CLR(1) Parser

Also known as LR(1) Parser

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

SLR : Review

	FOLLOW
E	\$, +,)
T	\$, +, *,)
F	\$, +, *,)

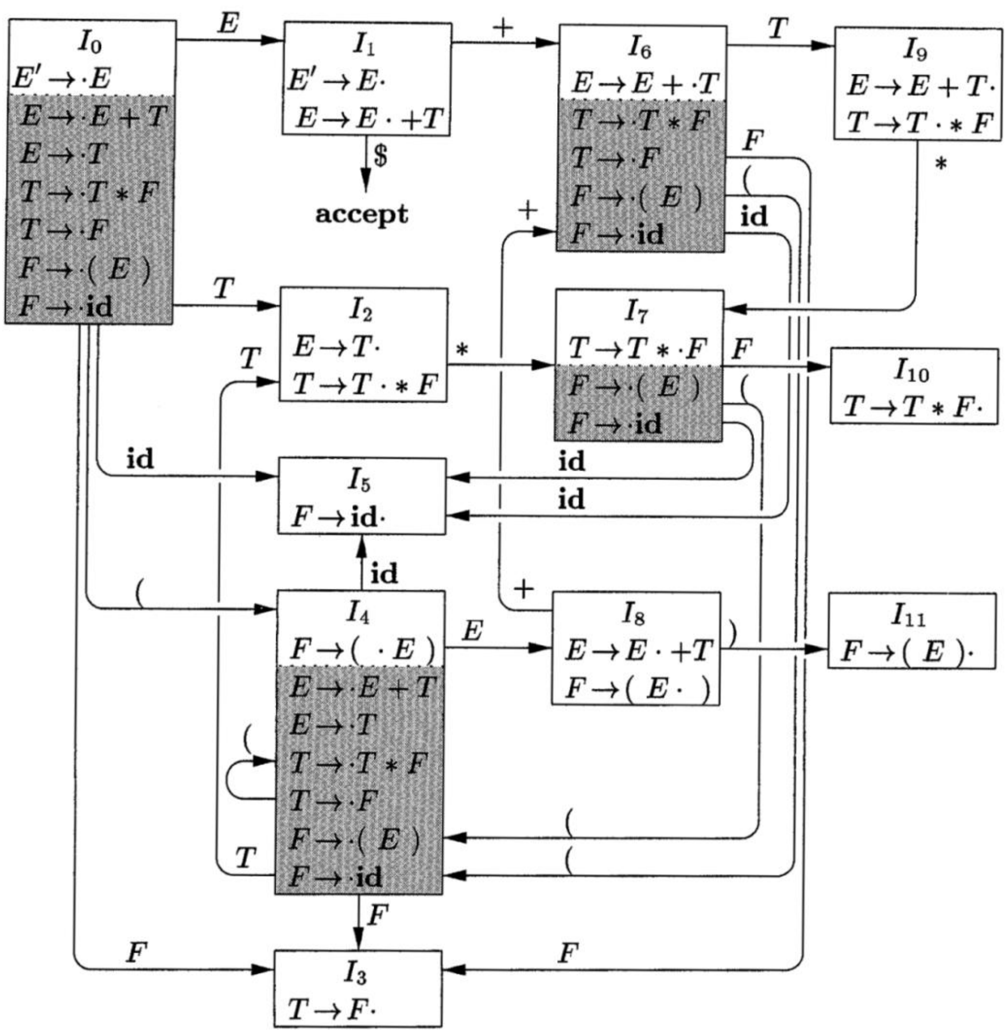


Figure 4.31: LR(0) automaton for the expression grammar (4.1)

Note: $*$ does not FOLLOW E.
Hence, while in state 2, on input $*$,
we can not reduce T to E.
So, we must shift $*$ on to stack.

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			

Figure 4.37: Parsing table for expression grammar

LR(0) Vs SLR(1)

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

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

LR(0) Parsing Table

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(1) Parsing Table

LR(0) Vs SLR(1)

- **You refused to reduce in many cases.**
- **Conflicts vanished (thank god).**
- **Many blank entries! Errors are caught early.**

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

LR(0) Parsing Table

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(1) Parsing Table

SLR or not?

$A \to \alpha \cdot$ is called a final item.

If we reached a state having a final item $A \to \alpha \cdot$

Then we are applying the reduction using $A \to \alpha$

This is not a mistake only if the current input terminal is in FOLLOW(A), and we do not have yet another item which causes either shift/reduce conflict or reduce/reduce conflict.

This characterizes the SLR grammars.

What LR(0) parser does:
Lookahead is not used as described above. As soon as a final item is reached it applies reduction (doesnot care about what is the lookahead).

Example: non-SLR grammar

$$
\begin{aligned}
S &\rightarrow L = R \mid R \\
L &\rightarrow *R \mid \mathbf{id} \\
R &\rightarrow L
\end{aligned}
\qquad (4.49)
$$

FOLLOW (R) contains $=$ (since, $S \Rightarrow L = R \Rightarrow *R = R$)

Assume that we are in state 2 and the next input is $=$

There is a shift/reduce conflict.

State	Action				Goto		
	=	*	id	$	S	L	R
0		s4	s5		1	2	3
1				acc			
2	s6/r5			r5			
3				r2			
4		s4	s5			8	7
5	r4			r4			
6		s4	s5			8	9
7	r3			r3			
8	r5			r5			
9				r1			

I_0: $S' \rightarrow \cdot S$
$S \rightarrow \cdot L = R$
$S \rightarrow \cdot R$
$L \rightarrow \cdot * R$
$L \rightarrow \cdot \mathbf{id}$
$R \rightarrow \cdot L$

I_1: $S' \rightarrow S \cdot$

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

I_3: $S \rightarrow R \cdot$

I_4: $L \rightarrow * \cdot R$
$R \rightarrow \cdot L$
$L \rightarrow \cdot * R$
$L \rightarrow \cdot \mathbf{id}$

I_5: $L \rightarrow \mathbf{id} \cdot$

I_6: $S \rightarrow L = \cdot R$
$R \rightarrow \cdot L$
$L \rightarrow \cdot * R$
$L \rightarrow \cdot \mathbf{id}$

I_7: $L \rightarrow *R \cdot$

I_8: $R \rightarrow L \cdot$

I_9: $S \rightarrow L = R \cdot$

Figure 4.39: Canonical LR(0) collection for grammar (4.49)

4.6.5 Viable Prefixes

Why can LR(0) automata be used to make shift-reduce decisions?

- The stack contents must be a prefix of a right-sentential form.

If the stack holds α and the rest of the input is x, then

$$S \underset{rm}{\overset{*}{\Rightarrow}} \alpha x.$$

4.6.5 Viable Prefixes

Why can LR(0) automata be used to make shift-reduce decisions?

- The stack contents must be a prefix of a right-sentential form.

If the stack holds α and the rest of the input is x, then

$S \underset{rm}{\overset{*}{\Rightarrow}} \alpha x.$

- Not all prefixes of a right-sentential form can appear on the stack.

$$E \underset{rm}{\overset{*}{\Rightarrow}} F * \mathbf{id} \underset{rm}{\Rightarrow} (E) * \mathbf{id}$$

Stack can be $ ... (E)
But it can not be $ ... (E)*

Viable Prefixes

The prefixes of right sentential forms that can appear on the stack of a shift-reduce parser are called *viable prefixes*.

Viable Prefixes

The prefixes of right sentential forms that can appear on the stack of a shift-reduce parser are called *viable prefixes.*

SLR parsing is based on the fact that LR(0) automata recognize viable prefixes.

If there is a derivation $S' \underset{rm}{\overset{*}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha\beta_1\beta_2 w$, $\alpha\beta_1$ is a viable prefix.

For the viable prefix $\alpha\beta_1$, we say $A \to \beta_1 \cdot \beta_2$ is a *valid* item.

In general, an item will be valid for many viable prefixes.

Similarly, many items can be valid for a viable prefix

Viable Prefixes

The prefixes of right sentential forms that can appear on the stack of a shift-reduce parser are called *viable prefixes.*

SLR parsing is based on the fact that LR(0) automata recognize viable prefixes.

If there is a derivation $S' \underset{rm}{\overset{*}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha\beta_1\beta_2 w$, $\alpha\beta_1$ is a viable prefix.

For the viable prefix $\alpha\beta_1$, we say $A \to \beta_1 \cdot \beta_2$ is a *valid* item.

In general, an item will be valid for many viable prefixes.

Similarly, many items can be valid for a viable prefix

- At any particular stage of LR parsing, there is utmost one handle.
- There can never be a stage where a handle is hidden (buried) in the stack.
- For a viable prefix, it should be possible for us to append only terminals to get a right-sentential form.

- Viable prefix along with a valid item tells us whether to shift or reduce.

If there is a derivation $S' \overset{*}{\underset{rm}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha\beta_1\beta_2 w$, $\alpha\beta_1$ is a viable prefix.

For the viable prefix $\alpha\beta_1$, we say $A \rightarrow \beta_1 \cdot \beta_2$ is a *valid* item.

- Viable prefix along with a valid item tells us whether to shift or reduce.

If there is a derivation $S' \underset{rm}{\overset{*}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha\beta_1\beta_2 w$, $\alpha\beta_1$ is a viable prefix.

For the viable prefix $\alpha\beta_1$, we say $A \rightarrow \beta_1 \cdot \beta_2$ is a *valid* item.

If $\beta_2 = \epsilon$, then it looks as if $A \rightarrow \beta_1$ is the handle, and we should reduce by this production. Otherwise, we need to shift further.

Inability to uniquely identify the handle is the problem.
Look-ahead in the input buffer can help us.

If there is a derivation $S' \overset{*}{\underset{rm}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha\beta_1\beta_2 w$, $\alpha\beta_1$ is a viable prefix.

For the viable prefix $\alpha\beta_1$, we say $A \to \beta_1 \cdot \beta_2$ is a *valid* item.

A notable fact : It is a central theorem of LR-parsing theory that the set of valid items for a viable prefix γ is exactly the set of items reached from the initial state along the path labeled γ in the LR(0) automaton for the grammar.

$$
\begin{array}{lcl}
E' & \rightarrow & E \\
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

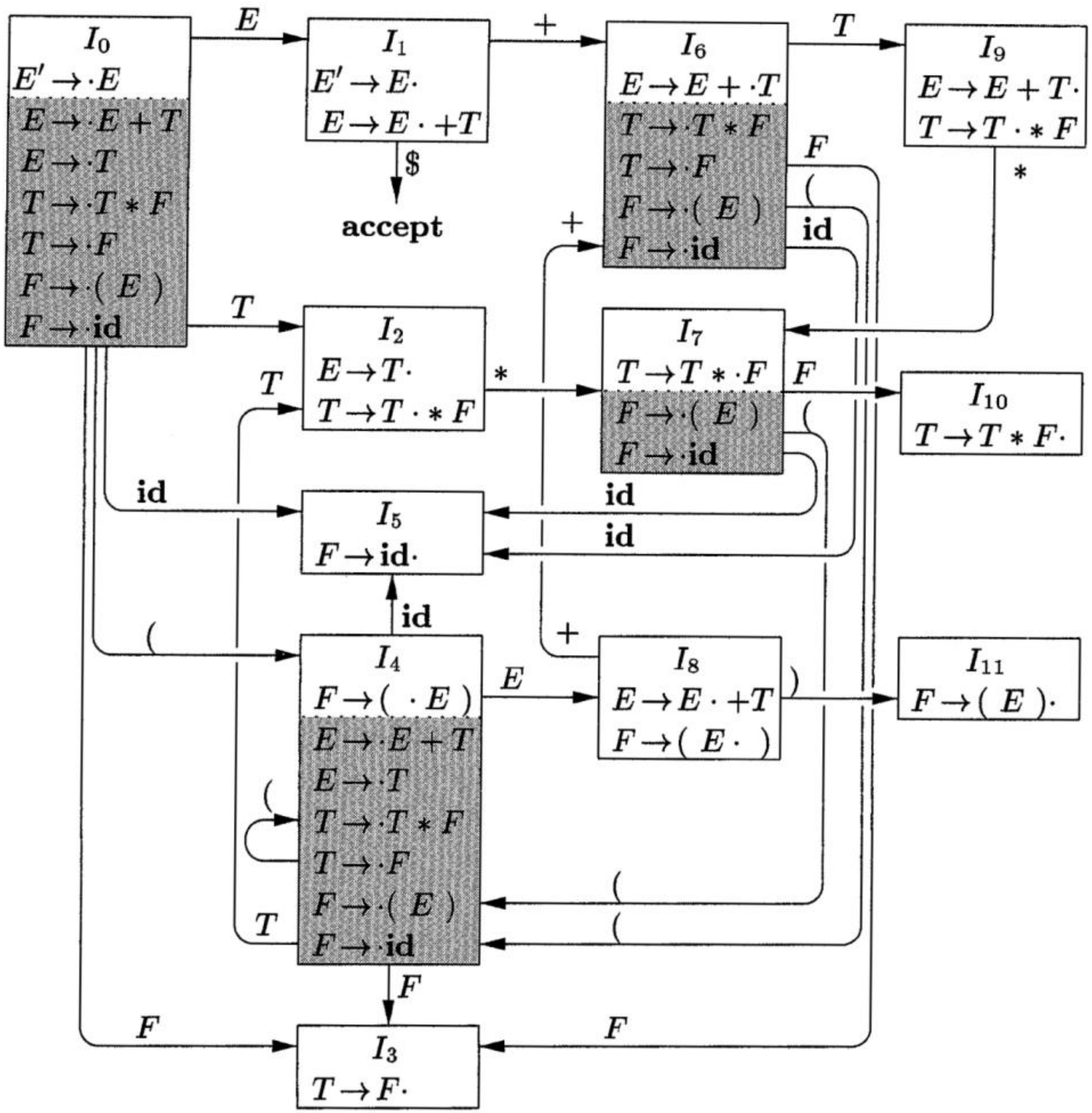


Figure 4.31: LR(0) automaton for the expression grammar (4.1)

$$
\begin{array}{lcl}
E' & \rightarrow & E \\
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & (E) \mid \textbf{id}
\end{array}
$$

$E + T*$ is a viable prefix

The automaton of Fig. 4.31 will be in state 7 after having read $E + T*$.

State 7 contains the items

$$
\boxed{\begin{array}{l}
T \rightarrow T * \cdot F \\
F \rightarrow \cdot(E) \\
F \rightarrow \cdot\textbf{id}
\end{array}}
$$

which are precisely the items valid for $E+T*$. It can be shown that there are no other valid items for $E + T*$, although we shall not prove that fact here.

To see why, consider the following three rightmost derivations

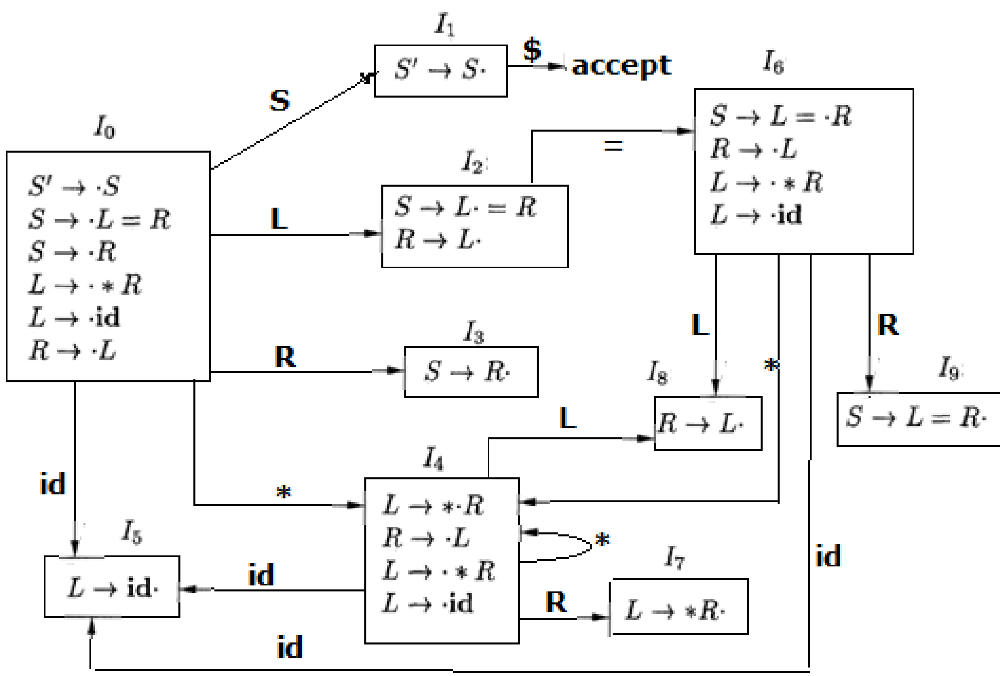
$$
\begin{array}{ll}
E' & \underset{rm}{\Rightarrow} E \\
 & \underset{rm}{\Rightarrow} E + T \\
 & \underset{rm}{\Rightarrow} E + T * F
\end{array}
\qquad
\begin{array}{ll}
E' & \underset{rm}{\Rightarrow} E \\
 & \underset{rm}{\Rightarrow} E + T \\
 & \underset{rm}{\Rightarrow} E + T * F \\
 & \underset{rm}{\Rightarrow} E + T * (E)
\end{array}
\qquad
\begin{array}{ll}
E' & \underset{rm}{\Rightarrow} E \\
 & \underset{rm}{\Rightarrow} E + T \\
 & \underset{rm}{\Rightarrow} E + T * F \\
 & \underset{rm}{\Rightarrow} E + T * \textbf{id}
\end{array}
$$

In SLR method, when we are at state i,
if the set of items I_i contains item $[A \to \alpha\cdot]$, a is the next input,
a is in **FOLLOW**(A), then reduction on $A \to \alpha$ is perhaps the correct action.

Definitely, if a is not in **FOLLOW**(A) then this reduction should not be applied.

- Let us consider

$$\begin{array}{lcl} S & \to & L = R \mid R \\ L & \to & *R \mid \mathbf{id} \\ R & \to & L \end{array} \qquad (4.49)$$



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

I_1: $S' \to S\cdot$

I_2: $S \to L\cdot = R$
$\quad R \to L\cdot$

I_3: $S \to R\cdot$

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

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

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

I_7: $L \to *R\cdot$

I_8: $R \to L\cdot$

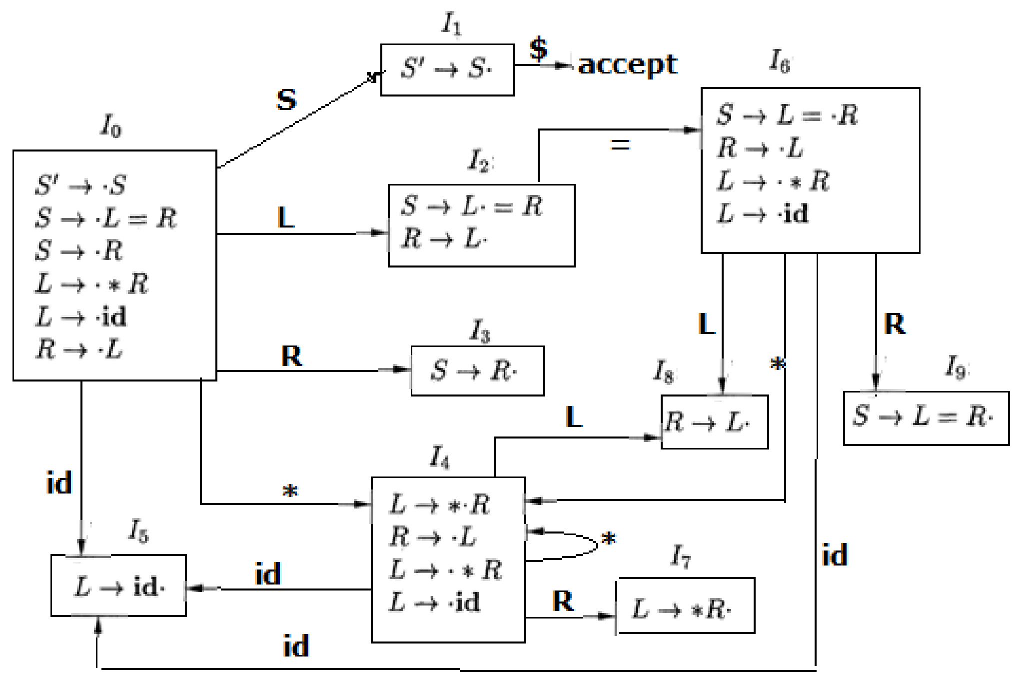
I_9: $S \to L = R\cdot$

Figure 4.39: Canonical LR(0) collection for grammar (4.49)

LR(0) automaton

$$
\begin{array}{llcl}
(1) & S & \rightarrow & L = R \\
(2) & S & \rightarrow & R \\
(3) & L & \rightarrow & *R \\
(4) & L & \rightarrow & \mathbf{id} \\
(5) & R & \rightarrow & L
\end{array}
\qquad (4.49)
$$

Variable	FOLLOW
S	$
L	$, =
R	$, =



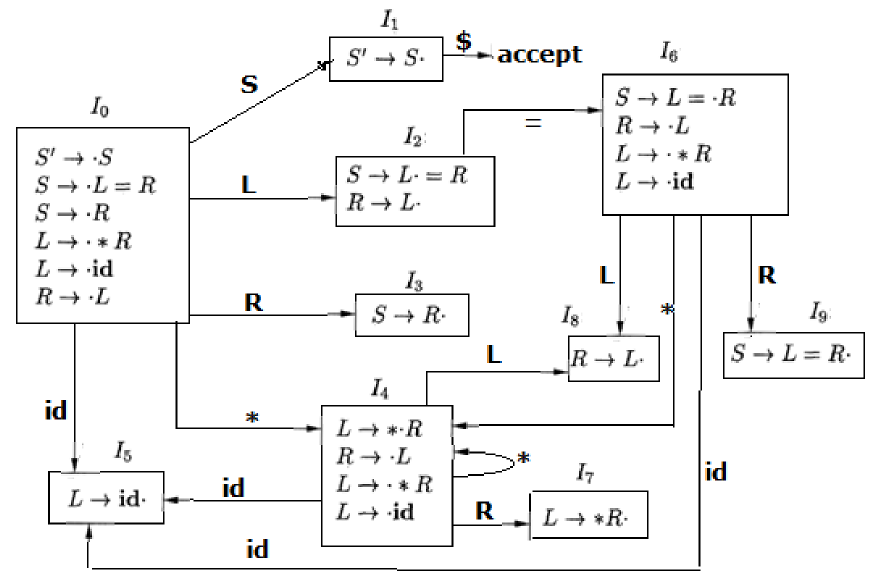
State	Action				Goto		
	=	*	id	$	S	L	R
0		s4	s5		1	2	3
1				acc			
2	s6/r5			r5			
3				r2			
4		s4	s5			8	7
5	r4			r4			
6		s4	s5			8	9
7	r3			r3			
8	r5			r5			
9				r1			

SLR(1) Parse Table.

Note the shift reduce conflict.

So the grammar is not SLR(1).

$$
\begin{array}{lcl}
S & \rightarrow & L = R \mid R \\
L & \rightarrow & *R \mid \mathbf{id} \\
R & \rightarrow & L
\end{array}
\qquad (4.49)
$$



This is not SLR(1) grammar.

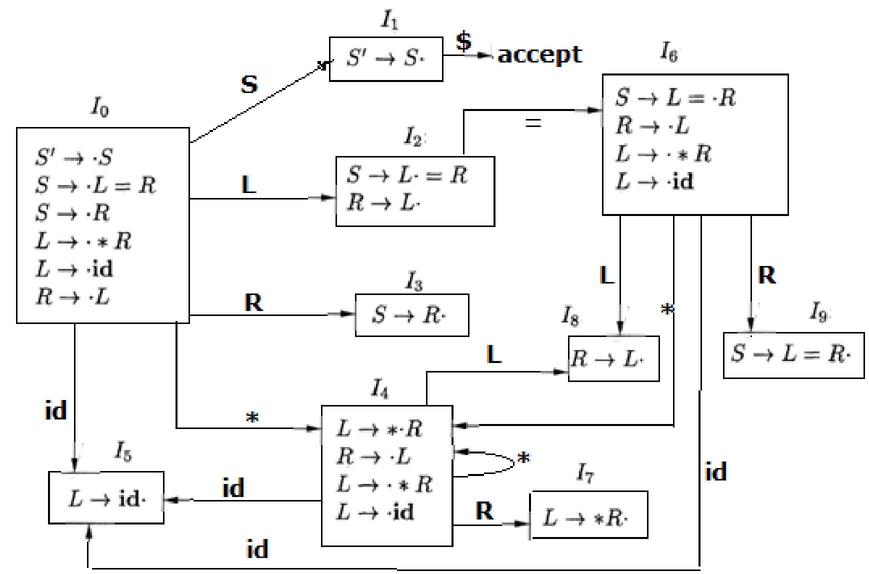
There is a shift/reduce conflict which SLR method cannot resolve.

FOLLOW (R) contains $=$ (since, $S \Rightarrow L = R \Rightarrow *R = R$)

Assume that we are in state 2 and the next input is =

There is a shift/reduce conflict.

$$
\begin{array}{lcl}
S & \rightarrow & L = R \mid R \\
L & \rightarrow & *R \mid \mathbf{id} \\
R & \rightarrow & L
\end{array}
\qquad (4.49)
$$



This is not SLR(1) grammar.

There is a shift/reduce conflict which SLR method cannot resolve.

FOLLOW (R) contains = (since, $S \Rightarrow L = R \Rightarrow *R = R$)

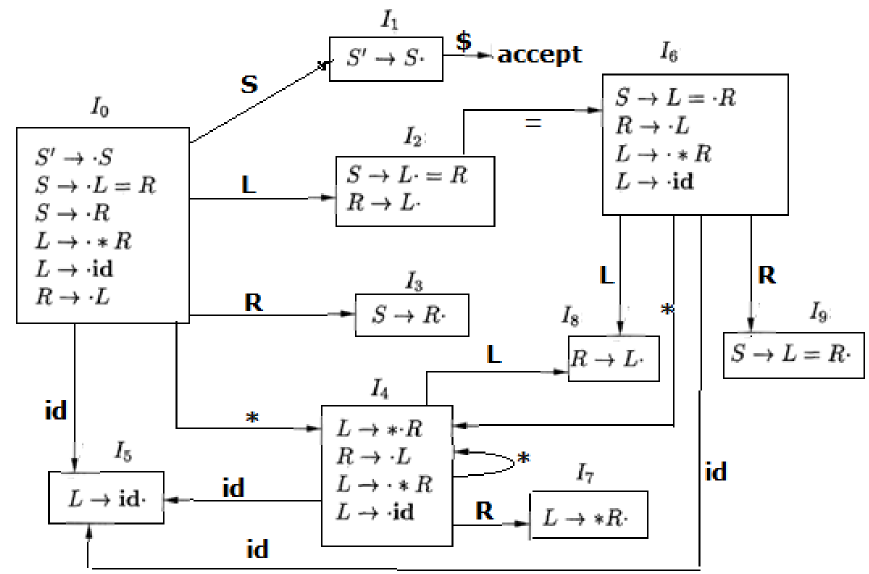
Assume that we are in state 2 and the next input is =

There is a shift/reduce conflict.

But,

Just because = is in FOLLOW(R) we should not apply reduction using $R \rightarrow L$.

$$
\begin{array}{lcl}
S & \rightarrow & L = R \mid R \\
L & \rightarrow & *R \mid \textbf{id} \\
R & \rightarrow & L
\end{array}
\qquad (4.49)
$$



This is not SLR(1) grammar.

There is a shift/reduce conflict which SLR method cannot resolve.

FOLLOW (R) contains = (since, $S \Rightarrow L = R \Rightarrow *R = R$)

Assume that we are in state 2 and the next input is =

There is a shift/reduce conflict.

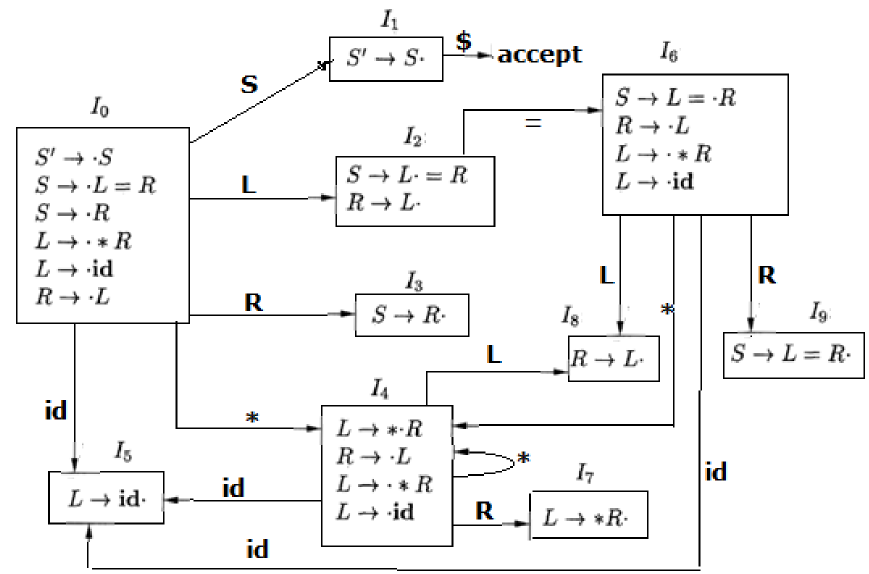
But,

Just because = is in FOLLOW(R) we should not apply reduction using $R \rightarrow L$.

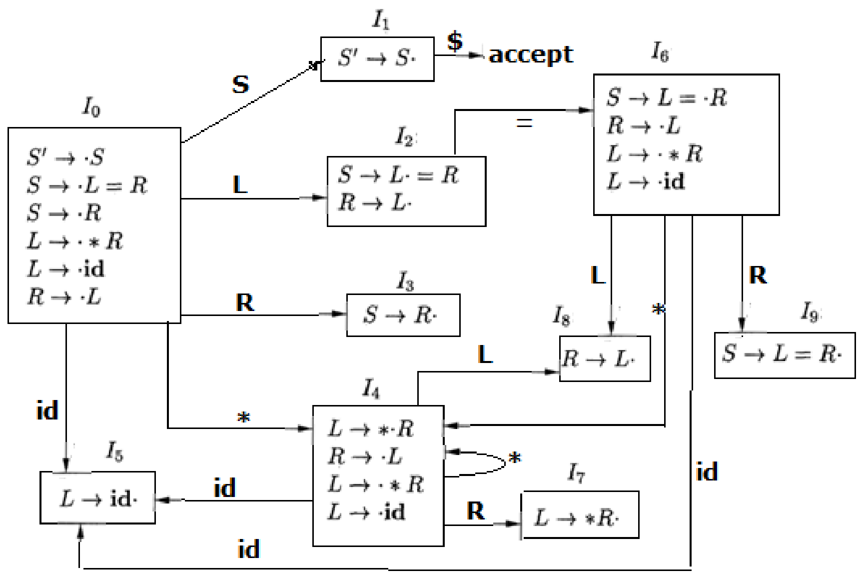
Answer: There is no right-sentential form that begins with R=…
Viable prefix when we are in state 2 is L only. So reduction is a mistake.
So, we should not apply the reduction , but we should shift = on to stack.

$$
\begin{array}{lcl}
S & \rightarrow & L = R \mid R \\
L & \rightarrow & *R \mid \mathbf{id} \\
R & \rightarrow & L
\end{array}
\qquad (4.49)
$$

Consider the string **id = id**



$$
\begin{array}{lcl}
S & \rightarrow & L = R \mid R \\
L & \rightarrow & *R \mid \mathbf{id} \\
R & \rightarrow & L
\end{array}
\qquad (4.49)
$$

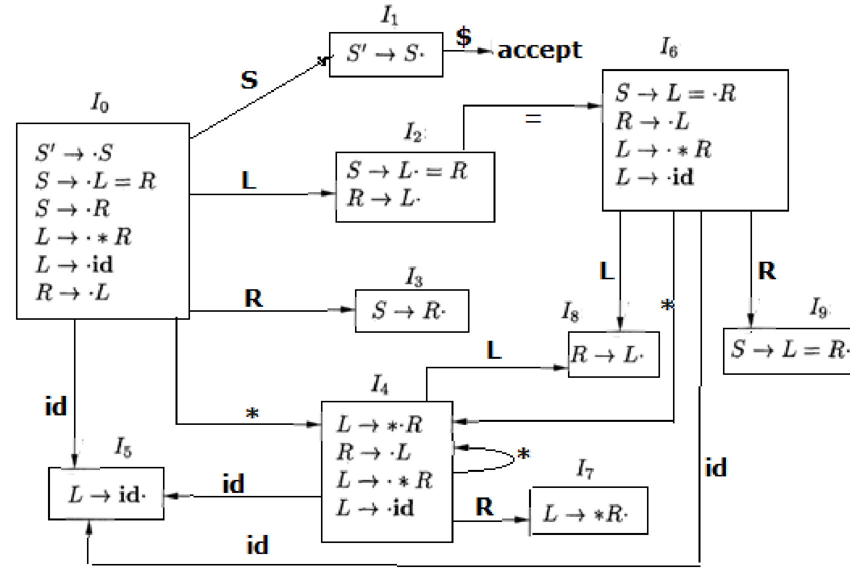


Consider the string **id = id**

$	id = id $	Shift
$ id	= id $	Reduce L --> id
$ L	= id $	Reduce R --> L
$ R	= id $	Error

Reduction is a wrong choice

$$
\begin{array}{lcl}
S & \rightarrow & L = R \mid R \\
L & \rightarrow & *R \mid \mathbf{id} \\
R & \rightarrow & L
\end{array}
\qquad (4.49)
$$



Consider the string **id = id**

$	id = id $	Shift
$ id	= id $	Reduce L --> id
$ L	= id $	Reduce R --> L
$ R	= id $	Error

Reduction is a wrong choice

$	id = id $	Shift
$ id	= id $	Reduce L --> id
$ L	= id $	Shift
$ L =	id $	Shift
$ L = id	$	Reduce L --> id
$ L = L	$	Reduce R --> L
$ L = R	$	Reduce S --> L=R
$ S	$	accept

Shift is a right choice

So,

- Just FOLLOW is not enough,

  the input should FOLLOW in a *right-sentential* form, also that should be valid with the state where we are in.
- LR(1) item adds some more information to LR(0) item.

CLR Parsing (by default: CLR(1))

- Uses canonical LR(1) items

A *LR(1)* item is in the form

$[A \to \alpha \cdot \beta, a]$, where $A \to \alpha\beta$ is a production and a is a terminal or the right endmarker \$.

The second component added is the lookahead of the item.

The 1 in LR(1) refers to the length of this lookahead.

If β is not ϵ, the lookahead has no effect in the item $[A \to \alpha \cdot \beta, a]$.

But, an item of the form $[A \to \alpha \cdot, a]$ calls for a reduction by $A \to \alpha$ only if the next input symbol is a.

Thus, we are compelled to reduce by $A \to \alpha$ only on those input symbols a for which $[A \to \alpha\cdot, a]$ is an LR(1) item in the state on top of the stack.

The set of such a's will always be a subset of FOLLOW(A), but it could be a proper subset, as in Example 4.51.

$$
\begin{array}{lcl}
S & \to & L = R \mid R \\
L & \to & *R \mid \textbf{id} \\
R & \to & L
\end{array}
\qquad (4.49)
$$

FOLLOW (R) contains = (since, $S \Rightarrow L = R \Rightarrow *R = R$)

= does not FOLLOW(R) when we are in state 2.
So we should not apply reduction.

Formally, we say LR(1) item $[A \to \alpha\cdot\beta, a]$ is *valid* for a viable prefix $\delta\alpha$ if there is a derivation $S \underset{rm}{\overset{*}{\Rightarrow}} \delta A w \underset{rm}{\Rightarrow} \delta\alpha\beta w$, where

either a is the first symbol of w, or w is ϵ and a is $.

In otherwords,

There is a right-sentential form ...$\alpha\beta a$ ... , and $A \to \alpha\ \beta$ is a production.

a follows $\alpha\beta$ in a right-sentential form.

Example 4.52 : Let us consider the grammar

$$
\begin{aligned}
S &\rightarrow B\ B \\
B &\rightarrow a\ B \ \mid \ b
\end{aligned}
$$

Example 4.52 : Let us consider the grammar

$$S \to B\ B$$
$$B \to a\ B \ \mid \ b$$

There is a rightmost derivation $S \underset{rm}{\overset{*}{\Rightarrow}} aaBab \underset{rm}{\Rightarrow} aaaBab$.

We see that item $[B \to a{\cdot}B, a]$ is valid for a viable prefix aaa

Example 4.52 : Let us consider the grammar

$$S \rightarrow B\ B$$
$$B \rightarrow a\ B \ \mid\ b$$

There is a rightmost derivation $S \underset{rm}{\overset{*}{\Rightarrow}} aaBab \underset{rm}{\Rightarrow} aaaBab$.

We see that item $[B \rightarrow a{\cdot}B, a]$ is valid for a viable prefix aaa

There is a rightmost derivation $S \underset{rm}{\overset{*}{\Rightarrow}} BaB \underset{rm}{\Rightarrow} BaaB$.

From this derivation we see that item $[B \rightarrow a{\cdot}B, \$]$ is valid for viable prefix Baa.

How to create LR(1) items, and LR(1) automaton

- CLOSURE, and
- GOTO used in LR(0) are extended.

- Begin from the basis item $S' \rightarrow \cdot S, \$$
- Inductively find CLOSURE, then apply GOTO
- Till no more item can be added.

CLOSURE

Given an item $[A \rightarrow \alpha \bullet B\beta, a]$, its closure contains the item and any other items that can generate legal substrings to follow α.
Thus, if the parser has viable prefix α on its stack, the input should reduce to $B\beta$ (or γ for some other item $[B \rightarrow \bullet\gamma, b]$ in the closure).

```
function closure(I)
repeat
  if [A → α•Bβ, a] ∈ I
    add [B → •γ, b] to I, where b ∈ FIRST(βa)
until no more items can be added to I
return I
```

CLOSURE

Given an item $[A \to \alpha \bullet B\beta, a]$, its closure contains the item and any other items that can generate legal substrings to follow α.
Thus, if the parser has viable prefix α on its stack, the input should reduce to $B\beta$ (or γ for some other item $[B \to \bullet\gamma, b]$ in the closure).

```
function closure(I)
repeat
  if [A → α•Bβ, a] ∈ I
    add [B → •γ, b] to I, where b ∈ FIRST(βa)
until no more items can be added to I
return I
```

The fact that $[A \to \alpha \cdot B\beta, a]$ is a valid LR(1) item, says that, a follows A in a right-sentential form.

Using the production $A \to \alpha\, B\beta$ somewhere, we are going to get some right-sential forms . Among these forms, replacing B by γ results in a still restricted set of right-sentential forms. In these forms $\text{FIRST}(\beta a)$ is going to follow B .

Many terminals may follow B, but we want those which are in $\text{FIRST}(\beta a)$.

CLOSURE as given in the Dragon Book

```
SetOfItems CLOSURE(I) {
    repeat
        for ( each item [A → α·Bβ, a] in I )
            for ( each production B → γ in G′ )
                for ( each terminal b in FIRST(βa) )
                    add [B → ·γ, b] to set I;
    until no more items are added to I;
    return I;
}
```

The fact that $[A \to \alpha{\cdot}B\beta, a]$ is a valid LR(1) item,
says that, a follows A in a right-sentential form.

Using the production $A \to \alpha\, B\beta$ somewhere, we are going to get some right-sential forms. Among these forms, replacing B by γ results in a still restricted set of right-sentential forms. In these forms FIRST(βa) is going to follow B.

Many terminals may follow B, but we want those which are in FIRST(βa).

GOTO

- Let I be a set of LR(1) items and X be a grammar symbol. Then, GOTO(I,X) is the closure of the set of all items

  $$[A \to \alpha X \bullet \beta, a] \textit{ such that } [A \to \alpha \bullet X\beta, a] \in I$$

  If I is the set of valid items for some viable prefix γ, then GOTO(I,X) is the set of valid items for the viable prefix γX.
  goto(I,X) represents state after recognizing X in state I.

```
function goto(I,X)
  let J be the set of items [A → αX•β,a]
    such that [A → α•Xβ,a] ∈ I
  return closure(J)
```

GOTO as given in the Dragon Book

SetOfItems GOTO(I, X) {
  initialize J to be the empty set;
  for (each item $[A \rightarrow \alpha{\cdot}X\beta, a]$ in I)
    add item $[A \rightarrow \alpha X{\cdot}\beta, a]$ to set J;
  return CLOSURE(J);
}

Begin from $S' \rightarrow \cdot S, \$$ then apply CLOSURE and GOTO repeatedly

Example 4.54 : Consider the following augmented grammar.

$$\begin{array}{lcl} S' & \rightarrow & S \\ S & \rightarrow & C\ C \\ C & \rightarrow & c\ C \ \mid \ d \end{array} \tag{4.55}$$

Begin from $S' \rightarrow \cdot S, \$$ then apply CLOSURE and GOTO repeatedly

Example 4.54 : Consider the following augmented grammar.

$$\begin{array}{rcl} S' & \rightarrow & S \\ S & \rightarrow & C\ C \\ C & \rightarrow & c\ C \mid d \end{array} \tag{4.55}$$

We begin by computing the closure of $\{[S' \rightarrow \cdot S, \$]\}$.

We want to add to the closure $[S \rightarrow \cdot CC,\ b]$

But what should be this b now?

Begin from $S' \rightarrow \cdot S, \$$ then apply CLOSURE and GOTO repeatedly

Example 4.54: Consider the following augmented grammar.

$$\begin{array}{lcl} S' & \rightarrow & S \\ S & \rightarrow & C\ C \\ C & \rightarrow & c\ C \mid d \end{array} \tag{4.55}$$

We begin by computing the closure of $\{[S' \rightarrow \cdot S, \$]\}$.

We want to add to the closure $[S \rightarrow \cdot CC,\ b]$

But what should be this b now?

b is FIRST($) which is $ itself.

Thus we add $[S \rightarrow \cdot CC, \$]$.

Begin from $S' \to \cdot S, \$$ then apply CLOSURE and GOTO repeatedly

Example 4.54: Consider the following augmented grammar.

$$\begin{array}{rcl} S' & \to & S \\ S & \to & C\ C \\ C & \to & c\ C \ \mid \ d \end{array} \tag{4.55}$$

We begin by computing the closure of $\{[S' \to \cdot S, \$]\}$.

We want to add to the closure $[S \to \cdot CC,\ b]$

But what should be this b now?

b is FIRST(\$) which is \$ itself.

Thus we add $[S \to \cdot CC, \$]$.

Then we add $[C \to \cdot cC,\ \text{FIRST}(C\$)\ ]$

$\text{FIRST}(C\$) = \text{FIRST}(C) = \{\ \text{c, d}\ \}$.

So, we add items $[C \to \cdot cC, c]$, $[C \to \cdot cC, d]$.

Similarly we add $[C \to \cdot d, c]$ and $[C \to \cdot d, d]$.

Example 4.54 : Consider the following augmented grammar.

$$\begin{array}{rcl} S' & \rightarrow & S \\ S & \rightarrow & C\ C \\ C & \rightarrow & c\ C \mid d \end{array} \tag{4.55}$$

$$\boxed{\begin{array}{ll} I_0 : & S \rightarrow \cdot S, \$ \\ & S \rightarrow \cdot CC,\ \$ \\ & C \rightarrow \cdot cC,\ c/d \\ & C \rightarrow \cdot d,\ c/d \end{array}}$$

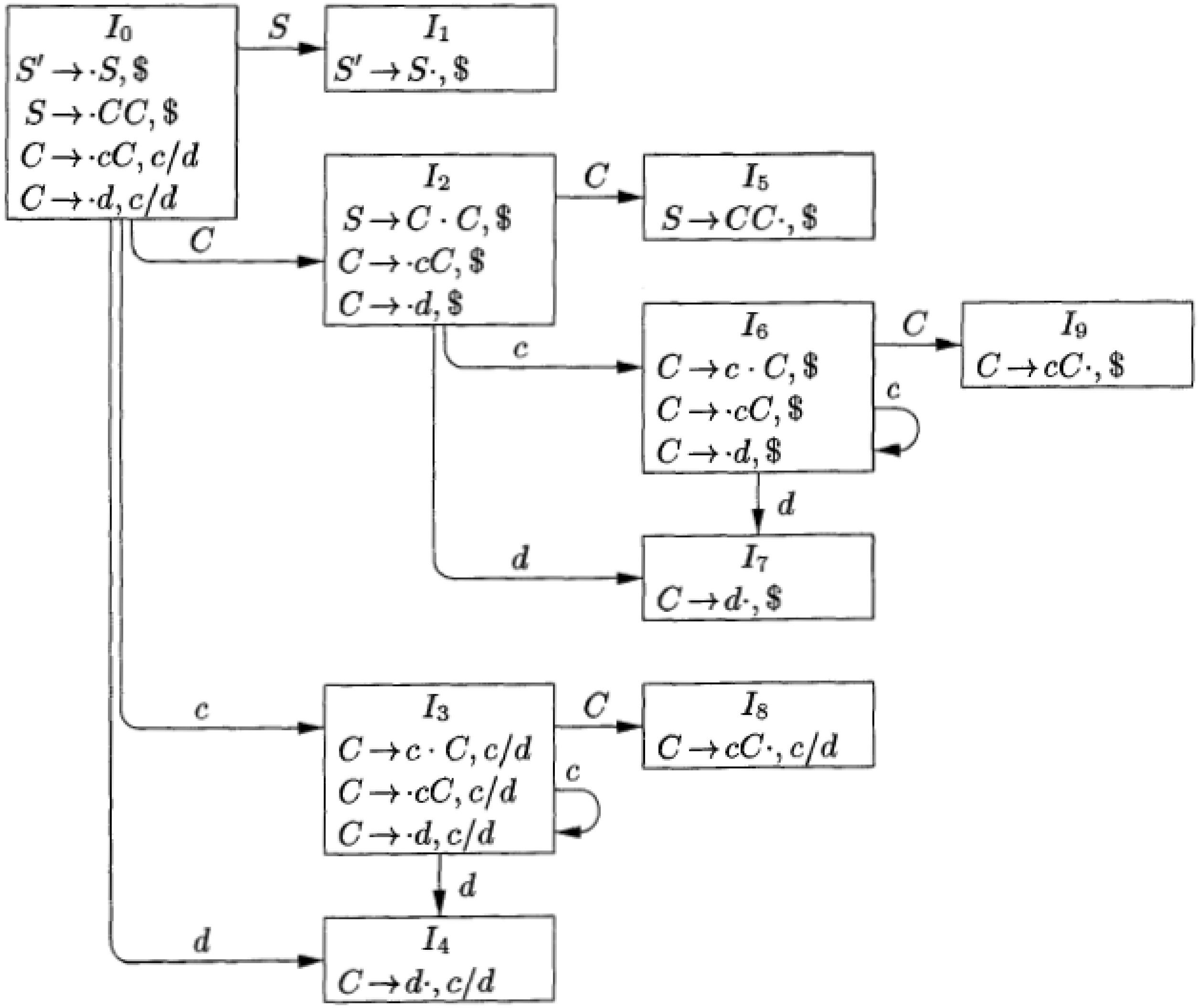


Figure 4.41: The GOTO graph for grammar (4.55)

Building Parse Table from LR(1) Automaton

The parsing action for state i is determined as follows.

(a) If $[A \rightarrow \alpha{\cdot}a\beta, b]$ is in I_i and $\text{GOTO}(I_i, a) = I_j$, then set $\text{ACTION}[i, a]$ to "shift j." Here a must be a terminal.

(b) If $[A \rightarrow \alpha\cdot, a]$ is in I_i, $A \neq S'$, then set $\text{ACTION}[i, a]$ to "reduce $A \rightarrow \alpha$."

(c) If $[S' \rightarrow S\cdot, \$]$ is in I_i, then set $\text{ACTION}[i, \$]$ to "accept."

If any conflicting actions result from the above rules, we say the grammar is not LR(1). The algorithm fails to produce a parser in this case.

(1) $S \rightarrow C\ C$

(2) $C \rightarrow c\ C$

(3) $C \rightarrow d$

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		

Figure 4.42: Canonical parsing table for grammar (4.55)

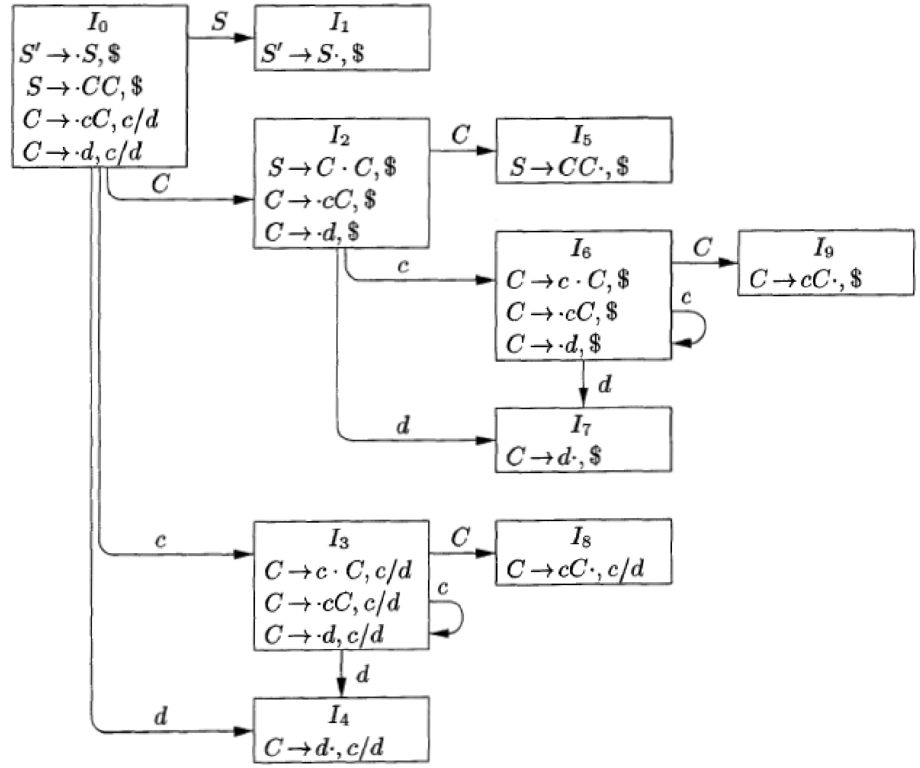


Figure 4.41: The GOTO graph for grammar (4.55)

(1)	S	$\rightarrow$	$C\ C$
(2)	C	$\rightarrow$	$c\ C$
(3)	C	$\rightarrow$	d

Stack of states	Stack of grammar Symbols	Input
0	$	cdd$
0 3	$c	dd$
0 3 4	$cd	d$
0 3 8	$cC	d$
0 2	$C	d$
0 2 7	$Cd	$
0 2 5	$CC	$
0 1	$S	$

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		

LR(0), LR(1) item sets are different.

$$
\begin{array}{lcl}
S & \rightarrow & L = R \mid R \\
L & \rightarrow & *R \mid \mathbf{id} \\
R & \rightarrow & L
\end{array}
\qquad (4.49)
$$

LR(0) item sets

$$
\begin{array}{ll}
I_0: S' \rightarrow \bullet S & I_5: L \rightarrow * \bullet R \\
\quad S \rightarrow \bullet L = R & \quad R \rightarrow \bullet L \\
\quad S \rightarrow \bullet R & \quad L \rightarrow \bullet * R \\
\quad L \rightarrow \bullet * R & \quad L \rightarrow \bullet \mathtt{id} \\
\quad L \rightarrow \bullet \mathtt{id} & I_6: S \rightarrow L = \bullet R \\
\quad R \rightarrow \bullet L & \quad R \rightarrow \bullet L \\
I_1: S' \rightarrow S \bullet \$ & \quad L \rightarrow \bullet * R \\
I_2: S \rightarrow L \bullet = R & \quad L \rightarrow \bullet \mathtt{id} \\
\quad R \rightarrow L \bullet & I_7: L \rightarrow *R \bullet \\
I_3: S \rightarrow R \bullet & I_8: R \rightarrow L \bullet \\
I_4: L \rightarrow \mathtt{id} \bullet & I_9: S \rightarrow L = R \bullet
\end{array}
$$

LR(1) item sets

$$
\begin{array}{lllll}
I_0: & S' \rightarrow \bullet S, & \$ & I_5: L \rightarrow \mathtt{id} \bullet, & = \$ \\
& S \rightarrow \bullet L = R, & \$ & I_6: S \rightarrow L = \bullet R, & \$ \\
& S \rightarrow \bullet R, & \$ & \quad R \rightarrow \bullet L, & \$ \\
& L \rightarrow \bullet * R, & = & \quad L \rightarrow \bullet * R, & \$ \\
& L \rightarrow \bullet \mathtt{id}, & = & \quad L \rightarrow \bullet \mathtt{id}, & \$ \\
& R \rightarrow \bullet L, & \$ & I_7: L \rightarrow *R \bullet, & = \$ \\
& L \rightarrow \bullet * R, & \$ & I_8: R \rightarrow L \bullet, & = \$ \\
& L \rightarrow \bullet \mathtt{id}, & \$ & I_9: S \rightarrow L = R \bullet, & \$ \\
I_1: & S' \rightarrow S \bullet, & \$ & I_{10}: R \rightarrow L \bullet, & \$ \\
I_2: & S \rightarrow L \bullet = R, & \$ & I_{11}: L \rightarrow * \bullet R, & \$ \\
& R \rightarrow L \bullet, & \$ & \quad R \rightarrow \bullet L, & \$ \\
I_3: & S \rightarrow R \bullet, & \$ & \quad L \rightarrow \bullet * R, & \$ \\
I_4: & L \rightarrow * \bullet R, & = \$ & \quad L \rightarrow \bullet \mathtt{id}, & \$ \\
& R \rightarrow \bullet L, & = \$ & I_{12}: L \rightarrow \mathtt{id} \bullet, & \$ \\
& L \rightarrow \bullet * R, & = \$ & I_{13}: L \rightarrow *R \bullet, & \$ \\
& L \rightarrow \bullet \mathtt{id}, & = \$ & &
\end{array}
$$