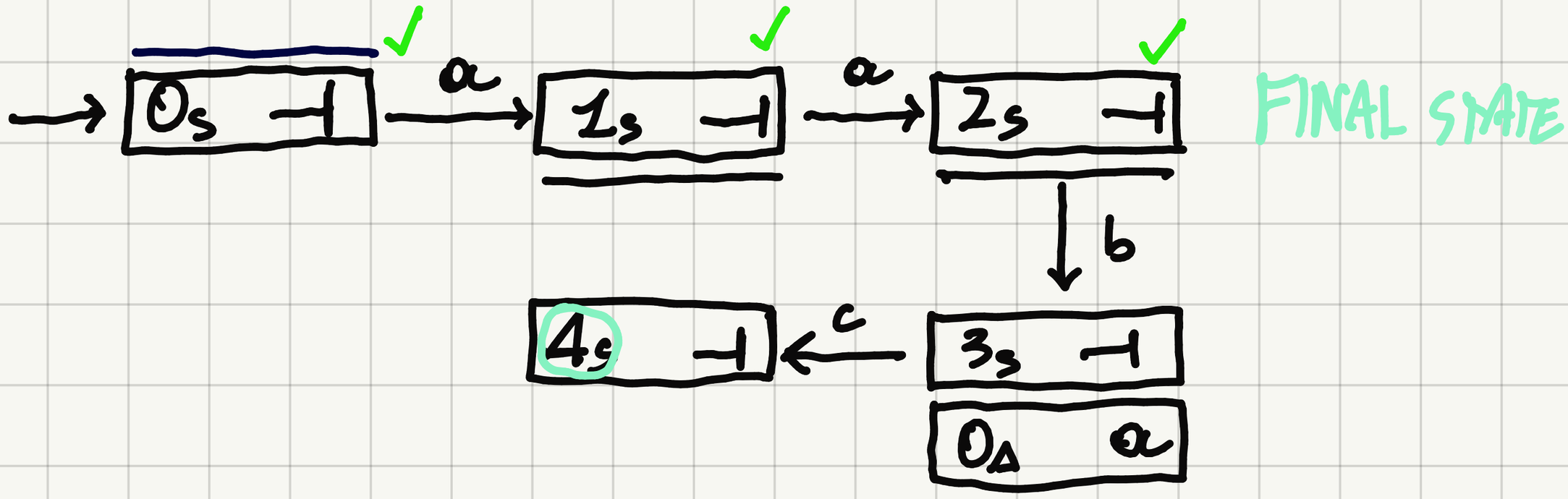
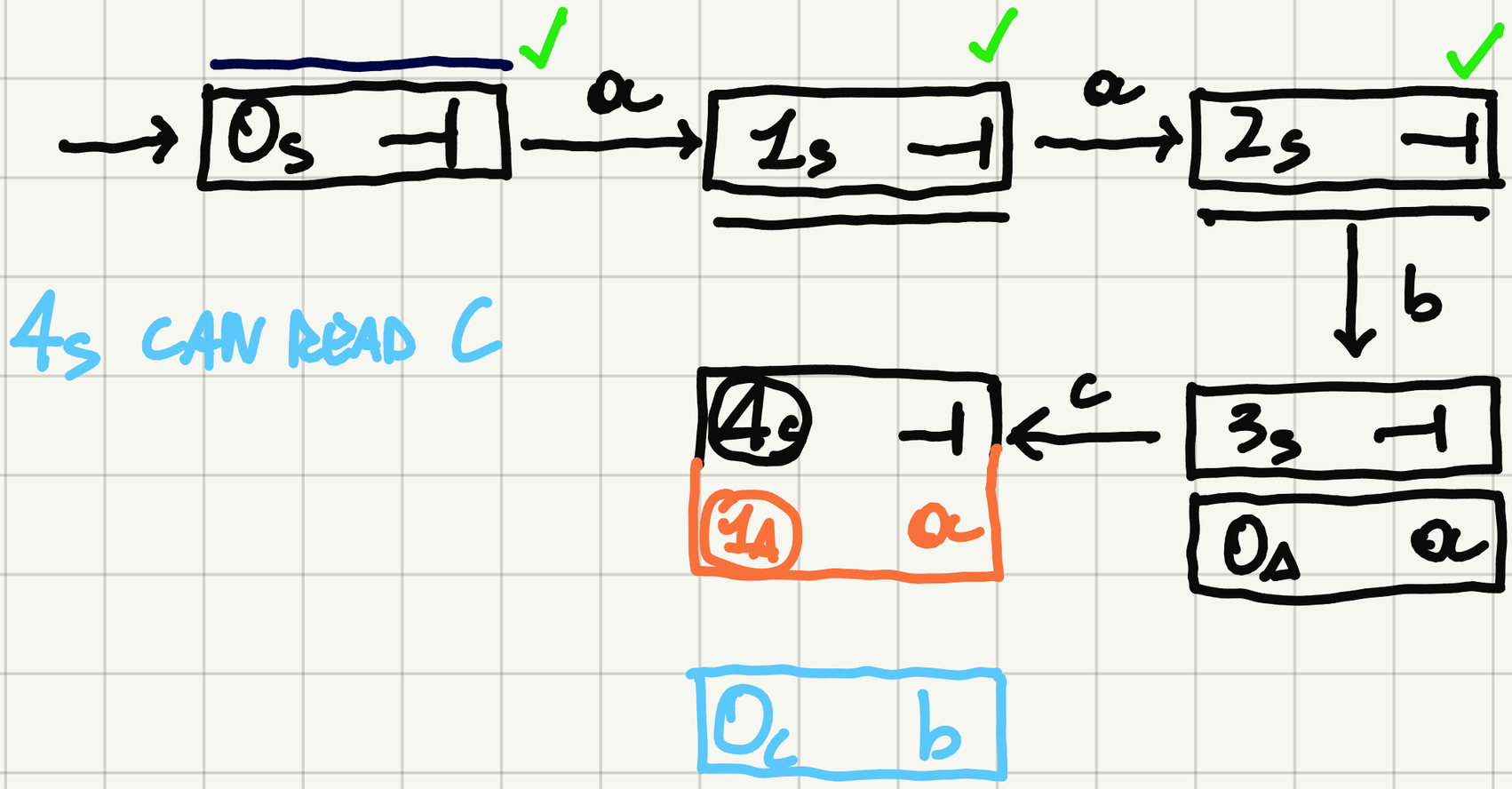
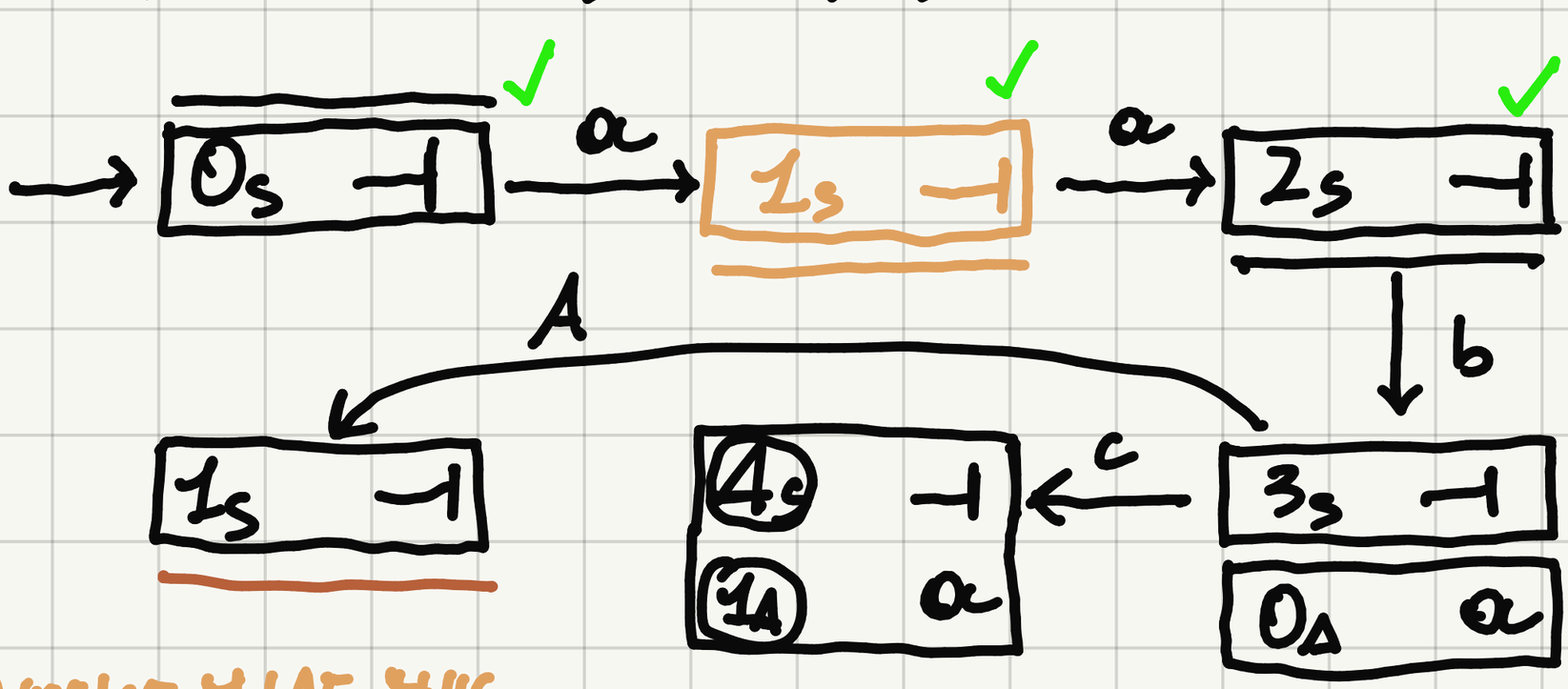



$0_A$ CAN READ c? YES



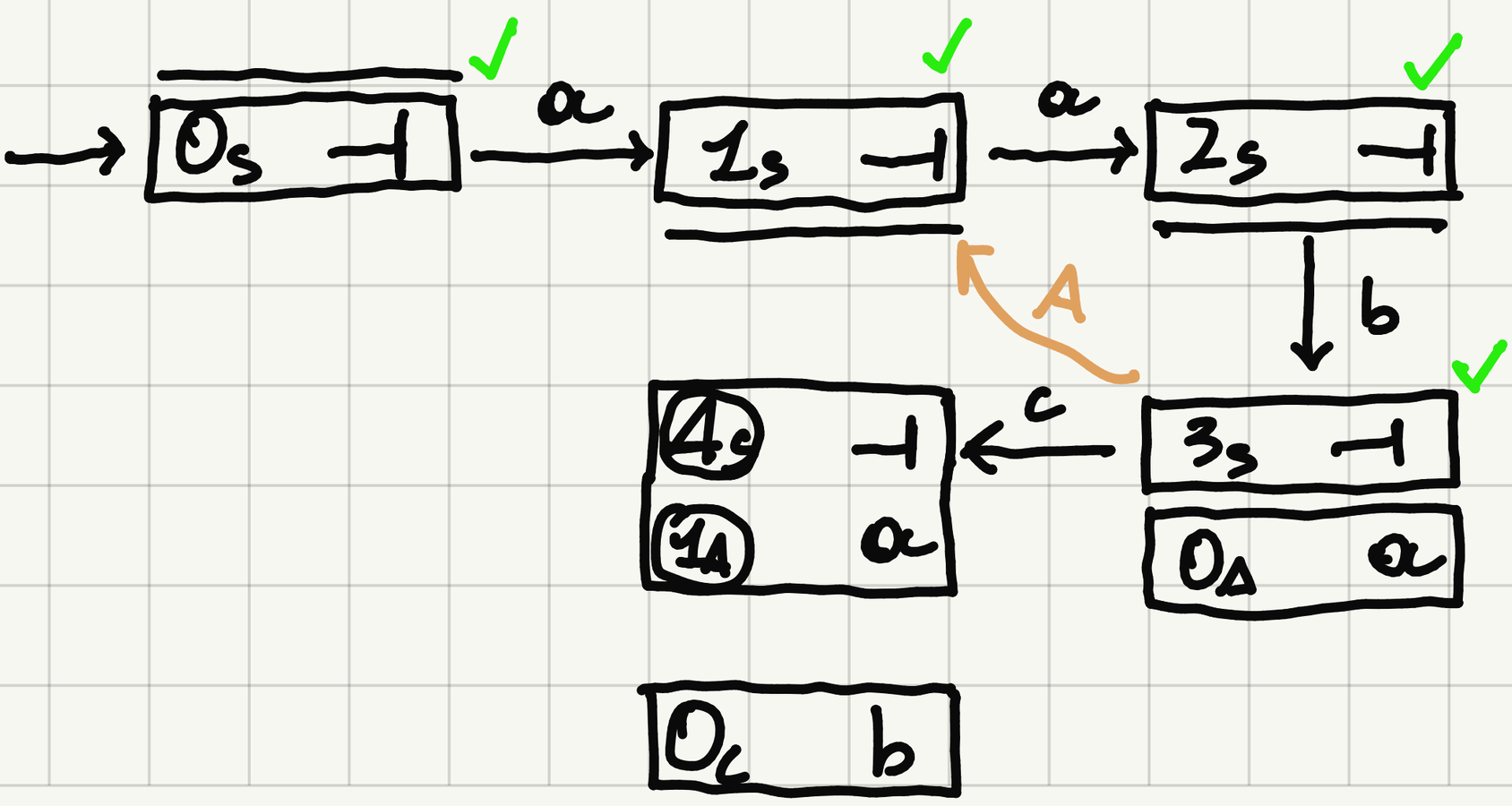

$4_s$ CAN READ C

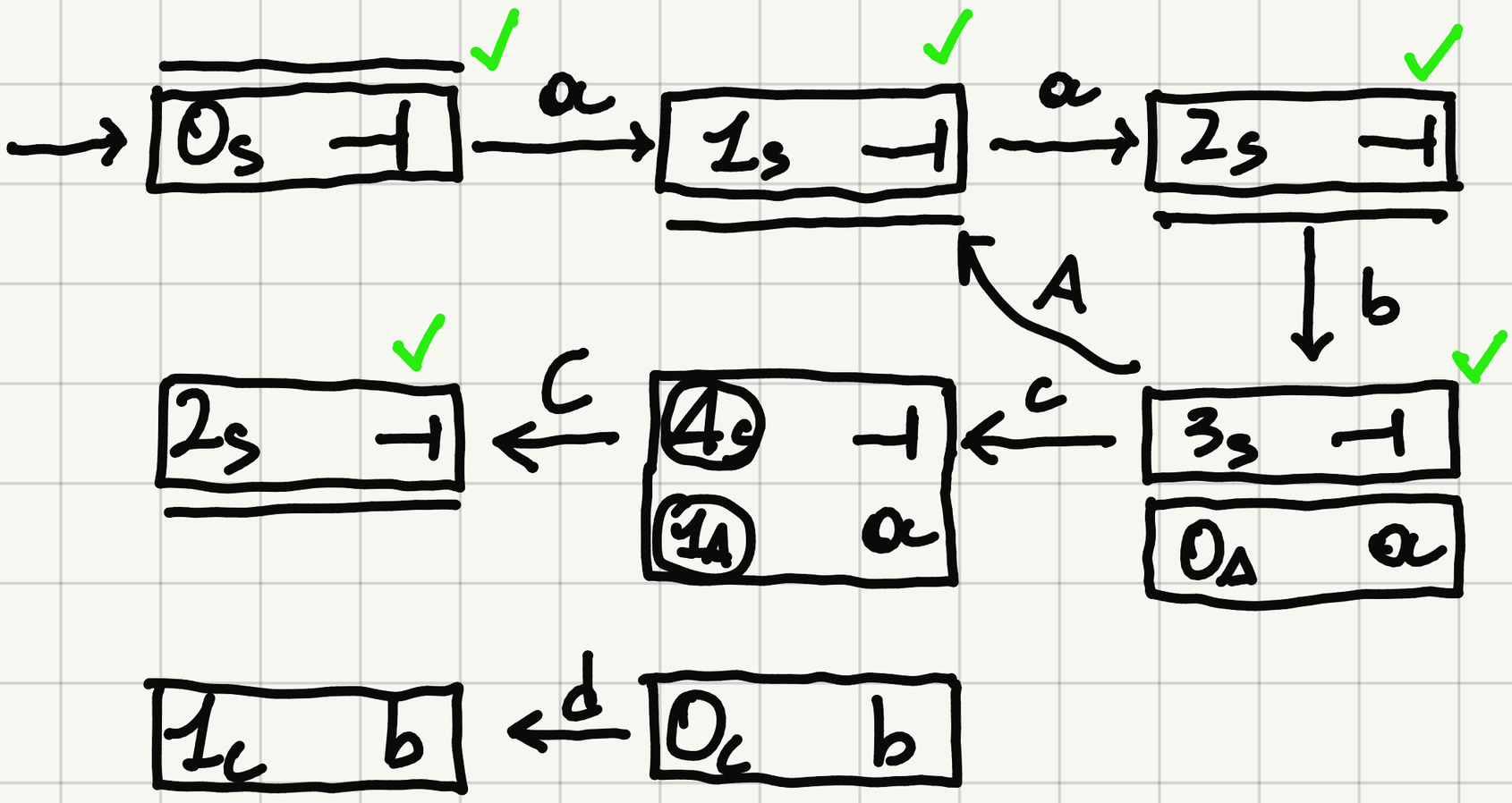
WHAT HAPPENS WHEN $3_s$ READS A?



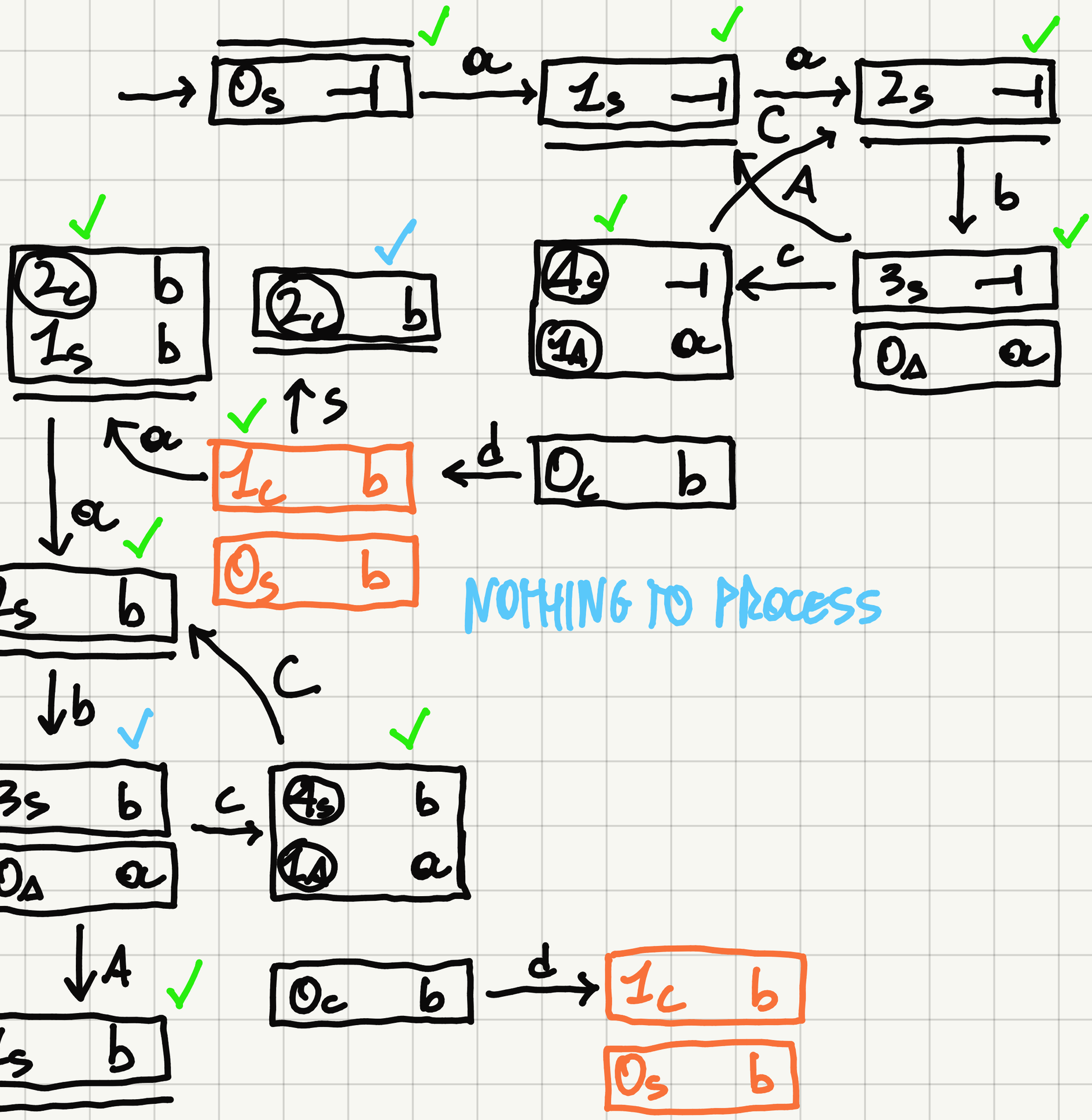

WE NOTICE THAT THIS STATE IS ALREADY EXISTENT

OBVIOUSLY, $0_A$ CAN'T READ NON-TERMINAL A

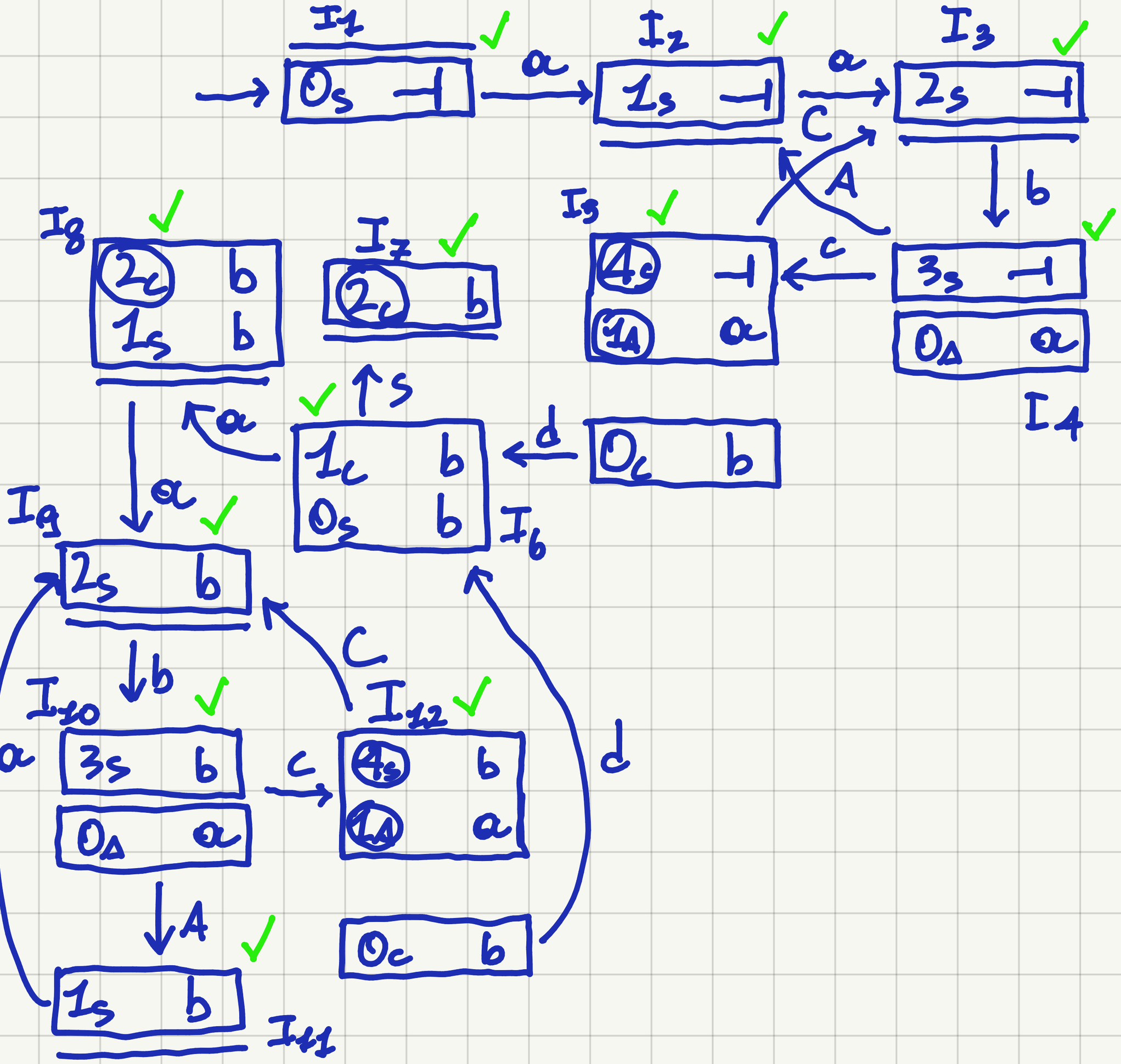





AFTER READING S, C GOES TO $2_c$.
THAT IS A FINAL STATE ⇒ WRITE THE SAME LOOK-AHEAD




NOTHING TO PROCESS

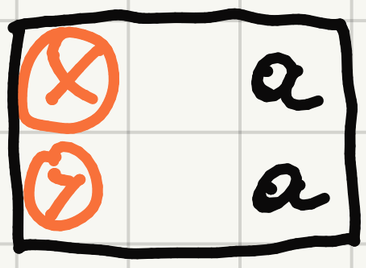



TO VERIFY ELR(1), CHECK THE 3 POSSIBLE CONFLICTS:

- SHIFT-REDUCE

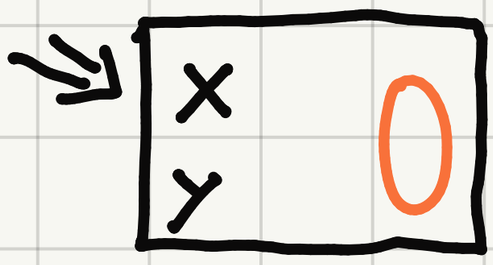

? LOOK AHEAD = OUTGOING ARC

- REDUCE-REDUCE




WHICH ONE TO REDUCE ?

- CONVERGENT ARC




SET OF LOOK-AHEADS INTERSECT

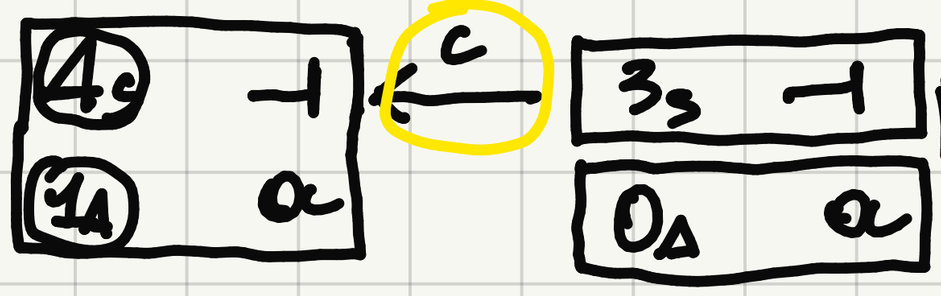
"SCANNING" THE PILOT, WE DIDN'T FOUND ANY CONFLICT => IT IS ELR(1)

TO VERIFY ELL(1), WE MUST VERIFY THAT IT IS ELR(1) (✓)

AND SEE IF WE CAN HAVE LEFT-RECURSIVE DERIVATIONS.

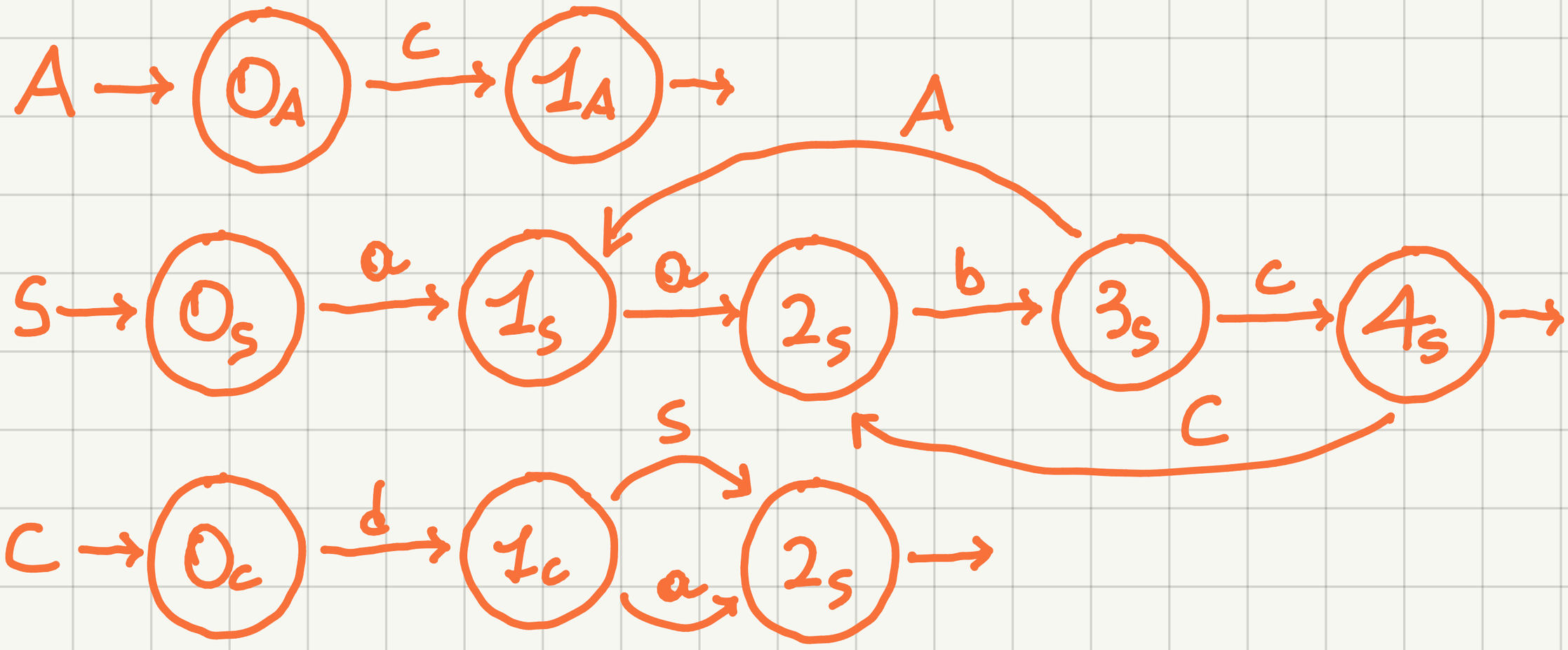
IN OTHER WORDS, LOOKING AT THE PILOT EACH SINGLE MICRO-STATE

MUST BE COMPOSED BY ONE STATE.

HERE, 

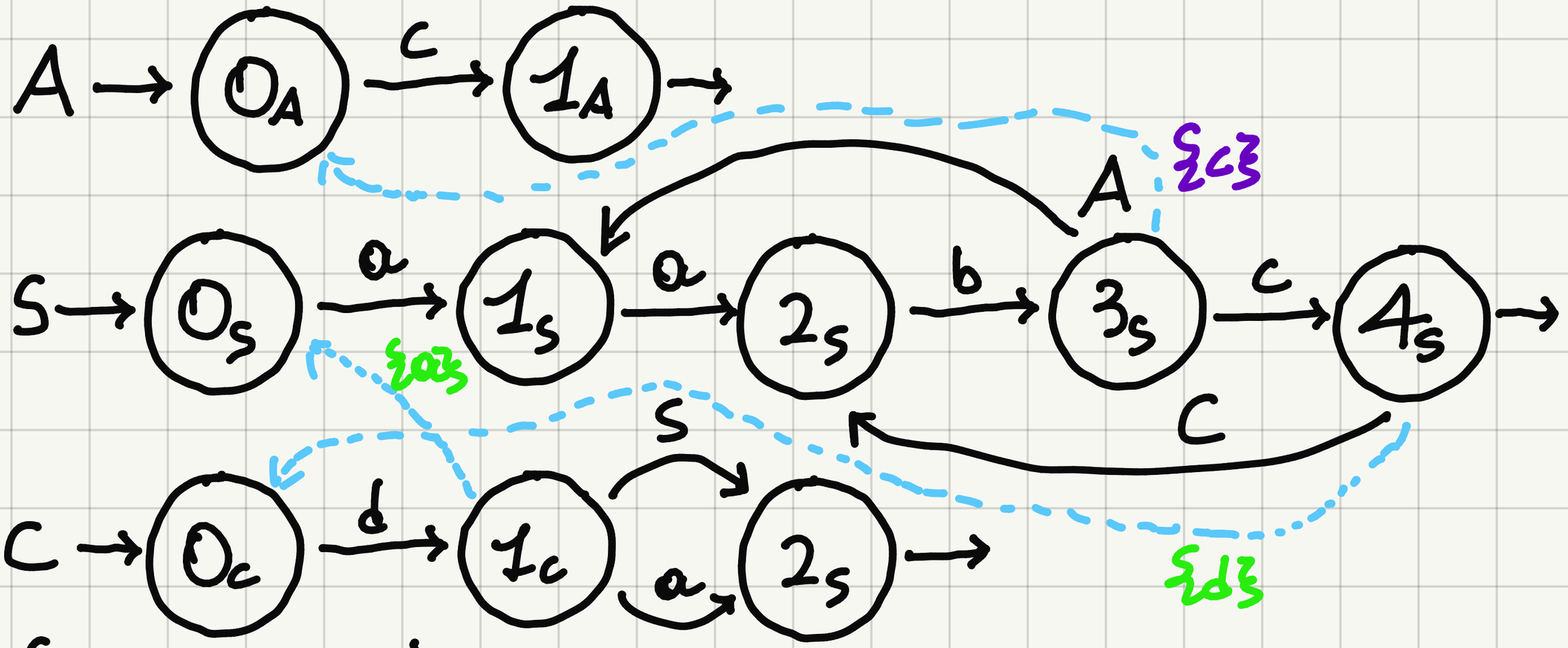
DOESN'T SATISFY THE CONDITION

⇒ IT IS NOT ELL(1)

b) 


LET'S ADD THE CALL ARCS. WHEN A MACHINE HAS AN ARC THAT

CALLS ANOTHER ONE, WE DRAW AN ARC TO ITS INIMAL STATE (S)

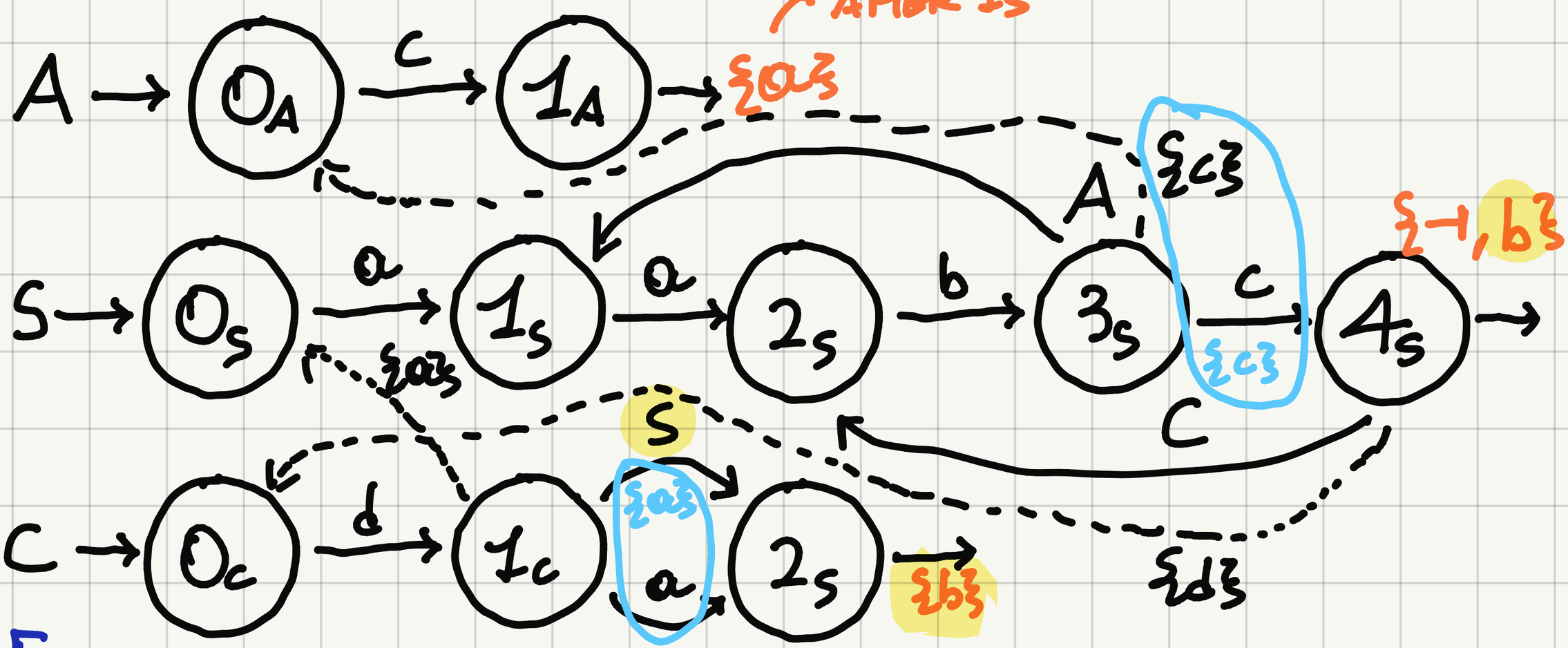



START FROM $3_S$. WE HAVE TO WRITE WHAT WE EXPECT FROM A

SIMILAR FOR THE OTHERS AND WE OBTAIN THE GUIDE SETS FOR

EACH CALL ARROWS

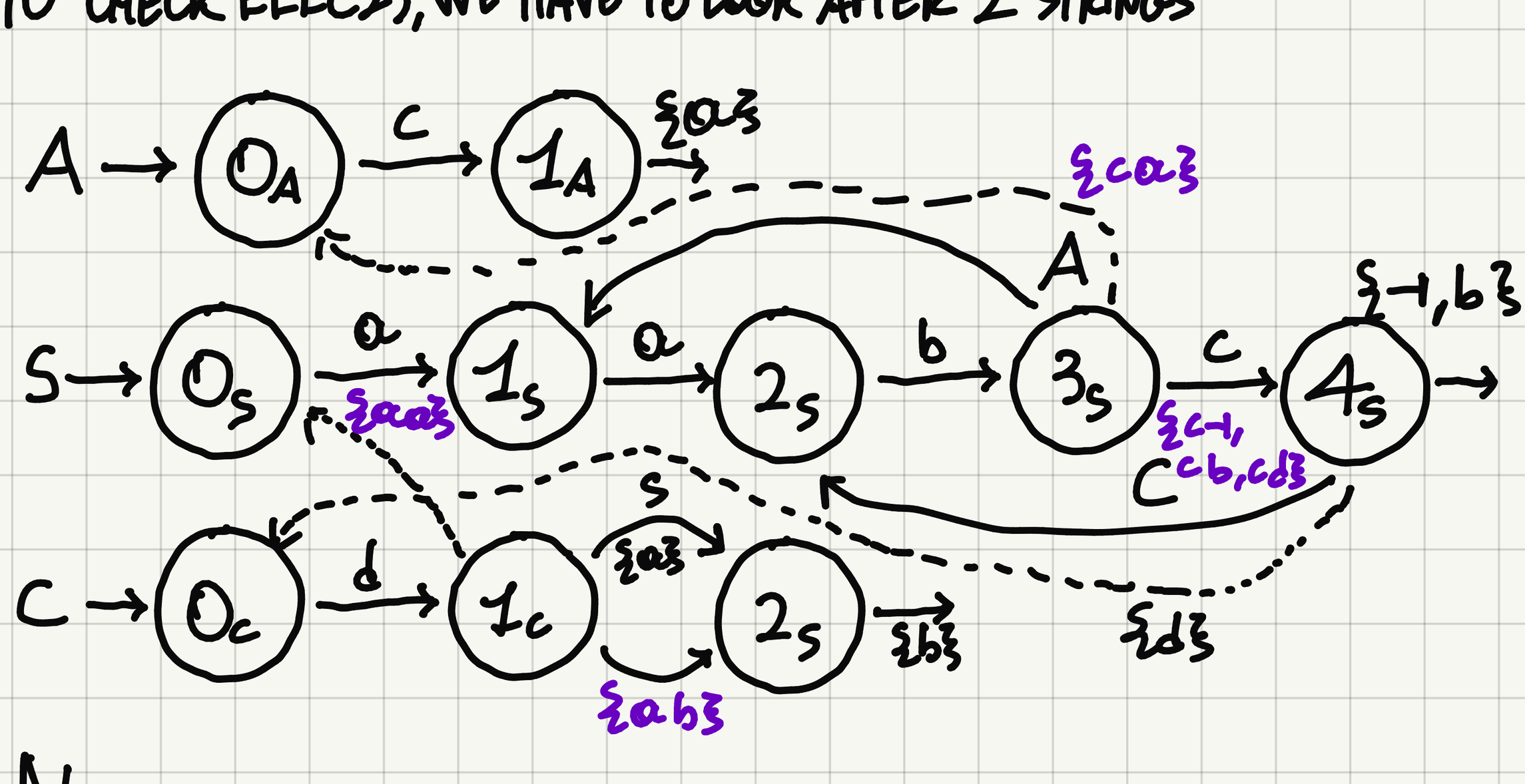
# NOW, WRITE THE GUIDE SETS FOR EXIT ARROWS




FROM THIS NET, WE CAN CONCLUDE THAT ELL(1) IS NOT SATISFIED.

TO DO SO, LOOK TO THE GUIDE SETS OF OUTGOING ARCS AND SEE IF SOME INTERSECTS

c) TO CHECK ELL(2), WE HAVE TO LOOK AFTER 2 STRINGS



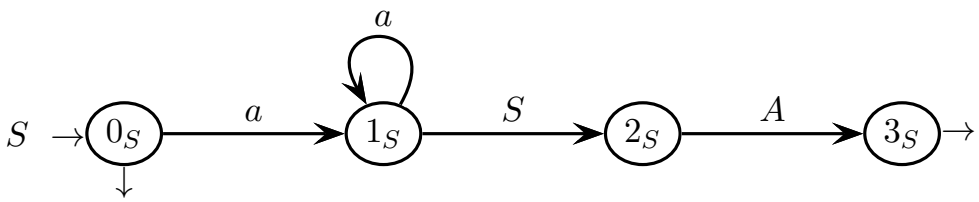

NOW, NO ARCS INTERSECT ANYMORE ⇒ IT IS ELL(2)

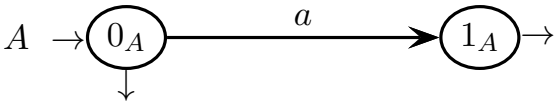
(IF IT WAS NOT, WE SHOULD CHECK ELL(3), ELL(4)...)

**14 JANUARY 2020**

## 3 Syntax Analysis and Parsing Methodologies 20%

1. Consider the grammar $G$ below, over the one-letter terminal alphabet $\Sigma = \{\ a\ \}$ and the two-letter nonterminal alphabet $V = \{\ A,\ S\ \}$ (axiom $S$):




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

(a) Draw the complete pilot automaton of grammar $G$. Determine whether grammar $G$ is of type $ELR(1)$. Highlight all the conflicts that may be present in the pilot automaton, and indicate their type.

(b) By finding all the guide sets on the call arcs and exit arrows, determine whether grammar $G$ is of type $ELL(1)$. Justify your answer.

(c) Draw the syntax tree (or all syntax trees if many exist) of the valid string $a\,a$.

(d) ~~(optional) Analyze the valid string $a\,a$ by means of the Earley method, and highlight the item(s) showing that the string is accepted.~~

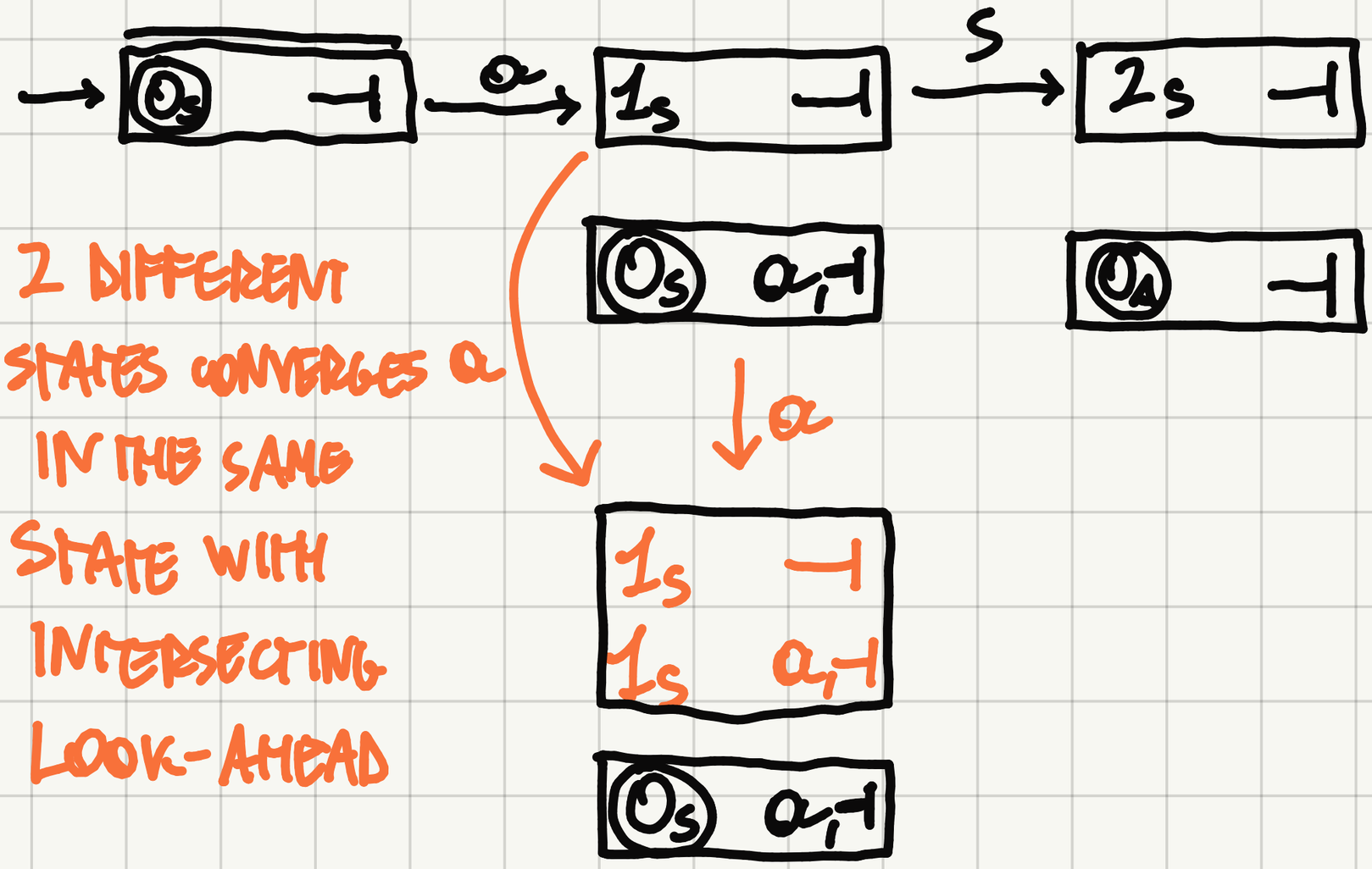
a)



*

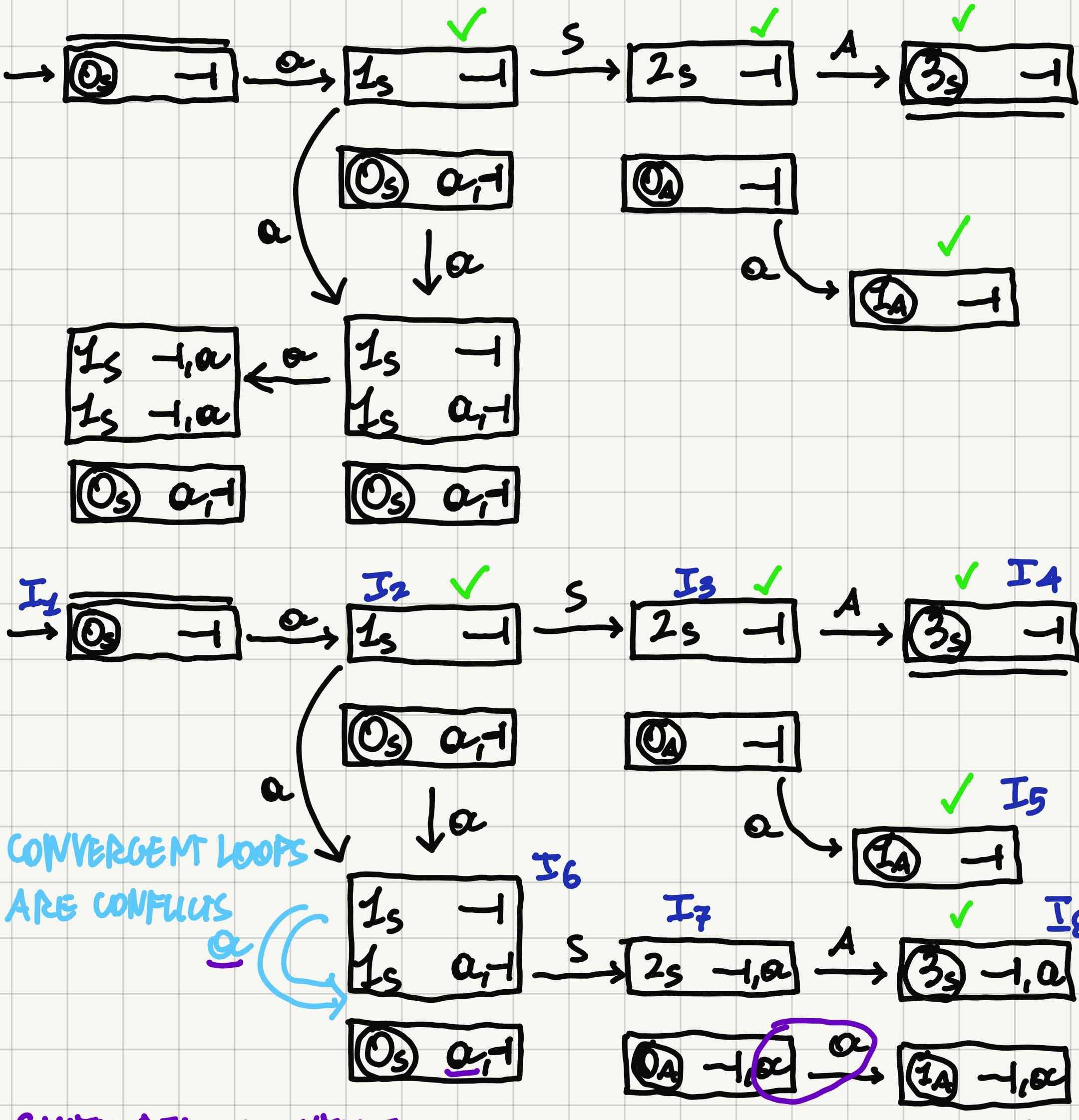
WE CAN'T WRITE A NON-TERMINAL IN THE LOOK-AHEAD.

LET'S STUDY A. WE FIND EITHER NON-TERMINAL OR EMPTY STRING (NOT ACCEPTABLE AS LOOK-AHEAD). IN THE SECOND CASE, WE SHOULD LOOK WHAT HAPPENS AFTER $3_s$, THAT IS FINAL AND, HENCE, INSERT IN THE LOOK-AHEAD THE SAME THAT HAS BEEN FOUND IN $1_s$ (-1)

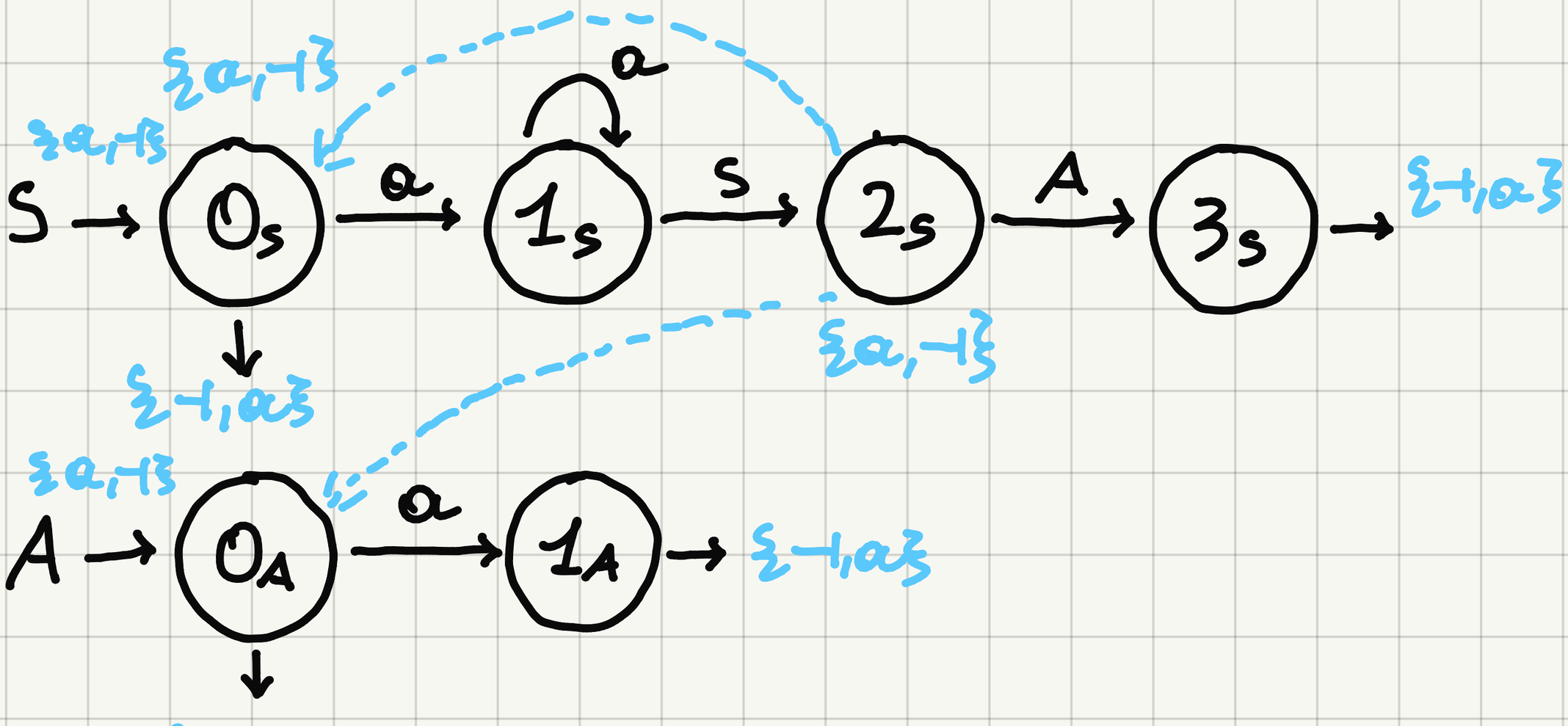



2 DIFFERENT STATES CONVERGES A IN THE SAME STATE WITH INTERSECTING LOOK-AHEAD

CONVERGENT ARC CONFLICT => WE CAN IMMEDIATELLY CONCLUDE IT IS NOT ELR(1)

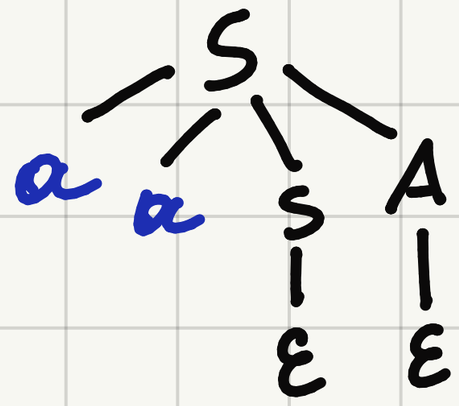


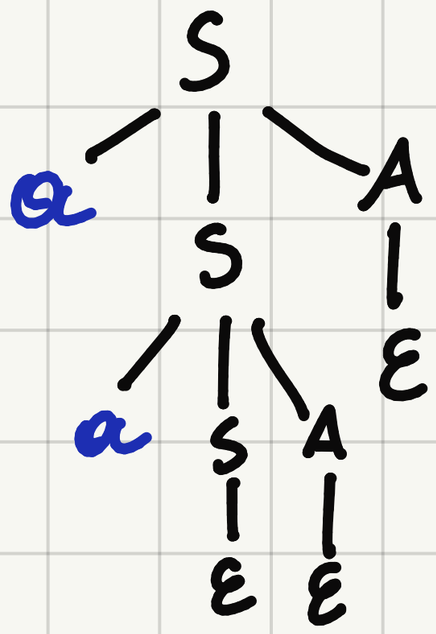

b) SINCE WE PROVED IT IS NOT ELR(1), IT WONT BE ELL(1) AS WELL

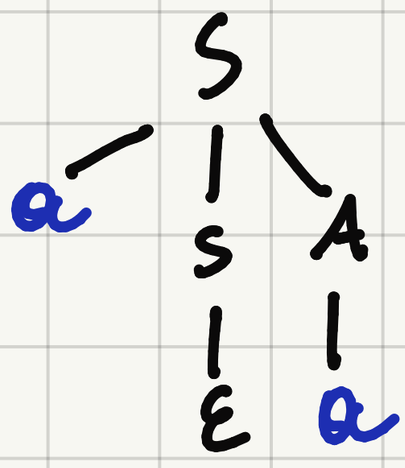


ALL THE GUIDE SETS OF OUTGOING ARCS INTERSECT

c)



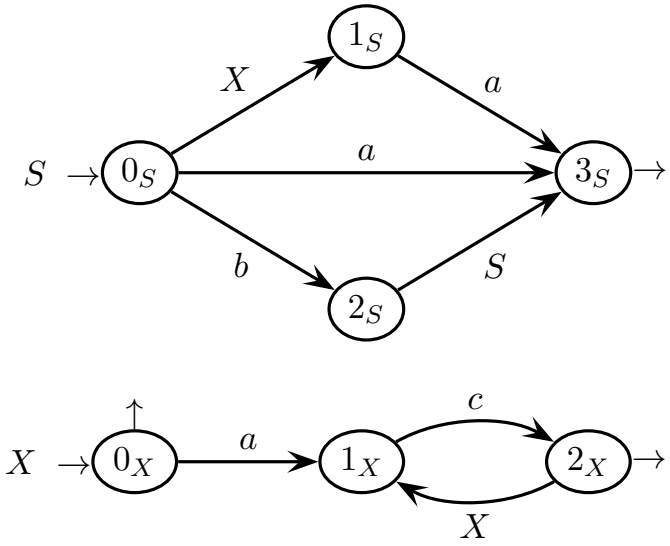

! GRAMMAR G IS AMBIGUOUS. MAKES SENSE, SINCE IT IS NOT

ELL

**5 FEBRUARY 2013**

## 3 Syntax Analysis and Parsing 20%

1. Consider the grammar specified below, represented as a network of recursive machines (axiom $S$):




Answer the following questions:

(a) Draw the network pilot graph and say if the grammar is of type $ELR$, with listing the possible conflicts and describing them shortly.

(b) Draw the network control flow graph $PCFG$ along with the prospect sets (in the final states) and the guide sets (on the call edges and the exit darts), and say if the graph is deterministic or not, with listing the possible nondeterminism points and describing them shortly. Furthermore say if the pilot graph satisfies the Single Transition Property ($STP$) and explain why.

(c) (optional) The letter $a$ from $1_S$ to $3_S$ becomes a $c$ and $0_X$ is not final any longer; the rest of the network is left unchanged. Is the grammar now $ELR$ ? Is it now $ELL$ ? Shortly explain your answers.

FOR WHAT FOLLOWS SEE THE WARNING ON THE TEXT COVER PAGE

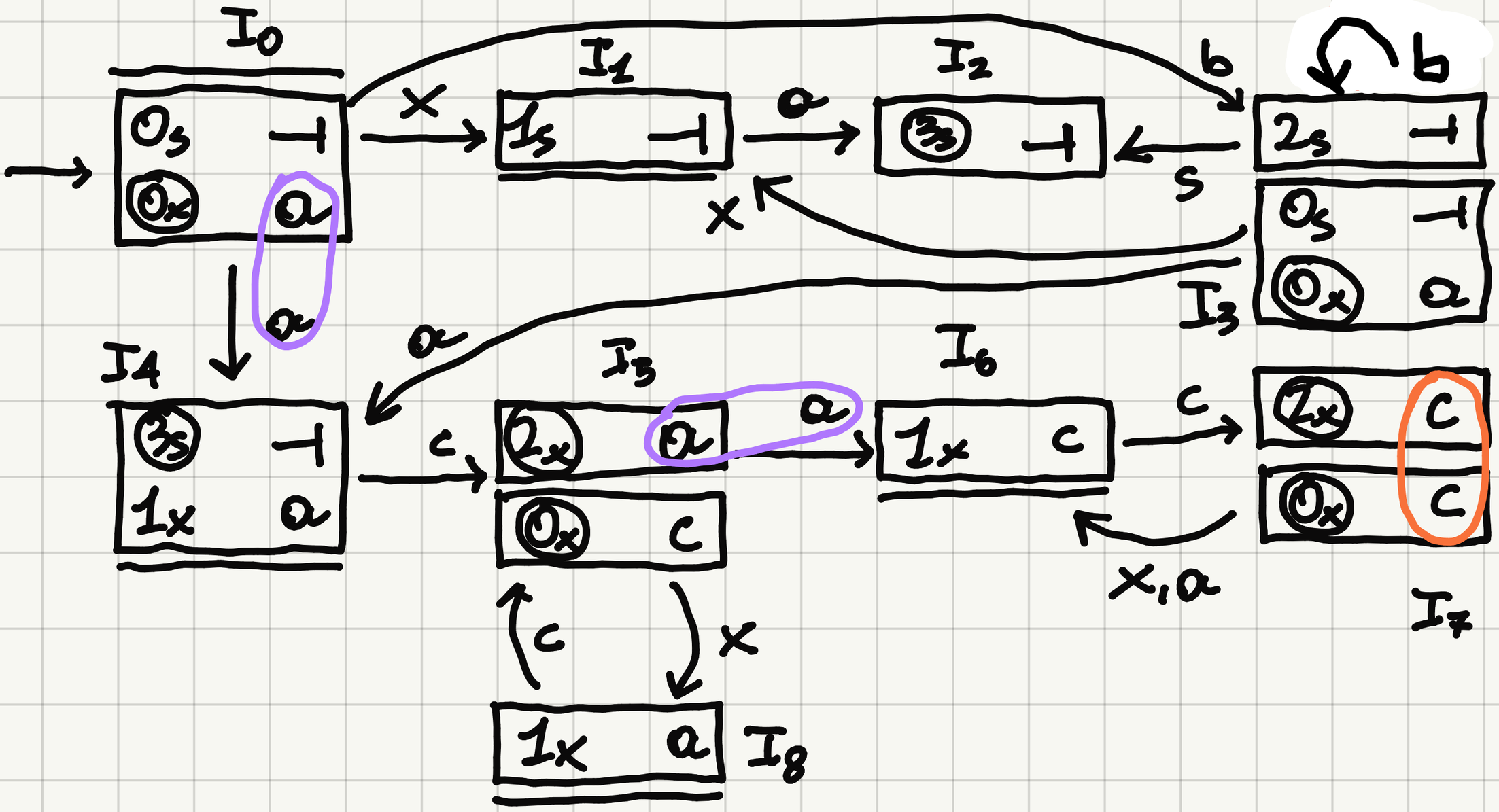
Those who prefer to answer (a-b) by means of the classical $LR$ and $LL$ analysis methodologies, please use the not extended ($BNF$) grammar below (axiom $S$).

$$\begin{cases} S & \rightarrow & X\ a \ \mid\ b\ S \ \mid\ a \\ X & \rightarrow & a\ c\ Y \ \mid\ \varepsilon \\ Y & \rightarrow & X\ c\ Y \ \mid\ \varepsilon \end{cases}$$

To answer (c) according to the classical $LR$ and $LL$ analysis methodologies, apply the specified changes to the $BNF$ grammar.

---

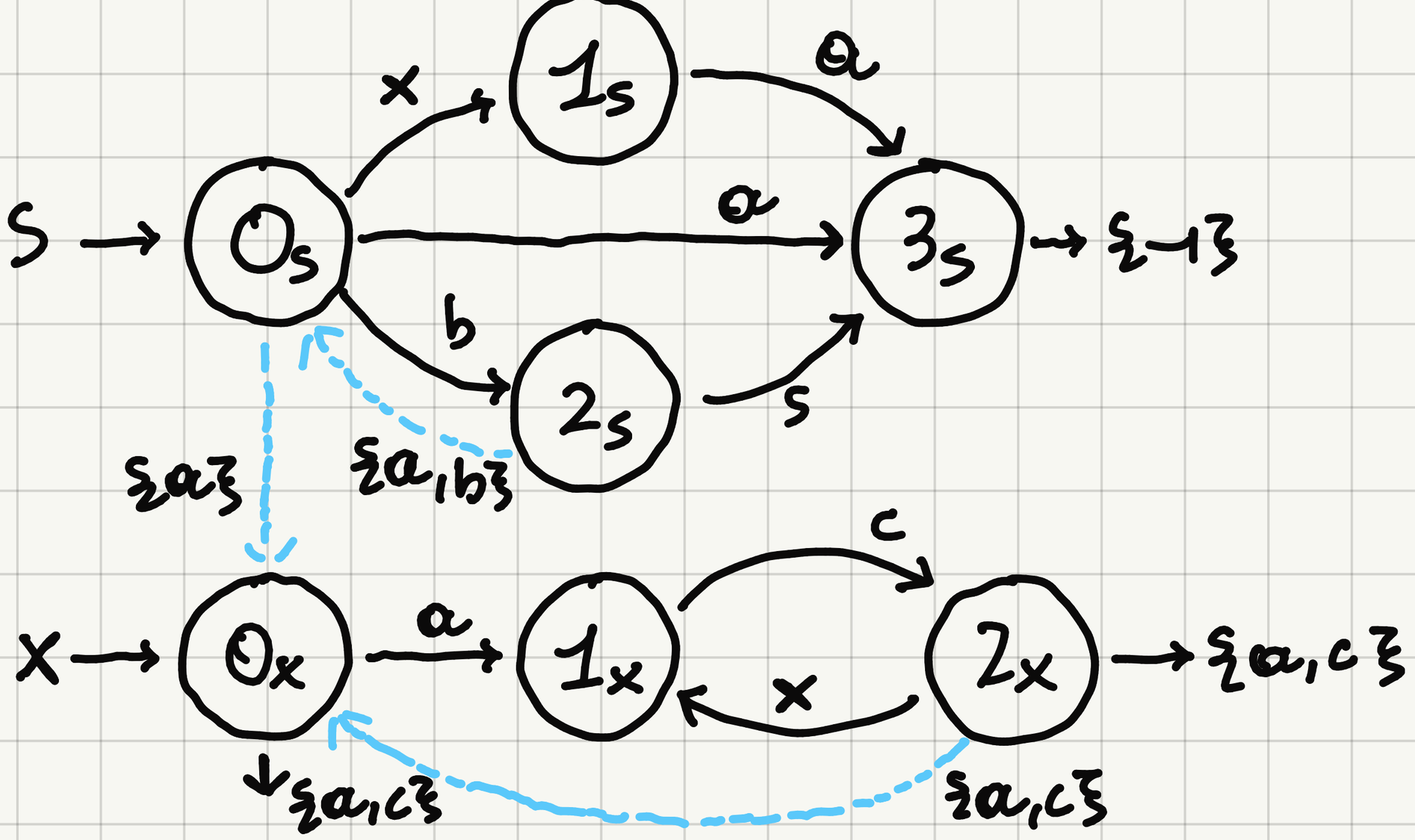
a)




- SHIFT-REDUCE CONFLICTS
- REDUCE-REDUCE CONFLICTS
- NO CONVERGENT ARC CONFLICTS

⇒ IT IS NOT ELR(1)

b)




GRAPH IS NON-DETERMINISTIC SINCE:

- $0_S \rightarrow$ SHIFT $0_S \xrightarrow{a} 3_S \wedge$ CALL $0_S \xrightarrow{a} 0_X$
- $0_X \rightarrow$ SHIFT $0_X \rightarrow 1_X \wedge$ RETURN $\rightarrow a$
- $2_X \rightarrow$ RETURN $a\,c \wedge$ CALL $a\,c$

$I_0 \xrightarrow{a} I_4$ AND $I_3 \xrightarrow{a} I_4 \Rightarrow$ IT IS NOT SLR

c) LOOK-AHEAD OF 1 TURNS IN $c \Rightarrow I_5 \equiv I_7$, $I_6 \equiv I_8$

$\Rightarrow$ NO MORE SHIFT-REDUCE AND REDUCE-REDUCE CONFLICTS

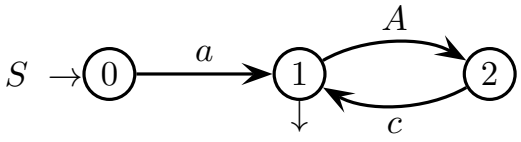
$\Rightarrow$ IT TURNS TO BE ELR(1).

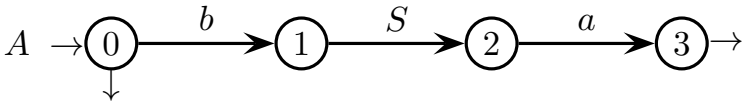
THE CONFLICTS DESCRIBED IN POINT b STILL HOLDS $\Rightarrow$ IT STILL IS NOT LL(1)

**2 OCTOBER 2012**

## 3 Syntax Analysis and Parsing 20%

1. Consider the extended (EBNF) grammar below, represented as a network of recursive machines over the terminal alphabet { $a$, $b$ } and the nonterminal alphabet { $S$, $A$ } (axiom $S$):




Answer the following questions (by the ext. analysis methodology $ELR$ and $ELL$):

(a) Draw the grammar pilot graph and explain if the grammar is of type $ELR(1)$.

(b) Explain if the grammar is of type $ELL(1)$.

(c) (optional) Simulate the $ELR$ analysis of the sample string below:

$$a\ b\ a\ a\ c$$

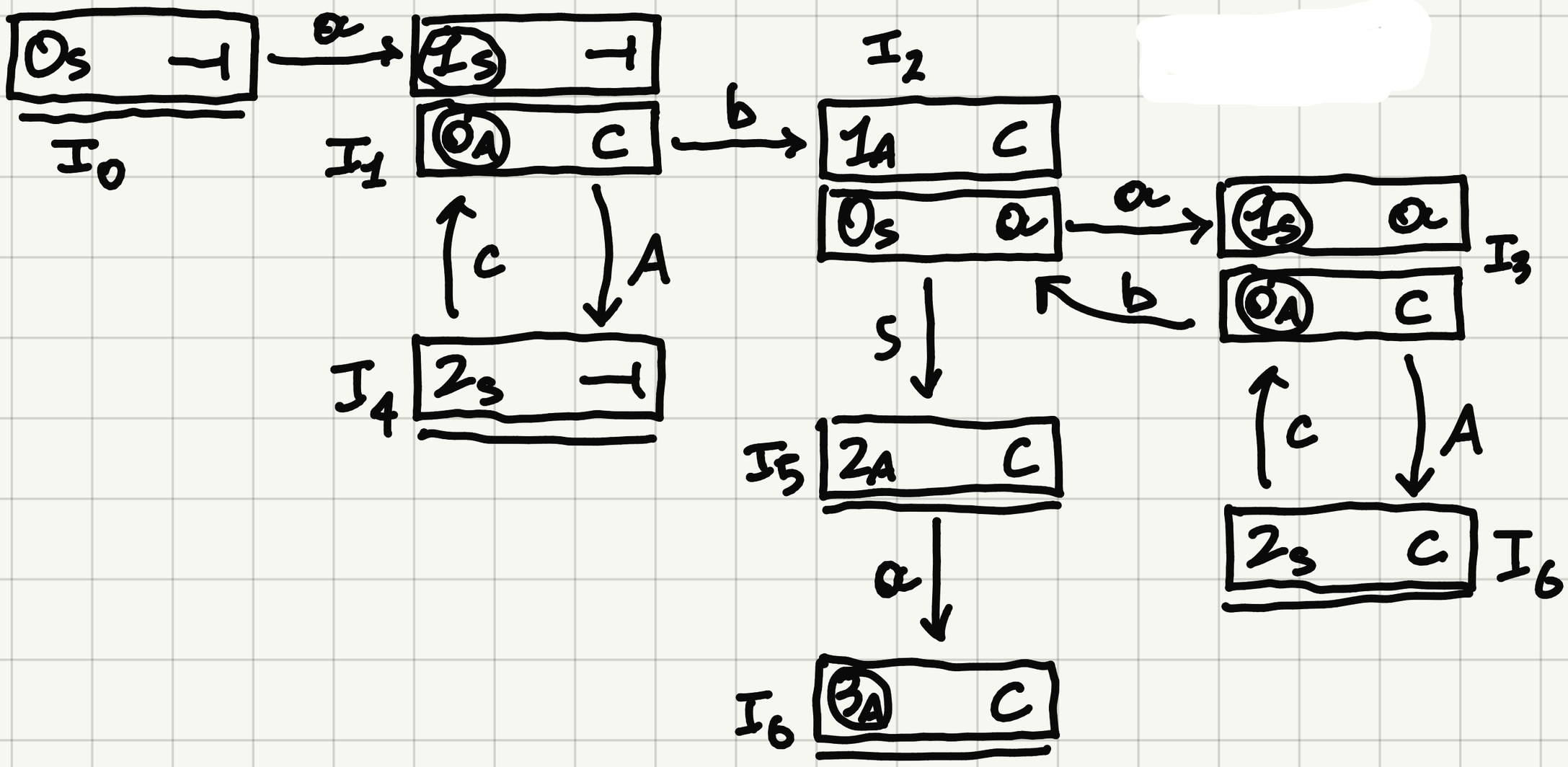
Please use the table on the next page (the number of rows is not significant).

Those who prefer to work with the classical $LR$ and $LL$ analysis methodologies, please use the not extended (BNF) grammar below, equivalent to the machine network.

$$\begin{cases} S \rightarrow a\ X \\ X \rightarrow A\ c\ X \mid \varepsilon \\ A \rightarrow b\ S\ a \mid \varepsilon \end{cases}$$

---

a)



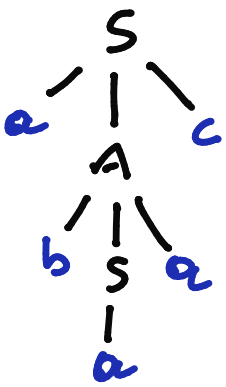

- NO SHIFT-REDUCE CONFLICTS
- NO REDUCE-REDUCE CONFLICTS
- NO CONVERGENT ARC CONFLICTS

} ⇒ IT IS ELR(1)

b) THE PILOT HAS NO LEFT RECURSIONS ⇒ IT IS LL(1)

c) $I_0$ $I_0 a I_1$ $I_0 a I_1 b I_2$ Etc...

| – | $a$ | $b$ | $a$ | $a$ | $c$ | mossa |
|---|---|---|---|---|---|---|
| $0_S \perp$ | a | b | a | a | c | SHIFT a |
| | 1 1s #1 1A ⊥ | b | a | a | c | SHIFT b |
| | 1 1s #1 1A ⊥ | 2 1A #2 0s ⊥ | a | a | c | SHIFT a |
| | 1 1s #1 1A ⊥ | 2 1A #2 0s ⊥ | 3 1s #2 0A ⊥ | a | c | a ⇝ S |
| | 1 1s #1 1A ⊥ | 2 1A #2 0s ⊥ | S | a | c | SHIFT S |
| | 1 1s #1 1A ⊥ | 2 1A #2 0s ⊥ | 5 2A #1 | a | c | SHIFT a |
| | 1 1s #1 1A ⊥ | 2 1A #2 0s ⊥ | 5 2A #1 | 7 3A #1 | c | bSa ⇝ A |
| | 1 1s #1 1A ⊥ | A | | | c | SHIFT A |
| | 1 1s #1 1A ⊥ | 4 2s #1 | | | c | SHIFT c |
| | 1 1s #1 1A ⊥ | 4 2s #1 | | | 1 1s #1 | aAc ⇝ S |
| | S | | | | | ACCEPT |
| | | | | | | |
| | | | | | | |
| | | | | | | |



# EXTRA QUESTION) WRITE THE RECURSIVE DESCENT PROCEDURE FOR NONTERMINAL S BASED ON ELL METHOD

```
program PARSER
    cc = next
    call S
    if cc ∈ {-1} then
        accept
    else
        error
end program
```

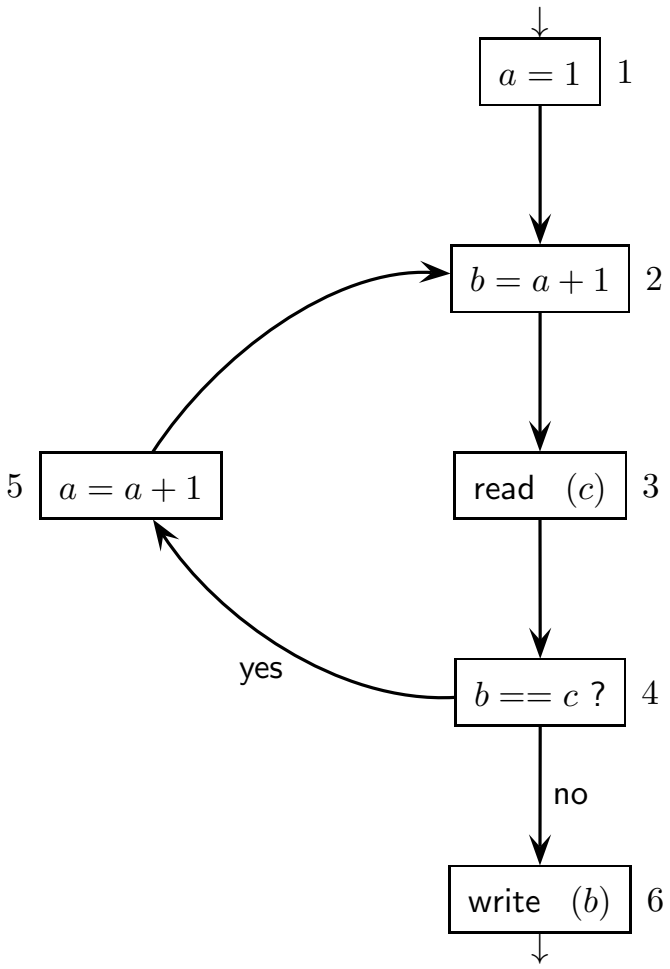
ALWAYS THE SAME

```
procedure S
O_S:  if cc∈{a} then
        cc=next
        goto 1_S
      error
1_S:  if cc∈{-1,a} then return
      if cc∈{b,c} then call A
      error
2_S:  if cc∈{c} then
        cc=next
        goto 1_S
      error
end procedure
```

```
procedure A
O_A:  if cc∈{b} then
        cc=next
        goto 1_A
      if cc∈{c} then return
      error
1_A:  if cc∈{a} then
        call S;
      else error
2_A:  if cc∈{a} then
        cc=next
        goto 3_A
      error
3_A:  if cc∈{c} then return
      error
end procedure
```

**13 JULY 2016**

2. Consider the following control flow graph (*CFG*) of a simple program:




Answer the following questions:

(a) Find the *live variables* of the given *CFG*, by means of the the systematic method of the liveness flow equations: first write the equations, then iteratively solve them. Write the live variables on the *CFG* prepared on the next pages.

(b) Find the *reaching definitions* of the given *CFG*, by intuitively applying the definition of reaching definition. Write the reaching definitions on the *CFG* prepared on the next pages.

(c) (optional) Write the flow equations for the reaching definitions.

---

a)

| # | definitions | uses | input | output |
|---|---|---|---|---|
| 1 | a | / | | |
| 2 | b | a | | |
| 3 | c | / | | |
| 4 | / | b, c | | |
| 5 | a | a | | |
| 6 | / | b | | |

INSTRUCTIONS THAT WRITES ON VARIABLES

VARIABLES RED BY AN INSTRUCTION

1 IS A CONSTANT. NO NEEDS TO WRITE IT

WE ARE USING a, c TO DEFINE b

"read(c)" MEANS THAT THE PROGRAM IS DEFINING c

READ b AND c FOR THE COMPARISON

"write(b)" MEANS THAT THE PROGRAM READS b

"LIVE VARIABLES" ARE "VARIABLES DEFINED NOW AND USED LATER"

$input(p) = use(p) \cup (output(p) \setminus def(p))$

$output(p) = \bigcup_{q \in succ(p)} input(q)$ "WHAT COMES NEXT?"

| # | definitions | uses | input | output |
|---|---|---|---|---|
| 1 | a | / | $\emptyset \cup (output(1) \setminus \{a\})$ | input(2) |
| 2 | b | a | $\{a\} \cup (output(2) \setminus \{b\})$ | input(3) |
| 3 | c | / | output(3) $\setminus \{c\}$ | input(4) |
| 4 | / | b, c | $\{b,c\} \cup$ (output(4)) | input(5) U input(6) |
| 5 | a | a | $\{a\} \cup (output(5) \setminus \{a\})$ | input(2) |
| 6 | / | b | $\{b\} \cup$ (output(6)) | $\emptyset$ |

| # | def | use | in | out | OUTPUT | INPUT |
|---|---|---|---|---|---|---|
| 1 | a | / | $\emptyset \cup (output(1) \setminus \{a\})$ | in(2) | – | – |
| 2 | b | a | $\{a\} \cup (output(2) \setminus \{b\})$ | in(3) | – | a |
| 3 | c | / | output(3) $\setminus \{c\}$ | in(4) | – | – |
| 4 | / | b, c | $\{b,c\} \cup$ (output(4)) | in(5) U in(6) | – | b, c |
| 5 | a | a | $\{a\} \cup (output(5) \setminus \{a\})$ | in(2) | – | a |
| 6 | / | b | $\{b\} \cup$ (output(6)) | $\emptyset$ | – | b |

REPEAT UNTIL YOU GET STACKED IN A LOOP

| ※ | in | OUT | OUT | in | OUTPUT | input |
|---|---|---|---|---|---|---|
| 1 | ∅ ∪ (OUTPUT(3) \ {a}) | in(2) | — | — | a | ∅ |
| 2 | {a} ∪ (OUTPUT(2) \ {b}) | in(3) | — | a | ∅ | a |
| 3 | OUTPUT(3) \ {c} | in(4) | — | — | b, c | b |
| 4 | {b,c} ∪ (OUTPUT(4)) | in(5) ∪ in(6) | — | b, c | a, b | b, c |
| 5 | {a} ∪ (OUTPUT(5) \ {a}) | in(2) | — | a | a | a |
| 6 | {b} ∪ (OUTPUT(6)) | ∅ | — | b | ∅ | b |

| ※ | in | OUT | OUT | in | OUTPUT | input |
|---|---|---|---|---|---|---|
| 1 | ∅ ∪ (OUTPUT(2) \ {a}) | in(2) | a | ∅ | a | ∅ |
| 2 | {a} ∪ (OUTPUT(2) \ {b}) | in(3) | ∅ | a | b | a |
| 3 | OUTPUT(3) \ {c} | in(4) | b, c | b | a, b, c | a, b |
| 4 | {b,c} ∪ (OUTPUT(4)) | in(5) ∪ in(6) | a, b | b, c | a, b | a, b, c |
| 5 | {a} ∪ (OUTPUT(5) \ {a}) | in(2) | a | a | a | a |
| 6 | {b} ∪ (OUTPUT(6)) | ∅ | ∅ | b | ∅ | b |