Chapter 4

Syntax Analysis

Part 2

BOTTOM-UP PARSING

Bottom-Up Parsing

mansoori@shirazu.ac.ir

- Constructing parse tree for input string beginning at the leaves (the bottom) and working up towards the root (the top)

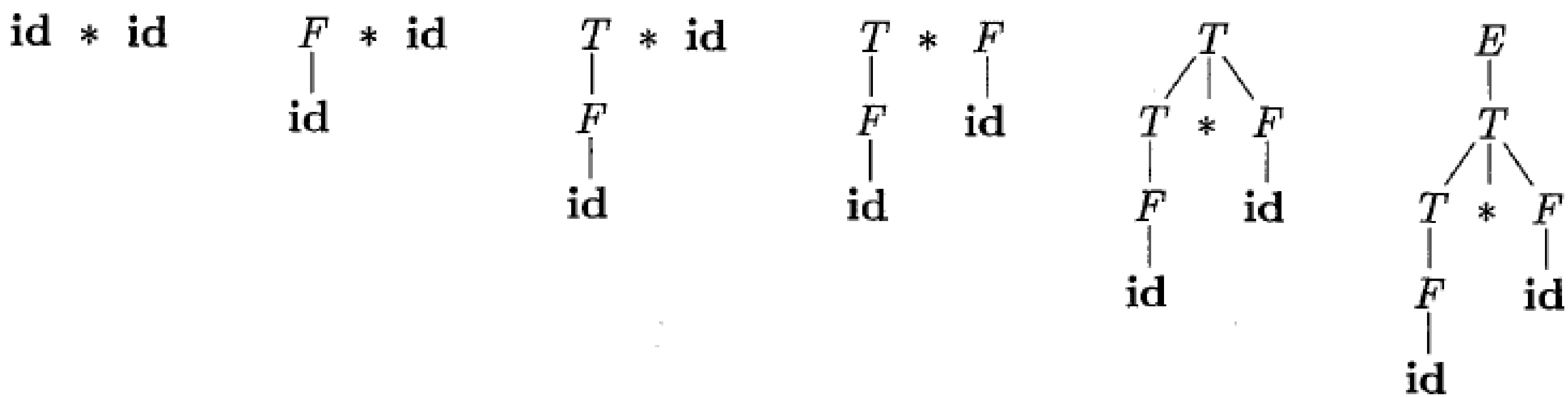
- Shift-reduce parsing

- LR methods (Left-to-right, Rightmost derivation)
  - Simple LR (SLR)
  - Canonical LR (CLR)
  - Look-Ahead LR (LALR)

Shift-Reduce Parsing

mansoori@shirazu.ac.ir

- Reducing string *w* to start symbol of grammar (reverse of rightmost derivation)
- At each step, reducing a specific substring matching the body of a production (a handle) to its head nonterminal



$$E \Rightarrow T \Rightarrow T * F \Rightarrow T * \mathbf{id} \Rightarrow F * \mathbf{id} \Rightarrow \mathbf{id} * \mathbf{id}$$

Shift-Reduce Parsing

mansoori@shirazu.ac.ir

Grammar:

$S \rightarrow \mathbf{a}\ A\ B\ \mathbf{e}$

$A \rightarrow A\ \mathbf{b}\ \mathbf{c} \mid \mathbf{b}$

$B \rightarrow \mathbf{d}$

Reducing a sentence:

$\mathbf{a}\ \underline{\mathbf{b}}\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$

$\mathbf{a}\ \underline{A\ \mathbf{b}\ \mathbf{c}}\ \mathbf{d}\ \mathbf{e}$

$\mathbf{a}\ A\ \underline{\mathbf{d}}\ \mathbf{e}$

$\underline{\mathbf{a}\ A\ B\ \mathbf{e}}$

S

Shift-reduce corresponds to a rightmost derivation:

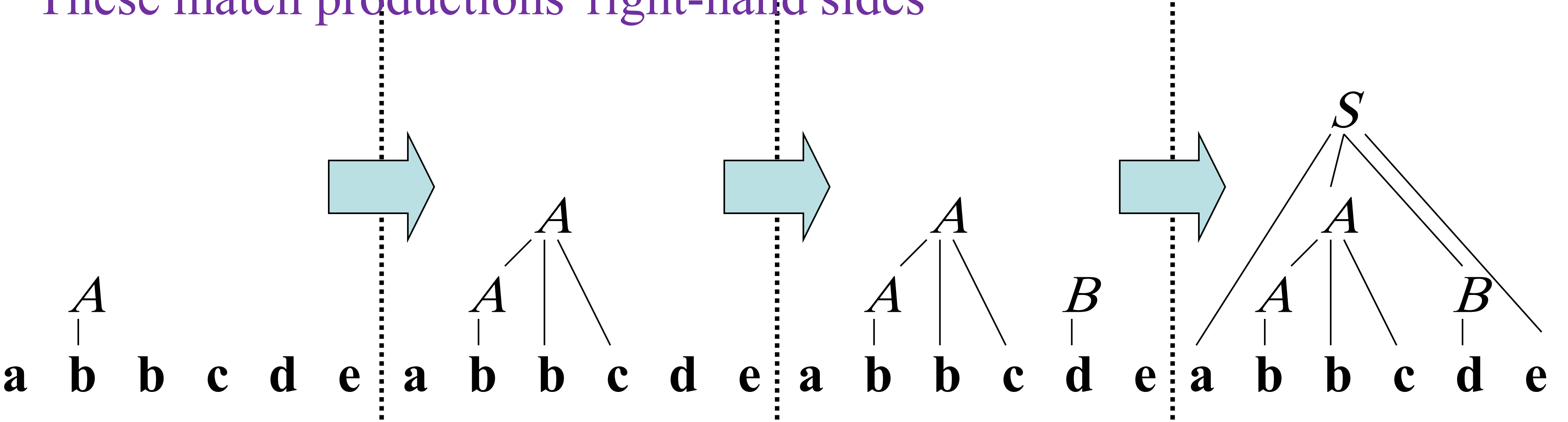
$S \Rightarrow_{rm} \mathbf{a}\ A\ B\ \mathbf{e}$

$\Rightarrow_{rm} \mathbf{a}\ A\ \mathbf{d}\ \mathbf{e}$

$\Rightarrow_{rm} \mathbf{a}\ A\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$

$\Rightarrow_{rm} \mathbf{a}\ \mathbf{b}\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$

These match productions' right-hand sides



Handle Pruning

mansoori@shirazu.ac.ir

- **Key decisions**: when to reduce, what production to apply
- **Handle**: a substring of grammar symbols in a right-sentential form that matches a right-hand side of a production

Grammar:

$S \rightarrow \mathbf{a}\ A\ B\ \mathbf{e}$

$A \rightarrow A\ \mathbf{b}\ \mathbf{c} \mid \mathbf{b}$

$B \rightarrow \mathbf{d}$

$\mathbf{a}\ \underline{\mathbf{b}}\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$

$\mathbf{a}\ \underline{A\ \mathbf{b}\ \mathbf{c}}\ \mathbf{d}\ \mathbf{e}$

$\mathbf{a}\ A\ \underline{\mathbf{d}}\ \mathbf{e}$

$\underline{\mathbf{a}\ A\ B\ \mathbf{e}}$

S

Handle

$\mathbf{a}\ \underline{\mathbf{b}}\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$

$\mathbf{a}\ A\ \underline{\mathbf{b}}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$

$\mathbf{a}\ A\ A\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$

… ?

NOT a handle

NOT a sentential form

Sentential form	Handle	Reduction
$\mathbf{id}_1 * \mathbf{id}_2$	$\mathbf{id}_1$	$F \rightarrow \mathbf{id}$
$F * \mathbf{id}_2$	F	$T \rightarrow F$
$T * \mathbf{id}_2$	$\mathbf{id}_2$	$F \rightarrow \mathbf{id}$
$T * F$	$T * F$	$T \rightarrow T * F$

Stack Implementation of Shift-Reduce Parsing

mansoori@shirazu.ac.ir

- A stack holds grammar symbols
- An input buffer holds the rest of string to be parsed
- Handle always appears at top of stack
- Initially:

STACK	INPUT
$	w $

- Parser repeatedly:
  - Shifts zero or more input symbols (tokens) onto the stack until a handle appears on stack
  - Then, reduces handle to head of production
- Finally:

STACK	INPUT
$ S	$

Stack Implementation of Shift-Reduce Parsing

mansoori@shirazu.ac.ir

- Four possible actions of shift-reduce parser:
  (1) Shift: shifts the next token onto top of stack
  (2) Reduce: locates handle at stack top and reduces it
  (3) Accept: announces successful completion of parsing
  (4) Error: discovers syntax error and calls error recovery routine

Grammar:

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow \textbf{id}$

Stack	Input	Action
$\$$	**id+id*id$**	shift
$\$\underline{\textbf{id}}$	**+id*id$**	reduce $E \rightarrow \textbf{id}$
$\$E$	**+id*id$**	shift
$\$E+$	**id*id$**	shift
$\$E+\underline{\textbf{id}}$	***id$**	reduce $E \rightarrow \textbf{id}$
$\$E+E$	***id$**	shift (or reduce?)
$\$E+E*$	**id$**	shift
$\$E+E*\underline{\textbf{id}}$	**$**	reduce $E \rightarrow \textbf{id}$
$\$E+\underline{E*E}$	**$**	reduce $E \rightarrow E * E$
$\$\underline{E+E}$	**$**	reduce $E \rightarrow E + E$
$\$E$	**$**	accept

How to resolve conflicts?

Find handles to reduce

Conflicts during Shift-Reduce Parsing

mansoori@shirazu.ac.ir

- Conflict types:
  - Shift-reduce conflict
  - Reduce-reduce conflict

- Conflicts caused by:
  - The limitations of the LR parsing method (even when the grammar is unambiguous)
  - Ambiguity of the grammar

Shift-Reduce Conflict in Shift-Reduce Parsing

mansoori@shirazu.ac.ir

Ambiguous grammar:

$S \rightarrow$ **if** E **then** S

| **if** E **then** S **else** S

| **other**

Stack	Input	Action
$…	…$	…
$…**if** *E* **then** *S*	**else**…$	shift or reduce?

Resolve in favor of shift, so **else** matches closest **if**

mansoori@shirazu.ac.ir

Reduce-Reduce Conflict in Shift-Reduce Parsing

$$
\begin{array}{rcl}
\mathit{stmt} & \rightarrow & \mathbf{id}\ (\ \mathit{parameter\_list}\ ) \\
\mathit{stmt} & \rightarrow & \mathit{expr}\ \mathbf{:=}\ \mathit{expr} \\
\mathit{parameter\_list} & \rightarrow & \mathit{parameter\_list}\ ,\ \mathit{parameter} \\
\mathit{parameter\_list} & \rightarrow & \mathit{parameter} \\
\mathit{parameter} & \rightarrow & \mathbf{id} \\
\mathit{expr} & \rightarrow & \mathbf{id}\ (\ \mathit{expr\_list}\ ) \\
\mathit{expr} & \rightarrow & \mathbf{id} \\
\mathit{expr\_list} & \rightarrow & \mathit{expr\_list}\ ,\ \mathit{expr} \\
\mathit{expr\_list} & \rightarrow & \mathit{expr}
\end{array}
$$

Stack	Input	Action
$…	…$	…
$…id (id	**, id)…$**	reduce which? *parameter* → **id** or *expr* → **id**
$…procid (id	**, id)…$**	

LR Parsing

mansoori@shirazu.ac.ir

- LR(k) parsing
  - k: no. of lookahead tokens, used in making parsing decisions
  - $k = 0$, $k = 1$: used in practice

- Why LR parser?
  - Can be constructed for most of programming constructs
  - Is the most general non-backtracking shift-reduce parser
  - Can detect a syntactic error as soon as is possible
  - Class of LR grammars is a proper superset of LL grammars

  - Too much work to construct an LR parser by hand

LR(0) Items of a Grammar

mansoori@shirazu.ac.ir

- An *LR*(0) item of a grammar G is a production of G with a • at some position of the right-hand side
- Thus, a production $A \rightarrow X\,Y\,Z$ has four items:

  $[A \rightarrow \bullet\, X\,Y\,Z]$
  $[A \rightarrow X \bullet\, Y\,Z]$
  $[A \rightarrow X\,Y \bullet\, Z]$
  $[A \rightarrow X\,Y\,Z \bullet]$

- Note that production $A \rightarrow \varepsilon$ has one item $[A \rightarrow \bullet]$
- An item indicates how much of a production had been seen at a given point in the parsing process
- Set of LR(0) items: $\{\ [T \rightarrow T * \bullet\, F], [F \rightarrow \bullet\, (\ E\ )], [F \rightarrow \bullet\, \textbf{id}]\ \}$

CLOSURE Operation for LR(0) Items

mansoori@shirazu.ac.ir

- I is a set of LR(0) items for a grammar G
- CLOSURE(I) constructs the set of LR(0) items J from I by these rules:

1. Add every item in I to J
2. If $[A \to \alpha \bullet B\beta] \in J$ then for each production $B \to \gamma$ in G, add the item $[B \to \bullet\gamma]$ to J if not already in it
3. Repeat 2 until no new items can be added to J

Example CLOSURE Operation

mansoori@shirazu.ac.ir

Grammar:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid \textbf{id}$

$\text{CLOSURE}(\{[E' \rightarrow \bullet E]\}) =$

$\{ [E' \rightarrow \bullet E] \}$

$\{ [E' \rightarrow \bullet E]$
$[E \rightarrow \bullet E + T]$
$[E \rightarrow \bullet T] \}$

Add $[E \rightarrow \bullet \gamma]$

$\{ [E' \rightarrow \bullet E]$
$[E \rightarrow \bullet E + T]$
$[E \rightarrow \bullet T]$
$[T \rightarrow \bullet T * F]$
$[T \rightarrow \bullet F] \}$

Add $[T \rightarrow \bullet \gamma]$

$\{ [E' \rightarrow \bullet E]$
$[E \rightarrow \bullet E + T]$
$[E \rightarrow \bullet T]$
$[T \rightarrow \bullet T * F]$
$[T \rightarrow \bullet F]$
$[F \rightarrow \bullet (E)]$
$[F \rightarrow \bullet \textbf{id}] \}$

Add $[F \rightarrow \bullet \gamma]$

mansoori@shirazu.ac.ir

Kernel and Nonkernel Items

- Kernel items: initial item, $S' \rightarrow \bullet S$, and all items whose dots are not at the left end
- Nonkernel items: all items with their dots at the left end, except for $S' \rightarrow \bullet S$
- Storage vs. speed

I_0
$E' \rightarrow \cdot E$
$E \rightarrow \cdot E + T$
$E \rightarrow \cdot T$
$T \rightarrow \cdot T * F$
$T \rightarrow \cdot F$
$F \rightarrow \cdot (E)$
$F \rightarrow \cdot \mathbf{id}$

I_1
$E' \rightarrow E \cdot$
$E \rightarrow E \cdot + T$

I_2
$E \rightarrow T \cdot$
$T \rightarrow T \cdot * F$

I_3
$T \rightarrow F \cdot$

I_4
$F \rightarrow (\cdot E)$
$E \rightarrow \cdot E + T$
$E \rightarrow \cdot T$
$T \rightarrow \cdot T * F$
$T \rightarrow \cdot F$
$F \rightarrow \cdot (E)$
$F \rightarrow \cdot \mathbf{id}$

I_5
$F \rightarrow \mathbf{id} \cdot$

I_6
$E \rightarrow E + \cdot T$
$T \rightarrow \cdot T * F$
$T \rightarrow \cdot F$
$F \rightarrow \cdot (E)$
$F \rightarrow \cdot \mathbf{id}$

I_7
$T \rightarrow T * \cdot F$
$F \rightarrow \cdot (E)$
$F \rightarrow \cdot \mathbf{id}$

I_8
$E \rightarrow E \cdot + T$
$F \rightarrow (E \cdot)$

I_9
$E \rightarrow E + T \cdot$
$T \rightarrow T \cdot * F$

I_{10}
$T \rightarrow T * F \cdot$

I_{11}
$F \rightarrow (E) \cdot$

GOTO Operation for LR(0) Items

mansoori@shirazu.ac.ir

- *I* is a set of LR(0) items and X is a symbol for grammar *G*
- GOTO(*I*,*X*) constructs a new set of LR(0) items *J*:

1. For each item $[A \rightarrow \alpha \bullet X\beta] \in I$, add the set of items CLOSURE($\{[A \rightarrow \alpha X \bullet \beta]\}$) to *J* if not already there
2. Repeat 1 until no more items can be added to *J*

Example GOTO Operation

mansoori@shirazu.ac.ir

Grammar:

$E \to E + T \mid T$

$T \to T * F \mid F$

$F \to (\, E\,) \mid \mathbf{id}$

$$I = \{\ [E' \to \bullet\ E], [E \to \bullet\ E + T], [E \to \bullet\ T], [T \to \bullet\ T * F], [T \to \bullet\ F], [F \to \bullet\ (\ E\ )], [F \to \bullet\ \mathbf{id}]\ \}$$

$$J = \mathrm{GOTO}(I, E) = \mathrm{CLOSURE}(\{[E' \to E\ \bullet], [E \to E\ \bullet + T]\}) = \{\ [E' \to E\ \bullet], [E \to E\ \bullet + T]\ \}$$

$$I = \{\ [E' \to E\ \bullet], [E \to E\ \bullet + T]\ \}$$

$$J = \mathrm{GOTO}(I, +) = \mathrm{CLOSURE}(\{[E \to E + \bullet\ T]\}) = \{[E \to E + \bullet\ T], [T \to \bullet\ T * F], [T \to \bullet\ F], [F \to \bullet\ (\ E\ )], [F \to \bullet\ \mathbf{id}]\ \}$$

Constructing Set of LR(0) Items

mansoori@shirazu.ac.ir

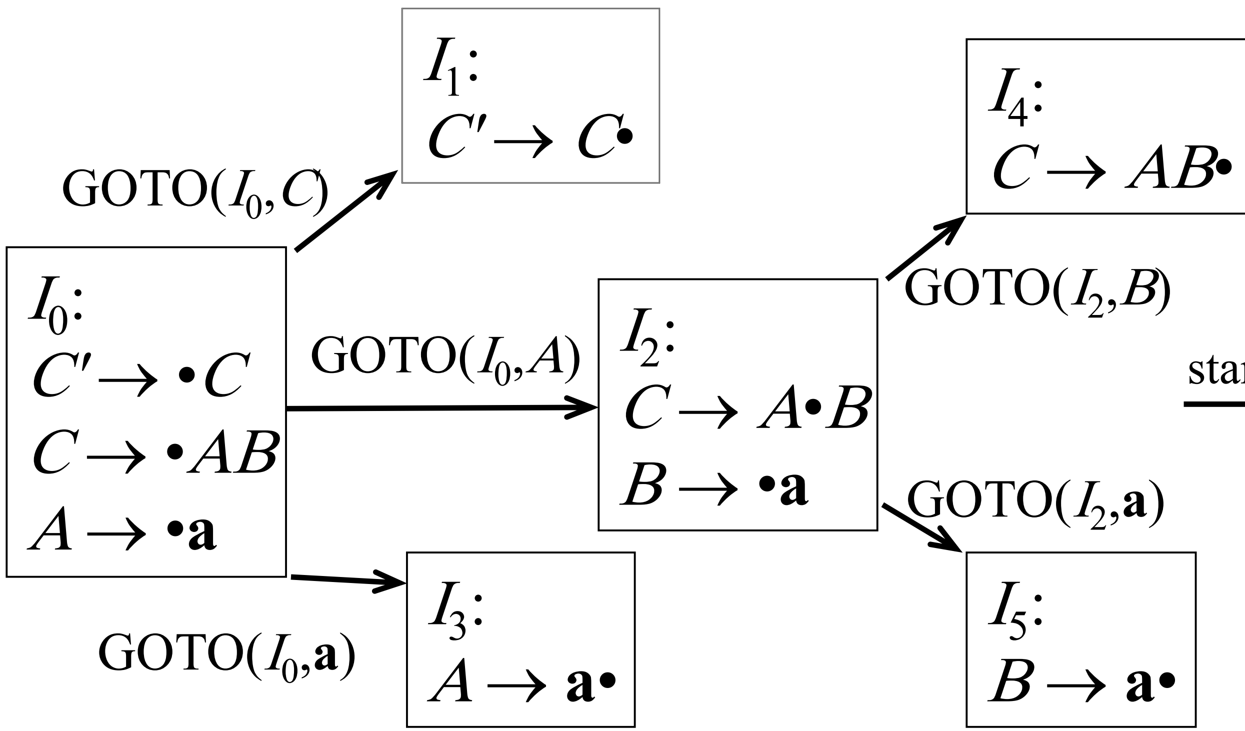
1. The grammar G is augmented to G' with a new start symbol S' and production $S' \rightarrow S$
2. Initially, set $C = \text{CLOSURE}(\{[S' \rightarrow \bullet S]\})$
   – This is the start state of a DFA ≡ LR(0) automaton
3. For each set of items $I \in C$ and each grammar symbol $X \in N \cup T$ such that $\text{GOTO}(I,X) \notin C$ and $\text{GOTO}(I,X) \neq \varnothing$, add the set of items $\text{GOTO}(I,X)$ to C
4. Repeat 3 until no more sets can be added to C

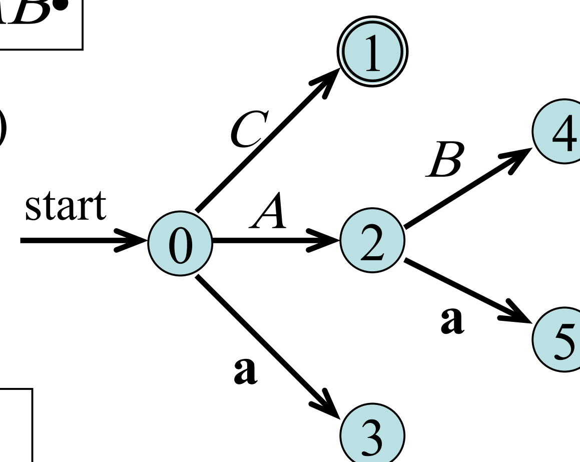
Constructing LR(0) Automaton (DFA)

mansoori@shirazu.ac.ir

- States: sets of LR(0) items (state $j \equiv$ set of items I_j)
  - Start state: CLOSURE($\{[S' \to \bullet\, S]\}$)
  - Final state: state contains item $[S' \to S\,\bullet]$
- Transitions: GOTO function

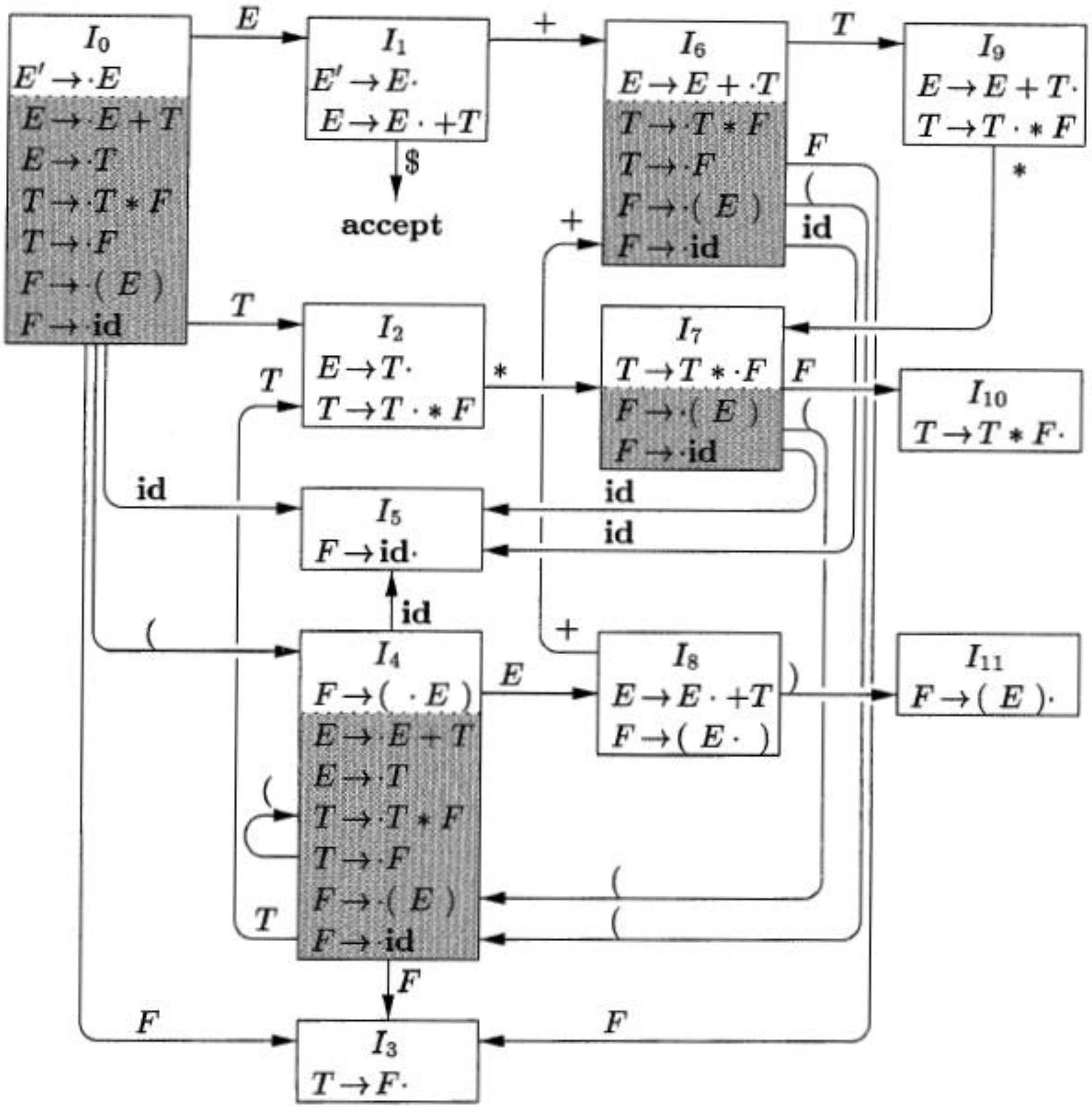
Grammar:
$C' \to C$
$C \to AB$
$A \to \mathbf{a}$
$B \to \mathbf{a}$





Example LR(0) DFA

mansoori @shirazu.ac.ir



Grammar:

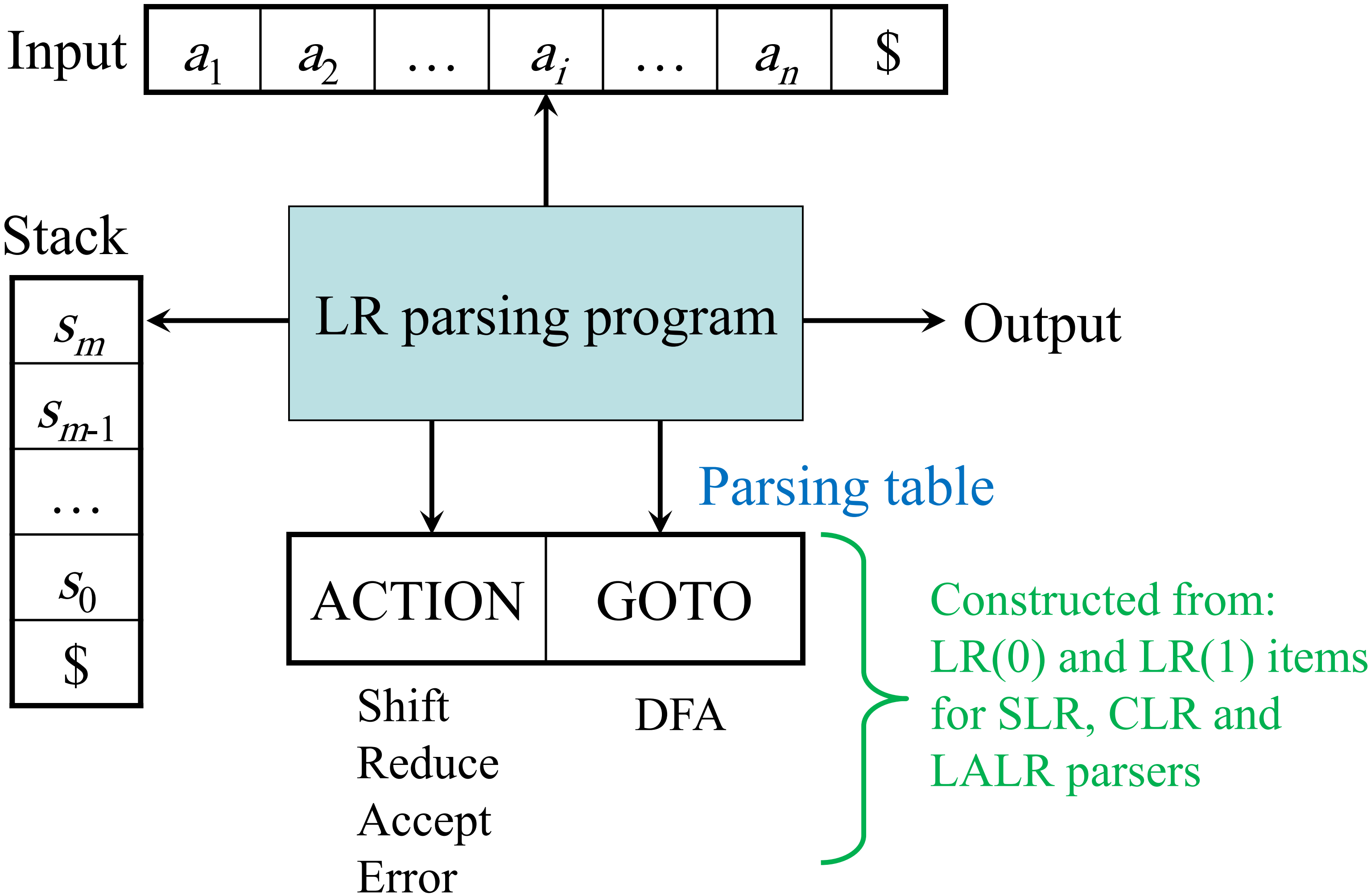
$E \to E + T \mid T$

$T \to T * F \mid F$

$F \to (E) \mid \mathbf{id}$

Model of LR Parsers

mansoori@shirazu.ac.ir



Structure of LR Parsing Table

mansoori@shirazu.ac.ir

- ACTION[i,a] for state I_i and terminal a (or \$):

  (1) Shift j: parser shifts input a to stack (indeed, state j to stack)

  (2) Reduce k (indeed, reduce $A \rightarrow \beta$): parser reduces β on top of stack to head A

  (3) Accept: parser accepts the input and finishes parsing

  (4) Error: parser discovers an error in its input and takes some corrective action

- GOTO[i,A] = j for state I_i and nonterminal A:
  - Parser maps I_i and A to I_j

Behavior of LR Parsers

mansoori@shirazu.ac.ir

LR parser configuration: $(\underbrace{s_0\ s_1\ \ldots\ s_m}_{\text{stack}},\ \underbrace{a_i\ a_{i+1}\ \ldots\ a_n\ \$}_{\text{input}})$

- **If** ACTION[s_m,a_i] = shift s **then** push s
  - Configuration: $(s_0\ s_1\ \ldots\ s_m\ s,\ a_{i+1}\ \ldots\ a_n\ \$)$
- **If** ACTION[s_m,a_i] = reduce $A \rightarrow \beta$ and GOTO[s_{m-r},A] = s with $r = |\beta|$ **then** pop r symbols and push s
  - Configuration: $(s_0\ s_1\ \ldots\ s_{m-r}\ s,\ a_i\ a_{i+1}\ \ldots\ a_n\ \$)$
- **If** ACTION[s_m,a_i] = accept **then** stop parsing
  - Configuration: $(s_0\ s_1,\ \$)$ where s_1 is final state
- **If** ACTION[s_m,a_i] = error **then** call error recovery routine

Example LR Parsing Table

mansoori@shirazu.ac.ir

Grammar:

$E' \to E$

1) $E \to E + T$
2) $E \to T$
3) $T \to T * F$
4) $T \to F$
5) $F \to (E)$
6) $F \to \mathbf{id}$

	ACTION						GOTO		
STATE	**id**	+	*	(	)	$	*E*	*T*	*F*
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Shift state 5

Reduce by $E \to E + T$

Example LR Parsing

mansoori@shirazu.ac.ir

Grammar:

$E' \rightarrow E$

1) $E \rightarrow E + T$

2) $E \rightarrow T$

3) $T \rightarrow T * F$

4) $T \rightarrow F$

5) $F \rightarrow (E)$

6) $F \rightarrow \mathbf{id}$

STACK	SYMBOL	INPUT	ACTION
0	$	**id*id+id$**	shift 5
0 5	$ **id**	***id+id$**	reduce by $F \rightarrow \mathbf{id}$
0 3	$ F	***id+id$**	reduce by $T \rightarrow F$
0 2	$ T	***id+id$**	shift 7
0 2 7	$ $T*$	**id+id$**	shift 5
0 2 7 5	$ $T*$ **id**	**+id$**	reduce by $F \rightarrow \mathbf{id}$
0 2 7 10	$ $T*F$	**+id$**	reduce by $T \rightarrow T * F$
0 2	$ T	**+id$**	reduce by $E \rightarrow T$
0 1	$ E	**+id$**	shift 6
0 1 6	$ $E+$	**id$**	shift 5
0 1 6 5	$ $E+$ **id**	$	reduce by $F \rightarrow \mathbf{id}$
0 1 6 3	$ $E+F$	$	reduce by $T \rightarrow F$
0 1 6 9	$ $E+T$	$	reduce by $E \rightarrow E + T$
0 1	$ E	$	accept

SLR Parsing

mansoori@shirazu.ac.ir

- In LR(0) DFA:
  - An LR(0) state is a set of LR(0) items
  - An LR(0) item is a production with a • in its right-hand side
- Build LR(0) DFA by:
  - CLOSURE operation to construct LR(0) items
  - GOTO operation to determine transitions
- Construct SLR parsing table from LR(0) DFA
- LR parser program which uses SLR parsing table to determine shift/reduce operations is called SLR parser

Constructing SLR Parsing Table

mansoori@shirazu.ac.ir

1. Augment grammar G with $S' \rightarrow S$ to get G'
2. Construct set $C = \{I_0, I_1, \ldots, I_n\}$ of LR(0) items for G'
3. If $[A \rightarrow \alpha \bullet a\beta] \in I_i$ and GOTO$(I_i, a) = I_j$ then set ACTION$[i,a]$ = shift j
4. If $[A \rightarrow \alpha \bullet] \in I_i$ then set ACTION$[i,a]$ = reduce $A \rightarrow \alpha$ for all $a \in$ FOLLOW(A) (apply only if $A \neq S'$)
5. If $[S' \rightarrow S \bullet] \in I_i$ then set ACTION$[i,\$]$ = accept
6. If GOTO$(I_i, A) = I_j$ then set GOTO$[i,A] = j$
7. Repeat 3-6 until no more entries added
8. The initial state i is the I_i holding item $[S' \rightarrow \bullet S]$

Any conflict in ACTION $\Rightarrow$ grammar G is not SLR

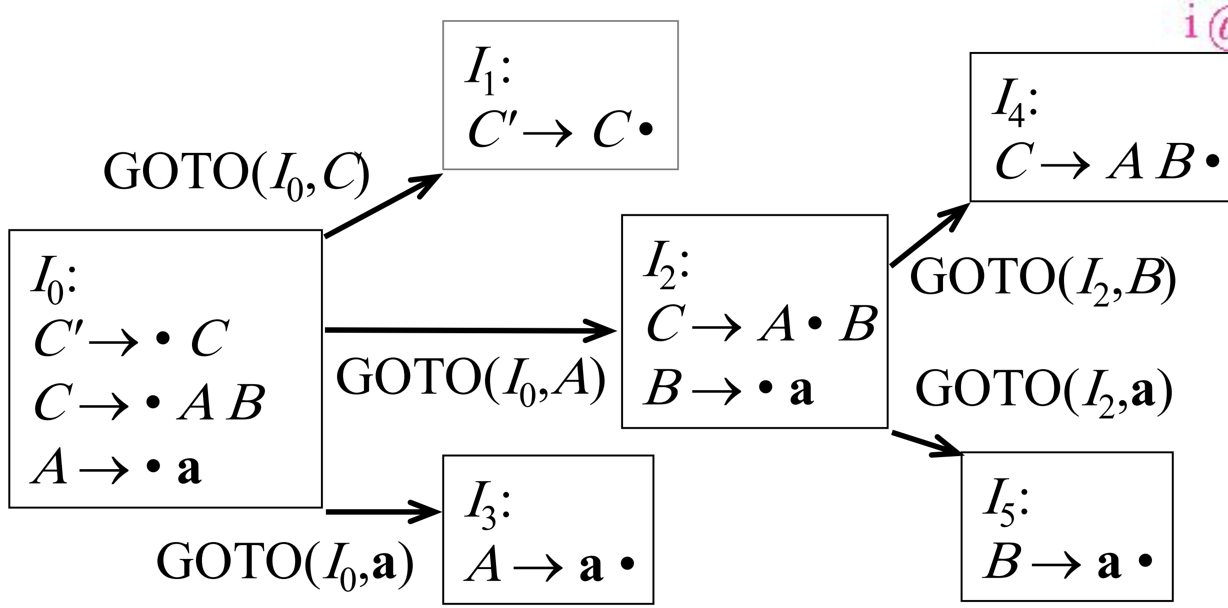
mansoori@shirazu.ac.ir

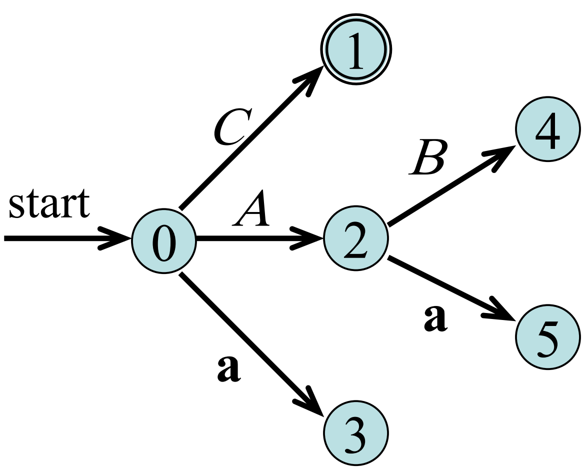
Example1 SLR Parsing Table

Grammar:

$C' \to C$

1) $C \to A\ B$
2) $A \to \mathbf{a}$
3) $B \to \mathbf{a}$





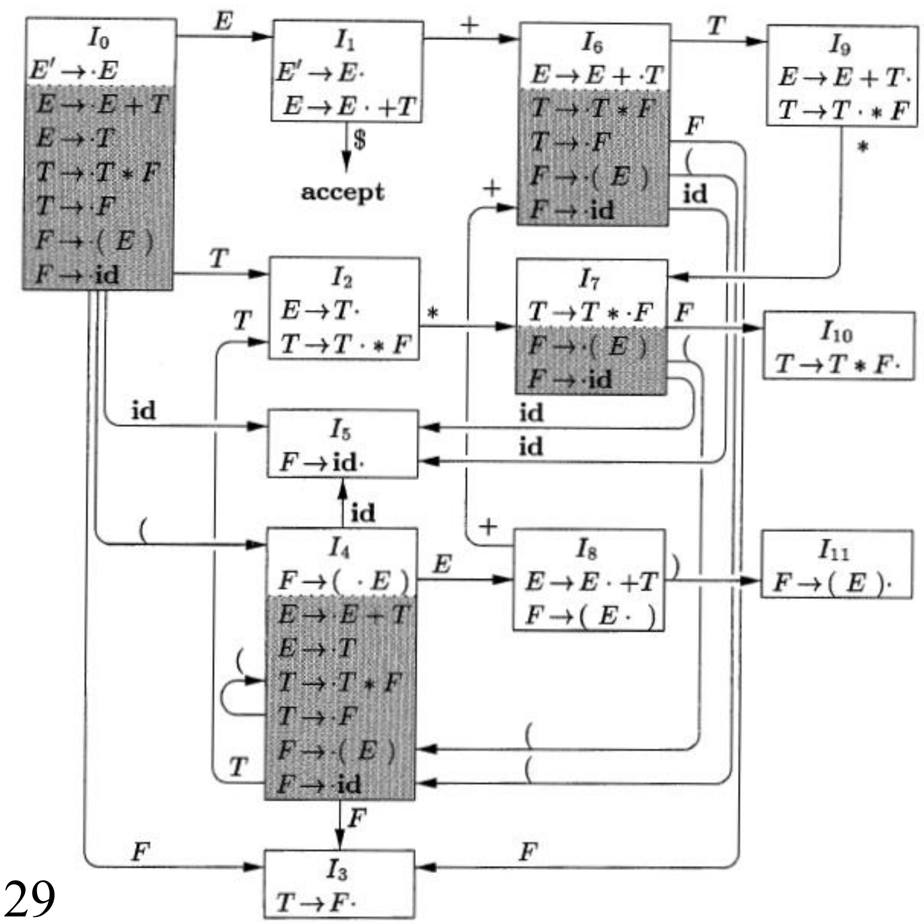
STATE	ACTION a	ACTION $	GOTO C	GOTO A	GOTO B
0	s3		1	2	
1		acc			
2	s5				4
3	r2				
4		r1			
5		r3			

Example2 SLR Parsing Table

mansoori@shirazu.ac.ir

$E' \rightarrow E$

1) $E \rightarrow E + T$
2) $E \rightarrow T$
3) $T \rightarrow T * F$
4) $T \rightarrow F$
5) $F \rightarrow (E)$
6) $F \rightarrow \textbf{id}$



STATE	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

SLR and Ambiguity

mansoori@shirazu.ac.ir

- Every SLR grammar is unambiguous, but **not** every unambiguous grammar is SLR

- An unambiguous grammar:

  FOLLOW(S) = {\$}

  FOLLOW(R) = FOLLOW(L) = {=,\$}

$$S' \to S$$
$$S \to L = R \mid R$$
$$L \to * R \mid \mathbf{id}$$
$$R \to L$$

I_0:
$S' \to \bullet S$
$S \to \bullet L{=}R$
$S \to \bullet R$
$L \to \bullet * R$
$L \to \bullet \mathbf{id}$
$R \to \bullet L$

I_1:
$S' \to S\bullet$

I_2:
$S \to L\bullet{=}R$
$R \to L\bullet$

I_3:
$S \to R\bullet$

I_4:
$L \to *\bullet R$
$R \to \bullet L$
$L \to \bullet * R$
$L \to \bullet \mathbf{id}$

I_5:
$L \to \mathbf{id}\bullet$

I_6:
$S \to L{=}\bullet R$
$R \to \bullet L$
$L \to \bullet * R$
$L \to \bullet \mathbf{id}$

I_7:
$L \to * R\bullet$

I_8:
$R \to L\bullet$

I_9:
$S \to L{=}R\bullet$

$$\left.\begin{array}{l}\text{ACTION}[2,=] = \text{s6} \\ \text{ACTION}[2,=] = \text{r5}\end{array}\right\}\ \text{Conflict in SLR parsing table}$$

Parsers more Powerful than SLR

mansoori@shirazu.ac.ir

- SLR (simple LR) parser
  - Is too simple with limited power

- More powerful LR parsers
  1. CLR (canonical LR)
     - Makes full use of lookahead symbol
     - Uses a large set of LR(1) items
  2. LALR (lookahead LR)
     - Is based on LR(0) items
     - Has many fewer states than CLR parser
     - Its parsing tables are no bigger than SLR tables

CLR vs. SLR

mansoori@shirazu.ac.ir

- SLR parser uses LR(0) automaton
- CLR parser uses LR(1) automaton
  - Uses lookahead to avoid conflicts in parsing table
  - LR(1) item = LR(0) item + lookahead
  - LR(0) item: $[A \rightarrow \alpha \bullet \beta]$ LR(1) item: $[A \rightarrow \alpha \bullet \beta, a]$
  - Splits LR(0) states by adding lookahead to obtain LR(1) states

I_2:
$S \rightarrow L \bullet = R$
$R \rightarrow L \bullet$

split

lookahead=$

$S \rightarrow L \bullet = R$ ⇒ ACTION[2,=] = s6

$R \rightarrow L \bullet$ ⇒ ACTION[2,$] = r5

Grammar:
$S \rightarrow L = R \mid R$
$L \rightarrow * R \mid \textbf{id}$
$R \rightarrow L$

Should not reduce on =, because no right-sentential form begins with R=

LR(1) Items

mansoori@shirazu.ac.ir

- An *LR*(1) *item* $[A \to \alpha \bullet \beta, a]$ contains a *lookahead* terminal a or endmarker \$, meaning α already on top of stack, expect to see βa
  - LR(1) items: $[R \to L\bullet, \$]$, $[S \to L\bullet{=}R, =]$, $[S \to L\bullet{=}R, =/\$]$
  - 1st part: core, 2nd part: lookahead

- For items of the form $[A \to \alpha \bullet \beta, a]$ with $\beta \neq \varepsilon$, lookahead a has no effect

- For items of the form $[A \to \alpha \bullet, a]$, lookahead a is used to reduce $A \to \alpha$ only if the next token is a

CLOSURE Operation for LR(1) Items

mansoori@shirazu.ac.ir

- I is a set of LR(1) items for a grammar G
- CLOSURE(I) constructs the set of LR(1) items J from I by these rules:

1. Add every item in I to J
2. If $[A \to \alpha \bullet B\beta, a] \in J$ then for each production $B \to \gamma$ in G and for each terminal $b \in \mathrm{FIRST}(\beta a)$, add the item $[B \to \bullet\gamma, b]$ to J if not already in it
3. Repeat 2 until no new items can be added to J

GOTO Operation for LR(1) Items

mansoori@shirazu.ac.ir

- I is a set of LR(1) items and X is a symbol for grammar G
- GOTO(I,X) constructs a new set of LR(1) items J:

1. For each item $[A \to \alpha \bullet X\beta, a] \in I$, add the set of items $CLOSURE(\{[A \to \alpha X \bullet \beta, a]\})$ to J if not already there
2. Repeat 1 until no more items can be added to J

Example CLOSURE and GOTO

mansoori@shirazu.ac.ir

Grammar:
$S \rightarrow B\ B$
$B \rightarrow \mathbf{a}\ B \mid \mathbf{b}$

CLOSURE($\{[S \rightarrow \bullet\ B\ B, \$]\}$) =
$\{[S \rightarrow \bullet\ B\ B, \$], [B \rightarrow \bullet\ \mathbf{a}\ B, a], [B \rightarrow \bullet\ \mathbf{a}\ B, b], [B \rightarrow \bullet\ \mathbf{b}, a], [B \rightarrow \bullet\ \mathbf{b}, b]\}$
=
$\{[S \rightarrow \bullet\ B\ B, \$], [B \rightarrow \bullet\ \mathbf{a}\ B, a/b], [B \rightarrow \bullet\ \mathbf{b}, a/b]\}$
CLOSURE($\{[B \rightarrow \mathbf{a}\bullet B, a/b]\}$) =
$\{[B \rightarrow \mathbf{a}\ \bullet\ B, a/b], [B \rightarrow \bullet\ \mathbf{a}\ B, a/b], [B \rightarrow \bullet\ \mathbf{b}, a/b]\}$

$I = \{[S \rightarrow \bullet\ B\ B, \$], [B \rightarrow \bullet\ \mathbf{a}\ B, a/b], [B \rightarrow \bullet\ \mathbf{b}, a/b]\}$
GOTO(I,B) = $\{[S \rightarrow B\ \bullet\ B, \$], [B \rightarrow \bullet\ \mathbf{a}\ B, \$], [B \rightarrow \bullet\ \mathbf{b}, \$]\}$

Constructing Set of LR(1) Items

mansoori@shirazu.ac.ir

1. The grammar G is augmented to G' with a new start symbol S' and production $S' \rightarrow S$

2. Initially, set $C = \text{CLOSURE}(\{[S' \rightarrow \bullet\, S, \$]\})$
   – This is the start state of a DFA ≡ LR(1) automaton

3. For each set of items $I \in C$ and each grammar symbol $X \in N \cup T$ such that $\text{GOTO}(I,X) \notin C$ and $\text{GOTO}(I,X) \neq \emptyset$, add the set of items $\text{GOTO}(I,X)$ to C

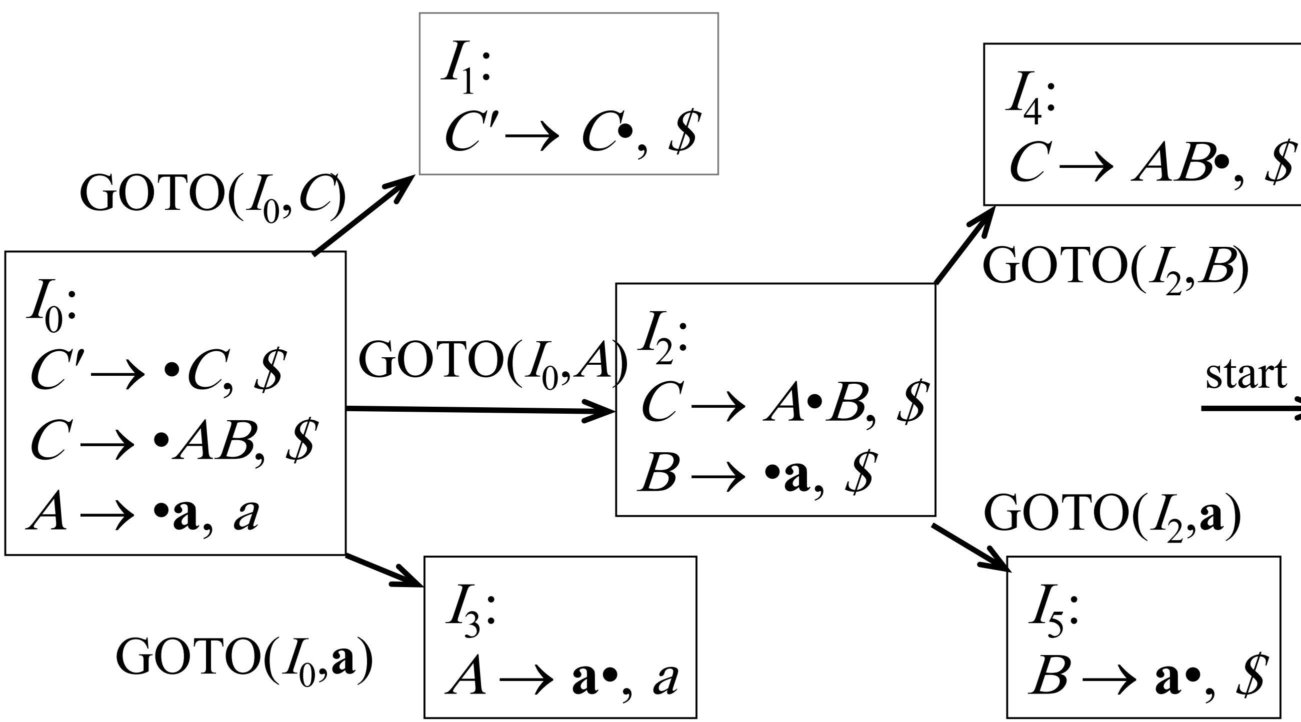
4. Repeat 3 until no more sets can be added to C

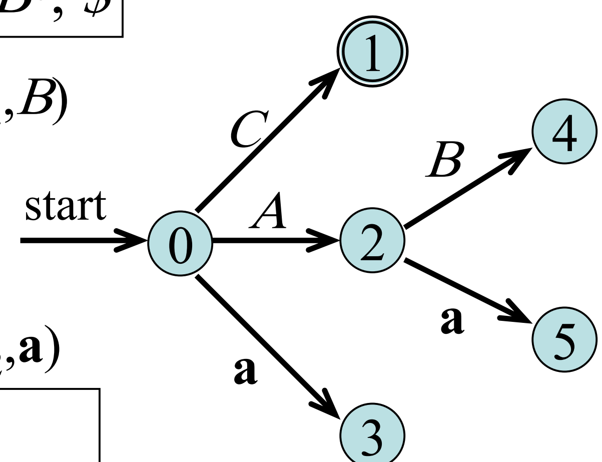
Constructing LR(1) DFA

mansoori@shirazu.ac.ir

- States: sets of LR(1) items (state j ≡ set of items I_j)
  - Start state: CLOSURE({[S' → •S, $]})
  - Final state: state contains item [S' → S•, $]
- Transitions: GOTO function

Grammar:
$C' \rightarrow C$
$C \rightarrow AB$
$A \rightarrow \mathbf{a}$
$B \rightarrow \mathbf{a}$





Example1 LR(1) DFA

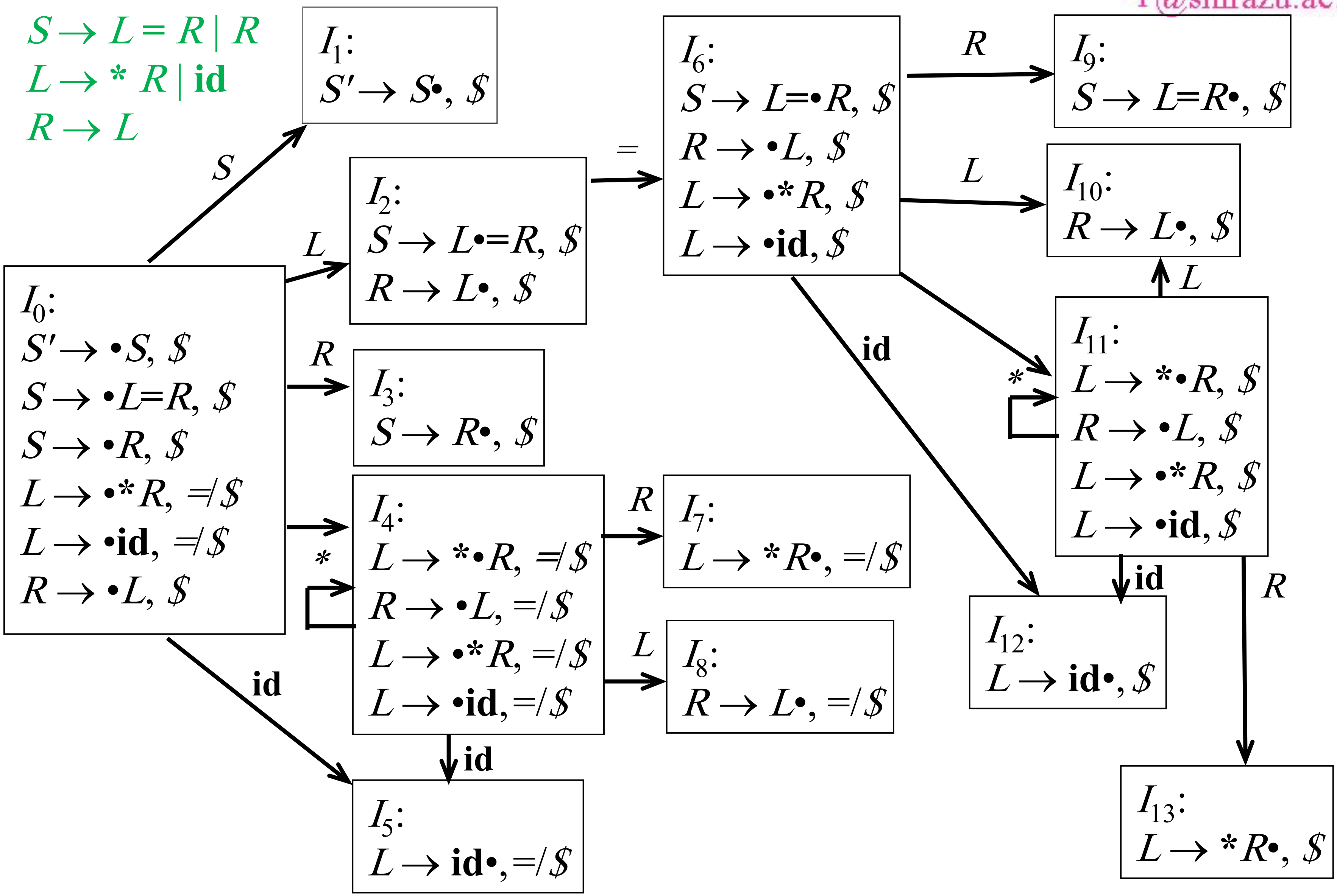
mansoori@shirazu.ac.ir

Grammar:

$S \to L = R \mid R$

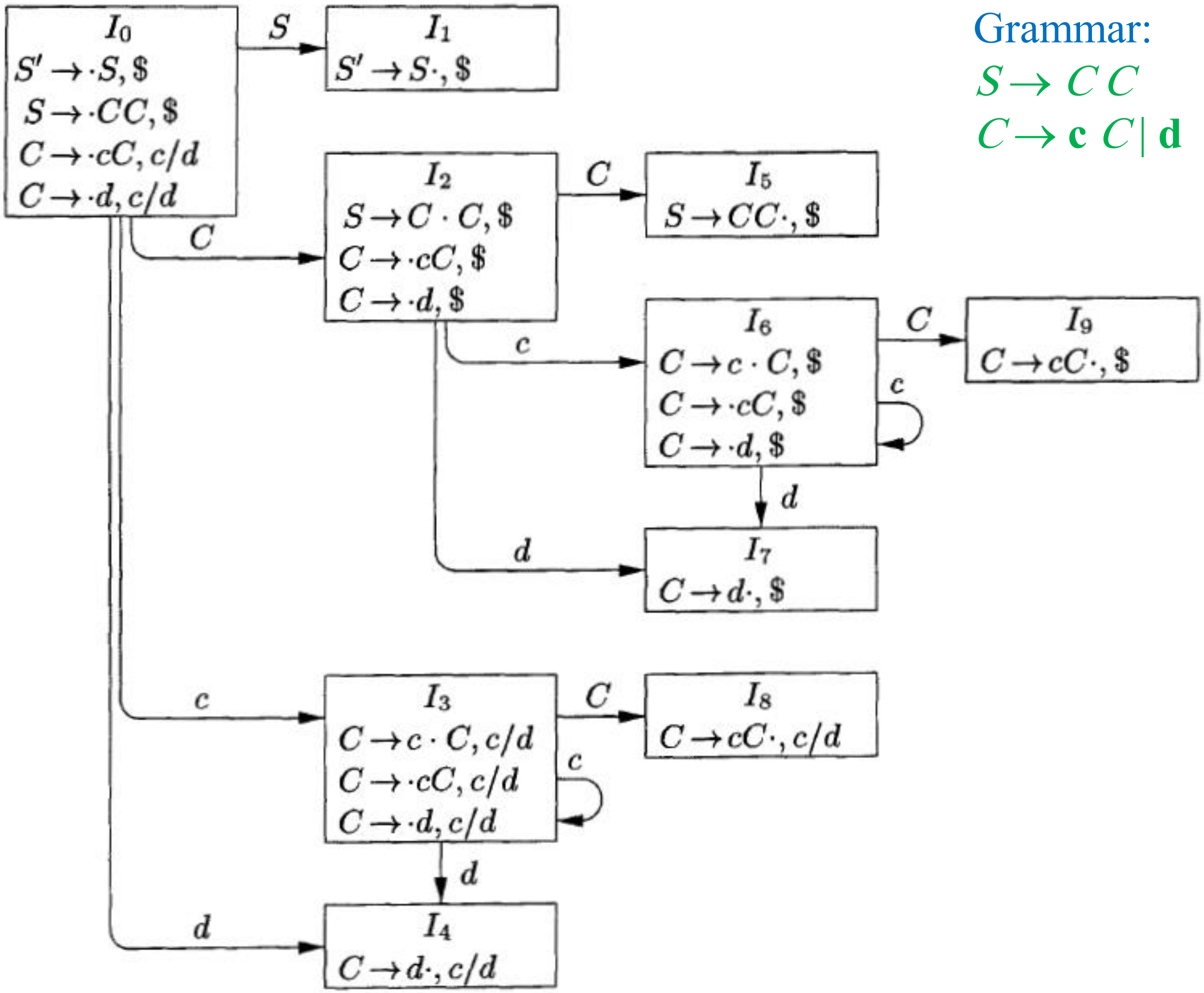
$L \to * R \mid \textbf{id}$

$R \to L$



Example2 LR(1) DFA

mansoori@shirazu.ac.ir



Grammar:

$S \to C\,C$

$C \to \mathbf{c}\ C \mid \mathbf{d}$

CLR Parsing

mansoori@shirazu.ac.ir

- In LR(1) DFA, each state is a set of LR(1) items
- Construct CLR parsing table from LR(1) DFA
- LR parser program which uses CLR parsing table to determine shift/reduce operations is called CLR parser

Constructing CLR Parsing Table

mansoori@shirazu.ac.ir

1. Augment grammar G with $S' \to S$ to get G'
2. Construct set $C' = \{I_0, I_1, \ldots, I_n\}$ of LR(1) items for G'
3. If $[A \to \alpha \bullet a\beta, b] \in I_i$ and $GOTO(I_i,a) = I_j$ then set $ACTION[i,a]$ = shift j
4. If $[A \to \alpha \bullet, a] \in I_i$ then set $ACTION[i,a]$ = reduce $A \to \alpha$ (apply only if $A \neq S'$)
5. If $[S' \to S \bullet, \$] \in I_i$ then set $ACTION[i,\$]$ = accept
6. If $GOTO(I_i,A) = I_j$ then set $GOTO[i,A] = j$
7. Repeat 3-6 until no more entries added
8. The initial state i is the I_i holding item $[S' \to \bullet S, \$]$

Any conflict in ACTION $\Rightarrow$ grammar G is not CLR

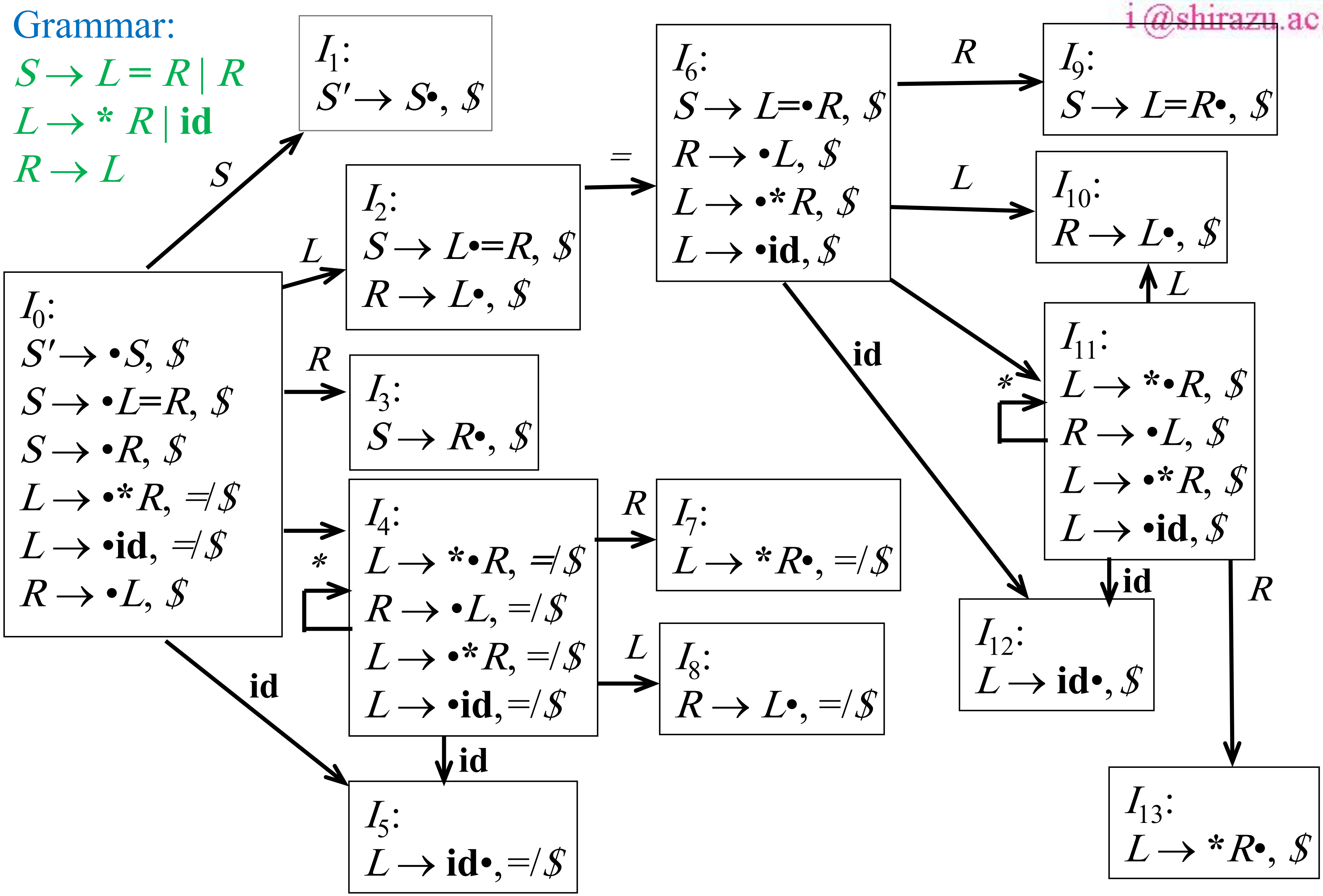
Example1 CLR Parsing Table

Grammar:

$S \rightarrow L = R \mid R$

$L \rightarrow * R \mid \mathbf{id}$

$R \rightarrow L$



Example1 CLR Parsing Table

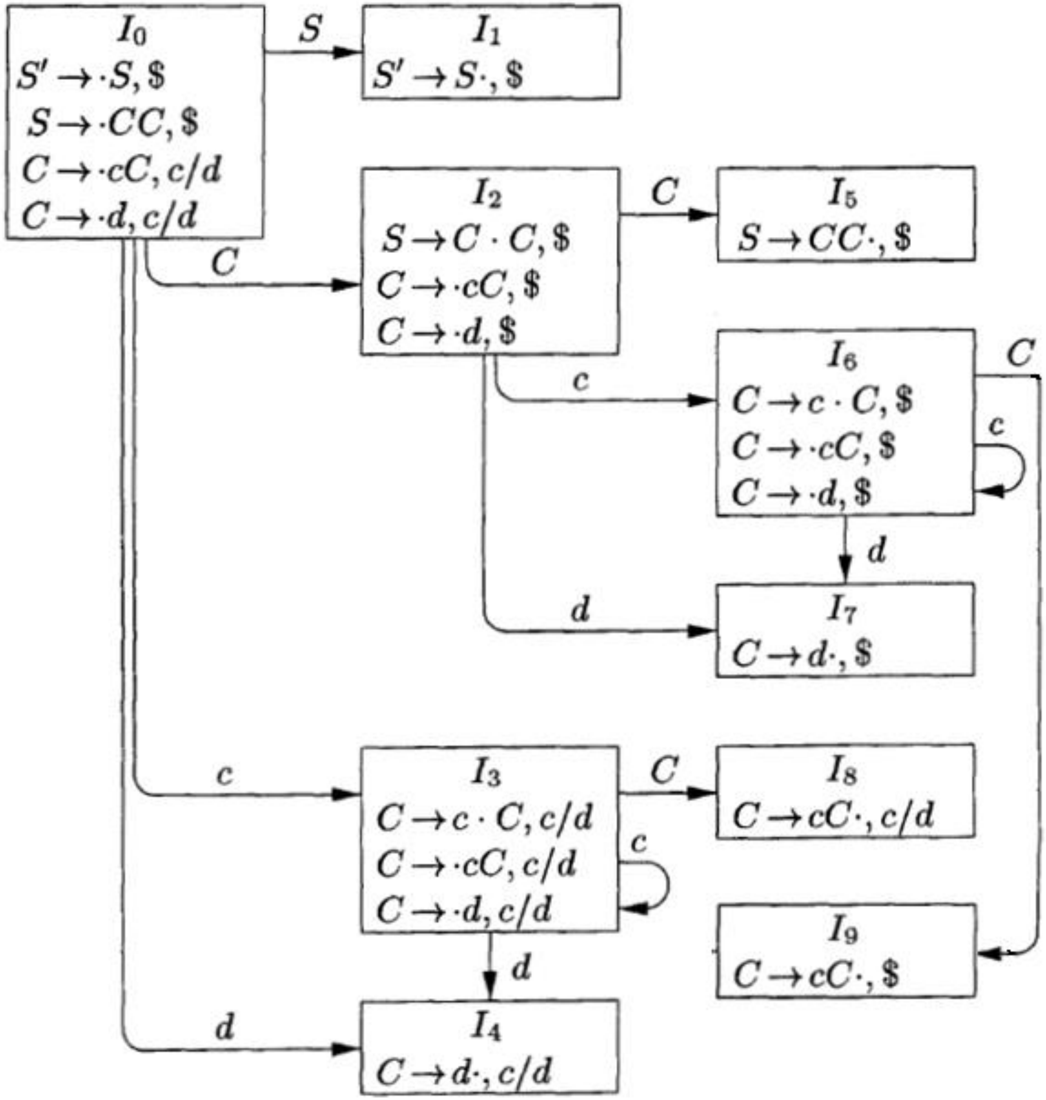
$S' \to S$
1) $S \to L = R$
2) $S \to R$
3) $L \to * R$
4) $L \to$ **id**
5) $R \to L$

	ACTION				GOTO		
STATE	**id**	*	=	$	S	L	R
0	s5	s4			1	2	3
1				acc			
2			s6	r5			
3				r2			
4	s5	s4				8	7
5			r4	r4			
6	s12	s11				10	9
7			r3	r3			
8			r5	r5			
9				r1			
10				r5			
11	s12	s11				10	13
12				r4			
13				r3			

mansoori@shirazu.ac.ir

Example2 CLR Parsing Table

mansoori@shirazu.ac.ir



$S' \to S$

1) $S \to C\ C$

2) $C \to$ **c** C

3) $C \to$ **d**

STATE	ACTION			GOTO	
	c	d	$\$$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

Power of CLR vs. SLR

mansoori@shirazu.ac.ir

- Every SLR grammar is a CLR grammar, but not every CLR is SLR

- For an SLR grammar, the CLR parser may have more states than the SLR parser for the same grammar

LALR Parser

mansoori@shirazu.ac.ir

- CLR parsing tables have many states (several thousand states for language C)
- LALR parsing combines CLR states to reduce table size to SLR (several hundred states for C)
- LALR is less powerful than CLR
  - Will not introduce shift-reduce conflicts
    - Because shifts do not use lookahead
  - May introduce reduce-reduce conflicts
    - But seldom do so for grammars of programming languages
  - Most common syntactic constructs of programming languages can be parsed by LALR

LALR Parsing Table

mansoori@shirazu.ac.ir

Combining LR(1) items with same core in CLR table ⇒ LALR table

G': $S' \to S$

1) $S \to C\,C$

2) $C \to \mathbf{c}\ C$

3) $C \to \mathbf{d}$

$L(G') = \underbrace{\mathbf{c^*\ d}}_{I_4}\ \underbrace{\mathbf{c^*\ d}}_{I_7}$

I_4:
$C \to \mathbf{d}\bullet,\ c/d$

I_7:
$C \to \mathbf{d}\bullet,\ \$$

$I_{47} = I_4 \cup I_7$

I_{47}:
$C \to \mathbf{d}\bullet,\ c/d/\$$

STACK	INPUT	ACTION
0	**cd**$	shift 3
0 3	**d**$	shift 4
0 3 4	$	error
0	**cd**$	shift 3
0 3	**d**$	shift 47
0 3 47	$	reduce by $C \to \mathbf{d}$
0 3 8	$	error

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

Conflict in LALR Parser

mansoori@shirazu.ac.ir

- In combining LR(1) items with same core
  - Unlikely for shift-reduce conflict
    - Suppose grammar G is CLR, and
    - There is a conflict after union: $[A\to\alpha\bullet,\ a]$ and $[B\to\beta\bullet a\gamma,\ b/c]$
    - This means: $\{[A\to\alpha\bullet,\ a],\ [B\to\beta\bullet a\gamma,\ b]\}\cup\{[A\to\alpha\bullet,\ a],\ [B\to\beta\bullet a\gamma,\ c]\}$
    - Obviously, each set of items has shift-reduce conflict, so G isnot CLR
  - Possible for reduce-reduce conflict

G': $S'\to S$

$S\to \mathbf{a}\ A\ \mathbf{d}\ |\ \mathbf{b}\ B\ \mathbf{d}\ |\ \mathbf{a}\ B\ \mathbf{e}\ |\ \mathbf{b}\ A\ \mathbf{e}$

$A\to \mathbf{c}$

$B\to \mathbf{c}$

Input: ac

$\{[A\to \mathbf{c}\bullet,\ d],\ [B\to \mathbf{c}\bullet,\ e]\}$

Input : bc

$\{[A\to \mathbf{c}\bullet,\ e],\ [B\to \mathbf{c}\bullet,\ d]\}$

No conflict

After union: reduce-reduce conflict

$\{[A\to \mathbf{c}\bullet,\ d/e],\ [B\to \mathbf{c}\bullet,\ d/e]\}$

Constructing LALR Parsing Table

mansoori@shirazu.ac.ir

1. Augment grammar G with $S' \rightarrow S$ to get G'
2. Construct set $C' = \{I_0, I_1, \ldots, I_n\}$ of LR(1) items for G'
3. For each core among LR(1) items in C', find all sets having that core, and replace these sets by their union to obtain $C'' = \{J_0, J_1, \ldots, J_m\}$
4. The parsing actions for state i are constructed from J_i as in CLR
5. If $J_i = I_1 \cup I_2 \cup \ldots \cup I_k$, then the cores of GOTO(I_1,A), GOTO(I_2,A), …, GOTO(I_k,A) are the same, so GOTO(J_i,A) = GOTO(I_1,A)

Any conflict in ACTION $\Rightarrow$ grammar G is not LALR

Example1 LALR Parsing Table

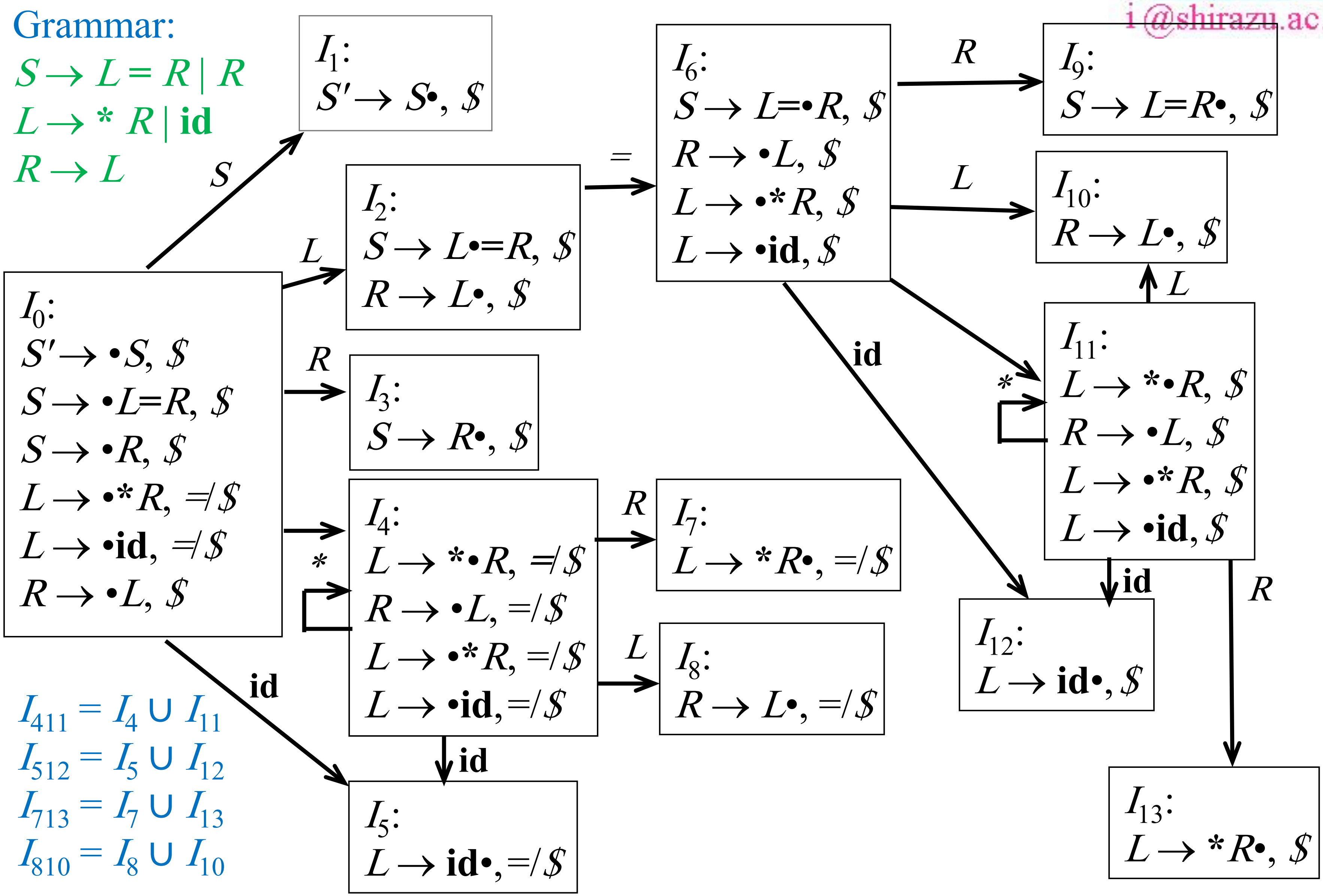
mansoori@shirazu.ac.ir

Grammar:

$S \rightarrow L = R \mid R$

$L \rightarrow * R \mid \textbf{id}$

$R \rightarrow L$



$I_{411} = I_4 \cup I_{11}$

$I_{512} = I_5 \cup I_{12}$

$I_{713} = I_7 \cup I_{13}$

$I_{810} = I_8 \cup I_{10}$

Example1 LALR Parsing Table

mansoori@shirazu.ac.ir

$S' \rightarrow S$

1) $S \rightarrow L = R$
2) $S \rightarrow R$
3) $L \rightarrow * R$
4) $L \rightarrow \textbf{id}$
5) $R \rightarrow L$

$I_{411} = I_4 \cup I_{11}$

$I_{512} = I_5 \cup I_{12}$

$I_{713} = I_7 \cup I_{13}$

$I_{810} = I_8 \cup I_{10}$

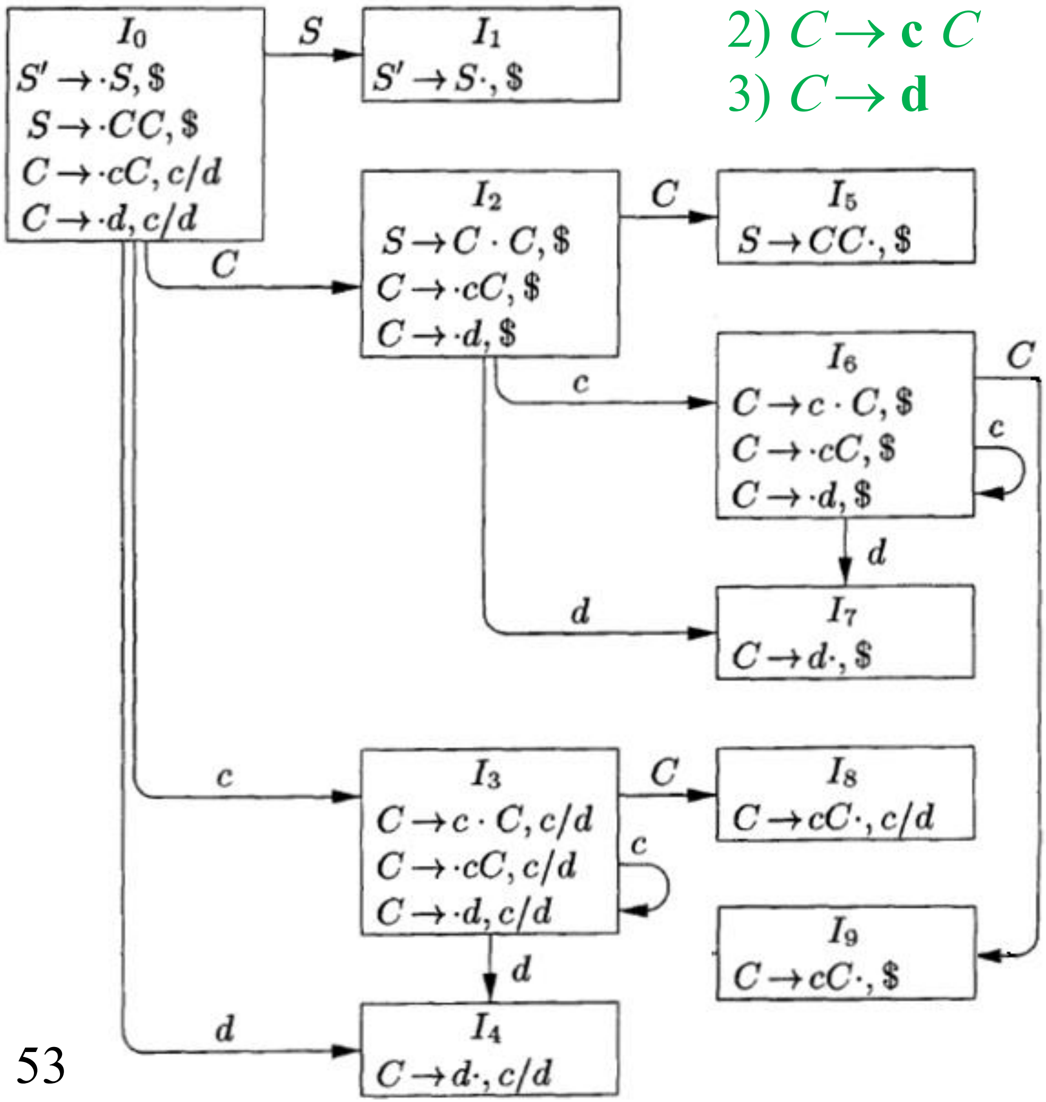
	ACTION				GOTO		
STATE	**id**	*	=	$	S	L	R
0	s5	s4			1	2	3
1				acc			
2			s6	r5			
3				r2			
4	s5	s4				8	7
5			r4	r4			
6	s12	s11				10	9
7			r3	r3			
8			r5	r5			
9				r1			
10				r5			
11	s12	s11				10	13
12				r4			
13				r3			

	ACTION				GOTO		
STATE	**id**	*	=	$	S	L	R
0	s512	s411			1	2	3
1				acc			
2			s6	r5			
3				r2			
411	s512	s411				810	713
512			r4	r4			
6	s512	s411				810	9
713			r3	r3			
810			r5	r5			
9				r1			

Example2 LALR Parsing Table

mansoori@shirazu.ac.ir

$S' \rightarrow S$
1) $S \rightarrow C\ C$
2) $C \rightarrow \mathbf{c}\ C$
3) $C \rightarrow \mathbf{d}$



STATE	ACTION			GOTO	
	c	d	\$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

STATE	ACTION			GOTO	
	c	d	\$	S	C
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

Example2 LALR Parsing

mansoori@shirazu.ac.ir

$S' \to S$
1) $S \to C\ C$
2) $C \to \mathbf{c}\ C$
3) $C \to \mathbf{d}$

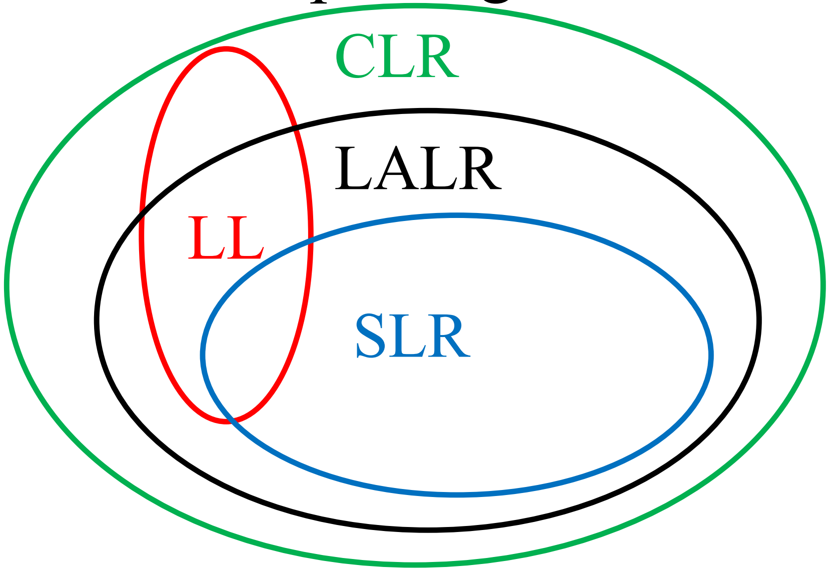
STATE	ACTION			GOTO	
	c	d	$	S	C
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

STACK	INPUT	ACTION
0	**cdcd**$	shift 36
0 36	**dcd**$	shift 47
0 36 47	**cd**$	reduce by $C \to \mathbf{d}$
0 36 89	**cd**$	reduce by $C \to \mathbf{c}\ C$
0 2	**cd**$	shift 36
0 2 36	**d**$	shift 47
0 2 36 47	$	reduce by $C \to \mathbf{d}$
0 2 36 89	$	reduce by $C \to \mathbf{c}\ C$
0 2 5	$	reduce by $S \to C\ C$
0 1	$	accept
0	**cd**$	shift 36
0 36	**d**$	shift 47
0 36 47	$	reduce by $C \to \mathbf{d}$
0 36 89	$	reduce by $C \to \mathbf{c}\ C$
0 2	$	error

LL, SLR, CLR and LALR Summary

mansoori@shirazu.ac.ir

- LL parsing tables are computed using FIRST & FOLLOW
  - Nonterminals × terminals → productions
- LR parsing tables are computed using CLOSURE & GOTO
  - LR states × terminals → shift/reduce actions (ACTION)
  - LR states × nonterminals → state transitions (GOTO)
- A grammar is LL/SLR/CLR/LALR if its parsing table has no conflicts



LR and Ambiguity

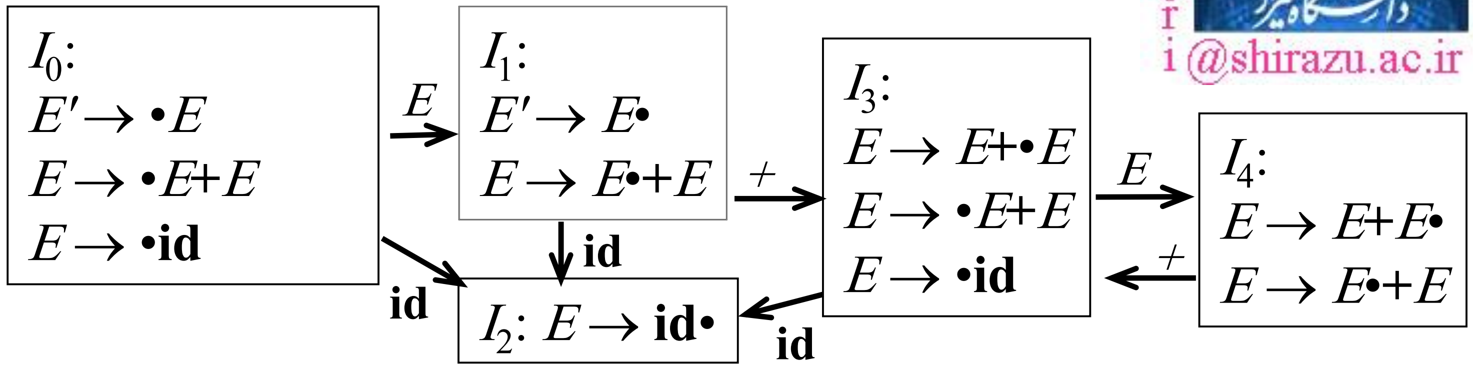
mansoori@shirazu.ac.ir

- Every ambiguous grammar fails to be LR (SLR, CLR, LALR)
- Some ambiguous grammars are useful in specification and implementation of languages
  - For expressions, an ambiguous grammar provides a shorter, more natural specification than unambiguous grammar
  - In ambiguous grammar of syntactic constructs, by adding new productions to the grammar, it can specify special-case constructs
- By specifying disambiguating rules
  - Overall language specification becomes unambiguous
  - So, it is possible to resolve conflicts in LR parsing tables

Associativity to Resolve Conflicts

mansoori@shirazu.ac.ir

$E' \to E$
1) $E \to E + E$
2) $E \to \textbf{id}$



	id	+	**$**	*E*
0	s2			1
1		s3	acc	
2		r2	r2	
3	s2			4
4		s3/r1	r1	

STACK	INPUT	ACTION
0	**id+id+id**$	shift 2
…	**…**	…
0 1 4	**+id**$	shift 3
…	**…**	…
0 1	$	accept: **id+(id+id)**
0	**id+id+id**$	shift 2
…	**…**	…
0 1 4	**+id**$	reduce by $E \to E + E$
…	**…**	…
0 1	$	accept: **(id+id)+id**

Left association: reduce
Right association: shift

Example1 Resolve Conflicts

mansoori@shirazu.ac.ir

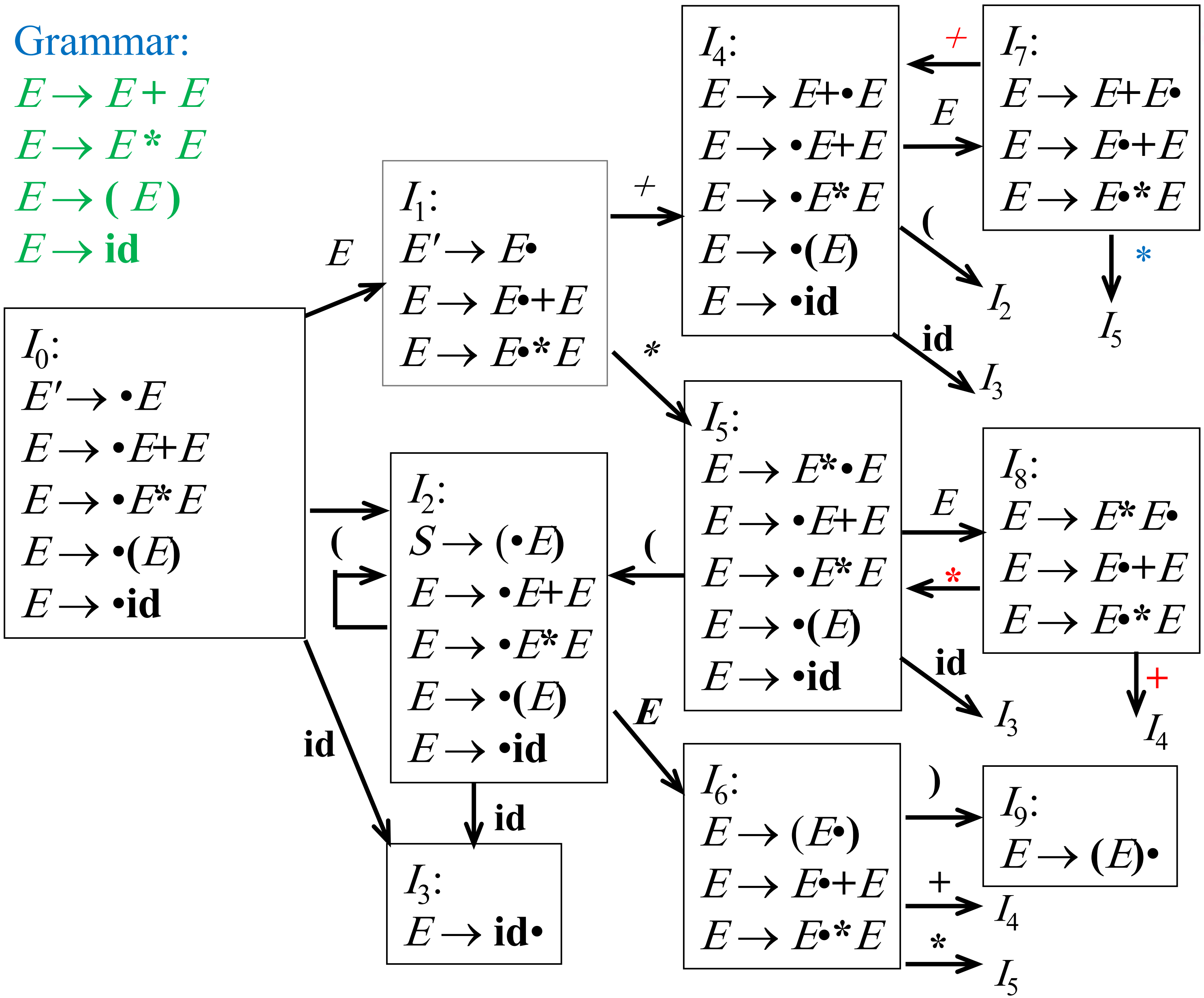
Grammar:

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow \mathbf{id}$



Example1 Resolve Conflicts

mansoori@shirazu.ac.ir

$E' \to E$

1) $E \to E + E$
2) $E \to E * E$
3) $E \to (E)$
4) $E \to \textbf{id}$

Left association for *, +

Precedence of * > +

STATE	ACTION						GOTO
	id	*	+	(	)	$	E
0	s3			s2			1
1		s5	s4			acc	
2	s3			s2			6
3		r4	r4		r4	r4	
4	s3			s2			7
5	s3			s2			8
6		s5	s4		s9		
7		s5/r1	s4/r1		r1	r1	
8		s5/r2	s4/r2		r2	r2	
9		r3	r3		r3	r3	

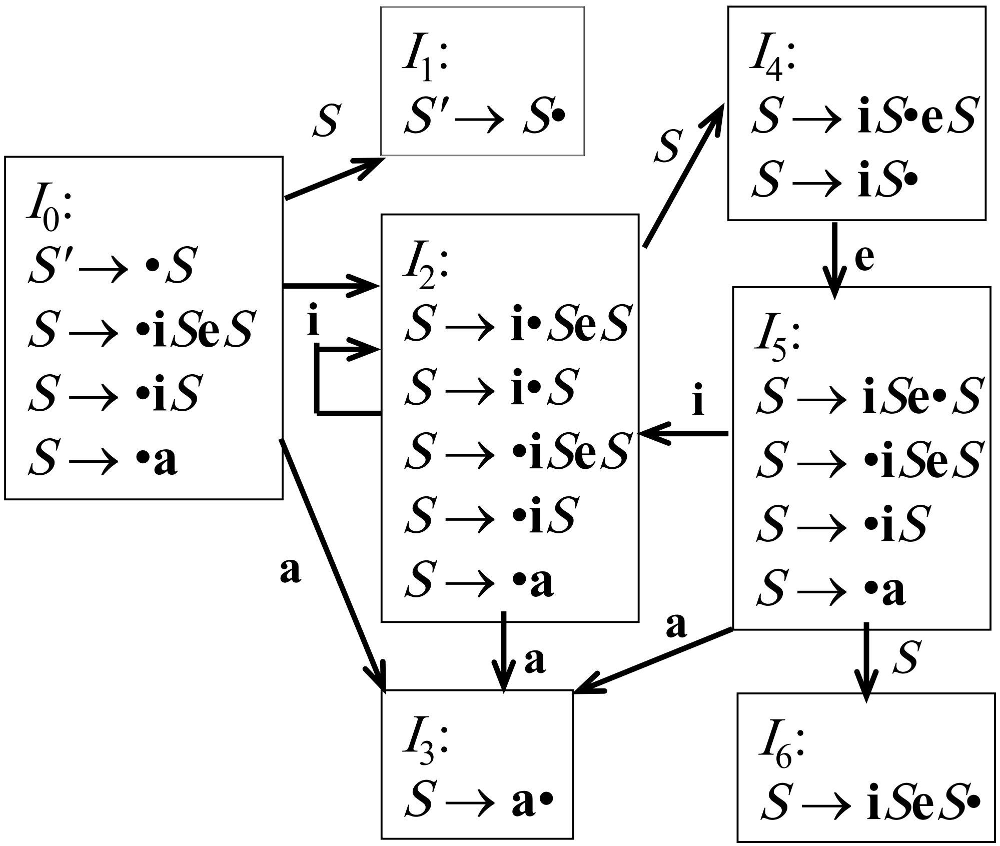
STATE	ACTION						GOTO
	id	*	+	(	)	$	E
0	s3			s2			1
1		s5	s4			acc	
2	s3			s2			6
3		r4	r4		r4	r4	
4	s3			s2			7
5	s3			s2			8
6		s5	s4		s9		
7		s5	r1		r1	r1	
8		r2	r2		r2	r2	
9		r3	r3		r3	r3	

Example2 Resolve Conflicts

mansoori@shirazu.ac.ir

stmt → **if** *expr* **then** *stmt* **else** *stmt*
| **if** *expr* **then** *stmt*
| **other**

$S' \rightarrow S$
1) $S \rightarrow \mathbf{i}\ S\ \mathbf{e}\ S$
2) $S \rightarrow \mathbf{i}\ S$
3) $S \rightarrow \mathbf{a}$



STATE	ACTION i	ACTION e	ACTION a	ACTION $	GOTO S
0	s2		s3		1
1				acc	
2	s2		s3		4
3		r3		r3	
4		s5/r2		r2	
5	s2		S3		6
6		r1		r1	