به نام خدا

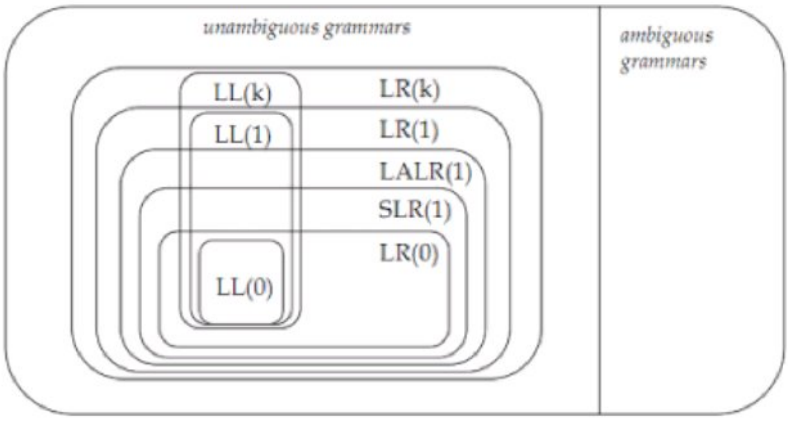
حدیث غفوری 9825413

سوال 1

عبارت زیر نادرست است.

$$LR(0) \subset SLR(1) \subset LALR(1) \subset LR(2) \subset LR(1) \subset LR(k)$$

زیرا طبق شکل زیر که روش های مختلف تجزیه را دسته بندی کرده LR(1) زیرمجموعه LR(K) قرار میگیرد پس نمیتوان گفت لزوما LR(2) زیرمجموعه ی LR(1) است.



بین این روش ها روابط زیر برقرار است.

- $LR(0) \subset SLR(1) \subset LALR(1) \subset LR(1)$
- $SLR(k) \subset LALR(k) \subset LR(k)$
- $SLR(1) \subset SLR(2) \subset \cdots \subset SLR(k)$
- $LALR(1) \subset LALR(2) \subset \cdots \subset LALR(k)$
- $LR(0) \subset LR(1) \subset LR(2) \subset \cdots \subset LR(k) \subset \cdots \subset LR$

سوال 2

So, it is true that if there is an LR(k) grammar for a language, then there is also an LR(1) grammar. But it is not the same grammar (unless k is 1) and it will therefore not produce the same parse tree.

It is possible to mechanically transform the LR(k) grammar into an LR(1) grammar in a way that the original parse tree can be recovered, but it is not a procedure you want to try by hand.

SLR Parser

SLR represents **"Simple LR Parser"**. It is very easy and cost-effective to execute. The SLR parsing action and goto function from the deterministic finite automata that recognizes viable prefixes. It will not make specificially defined parsing action tables for all grammars but does succeed on several grammars for programming languages. Given a grammar G. It augment G to make G', and from G' it can construct C, the canonical collection of a set of items for G'. It can construct ACTION the parsing action function, and GOTO, the goto function, from C using the following simple LR Parsing table construction technique. It needed us to understand FOLLOW (A) for each non-terminal A of a grammar.

LR(1) , LR(K)

LR parsers are deterministic; they produce a single correct parse without guesswork or backtracking, in linear time. This is ideal for computer languages, but LR parsers are not suited for human languages which need more flexible but inevitably slower methods. Some methods which can parse arbitrary context-free languages have worst-case performance of $O(n^3)$ time. Other methods which backtrack or yield multiple parses may even take exponential time when they guess badly.
In computer science, a **canonical LR parser** or **LR(1) parser** is an LR(k) parser for *k=1*, i.e. with a single lookahead terminal. The special attribute of this parser is that any LR(k) grammar with *k>1* can be transformed into an LR(1) grammar.[1] However, back-substitutions are required to reduce k and as back-substitutions increase, the grammar can quickly become large, repetitive and hard to understand. LR(k) can handle all deterministic context-free languages. In the past this LR(k) parser has been avoided because of its huge memory requirements in favor of less powerful alternatives such as the LALR and the LL(1) parser.

LALR Parser

LALR Parser is Look Ahead LR Parser. It is intermediate in power between SLR and CLR parser. It is the compaction of CLR Parser, and hence tables obtained in this will be smaller than CLR Parsing Table.

For constructing the LALR (1) parsing table, the canonical collection of LR (1) items is used. In the LALR (1) parsing, the **LR (1)** items with the equal productions but have several look ahead are grouped to form an individual set of items. It is frequently the similar as **CLR (1)** parsing except for the one difference that is the parsing table.

The overall structure of all these LR Parsers is the same. There are some common factors such as size, class of context-free grammar, which they support, and cost in terms of time and space in which they differ.

Canonical LR(1) parsers have the practical disadvantage of having enormous memory requirements for their internal parser-table representation. In 1969, Frank DeRemer suggested two simplified versions of the LR parser called LALR and SLR. These parsers require much less memory than Canonical LR(1) parsers, but have slightly less language-recognition power.[5] LALR(1) parsers have been the most common implementations of the LR Parser.

S. no.	SLR Parsers	Canonical LR Parsers (LR(1))	LALR Parsers
1	SLR parser are easiest to implement.	CLR parsers are difficult to implement.	LALR parsers are difficult to implement than SLR parser but less than CLR parsers.
2.	SLR parsers make use of canonical collection of LR(0) items for constructing the parsing tables.	CLR parsers are uses LR(1) collection of items for constructing the parsing tables part.	LALR parsers LR(1) collection , items with items having same core merged into a single itemset.
3	SLR parsers don't do any lookahead i.e., they lookahead zero	CLR parsers lookahead one symbol.	LALR parsers lookahead one symbol.
4.	SLR parsers are cost effective to construct in terms of time and space.	CLR parsers are expensive to construct in terms – of time and space.	The cost of constructing LALR parsers is i intermediate between SLR and CLR parser.
5.	SLR parsers have hundreds of states.	CLR parses have thousands of states.	LALR parsers have hundreds of state and is same as number states in SLR parsers.

S. no.	SLR Parsers	Canonical LR Parsers (LR(1))	LALR Parsers
6.	SLR parsers uses FOLLOW information to guide reductions.	CLR parsers uses lookahead symbol to guide reductions.	LALR parsers uses lookahead symbol to guide reductions.
7.	SLR parsers may fail to I produce a table for certain class of grammars on which ether succeed.	CLR parser works on very large class of grammar.	LALR parser works on very large class grammars.
8.	Every SLR(1) grammar is LR(1) grammar and LALR(1).	Every LR(1) grammar may not be SLR(1) grammar.	Every LALR(1) grammar may not be SLR(1) but every LALR(1) grammar is LR(1) grammar.
9.	A shift-reduce or reduce-reduce conflict may arise in SLR parsing table.	A shift-reduce or reduce-reduce conflicted may arise but chances are less than that in SLR parsing tables.	A shift-reduce conflict can not arise but a reduce-reduce conflict may arise.
10.	SLR parser is least powerful.	A CLR parsers is most powerful among the family canonical of bottom-up parsers.	A LALR parser is intermediate in power between SLR and LR parser.

سوال ۳ — گرامر اول

$S \rightarrow ABdD \mid bD$ (1) (2)

$A \rightarrow aA \mid \varepsilon$ (3) (4)

$B \rightarrow b$ (5)

$D \rightarrow ab \mid \varepsilon$ (6) (7)

first(B) = {b}

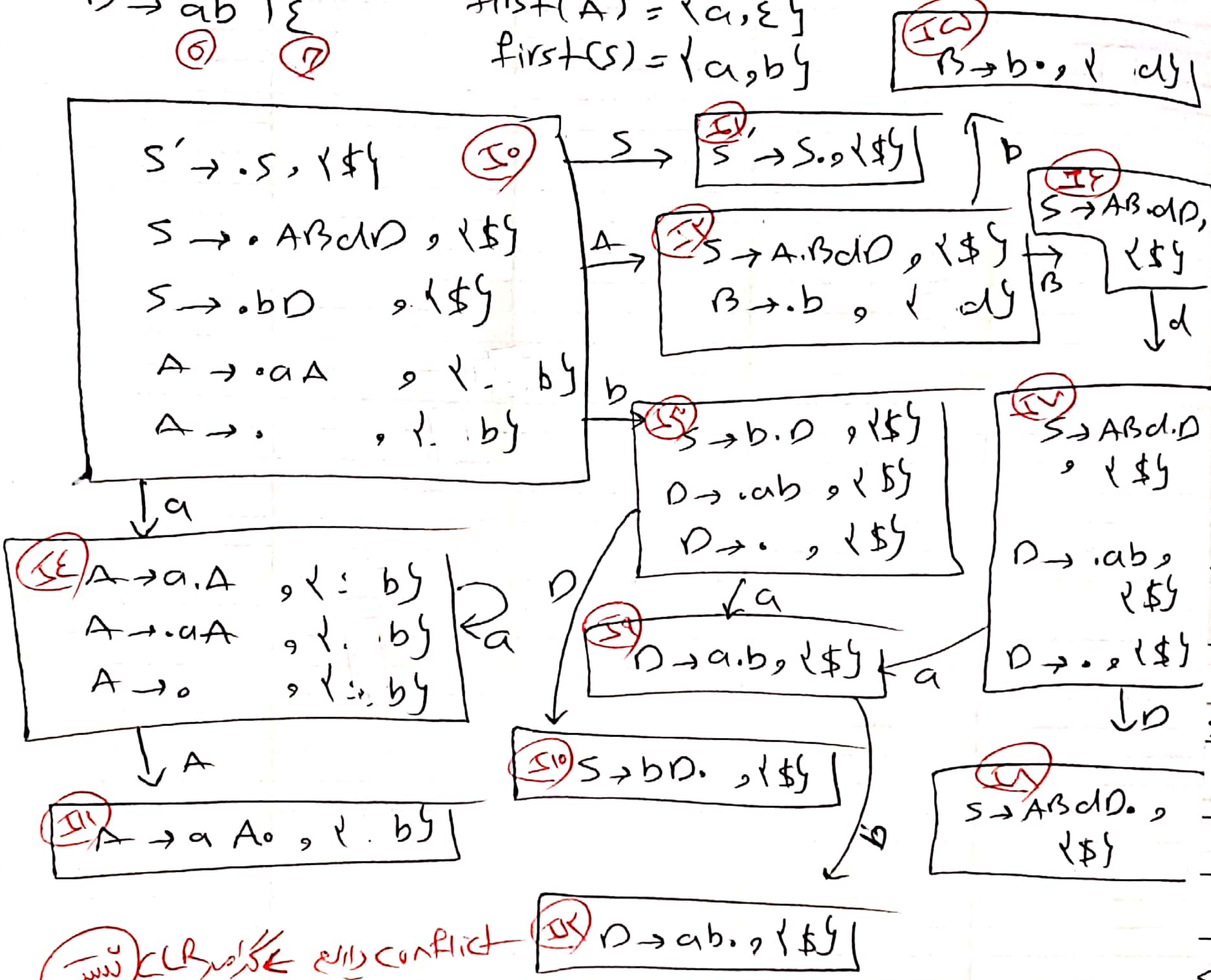
first(D) = {a, ε}

first(A) = {a, ε}

first(S) = {a, b}

رسم گراف CLR

(اتوماتای LR(1))



تست: گرامر CLR نیست ← دارای conflict

جدول پارسینگ CLR

state	Action a	Action b	Action d	Action $	goto S	goto A	goto B	goto D
0	S4	S3/r4			1	2		
1				accept				
2		S5					6	
3	S9			r7				10
4	S4	r4				11		
5			r5					
6			S7					
7	S9			r7				8
8				r1				
9		S12						
10				r2				
11		r3						
12				r6				

ادامه ۳_۶

درصورتی که جدول تجزیه بدست آمده از اتوماتای LR(1)، conflict نداشته باشد.

conflict رخ داده در state ۰ ←

S → .bD , {$}
A → . , {$, b}

conflict s/r دارد که اینجا هم handle، هم شیفت به صورت این که {$, b} ∩ first (bD) ≠ ∅ بشود

= {b}

چون کانفلیکت s/r داریم ← گرامر CLR نیست

در گرام مرادل، چون در جدول CLR (1)، conflict داشتیم ← گرامر CLR نبود
وچون هسته‌های مشابه در state ای نیستیم نیست ← اتوماتای LALR(1) این گرامر
همان اتوماتای CLR(1) می‌شود ← جدول هم همان می‌شود ← گرامر LALR(1) هم

نیست

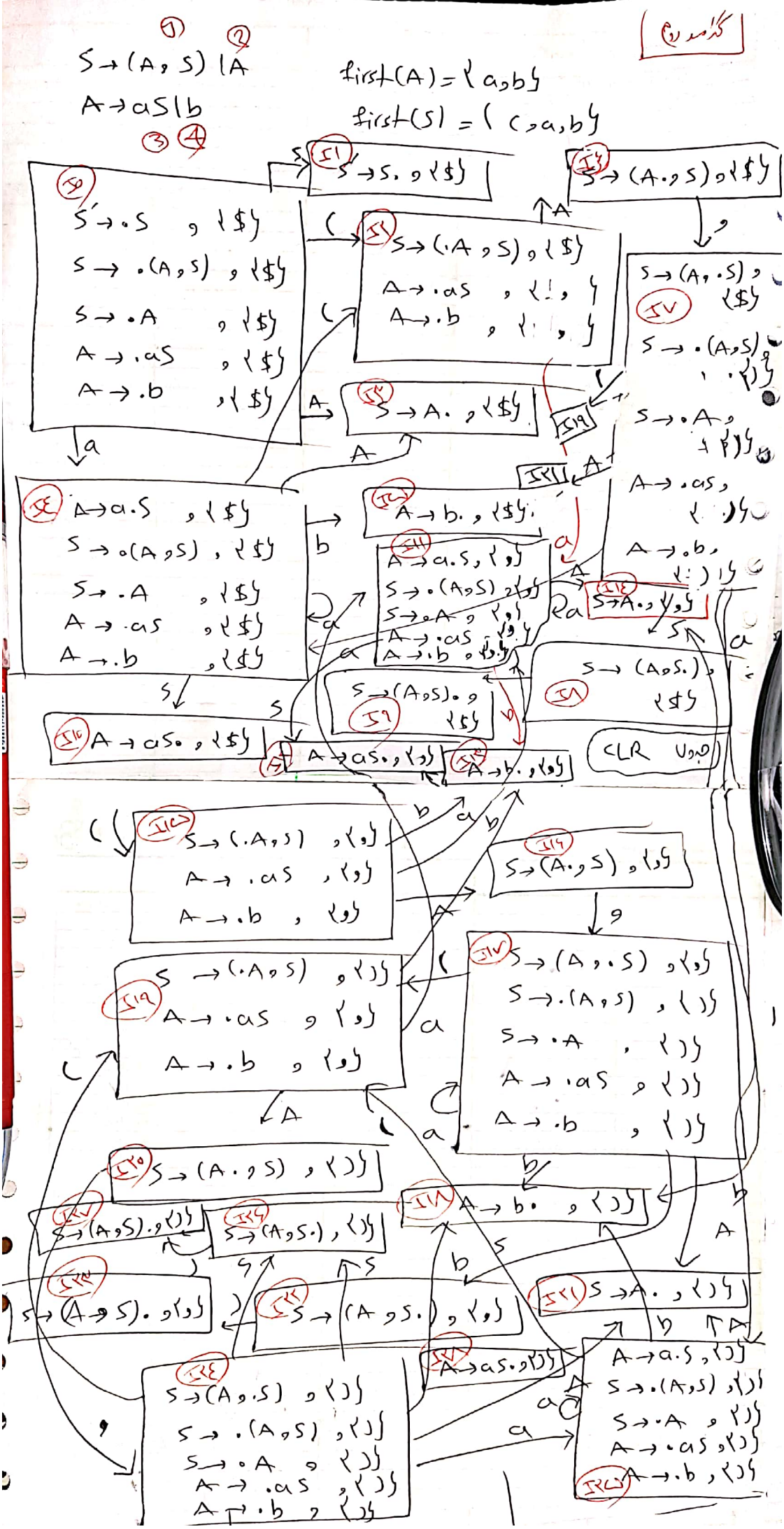
گرامر (۲)

① $S \to (A, S)$ ② $S \to A$

③ $A \to aS$ ④ $A \to b$

$first(A) = \{a, b\}$

$first(S) = \{(, a, b\}$



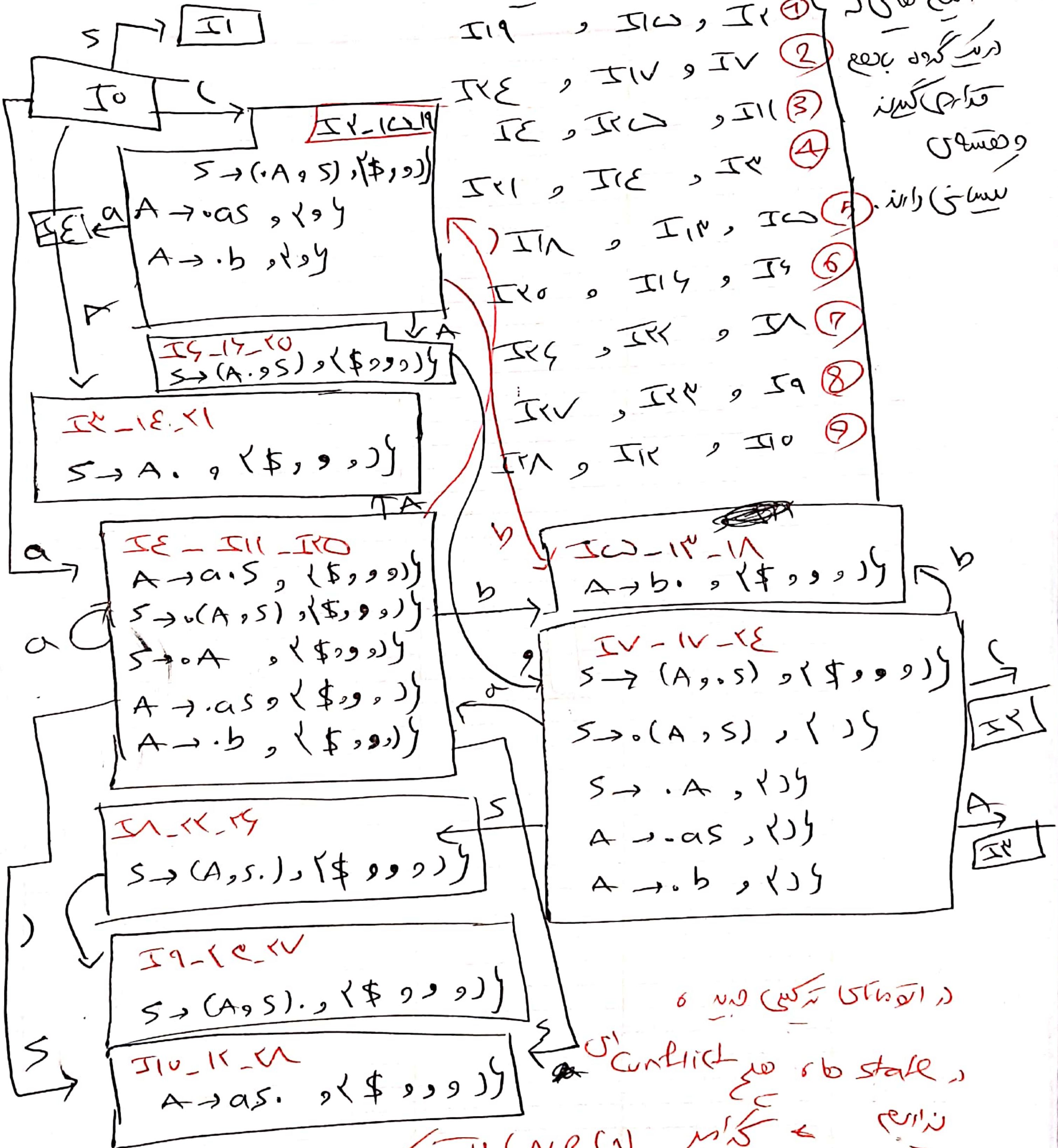
State	Action					goto	
	(	)	a	b	$	A	S
0	s2		s3	s4		5	1
1					accept		
2			s3	s4		6	
3				r2			
4	s2		s3	s4		7	10
5				r4			
6	s9						
7		s8	s13	s14		11	8
8		r5					
9				r1			
10				r3			
11	s15		s16	s17		18	12
12		r6					
13		r4					
14		r2					
15			s16	s17		19	
16	s9						
17		s8	s13	s14		21	22
18							
19			s16	s17		20	
20		r6					
21		r4					
22		r2					
23				r1			
24		r3					
25	s9		s13	s14		21	24
26	s9		s13	s14		21	27
27		r5					
28		r1					
29		r3					

گام سوم (۳)

- روش LALR(1) : در این روش حالت‌هایی که هسته یکسان دارند را یکی می‌کنیم و Lookahead هایشان را با هم اجتماع می‌کنیم.

1) I۲ , I۱۵ و I۱۹
2) I۷ و I۱۷ و I۲۴
3) I۱۱ و I۲۵ و I۴
4) I۳ و I۱۴ و I۲۱
5) I۵ , I۱۳ و I۱۸
6) I۶ و I۱۶ و I۲۰
7) I۸ و I۲۲ و I۲۶
8) I۹ و I۲۳ و I۲۷
9) I۱۰ و I۱۲ و I۲۸

استیت‌هایی که در یک گروه با هم قرار می‌گیرند و هسته یکسانی دارند.



در اتوماتای ترکیبی جدید در state ها هیچ conflict ای نداریم ← گرامر LALR(1) است ✓

جدول LALR(1) گرامر (د)

اومدیم اینجا استیت‌های ترتیبی را به عنوان state در جدول درنظر گرفتیم

state	Action						goto	
	(	,	)	a	b	$	A	S
0	s2			s4	s5		3	1
1						accept		
2				s4	s5		6	
3		r2	r2			r2		
4	s2			s4	s5		3	10
5		r4	r4			r4		
6		s7						
7	s2			s4	s5		3	8
8			s9					
9		r1	r1			r1		
10		r3	r3			r3		

هیچ Conflict ای نداریم ← LALR(1) است.

گرامر CLR(1) هم هست چون طبق جدولی که برای CLR الگوریتم کشیدیم هیچ کانفلیکتی نداشتیم