LR Parsing

$LR(0)$ Parsing

An $LR(0)$ parser can take *shift-reduce* decisions entirely on the basis of the states of $LR(0)$ *automaton*[a] of the grammar. Consider the following grammar and its augmented form.

[a] The parsing table can be filled from the automaton.

Example

The production rules are,
$S \rightarrow aSa \mid bSb \mid c$
The production rules of the augmented grammar are,
$S' \rightarrow S$
$S \rightarrow aSa \mid bSb \mid c$
The states of the $LR(0)$ automaton are the following:

States

q_0 :	$S' \rightarrow \bullet S\$ \quad S \rightarrow \bullet aSa \quad S \rightarrow \bullet bSb$ $S \rightarrow \bullet c$
q_1 :	$S' \rightarrow S \bullet \$$
q_2 :	$S \rightarrow a \bullet Sa \quad S \rightarrow \bullet aSa \quad S \rightarrow \bullet bSb$ $S \rightarrow \bullet c$
q_3 :	$S \rightarrow b \bullet Sb \quad S \rightarrow \bullet aSa \quad S \rightarrow \bullet bSb$ $S \rightarrow \bullet c$
q_4 :	$S \rightarrow c\bullet$

States

q_5 :	$S \rightarrow aS \bullet a$
q_6 :	$S \rightarrow bS \bullet b$
q_7 :	$S \rightarrow aSa\bullet$
q_8 :	$S \rightarrow bSb\bullet$

Complete and Incomplete Items

An $LR(0)$ item is called *complete* if the '$\bullet$' is at the right end of the production, $A \rightarrow \alpha\bullet$. This indicates that the DFA has already 'seen' a *handle* and it is on the top of the stack.

Note

The interesting property of this $LR(0)$ DFA is that, no state is formed with both *complete* and *incomplete* items. Also, no state has two complete items.
A state with a unique complete item $A \rightarrow \alpha\bullet$, indicates a reduce action using the rule $A \rightarrow \alpha$ by the parser.
A state with incomplete items indicates a shift action by the parser. The parsing table for the given grammar is as follows.

Parsing table

State	Action				Goto
	a	b	c	\$	S
0	s_2	s_3	s_4		1
1				*accept*	
2	s_2	s_3	s_4		5
3	s_2	s_3	s_4		6
4	r_3	r_3	r_3	r_3	
5	s_7				
6		s_8			
7	r_1	r_1	r_1	r_1	
8	r_2	r_2	r_2	r_2	

Note

The parser does not *look-ahead* for any *shift* operation. It gets the current state from the *top-of-stack* and the *token* from the scanner. Using the parsing table it gets the next state and pushes it in the stack[a]. The token is consumed.

[a] It may push the token and its attributes in the value stack for semantic action.

Note

In case of $LR(0)$ parser it does not *look-ahead* even for any *reduce* operation[a]. It gets the current state from the *top-of-stack* and the production rule number from the parsing table (for all correct input they are same), and reduces the *right sentential form* by the rule[b].

[a] It may read the input to detect error. Note the column corresponding to '`c`' for the states 4, 7, 8 with unique complete items.

[b] The Goto portion of the table is used to push a new state on the stack after a reduction.

Parsing Example

Stack	Input	Handle	Action
$\$q_0$	aabcbaa$	*nil*	s_2
$\$q_0q_2$	abcbaa$	*nil*	s_2
$\$q_0q_2q_2$	bcbaa$	*nil*	s_3
$\$q_0q_2q_2q_3$	cbaa$	*nil*	s_4
$\$q_0q_2q_2q_3q_4$	baa$	$S \rightarrow c$	r_3

[a]

[a] The length of $|c| = 1$, so q_4 is popped out and $\text{Goto}(q_3, S) = q_6$ is pushed in the stack.

Parsing Example

Stack	Input	Handle	Action
$\$q_0q_2q_2q_3q_4$	baa$	$S \rightarrow c$	r_3
$\$q_0q_2q_2q_3q_6$	baa$	*nil*	s_8
$\$q_0q_2q_2q_3q_6q_8$	aa$	$S \rightarrow bSb$	r_2
$\$q_0q_2q_2q_5$	aa$	*nil*	s_7
$\$q_0q_2q_2q_5q_7$	a$	$S \rightarrow aSa$	r_1

[a]

[a] The length of $|bSb| = 3$, so $q_3q_6q_8$ are popped out and $\text{Goto}(q_2, S) = q_5$ is pushed in the stack.

Parsing Example

Stack	Input	Handle	Action
$\$q_0q_2q_2q_5q_7$	a$	$S \rightarrow aSa$	r_1
$\$q_0q_2q_5$	a$	*nil*	s_7
$\$q_0q_2q_5q_7$	$	$S \rightarrow aSa$	r_1
$\$q_0q_1$	$	$S' \rightarrow S$	*accept*

[a] The length of $|aSa| = 3$, so $q_2q_5q_7$ are popped out and $\text{Goto}(q_2, S) = q_5$ is pushed in the stack. Similarly, $\text{Goto}(q_0, S) = q_1$ is pushed in the stack.

$SLR(1)$ Parsing

We consider our old grammar (augmented with S').

$$
\begin{aligned}
0: &\quad S' \rightarrow P \\
1: &\quad P \rightarrow m\ L\ s\ e \\
2: &\quad L \rightarrow D\ L \\
3: &\quad L \rightarrow D \\
4: &\quad D \rightarrow T\ V\ ; \\
5: &\quad V \rightarrow d\ V \\
6: &\quad V \rightarrow d \\
7: &\quad T \rightarrow i \\
8: &\quad T \rightarrow f
\end{aligned}
$$

States

q_0 :	$S' \rightarrow \bullet P$	$P \rightarrow \bullet m\ L\ s\ e$	
q_1 :	$S' \rightarrow P\bullet$		
q_2 :	$P \rightarrow m \bullet L\ s\ e$	$L \rightarrow \bullet D\ L$	$L \rightarrow \bullet D$
	$D \rightarrow \bullet T\ V\ ;$	$T \rightarrow \bullet i$	$T \rightarrow \bullet f$
q_3 :	$P \rightarrow m\ L \bullet s\ e$		
q_4 :	$L \rightarrow D \bullet L$	$L \rightarrow D\bullet$	$L \rightarrow \bullet D\ L$
	$L \rightarrow \bullet D$	$D \rightarrow \bullet T\ V\ ;$	$T \rightarrow \bullet i$
	$T \rightarrow \bullet f$		

States

q_5 :	$D \rightarrow T \bullet V$; $\quad V \rightarrow \bullet d\ V \quad V \rightarrow \bullet d$
q_6 :	$T \rightarrow i\bullet$
q_7 :	$T \rightarrow f\bullet$
q_8 :	$P \rightarrow m\ L\ s \bullet e$
q_9 :	$L \rightarrow D\ L\bullet$
q_{10} :	$D \rightarrow T\ V\bullet;$
q_{11} :	$V \rightarrow d \bullet V \quad V \rightarrow d\bullet \quad V \rightarrow \bullet d\ V$ $V \rightarrow \bullet d$

States

q_{12} :	$P \rightarrow m\ L\ s\ e\bullet$
q_{13} :	$D \rightarrow T\ V\ ;\bullet$
q_{14} :	$V \rightarrow d\ V\bullet$

Note

In the $LR(0)$ automaton of the grammar two states, q_4 and q_{11} have both *complete* and *incomplete* items. So the grammar is not of type $LR(0)$.

Note

Consider the state q_4.
The *complete item* is $L \rightarrow D\bullet$ and the *incomplete items* are $T \rightarrow \bullet i$ and $T \rightarrow \bullet f$.
The Follow$(L) = \{s\}$ is different from i, f. So we can put Action$(4, i) = s_6$, Action$(4, f) = s_7$ and Action$(4, s) = r_3$ (reduce by the production rule number 3) in the parsing table.

SLR Parsing Table: *Action*

- If $A \rightarrow \alpha \bullet a\beta \in q_i$ $(a \in \Sigma)$ and $Goto(q_i, a) = q_j$, then $\text{Action}(i, a) = s_j$.
- If $A \rightarrow \alpha\bullet \in q_i$ $(A \neq S')$ and $b \in \text{Follow}(A)$, then $\text{Action}(i, b) = r_k$, where k is the rule number of $A \rightarrow \alpha$.
- If $S' \rightarrow S\bullet \in q_i$, then $\text{Action}(i, \$) = accept$.

Note

If the process does not lead to a table with multiple entries, the grammar is SLR (simple LR).

SLR Parsing Table: *Goto*

If $A \rightarrow \alpha \bullet B\beta \in q_i$ $(B \in N)$ and
$\text{Goto}(q_i, B) = q_j$, then in the table
$\text{Goto}(i, B) = j$.
All other entries of the table are errors.

FOLLOW() Sets

Non-terminal	Follow
P	\$
L	s
D	i, f, s
T	d
V	;

SLR Parsing Table

S	*Action*								*Goto*				
	m	s	e	;	d	i	f	\$	P	L	D	V	T
0	s_2								1				
1								A					
2						s_6	s_7			3	4		5
3		s_8											
4		r_3				s_6	s_7			9	4		5
5					s_{11}							10	

Example

S	*Action*								*Goto*				
	m	s	e	$;$	d	i	f	$\$$	P	L	D	V	T
6					r_7								
7					r_8								
8				s_{12}									
9		r_2											
10				s_{13}									
11				r_6	s_{11}							14	

Example

S	*Action*								*Goto*				
	m	s	e	$;$	d	i	f	$\$$	P	L	D	V	T
12								r_1					
13		r_4				r_4	r_4						
14				r_5									

Non-SLR Grammar

Consider the following grammar G_{rr} (augmented by the S').

$$0: \quad S' \rightarrow S$$

$$1: \quad S \rightarrow E + T$$

$$2: \quad S \rightarrow T$$

$$3: \quad T \rightarrow i * E$$

$$4: \quad T \rightarrow i$$

$$5: \quad E \rightarrow T$$

States

The states of the $LR(0)$ automaton are as follows:

q_0 :	$S' \rightarrow \bullet S$ $S \rightarrow \bullet E + T$ $S \rightarrow \bullet T$ $E \rightarrow \bullet T$ $T \rightarrow \bullet i * E$ $T \rightarrow \bullet i$
q_1 :	$S' \rightarrow S\bullet$
q_2 :	$S \rightarrow E \bullet + T$
q_3 :	$S \rightarrow T\bullet$ $E \rightarrow T\bullet$
q_4 :	$T \rightarrow i \bullet * E$ $T \rightarrow i\bullet$
q_5 :	$S \rightarrow E + \bullet T$ $T \rightarrow \bullet i * E$ $T \rightarrow \bullet i$

States

q_6 :	$T \rightarrow i * \bullet E \quad E \rightarrow \bullet T \quad T \rightarrow \bullet i * E$ $T \rightarrow \bullet i$
q_7 :	$S \rightarrow E + T\bullet$
q_8 :	$T \rightarrow i * E\bullet$
q_9 :	$E \rightarrow T\bullet$

Note

The state q_3 has two complete items $S \rightarrow T\bullet$ and $E \rightarrow T\bullet$. When the automaton is in this state, there are two possible handles on the stack. The question is how to decide about the reduction rule.
If $\text{Follow}(S) \cap \text{Follow}(E) = \emptyset$, then the choice can be made on the basis of the look-ahead symbol.
But in this case $\text{Follow}(S) = \{\$\}$ and $\text{Follow}(E) = \{\$, +\}$.
The *Action* part of the SLR table corresponding to the *row-*q_3 and the *column-*\$ will have two entries - the grammar is non-SLR.

Note

In this example Action$[q_3][\$] = \{r_2, r_5\}$, the *frontier* can be reduced by *rule 2* or by *rule 5*. This is known as *reduce/reduce conflict* of SLR table. We shall see how a more powerful parsing technique can be used to resolve this conflict. The information in the *Follow()* set is not sufficient to resolve this conflict.

Consider the grammar G_{sr} (augmented by the S').

$$0: \; S' \rightarrow A$$

$$1: \; A \rightarrow B\ a$$

$$2: \; A \rightarrow C\ b$$

$$3: \; A \rightarrow a\ C\ a$$

$$4: \; C \rightarrow B$$

$$5: \; B \rightarrow c\ A$$

$$6: \; B \rightarrow b$$

States

Some of the states of the $LR(0)$ automaton are as follows:

q_0 :	$S' \rightarrow \bullet A$ $A \rightarrow \bullet B\ a$ $A \rightarrow \bullet C\ b$ $A \rightarrow \bullet a\ C\ a$ $B \rightarrow \bullet c\ A$ $B \rightarrow \bullet b$ $C \rightarrow \bullet B$
q_1 :	$S' \rightarrow A\bullet$
q_2 :	$A \rightarrow B \bullet a$ $C \rightarrow B\bullet$
q_3 :	$A \rightarrow C \bullet b$
q_4 :	$A \rightarrow a \bullet C\ a$ $C \rightarrow \bullet B$

States

q_5 :	$B \rightarrow c \bullet A \quad A \rightarrow \bullet B\ a \quad A \rightarrow \bullet C\ b$ $A \rightarrow \bullet a\ C\ a \quad B \rightarrow \bullet c\ A \quad B \rightarrow \bullet b$ $C \rightarrow \bullet B$
q_6 :	$B \rightarrow b\bullet$
q_7 :	$A \rightarrow B\ a\bullet$
q_8 :	
q_9 :	

Note

The state q_2 has one complete item, $C \rightarrow B\bullet$ and one incomplete item, $A \rightarrow B \bullet a$. When the automaton is in this state, the parser is to decide whether to *shift* the input or to *reduce*. Unfortunately, the SLR parsing table will have two entries for $Action[q_2][a] = \{s_7, r_4\}$, as $a \in$ Follow(C).

The grammar is non-SLR and the problem is known as *shift-reduce* conflict of SLR parsing table. We shall see how a powerful technique can be used to build a parsing table.

Note

If the state of an $LR(0)$ automaton contains a complete item $A \rightarrow \alpha\bullet$ and the next input $a \in$ FOLLOW(A), the action in the SLR table will be *reduction of handle* by the rule $A \rightarrow \alpha$.
But the set FOLLOW(A) is the super set of what can follow a particular viable prefix. In the grammar G_{rr}, the *viable prefix* E cannot be followed by a \$ and similarly the the *viable prefix* S cannot be followed by a $+$.
So the *reduce/reduce* conflict of q_3 can be resolved by explicitly carrying the look-ahead information.

Canonical $LR(1)$ Item

An object of the form $(A \rightarrow \alpha \bullet \beta, a)$, where $A \rightarrow \alpha\beta$ is a production rule and $a \in \Sigma \cup \{\$\}$, is called an $LR(1)$ item. The second component[a] is known as the look-ahead symbol.

[a] The '1' in $LR(1)$ refers to the second component of the item.

Reduction

The *look-ahead* symbol of an $LR(1)$ item $(A \rightarrow \alpha \bullet \beta, a)$ is important when the item is complete i.e. $\beta = \varepsilon$.
The reduction by the rule $A \rightarrow \alpha$ can take place if the *look-ahead* symbol is '`a`'.
The *look-ahead* symbol '`a`' is an element of FOLLOW(A), but we carry it explicitly with the item.

Valid Item

An $LR(1)$ item $(A \rightarrow \alpha \bullet \beta, a)$ is *valid* for a viable prefix '$u\alpha$', if there is a *rightmost derivation*: $S \rightarrow uAx \rightarrow u\alpha\beta x$, so that $a \in \text{FIRST}(x)$ or if $x = \varepsilon$, then $a = \$$.

Closure()

If i is an $LR(1)$ item, then Closure(i) is defined as follows:

- $i \in$ Closure(i) - basis,
- If $(A \rightarrow \alpha \bullet B\beta, a) \in$ Closure(i) and $B \rightarrow \gamma$ is a production rule, then $(B \rightarrow \bullet\gamma, b) \in$ Closure(i), where $b \in$ FIRST(βa).

Closure()

The closure of I, a set of $LR(1)$ items, is defined as Closure$(I) = \cup_{i \in I}$ Closure(i).

Goto(I, X)

Let I be a set of $LR(1)$ items and $X \in \Sigma \cup N$.

The set of $LR(1)$ items Goto(I, X) is

$$\text{Closure}\left(\{(A \rightarrow \alpha\ X \bullet \beta, a) :\ (A \rightarrow \alpha \bullet X\ \beta, a) \in I\}\right).$$

$LR(1)$ Automaton

The start state of the $LR(1)$ automaton is $Closure(S' \rightarrow \bullet S, \$)$. Other reachable and final states can be constructed by computing GOTO() of already existing states. This is a *fixed-point computation*.

Consider the grammar G_{rr} (augmented by the S').

$$0: \quad S' \rightarrow S$$

$$1: \quad S \rightarrow E + T$$

$$2: \quad S \rightarrow T$$

$$3: \quad T \rightarrow i * E$$

$$4: \quad T \rightarrow i$$

$$5: \quad E \rightarrow T$$

States

The states of the $LR(1)$ automaton are as follows:

q_0 :	$S' \rightarrow \bullet S, \$$	$S \rightarrow \bullet E + T, \$$	$S \rightarrow \bullet T, \$$
	$E \rightarrow \bullet T, +$	$T \rightarrow \bullet i * E, +/\$$	$T \rightarrow \bullet i, +/\$$
q_1 :	$S' \rightarrow S\bullet, \$$		
q_2 :	$S \rightarrow E \bullet + T, \$$		
q_3 :	$S \rightarrow T\bullet, \$$	$E \rightarrow T\bullet, +$	
q_4 :	$T \rightarrow i \bullet * E, +/\$$	$T \rightarrow i\bullet, +/\$$	
q_5 :	$S \rightarrow E + \bullet T, \$$	$T \rightarrow \bullet i * E, \$$	$T \rightarrow \bullet i, \$$

States

q_6 :	$T \rightarrow i \ * \bullet E, +/\$ \quad E \rightarrow \bullet T, +/\$ \quad T \rightarrow \bullet i \ * \ E, +/\$$ $T \rightarrow \bullet i, +/\$$
q_7 :	$S \rightarrow E \ + \ T \bullet, \$$
q_8 :	$T \rightarrow i \bullet * E, \$ \qquad T \rightarrow i \bullet, \$$
q_9 :	$T \rightarrow i \ * \ E \bullet, +/\$$
q_{10} :	$E \rightarrow T \bullet, +/\$$
q_{11} :	$T \rightarrow i \ * \bullet E, \$ \qquad E \rightarrow \bullet T, \$ \qquad T \rightarrow \bullet i \ * \ E, \$$ $T \rightarrow \bullet i, \$$

States

q_{12} :	$T \rightarrow i \ * \ E\bullet, \$$
q_{13} :	$E \rightarrow T\bullet, \$$

Note

Number of states of the $LR(1)$ automaton are more than that of $LR(0)$ automaton.
Several states have the same core $LR(0)$ items, but different *look-ahead* symbols - (q_4, q_8), (q_6, q_{11}), (q_9, q_{12}).

$LR(1)$ Parsing Table: *Action*

- If $(A \rightarrow \alpha \bullet a\beta, b) \in q_i$ $(a \in \Sigma)$ and $Goto(q_i, a) = q_j$, then $\text{Action}(i, a) = s_j$.
- If $(A \rightarrow \alpha\bullet, a) \in q_i$ $(A \neq S')$, then $\text{Action}(i, a) = r_k$, where k is the rule number of $A \rightarrow \alpha$.
- If $(S' \rightarrow S\bullet, \$) \in q_i$, then $\text{Action}(i, \$) =$ *accept*.

$LR(1)$ Parsing Table: *Goto*

If $A \rightarrow \alpha \bullet B\beta \in q_i$ $(B \in N)$ and
$\text{Goto}(q_i, B) = q_j$, then in the table
$\text{Goto}(i, B) = j$.
All other entries of the table are errors.

Note

If the process does not lead to a table with multiple entries, the grammar is $LR(1)$.

$LR(1)$ Parsing Table

S	*Action*				*Goto*		
	$+$	$*$	i	\$	S	E	T
0			s_4		1	2	3
1				A			
2	s_5						
3	r_5			r_2			
4	r_4	s_6		r_4			
5			s_8				7
6			s_4		9		10

$LR(1)$ Parsing Table

S	*Action*				*Goto*		
	$+$	$*$	i	\$	S	E	T
7				r_1			
8		s_{11}		r_4			
9	r_3			r_3			
10	r_5			r_5			
11			s_8			12	13
12				r_3			
13				r_5			

Non-$LR(1)$ Grammar

$$0: \quad S \rightarrow A$$

$$1: \quad A \rightarrow a\ A\ a$$

$$2: \quad A \rightarrow a\ Aa\ a\ b$$

$$3: \quad A \rightarrow a\ b$$

States of $LR(1)$ Automaton

q_0 :	$S \rightarrow \bullet A, \$$	$A \rightarrow \bullet aAa, \$$	$A \rightarrow \bullet aAaab, \$$
	$A \rightarrow \bullet ab, \$$		
q_1 :	$S \rightarrow A\bullet, \$$		
q_2 :	$A \rightarrow a \bullet Aa, \$$	$A \rightarrow a \bullet Aaab, \$$	$A \rightarrow a \bullet b, \$$
	$A \rightarrow \bullet aAa, a$	$A \rightarrow \bullet aAaab, a$	$A \rightarrow \bullet ab, a$
q_3 :	$A \rightarrow aA \bullet a, \$$	$A \rightarrow aA \bullet aab, \$$	
q_4 :	$A \rightarrow ab\bullet, \$$		
q_5 :	$A \rightarrow a \bullet Aa, a$	$A \rightarrow a \bullet Aaab, a$	$A \rightarrow a \bullet b, a$
	$A \rightarrow \bullet aAa, a$	$A \rightarrow \bullet aAaab, a$	$A \rightarrow \bullet ab, a$

States of $LR(1)$ Automaton

q_6 :	$A \rightarrow aAa\bullet, \$$	$A \rightarrow aAa \bullet ab, \$$
q_7 :	$A \rightarrow aA \bullet a, a$	$A \rightarrow aA \bullet aab, a$
q_8 :	$A \rightarrow ab\bullet, a$	
q_9	$A \rightarrow aAaa \bullet b, \$$	
q_{10} :	$A \rightarrow aAa\bullet, a$	$aAa \bullet ab, a$
q_{11}	$A \rightarrow aAaab\bullet, \$$	

Note

In state q_{10}, the *shift/reduce* conflict cannot be resolved and there will be multiple entries in $\text{Action}(10, a) = \{s_i, r_1\}$, where $\text{Goto}(q_{10}, a) = q_i$. This can be resolved with 2-look-aheads

States of $LR(2)$ Automaton

q_0 :	$S \rightarrow \bullet A, \$$	$A \rightarrow \bullet aAa, \$$	$A \rightarrow \bullet aAaab, \$$
	$A \rightarrow \bullet ab, \$$		
q_1 :	$S \rightarrow A\bullet, \$$		
q_2 :	$A \rightarrow a \bullet Aa, \$$	$A \rightarrow a \bullet Aaab, \$$	$A \rightarrow a \bullet b, \$$
	$A \rightarrow \bullet aAa, aa/a\$$	$A \rightarrow \bullet aAaab, aa/a\$$	$A \rightarrow \bullet ab, aa/a\$$
q_3 :	$A \rightarrow aA \bullet a, \$$	$A \rightarrow aA \bullet aab, \$$	
q_4 :	$A \rightarrow ab\bullet, \$$		
q_5 :	$A \rightarrow a \bullet Aa, aa/a\$$	$A \rightarrow a \bullet Aaab, aa/a\$$	$A \rightarrow a \bullet b, aa/a\$$
	$A \rightarrow \bullet aAa, aa$	$A \rightarrow \bullet aAaab, aa$	$A \rightarrow \bullet ab, aa$

States of $LR(2)$ Automaton

q_6 :	$A \rightarrow aAa\bullet, \$$	$A \rightarrow aAa \bullet ab, \$$
q_7 :	$A \rightarrow aA \bullet a, aa/a\$$	$A \rightarrow aA \bullet aab, aa/a\$$
q_8 :	$A \rightarrow ab\bullet, aa/a\$$	
q_9	$A \rightarrow aAaa \bullet b, \$$	
q_{10} :	$A \rightarrow aAa\bullet, aa/a\$$	$aAa \bullet ab, aa/a\$$
q_{11}	$A \rightarrow aAaab\bullet, \$$	

Note

In state q_{10}, the action is r_1 if the next two symbols are either 'aa' or 'a$'. The action is shift if they are 'ab'. But we shall not use $LR(2)$ parsing.

LALR Parser

We observed in connection to the $LR(1)$ states of the grammar G_{rr}, that there are pairs of states with the same core $LR(0)$ items, but with different *look-ahead* symbols - (q_4, q_8), (q_6, q_{11}), (q_9, q_{12}).
If we can *merge* states with same core of $LR(0)$ items, we get a parsing table of lesser number of states or rows. For some $LR(1)$ grammar this merging will not lead to *multiple entries* in the parsing table and the corresponding grammar is known as LALR (*lookahead LR*) grammar.

Note

Merging of two $LR(1)$ states with the same the core cannot give rise to shift/reduce conflict, if it was not present in the original automaton. If there is a pair of items of the form $\{A \rightarrow \alpha \bullet a\beta, \cdots B \rightarrow \gamma\bullet, a\}$ in the merged state, it originally was there is one of the $LR(1)$ states. So the $LR(1)$ automaton was not conflict free.

Note

Two states of an LALR parser cannot have the same set of $LR(0)$ items. So, for a given grammar, the number of states of an SLR parser is same as the number of states of an LALR parser.
An LALR parser uses a better heuristic about symbols that can follow a viable prefix, than the global FOLLOW() sets of non-terminals.

LALR States

The states of the $LR(1)$ automaton are as follows:

q_0 :	$S' \rightarrow \bullet S, \$$	$S \rightarrow \bullet E + T, \$$	$S \rightarrow \bullet T, \$$
	$E \rightarrow \bullet T, +$	$T \rightarrow \bullet i * E, +/\$$	$T \rightarrow \bullet i, +/\$$
q_1 :	$S' \rightarrow S\bullet, \$$		
q_2 :	$S \rightarrow E \bullet + T, \$$		
q_3 :	$S \rightarrow T\bullet, \$$	$E \rightarrow T\bullet, +$	
$q_{4,8}$:	$T \rightarrow i \bullet * E, +/\$$	$T \rightarrow i\bullet, +/\$$	
q_5 :	$S \rightarrow E + \bullet T, \$$	$T \rightarrow \bullet i * E, \$$	$T \rightarrow \bullet i, \$$

States

$q_{6\cdot 11}$:	$T \rightarrow i \ * \bullet E, +/\$ \quad E \rightarrow \bullet T, +/\$ \quad T \rightarrow \bullet i \ * \ E, +/\$$ $T \rightarrow \bullet i, +/\$$
q_7 :	$S \rightarrow E \ + \ T\bullet, \$$
$q_{9\cdot 12}$:	$T \rightarrow i \ * \ E\bullet, +/\$$
q_{10} :	$E \rightarrow T\bullet, +/\$$
q_{13} :	$E \rightarrow T\bullet, \$$

LALR Parsing Table

S	*Action*				*Goto*		
	$+$	$*$	i	\$	S	E	T
0			$s_{4\cdot 8}$		1	2	3
1				A			
2	s_5						
3	r_5			r_2			
$4 \cdot 8$	r_4	$s_{6\cdot 11}$		r_4			
5			$s_{4\cdot 8}$				7
$6 \cdot 11$			$s_{4\cdot 8}$		$9 \cdot 12$		10

LALR Parsing Table

S	*Action*				*Goto*		
	$+$	$*$	i	\$	S	E	T
7				r_1			
$9 \cdot 12$	r_3			r_3			
10	r_5			r_5			
13				r_5			

$LR(1)$ but not LALR

Consider the grammar

$$
\begin{aligned}
0: &\quad S \rightarrow A \\
1: &\quad A \rightarrow a\ B\ a \\
2: &\quad A \rightarrow b\ B\ b \\
3: &\quad A \rightarrow a\ D\ b \\
4: &\quad A \rightarrow b\ D\ a \\
5: &\quad B \rightarrow c \\
6: &\quad D \rightarrow c
\end{aligned}
$$

States of $LR(1)$ Automaton

q_0 :	$S \rightarrow \bullet A, \$$	$A \rightarrow \bullet aBa, \$$	$A \rightarrow \bullet bBb, \$$
	$A \rightarrow \bullet aDb, \$$	$A \rightarrow \bullet bDa, \$$	
q_1 :	$A \rightarrow a \bullet Ba, \$$	$A \rightarrow a \bullet Db, \$$	$B \rightarrow \bullet c, a$
	$D \rightarrow \bullet c, b$		
q_2 :	$A \rightarrow b \bullet Bb, \$$	$A \rightarrow b \bullet Da, \$$	$B \rightarrow \bullet c, b$
	$D \rightarrow \bullet c, a$		
q_3 :	$A \rightarrow aB \bullet a, \$$		
q_4 :	$A \rightarrow aD \bullet b, \$$		

States of $LR(1)$ Automaton

q_5 :	$B \rightarrow c\bullet, a \qquad D \rightarrow c\bullet, b$
q_6 :	$A \rightarrow bB \bullet b, \$$
q_7 :	$A \rightarrow bD \bullet a, \$$
q_8 :	$B \rightarrow c\bullet, b \qquad D \rightarrow c\bullet, a$

The states q_5 and q_8 have the same $LR(0)$ core, but they cannot be merged to form a LALR state as that will lead to *reduce/reduce* conflicts. So the grammar is $LR(1)$ but not LALR.

Ambiguous Grammar & LR Parsing

An *ambiguous* grammar cannot be LR. But for some ambiguous grammars[a] it is possible to use LR parsing techniques efficiently with the help of some *extra grammatical information* such as *operator associativity, precedence* etc.

[a] As an example for operator-precedence grammars: CFG with no ε production and no production rule with two non-terminals coming side by side.

Example

Consider the expression grammar G_a

$$
\begin{aligned}
0: \quad & S \rightarrow E \\
1: \quad & E \rightarrow E - E \\
2: \quad & E \rightarrow E * E \\
3: \quad & E \rightarrow (E) \\
4: \quad & E \rightarrow -E \\
5: \quad & E \rightarrow i
\end{aligned}
$$

Note that the terminal '-' is used both as binary as well as unary operator.

States of $LR(0)$ Automaton

q_0 :	$S \rightarrow \bullet E$	$E \rightarrow \bullet E - E$	$E \rightarrow \bullet E * E$
	$E \rightarrow \bullet (E)$	$E \rightarrow \bullet - E$	$E \rightarrow \bullet i$
q_1 :	$S \rightarrow E \bullet$	$E \rightarrow E \bullet - E$	$E \rightarrow E \bullet * E$
q_2 :	$E \rightarrow (\bullet E)$	$E \rightarrow \bullet E - E$	$E \rightarrow \bullet E * E$
	$E \rightarrow \bullet (E)$	$E \rightarrow \bullet - E$	$E \rightarrow \bullet i$
q_3 :	$E \rightarrow - \bullet E$	$E \rightarrow \bullet E - E$	$E \rightarrow \bullet E * E$
	$E \rightarrow \bullet (E)$	$E \rightarrow \bullet - E$	$E \rightarrow \bullet i$
q_4 :	$E \rightarrow i \bullet$		

States of $LR(0)$ Automaton

q_5 :	$E \rightarrow E - \bullet E$	$E \rightarrow \bullet E - E$	$E \rightarrow \bullet E * E$
	$E \rightarrow \bullet(E)$	$E \rightarrow \bullet - E$	$E \rightarrow \bullet i$
q_6 :	$E \rightarrow E * \bullet E$	$E \rightarrow \bullet E - E$	$E \rightarrow \bullet E * E$
	$E \rightarrow \bullet(E)$	$E \rightarrow \bullet - E$	$E \rightarrow \bullet i$
q_7 :	$E \rightarrow (E\bullet)$	$E \rightarrow E \bullet - E$	$E \rightarrow E \bullet * E$
q_8 :	$E \rightarrow -E\bullet$	$E \rightarrow E \bullet - E$	$E \rightarrow E \bullet * E$
q_9 :	$E \rightarrow E - E\bullet$	$E \rightarrow E \bullet - E$	$E \rightarrow E \bullet * E$
q_{10} :	$E \rightarrow E * E\bullet$	$E \rightarrow E \bullet - E$	$E \rightarrow E \bullet * E$

There are a few more states.

Note

The states q_8, q_9 and q_{10} have *complete* and *incomplete* items. FOLLOW$(E) = \{\$, -, *,)\}$ cannot resolve the conflict. In fact no amount of *look-ahead* can help - the $LR(1)$ initial state is

q_0 :	$S \rightarrow \bullet E, \$$	$E \rightarrow \bullet E - E, \$/-/*$	$E \rightarrow \bullet E * E, \$/-/*$
	$E \rightarrow \bullet(E), \$/-/*$	$E \rightarrow \bullet - E, \$/-/*$	$E \rightarrow \bullet i, \$/-/*$

$q_8 : E \rightarrow -E\bullet,\ E \rightarrow E \bullet -E,\ E \rightarrow E \bullet *E$

The *higher precedence* of unary '$-$' over the binary '$-$' and binary '$*$' will help to resolve the conflict. The parser *reduces* the handle i.e. $\text{Action}(8, -) = \text{Action}(8, *) = \text{Action}(8,)) = \text{Action}(8, \$) = r_4$.

$$q_9 : E \rightarrow E - E\bullet,\ E \rightarrow E \bullet - E,\ E \rightarrow E \bullet * E$$

In this case if the *look-ahead* symbol is a '$-$' (it must be binary), the parser *reduces* due to the left associativity of binary '$-$'. But if the *look-ahead* symbol is a '$*$', the parser *shifts* i.e. $\text{Action}(9, -) = \text{Action}(9,)) = \text{Action}(9, \$) = r_4$ but $\text{Action}(9, *) = s_6$.

$q_{10} : E \rightarrow E * E\bullet,\ E \rightarrow E \bullet - E,\ E \rightarrow E \bullet * E$

Actions are always *reduce*.