LR parsing techniques

- SLR
  - Simple LR parsing
  - Easy to implement, not strong enough
  - Uses LR(0) items
- Canonical LR
  - Larger parser but powerful
  - Uses LR(1) items
- LALR (not in the book)
  - Condensed version of canonical LR
  - May introduce conflicts
  - Uses LR(1) items

SLR PARSER

Steps:

1. create augment grammar
2. generate kernel items
3. find closure
4. compute goto()
5. construct parsing table
6. parse the string

Let us consider grammar:

S->L=R

S->R

L->*R

L->id

R->L

- **Step :1 Create augment grammar**

  S is start symbol in the given grammar

  Augment grammar is S'-> S

- **Step :2 Generate kernel items**

Introduce dot in RHS of the production

S'-> ●S

- **Step :3 Find closure**

(*Rule*: *A -> α.Xβ i.e if there is nonterminal next to dot then Include X production))

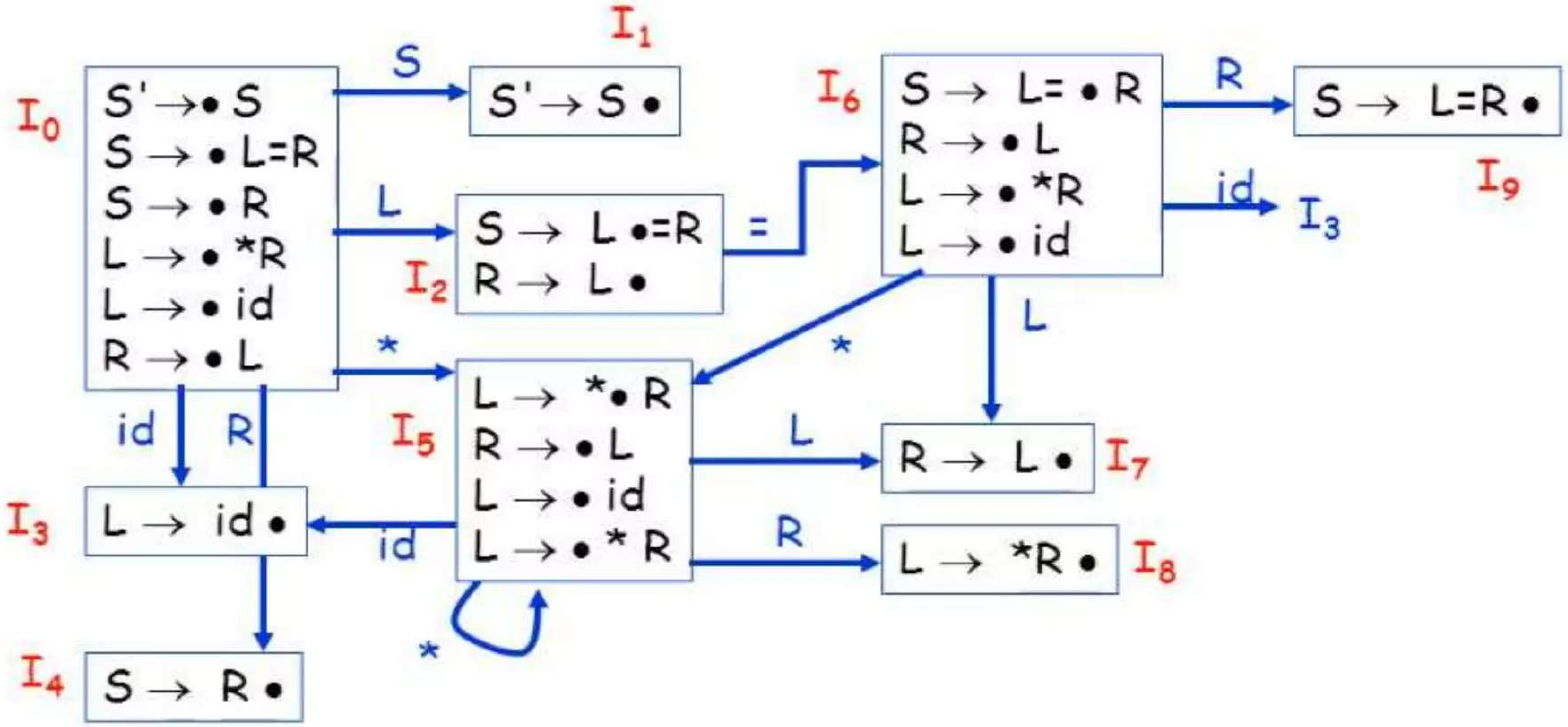
$$\boxed{\begin{array}{l} S' \to .S \\ S \to .\ L=R \\ S \to .\ R \\ L \to .\ {*}R \\ L \to .\ Id \\ R \to .\ L \end{array}} \Longrightarrow I0$$

Since presence of S (nonterminal)next to dot, introduce S production

Since presence of L (nonterminal)next to dot, introduce L production

Since presence of R (nonterminal)next to dot, introduce R production

SLR parsing



I0
S' → • S
S → • L=R
S → • R
L → • *R
L → • id
R → • L
S
I1
S' → S •
L
I2
S → L •=R
R → L •
=
I6
S → L= • R
R → • L
L → • *R
L → • id
R
S → L=R •
I9
id
I3
*
I5
L → *• R
R → • L
L → • id
L → • * R
L
R → L •
I7
R
L → *R •
I8
id
R
I3
L → id •
I4
S → R •
*

CLR PARSER

Steps:

1. create augment grammar
2. generate kernel items and add 2nd component
3. find closure
4. compute goto()
5. construct parsing table
6. parse the string

Let us consider grammar:

S->L=R

S->R

L->*R

L->id

R->L

Rule to find 2nd component:

Consider the production of the form : A-> α .Bβ , a

THEN 2nd component of B is : β , if it is terminal
First (β) if it is non terminal
a, if there is no β

- **Step :1 Create augment grammar**
  S is start symbol in the given grammar
  Augment grammar is S'-> S

- **Step :2 Generate kernel items and add 2nd component**

Introduce dot in RHS of the production

Add $ as 2nd component separated by comma

S'-> •S , $

- **Step :3 Find closure**

(*Rule:* *A -> α.Xβ i.e if there is nonterminal next to dot then Include X production))*

$$\boxed{\begin{array}{l} S' \to .S\ ,\ \$ \\ S \to .\ L{=}R \\ S \to .\ R \\ L \to .\ {*}R \\ L \to .\ Id \\ R \to .\ L \end{array}} \Longrightarrow I0$$

S'-> .S , $
S-> . L=R, $
S-> . R, $
L-> . *R, =/$
L-> . Id,=/$
R-> . L, $

$\Longrightarrow$ I0

Next find 2nd component:

compare each of the production with A-> α .Bβ , a

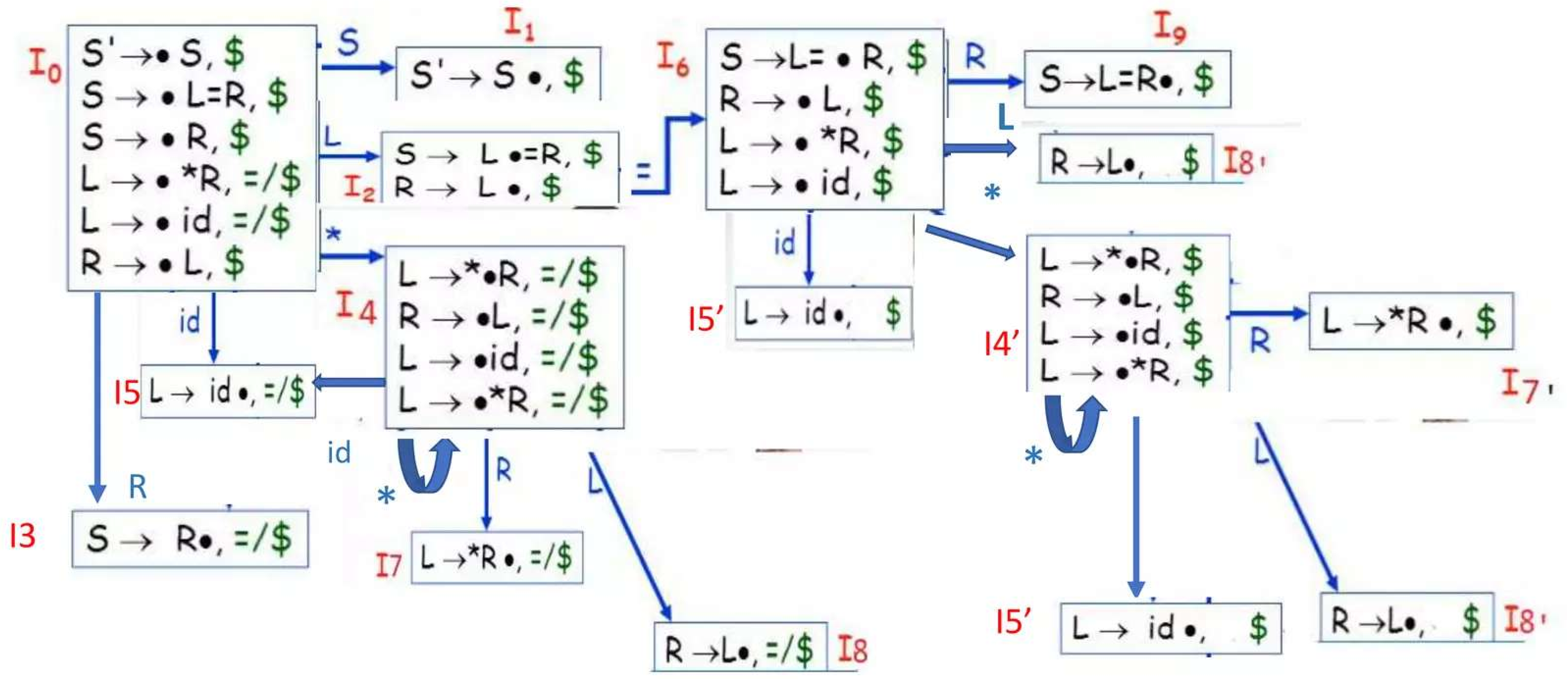
S' -> . s, $	here no β, so $ is 2nd comp to S
S -> . L=R,$	here β is = so add it as 2nd comp to L
S-> . R, $	here no β, so $ is 2nd comp to R
L	production is not in standard form
R-> . L, $	here no β, so $ is 2nd comp to L

Rule to find 2nd component:

Consider the production of the form : A-> α .Bβ , a

THEN 2nd component of B is : β , if it is terminal
First (β) if it is non terminal
a, if there is no β

Step :4 compute goto()



To construct: LALR PARSER

We notice that some states in CLR parser have the same core items and differ only in possible lookahead symbols.

Such as

I4 and I4’
I5 and I5’
I7 and I7’
I8 and I8’

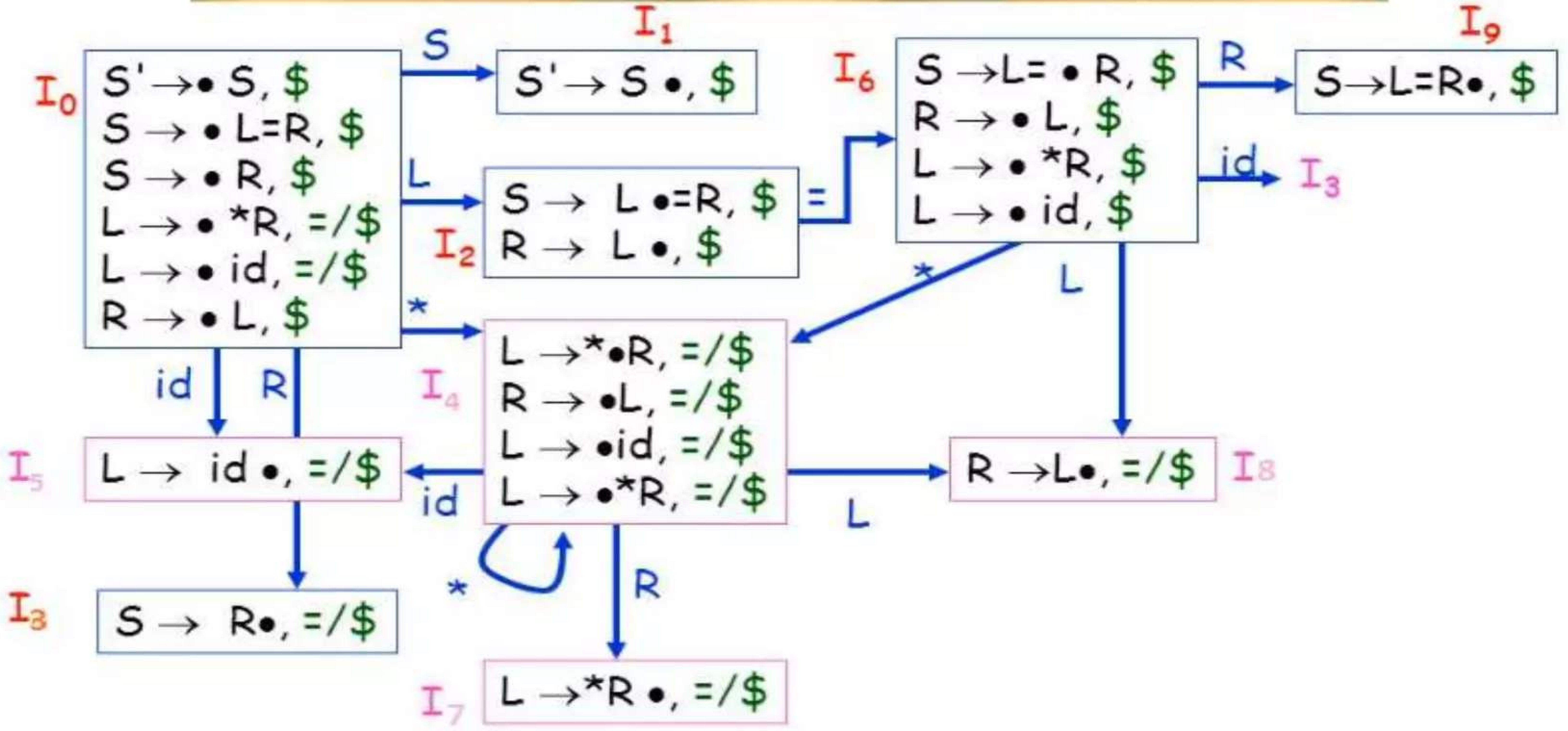
So we shrink the obtained CLR parser by merging such states to form LALR Parser

Hence

CLR PARSER has 14 States (I0, I1,I2,I3,I4,I4’,I5,I5’,I6,I7,I7’,I8,I8’,I9)

LALR PARSER has 10 states (I0, I1,I2,I3,I4,I5,I6,I7,I8,I9)

LALR PARSER


I0
S' → • S, $
S → • L=R, $
S → • R, $
L → • *R, =/$
L → • id, =/$
R → • L, $
S
I1
S' → S •, $
L
I2
S → L •=R, $
R → L •, $
=
I6
S → L= • R, $
R → • L, $
L → • *R, $
L → • id, $
R
I9
S → L=R•, $
id
I3
*
I4
L → *•R, =/$
R → •L, =/$
L → •id, =/$
L → •*R, =/$
id
R
I5
L → id •, =/$
I3
S → R•, =/$
*
R
I7
L → *R •, =/$
L
L
I8
R → L•, =/$