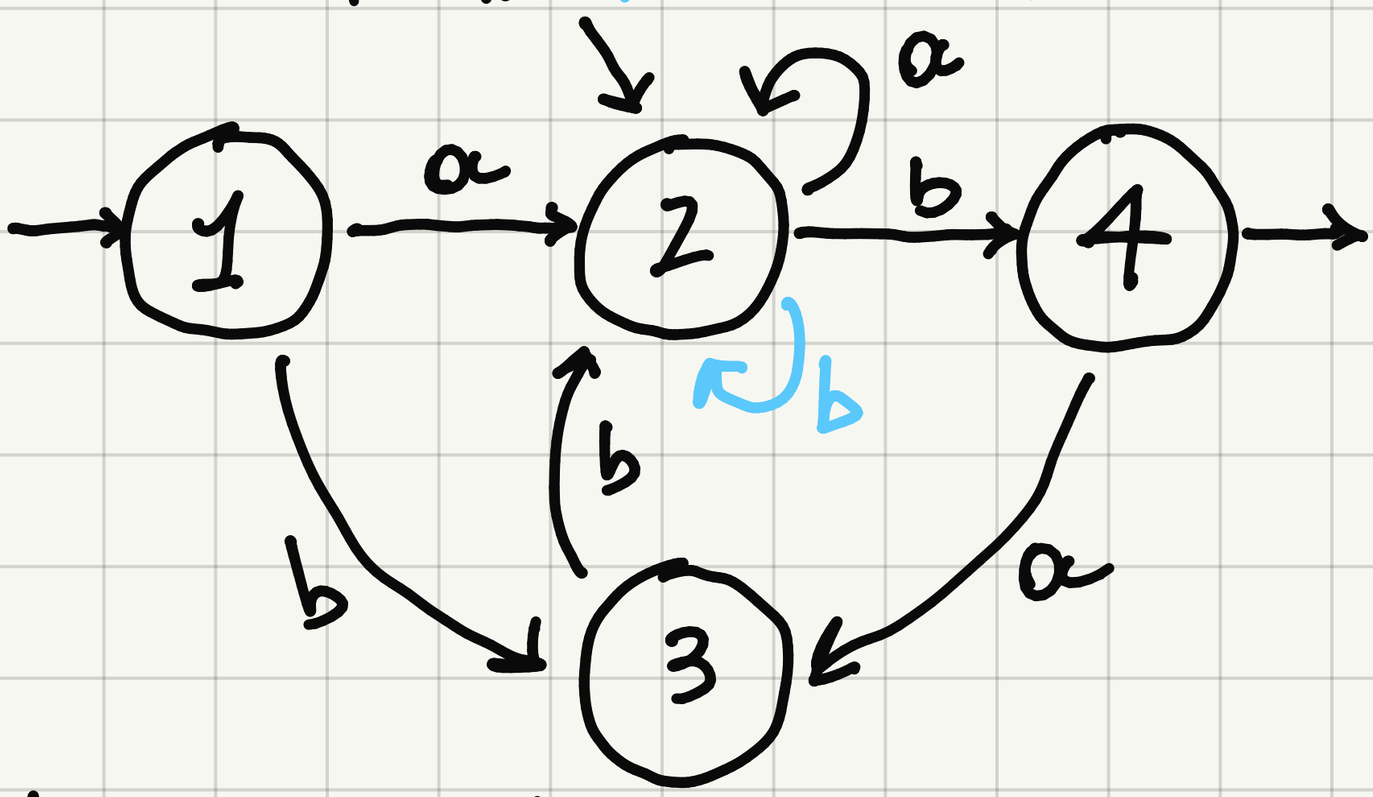


a) ab CAN BE RECOGNIZED BY AUTOMATON A WITH THOSE DIFFERENT COMPUTATIONS: $1 \xrightarrow{a} 2 \xrightarrow{b} 4$; $2 \xrightarrow{a} 2 \xrightarrow{b} 4$

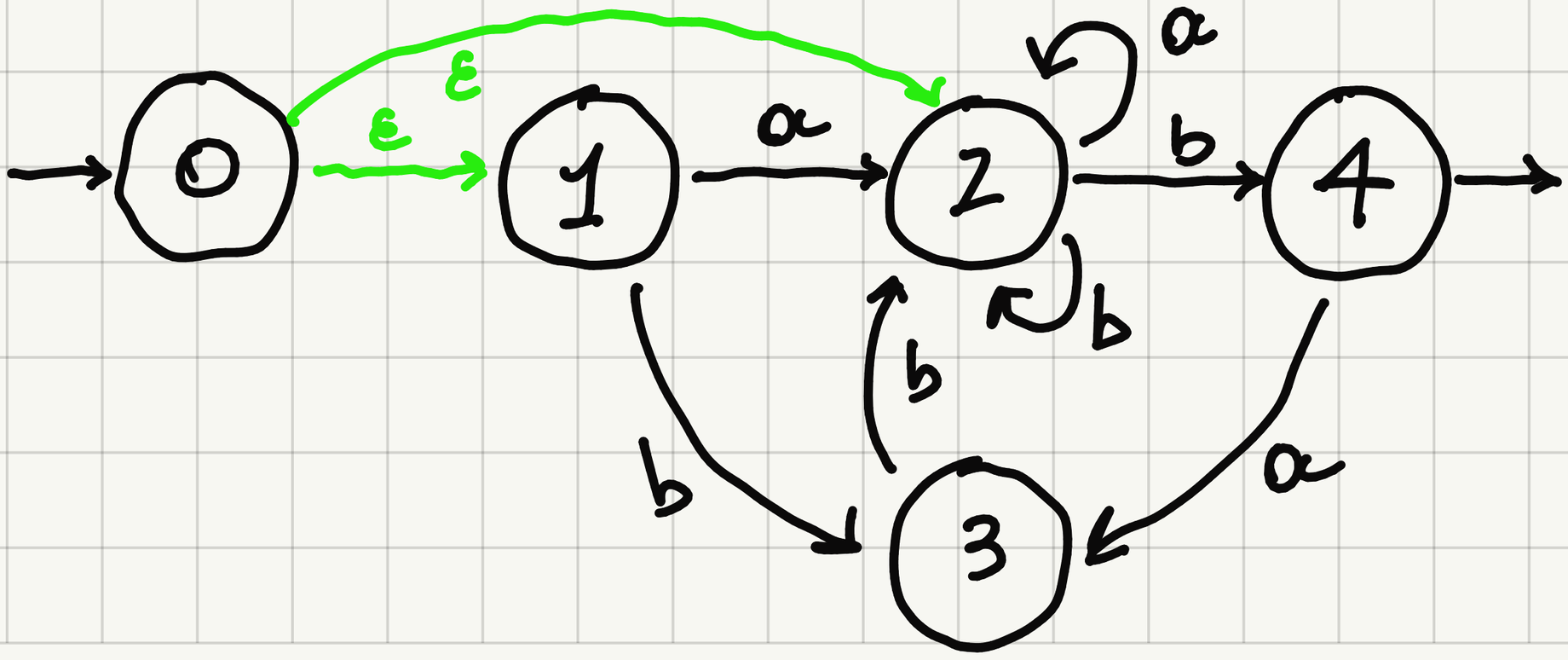
WE CAN'T HAVE AMBIGUOUS STRINGS OF LENGHT 1 SINCE THE ONLY ONE CAN BE b, WHOSE UNIQUE COMPUTATION IS $2 \xrightarrow{b} 4$

DOMANDA: DEVO VERIFICARE bb, aa, ba?

b) LET'S START BY REMOVING ε

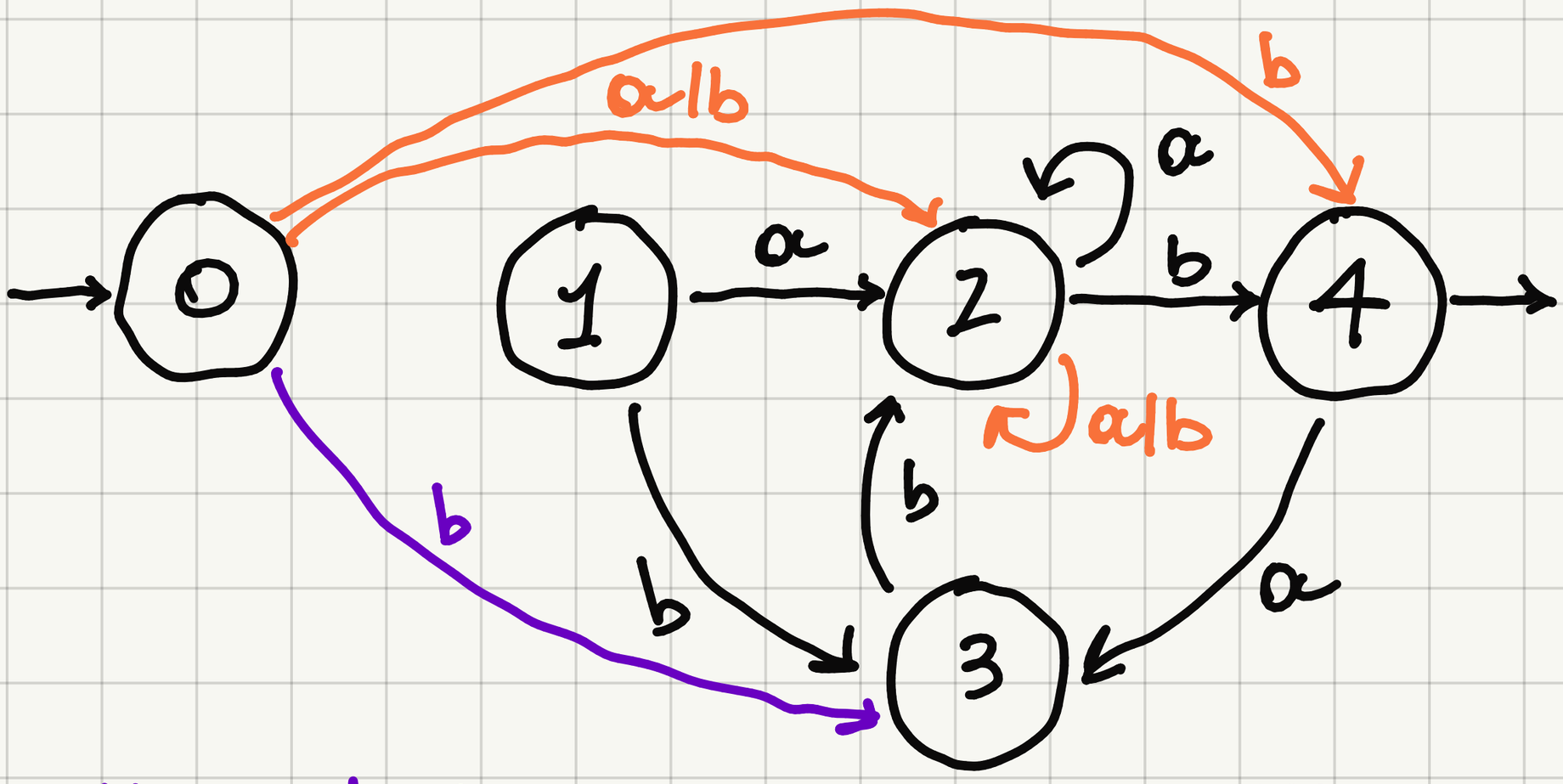


NOW, THE IDEA IS TO ADD A NEW UNIQUE INIMAL STATE



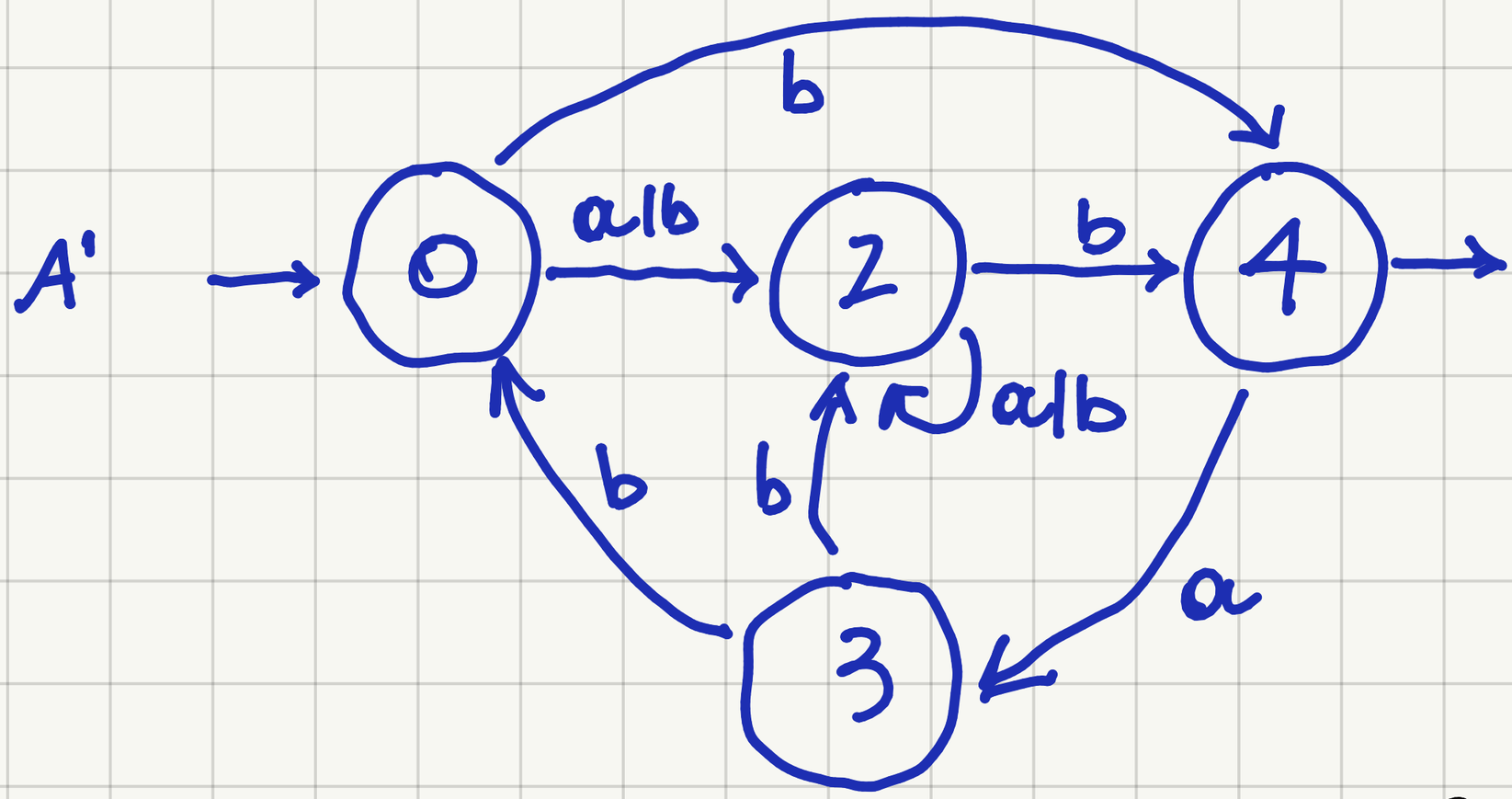
BUT NOW WE HAVE A NEW PROBLEM: DELETE THE "NEW"

EMPTY MOVES



- MOVES PERFORMED FROM PREVIOUS INITIAL STATE 1
- MOVES PERFORMED FROM PREVIOUS INITIAL STATE 2

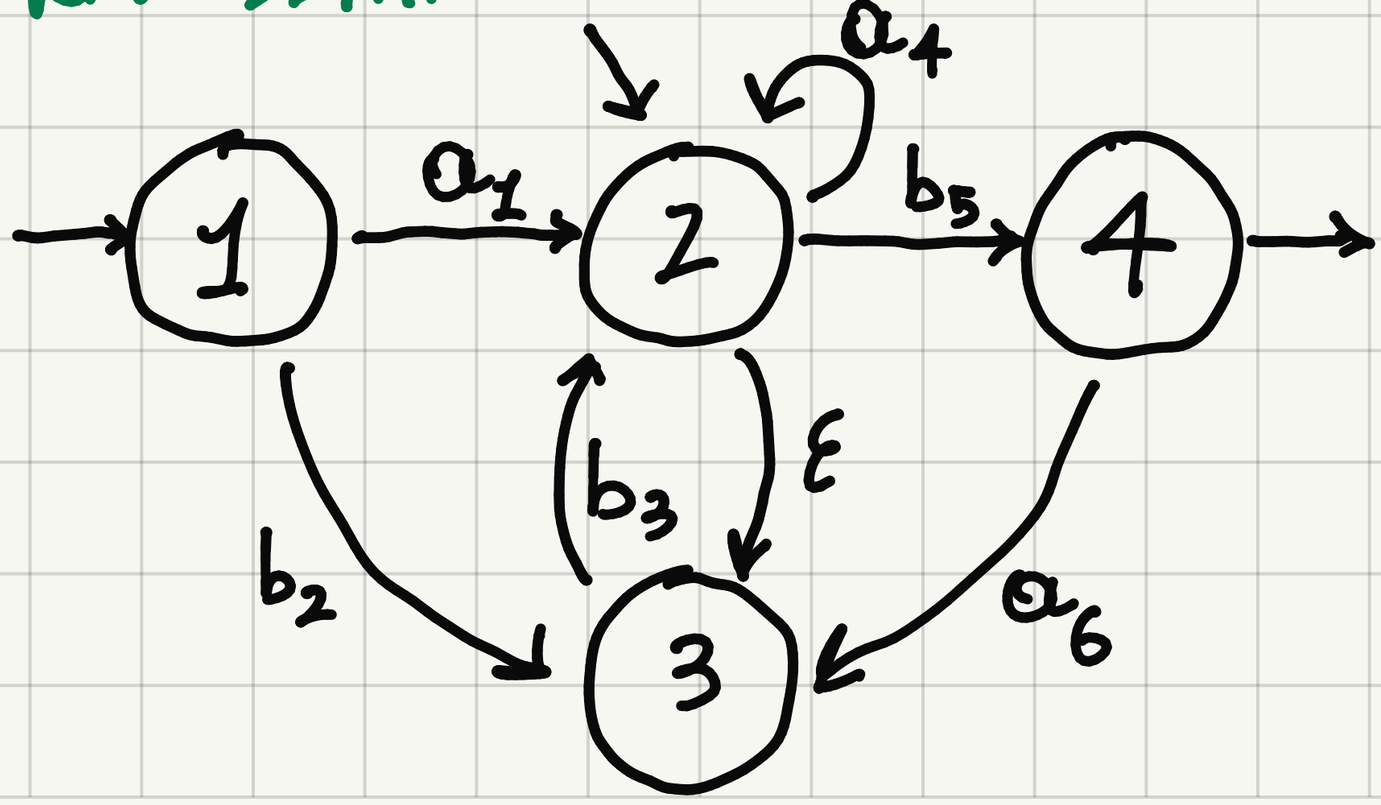
NOTICE THAT THERE IS NO PATH THAT GOES TO 1



c) BERRY-SETHI

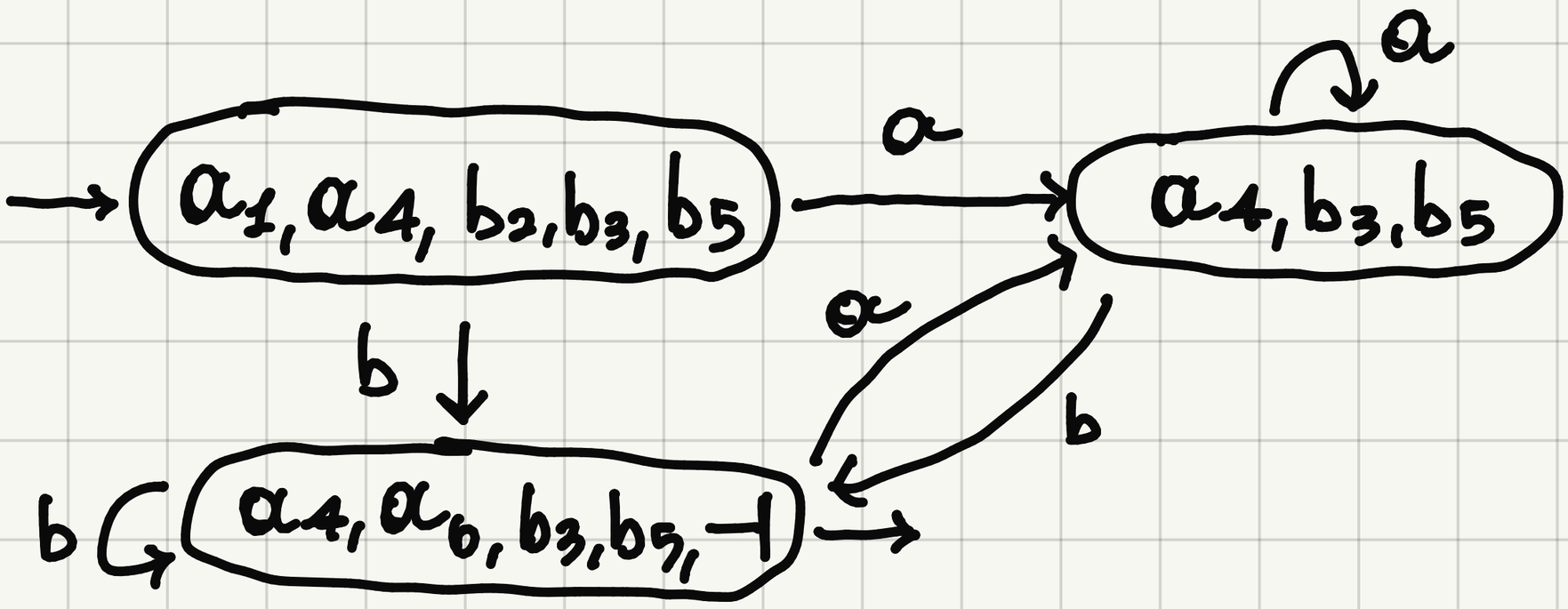
initials = $\{a_1, b_2, b_3, a_4, b_5\}$

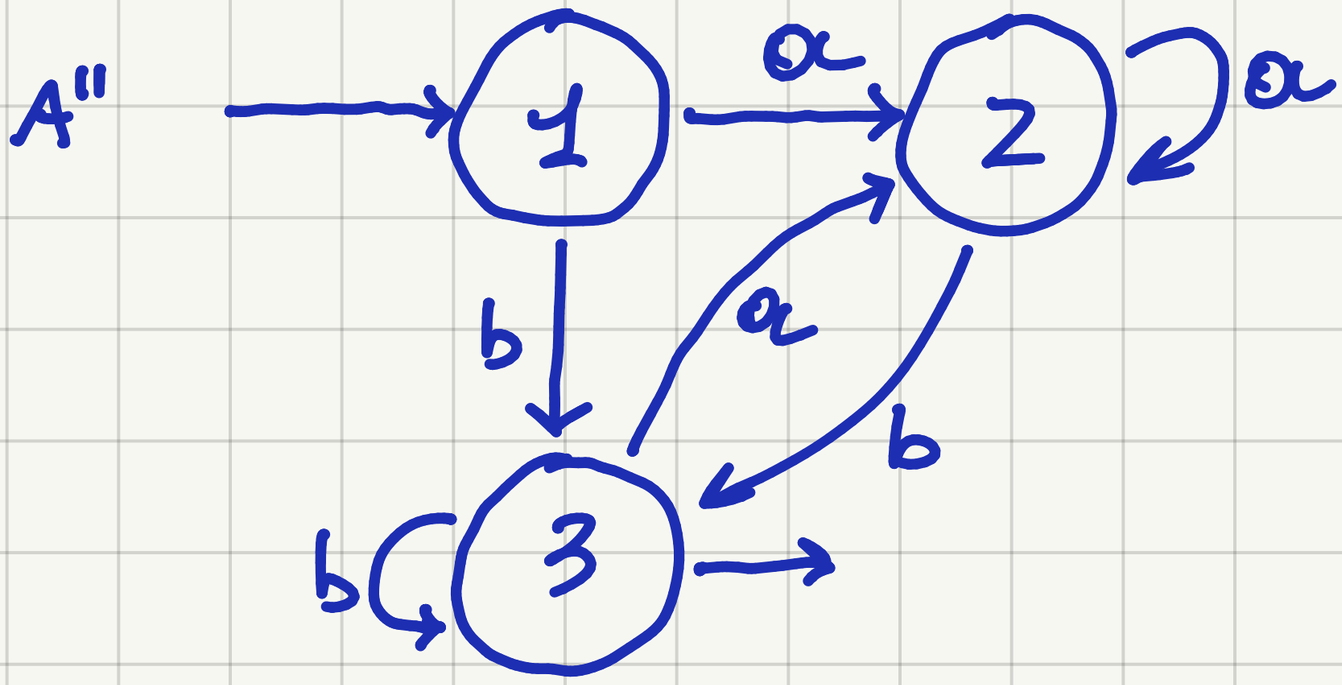
$= \{a_1, a_4, b_2, b_3, b_5\}$



Terminals	Followers
a_1, b_3, a_4	a_1, b_3, b_5
b_2, a_6	b_3
b_5	$a_6, \dashv$

$b_3 = \varepsilon b_3$





d)

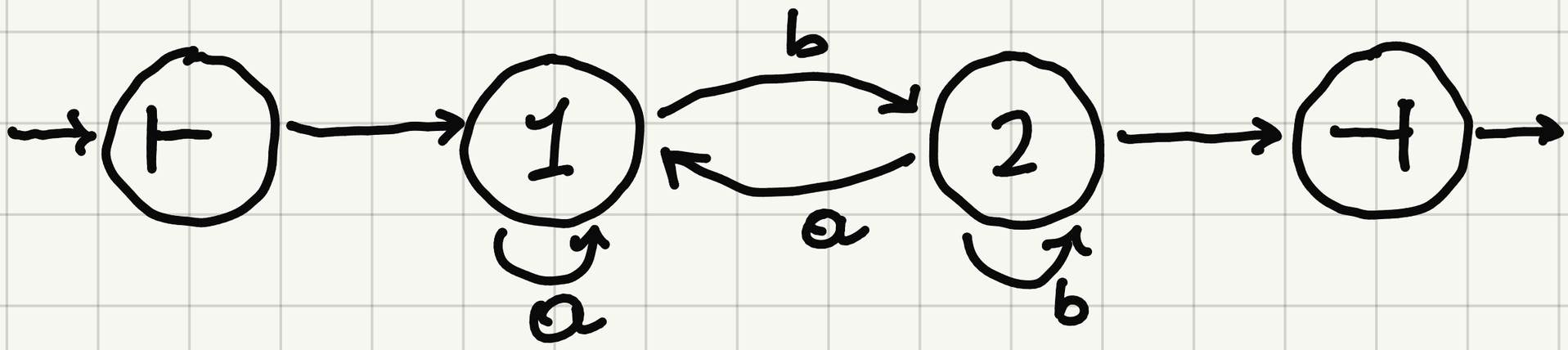
	1	2	3
1			
2	=		
3	X	X	

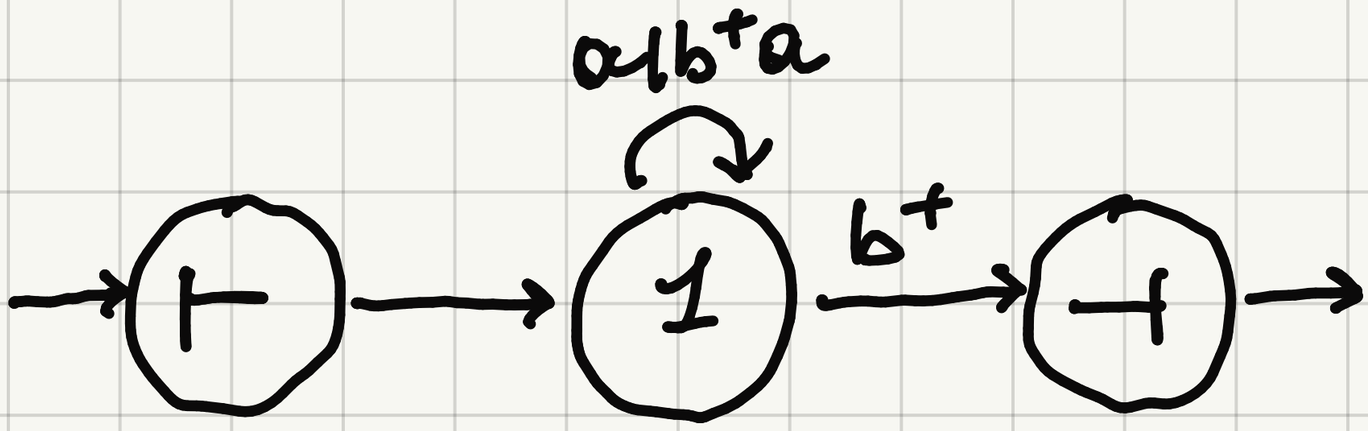
3 IS FINAL, WHILE 1 AND 2 ARE NOT

$\begin{cases} \delta(1,a) = 2 \\ \delta(1,b) = 3 \end{cases}$ $\begin{cases} \delta(2,a) = 2 \\ \delta(2,b) = 3 \end{cases}$

$\Rightarrow$ 1 AND 2 ARE UNDISTINGUISHABLE

WE NEED TO MINIMIZE A''





$R = \Sigma^* b$

5 MARCH 2014

1 Regular Expressions and Finite Automata 20%

1. Consider the right-linear grammar G below (axiom S), over the terminal alphabet $\{ a, b, c \}$:

$$G \begin{cases} S & \rightarrow & a\, A \\ A & \rightarrow & a\, B \mid \varepsilon \\ B & \rightarrow & c\, S \mid b\, A \end{cases}$$

Answer the following questions:

(a) ~~By means of the method of the linear language equations, find a regular expression R equivalent to grammar G.~~

(b) By means of the Berry-Sethi method, applied to the regular expression R found before, build a deterministic finite automaton A that recognizes the language $L(G)$ of grammar G.

(c) (Optional) Build a deterministic finite automaton A' that recognizes the language $\neg L(G)$, i.e., the complement language of $L(G)$.

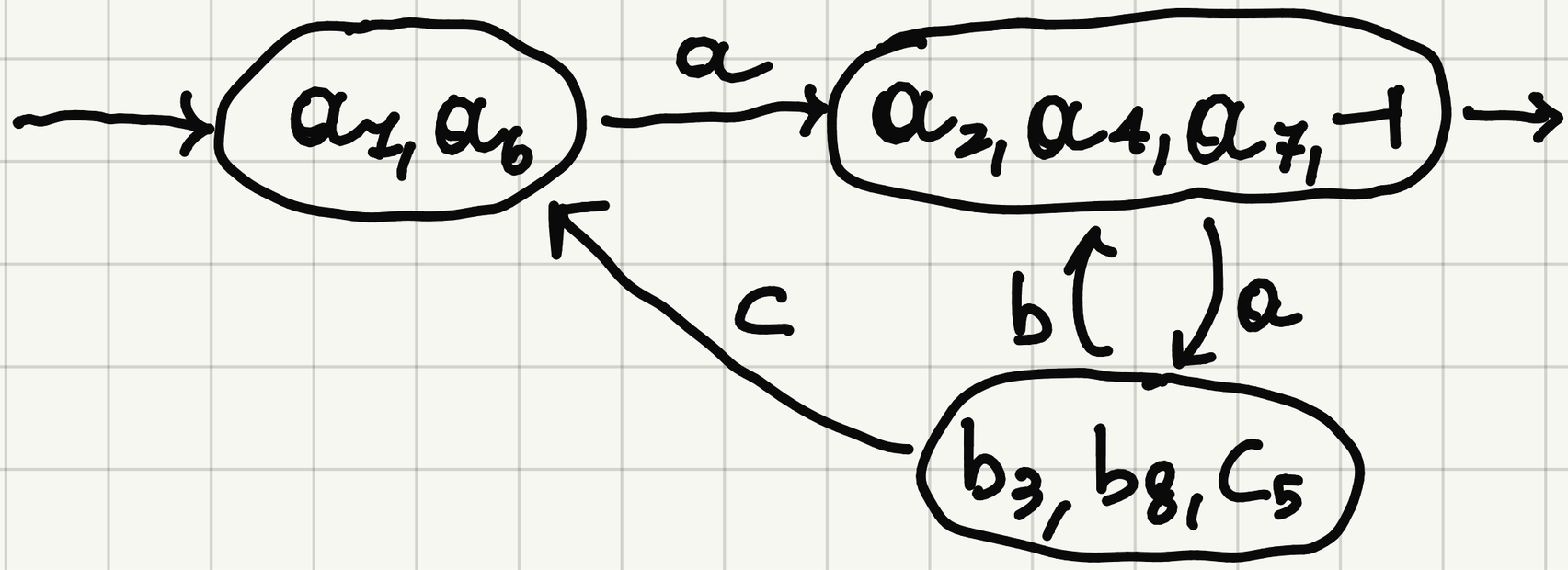
SPOILER FROM a) $R=(a(ab)^*ac)^*a(ab)^*$

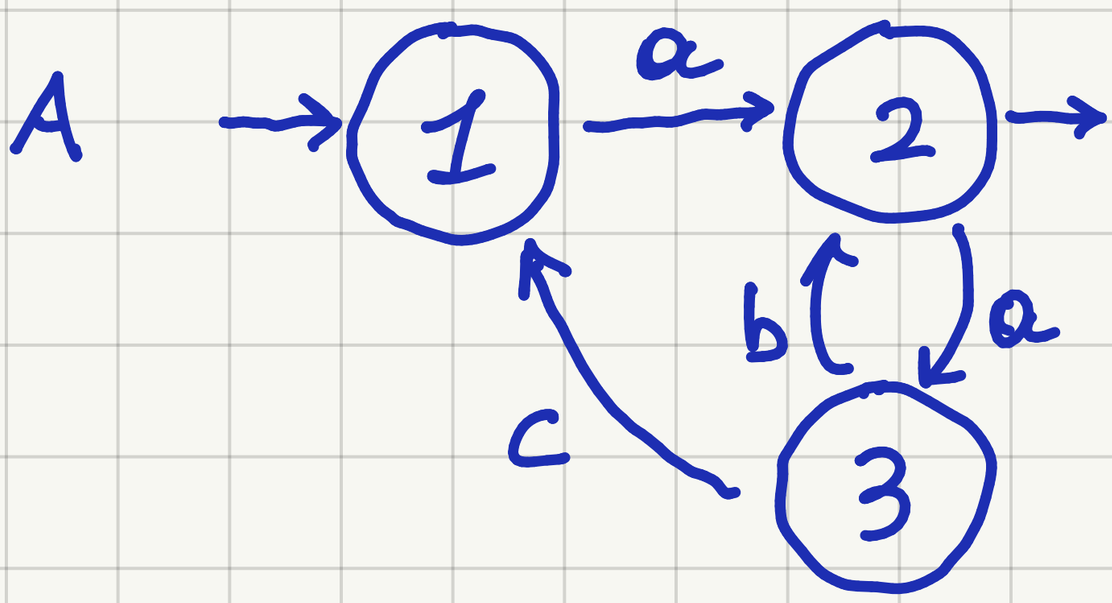
b) BERRY-SETHI

$R'=(a_1(a_2b_3)^*a_4c_5)^*a_6(a_7b_8)^*$

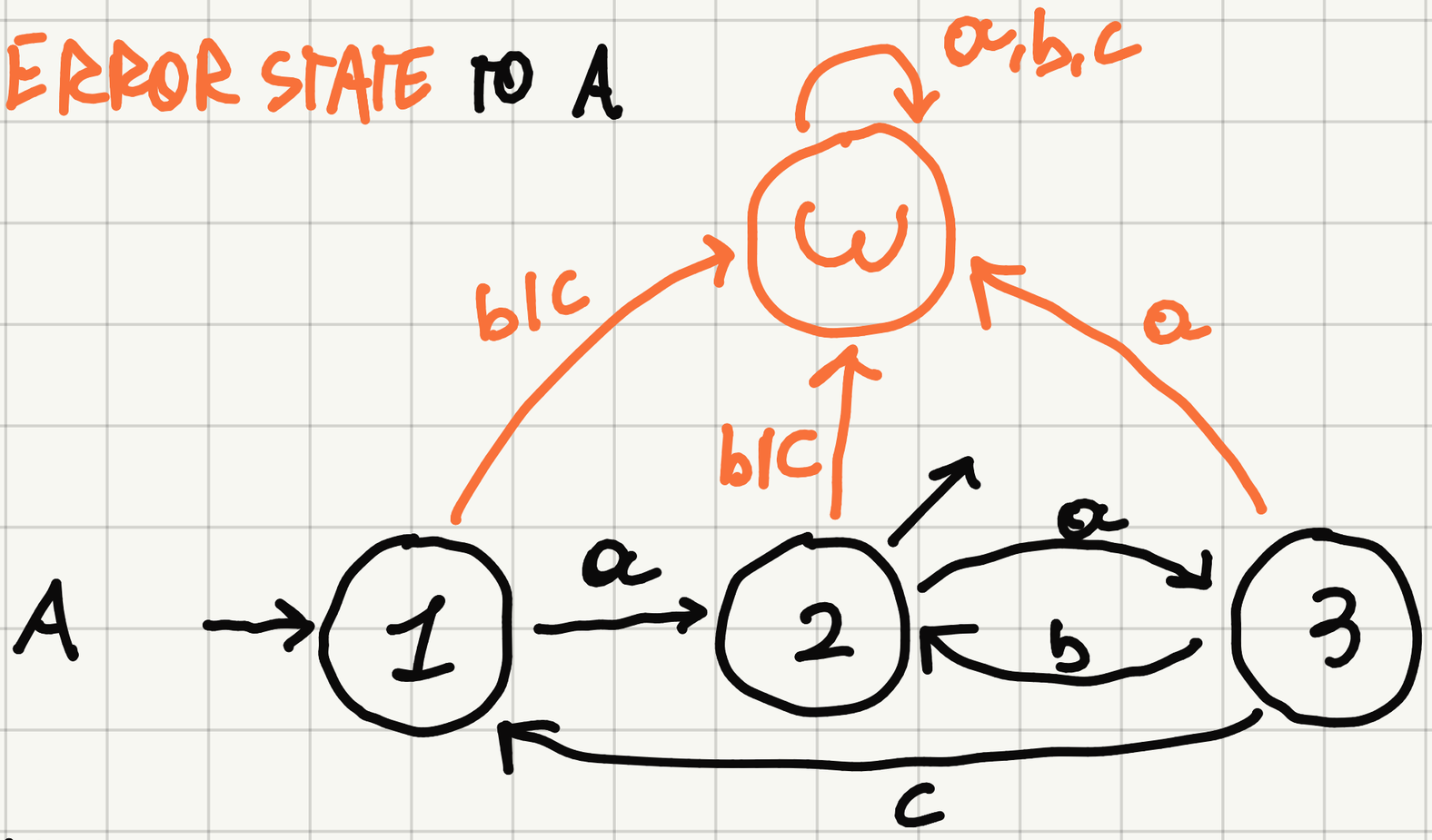
$initials=\{a_1,a_6\}$

Terminals	Followers
a_1, b_3	a_2, a_4
a_2	b_3
a_4	c_5
c_5	a_6, a_6
a_6, a_8	$a_7, \dashv$
a_7	a_8

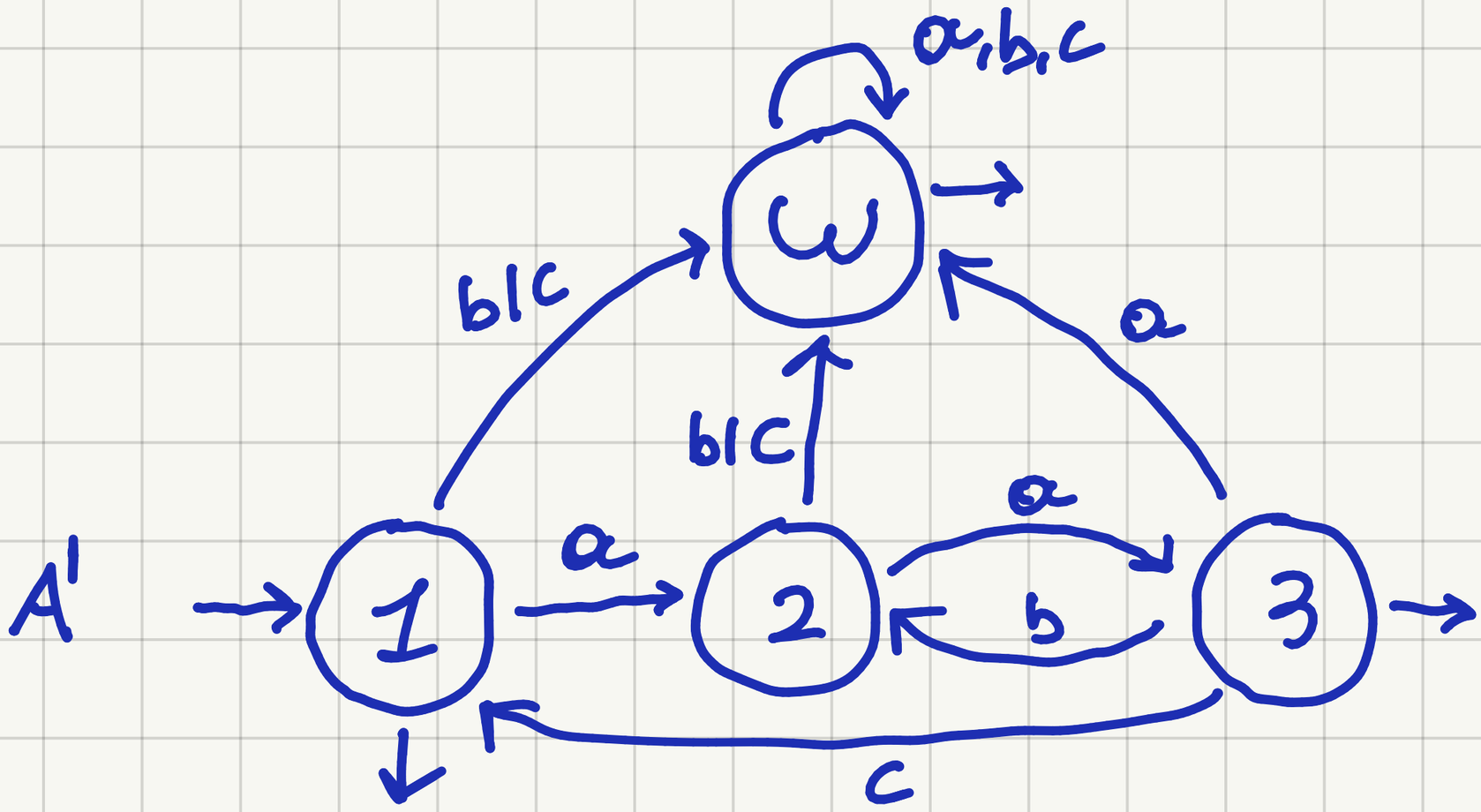




c) TO OBTAIN THE COMPLEMENT, WE FIRST ADD THE ERROR STATE TO A



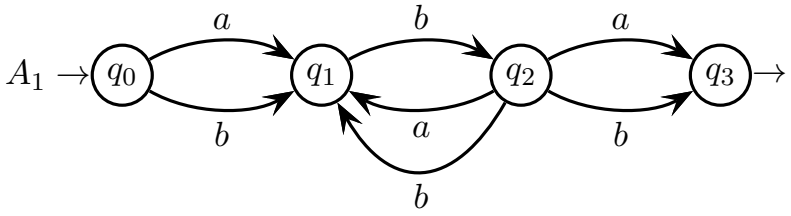
THEN, ALL THE FINAL STATES TURN NON-FINAL AND NICEVERSA



11 FEBRUARY 2014

1 Regular Expressions and Finite Automata 20%

1. Consider the finite automaton A_1 shown below:



and the following regular expression e_2:

$$e_2 = (\, b\; b\,)^*\; a$$

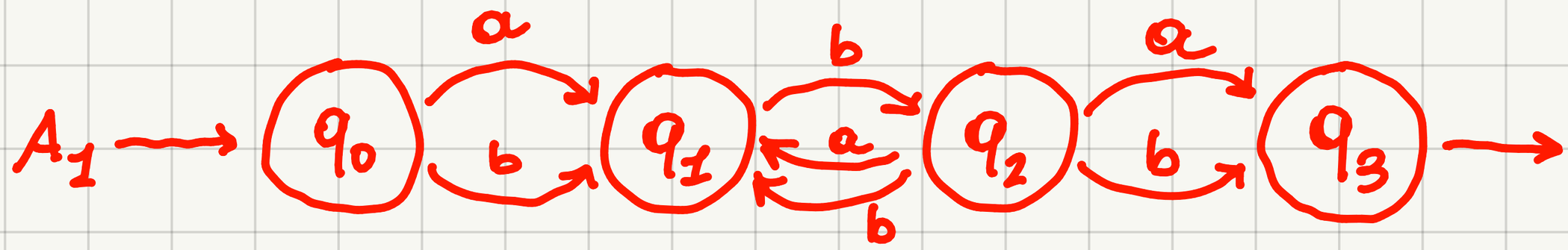
Answer the following questions, and explain the obtained results:

(a) By means of the *BMC* method (node elimination), write a regular expression e_1 such that the language equality $L(e_1) = L(A_1)$ holds. If possible, modify the regular expression e_1 to make it more compact.

(b) By means of the *BS* method (Berry-Sethi), build a deterministic finite automaton A_2 that recognizes language $L(e_2)$.

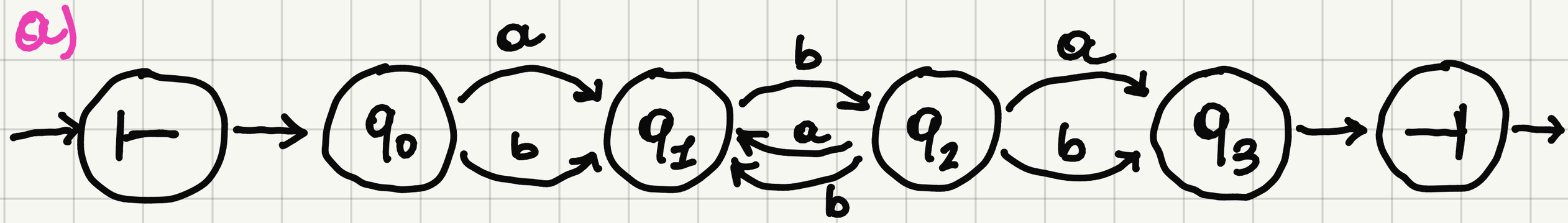
(c) By means of the automaton product construction (cartesian product), build a finite automaton A_3 that recognizes the intersection language $L(A_1) \cap L(A_2)$.

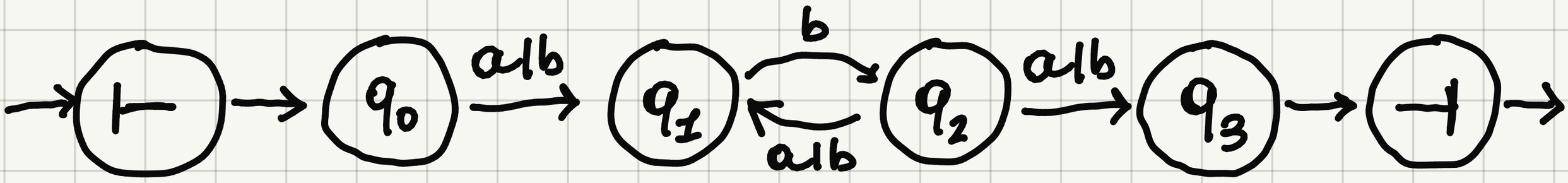
(d) If necessary, minimize the state number of the automaton A_3 found before, by means of a systematic method.

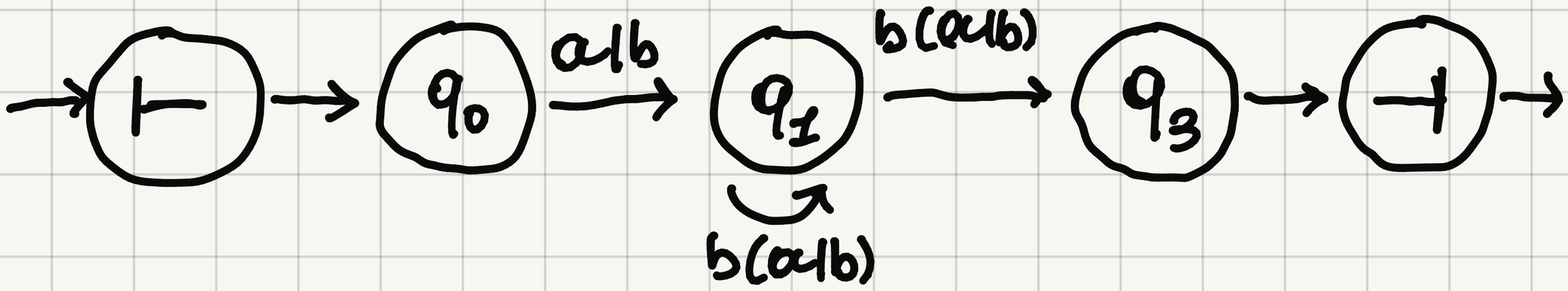


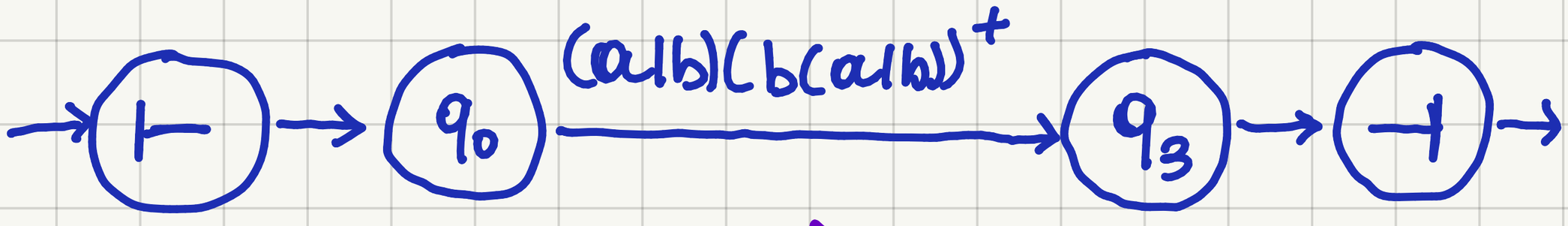
$e_2 = (bb)^* a$

a)









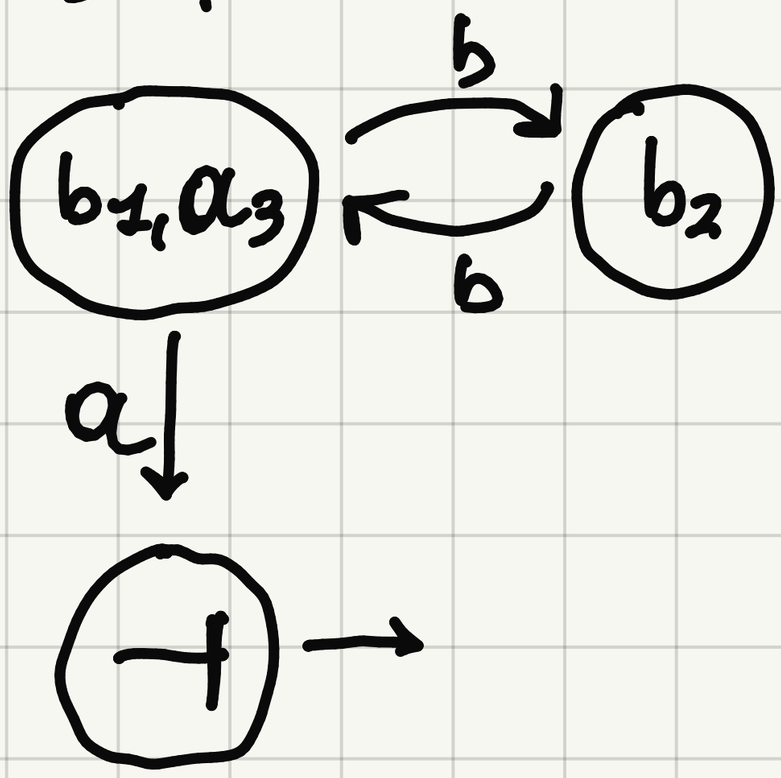
$e_1 = (a|b)(b(a|b))^+$

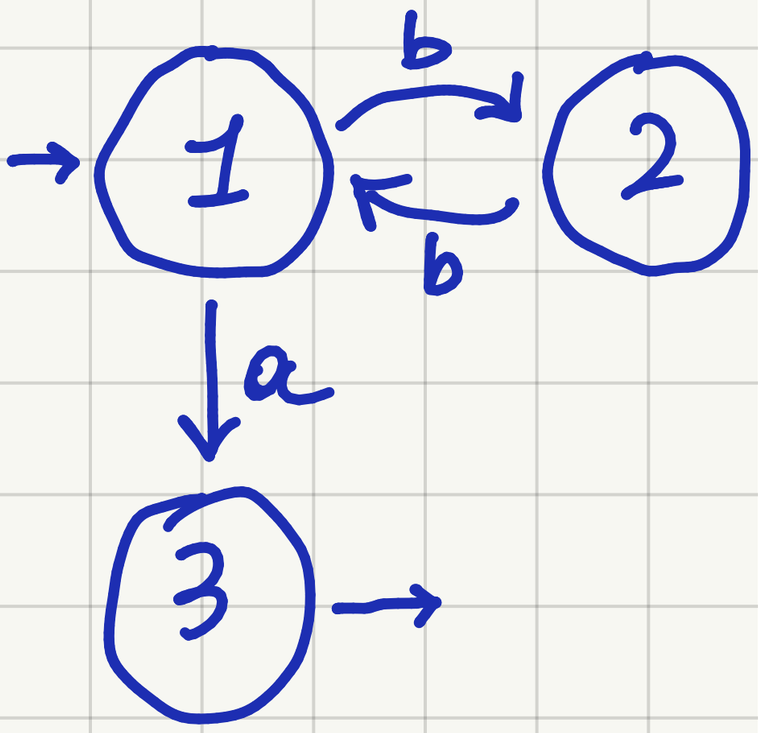
REMEMBER THAT
$(a|b)^+ = (a|b)(a|b)^*$

b) BERRY-SETHI

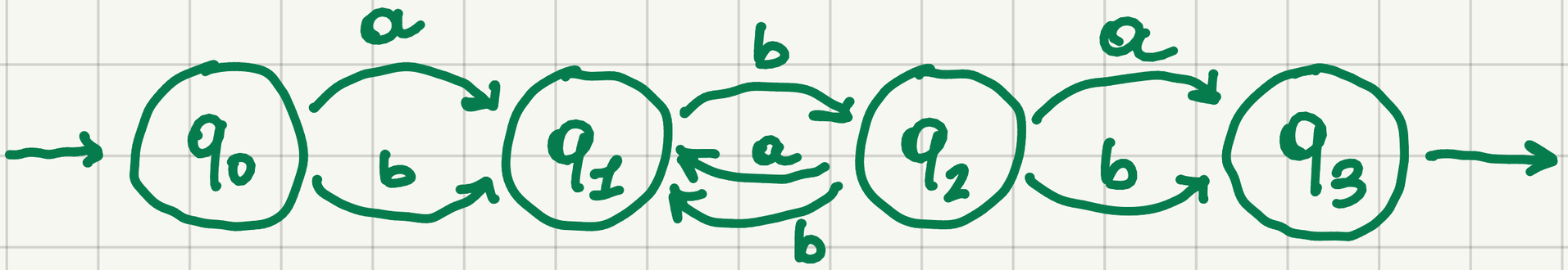
$Re_2 = (b_1 b_2)^* a_3$ $\quad initials = \{b_1, a_3\}$

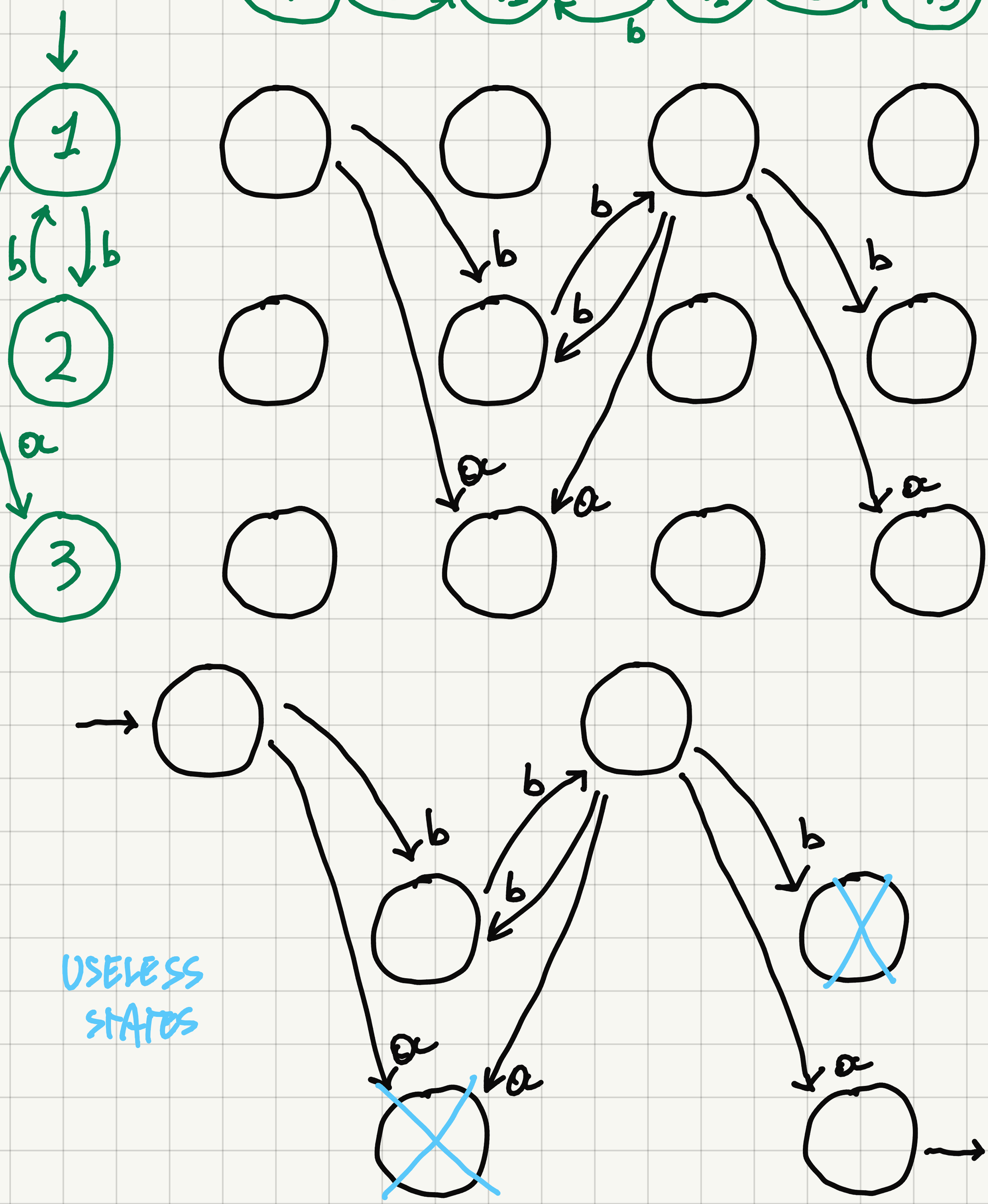
Terminals	Followers
b_1	b_2
b_2	b_1, a_3
a_3	⊣



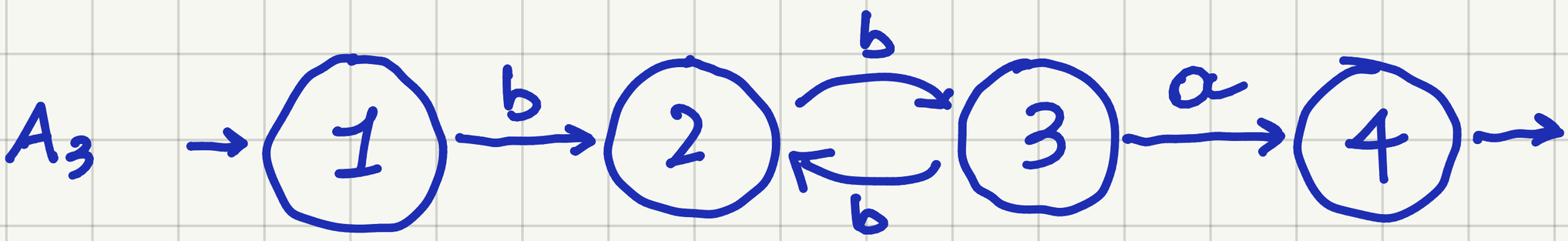


c) $L(A_1) \cap L(A_2)$





USELESS STATES



d)

	1	2	3	4
1				
2	X			
3	X	X		
4	X	X	X	

4 IS THE ONLY FINAL STATE
→ DISTINGUISHABLE

$$\begin{cases} \delta(2,a)=w \\ \delta(2,b)=3 \end{cases} \qquad \begin{cases} \delta(1,a)=w \\ \delta(1,b)=2 \end{cases}$$

$$\begin{cases} \delta(3,a)=4 \\ \delta(3,b)=2 \end{cases}$$

THEY ARE ALL DISTINGUISHABLE

⇒ A_3 IS IN MINIMAL FORM ALREADY

4 SEPTEMBER 2015

2 Free Grammars and Pushdown Automata 20%

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

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

Answer the following questions:

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

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

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

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

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

$\Sigma = \{a, b, c\}$ $L = \{ xcw^Rcy \mid x, y \in \{a, b\}^* \wedge w^R = yx \}$

a) TO PROVIDE AN IDEA, LET'S START BY STUDYING HOW w BEHAVE WITH SOME EXAMPLES OF x AND y

x	y	w^R	w
aa	ε	aa	aa
ε	ba	ba	ab
ba	a	aba	aba
babb	bb	bbbabb	bbabbb

$w = (xy)^R =$
$= y^R x^R$

$L = \underbrace{xcx^R}_{S} \underbrace{y^R cy}_{S'}$ REMINDS OF PALINDROMES, WELL-KNOWN

$A \to SS'$

$S \to aSa \mid bSb \mid c$

$S' \to aS'a \mid bS'b \mid c$

SINCE S AND S' ARE THE SAME, WE CAN SIMPLIFY

$$G \begin{cases} A \to SS \\ S \to aSa \mid bSb \mid c \end{cases}$$

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

AGAIN, LET'S START BY PROVIDING A GENERAL IDEA WITH SOME EXAMPLES

	$(ab)^* c \Sigma^* c (ab)^*$	L
acacacb	✗	✓
abcaacacab	✓	✗
abcbababacab	✓	✓

(acacacb: x = ε, x^R = ε, y = ε, y^R = b)

(abcaacacab: x = ab, x^R = ba, y = ab, y^R = ba)

(abcbababacab: x = ab, x^R = ba, y = ab, y^R = ba)

SO, WE CAN GENERALIZE BY WRITING $L' = \{ xcx^R \cdot y^R cy, : x, y \in \{a,b\}^* \}$

G':

$S \to AB$

$A \to abAba \mid c$

$B \to baBab \mid c$

c) $L'' = \overline{L'} \in REG$?

L'' STILL RECOGNIZE PALINDROMES, EVEN IF WITH A DIFFERENT

STRUCTURE FROM L' $\Rightarrow$ IT NEEDS A PUSH-DOWN AUTOMATON

$\Rightarrow$ IT IS NOT REGULAR, SINCE A FINITE STATE AUTOMATON IS NOT ENOUGH

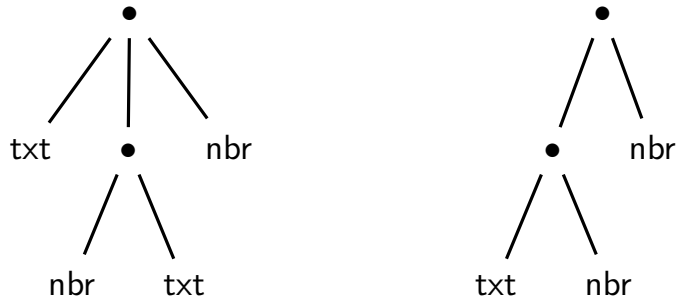
2. A marked language, inspired to *XML*, is used to encode trees in textual form. Such textual representations are subject to the following syntax constraints.

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

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

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

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



   Answer the following questions:

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

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

Q1) <LANG> → <FORESTS>

<FORESTS> → (<TREES> ',')* <TREES>

<TREE> → <NODE1>

<NODE1> → <TERNARY> | <NODE2>

<NODE2> → <BINARY> | <LEAF>

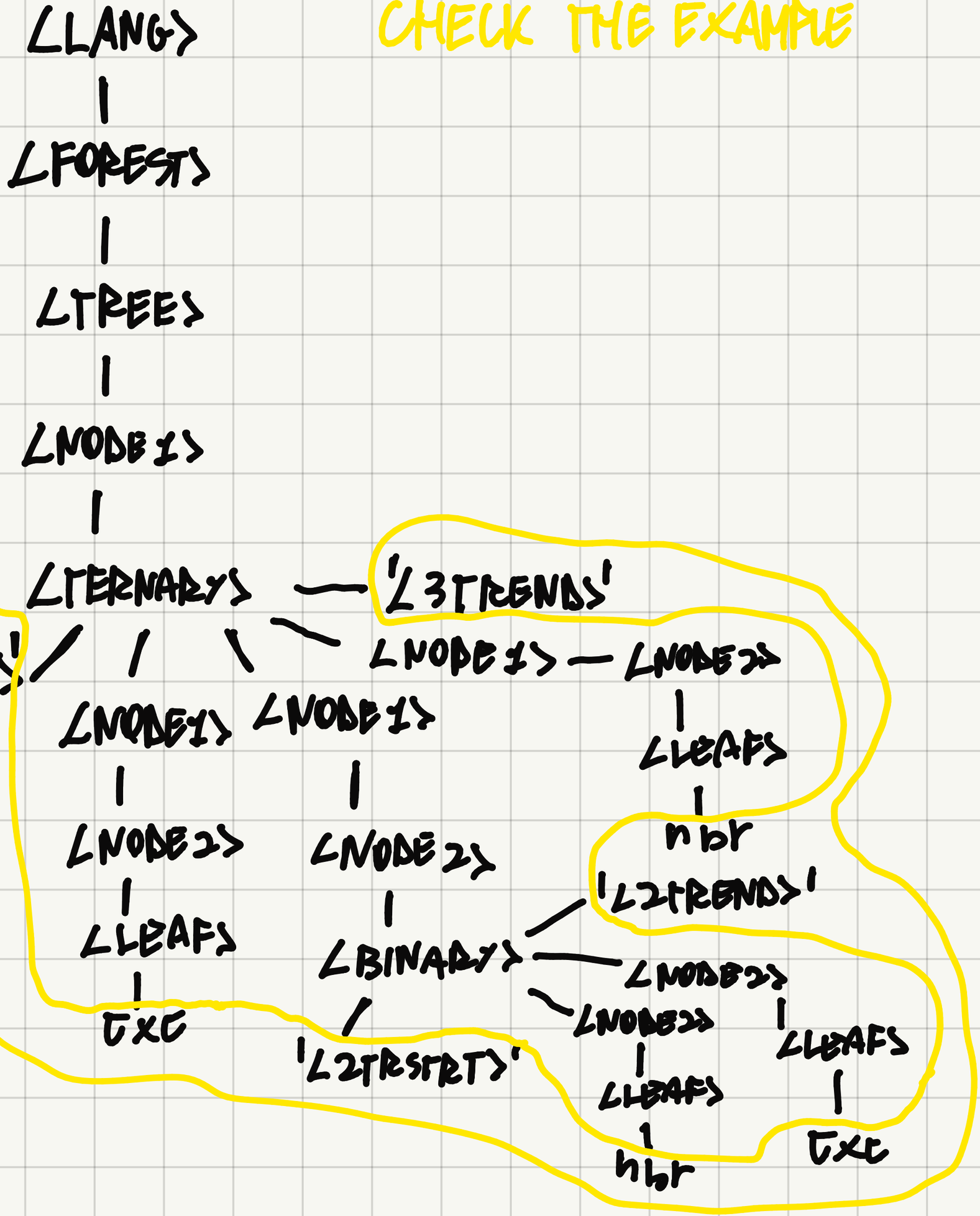
} AVOIDS AMBIGUITY

<TERNARY> → '<3TRSTRT>' <NODE1>{3} '<3TRENDS>'

<BINARY> → '<2TRSTRT>' <NODE1>{2} '<2TRENDS>'

<LEAF> → nbr | txt

CHECK THE EXAMPLE



b) <N_NODES> → '<>3TRSTR>' <NODE1>^{3} <NODE1>$^{+}$ |

'<>3TRENDS>'

MODIFY <NODE1>

<NODE1> → <TERNARY> | <NODE2> | <N_NODE>

18 JANUARY 2018

2 Free Grammars and Pushdown Automata 20%

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

$$L = \{ a^m b^n \mid \quad m \geq n \geq 0 \quad \wedge \quad m - n \text{ is even} \}$$

Sample valid strings:

$$\varepsilon,\ a^2,\ a\,b,\ a^3\,b\ \in L$$

Sample invalid strings:

$$b,\ a^2\,b\ \notin L$$

Answer the following questions:

(a) Write an *ambiguous BNF* grammar G_1 that generates language L.

(b) Draw at least two syntax trees for string $a\ a\ a\ b$ according to grammar G_1.

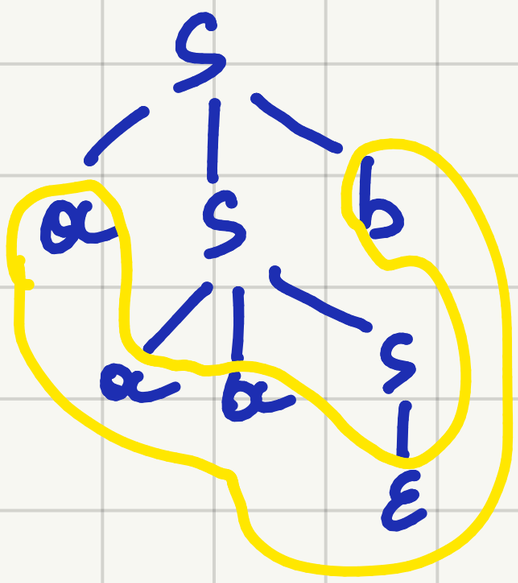
(c) Write an *unambiguous BNF* grammar G_2 that generates language L.

(d) Draw the (unique) syntax tree for string $a\ a\ a\ b$ according to grammar G_2.

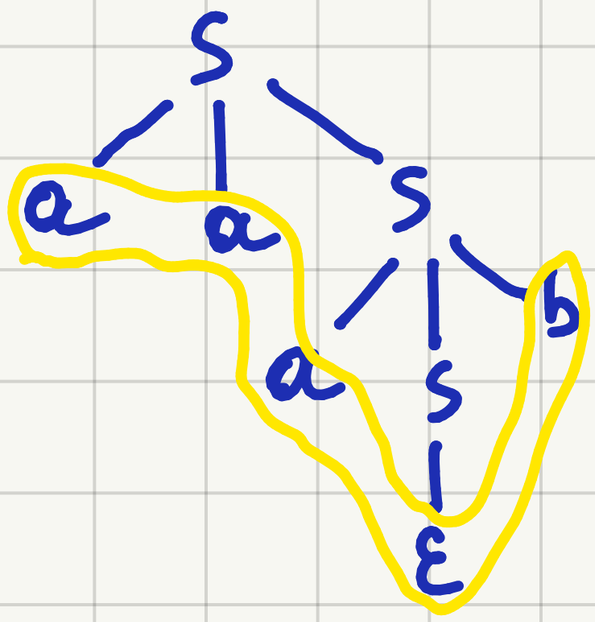
$L = \{a^m b^n \mid m \geq n \geq 0 \wedge m-n \text{ EVEN}\}$ $\quad \Sigma = \{a, b\}$

a) $G_1 \{ S \to \varepsilon \mid aaS \mid aSb$

b) • $S \to aaS \to aaaaSb \to aaaa\varepsilon b \to aaaab$



• $S \to aSb \to aaaaSb \to aaaa\varepsilon b \to aaaab$



c) THE IDEA IS TO FORCE FIRST TO PRINT ALL THE a'S AND THEN THE b'S, OR VICEVERSA

• $G_2 \begin{cases} S \to \varepsilon \mid aSb \mid X \\ X \to aaXb \end{cases}$

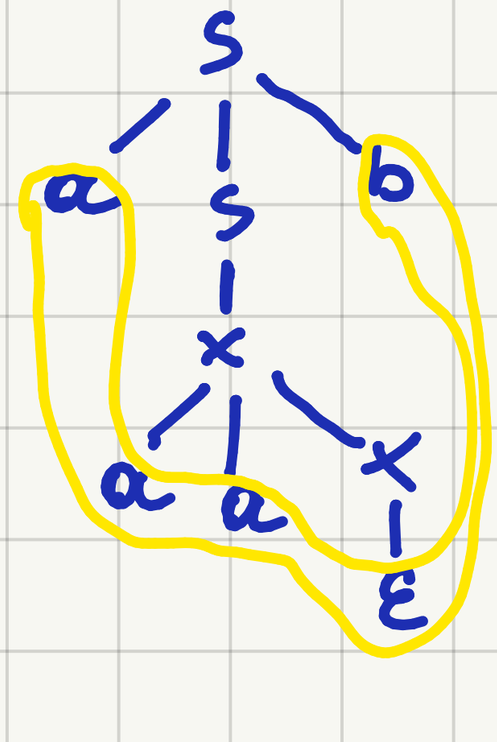
• $G_2 \begin{cases} S \to \varepsilon \mid aaaS \mid X \\ X \to aXa \mid \varepsilon \end{cases}$

BOTH SOLUTIONS ARE VALID

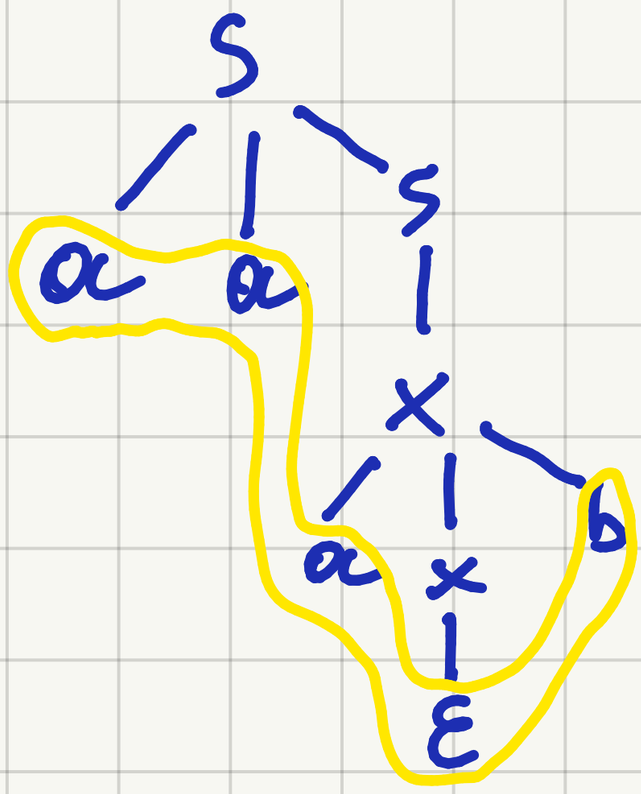
IN THE EXAM, IT SUFFICES TO PROVIDE ONLY ONE OF THEM

d)

- $S \to aSb \to aXb \to aaaXb \to aaa\varepsilon b \to aaab$



- $S \to aaS \to aaX \to aaaXb \to aaa\varepsilon b \to aaab$



2. Consider the fragment of a programming language that consists of assignments of two types of arithmetic expression, one simple and one complex, to variables. This fragmentary language is informally specified as follows:

   - Each phrase of the language is a non-empty list of assignments separated by semicolon "`;`".
   - An assignment consists of a variable name, an assignment operator and an arithmetic expression.
   - There are two assignment operators: "`=`" and "`:=`". The former wants a *simple* expression on the right, and the latter wants a *complex* one instead.
   - A *simple* arithmetic expression has at most two levels, without parentheses. It can use addition "`+`" and multiplication "`*`", of variables and constants. The associativity of addition and multiplication is unspecified. Precedence is uncommon: multiplication has lower priority and addition higher.
   - A *complex* arithmetic expression has arbitrarily many levels, with parenthesized subexpressions. It can use addition "`+`" and multiplication "`*`", of variables and constants. Addition is left-associative and multiplication is right-associative. Precedence is as usual: multiplication has higher priority and addition lower.
   - A variable identifier is alphanumerical (for simplicity use only lowercase letters), must have at least one heading letter and may freely contain underscores "`_`".
   - A constant is a decimal integer and may have a unary minus sign "`-`".

   For any unspecified minor detail, see the example below, and you may also follow the usual conventions, e.g., those of the C language. Example:

```
a = b + c + 2 * d;

e := f + g + h * 3;

a1 = a + -5 * c * d + b;

b34_c := a * (c + d) * -040;
```

   Answer the following questions:

   (a) Write a non-ambiguous grammar, in general of type *EBNF*, that models the described fragmentary language.

   (b) (optional) In order to (partially) test the behaviour of your grammar, separately draw the two syntax subtrees of the first two sample assignments. You may condense the trees by schematizing variables and constants with a terminal "`a`".

---

a)

⟨LANG⟩ → (⟨ASSIGN⟩ ';')+

⟨ASSIGN⟩ → ⟨S_ASSIGN⟩ | ⟨C_ASSIGN⟩

⟨S_ASSIGN⟩ → ⟨VAR⟩ '=' ⟨S_EXPR⟩

⟨C_ASSIGN⟩ → ⟨VAR⟩ ':=' ⟨C_EXPR⟩

⟨S_EXPR⟩ → ⟨S_FACTOR⟩ ('*' ⟨S_FACTOR⟩)*

⟨S_FACTOR⟩ → ⟨ATOM⟩ ('+' ⟨ATOM⟩)*

NO ASSOCIATIVITY
PRECEDENCE OF +

⟨C_EXPR⟩ → ⟨C_EXPR⟩ '+' ⟨C_TERM⟩ | ⟨C_TERM⟩

LEFT ASSOCIATIVITY

⟨C_TERM⟩ → ⟨C_FACTOR⟩ '*' ⟨C_TERM⟩ | ⟨C_FACTOR⟩

RIGHT ASSOCIATIVITY

⟨C_FACTOR⟩ → ⟨ATOM⟩ | ('C' ⟨C_EXPR⟩ ')')

⟨ATOM⟩ → ⟨VAR⟩ | ⟨CST⟩

⟨VAR⟩ → ⟨CHAR⟩ (⟨ALNUM⟩ | '_')*

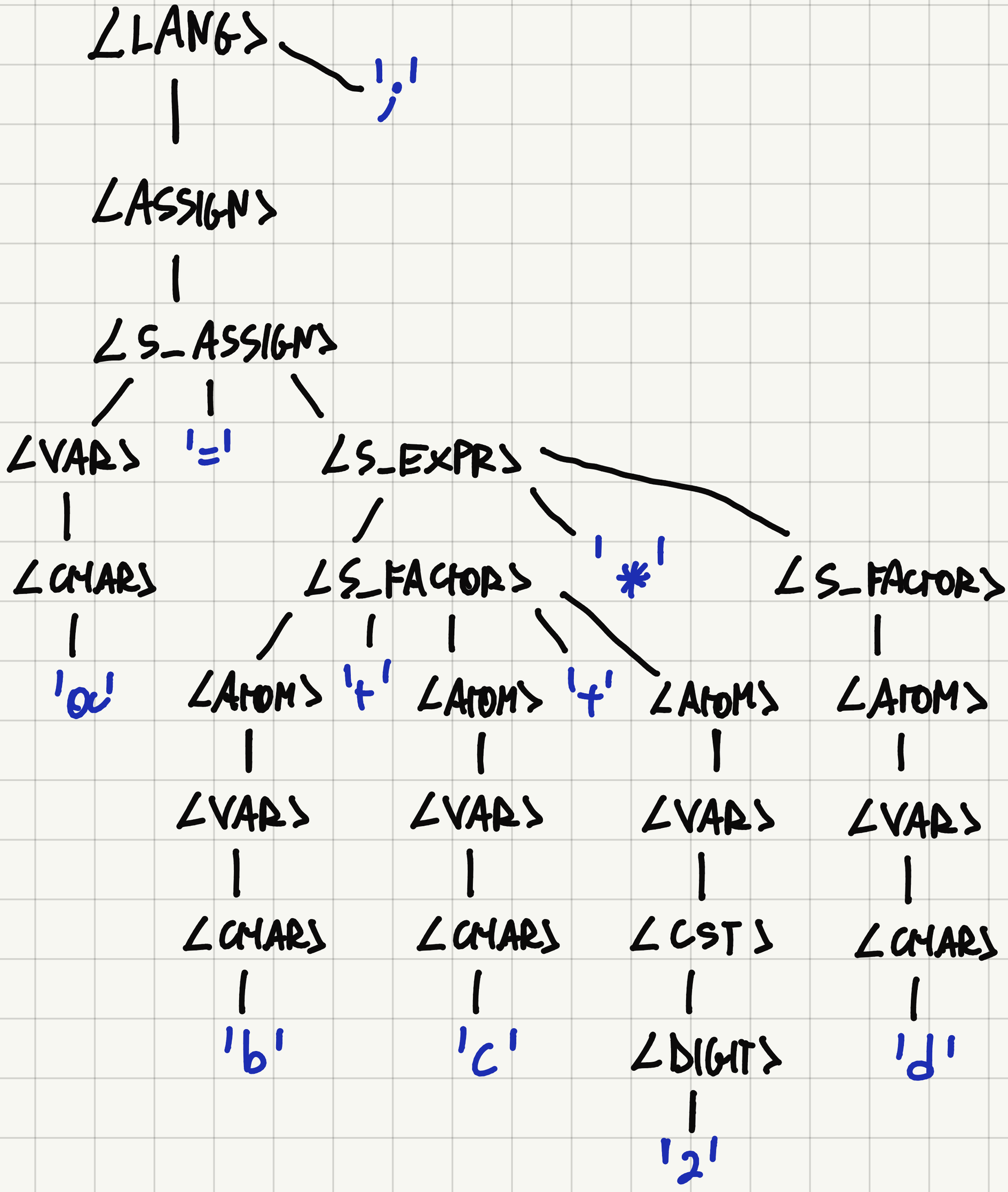
⟨CHAR⟩ → ['a'-'z']

⟨ALNUM⟩ → ⟨CHAR⟩ | ⟨DIGIT⟩

⟨CST⟩ → ['-'] ⟨DIGIT⟩+

⟨DIGIT⟩ → ['0'-'9']

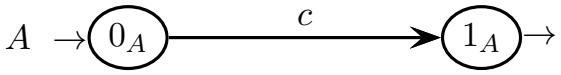
b) a = b + c + 2 * d

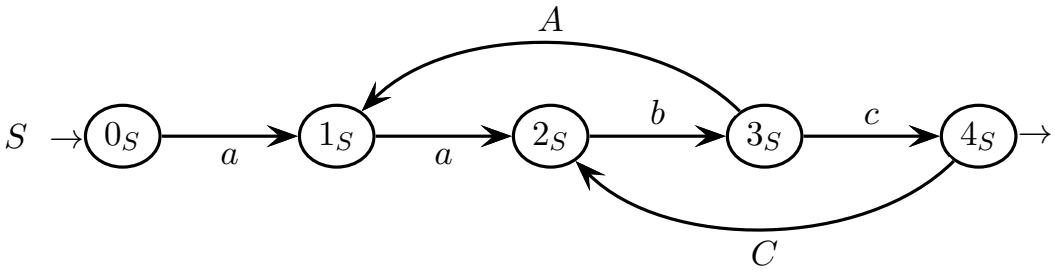

<LANG>
;
<ASSIGN>
<S_ASSIGN>
<VAR>
'='
<S_EXPR>
<CHAR>
<S_FACTOR>
'*'
<S_FACTOR>
'a'
<ATOM>
'+'
<ATOM>
'+'
<ATOM>
<ATOM>
<VAR>
<VAR>
<VAR>
<VAR>
<CHAR>
<CHAR>
<CST>
<CHAR>
'b'
'c'
<DIGIT>
'd'
'2'

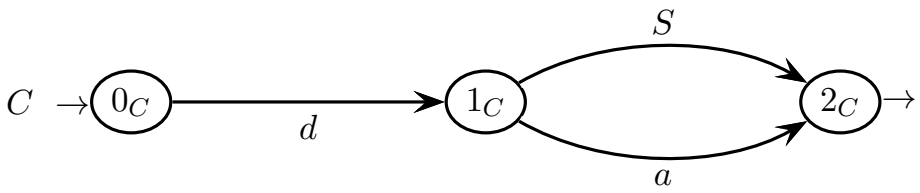
3 MARCH 2015

3 Syntax Analysis and Parsing Methodologies 20%

1. Consider the following machine net (axiom S), over the four-letter alphabet $\{\, a,\ b,\ c,\ d \,\}$:







Answer the following questions:

(a) Draw the complete pilot graph of the machine net and say if the net is of type $ELR\,(1)$ or not (explain your answer). Is the $ELL\,(1)$ condition satisfied by the pilot or not (explain your answer) ? Use the space left on the next pages.

(b) Write the guide sets on each call and exit arrow of the machine net, and using these guide sets (as well as those on the terminal shift arcs) say if the net is of type $ELL\,(1)$ or not (explain your answer). Use the net on the next pages.

(c) (optional) Examine the net and say if it is of type $ELL\,(k)$ for some $k \geq 2$ or not. This may require to find a few guide sets of a length greater than one, and to reason using them. Please answer concisely but rigorously. If you wish, you can use the net on the next pages.

---

a) HOW TO BUILD THE PILOT AUTOMATON?

1 - START FROM AXIOM AND SPECIFY THE LOOK-AHEAD.

AT THE BEGINNING, STRUCTURE WILL ALWAYS BE LIKE

(BEGINNING STATE OF THE AXIOM END OF STRING)

2 - COMPUTE THE CLOSURE

- LOOK AT THE STATE OPPOSING THE PHASE PART AND SEE IF

WE CAN GO TO SOME ARCS GOING TO SOME NON-TERMINALS

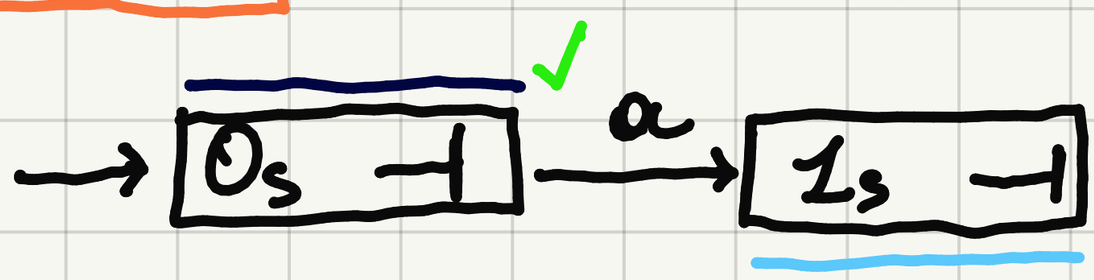
BASE PART

IN GENERAL

0_S HAS ONLY 1 OUTGOING ARC (α), WHICH

CLOSURE

IS A TERMINAL ⇒ THE CLOSURE IS ITSELF

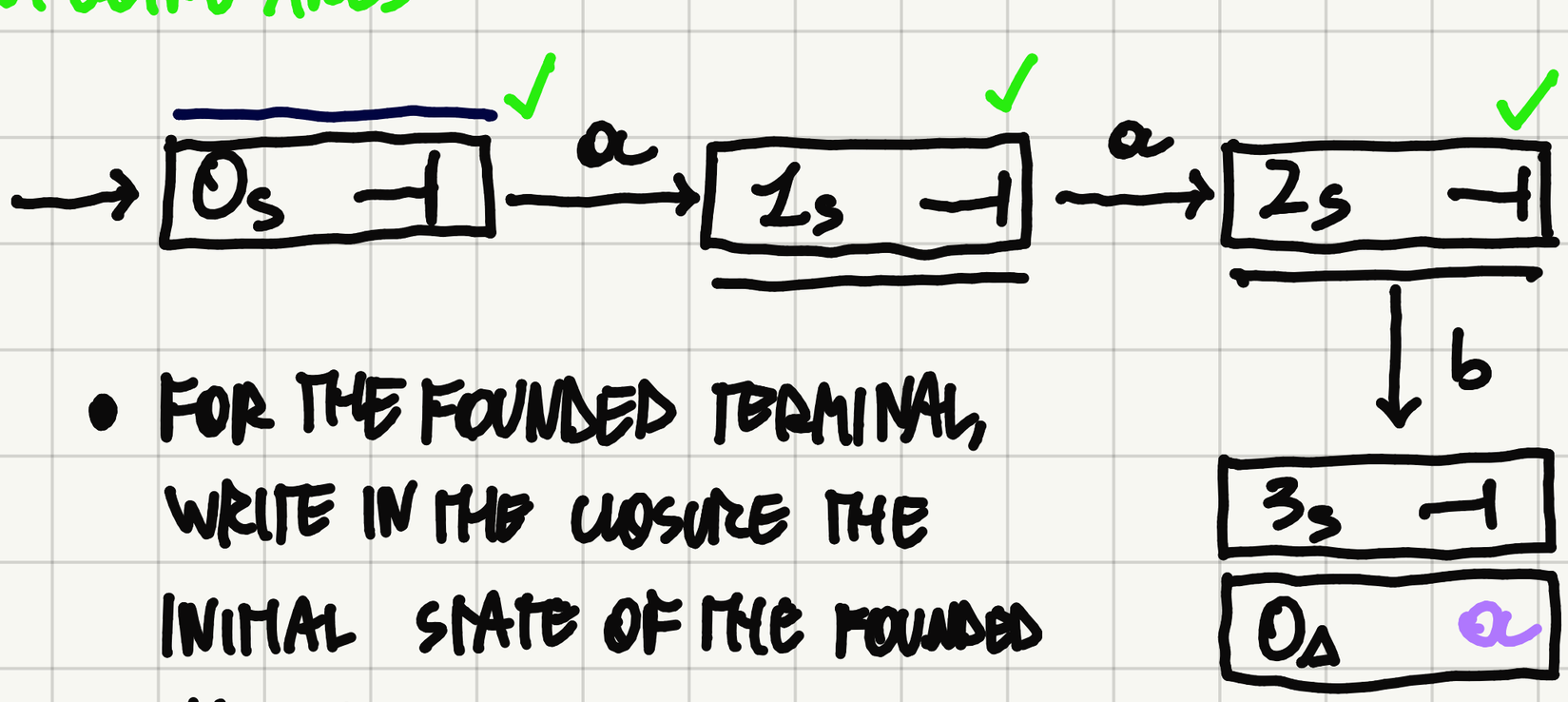


AGAIN $1_S \xrightarrow{a}$ (NON TERMINAL)

SUGGESTION: WHEN FINISHED TO COMPUTE THE MACRO-STATE, MARK IT.

ESPECIALLY WHEN COMES TO MULTIPLE STATE IN THE BASE AND MULTIPLE

OUTGOING ARCS



AFTER READING A, S GOES BACK TO 1_S. IT CAN ONLY COUNTER THE NON-TERMINAL a

- FOR THE FOUNDED TERMINAL, WRITE IN THE CLOSURE THE INIMAL STATE OF THE FOUNDED MACHINE