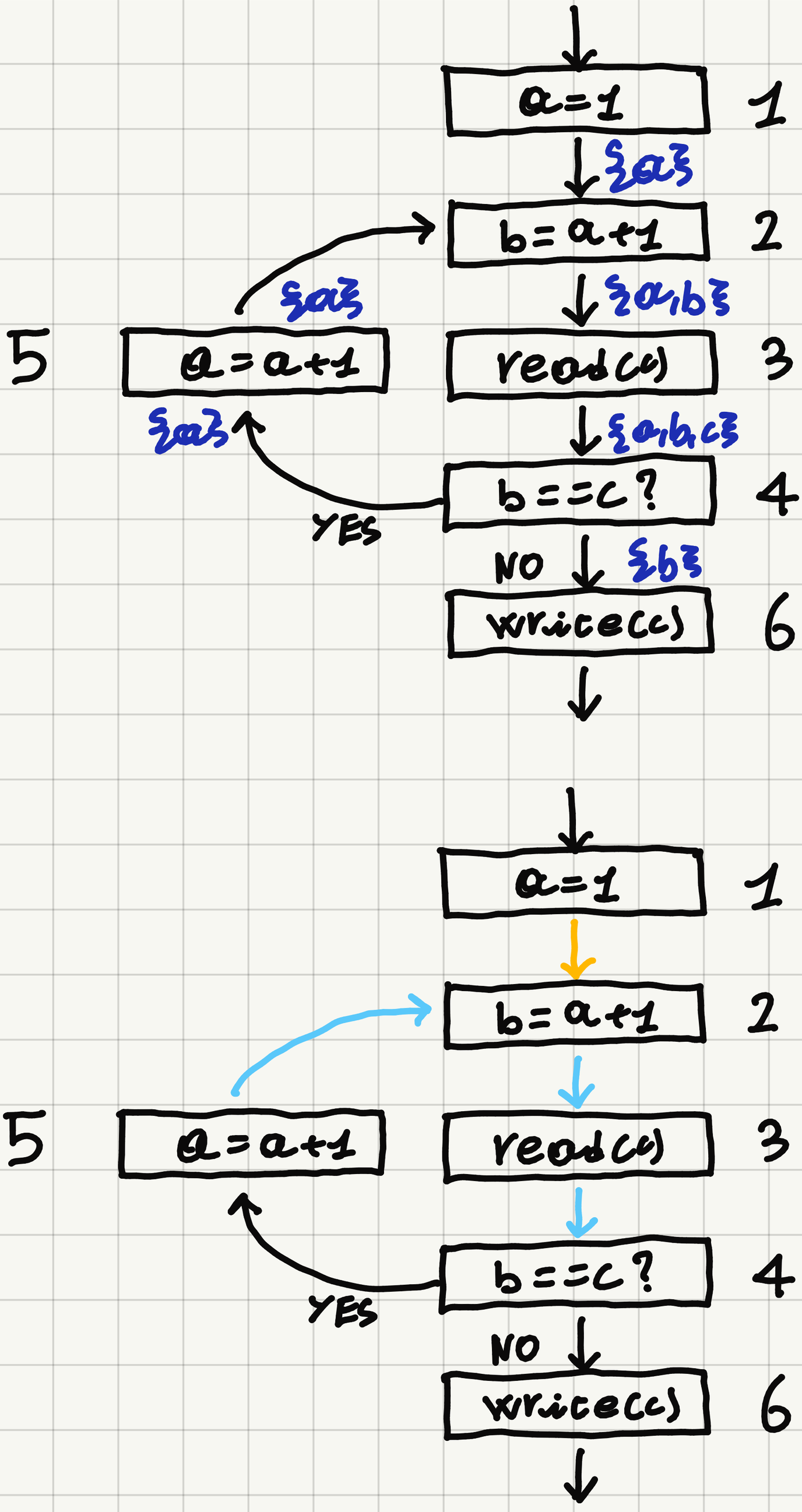
#	in	OUT	OUT	in	OUTPUT	INPUT
1	$\emptyset \cup (OUTPUT(3) \setminus \{a\})$	$in(2)$	a	$\emptyset$	a	$\emptyset$
2	$\{a\} \cup (OUTPUT(2) \setminus \{b\})$	$in(3)$	b	a	a,b	a
3	$OUTPUT(3) \setminus \{c\}$	$in(4)$	a,b,c	a,b	a,b,c	a,b
4	$\{b,c\} \cup (OUTPUT(4))$	$in(5) \cup in(6)$	a,b	a,b,c	a,b	a,b,c
5	$\{a\} \cup (OUTPUT(5) \setminus \{a\})$	$in(3)$	a	a	a	a
6	$\{b\} \cup (OUTPUT(6))$	$\emptyset$	$\emptyset$	b	$\emptyset$	b

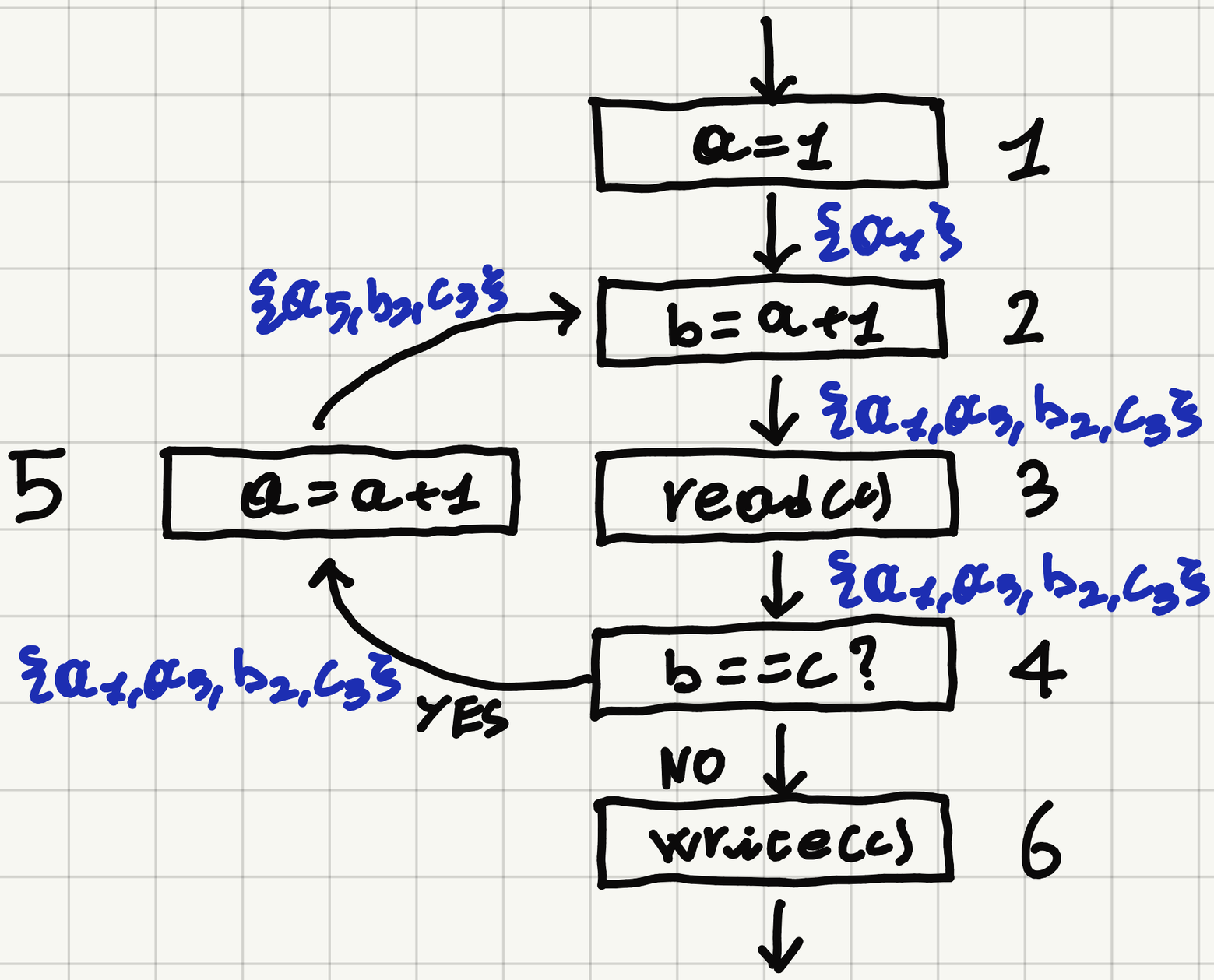
#	in	OUT	OUT	in	OUTPUT
1	$\emptyset \cup (OUTPUT(3) \setminus \{a\})$	$in(2)$	a	$\emptyset$	a
2	$\{a\} \cup (OUTPUT(2) \setminus \{b\})$	$in(3)$	a,b	a	a,b
3	$OUTPUT(3) \setminus \{c\}$	$in(4)$	a,b,c	a,b	a,b,c
4	$\{b,c\} \cup (OUTPUT(4))$	$in(5) \cup in(6)$	a,b	a,b,c	a,b
5	$\{a\} \cup (OUTPUT(5) \setminus \{a\})$	$in(3)$	a	a	a
6	$\{b\} \cup (OUTPUT(6))$	$\emptyset$	$\emptyset$	b	$\emptyset$

Now, let's place the results in the CFG



b)

$$IN \begin{cases} \divideontimes 1 = \emptyset \\ \divideontimes 2 = a_1, a_5, b_2, c_3 \\ \divideontimes 3 = a_1, a_5, b_2, c_3 \\ \divideontimes 4 = a_1, a_5, b_2, c_3 \\ \divideontimes 5 = a_1, a_5, b_2, c_3 \end{cases}$$



c) $sup(p) = \{a_q \mid a \in def(p) \wedge a \in def(q) \wedge q \neq p\}$

INITIAL NODE $in(1) = \emptyset$

OTHER NODES $OUT(p) = def(p) \cup (in(p) \setminus sup(p))$

$$in(p) = \bigcup_{\forall q \in pred(p)} OUT(q)$$

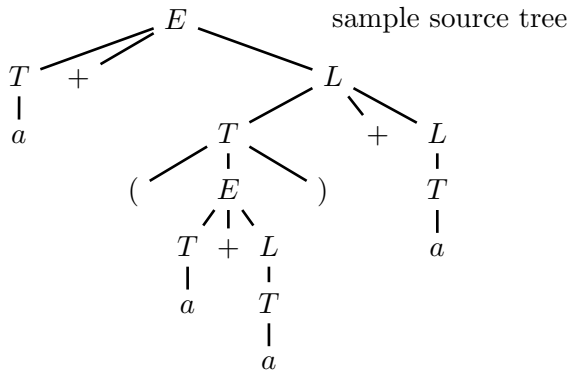
- $in(1) = \emptyset$ $\quad OUT(1) = \{a_1\} \cup (in(1) - \{a_3\})$
- $in(2) = OUT(1) \cup OUT(5)$ $\quad OUT(2) = \{b_2\} \cup in(2)$
- $in(3) = OUT(2)$ $\quad OUT(3) = \{c_3\} \cup in(4)$
- $in(4) = OUT(3)$ $\quad OUT(4) = in(4)$
- $in(5) = OUT(4)$ $\quad OUT(5) = \{a_5\} \cup (in(5) - \{a_1\})$
- $in(6) = OUT(4)$ $\quad OUT(6) = in(6)$

30 JANUARY 2019

4 Language Translation and Semantic Analysis 20%

1. Consider the source grammar G_s below, for the arithmetic expressions with infix addition $+$, round brackets (subexpressions), and variables schematized by terminal a (axiom E):

$$G_s \begin{cases} E \to T + L \\ L \to T + L \\ L \to T \\ T \to (E) \\ T \to a \end{cases}$$



Sample source strings are: $a+a$, $a+a+a$, $a+(a+a)+a$ (syntax tree above right).

Consider a translation that converts addition into prefix form with multiple addends (possibly more than two). The idea is to convert $a+a+a$ into $+3\ a\ a\ a$, for instance, where number 3 specifies the actual number of addends.

Here we want to design a syntactic translation function τ that produces such a multiple addend prefix form, though it does not compute the number of addends. Instead, it outputs a placeholder schematized by terminal n, e.g., $\tau\,(a+a+a) = +n\ a\ a\ a$. A semantic translation, *not to be considered here*, will compute the actual number of addends and will substitute such a number to the placeholder n.

See these examples (the actual number of addends is shown below n as a comment):

$$\begin{aligned} \tau\,(a+a) &= +\underset{2}{n}\ a\ a \\ \tau\,(a+a+a) &= +\underset{3}{n}\ a\ a\ a \\ \tau\,(a+(a+a)+a) &= +\underset{3}{n}\ a\ +\ \underset{2}{n}\ a\ a\ a \qquad \text{– translation } \tau \text{ of the example} \end{aligned}$$

Answer the following questions:

(a) Write a translation scheme G_τ that defines the syntactic translation function τ described above. Do not change the source grammar G_s. Test scheme G_τ by drawing the syntax translation tree of the sample translation τ above.

(b) Write a translation scheme $G_{\tau'}$ for an optimized syntactic translation τ' that does not output the placeholder n if addition has only two operands, as follows:

$$\tau'\,(a+(a+a)+a) = +\underset{3}{n}\ a\ +\ a\ a\ a \qquad \text{– sample translation } \tau'$$

For τ' you may change the source grammar G_s, if necessary. Test scheme $G_{\tau'}$ by drawing the syntax translation tree of the sample translation τ' above.

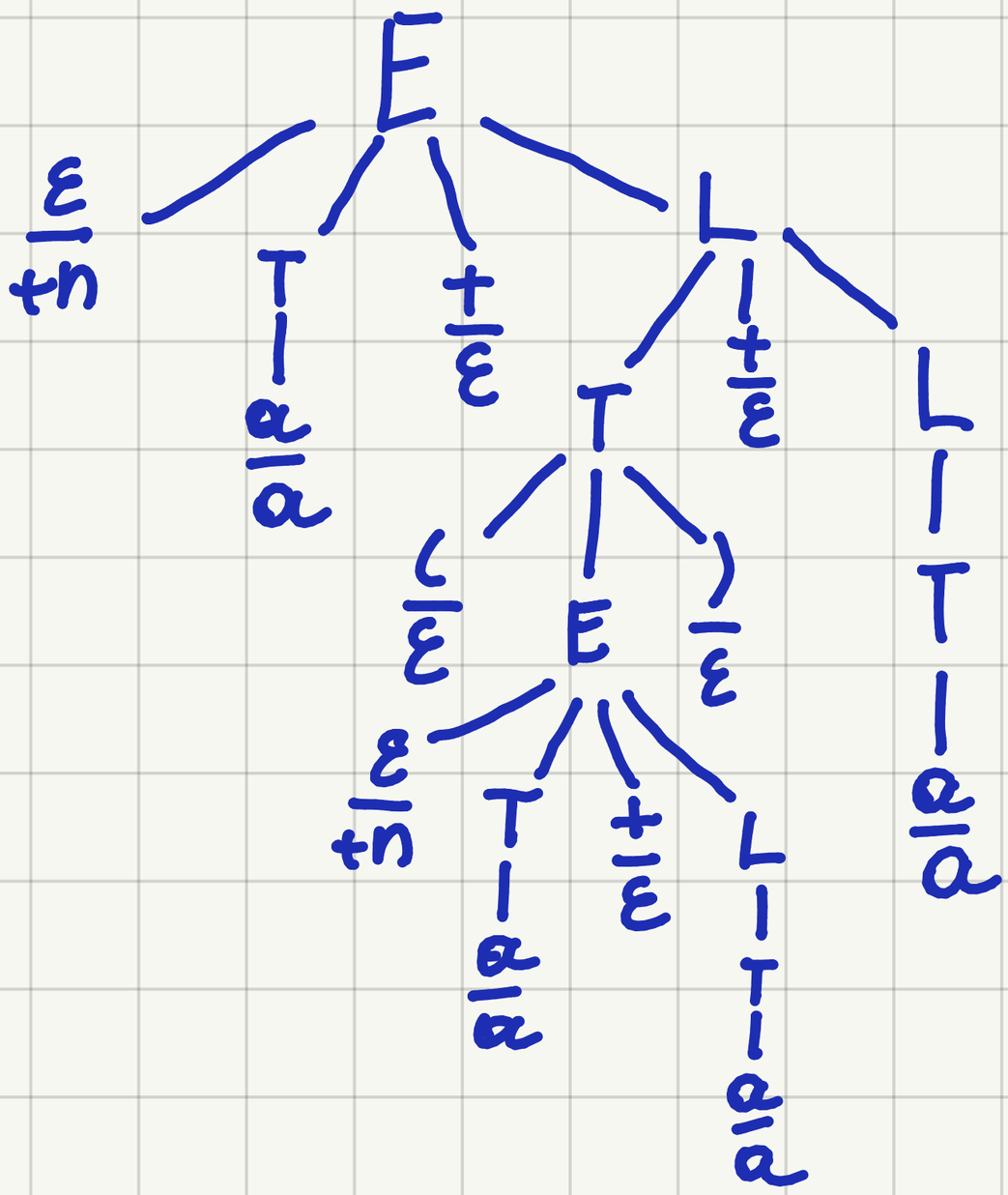
(c) (optional) Determine whether translations τ and τ' can be computed deterministically by an $ELL(k)$ parser with write actions, for suitable values of $k \geq 1$, and motivate your answers (yes or no).

a)

G_d:
- $E \to +n\ T\ L$
- $L \to T\ L$
- $L \to T$
- $T \to E$
- $T \to a$

G_{ε}:
- $E \to \underset{+n}{\varepsilon}\ T\ \underset{\varepsilon}{+}\ L$
- $L \to T\ \underset{\varepsilon}{\pm}\ L$
- $L \to T$
- $T \to \underset{\varepsilon}{(}\ E\ \underset{\varepsilon}{)}$
- $T \to a$

IN GENERAL WHAT TO REMOVE REPLACEMENT



CASE 2 a SUMMED

b)

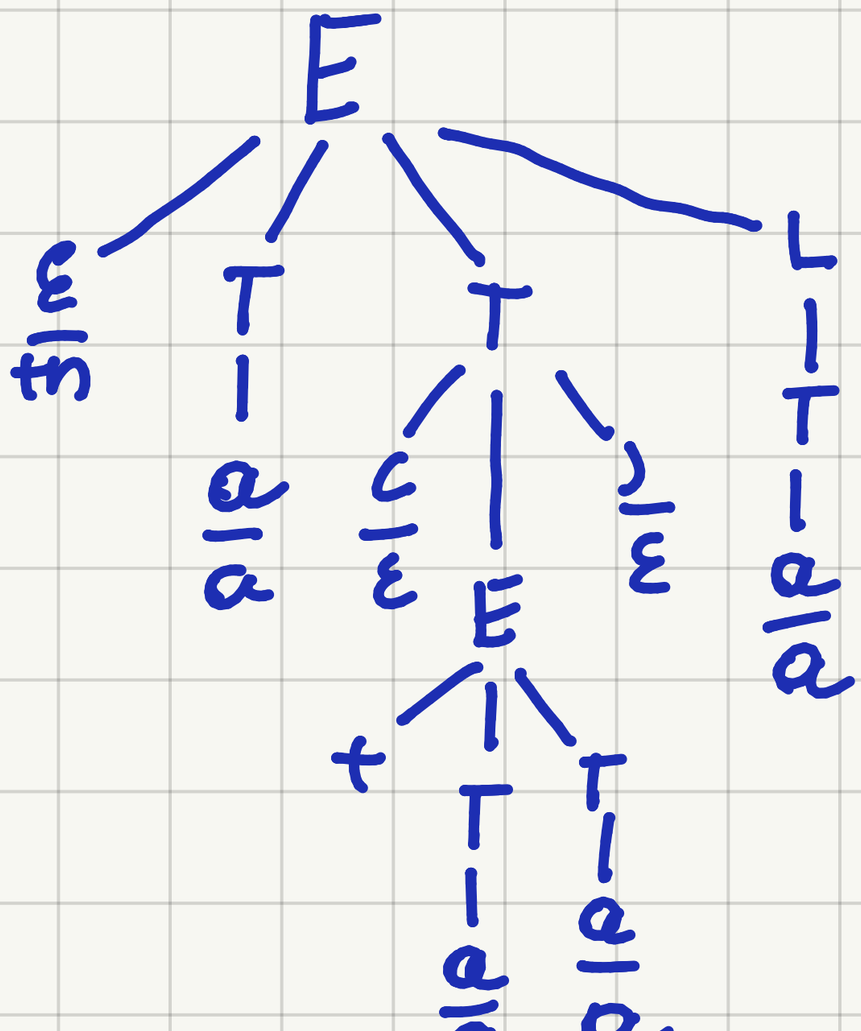
G_3:
- $E \to T+L$
- $L \to T+L$
- $L \to T$
- $T \to (E)$
- $T \to a$

$\Rightarrow$

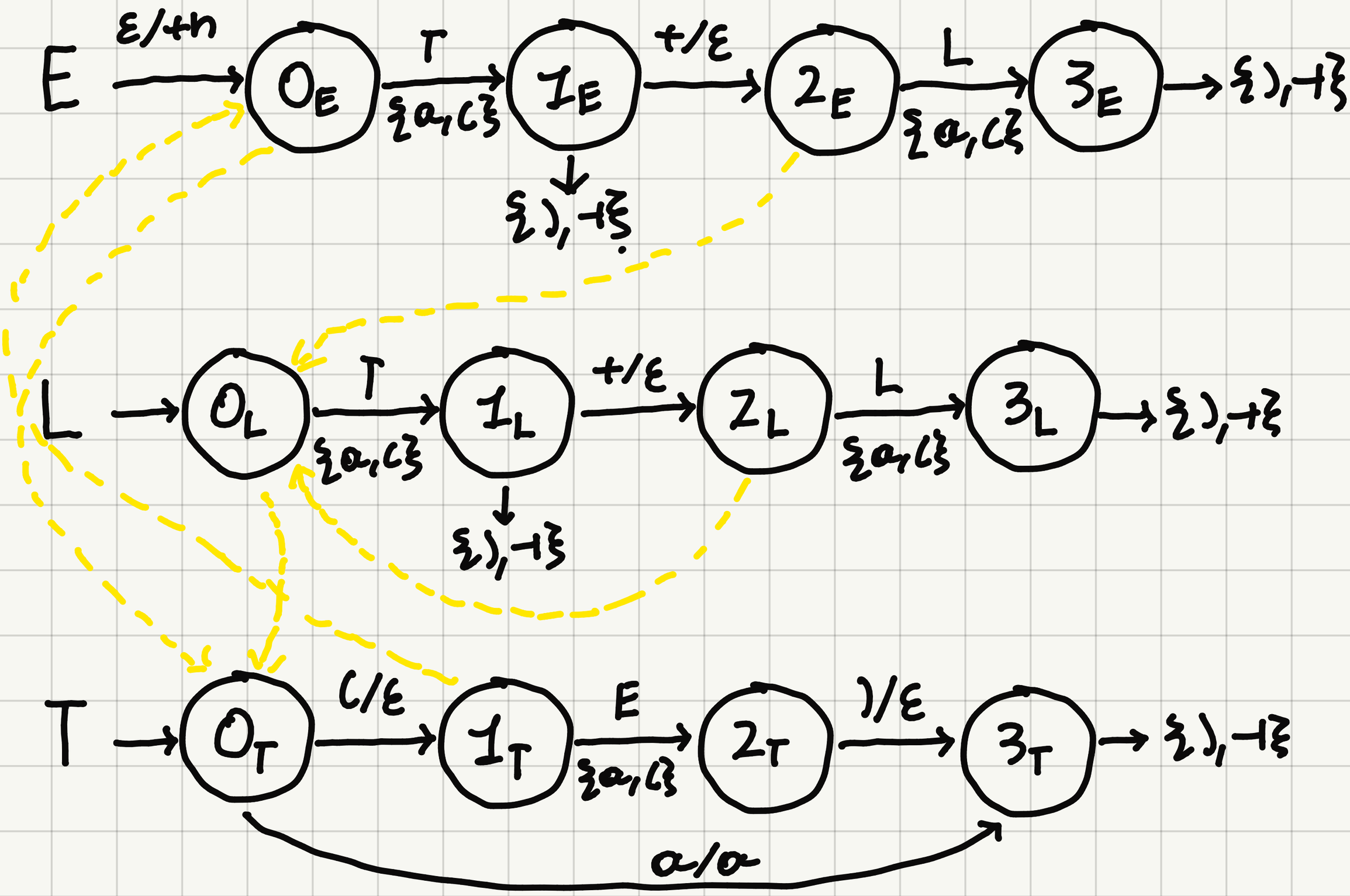
$G_{3'}$:
- $E \to T+T$
- $E \to T+T+L$
- $L \to T+L$
- $L \to T$
- $T \to (E)$
- $T \to a$

$G_{1'}$:
- $E \to +TT$
- $E \to +nTTL$
- $L \to TL$
- $L \to T$
- $T \to E$
- $T \to a$

$G_{\varepsilon'}$:
- $E \to \underset{+}{\varepsilon}\ TT$
- $E \to \underset{+n}{\varepsilon}\ T\ \underset{\varepsilon}{+}\ T\ \underset{\varepsilon}{+}\ L$
- $L \to T\ \underset{\varepsilon}{+}\ L$
- $L \to T$
- $T \to \underset{\varepsilon}{(}\ E\ \underset{\varepsilon}{)}$
- $T \to a$

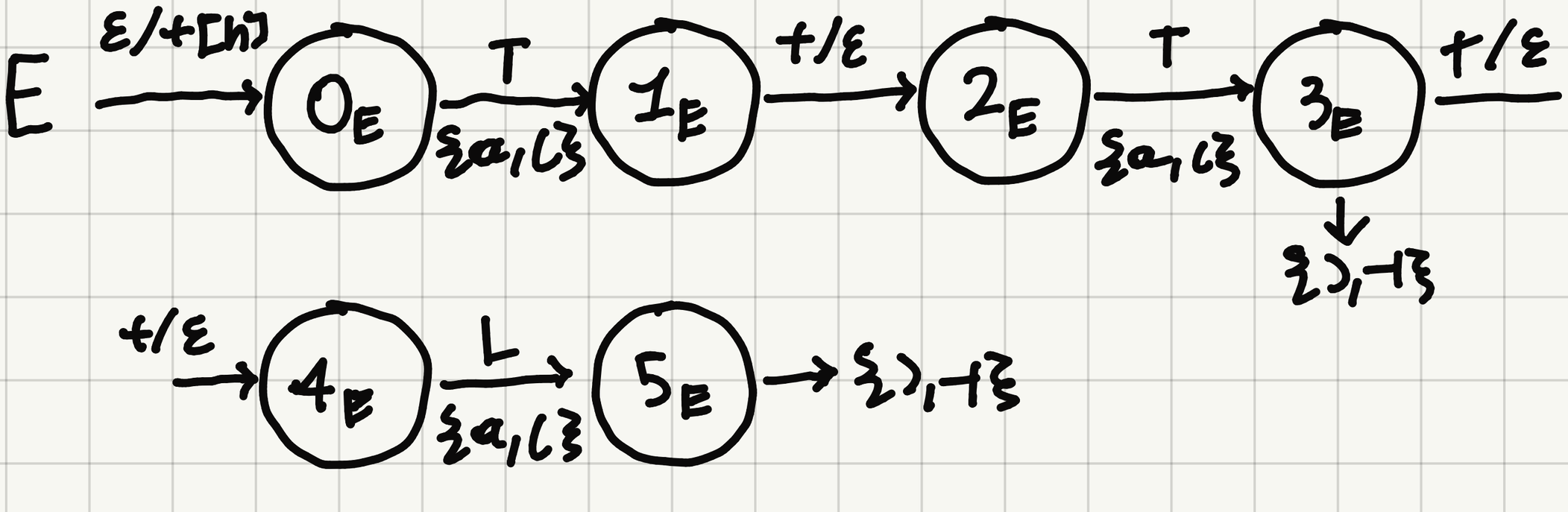


c) Gr



TRANSDUCTION NET DOES NOT HAVE ANY GUIDE SET CONFLICTS ON THE BIFURCATION STATES (1_E, 1_L, O_T) AND IT OUTPUTS A SINGLE STRING FOR EACH INPUT STRING $\Rightarrow$ IT IS IMPLEMENTABLE AS ELL(1) ✓

Grz



TRANSDUCER IS ELL(-1) ON THE SOURCE, BUT IT IS UNABLE TO WRITE

THE TRANSLATION WHEN IT HAS TO BE OUTPUT.

7 FEBRUARY 2018

2. Consider the grammar below (axiom S), which generates a (possibly empty) two-level list. Each sublist consists of letters a (possibly none) and ends with one letter b.

$$\begin{cases} 1\colon\ S \rightarrow X \\ 2\colon\ X \rightarrow A\ X \\ 3\colon\ X \rightarrow \varepsilon \\ 4\colon\ A \rightarrow a\ A \\ 5\colon\ A \rightarrow b \end{cases}$$

A sample two-level list, with three sublists, is:

$$b\ a\ b\ a^2\ b$$

See also the syntax tree on the next pages.

The *length* ≥ 0 of a sublist is the number of elements, i.e., letters a, in the sublist.

We want to decide whether in a list the length of the sublists increases, from left to right, exactly by one, starting from an empty sublist (length 0). If this property is satisfied, a boolean attribute e assumes value *true*, else *false*. For the sample list above, the attribute e is *true*. If the whole list is empty, the attribute e is conventionally *true*.

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

name	*type*	*domain*	*symbol*	*meaning*
l	left	integer	A	length ≥ 0 of a sublist
n	right	integer	X	required length of the sublist immediately appended to a node X
e	left	boolean	X, S	*true* if the sublists appended to a node X, or to the root S, have a length that increases by one from left to right (starting from 0), else *false*

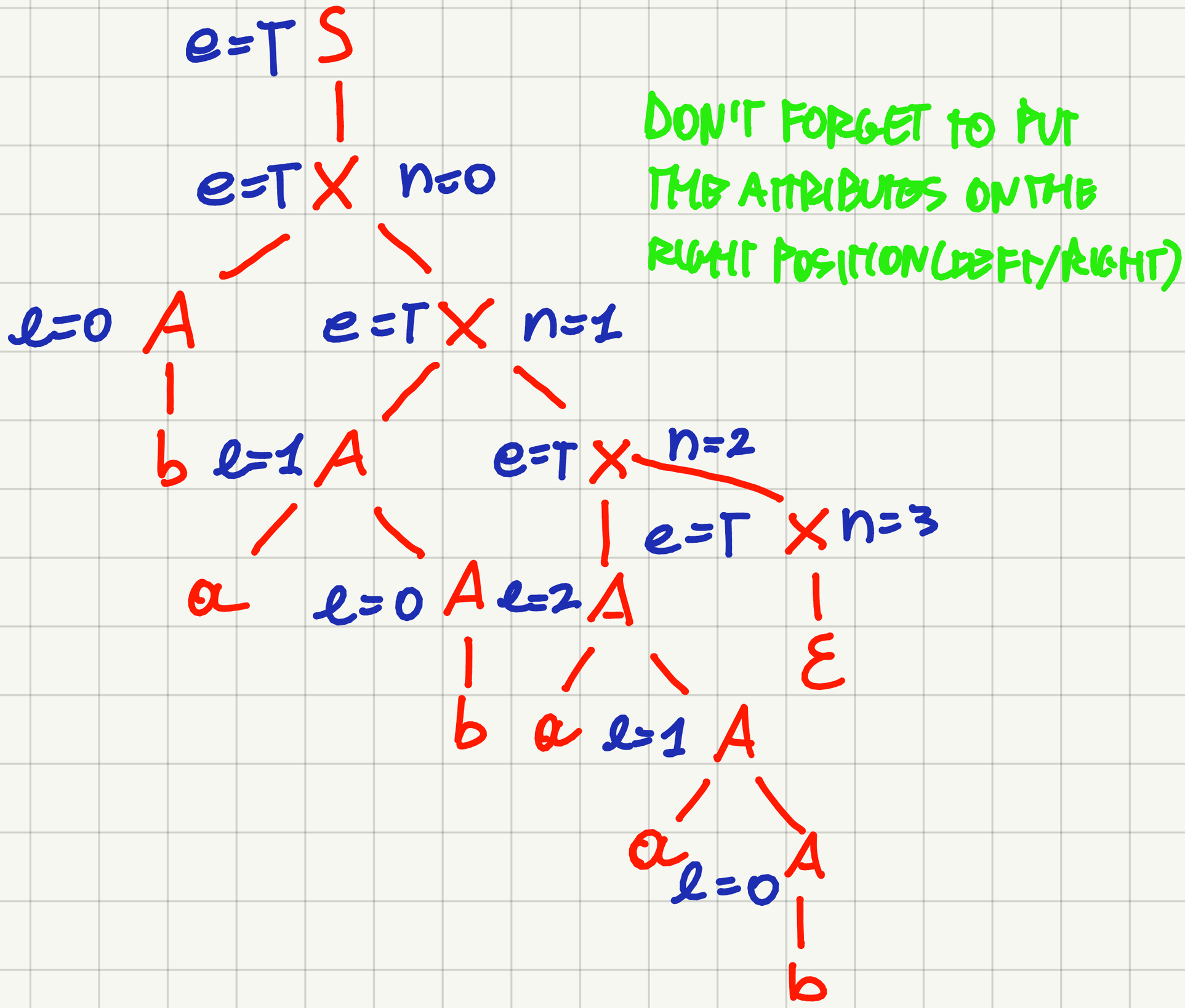
Answer the following questions (use the tables / trees / spaces on the next pages):

(a) Decorate the tree of the sample string $b\ a\ b\ a^2\ b$, prepared on the next page, with the attribute values. Pay attention to the type of each attribute (left or right).

(b) Write an attribute grammar, based on the syntax above and using only the permitted attributes (do not change the attribute types), that computes in the tree root the correctness attribute e for the whole list. The attribute grammar must be of type one-sweep. Please argue that your grammar is so.

(c) (optional) Write the procedure of the semantic analyzer (which exists as the attribute grammar must be of type one-sweep) for nonterminal X. Is it possible to integrate syntactic and semantic analysis ? Please explain your answer.

a)



DON'T FORGET TO PUT THE ATTRIBUTES ON THE RIGHT POSITION (LEFT/RIGHT)

b) WRITE OUR RULES

※	rules	semantics
1	$S_0 \to X_1$	$n_1=0 \quad e_0=e_1$
2	$X_0 \to A_1 X_2$	$n_2=n_1+1$ if $(l==n_0)$ then $e_0=e_2$ else $e_0=$ false endif
3	$X_0 \to \varepsilon$	$e_0=$ true
4	$A_0 \to a A_1$	$l_0=l_1+1$
5	$A_0 \to b$	$l_0=0$

c) procedure X(n_0:in, e_0:out)

var l_1, n_2, e_2

switch rule do

case 2: $X \to AX$ do

call A(l_1)

$n_2 = n_0 + 1$

call X(n_2, e_2)

if ($l_1 == n_0$) then $e_0 = e_2$ else e_0 = false

case 3: $X \to \varepsilon$ do e_0 = true

Otherwise do error

end procedure X

THE INTEGRATION IS POSSIBLE SINCE:

- RULES 2,3 (X) AND 4,5(A) HAVE DISJOINT GUIDE SETS ⇒ THE SYNTACTIC SUPPORT IS OF TYPE LL(1)
- RIGHT ATTRIBUTE n DEPENDS ONLY ON ITSELF FROM THE PARENT NODE
- LEFT ATTRIBUTE l DEPENDS ONLY ON ITSELF FROM A CHILD NODE
- LEFT ATTRIBUTE e ON l FROM A CHILD NODE, ON ITSELF FROM A CHILD NODE AND ON n IN ITS OWN NODE

extra:

ONE-SWEEP ATTRIBUTE WHEN

- RIGHT ATTRIBUTES DEPEND ONLY ON THEMSELF IN THE PARENT NODE
- LEFT ATTRIBUTES DEPEND ONLY ON THEMSELF IN THE CHILD NODE AND ON RIGHT ATTRIBUTES IN THE SAME NODE
- NO RIGHT ATTRIBUTE DEPENDS ON A LEFT NODE

L-TYPE ATTRIBUTE WHEN

- "LEFT-TO-RIGHT" PARSING