Algorithm — LR(0) parser or LR-parser

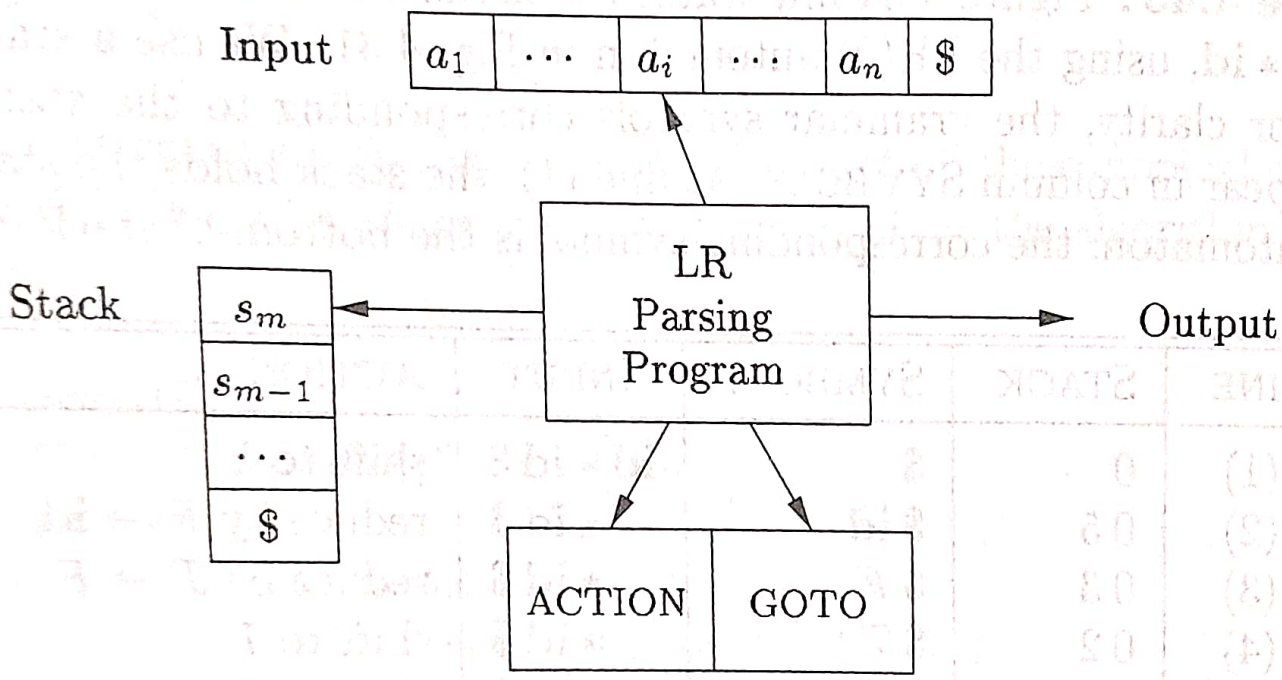


Figure 4.35: Model of an LR parser

Structure of the LR Parsing Table

The parsing table consists of two parts: a parsing-action function ACTION and a goto function GOTO.

1. The ACTION function takes as arguments a state i and a terminal a (or \$, the input endmarker). The value of ACTION$[i, a]$ can have one of four forms:

   (a) Shift j, where j is a state. The action taken by the parser effectively shifts input a to the stack, but uses state j to represent a.

   (b) Reduce $A \to \beta$. The action of the parser effectively reduces β on the top of the stack to head A.

   (c) Accept. The parser accepts the input and finishes parsing.

   (d) Error. The parser discovers an error in its input and takes some corrective action. We shall have more to say about how such error-recovery routines work in Sections 4.8.3 and 4.9.4.

2. We extend the GOTO function, defined on sets of items, to states: if GOTO$[I_i, A] = I_j$, then GOTO also maps a state i and a nonterminal A to state j.

Algorithm 4.44: LR-parsing algorithm.

INPUT: An input string w and an LR-parsing table with functions ACTION and GOTO for a grammar G.

OUTPUT: If w is in $L(G)$, the reduction steps of a bottom-up parse for w; otherwise, an error indication.

METHOD: Initially, the parser has s_0 on its stack, where s_0 is the initial state, and $w\$$ in the input buffer. The parser then executes the program in Fig. 4.36. □

```
let a be the first symbol of w$;
while(1) { /* repeat forever */
        let s be the state on top of the stack;
        if ( ACTION[s, a] = shift t ) {
                push t onto the stack;
                let a be the next input symbol;
        } else if ( ACTION[s, a] = reduce A → β ) {
                pop |β| symbols off the stack;
                let state t now be on top of the stack;
                push GOTO[t, A] onto the stack;
                output the production A → β;
        } else if ( ACTION[s, a] = accept ) break; /* parsing is done */
        else call error-recovery routine;
}
```

Main steps for constructing LR table:

1. Create augmented grammar.
2. Find closure properties (CLOSURE(), GOTO())
3. Create LR table (ACTION-GOTO)
4. check with an input string/ input string parsing.

LR(0) items will be used to construct LR(0) and SLR(1) parsers.

Example 1

$S \rightarrow AA$

$A \rightarrow aA \mid b$

like FIRST & FOLLOW we use CLOSURE() and GOTO here

Step 1: Create augmented grammar.

$S' \rightarrow S$

$S \rightarrow AA$

$A \rightarrow aA \mid b$

$S' \rightarrow \cdot S$ no apply closure.

[closure include all the productions of S.]

$S' \rightarrow .S$

$S \rightarrow .AA$

$A \rightarrow .aA \mid .b$

now we can stop.

$S \rightarrow \cdot AA$

any production with a '.' in the RHS is called item.

Here, we've LR(0) items.

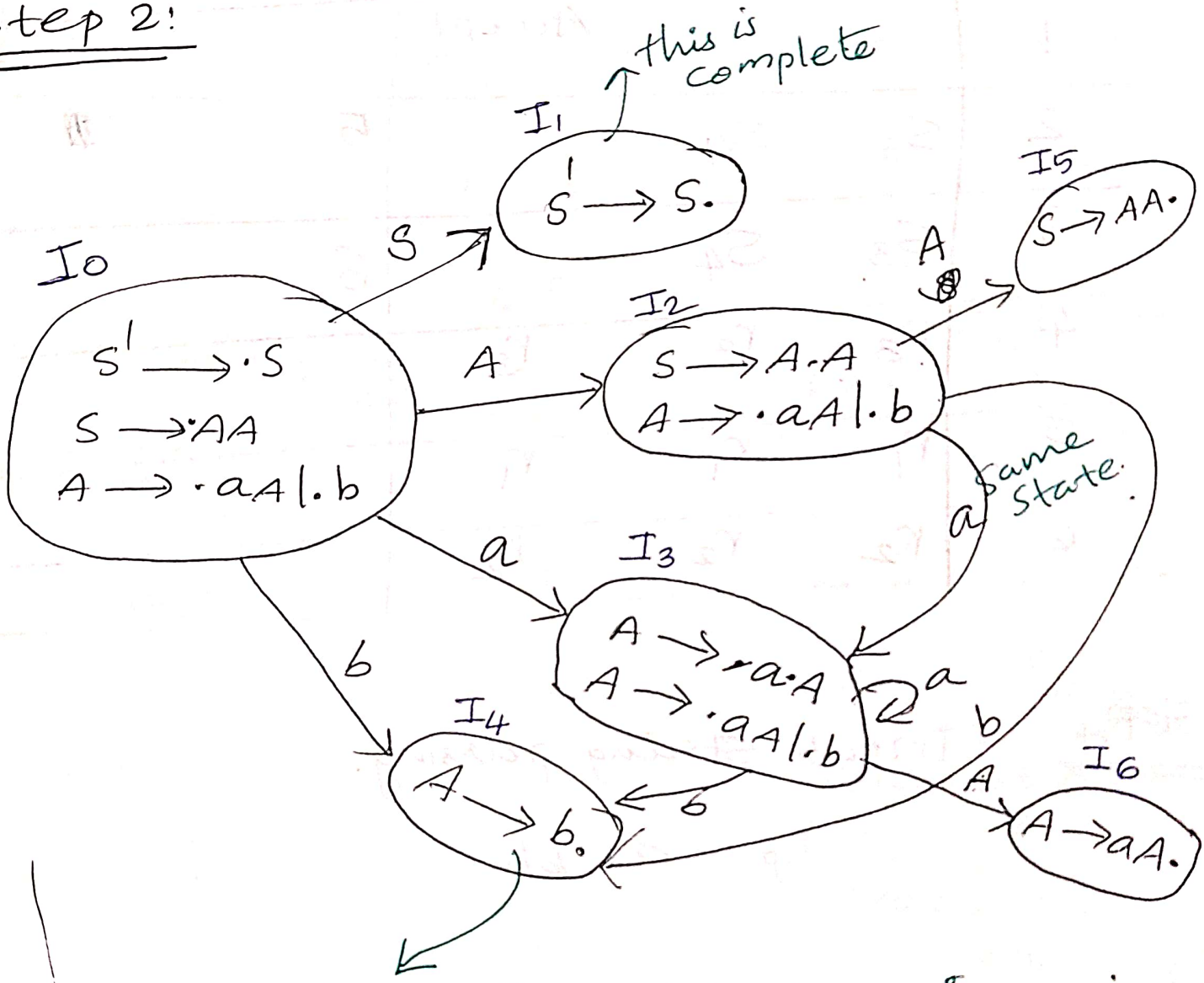
$S \rightarrow A.A$

$S \rightarrow AA.$

In this, at RHS, we've seen everything, so as to reduce AA to S.

Next is GOTO. (its simply like a DFA).

Step 2:



whenever . is on the RHS, that item is complete.

$S' \rightarrow S$

$S \rightarrow AA$ —— ①

$A \rightarrow aA$ —— ②

$A \rightarrow b$ —— ③.

(Number the production)

Step3:

	ACTION			GOTO	
	a	b	$	A	S
I_0	S_3	S_4		2	1
1			Accept		
2	S_3	S_4		5	
3	S_3	S_4		6	
4	r_3	r_3	r_3		
5	r_1	r_1	r_1		
6	r_2	r_2	r_2		

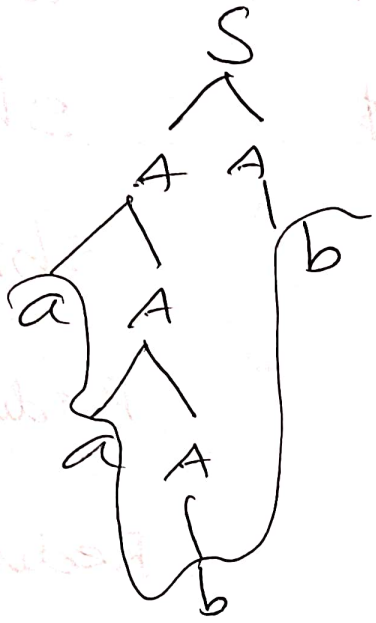
Step4: input string parsing.

I/p : aabb

Stack	Input	Action
$0	aabb$	shift with 3
$0a3	abb$	shift s_3
$0a3a3	bb$	shift s_4
$0a3a3b4 (poping 2 char.)	b$	Reduce - r_3 3rd prod $A \to b$
$0a3a3A6 (pop.)	b$	Reduce r_2 2rd prod. $A \to aA$
$0a3A6	b$	Reduce. r_2 $A \to aA$.
$0A2	b$	shift s_4
$0A2b4	$	Reduce r_3 $A \to b$
$0A2A5	$	Reduce r_1 $S \to AA$.
$0S1	$	Accept.

Step 5:

Parse tree



② $S \rightarrow SA/A$

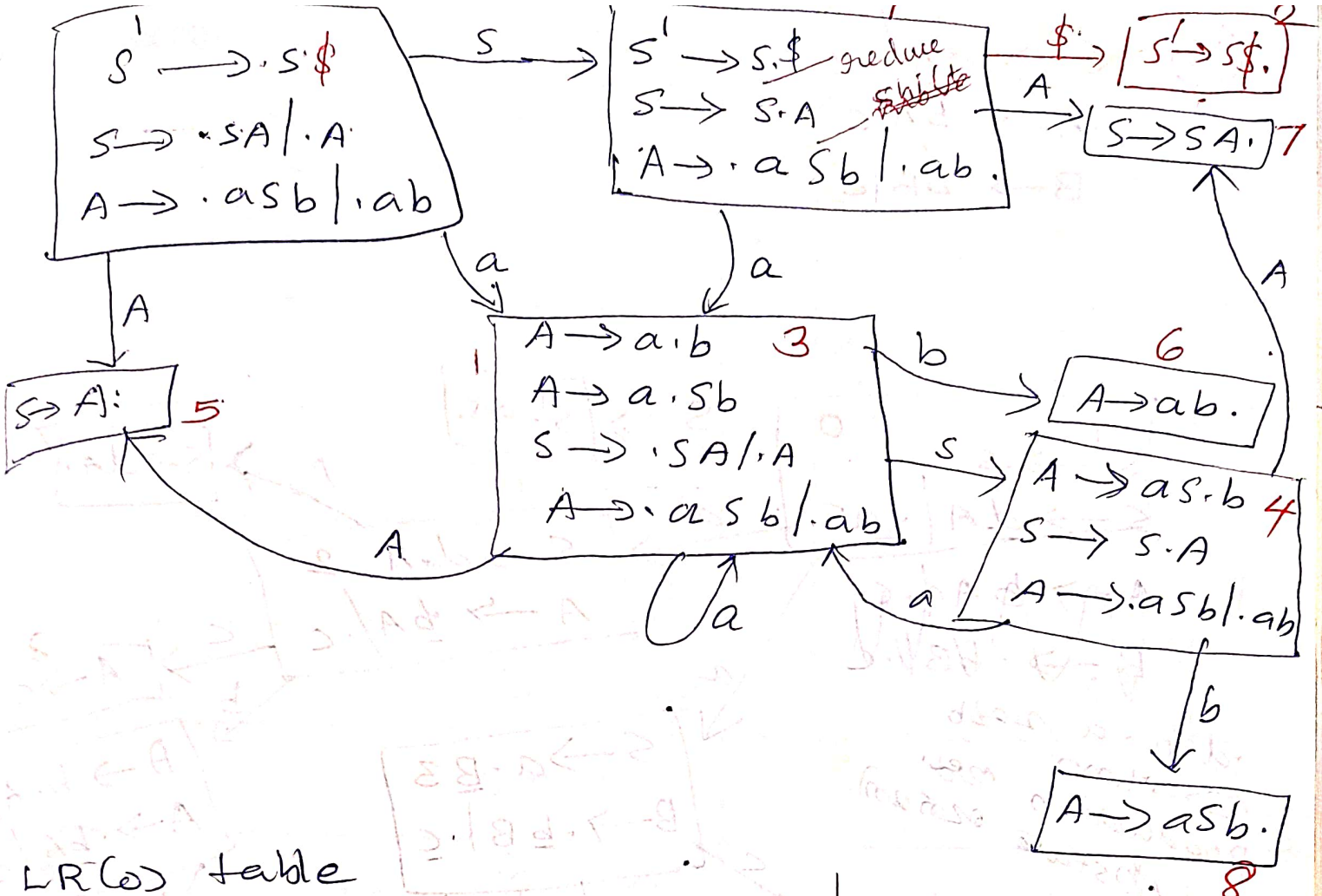
$A \rightarrow aSb/ab$

augment grammer

$S' \rightarrow .S$

$S \rightarrow .SA/.A$

$A \rightarrow .aSb/.ab$



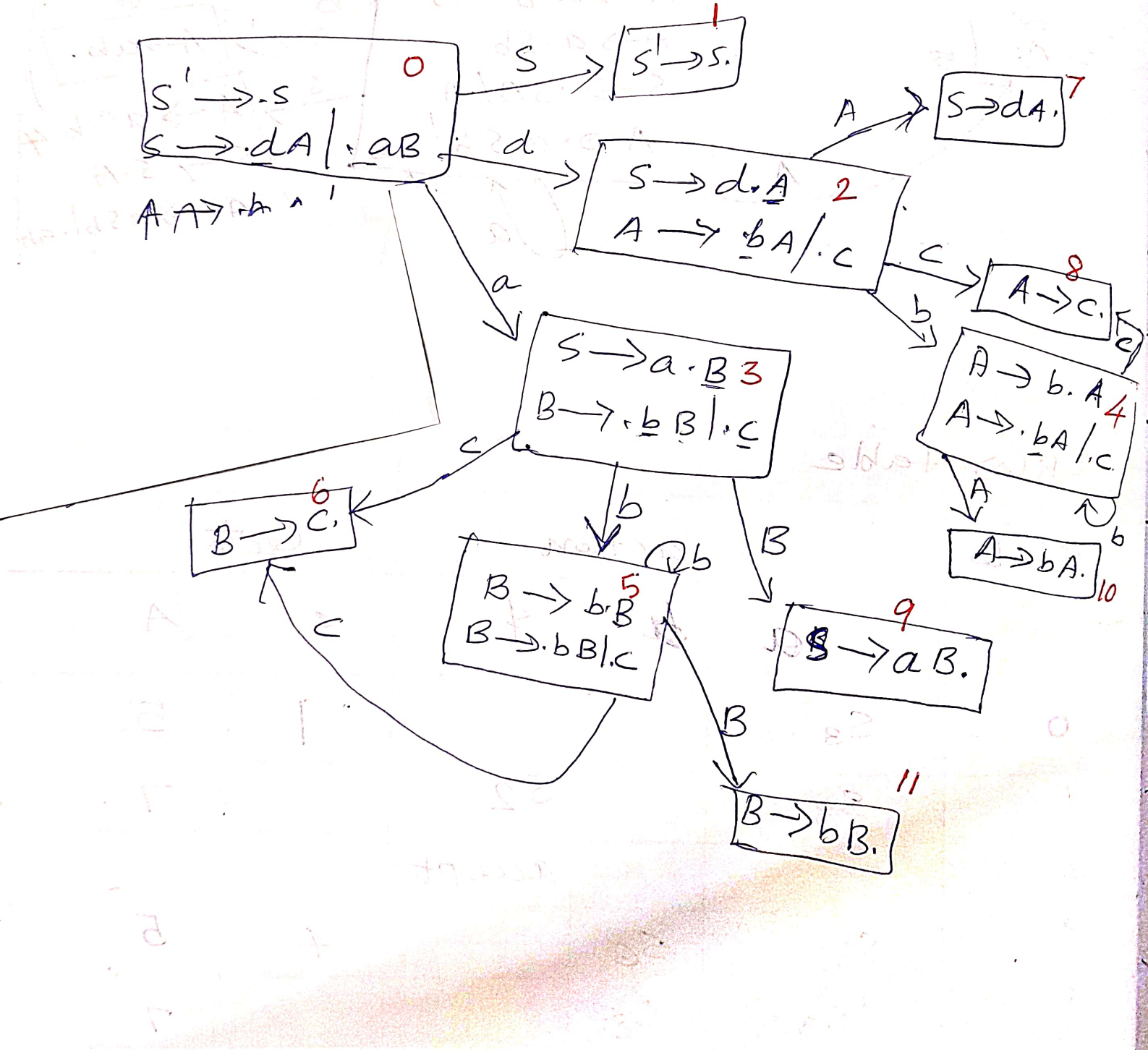
LR(0) table

state	action			Goto	
	a	b	$	S	A
0	S_3			1	5
1	S_3		S_2		7
2			accept		
3	S_3	S_6		4	5
4	S_3	S_8			7
5	r_2	r_2	r_2		
6	r_4	r_4	r_4		
7	r_1	r_1	r_1		
8	r_3	r_3	r_3		

③ $S \rightarrow dA^1 \mid aB^2$

$A \rightarrow bA^3 \mid c^4$

$B \rightarrow bB^5 \mid c^6$



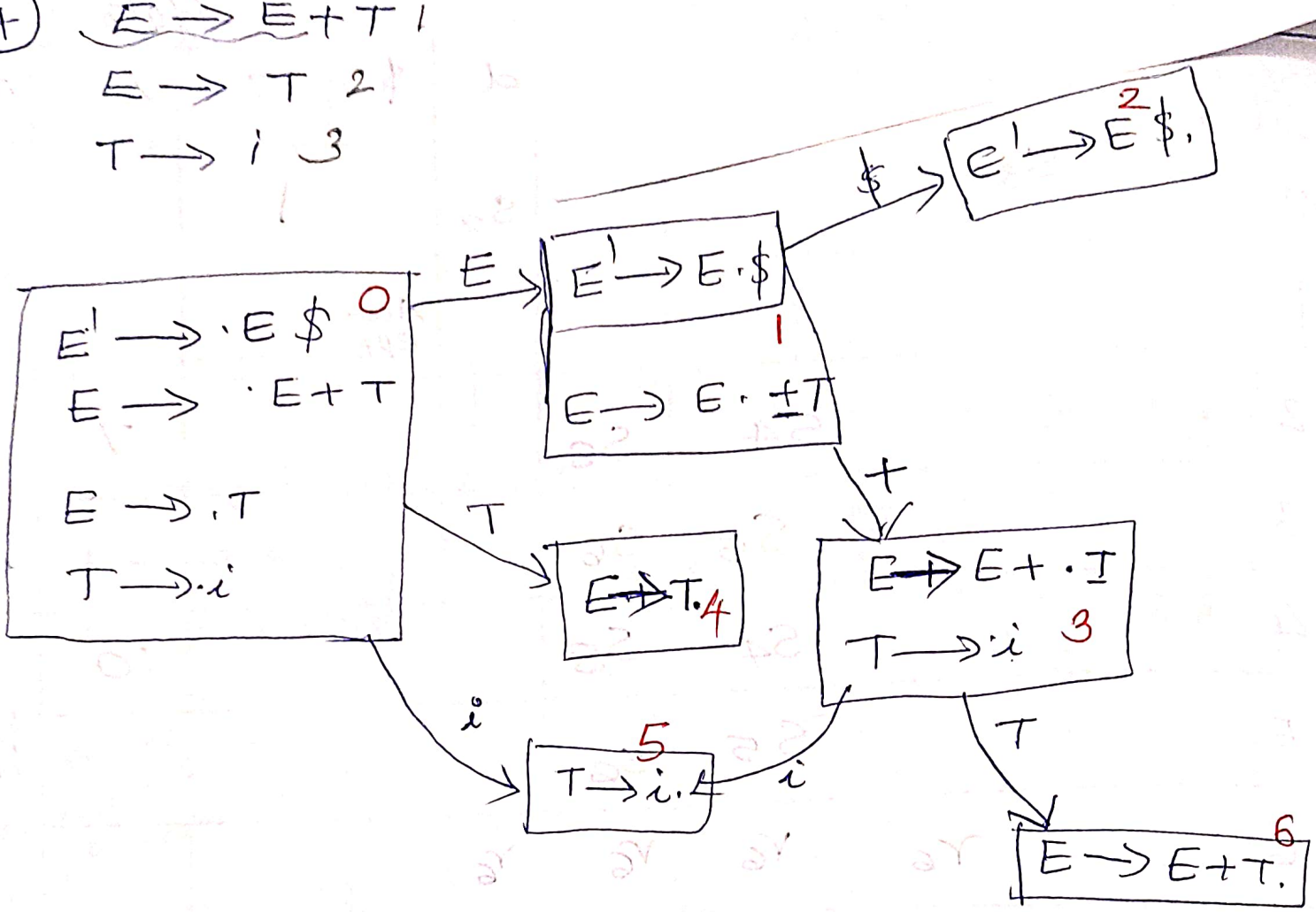
State ~~Stack~~	action					goto		
	a	b	c	d	$	S'	A	B
0	S_3			S_2		1		
1					Accept			
2		S_4	S_8				7	
3		S_5	S_6					9
4		S_4	S_8				10	
5		S_5	S_6					11
6	r_6	r_6	r_6	r_6				
7	r_1	r_1	r_1	r_1				
8	r_4	r_4	r_4	r_4				
9	r_2	r_2	r_2	r_2				
10	r_3	r_3	r_3	r_3				
11	r_5	r_5	r_5	r_5				

LR(0) table.

(4) $E \to E+T$ 1

$E \to T$ 2

$T \to i$ 3

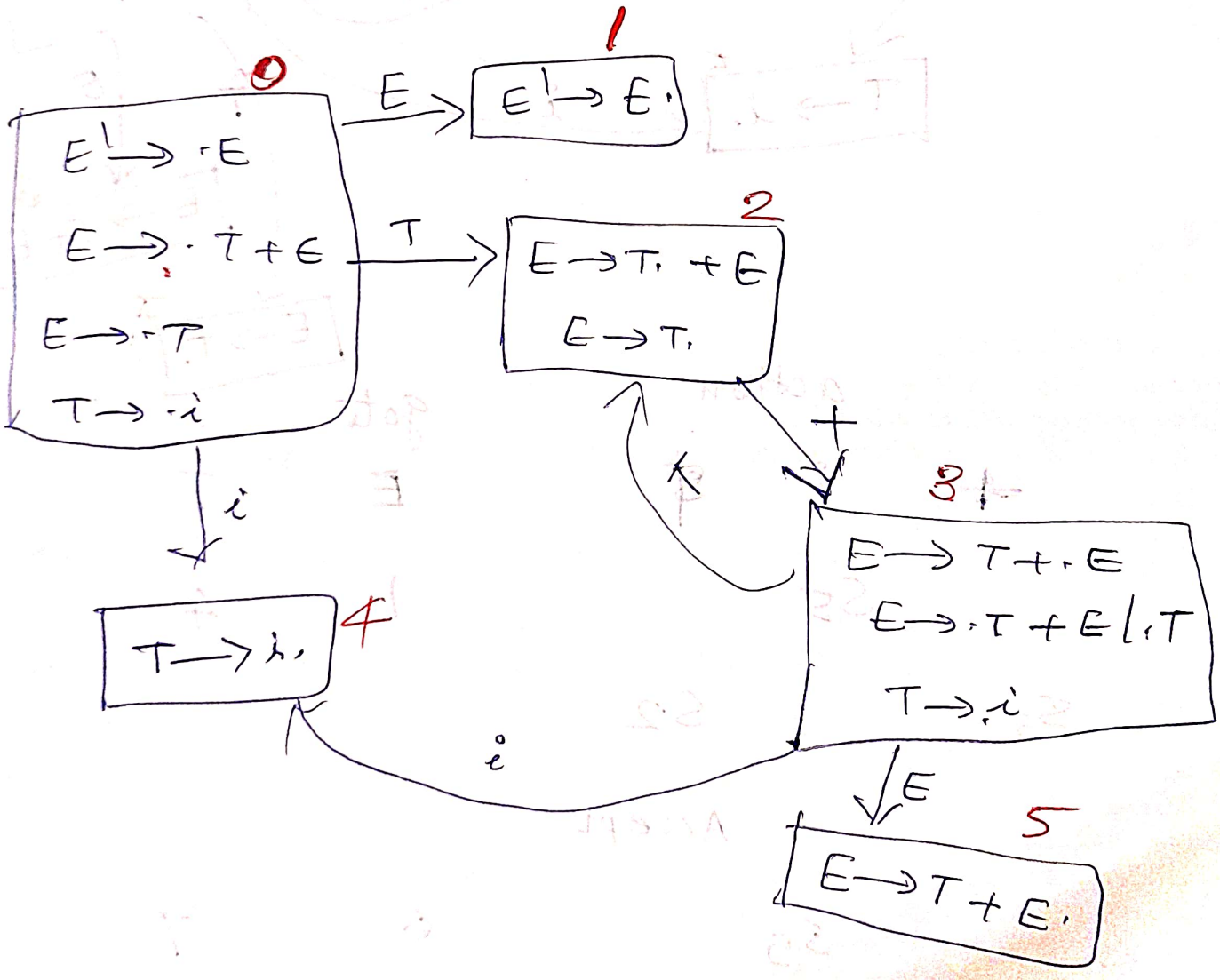


state	action			goto	
	~~+~~	i	$	E	T
0		S_5		1	4
1	S_3		S_2		
2			accept		
3		S_5			6
4	r_2	r_2	r_2		
5	r_3	r_3	r_3		
6	r_1	r_1	r_1		

HW

$E \to T + E \mid T$

$T \to i$



	Action			Goto	
	+	i	$	E	T
0		s_4		1	2
1	conflict came. Accept				
2	s_3/r_2	r_2	r_2		
3		s_4		5	
4	r_3	r_3	r_3		
5	r_1	r_1	r_1		

shift-reduce conflict came.

So the given grammar is not SLR(0).

S-R, R-R } two types of conflicts can come.

LR(0) is least powerful.

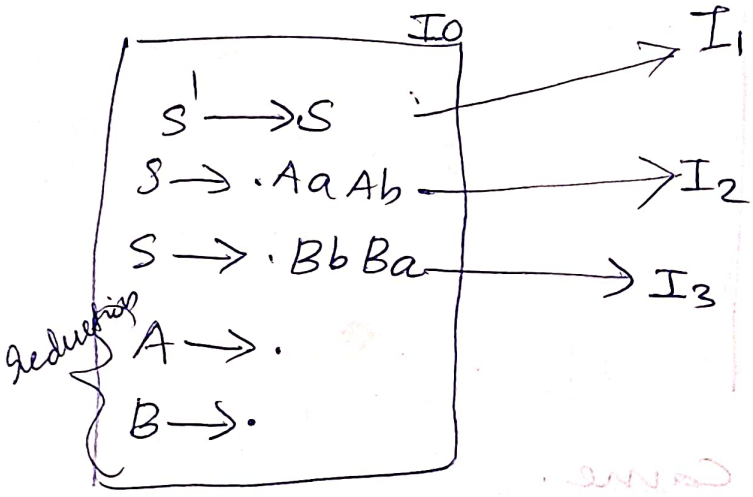
(6) check LR(0) or not.

$S \rightarrow AaAb$ 1

$S \rightarrow BbBa$ 2

$A \rightarrow \varepsilon$ 3

$B \rightarrow \varepsilon$ 4



	Action			goto
	a	b	$	
0	r3/r4	r3/r4	r3/r4	
1				
2				

R-R conflict came.

So not LR(0) grammer.