Syntax Analysis:

Context-free Grammars, Pushdown Automata and Parsing
Part - 6

Y.N. Srikant

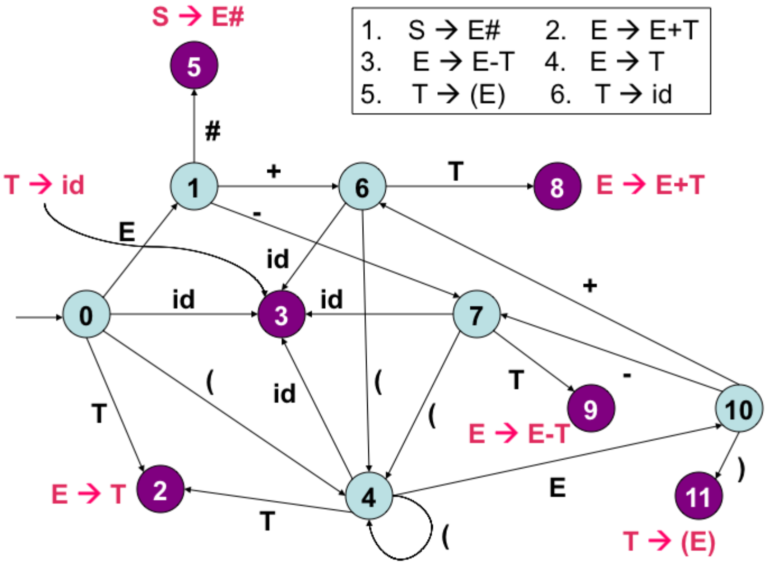
Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560 012

NPTEL Course on Principles of Compiler Design

Outline of the Lecture

- What is syntax analysis? (covered in lecture 1)
- Specification of programming languages: context-free grammars (covered in lecture 1)
- Parsing context-free languages: push-down automata (covered in lectures 1 and 2)
- Top-down parsing: LL(1) parsing (covered in lectures 2 and 3)
- Recursive-descent parsing (covered in lecture 4)
- Bottom-up parsing: LR-parsing (continued)

DFA for Viable Prefixes - LR(0) Automaton



Construction of Sets of Canonical LR(0) Items

void $\mathit{Set\_of\_item\_sets}(G')$\{ /* G' is the augmented grammar */
  $C = \{closure(\{S' \rightarrow .S\})\}$;/* C is a set of item sets */
  while (more item sets can be added to C) \{
    for each item set $I \in C$ and each grammar symbol X
    /* X is a grammar symbol, a terminal or a nonterminal */
      if $((GOTO(I, X) \neq \emptyset)$ && $(GOTO(I, X) \notin C))$
        $C = C \cup GOTO(I, X)$
  \}
\}

- Each set in C (above) corresponds to a state of a DFA (LR(0) DFA)
- This is the DFA that recognizes viable prefixes

Construction of an LR(0) Automaton - Example 1

State 0
S → .E#
E → .E+T
E → .E-T
E → .T
T → .(E)
T → .id

State 1
S → E.#
E → E.+T
E → E.-T

State 2
E → T.

State 3
T → id.

State 4
T → (.E)
E → .E+T
E → .E-T
E → .T
T → .(E)
T → .id

State 5
S → E#.

State 6
E → E+.T
T → .(E)
T → .id

State 7
E → E-.T
T → .(E)
T → .id

State 8
E → E+T.

State 9
E → E-T.

State 10
T → (E.)
E → E.+T
E → E.-T

State 11
T → (E).

indicates closure items

indicates kernel items

Shift and Reduce Actions

- If a state contains an item of the form $[A \rightarrow \alpha.]$ (“reduce item”), then a reduction by the production $A \rightarrow \alpha$ is the action in that state
- If there are no “reduce items” in a state, then shift is the appropriate action
- There could be shift-reduce conflicts or reduce-reduce conflicts in a state
  - Both shift and reduce items are present in the same state (S-R conflict), or
  - More than one reduce item is present in a state (R-R conflict)
  - It is normal to have more than one shift item in a state (no shift-shift conflicts are possible)
- If there are no S-R or R-R conflicts in any state of an LR(0) DFA, then the grammar is LR(0), otherwise, it is not LR(0)

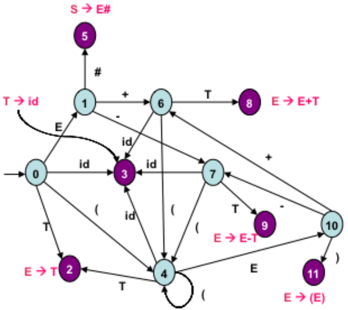
LR(0) Parser Table - Example 1

STATE	ACTION						GOTO		
	+	-	(	)	id	#	S	E	T
0			S4		S3			1	2
1	S6	S7				S5			
2	R4	R4	R4	R4	R4	R4			
3	R6	R6	R6	R6	R6	R6			
4			S4		S3			10	2
5	R1 acc	R1 acc	R1 acc	R1 acc	R1 acc	R1 acc			
6			S4		S3				8
7			S4		S3				9
8	R2	R2	R2	R2	R2	R2			
9	R3	R3	R3	R3	R3	R3			
10	S6	S7		S11					
11	R5	R5	R5	R5	R5	R5			

1. S → E#
2. E → E+T
3. E → E-T
4. E → T
5. T → (E)
6. T → id

Construction of an LR(0) Parser Table - Example 1

STATE	ACTION						GOTO		
	+	-	(	)	id	#	S	E	T
0			S4		S3			1	2
1	S6	S7				S5			
2	R4	R4	R4	R4	R4	R4			
3	R6	R6	R6	R6	R6	R6			
4			S4		S3			10	2
5	R1 acc	R1 acc	R1 acc	R1 acc	R1 acc	R1 acc			
6			S4		S3				8
7			S4		S3				9
8	R2	R2	R2	R2	R2	R2			
9	R3	R3	R3	R3	R3	R3			
10	S6	S7		S11					
11	R5	R5	R5	R5	R5	R5			



1. S → E#
2. E → E+T
3. E → E-T
4. E → T
5. T → (E)
6. T → id

State 0
S → .E#
E → .E+T
E → .E-T
E → .T
T → .(E)
T → .id

State 1
S → E.#
E → E.+T
E → E.-T

State 2
E → T.

State 3
T → id.

State 6
E → E+.T
T → .(E)
T → .id

State 4
T → (.E)
E → .E+T
E → .E-T
E → .T
T → .(E)
T → .id

State 5
S → E#.

State 7
E → E-.T
T → .(E)
T → .id

State 8
E → E+T.

State 9
E → E-T.

State 10
T → (E.)
E → E.+T
E → E.-T

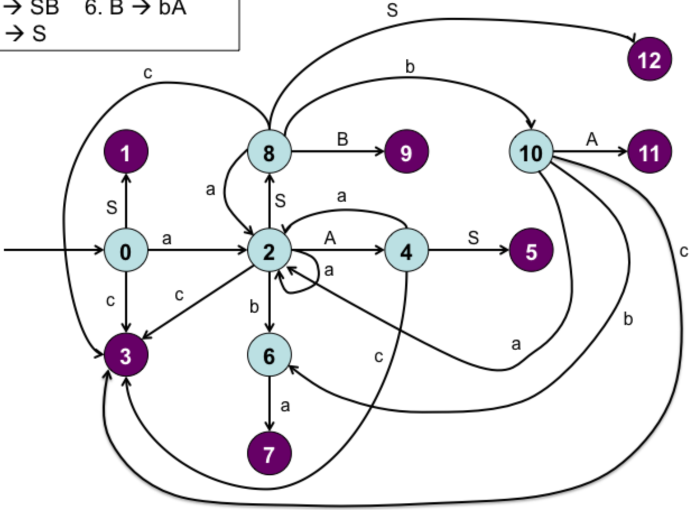
State 11
T → (E).

indicates closure items

indicates kernel items

LR(0) Automaton - Example 2

1. S' → S 2. S → aAS
3. S → c 4. A → ba
5. A → SB 6. B → bA
7. B → S



Construction of an LR(0) Automaton - Example 2

State 0
S' → .S
S → .aAS
S → .c

State 1
S' → S.

State 2
S → a.AS
A → .ba
A → .SB
S → .aAS
S → .c

State 3
S → c.

State 4
S → aA.S
S → .aAS
S → .c

State 5
S → aAS.

State 6
A → b.a

State 7
A → ba.

State 8
A → S.B
B → .bA
B → .S
S → .aAS
S → .c

State 9
A → SB.

State 10
B → b.A
A → .ba
A → .SB
S → .aAS
S → .c

State 11
B → bA.

State 12
B → S.

● (orange) indicates closure items

● (black) indicates kernel items

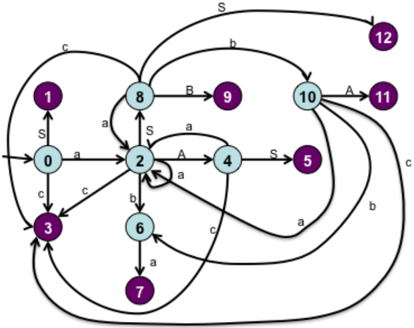
1. S' → S	2. S → aAS	3. S → c
4. A → ba	5. A → SB	
6. B → bA	7. B → S	

LR(0) Parser Table - Example 2

STATE	ACTION				GOTO		
	a	b	c	$	S	A	B
0	S2		S3		1		
1				R1 acc			
2	S2	S6	S3		8	4	
3	R3	R3	R3	R3			
4	S2		S3		5		
5	R2	R2	R2	R2			
6	S7						
7	R4	R4	R4	R4			
8	S2	S10	S3		12		9
9	R5	R5	R5	R5			
10	S2	S6	S3		8	11	
11	R6	R6	R6	R6			
12	R7	R7	R7	R7			

1. S' → S
2. S → aAS
3. S → c
4. A → ba
5. A → SB
6. B → bA
7. B → S

STATE	ACTION				GOTO		
	a	b	c	$	S	A	B
0	S2		S3		1		
1				R1 acc			
2	S2	S6	S3		8	4	
3	R3	R3	R3	R3			
4	S2		S3		5		
5	R2	R2	R2	R2			
6	S7						
7	R4	R4	R4	R4			
8	S2	S1 0	S3		12		9
9	R5	R5	R5	R5			
10	S2	S6	S3		8	11	
11	R6	R6	R6	R6			
12	R7	R7	R7	R7			



1. S' → S
2. S → aAS
3. S → c
4. A → ba
5. A → SB
6. B → bA
7. B → S

State 0
S' → .S
S → .aAS
S → .c

State 1
S' → S.

State 2
S → a.AS
A → .ba
A → .SB
S → .aAS
S → .c

State 3
S → c.

State 4
S → aA.S
S → .aAS
S → .c

State 5
S → aAS.

State 6
A → b.a

State 7
A → ba.

State 9
A → SB.

State 8
A → S.B
B → .bA
B → .S
S → .aAS
S → .c

State 11
B → bA.

State 10
B → b.A
A → .ba
A → .SB
S → .aAS
S →.c

State 12
B → S.

indicates closure items   indicates kernel items

A Grammar that is not LR(0) - Example 1

State 0
S → .E
E → .E+T
E → .E-T
E → .T
T → .(E)
T → .id

State 1
S → E.
E → E.+T
E → E.-T

shift-reduce conflicts in state 1

State 2
E → T.

State 3
T → id.

State 4
T → (.E)
E → .E+T
E → .E-T
E → .T
T → .(E)
T → .id

State 5
E → E+.T
T → .(E)
T → .id

State 6
E → E-.T
T → .(E)
T → .id

State 7
E → E+T.

State 8
E → E-T.

State 9
T → (E.)
E → E.+T
E → E.-T

State 10
T → (E).

● (orange) indicates closure items

● (black) indicates kernel items

follow(S) = {$}, where $ is EOF
Reduction on $, and shifts on + and - , will resolve the conflicts
This is similar to having an end marker such as #

Grammar is not LR(0), but is SLR(1)

SLR(1) Parsers

- If the grammar is not LR(0), we try to resolve conflicts in the states using one look-ahead symbol
- Example: The expression grammar that is not LR(0)
  The state containing the items $[T \rightarrow F.]$ and $[T \rightarrow F. * T]$ has S-R conflicts
  - Consider the reduce item $[T \rightarrow F.]$ and the symbols in $FOLLOW(T)$
  - $FOLLOW(T) = \{+,), \$\}$, and reduction by $T \rightarrow F$ can be performed on seeing one of these symbols in the input (look-ahead), since shift requires seeing $*$ in the input
  - Recall from the definition of $FOLLOW(T)$ that symbols in $FOLLOW(T)$ are the only symbols that can legally follow T in any sentential form, and hence reduction by $T \rightarrow F$ when one of these symbols is seen, is correct
  - If the S-R conflicts can be resolved using the FOLLOW set, the grammar is said to be SLR(1)

A Grammar that is not LR(0) - Example 2

State 0
S → .E
E → .E+T
E → .T
T → .F*T
T → .F
F → .(E)
F → .id

State 1
S → E.
E → E.+T
Shift-reduce conflict

State 2
E → T.

State 3
T → F.*T
T → F.
Shift-reduce conflict

State 4
F → (.E)
E → .E+T
E → .T
T → .F*T
T → .F
F → .(E)
F → .id

State 5
F → id.

State 6
E → E+.T
T → .F*T
T → .F
F → .(E)
F → .id

State 7
T → F*.T
T → .F*T
T → .F
F → .(E)
F → .id

State 8
F → (E.)
E → E.+T

State 9
E → E+T.

State 10
E → F*T.

State 11
F → (E).

follow(S) = {$}, Reduction on $ and shift on +, eliminates conflicts
follow(T) = {$,), +}, where $ is EOF
Reduction on $,), and +, and shift on *, eliminates conflicts

Grammar is not LR(0), but is SLR(1)

Construction of an SLR(1) Parsing Table

Let $C = \{I_0, I_1, ..., I_i, ..., I_n\}$ be the canonical LR(0) collection of items, with the corresponding states of the parser being 0, 1, ... , i, ... , n
Without loss of generality, let 0 be the initial state of the parser (containing the item $[S' \rightarrow .S]$)
Parsing actions for state *i* are determined as follows
1. If $([A \rightarrow \alpha.a\beta] \in I_i)$ && $([A \rightarrow \alpha a.\beta] \in I_j)$
   set ACTION[i, a] = *shift j* /* *a* is a terminal symbol */
2. If $([A \rightarrow \alpha.] \in I_i)$
   set ACTION[i, a] = *reduce* $A \rightarrow \alpha$, for all $a \in follow(A)$
3. If $([S' \rightarrow S.] \in I_i)$ set ACTION[i, \$] = *accept*

S-R or R-R conflicts in the table imply grammar is not SLR(1)
4. If $([A \rightarrow \alpha.A\beta] \in I_i)$ && $([A \rightarrow \alpha A.\beta] \in I_j)$
   set GOTO[i, A] = *j* /* *A* is a nonterminal symbol */

All other entries not defined by the rules above are made *error*

A Grammar that is not LR(0) - Example 3

Grammar
$S' \rightarrow S,\ \ S \rightarrow aSb,\ \ S \rightarrow \epsilon$

follow(S) = {$, b}

State 0
S' → .S
S → .aSb
S → .

State 3
S → aS.b

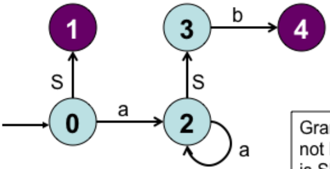
State 1
S' → S.

State 4
S → aSb.

State 2
S → a.Sb
S → .aSb
S → .

shift-reduce conflict in states 0, 2

	a	b	$	S
0	S2	reduce S → ε	reduce S → ε	1
1			accept	
2	S2	reduce S → ε	reduce S → ε	3
3		S4		
4		reduce S → aSb	reduce S → aSb	



Grammar is not LR(0), but is SLR(1)

● indicates closure items

● indicates kernel items

A Grammar that is not SLR(1) - Example 1

Grammar: S' → S,
S → aSb, S → ab, S → ε

follow(S) = {$, b}
State 0: Reduction on *$* and *b*, by S → ε, and shift on *a* resolves conflicts
State 2: S-R conflict on *b* still remains

State 0
S' → .S
S → .aSb
S → .ab
S → .

State 1
S' → S.

State 2
S → a.Sb
S → a.b
S → .aSb
S → .ab
S → .

State 3
S → aS.b

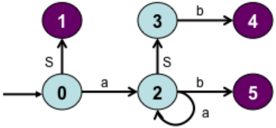
State 4
S → aSb.

State 5
S → ab.

shift-reduce conflict in states 0, 2

Grammar is neither LR(0) nor SLR(1)

	a	b	$	S
0	S2	R: S → ε	R: S → ε	1
1			accept	
2	S2	**S5, R: S → ε**	R: S → ε	3
3		S4		
4		R: S → aSb	R: S → aSb	
5		R: S → ab	R: S → ab	



A Grammar that is not SLR(1) - Example 2

Grammar
S' → S
S → L=R
S → R
L → *R
L → id
R → L

Grammar is neither LR(0) nor SLR(1)

State 0
S' → .S
S → .L=R
S → .R
L → .*R
L → .id
R → .L

State 1
S' → S.

State 3
S → R.

State 2
S → L .=R
R → L .
shift-reduce conflict

State 4
L → *.R
R → .L
L → .*R
L → .id

State 5
L → id.

State 6
S → L=.R
R → .L
L → .*R
L → .id

State 7
L → *R.

State 8
R → L .

State 9
S → L=R.

Follow(R) = {$,=} does not resolve S-R conflict

The Problem with SLR(1) Parsers

- SLR(1) parser construction process does not remember enough left context to resolve conflicts
  - In the "$L = R$" grammar (previous slide), the symbol '=' got into follow(R) because of the following derivation: $S' \Rightarrow S \Rightarrow L = R \Rightarrow L = L \Rightarrow L = id \Rightarrow *\underline{R =} id \Rightarrow ...$
  - The production used is $L \rightarrow *R$
  - The following rightmost derivation in *reverse* does not exist (and hence reduction by $R \rightarrow L$ on '=' in state 2 is illegal) $id = id \Leftarrow L = id \Leftarrow R = id ...$
- Generalization of the above example
  - In some situations, when a state i appears on top of the stack, a viable prefix $\beta\alpha$ may be on the stack such that βA cannot be followed by 'a' in any right sentential form
  - Thus, the reduction by $A \rightarrow \alpha$ would be invalid on 'a'
  - In the above example, β = ϵ, α = L, and $A = R$; L cannot be reduced to R on '=', since it would lead to the above illegal derivation sequence

LR(1) Parsers

- LR(1) items are of the form $[A \rightarrow \alpha.\beta,\ a]$, a being the "lookahead" symbol
- Lookahead symbols have no part to play in shift items, but in reduce items of the form $[A \rightarrow \alpha.,\ a]$, reduction by $A \rightarrow \alpha$ is valid only if the next input symbol is 'a'
- An LR(1) item $[A \rightarrow \alpha.\beta,\ a]$ is *valid* for a viable prefix γ, if there is a derivation $S \Rightarrow^*_{rm} \delta A w \Rightarrow_{rm} \delta\alpha\beta w$, where, $\gamma = \delta\alpha$, $a = first(w)$ or $w = \epsilon$ and $a = \$$
- Consider the grammar: $S' \rightarrow S,\ S \rightarrow aSb \mid \epsilon$
  - $[S \rightarrow a.Sb,\ \$]$ is valid for the VP a, $S' \Rightarrow S \Rightarrow aSb$
  - $[S \rightarrow a.Sb,\ b]$ is valid for the VP aa, $S' \Rightarrow S \Rightarrow aSb \Rightarrow aaSbb$
  - $[S \rightarrow .,\ \$]$ is valid for the VP ϵ, $S' \Rightarrow S \Rightarrow \epsilon$
  - $[S \rightarrow aSb.,\ b]$ is valid for the VP $aaSb$, $S' \Rightarrow S \Rightarrow aSb \Rightarrow aaSbb$

LR(1) Grammar - Example 1

Grammar
S' → S, S → aSb, S → ε

State 0
S' → .S , $
S → .aSb , $
S → . , $

State 1
S' → S. , $

State 2
S → a.Sb , $
S → .aSb , b
S → . , b

State 3
S → aS.b , $

State 4
S → a.Sb , b
S → .aSb , b
S → . , b

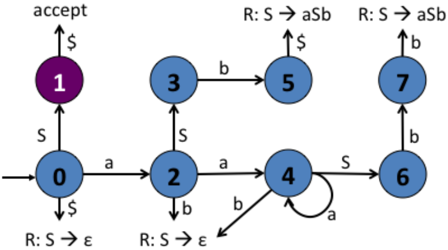
State 5
S → aSb. , $

State 6
S → aS.b , b

State 7
S → aSb. , b

Grammar is LR(1)

	a	b	$	S
0	S2		R: S → ε	1
1			accept	
2	S4	R: S → ε		3
3		S5		
4	S4	R: S → ε		6
5			R: S → aSb	
6		S7		
7		R: S → aSb		



Closure of a Set of LR(1) Items

Itemset closure(I){ /* I is a set of LR(1) items */
    while (more items can be added to I) {
        for each item $[A \to \alpha.B\beta,\ a] \in I$ {
            for each production $B \to \gamma \in G$
                for each symbol $b \in \mathit{first}(\beta a)$
                    if (item $[B \to .\gamma,\ b] \notin I$) add item $[B \to .\gamma,\ b]$ to I
    }
    return I
}

Grammar	**State 0**	**State 3**	**State 4**	**State 7**
S' → S	S' → .S , $	S → aS.b , $	S → a.Sb , b	S → aSb. , b
S → aSb | ε	S → .aSb , $		S → .aSb , b	
	S → . , $		S → . , b	

GOTO set computation

Itemset GOTO(I, X){ /* I is a set of LR(1) items
X is a grammar symbol, a terminal or a nonterminal */
Let I' = $\{[A \rightarrow \alpha X.\beta,\ a] \mid [A \rightarrow \alpha.X\beta,\ a] \in I\}$;
return (*closure*(I'))
}

Grammar	**State 0**	**State 1**	**State 2**	**State 4**
S' → S	**S' → .S , $**	**S' → S. , $**	**S → a.Sb , $**	**S → a.Sb , b**
S → aSb | ε	**S → .aSb , $**		**S → .aSb , b**	**S → .aSb , b**
	S → . , $		**S → . , b**	**S → . , b**

GOTO(0, S) = 1, GOTO(0,a) = 2, GOTO(2,a) = 4

Construction of Sets of Canonical of LR(1) Items

void Set_of_item_sets(G'){ /* G' is the augmented grammar */
C = {$closure(\{S' \rightarrow .S,\ \$\})$};/* C is a set of LR(1) item sets */
while (more item sets can be added to C) {
for each item set $I \in C$ and each grammar symbol X
/* X is a grammar symbol, a terminal or a nonterminal */
if $((GOTO(I, X) \neq \emptyset)$ && $(GOTO(I, X) \notin C))$
$C = C \cup GOTO(I, X)$
}
}

- Each set in C (above) corresponds to a state of a DFA (LR(1) DFA)
- This is the DFA that recognizes viable prefixes

LR(1) DFA Construction - Example 1

Grammar
S' → S, S → aSb, S → ε

State 0
S' → .S , $
S → .aSb , $
S → . , $

State 1
S' → S. , $

State 2
S → a.Sb , $
S → .aSb , b
S → . , b

State 3
S → aS.b , $

State 4
S → a.Sb , b
S → .aSb , b
S → . , b

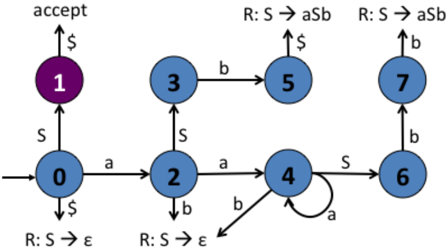
State 5
S → aSb. , $

State 6
S → aS.b , b

State 7
S → aSb. , b

Grammar is LR(1)

	a	b	$	S
0	S2		R: S → ε	1
1			accept	
2	S4	R: S → ε		3
3		S5		
4	S4	R: S → ε		6
5			R: S → aSb	
6		S7		
7		R: S → aSb		



Construction of an LR(1) Parsing Table

Let $C = \{I_0, I_1, ..., I_i, ..., I_n\}$ be the canonical LR(1) collection of items, with the corresponding states of the parser being 0, 1, ... , i, ... , n

Without loss of generality, let 0 be the initial state of the parser (containing the item $[S' \rightarrow .S,\ \$]$)

Parsing actions for state *i* are determined as follows

1. If $([A \rightarrow \alpha.a\beta,\ b] \in I_i)$ && $([A \rightarrow \alpha a.\beta,\ b] \in I_j)$
   set ACTION[i, a] = *shift j* /* *a* is a terminal symbol */
2. If $([A \rightarrow \alpha.,\ a] \in I_i)$
   set ACTION[i, a] = *reduce* $A \rightarrow \alpha$
3. If $([S' \rightarrow S.,\ \$] \in I_i)$ set ACTION[i, \$] = *accept*

S-R or R-R conflicts in the table imply grammar is not LR(1)

4. If $([A \rightarrow \alpha.A\beta,\ a] \in I_i)$ && $([A \rightarrow \alpha A.\beta,\ a] \in I_j)$
   set GOTO[i, A] = *j* /* *A* is a nonterminal symbol */

All other entries not defined by the rules above are made *error*