Limitation of SLR(1) Parsing

- We illustrate the limitations of SLR(1) parsing by using the pointer assignment grammar given below

$$
\begin{array}{rcl}
S & \rightarrow & L = R \mid R \\
L & \rightarrow & *R \mid \text{id} \\
R & \rightarrow & L
\end{array}
$$

- We compute the FOLLOW sets and sets of LR(0) items to demonstrate the problem
- We explain the cause of the problem
- This explanation leads us to a more precise method of CLR(1) parsing (Canonical LR(1) parsing that uses the LR(1) items)

IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Syntax Analysis

Section:

Computing the FOLLOW Sets for Pointer Assignment Grammar

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow L = R \mid R \\
L &\rightarrow *R \mid \text{id} \\
R &\rightarrow L
\end{aligned}
$$

Computing the FOLLOW Sets for Pointer Assignment Grammar

$$
\begin{array}{rcl}
S' & \rightarrow & S \\
S & \rightarrow & L = R \mid R \\
L & \rightarrow & *R \mid \text{id} \\
R & \rightarrow & L
\end{array}
\qquad
\Rightarrow
\begin{array}{rcl}
\text{FOLLOW}(S') & \supseteq & \{\$\} \\
\text{FOLLOW}(S) & \supseteq & \text{FOLLOW}(S')
\end{array}
$$

Computing the FOLLOW Sets for Pointer Assignment Grammar

$$
\begin{array}{rcll}
S' & \rightarrow & S & \Rightarrow \text{FOLLOW}(S') \supseteq \{\$\} \\
 & & & \text{FOLLOW}(S) \supseteq \text{FOLLOW}(S') \\
S & \rightarrow & L = R \mid R & \Rightarrow \text{FOLLOW}(L) \supseteq \{=\} \\
 & & & \text{FOLLOW}(R) \supseteq \text{FOLLOW}(S) \\
L & \rightarrow & *R \mid \text{id} & \\
R & \rightarrow & L &
\end{array}
$$

Computing the FOLLOW Sets for Pointer Assignment Grammar

$$
\begin{array}{rcl}
S' & \rightarrow & S \\
S & \rightarrow & L = R \mid R \\
L & \rightarrow & *R \mid \text{id} \\
R & \rightarrow & L
\end{array}
\qquad
\begin{array}{l}
\Rightarrow \\
\\
\Rightarrow \\
\\
\Rightarrow
\end{array}
\qquad
\begin{array}{rcl}
\text{FOLLOW}(S') & \supseteq & \{\$\} \\
\text{FOLLOW}(S) & \supseteq & \text{FOLLOW}(S') \\
\text{FOLLOW}(L) & \supseteq & \{=\} \\
\text{FOLLOW}(R) & \supseteq & \text{FOLLOW}(S) \\
\text{FOLLOW}(R) & \supseteq & \text{FOLLOW}(L)
\end{array}
$$

Computing the FOLLOW Sets for Pointer Assignment Grammar

$$
\begin{array}{rcllcl}
S' & \rightarrow & S & \Rightarrow & \text{FOLLOW}(S') \supseteq & \{\$\} \\
 & & & & \text{FOLLOW}(S) \supseteq & \text{FOLLOW}(S') \\
S & \rightarrow & L = R \mid R & \Rightarrow & \text{FOLLOW}(L) \supseteq & \{=\} \\
 & & & & \text{FOLLOW}(R) \supseteq & \text{FOLLOW}(S) \\
L & \rightarrow & *R \mid \text{id} & \Rightarrow & \text{FOLLOW}(R) \supseteq & \text{FOLLOW}(L) \\
R & \rightarrow & L & \Rightarrow & \text{FOLLOW}(L) \supseteq & \text{FOLLOW}(R)
\end{array}
$$

Computing the FOLLOW Sets for Pointer Assignment Grammar

$$
\begin{array}{rcl@{\quad}l@{\quad}rcl}
S' & \rightarrow & S & \Rightarrow & \text{FOLLOW}(S') & \supseteq & \{\$\} \\
 & & & & \text{FOLLOW}(S) & \supseteq & \text{FOLLOW}(S') \\
S & \rightarrow & L = R \mid R & \Rightarrow & \text{FOLLOW}(L) & \supseteq & \{=\} \\
 & & & & \text{FOLLOW}(R) & \supseteq & \text{FOLLOW}(S) \\
L & \rightarrow & *R \mid \text{id} & \Rightarrow & \text{FOLLOW}(R) & \supseteq & \text{FOLLOW}(L) \\
R & \rightarrow & L & \Rightarrow & \text{FOLLOW}(L) & \supseteq & \text{FOLLOW}(R)
\end{array}
$$

	FOLLOW
S'	$\{\$\}$
S	$\{\$\}$
R	$\{=, \$\}$
L	$\{=, \$\}$

LR(0) Item Sets for Pointer Assignment Grammar

IIT Bombay
cs302: Implementation of Programming Languages

Topic:
Syntax Analysis

Section:
Grammars, Derivations, and Parse Trees
Shift Reduce Parsing
SLR(1) Parsing
Conceptual Issues in Parsing
CLR(1) Parsing
LALR(1) Parsing

I_0

$$\boxed{\begin{array}{l} S' \rightarrow \bullet S \\ \hline S \rightarrow \bullet L = R \\ S \rightarrow \bullet R \\ L \rightarrow \bullet * R \\ L \rightarrow \bullet \mathrm{id} \\ R \rightarrow \bullet L \end{array}}$$

LR(0) Item Sets for Pointer Assignment Grammar

$I_0 \xrightarrow{S} I_1$

I_0

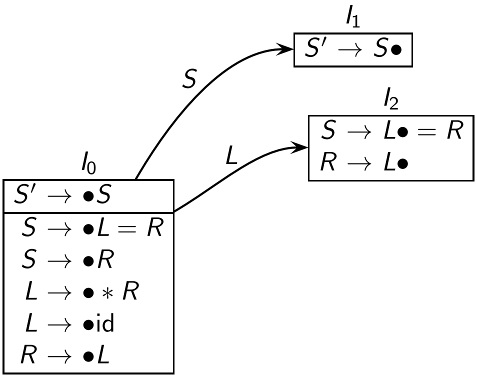
$S' \rightarrow \bullet S$

$S \rightarrow \bullet L = R$
$S \rightarrow \bullet R$
$L \rightarrow \bullet * R$
$L \rightarrow \bullet \text{id}$
$R \rightarrow \bullet L$

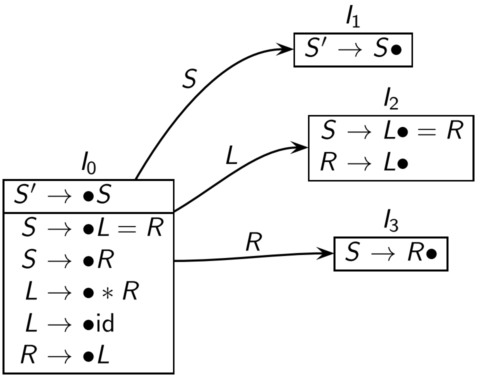
I_1

$S' \rightarrow S \bullet$

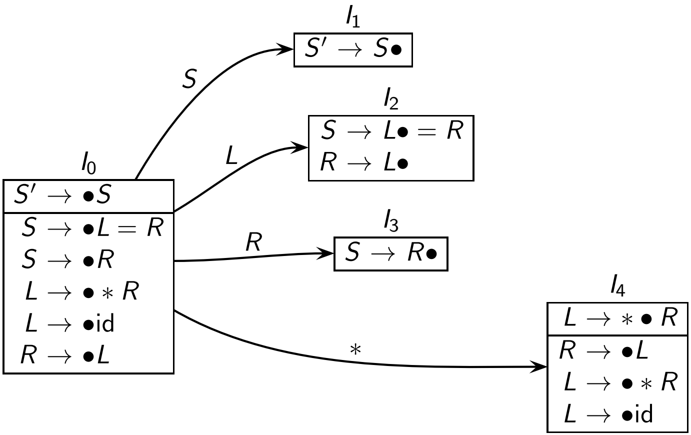
LR(0) Item Sets for Pointer Assignment Grammar



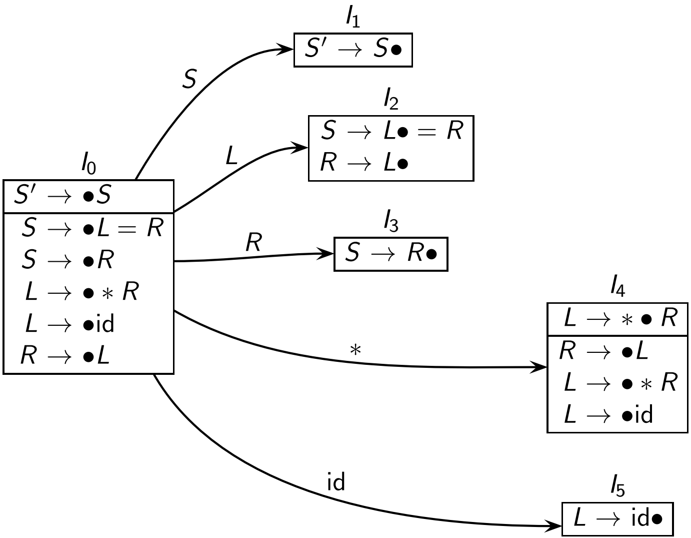
LR(0) Item Sets for Pointer Assignment Grammar



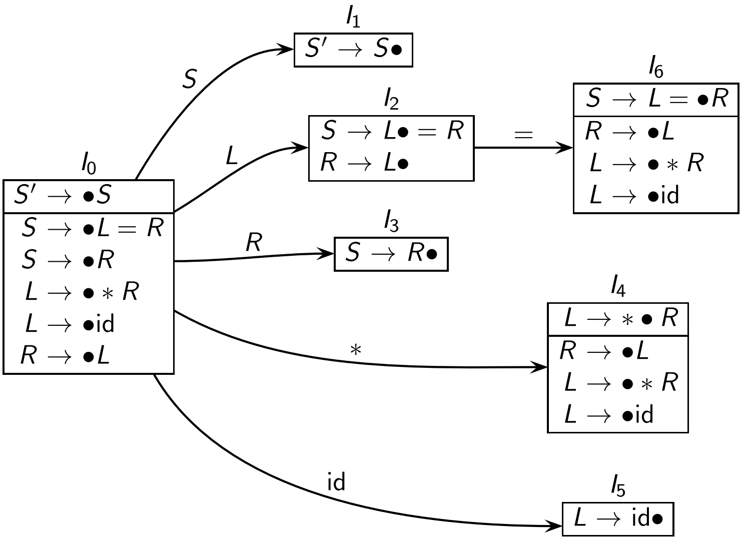
LR(0) Item Sets for Pointer Assignment Grammar



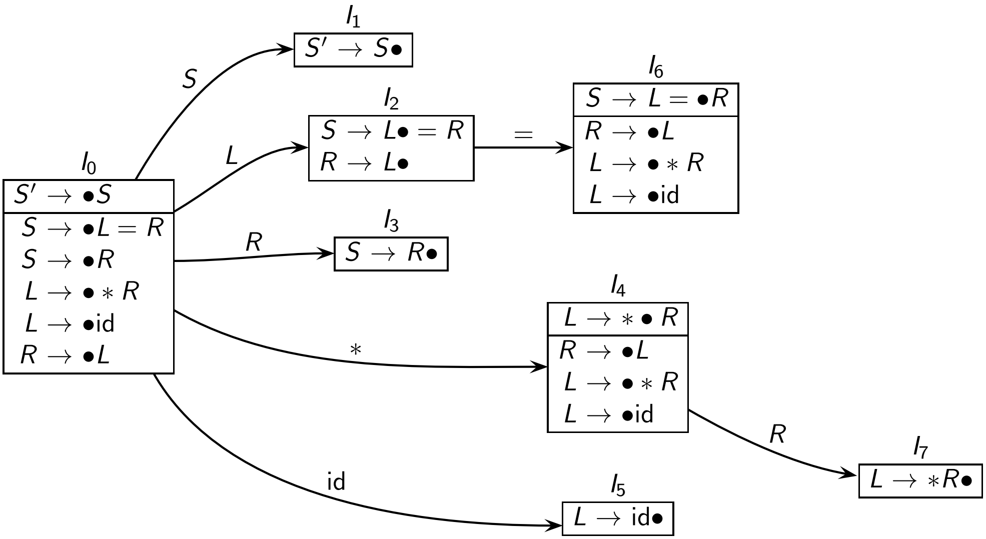
LR(0) Item Sets for Pointer Assignment Grammar



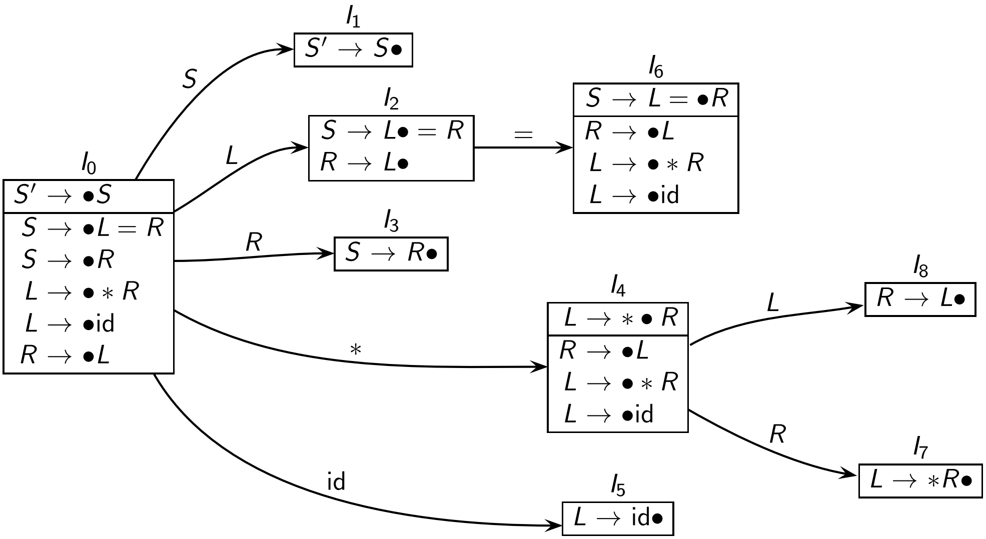
LR(0) Item Sets for Pointer Assignment Grammar



LR(0) Item Sets for Pointer Assignment Grammar



LR(0) Item Sets for Pointer Assignment Grammar



IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Syntax Analysis

Section:

Grammars, Derivations, and Parse Trees

Shift Reduce Parsing

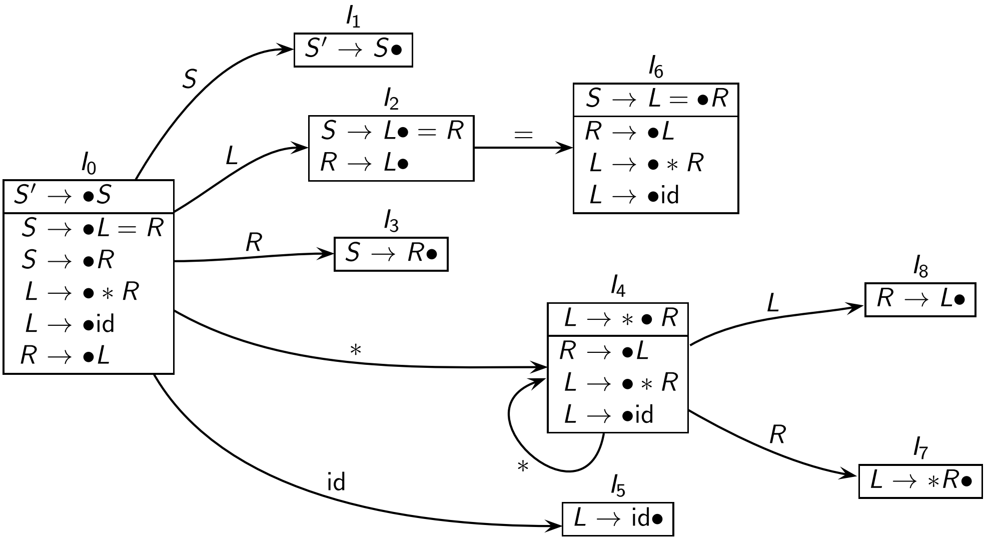
SLR(1) Parsing

Conceptual Issues in Parsing

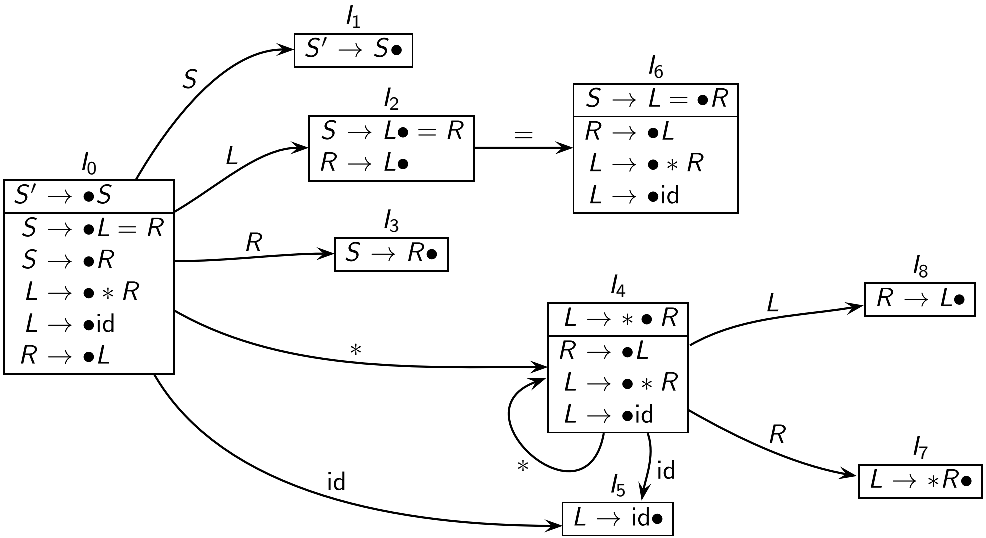
CLR(1) Parsing

LALR(1) Parsing

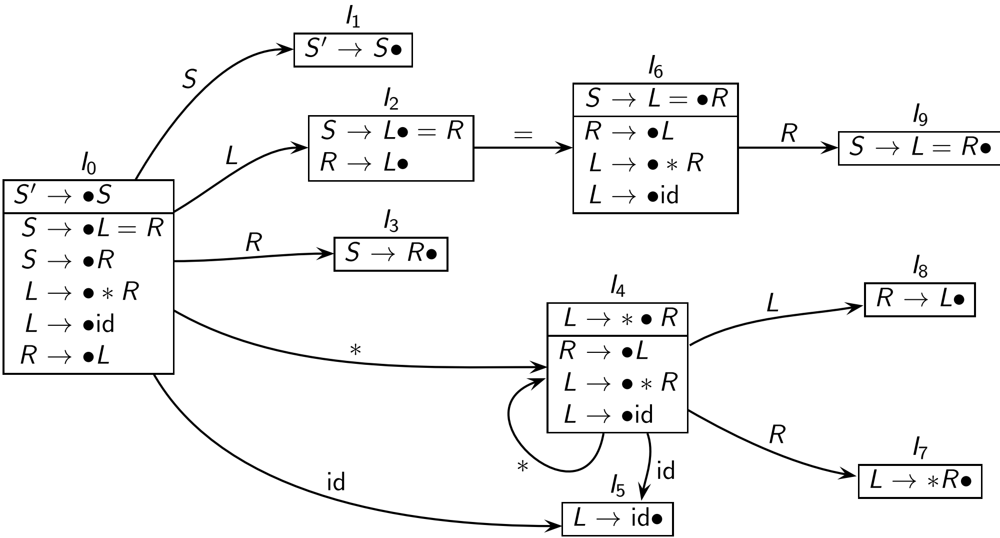
LR(0) Item Sets for Pointer Assignment Grammar



LR(0) Item Sets for Pointer Assignment Grammar



LR(0) Item Sets for Pointer Assignment Grammar



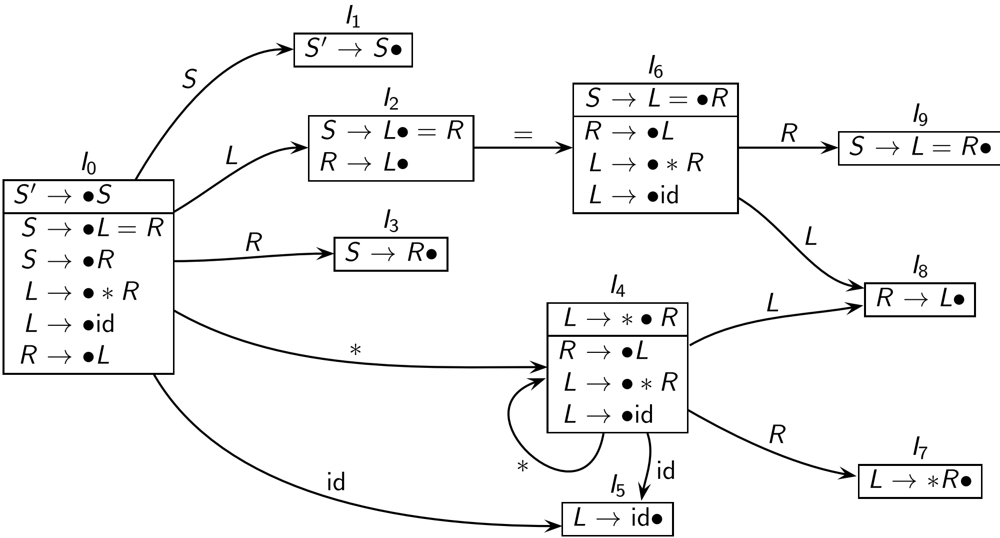
IIT Bombay
cs302: Implementation of Programming Languages

Topic:

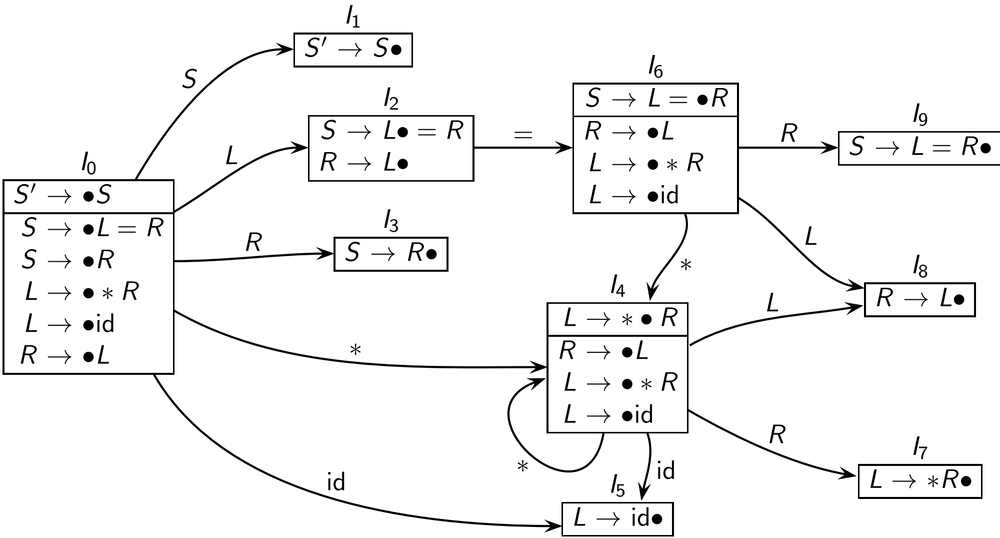
Syntax Analysis

Section:

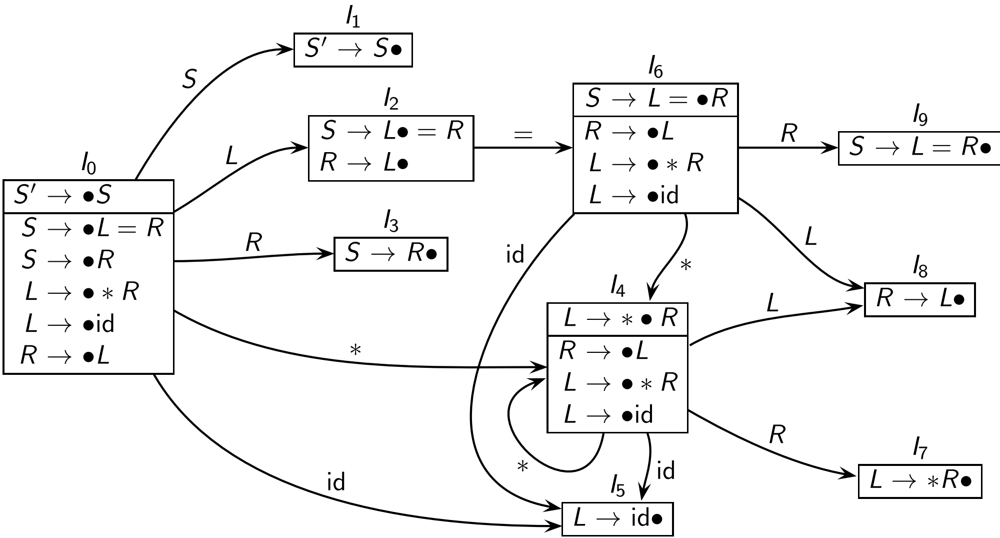
LR(0) Item Sets for Pointer Assignment Grammar



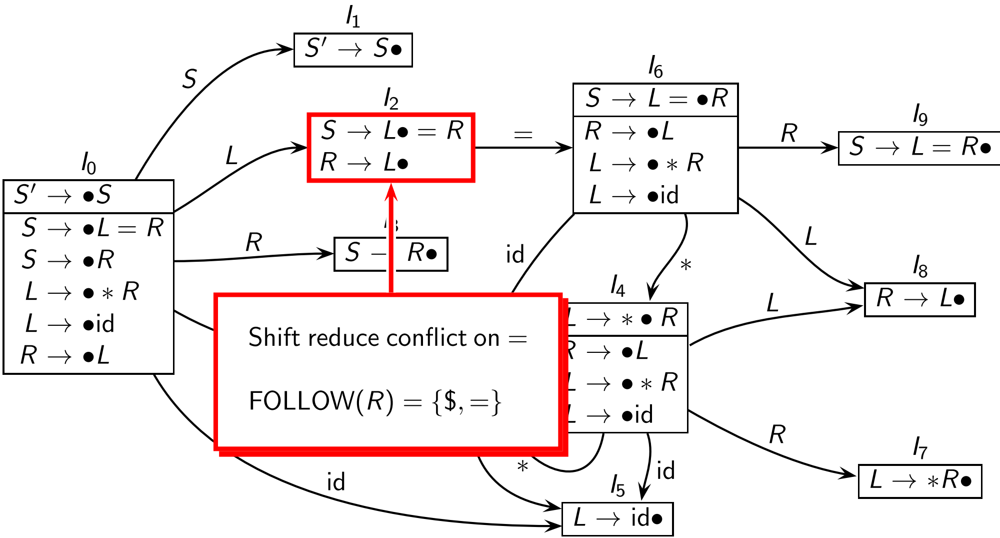
LR(0) Item Sets for Pointer Assignment Grammar



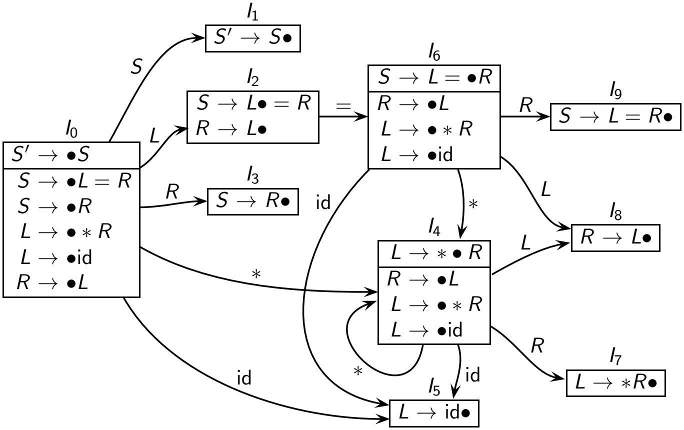
LR(0) Item Sets for Pointer Assignment Grammar



LR(0) Item Sets for Pointer Assignment Grammar

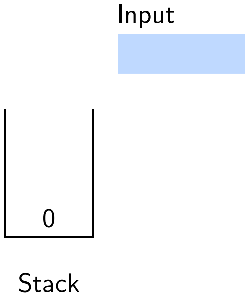


Limitation of SLR(1) Parsing

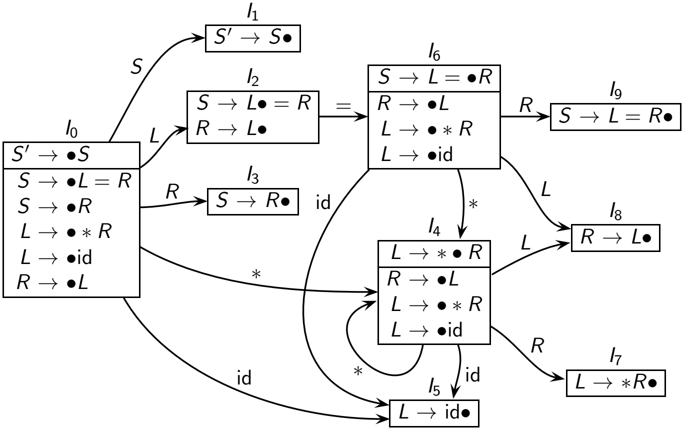


1	S	$\rightarrow$	$L = R$
2	S	$\rightarrow$	R
3	L	$\rightarrow$	$*R$
4	L	$\rightarrow$	id
5	R	$\rightarrow$	L

	FOLLOW
S'	$\{\$\}$
S	$\{\$\}$
R	$\{=, \$\}$
L	$\{=, \$\}$



Limitation of SLR(1) Parsing



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

	FOLLOW
S'	$\{\$\}$
S	$\{\$\}$
R	$\{=, \$\}$
L	$\{=, \$\}$

Shift 5

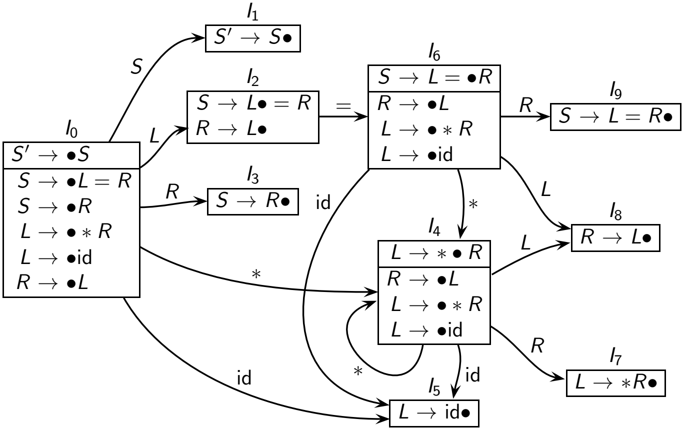
Input

$\text{id} = \text{id}\$$

0

Stack

Limitation of SLR(1) Parsing



1	S	$\rightarrow$	$L = R$
2	S	$\rightarrow$	R
3	L	$\rightarrow$	$*R$
4	L	$\rightarrow$	id
5	R	$\rightarrow$	L

	FOLLOW
S'	$\{\$\}$
S	$\{\$\}$
R	$\{=, \$\}$
L	$\{=, \$\}$

Reduce by 4

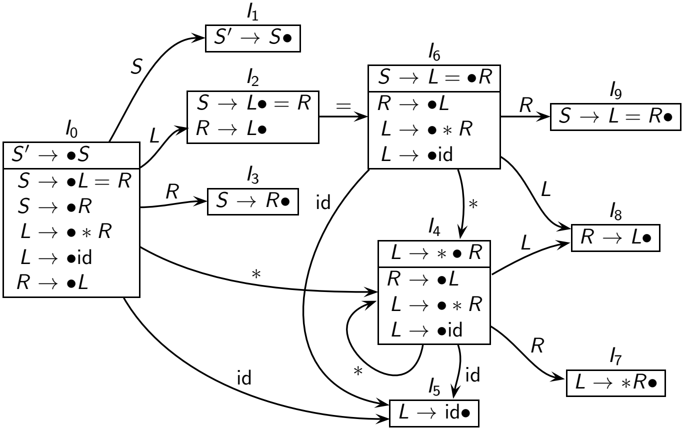
Input

$= \text{id}\$$

5
id
0

Stack

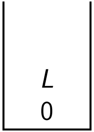
Limitation of SLR(1) Parsing



Cover by 2

Input

$= \text{id}\$$

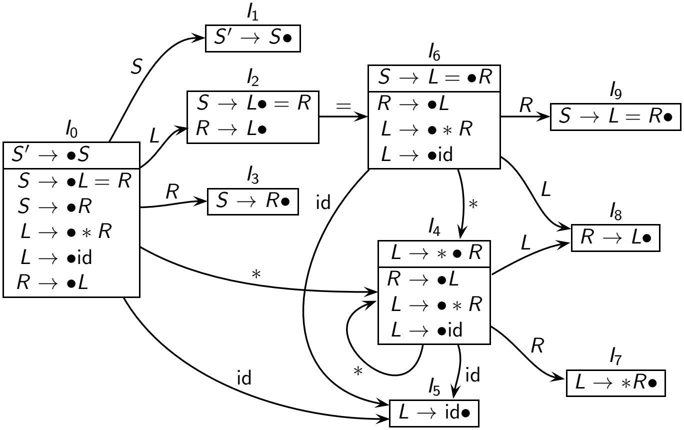


Stack

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

	FOLLOW
S'	$\{\$\}$
S	$\{\$\}$
R	$\{=, \$\}$
L	$\{=, \$\}$

Limitation of SLR(1) Parsing



Reduce by 5

Input

$= \text{id}\$$

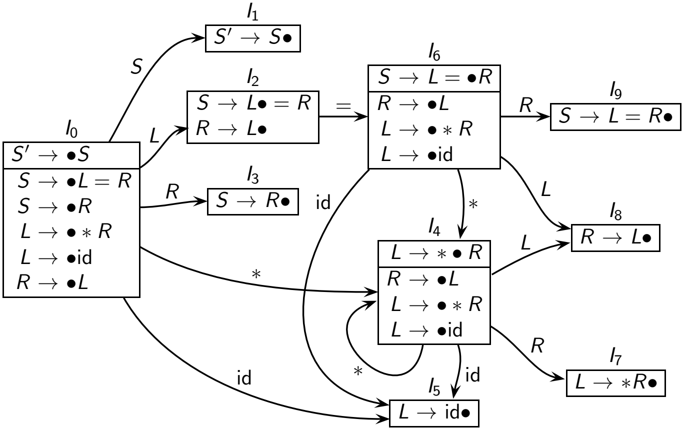
1	S	$\to$	$L = R$
2	S	$\to$	R
3	L	$\to$	$*R$
4	L	$\to$	id
5	R	$\to$	L

	FOLLOW
S'	$\{\$\}$
S	$\{\$\}$
R	$\{=, \$\}$
L	$\{=, \$\}$

Stack
2
L
0

Stack

Limitation of SLR(1) Parsing



Cover by 3

1	S	$\rightarrow$	$L = R$
2	S	$\rightarrow$	R
3	L	$\rightarrow$	$*R$
4	L	$\rightarrow$	id
5	R	$\rightarrow$	L

	FOLLOW
S'	$\{\$\}$
S	$\{\$\}$
R	$\{=, \$\}$
L	$\{=, \$\}$

Input

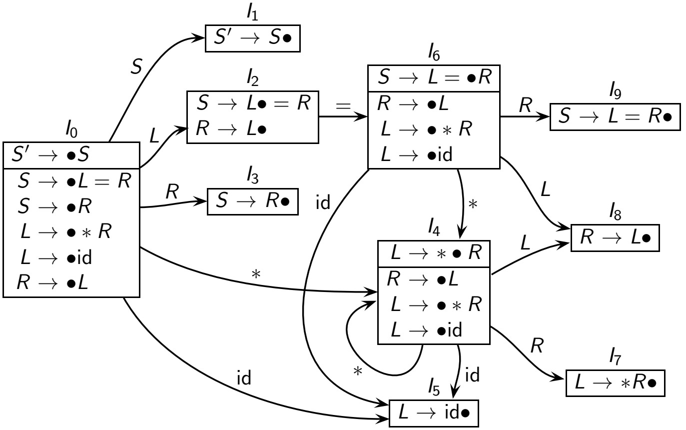
$= id\$$

R

0

Stack

Limitation of SLR(1) Parsing



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

	FOLLOW
S'	$\{\$\}$
S	$\{\$\}$
R	$\{=, \$\}$
L	$\{=, \$\}$

Error

No action on =

Input

$= \text{id}\$$

3
R
0

Stack

Limitation of SLR(1) Parsing: Use of FOLLOW Information

- Let FOLLOW(A) $= \{b, c\}$. Then b may follow A in some right sentential forms whereas in some other right sentential form, c may follow A

  A symbol in follow set need not follow A in *every* right sentential form

Limitation of SLR(1) Parsing: Use of FOLLOW Information

- Let FOLLOW(A) $= \{b, c\}$. Then b may follow A in some right sentential forms whereas in some other right sentential form, c may follow A

  A symbol in follow set need not follow A in *every* right sentential form

- We should declare handle $A \rightarrow \alpha$ in a viable prefix γ only if the follow symbols actually follows A in the right sentential form containing γ

Limitation of SLR(1) Parsing: Use of FOLLOW Information

- Let FOLLOW(A) $= \{b, c\}$. Then b may follow A in some right sentential forms whereas in some other right sentential form, c may follow A

  A symbol in follow set need not follow A in *every* right sentential form

- We should declare handle $A \rightarrow \alpha$ in a viable prefix γ only if the follow symbols actually follows A in the right sentential form containing γ

- In our grammar, there is no right sentential form with a prefix '$R =$'

Limitation of SLR(1) Parsing: Use of FOLLOW Information

- Let FOLLOW(A) $= \{b, c\}$. Then b may follow A in some right sentential forms whereas in some other right sentential form, c may follow A

  A symbol in follow set need not follow A in *every* right sentential form
- We should declare handle $A \rightarrow \alpha$ in a viable prefix γ only if the follow symbols actually follows A in the right sentential form containing γ
- In our grammar, there is no right sentential form with a prefix '$R =$'
  - Every right sentential form containing '$R =$' begins with a '$*$' and has a viable prefix '$*R$'
    We will never see '$=$' after an R without seeing a '$*$' before the 'R'

Limitation of SLR(1) Parsing: Use of FOLLOW Information

- Let FOLLOW(A) $= \{b, c\}$. Then b may follow A in some right sentential forms whereas in some other right sentential form, c may follow A

  A symbol in follow set need not follow A in *every* right sentential form
- We should declare handle $A \rightarrow \alpha$ in a viable prefix γ only if the follow symbols actually follows A in the right sentential form containing γ
- In our grammar, there is no right sentential form with a prefix '$R =$'
  - Every right sentential form containing '$R =$' begins with a '$*$' and has a viable prefix '$*R$'

    We will never see '$=$' after an R without seeing a '$*$' before the 'R'
  - $S \overset{rm}{\Rightarrow} L = R \overset{rm}{\Rightarrow} L = L \overset{rm}{\Rightarrow} L = \text{id} \overset{rm}{\Rightarrow} \text{id} = \text{id}$

    $S \overset{rm}{\Rightarrow} L = R \overset{rm}{\Rightarrow} L = \text{id} \overset{rm}{\Rightarrow} *R = \text{id} \overset{rm}{\Rightarrow} *\text{id} = \text{id}$

    $S \overset{rm}{\Rightarrow} L = R \overset{rm}{\Rightarrow} L = \text{id} \overset{rm}{\Rightarrow} *R = \text{id} \overset{rm}{\Rightarrow} *L = \text{id} \overset{rm}{\Rightarrow} *\text{id} = \text{id}$

    $\ldots$

Limitation of SLR(1) Parsing: Use of FOLLOW Information

- Let FOLLOW(A) $= \{b, c\}$. Then b may follow A in some right sentential forms whereas in some other right sentential form, c may follow A

  A symbol in follow set need not follow A in *every* right sentential form

- We should declare handle $A \rightarrow \alpha$ in a viable prefix γ only if the follow symbols actually follows A in the right sentential form containing γ

- In our grammar, there is no right sentential form with a prefix '$R =$'
  - Every right sentential form containing '$R =$' begins with a '$*$' and has a viable prefix '$*R$'
    We will never see '$=$' after an R without seeing a '$*$' before the 'R'
  - $S \overset{rm}{\Rightarrow} L = R \overset{rm}{\Rightarrow} L = L \overset{rm}{\Rightarrow} L = \text{id} \overset{rm}{\Rightarrow} \text{id} = \text{id}$
    $S \overset{rm}{\Rightarrow} L = R \overset{rm}{\Rightarrow} L = \text{id} \overset{rm}{\Rightarrow} *R = \text{id} \overset{rm}{\Rightarrow} *\text{id} = \text{id}$
    $S \overset{rm}{\Rightarrow} L = R \overset{rm}{\Rightarrow} L = \text{id} \overset{rm}{\Rightarrow} *R = \text{id} \overset{rm}{\Rightarrow} *L = \text{id} \overset{rm}{\Rightarrow} *\text{id} = \text{id}$
    ...
  - '$=$' is in FOLLOW(R) only for the right sentential forms that begin with a '$*$'
    Input 'id $=$ id' does not begin with a '$*$' so L cannot be reduced to R on '$=$'

LR(1) Item Sets for Pointer Assignment Grammar

Two changes from LR(0) construction

- Items are of the form $A \rightarrow \alpha \bullet \beta, a$ consisting of
  - the *core* $A \rightarrow \alpha \bullet \beta$ and
  - the *lookahead* a

  If S is the start symbol, then I_0 contains $S' \rightarrow \bullet S, \$$
- Closure of an item $A \rightarrow \alpha \bullet B\beta, a$ contains the items of the form $B \rightarrow \bullet\gamma, \text{FIRST}(\beta a)$

Transition of an item $A \rightarrow \alpha \bullet B\beta, a$ on B gives an item $A \rightarrow \alpha B \bullet \beta, a$

The lookahead does not change during a transition

LR(1) Item Sets for Pointer Assignment Grammar

Two changes from LR(0) construction

- Items are of the form $A \rightarrow \alpha \bullet \beta, a$ consisting of
  - the *core* $A \rightarrow \alpha \bullet \beta$ and
  - the *lookahead* a

  If S is the start symbol, then I_0 contains $S' \rightarrow \bullet S, \$$

- Closure of an item $A \rightarrow \alpha \bullet B\beta, a$ contains the items of the form $B \rightarrow \bullet\gamma, \text{FIRST}(\beta a)$

Transition of an item $A \rightarrow \alpha \bullet B\beta, a$ on B gives an item $A \rightarrow \alpha B \bullet \beta, a$

The lookahead does not change during a transition

$S' \rightarrow \bullet S,$	$\$$

I_0

LR(1) Item Sets for Pointer Assignment Grammar

Two changes from LR(0) construction

- Items are of the form $A \to \alpha \bullet \beta, a$ consisting of
  - the *core* $A \to \alpha \bullet \beta$ and
  - the *lookahead* a

  If S is the start symbol, then I_0 contains $S' \to \bullet S, \$$
- Closure of an item $A \to \alpha \bullet B\beta, a$ contains the items of the form $B \to \bullet\gamma, \text{FIRST}(\beta a)$

Transition of an item $A \to \alpha \bullet B\beta, a$ on B gives an item $A \to \alpha B \bullet \beta, a$

The lookahead does not change during a transition

I_0	
$S' \to \bullet S,$	$\$$
$S \to \bullet L = R,$	$\$$
$S \to \bullet R,$	$\$$

LR(1) Item Sets for Pointer Assignment Grammar

Two changes from LR(0) construction

- Items are of the form $A \rightarrow \alpha \bullet \beta, a$ consisting of
  - the *core* $A \rightarrow \alpha \bullet \beta$ and
  - the *lookahead* a

  If S is the start symbol, then I_0 contains $S' \rightarrow \bullet S, \$$
- Closure of an item $A \rightarrow \alpha \bullet B\beta, a$ contains the items of the form $B \rightarrow \bullet\gamma, \text{FIRST}(\beta a)$

Transition of an item $A \rightarrow \alpha \bullet B\beta, a$ on B gives an item $A \rightarrow \alpha B \bullet \beta, a$

The lookahead does not change during a transition

I_0

$S' \rightarrow \bullet S,$	$\$$
$S \rightarrow \bullet L = R,$	$\$$
$S \rightarrow \bullet R,$	$\$$
$L \rightarrow \bullet * R,$	$=$
$L \rightarrow \bullet \text{id},$	$=$

LR(1) Item Sets for Pointer Assignment Grammar

Two changes from LR(0) construction

- Items are of the form $A \rightarrow \alpha \bullet \beta, a$ consisting of
  - the *core* $A \rightarrow \alpha \bullet \beta$ and
  - the *lookahead* a

  If S is the start symbol, then I_0 contains $S' \rightarrow \bullet S, \$$

- Closure of an item $A \rightarrow \alpha \bullet B\beta, a$ contains the items of the form $B \rightarrow \bullet\gamma, \text{FIRST}(\beta a)$

Transition of an item $A \rightarrow \alpha \bullet B\beta, a$ on B gives an item $A \rightarrow \alpha B \bullet \beta, a$

The lookahead does not change during a transition

I_0

$S' \rightarrow \bullet S,$	$\$$
$S \rightarrow \bullet L = R,$	$\$$
$S \rightarrow \bullet R,$	$\$$
$L \rightarrow \bullet * R,$	$=$
$L \rightarrow \bullet \text{id},$	$=$
$R \rightarrow \bullet L,$	$\$$

LR(1) Item Sets for Pointer Assignment Grammar

Two changes from LR(0) construction

- Items are of the form $A \rightarrow \alpha \bullet \beta, a$ consisting of
  - the *core* $A \rightarrow \alpha \bullet \beta$ and
  - the *lookahead* a

  If S is the start symbol, then I_0 contains $S' \rightarrow \bullet S, \$$

- Closure of an item $A \rightarrow \alpha \bullet B\beta, a$ contains the items of the form $B \rightarrow \bullet\gamma, \text{FIRST}(\beta a)$

Transition of an item $A \rightarrow \alpha \bullet B\beta, a$ on B gives an item $A \rightarrow \alpha B \bullet \beta, a$

The lookahead does not change during a transition

I_0

$S' \rightarrow \bullet S,$	$\$$
$S \rightarrow \bullet L = R,$	$\$$
$S \rightarrow \bullet R,$	$\$$
$L \rightarrow \bullet * R,$	$=$
$L \rightarrow \bullet \text{id},$	$=$
$R \rightarrow \bullet L,$	$\$$
$L \rightarrow \bullet * R,$	$\$$
$L \rightarrow \bullet \text{id},$	$\$$

LR(1) Item Sets for Pointer Assignment Grammar

Two changes from LR(0) construction

- Items are of the form $A \rightarrow \alpha \bullet \beta, a$ consisting of
  - the *core* $A \rightarrow \alpha \bullet \beta$ and
  - the *lookahead* a

  If S is the start symbol, then I_0 contains $S' \rightarrow \bullet S, \$$
- Closure of an item $A \rightarrow \alpha \bullet B\beta, a$ contains the items of the form $B \rightarrow \bullet\gamma, \text{FIRST}(\beta a)$

Transition of an item $A \rightarrow \alpha \bullet B\beta, a$ on B gives an item $A \rightarrow \alpha B \bullet \beta, a$

The lookahead does not change during a transition

I_0

Item	Lookahead
$S' \rightarrow \bullet S,$	$\$$
$S \rightarrow \bullet L = R,$	$\$$
$S \rightarrow \bullet R,$	$\$$
$L \rightarrow \bullet * R,$	$=$
$L \rightarrow \bullet \text{id},$	$=$
$R \rightarrow \bullet L,$	$\$$
$L \rightarrow \bullet * R,$	$\$$
$L \rightarrow \bullet \text{id},$	$\$$

$I_0 \xrightarrow{L} I_2$

I_2

Item	Lookahead
$S \rightarrow L \bullet = R,$	$\$$
$R \rightarrow L \bullet,$	$\$$

LR(1) Item Sets for Pointer Assignment Grammar

Two changes from LR(0) construction

- Items are of the form $A \rightarrow \alpha \bullet \beta, a$ consisting of
  - the *core* $A \rightarrow \alpha \bullet \beta$ and
  - the *lookahead* a

  If S is the start symbol, then I_0 contains $S' \rightarrow \bullet S, \$$
- Closure of an item $A \rightarrow \alpha \bullet B\beta, a$ contains the items of the form $B \rightarrow \bullet\gamma, \text{FIRST}(\beta a)$

Transition of an item $A \rightarrow \alpha \bullet B\beta, a$ on B gives an item $A \rightarrow \alpha B \bullet \beta, a$

The lookahead does not change during a transition

I_0:

$S' \rightarrow \bullet S,$	$\$$
$S \rightarrow \bullet L = R,$	$\$$
$S \rightarrow \bullet R,$	$\$$
$L \rightarrow \bullet * R,$	$=$
$L \rightarrow \bullet \text{id},$	$=$
$R \rightarrow \bullet L,$	$\$$
$L \rightarrow \bullet * R,$	$\$$
$L \rightarrow \bullet \text{id},$	$\$$

$I_0 \xrightarrow{L} I_2$

I_2:

$S \rightarrow L \bullet = R,$	$\$$
$R \rightarrow L \bullet,$	$\$$

Reduction by $R \rightarrow L\bullet$ only on $\$$ and not on $=$

No shift reduce conflict

LR(1) Sets of Items for Pointer Assignment Grammar

I_0

$S' \rightarrow \bullet S,$		$\$$
$S \rightarrow \bullet L = R,$		$\$$
$S \rightarrow \bullet R,$		$\$$
$L \rightarrow \bullet * R,$		$=/\$$
$L \rightarrow \bullet \text{id},$		$=/\$$
$R \rightarrow \bullet L,$		$\$$

LR(1) Sets of Items for Pointer Assignment Grammar

I_0

$S' \rightarrow \bullet S,$	$\$$
$S \rightarrow \bullet L = R,$	$\$$
$S \rightarrow \bullet R,$	$\$$
$L \rightarrow \bullet * R,$	$= /\$$
$L \rightarrow \bullet \text{id},$	$= /\$$
$R \rightarrow \bullet L,$	$\$$

$$I_0 \xrightarrow{S} I_1$$

I_1

$S' \rightarrow S\bullet, \$$

LR(1) Sets of Items for Pointer Assignment Grammar

I_0

$S' \rightarrow \bullet S,$	$\$$
$S \rightarrow \bullet L = R,$	$\$$
$S \rightarrow \bullet R,$	$\$$
$L \rightarrow \bullet * R,$	$= / \$$
$L \rightarrow \bullet \text{id},$	$= / \$$
$R \rightarrow \bullet L,$	$\$$

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{L} I_2$

I_1

$S' \rightarrow S \bullet,$	$\$$

I_2

$S \rightarrow L \bullet = R,$	$\$$
$R \rightarrow L \bullet,$	$\$$

LR(1) Sets of Items for Pointer Assignment Grammar

I_0

$S' \to \bullet S,$	$\$$
$S \to \bullet L = R,$	$\$$
$S \to \bullet R,$	$\$$
$L \to \bullet * R,$	$= / \$$
$L \to \bullet \text{id},$	$= / \$$
$R \to \bullet L,$	$\$$

$I_0 \xrightarrow{S} I_1$

I_1

$S' \to S\bullet,$	$\$$

$I_0 \xrightarrow{L} I_2$

I_2

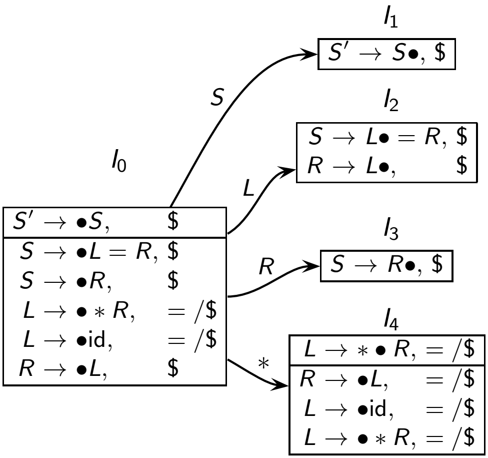
$S \to L\bullet = R,$	$\$$
$R \to L\bullet,$	$\$$

$I_0 \xrightarrow{R} I_3$

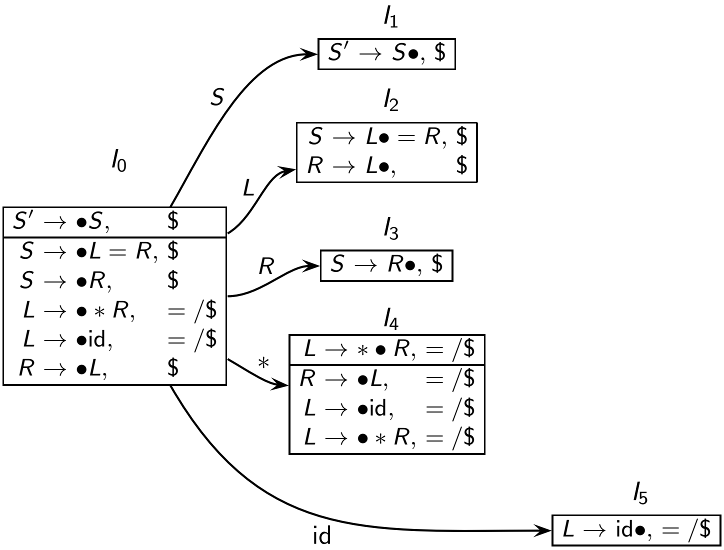
I_3

$S \to R\bullet,$	$\$$

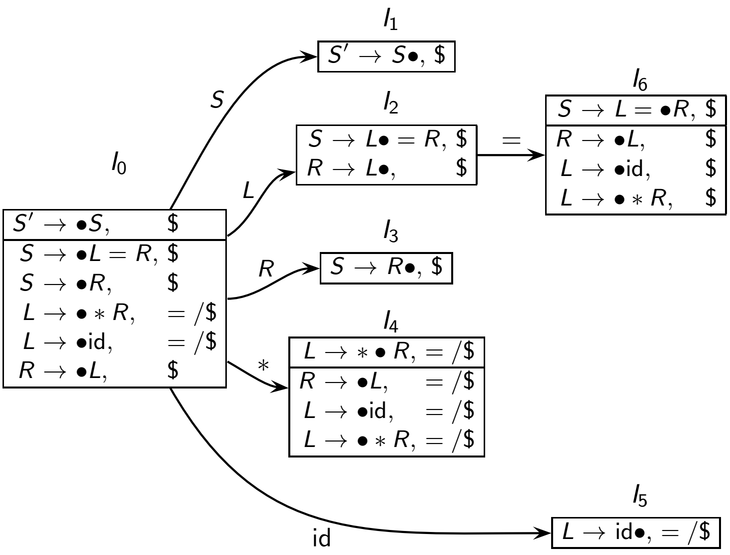
LR(1) Sets of Items for Pointer Assignment Grammar



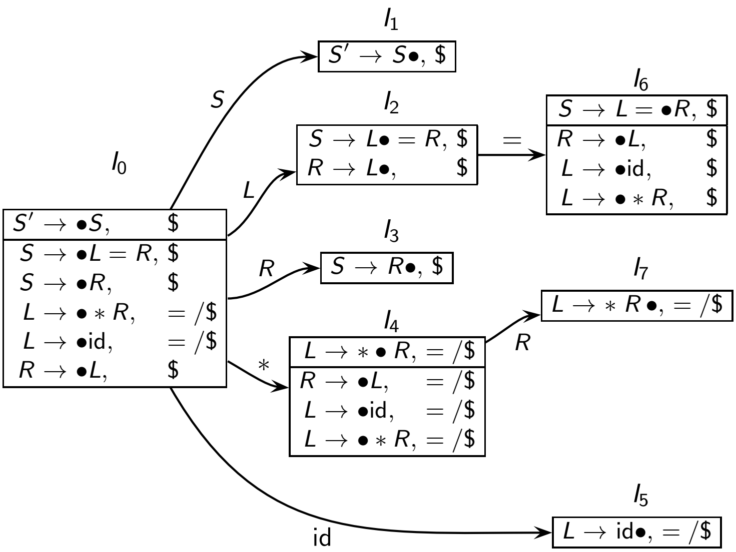
LR(1) Sets of Items for Pointer Assignment Grammar



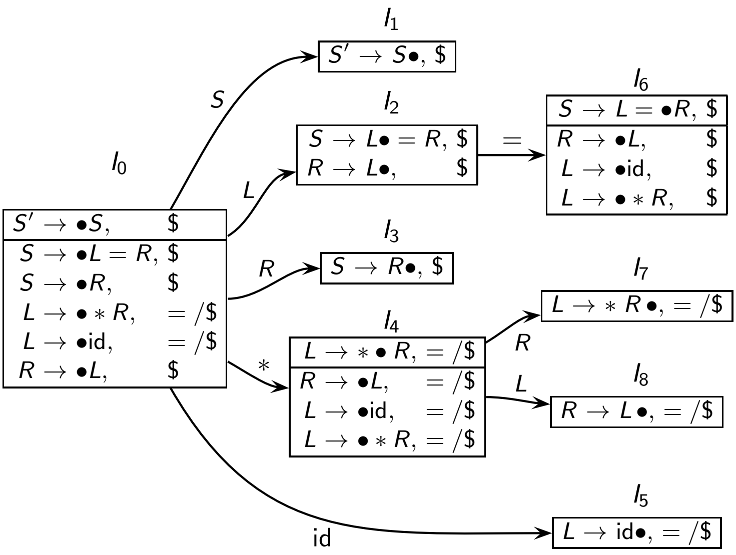
LR(1) Sets of Items for Pointer Assignment Grammar



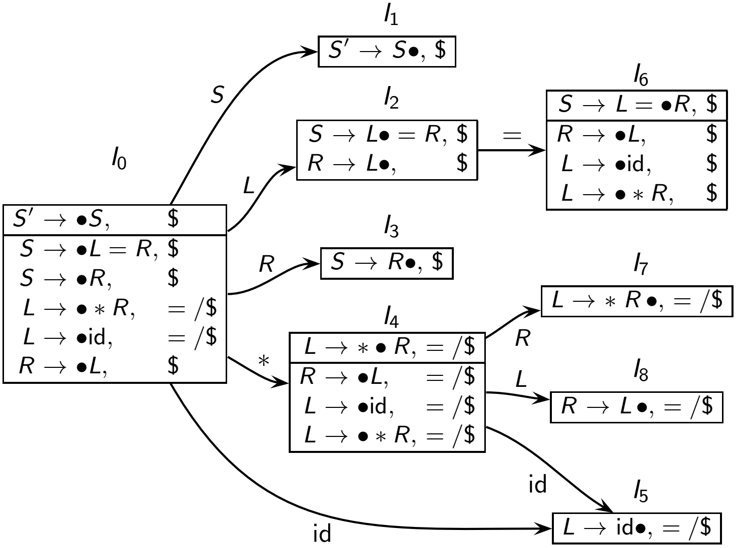
LR(1) Sets of Items for Pointer Assignment Grammar



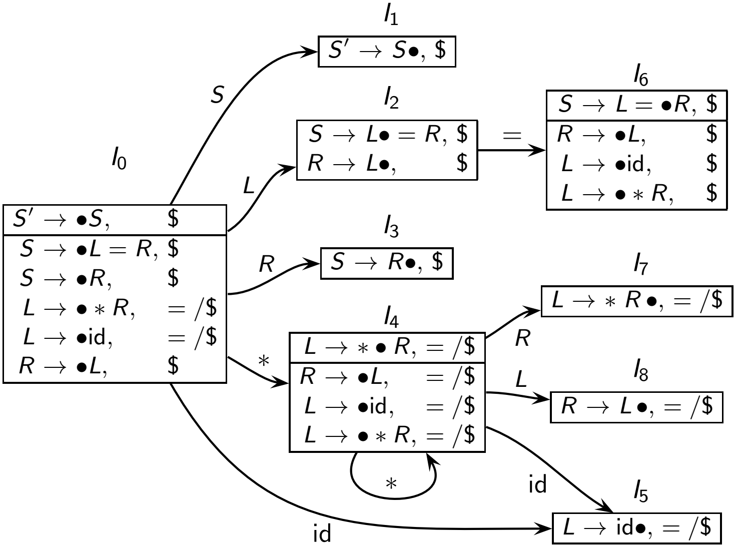
LR(1) Sets of Items for Pointer Assignment Grammar



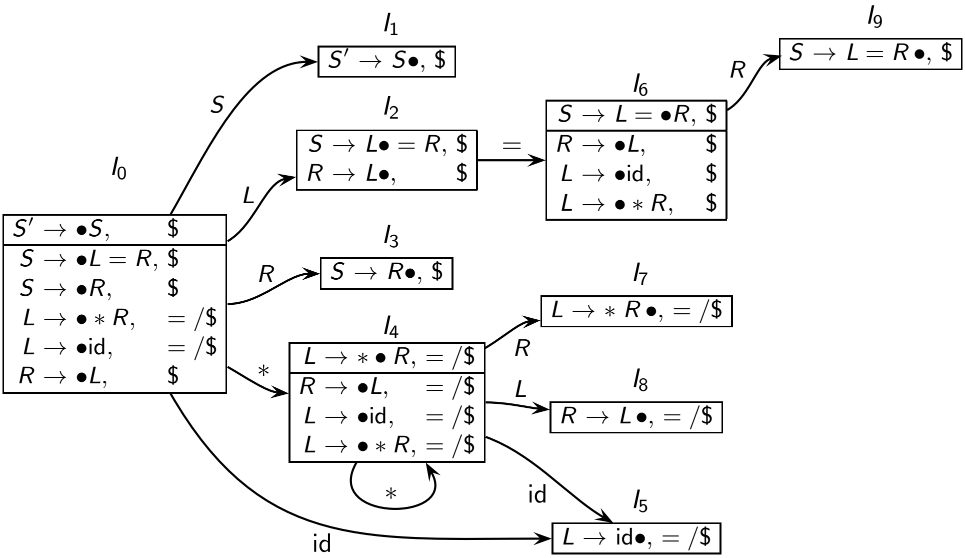
LR(1) Sets of Items for Pointer Assignment Grammar



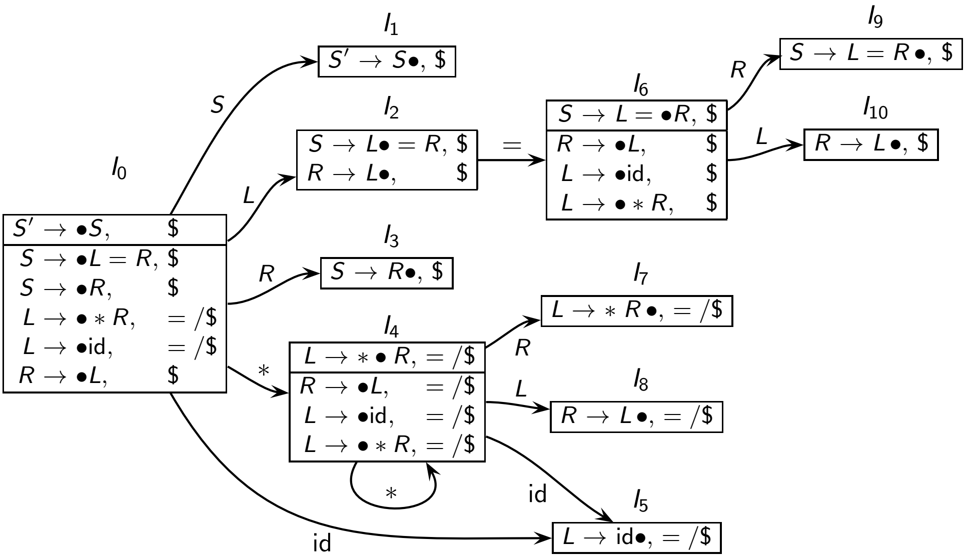
LR(1) Sets of Items for Pointer Assignment Grammar



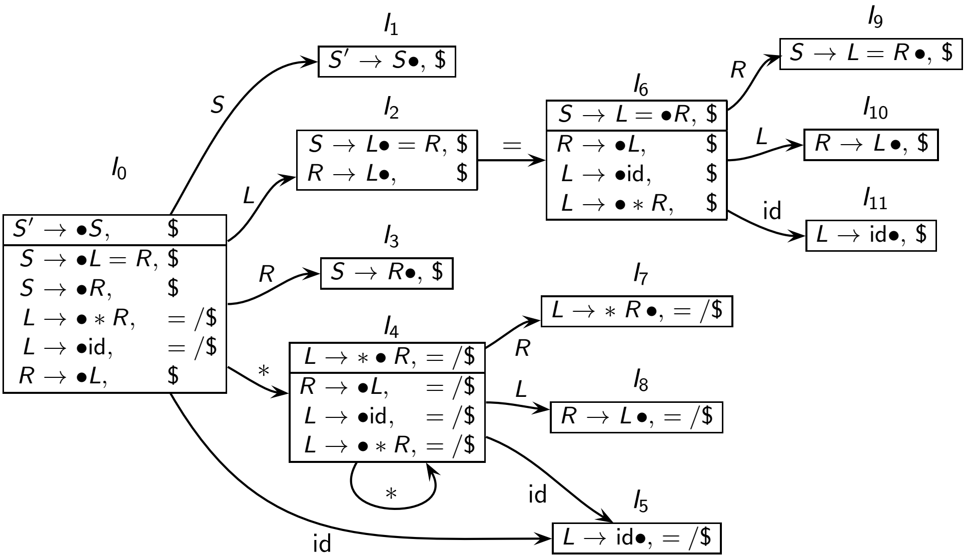
LR(1) Sets of Items for Pointer Assignment Grammar



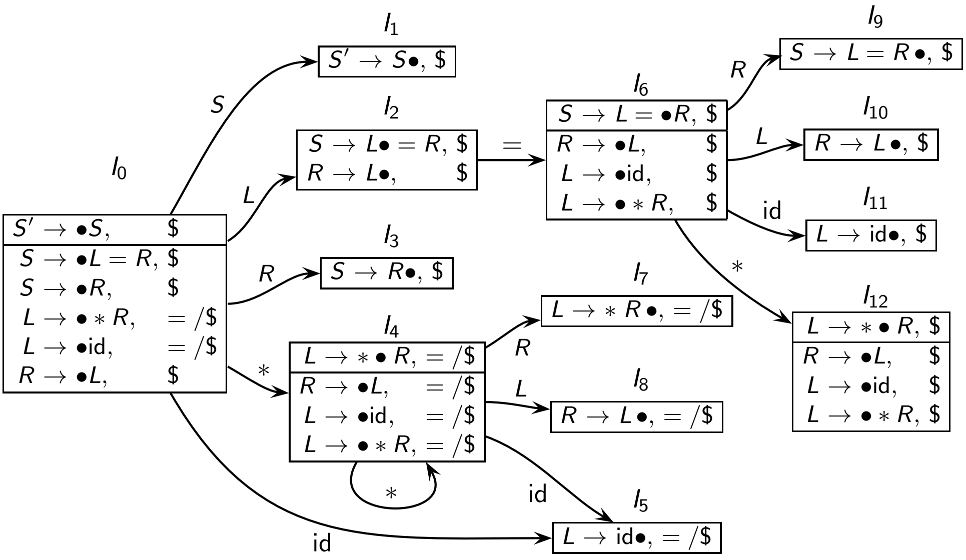
LR(1) Sets of Items for Pointer Assignment Grammar



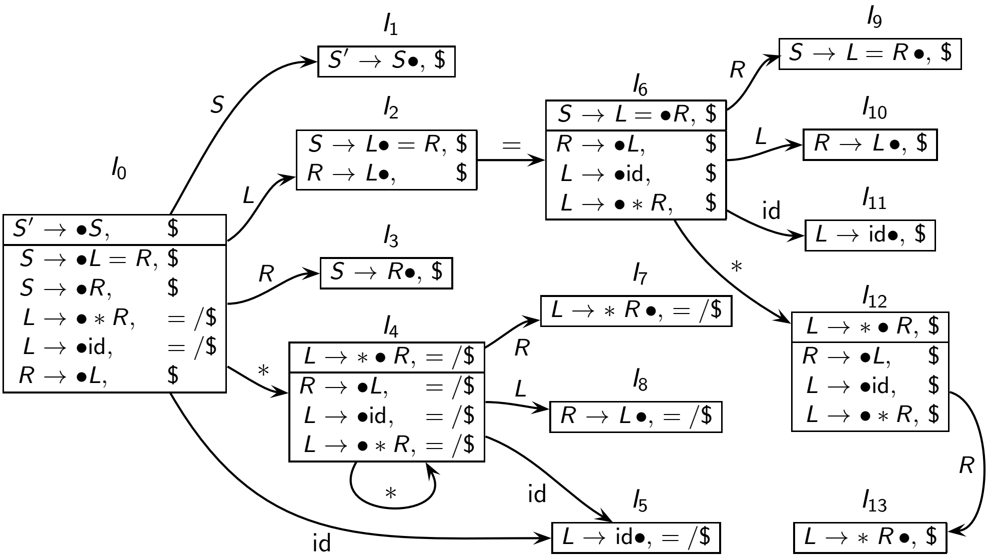
LR(1) Sets of Items for Pointer Assignment Grammar



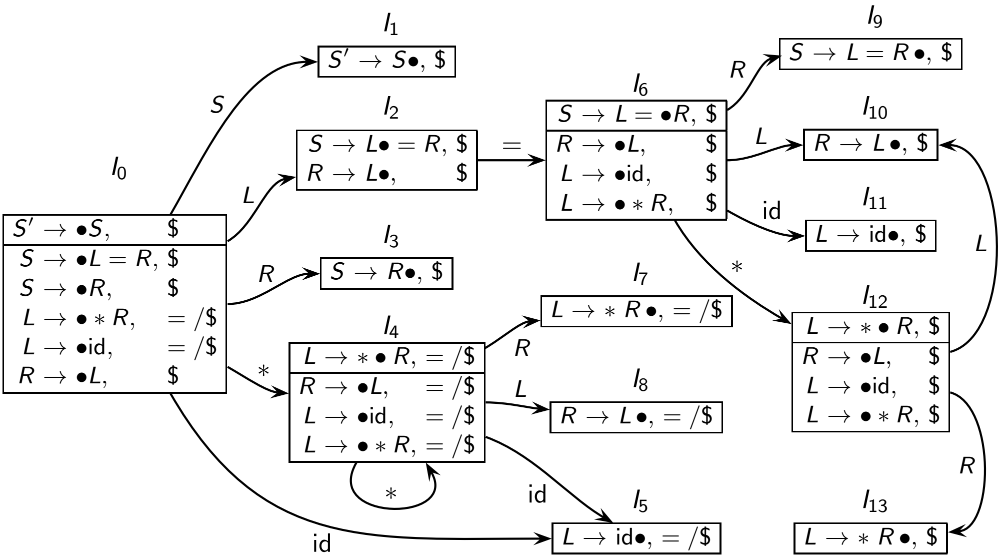
LR(1) Sets of Items for Pointer Assignment Grammar



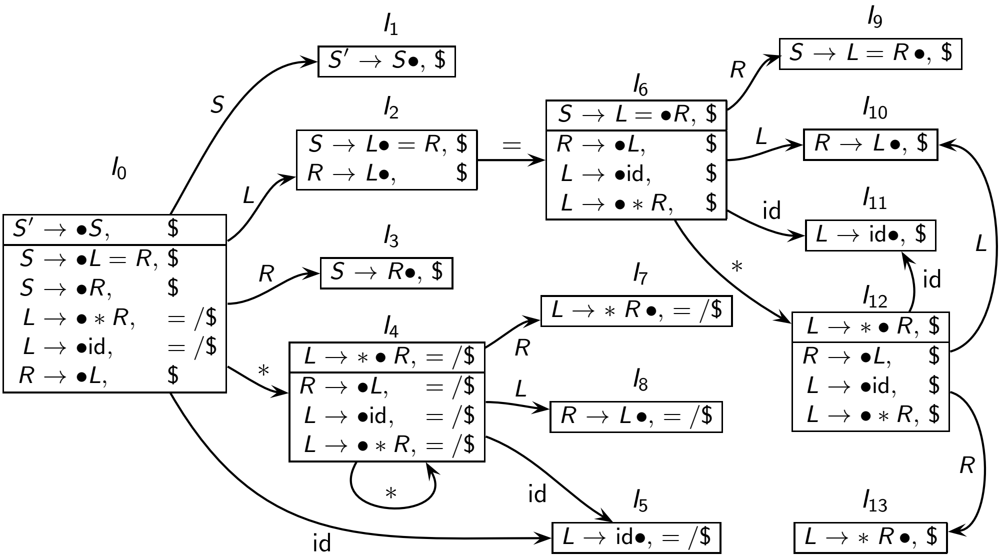
LR(1) Sets of Items for Pointer Assignment Grammar



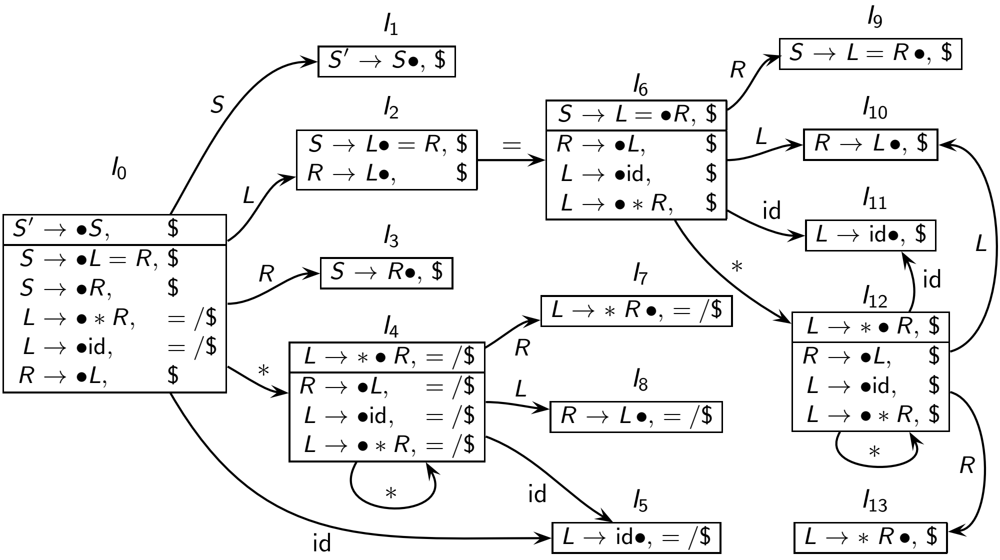
LR(1) Sets of Items for Pointer Assignment Grammar



LR(1) Sets of Items for Pointer Assignment Grammar



LR(1) Sets of Items for Pointer Assignment Grammar



LR(1) (aka CLR(1)) Parsing Table for Pointer Assignment Grammar

0	$S' \rightarrow S$
1	$S \rightarrow L = R$
2	$S \rightarrow R$
3	$L \rightarrow * R$
4	$L \rightarrow \text{id}$
5	$R \rightarrow L$

State	Action				Goto		
	id	*	=	$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s11$	$s12$				$c10$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			
10				$r5$			
11				$r4$			
12	$s11$	$s12$				c10	c13
13				$r3$			

LR(1) (aka CLR(1)) Parsing for the Pointer Assignment Grammar

State	Action				Goto		
	id	*	=	$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s11$	$s12$				$c10$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			
10				$r5$			
11				$r4$			
12	$s11$	$s12$				c10	c13
13				$r3$			

0 $S' \to S$
1 $S \to L = R$
2 $S \to R$
3 $L \to * R$
4 $L \to \text{id}$
5 $R \to L$

Input

0

Stack

LR(1) (aka CLR(1)) Parsing for the Pointer Assignment Grammar

State	Action				Goto		
	id	*	=	$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s11$	$s12$				$c10$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			
10				$r5$			
11				$r4$			
12	$s11$	$s12$				c10	c13
13				$r3$			

0 $S' \rightarrow S$
1 $S \rightarrow L = R$
2 $S \rightarrow R$
3 $L \rightarrow * R$
4 $L \rightarrow \text{id}$
5 $R \rightarrow L$

Input

id = id$

Shift 5

0

Stack

IIT Bombay
cs302: Implementation of Programming Languages

Topic:
Syntax Analysis

Section:

LR(1) (aka CLR(1)) Parsing for the Pointer Assignment Grammar

State	Action				Goto		
	id	*	=	$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s11$	$s12$				$c10$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			
10				$r5$			
11				$r4$			
12	$s11$	$s12$				c10	c13
13				$r3$			

0 $S' \rightarrow S$
1 $S \rightarrow L = R$
2 $S \rightarrow R$
3 $L \rightarrow * R$
4 $L \rightarrow \text{id}$
5 $R \rightarrow L$

Input

$= \text{id}\$$

Reduce by 4

5
id
0

Stack

LR(1) (aka CLR(1)) Parsing for the Pointer Assignment Grammar

State	Action				Goto		
	id	*	=	$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s11$	$s12$				$c10$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			
10				$r5$			
11				$r4$			
12	$s11$	$s12$				c10	c13
13				$r3$			

0	$S' \rightarrow S$
1	$S \rightarrow L = R$
2	$S \rightarrow R$
3	$L \rightarrow * R$
4	$L \rightarrow \text{id}$
5	$R \rightarrow L$

Input

$= \text{id}\$$

Cover by 2

L
0

Stack

LR(1) (aka CLR(1)) Parsing for the Pointer Assignment Grammar

State	Action				Goto		
	id	*	=	$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s11$	$s12$				$c10$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			
10				$r5$			
11				$r4$			
12	$s11$	$s12$				c10	c13
13				$r3$			

0 $S' \rightarrow S$
1 $S \rightarrow L = R$
2 $S \rightarrow R$
3 $L \rightarrow * R$
4 $L \rightarrow \text{id}$
5 $R \rightarrow L$

Input

$= \text{id}\$$

Shift 6

2
L
0

Stack

LR(1) (aka CLR(1)) Parsing for the Pointer Assignment Grammar

State	Action				Goto		
	id	*	=	$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s11$	$s12$				$c10$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			
10				$r5$			
11				$r4$			
12	$s11$	$s12$				c10	c13
13				$r3$			

0 $S' \rightarrow S$
1 $S \rightarrow L = R$
2 $S \rightarrow R$
3 $L \rightarrow * R$
4 $L \rightarrow \text{id}$
5 $R \rightarrow L$

Input

id$

Stack
6
=
2
L
0

Shift 11

Stack

LR(1) (aka CLR(1)) Parsing for the Pointer Assignment Grammar

State	Action				Goto		
	id	*	=	$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s11$	$s12$				$c10$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			
10				$r5$			
11				$r4$			
12	$s11$	$s12$				c10	c13
13				$r3$			

0 $S' \rightarrow S$
1 $S \rightarrow L = R$
2 $S \rightarrow R$
3 $L \rightarrow * R$
4 $L \rightarrow \text{id}$
5 $R \rightarrow L$

Input

$

11
id
6
=
2
L
0

Stack

Reduce by 4

LR(1) (aka CLR(1)) Parsing for the Pointer Assignment Grammar

State	Action				Goto		
	id	*	=	$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s11$	$s12$				$c10$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			
10				$r5$			
11				$r4$			
12	$s11$	$s12$				c10	c13
13				$r3$			

0 $S' \rightarrow S$
1 $S \rightarrow L = R$
2 $S \rightarrow R$
3 $L \rightarrow * R$
4 $L \rightarrow \text{id}$
5 $R \rightarrow L$

Input

$

Cover by 10

L
6
$=$
2
L
0

Stack

LR(1) (aka CLR(1)) Parsing for the Pointer Assignment Grammar

State	Action				Goto		
	id	*	=	$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s11$	$s12$				$c10$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			
10				$r5$			
11				$r4$			
12	$s11$	$s12$				c10	c13
13				$r3$			

0	$S' \rightarrow S$
1	$S \rightarrow L = R$
2	$S \rightarrow R$
3	$L \rightarrow * R$
4	$L \rightarrow \text{id}$
5	$R \rightarrow L$

Input

$

Reduce by 5

Stack
10
L
6
$=$
2
L
0

Stack

LR(1) (aka CLR(1)) Parsing for the Pointer Assignment Grammar

State	Action				Goto		
	id	*	=	$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s11$	$s12$				$c10$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			
10				$r5$			
11				$r4$			
12	$s11$	$s12$				c10	c13
13				$r3$			

0 $S' \rightarrow S$
1 $S \rightarrow L = R$
2 $S \rightarrow R$
3 $L \rightarrow * R$
4 $L \rightarrow \text{id}$
5 $R \rightarrow L$

Input

$

Cover by 9

Stack
R
6
$=$
2
L
0

Stack

LR(1) (aka CLR(1)) Parsing for the Pointer Assignment Grammar

State	Action				Goto		
	id	*	=	$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s11$	$s12$				$c10$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			
10				$r5$			
11				$r4$			
12	$s11$	$s12$				c10	c13
13				$r3$			

0 $S' \rightarrow S$
1 $S \rightarrow L = R$
2 $S \rightarrow R$
3 $L \rightarrow * R$
4 $L \rightarrow \text{id}$
5 $R \rightarrow L$

Input

$

9
R
6
$=$
2
L
0

Stack

Reduce by 1

LR(1) (aka CLR(1)) Parsing for the Pointer Assignment Grammar

State	Action				Goto		
	id	*	=	$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s11$	$s12$				$c10$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			
10				$r5$			
11				$r4$			
12	$s11$	$s12$				c10	c13
13				$r3$			

0	$S' \rightarrow S$
1	$S \rightarrow L = R$
2	$S \rightarrow R$
3	$L \rightarrow * R$
4	$L \rightarrow \text{id}$
5	$R \rightarrow L$

Input

$

Cover by 1

Stack
S
0

Stack

LR(1) (aka CLR(1)) Parsing for the Pointer Assignment Grammar

IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Syntax Analysis

Section:

Grammars, Derivations, and Parse Trees

Shift Reduce Parsing

SLR(1) Parsing

Conceptual Issues in Parsing

CLR(1) Parsing

LALR(1) Parsing

State	Action				Goto		
	id	*	=	$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s11$	$s12$				$c10$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			
10				$r5$			
11				$r4$			
12	$s11$	$s12$				c10	c13
13				$r3$			

0 $S' \rightarrow S$
1 $S \rightarrow L = R$
2 $S \rightarrow R$
3 $L \rightarrow * R$
4 $L \rightarrow \text{id}$
5 $R \rightarrow L$

Input

$

Accept

1
S
0

Stack

Another Example of LR(1) (aka CLR(1)) Parsing

$$
\begin{aligned}
A &\rightarrow aBe \\
A &\rightarrow aCd \\
A &\rightarrow bBd \\
A &\rightarrow bCe \\
B &\rightarrow f \\
C &\rightarrow f
\end{aligned}
$$

Another Example of LR(1) (aka CLR(1)) Parsing

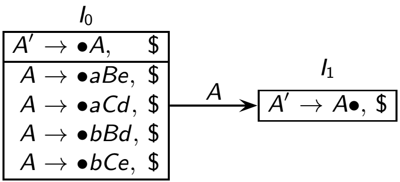
$$A \rightarrow aBe$$
$$A \rightarrow aCd$$
$$A \rightarrow bBd$$
$$A \rightarrow bCe$$
$$B \rightarrow f$$
$$C \rightarrow f$$

I_0

$A' \rightarrow \bullet A,$	$\$$
$A \rightarrow \bullet aBe,$	$\$$
$A \rightarrow \bullet aCd,$	$\$$
$A \rightarrow \bullet bBd,$	$\$$
$A \rightarrow \bullet bCe,$	$\$$

Another Example of LR(1) (aka CLR(1)) Parsing

$$
\begin{aligned}
A &\rightarrow aBe \\
A &\rightarrow aCd \\
A &\rightarrow bBd \\
A &\rightarrow bCe \\
B &\rightarrow f \\
C &\rightarrow f
\end{aligned}
$$



Another Example of LR(1) (aka CLR(1)) Parsing

$$
\begin{aligned}
A &\rightarrow aBe\\
A &\rightarrow aCd\\
A &\rightarrow bBd\\
A &\rightarrow bCe\\
B &\rightarrow f\\
C &\rightarrow f
\end{aligned}
$$

I_0:

$$
\begin{array}{|ll|}
\hline
A' \rightarrow \bullet A, & \$\\
\hline
A \rightarrow \bullet aBe, & \$\\
A \rightarrow \bullet aCd, & \$\\
A \rightarrow \bullet bBd, & \$\\
A \rightarrow \bullet bCe, & \$\\
\hline
\end{array}
$$

I_1:

$$
\begin{array}{|l|}
\hline
A' \rightarrow A\bullet,\ \$\\
\hline
\end{array}
$$

I_2:

$$
\boxed{
\begin{array}{ll}
A \rightarrow a \bullet Be, & \$\\
A \rightarrow a \bullet Cd, & \$\\
\hline
B \rightarrow \bullet f, & e\\
C \rightarrow \bullet f, & d
\end{array}}
$$

$$I_0 \xrightarrow{a} I_2 \qquad I_0 \xrightarrow{A} I_1$$

Closure of $\boxed{P \rightarrow \alpha \bullet Q\beta, p}$ contains items of the form $\boxed{Q \rightarrow \bullet \gamma,\ \text{FIRST}(\beta p)}$

In our example

- For $Q = B$, β is e and p is \$

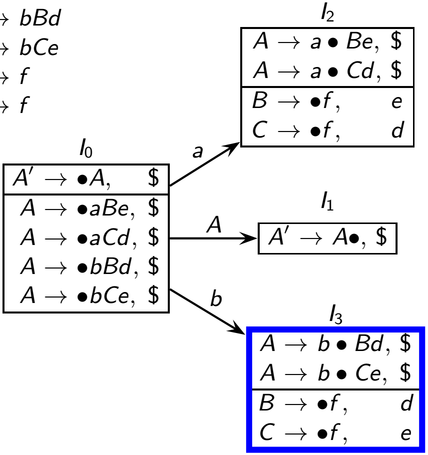
  If we expect to see a string derivable from B in this state, the string must be followed by $\text{FIRST}(\beta p) = \text{FIRST}(e\$) = e$

- For $Q = C$, β is d and p is \$

  If we expect to see a string derivable from C in this state, the string must be followed by $\text{FIRST}(\beta p) = \text{FIRST}(d\$) = d$

Another Example of LR(1) (aka CLR(1)) Parsing

$$
\begin{aligned}
A &\to aBe\\
A &\to aCd\\
A &\to bBd\\
A &\to bCe\\
B &\to f\\
C &\to f
\end{aligned}
$$



Closure of $\boxed{P \to \alpha \bullet Q\beta, p}$ contains items of the form $\boxed{Q \to \bullet\, \gamma, \ \text{FIRST}(\beta p)}$

In our example

- For $Q = B$, β is d and p is \$

  If we expect to see a string derivable from B in this state, the string must be followed by
  $\text{FIRST}(\beta p) = \text{FIRST}(d\$) = d$
- For $Q = C$, β is e and p is \$

  If we expect to see a string derivable from C in this state, the string must be followed by
  $\text{FIRST}(\beta p) = \text{FIRST}(e\$) = e$

Another Example of LR(1) (aka CLR(1)) Parsing

$$
\begin{aligned}
A &\rightarrow aBe\\
A &\rightarrow aCd\\
A &\rightarrow bBd\\
A &\rightarrow bCe\\
B &\rightarrow f\\
C &\rightarrow f
\end{aligned}
$$

I_0

$$
\begin{array}{|lll|}
\hline
A' \rightarrow & \bullet A, & \$ \\
\hline
A \rightarrow & \bullet aBe, & \$ \\
A \rightarrow & \bullet aCd, & \$ \\
A \rightarrow & \bullet bBd, & \$ \\
A \rightarrow & \bullet bCe, & \$ \\
\hline
\end{array}
$$

I_1

$$
\begin{array}{|lll|}
\hline
A' \rightarrow & A\bullet, & \$ \\
\hline
\end{array}
$$

I_2

$$
\begin{array}{|lll|}
\hline
A \rightarrow & a \bullet Be, & \$ \\
A \rightarrow & a \bullet Cd, & \$ \\
\hline
B \rightarrow & \bullet f, & e \\
C \rightarrow & \bullet f, & d \\
\hline
\end{array}
$$

I_3

$$
\begin{array}{|lll|}
\hline
A \rightarrow & b \bullet Bd, & \$ \\
A \rightarrow & b \bullet Ce, & \$ \\
\hline
B \rightarrow & \bullet f, & d \\
C \rightarrow & \bullet f, & e \\
\hline
\end{array}
$$

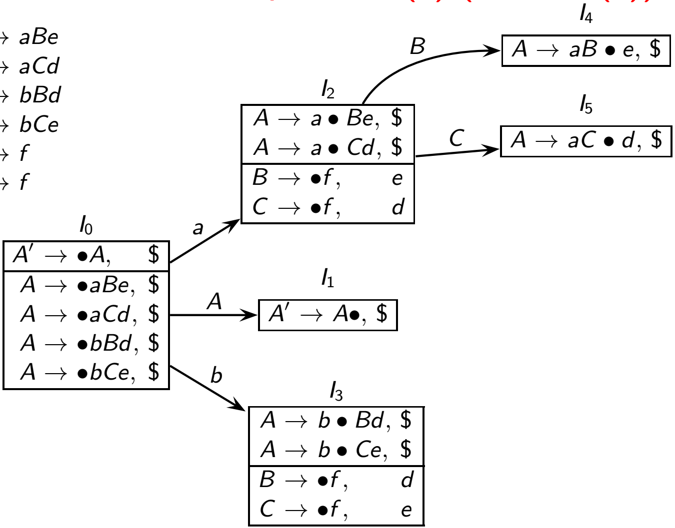
I_4

$$
\begin{array}{|lll|}
\hline
A \rightarrow & aB \bullet e, & \$ \\
\hline
\end{array}
$$

$I_0 \xrightarrow{a} I_2$, $I_0 \xrightarrow{A} I_1$, $I_0 \xrightarrow{b} I_3$, $I_2 \xrightarrow{B} I_4$

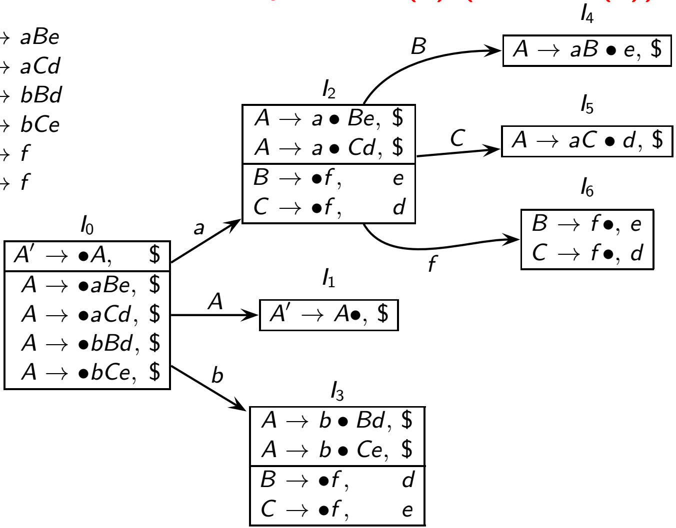
Another Example of LR(1) (aka CLR(1)) Parsing

$$
\begin{aligned}
A &\rightarrow aBe\\
A &\rightarrow aCd\\
A &\rightarrow bBd\\
A &\rightarrow bCe\\
B &\rightarrow f\\
C &\rightarrow f
\end{aligned}
$$



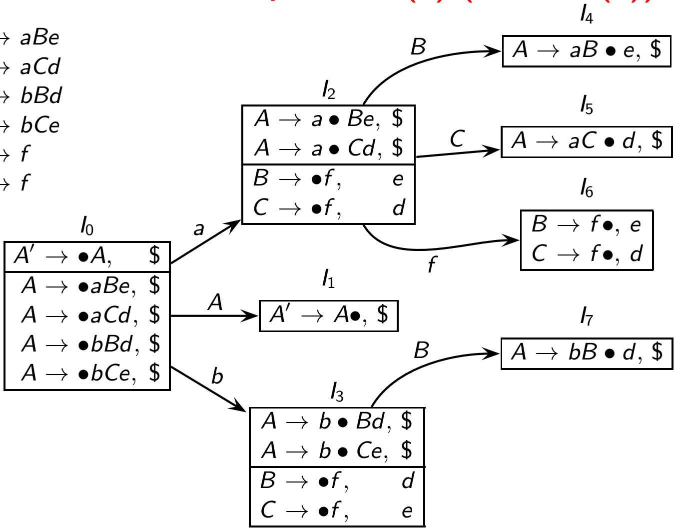
Another Example of LR(1) (aka CLR(1)) Parsing

$A \rightarrow aBe$
$A \rightarrow aCd$
$A \rightarrow bBd$
$A \rightarrow bCe$
$B \rightarrow f$
$C \rightarrow f$



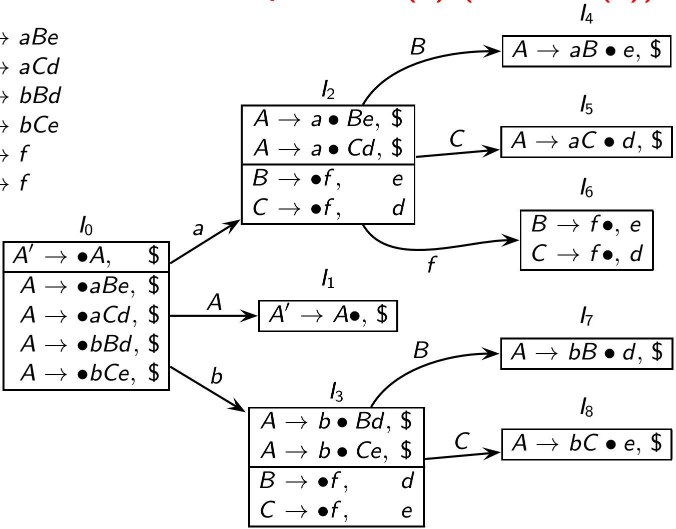
Another Example of LR(1) (aka CLR(1)) Parsing

$$A \rightarrow aBe$$
$$A \rightarrow aCd$$
$$A \rightarrow bBd$$
$$A \rightarrow bCe$$
$$B \rightarrow f$$
$$C \rightarrow f$$



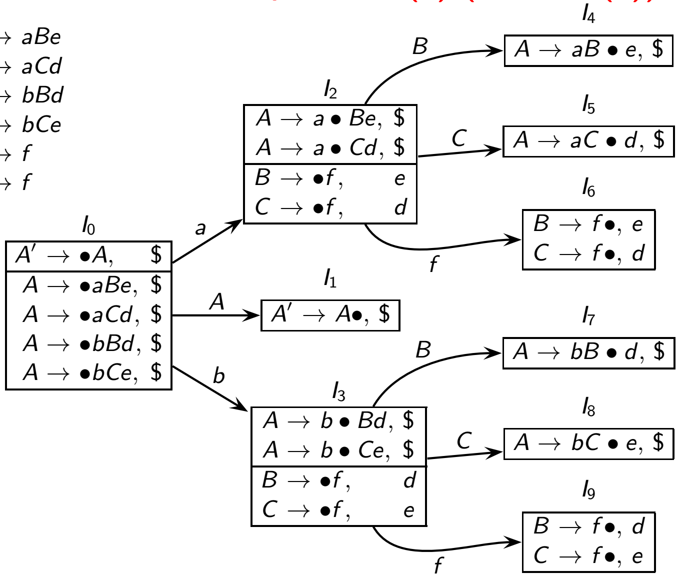
Another Example of LR(1) (aka CLR(1)) Parsing

$A \rightarrow aBe$
$A \rightarrow aCd$
$A \rightarrow bBd$
$A \rightarrow bCe$
$B \rightarrow f$
$C \rightarrow f$



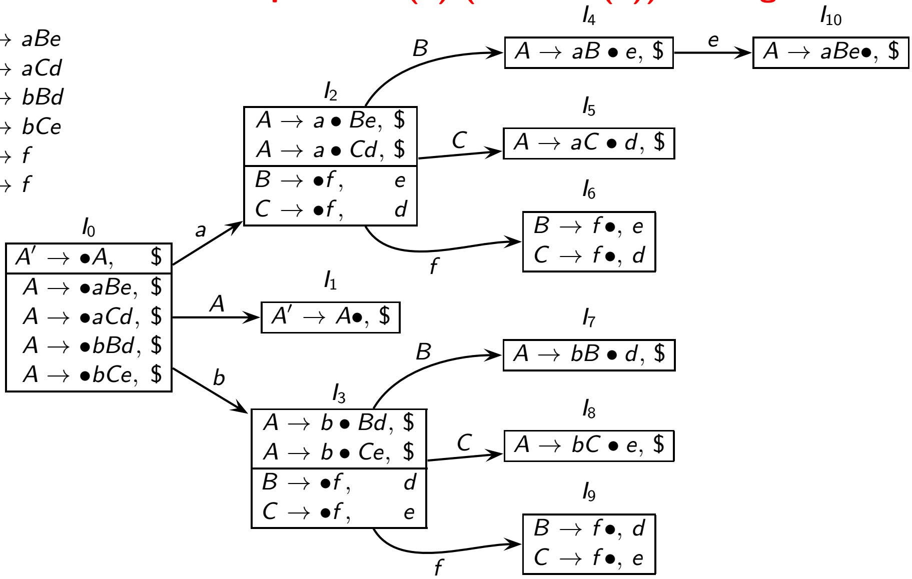
Another Example of LR(1) (aka CLR(1)) Parsing

$$
\begin{aligned}
A &\rightarrow aBe\\
A &\rightarrow aCd\\
A &\rightarrow bBd\\
A &\rightarrow bCe\\
B &\rightarrow f\\
C &\rightarrow f
\end{aligned}
$$



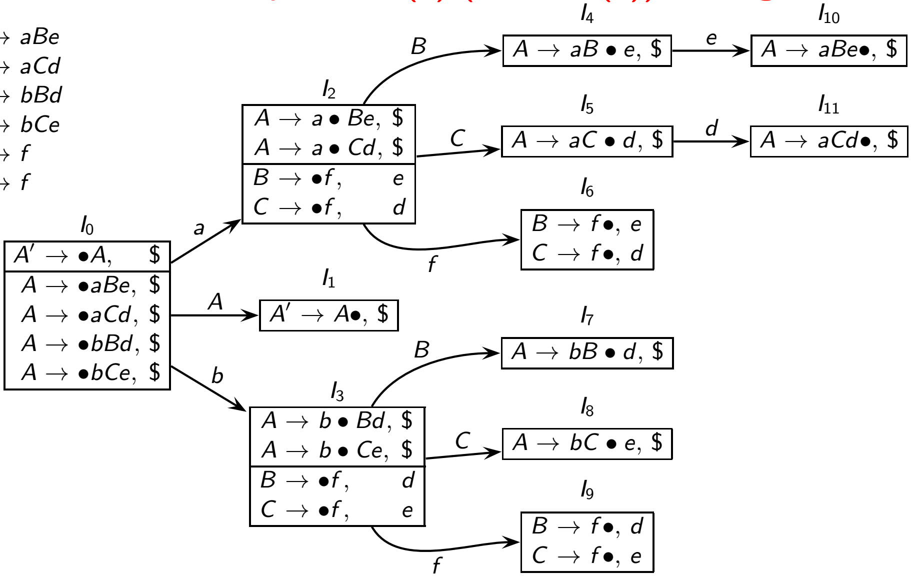
Another Example of LR(1) (aka CLR(1)) Parsing

$A \rightarrow aBe$
$A \rightarrow aCd$
$A \rightarrow bBd$
$A \rightarrow bCe$
$B \rightarrow f$
$C \rightarrow f$



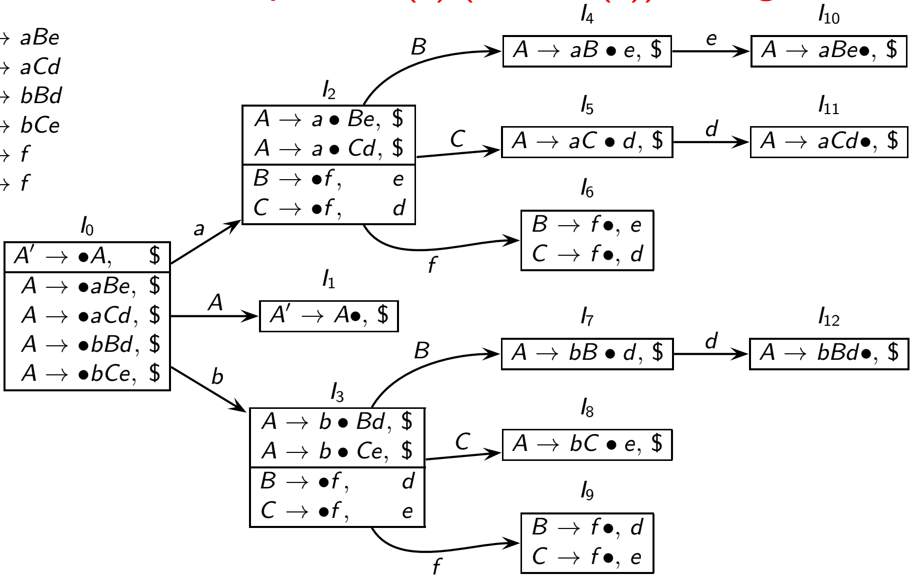
Another Example of LR(1) (aka CLR(1)) Parsing

$$
\begin{aligned}
A &\rightarrow aBe\\
A &\rightarrow aCd\\
A &\rightarrow bBd\\
A &\rightarrow bCe\\
B &\rightarrow f\\
C &\rightarrow f
\end{aligned}
$$



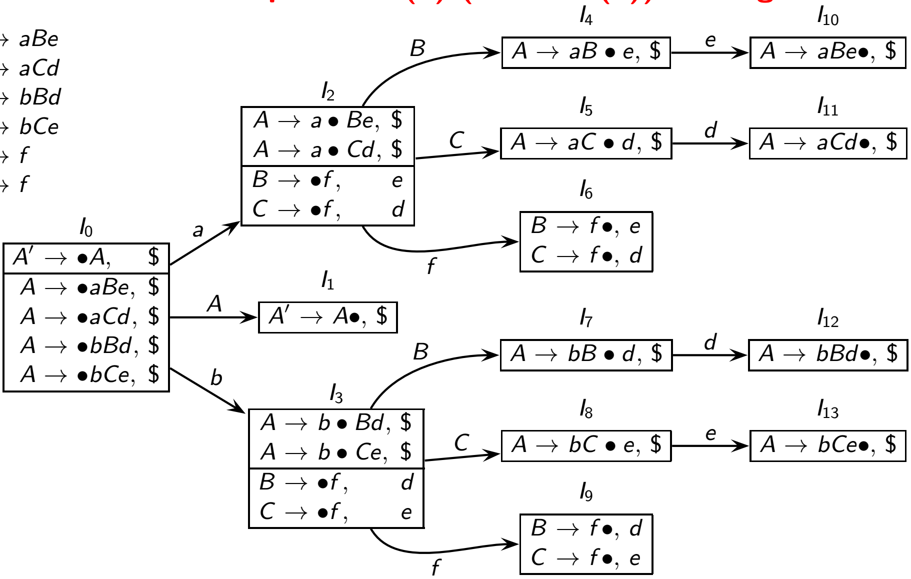
Another Example of LR(1) (aka CLR(1)) Parsing

$A \rightarrow aBe$
$A \rightarrow aCd$
$A \rightarrow bBd$
$A \rightarrow bCe$
$B \rightarrow f$
$C \rightarrow f$



Another Example of LR(1) (aka CLR(1)) Parsing

$$
\begin{aligned}
A &\rightarrow aBe\\
A &\rightarrow aCd\\
A &\rightarrow bBd\\
A &\rightarrow bCe\\
B &\rightarrow f\\
C &\rightarrow f
\end{aligned}
$$



LALR(1) Parsing

LALR(1) Parsing

- Merge item sets with identical cores (may have different lookaheads)

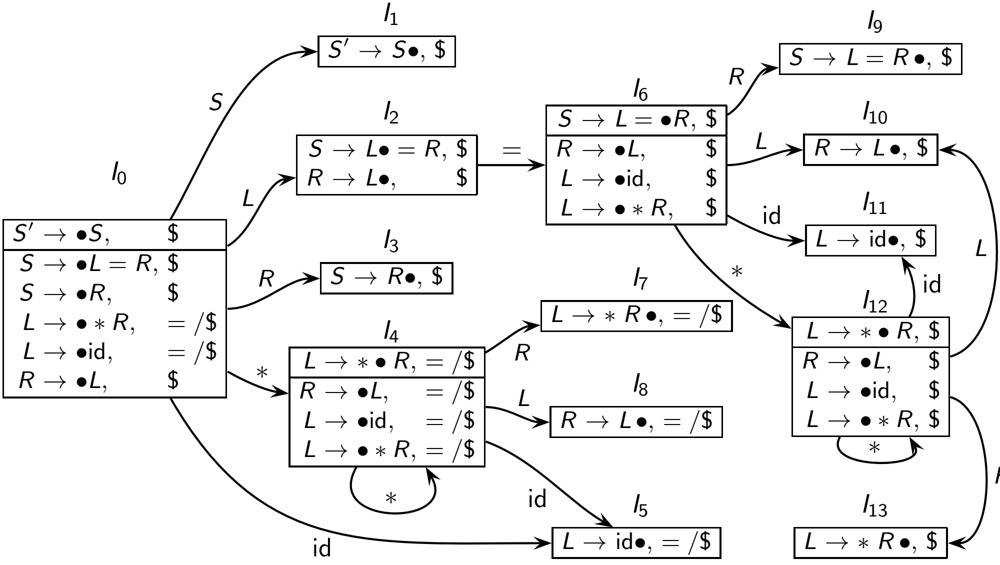
  States I_i : $A \rightarrow \alpha \bullet \beta, a$ and I_j : $A \rightarrow \alpha \bullet \beta, b$

  can be merged to create a new state I_{ij} : $A \rightarrow \alpha \bullet \beta, a/b$

- In practice, we do not construct LR(1) items to construct LALR(1) parser

  We construct LR(0) items and use a look-ahead propagation algorithm

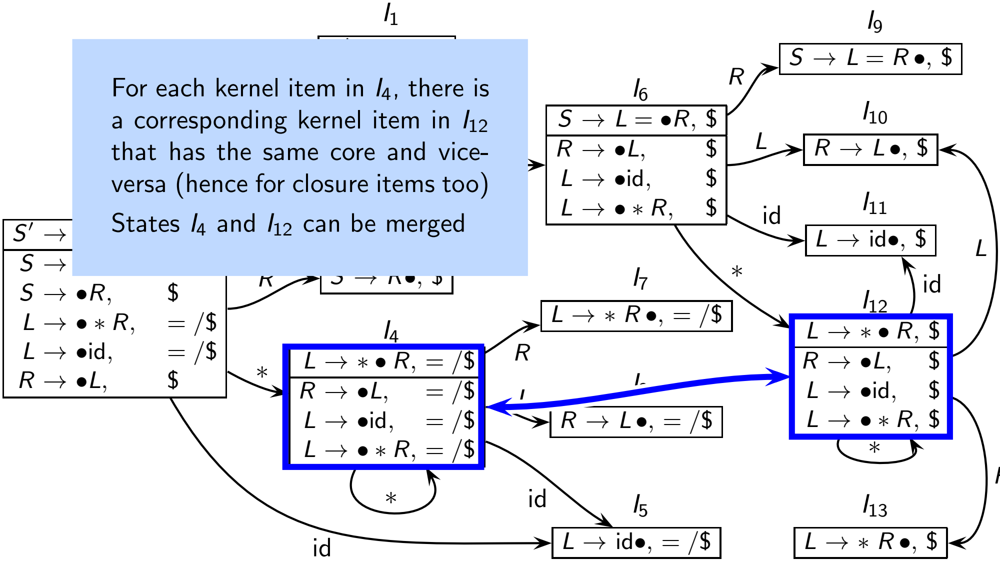
LALR(1) Parsing for Pointer Assignment Grammar



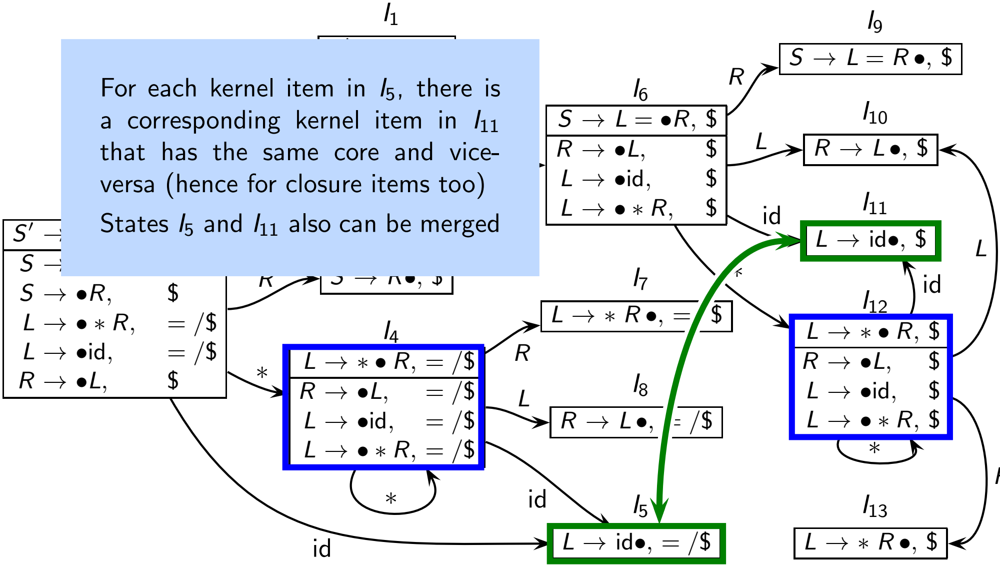
LALR(1) Parsing for Pointer Assignment Grammar

For each kernel item in I_4, there is a corresponding kernel item in I_{12} that has the same core and vice-versa (hence for closure items too)

States I_4 and I_{12} can be merged



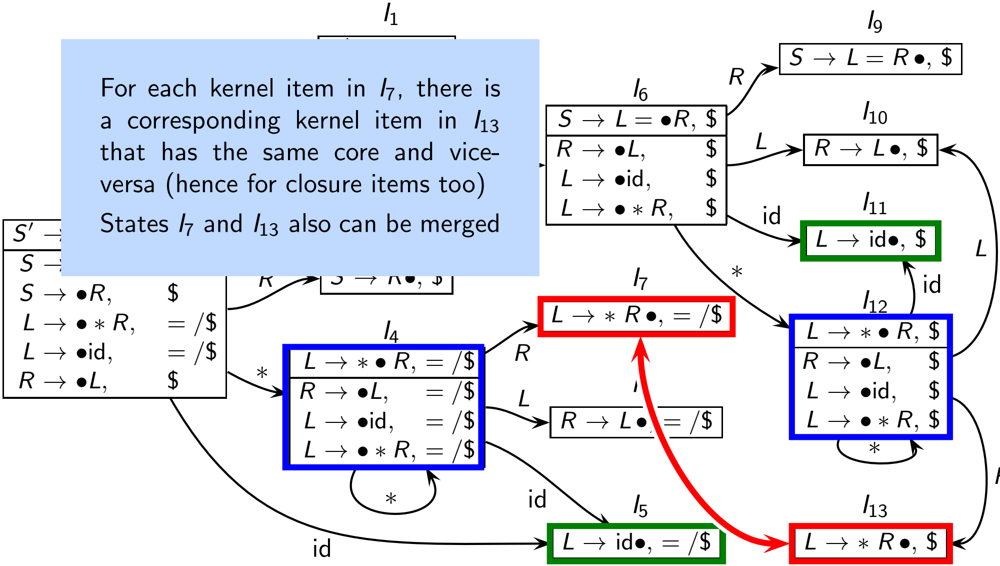
LALR(1) Parsing for Pointer Assignment Grammar



For each kernel item in I_5, there is a corresponding kernel item in I_{11} that has the same core and vice-versa (hence for closure items too)

States I_5 and I_{11} also can be merged

LALR(1) Parsing for Pointer Assignment Grammar



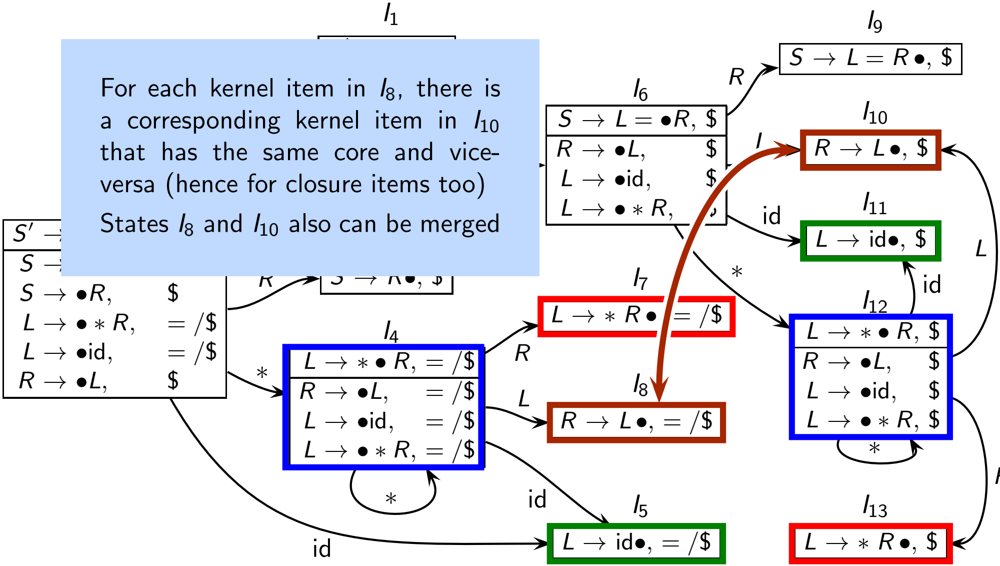
For each kernel item in I_7, there is a corresponding kernel item in I_{13} that has the same core and vice-versa (hence for closure items too)

States I_7 and I_{13} also can be merged

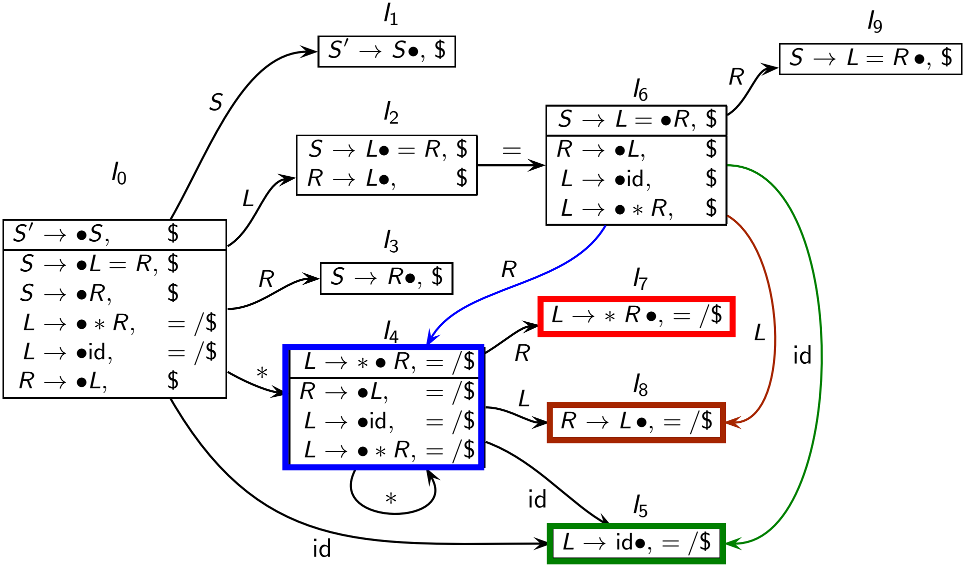
LALR(1) Parsing for Pointer Assignment Grammar

For each kernel item in I_8, there is a corresponding kernel item in I_{10} that has the same core and vice-versa (hence for closure items too)

States I_8 and I_{10} also can be merged



LALR(1) Parsing for Pointer Assignment Grammar



LALR(1) Parsing Table for Pointer Assignment Grammar

0	$S' \rightarrow S$
1	$S \rightarrow L = R$
2	$S \rightarrow R$
3	$L \rightarrow * R$
4	$L \rightarrow \text{id}$
5	$R \rightarrow L$

State	Action				Goto		
	id	*	=	\$	S	L	R
0	$s5$	$s4$			$c1$	$c2$	$c3$
1				acc			
2			$s6$	$r5$			
3				$r2$			
4	$s5$	$s4$				$c8$	$c7$
5			$r4$	$r4$			
6	$s5$	$s4$				$c8$	$c9$
7			$r3$	$r3$			
8			$r5$	$r5$			
9				$r1$			

LALR(1) Vs CLR(1) Parsing

- Can merging of LR(1) states introduce shift-reduce conflict?
- Can merging of LR(1) states introduce reduce-reduce conflict?

Can Merging LR(1) Sets of Items Introduce Shift-Reduce Conflict?

- To merge states I_i and I_j, they should have identical cores but different lookaheads (if the lookaheads are same then the states will not be distinct)

Can Merging LR(1) Sets of Items Introduce Shift-Reduce Conflict?

- To merge states I_i and I_j, they should have identical cores but different lookaheads (if the lookaheads are same then the states will not be distinct)
- Let I_i : $\begin{array}{ll} A \rightarrow \alpha \bullet a\beta, & p \\ B \rightarrow \gamma\bullet, & q \end{array}$ and I_j : $\begin{array}{ll} A \rightarrow \alpha \bullet a\beta, & r \\ B \rightarrow \gamma\bullet, & s \end{array}$ where p, q, r, s are arbitrary terminals

  So that the merged state is I_{ij} : $\begin{array}{ll} A \rightarrow \alpha \bullet a\beta, & p/r \\ B \rightarrow \gamma\bullet, & q/s \end{array}$

Can Merging LR(1) Sets of Items Introduce Shift-Reduce Conflict?

- To merge states I_i and I_j, they should have identical cores but different lookaheads (if the lookaheads are same then the states will not be distinct)
- Let I_i : $\begin{array}{ll} A \to \alpha \bullet a\beta, & p \\ B \to \gamma\bullet, & q \end{array}$ and I_j : $\begin{array}{ll} A \to \alpha \bullet a\beta, & r \\ B \to \gamma\bullet, & s \end{array}$ where p, q, r, s are arbitrary terminals

  So that the merged state is I_{ij} : $\begin{array}{ll} A \to \alpha \bullet a\beta, & p/r \\ B \to \gamma\bullet, & q/s \end{array}$
- For a shift-reduce conflict in I_{ij}, either q or s must be a.

Can Merging LR(1) Sets of Items Introduce Shift-Reduce Conflict?

- To merge states I_i and I_j, they should have identical cores but different lookaheads (if the lookaheads are same then the states will not be distinct)
- Let I_i : $\begin{array}{ll} A \rightarrow \alpha \bullet a\beta, & p \\ B \rightarrow \gamma\bullet, & q \end{array}$ and I_j : $\begin{array}{ll} A \rightarrow \alpha \bullet a\beta, & r \\ B \rightarrow \gamma\bullet, & s \end{array}$ where p, q, r, s are arbitrary terminals

  So that the merged state is I_{ij} : $\begin{array}{ll} A \rightarrow \alpha \bullet a\beta, & p/r \\ B \rightarrow \gamma\bullet, & q/s \end{array}$
- For a shift-reduce conflict in I_{ij}, either q or s must be a.
  - If q is a, then I_i is $\begin{array}{ll} A \rightarrow \alpha \bullet a\beta, & p \\ B \rightarrow \gamma\bullet, & a \end{array}$ and thus I_i has a shift-reduce conflict

Can Merging LR(1) Sets of Items Introduce Shift-Reduce Conflict?

- To merge states I_i and I_j, they should have identical cores but different lookaheads (if the lookaheads are same then the states will not be distinct)
- Let I_i : $\begin{array}{ll} A \rightarrow \alpha \bullet a\beta, & p \\ B \rightarrow \gamma \bullet, & q \end{array}$ and I_j : $\begin{array}{ll} A \rightarrow \alpha \bullet a\beta, & r \\ B \rightarrow \gamma \bullet, & s \end{array}$ where p, q, r, s are arbitrary terminals

  So that the merged state is I_{ij} : $\begin{array}{ll} A \rightarrow \alpha \bullet a\beta, & p/r \\ B \rightarrow \gamma \bullet, & q/s \end{array}$
- For a shift-reduce conflict in I_{ij}, either q or s must be a.
  - If q is a, then I_i is $\begin{array}{ll} A \rightarrow \alpha \bullet a\beta, & p \\ B \rightarrow \gamma \bullet, & a \end{array}$ and thus I_i has a shift-reduce conflict
  - If s is a, then I_j is $\begin{array}{ll} A \rightarrow \alpha \bullet a\beta, & r \\ B \rightarrow \gamma \bullet, & a \end{array}$ and thus I_j has a shift-reduce conflict

Can Merging LR(1) Sets of Items Introduce Shift-Reduce Conflict?

- To merge states I_i and I_j, they should have identical cores but different lookaheads (if the lookaheads are same then the states will not be distinct)
- Let I_i : $A \rightarrow$ … $B \rightarrow$ … e
  arbitrary term…

  So that the …
- For a shift-re…
  - If q is a, … $B \rightarrow \gamma\bullet, \quad a$ … e conflict
  - If s is a, then I_j is $\begin{array}{ll} A \rightarrow \alpha \bullet a\beta, & r \\ B \rightarrow \gamma\bullet, & a \end{array}$ and thus I_j has a shift-reduce conflict

A set I_{ij} of items in an LALR(1) parser can have a shift-reduce conflict *if and only if* a set I_i of LR(1) items merged to form I_{ij} has the same shift-reduce conflict

Can Merging LR(1) Sets of Items Introduce Reduce-Reduce Conflict?

- Let I_i : $\begin{array}{ll} A \to \alpha\bullet, & p \\ B \to \alpha\bullet, & q \end{array}$ and I_j : $\begin{array}{ll} A \to \alpha\bullet, & r \\ B \to \alpha\bullet, & s \end{array}$

  So that the merged state is I_{ij} : $\begin{array}{ll} A \to \alpha\bullet, & p/r \\ B \to \alpha\bullet, & q/s \end{array}$

Can Merging LR(1) Sets of Items Introduce Reduce-Reduce Conflict?

- Let I_i : $\begin{array}{ll} A \rightarrow \alpha\bullet, & p \\ B \rightarrow \alpha\bullet, & q \end{array}$ and I_j : $\begin{array}{ll} A \rightarrow \alpha\bullet, & r \\ B \rightarrow \alpha\bullet, & s \end{array}$

  So that the merged state is I_{ij} : $\begin{array}{ll} A \rightarrow \alpha\bullet, & p/r \\ B \rightarrow \alpha\bullet, & q/s \end{array}$

- For a reduce-reduce conflict in I_{ij} such that there is no reduce-reduce conflict in I_i or I_j,
  - $p = s$. This is possible without a reduce-reduce conflict in I_i and I_j
  - $r = q$. This is also possible without a reduce-reduce conflict in I_i and I_j

Can Merging LR(1) Sets of Items Introduce Reduce-Reduce Conflict?

- Let I_i : $\begin{array}{ll} A \rightarrow \alpha\bullet, & p \\ B \rightarrow \alpha\bullet & q \end{array}$ and I_j : $\begin{array}{ll} A \rightarrow \alpha\bullet, & r \\ B \rightarrow \alpha\bullet & s \end{array}$

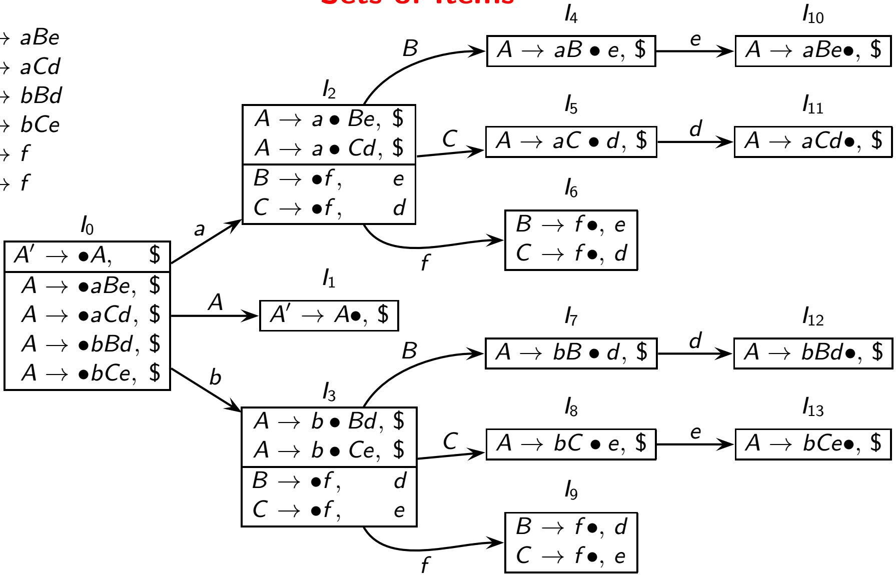
  So that the r

- For a reduce- nflict in I_i or I_j,
  - $p = s$.
  - $r = q$. This is also possible without a reduce-reduce conflict in I_i and I_j

Merging LR(1) sets of items can introduce reduce-reduce conflicts *even* if the original sets do not have a reduce-reduce conflict

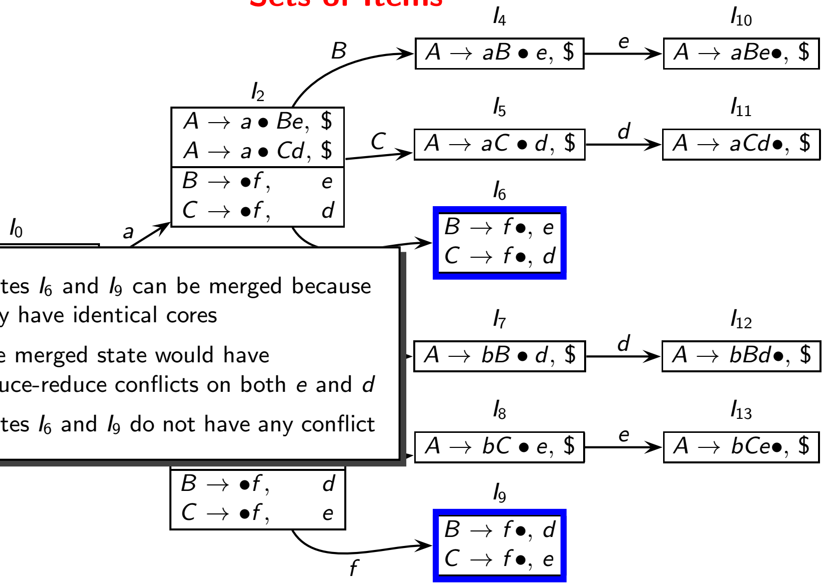
Example of Reduce-Reduce Conflict Caused by Merging LR(1) Sets of Items

$A \rightarrow aBe$
$A \rightarrow aCd$
$A \rightarrow bBd$
$A \rightarrow bCe$
$B \rightarrow f$
$C \rightarrow f$



Example of Reduce-Reduce Conflict Caused by Merging LR(1) Sets of Items

$A \rightarrow aBe$
$A \rightarrow aCd$
$A \rightarrow bBd$
$A \rightarrow bCe$
$B \rightarrow f$
$C \rightarrow f$



- States I_6 and I_9 can be merged because they have identical cores
- The merged state would have reduce-reduce conflicts on both e and d
- States I_6 and I_9 do not have any conflict

A Practical Example of Reduce-Reduce Conflict in LR(1) Parsing

$$
\begin{array}{rcl}
\text{program} & \rightarrow & \text{func\_decl var\_decl} \\
\text{program} & \rightarrow & \text{var\_decl func\_decl} \\
\text{var\_decl} & \rightarrow & \text{data\_type ID ;} \\
\text{data\_type} & \rightarrow & \text{INT} \\
\text{func\_decl} & \rightarrow & \text{return\_type ID ( )} \\
\text{return\_type} & \rightarrow & \text{INT} \\
\text{return\_type} & \rightarrow & \text{VOID}
\end{array}
$$

For the input "int f . . .", when we see the token INT, the next token is ID

In this situation, the parser does not know if it should reduce INT to return_type or data_type

State I_0 contains the following items

$$
\begin{array}{rcl}
\text{data\_type} & \rightarrow & \bullet\ \text{INT, ID} \\
\text{return\_type} & \rightarrow & \bullet\ \text{INT, ID}
\end{array}
$$

The transition on INT gives the following set of items showing a reduce-reduce conflict on ID

$$
\begin{array}{rcl}
\text{data\_type} & \rightarrow & \text{INT}\ \bullet\text{, ID} \\
\text{return\_type} & \rightarrow & \text{INT}\ \bullet\text{, ID}
\end{array}
$$

A Practical Example of Reduce-Reduce Conflict in LR(1) Parsing

In this particular case, the conflict can be removed by replacing every occurrence of the non-terminals data_type and return_type by every RHS of the non-terminal

Original Grammar			Transformed Grammar		
program	$\rightarrow$	func_decl var_decl	program	$\rightarrow$	func_decl var_decl
program	$\rightarrow$	var_decl func_decl	program	$\rightarrow$	var_decl func_decl
var_decl	$\rightarrow$	data_type ID ;	var_decl	$\rightarrow$	INT ID ;
data_type	$\rightarrow$	INT	func_decl	$\rightarrow$	INT ID ( )
func_decl	$\rightarrow$	return_type ID ( )	func_decl	$\rightarrow$	VOID ID ( )
return_type	$\rightarrow$	INT			
return_type	$\rightarrow$	VOID			

A Summary of Bottom Up Parsing Methods

Parsing Method	Items Used	Reduction by $A \rightarrow \alpha$	Remarks
SLR(0)	LR(0)	On any terminal	
SLR(1)	LR(0)	On the terminals in FOLLOW(A)	
LR(1), also known as Canonical LR(1) or CLR(1)	LR(1)	On lookahead a in the item "$A \rightarrow \alpha\bullet, a$"	
LALR(1)	LR(1)	On lookahead a in the item "$A \rightarrow \alpha\bullet, a$"	Conceptually, the sets of items are obtained by merging LR(1) item sets that differ only in the lookahead symbols Practically, lookaheads are propagated starting from \$ on LR(0) items

Comparison of Bottom-Up Methods and Corresponding Grammars

- A grammar G is accepted by a parsing method P if a conflict-free parser can be constructed for G using P
- An ambiguous grammar is not accepted by any parsing method
- A grammar is called SLR(0), SLR(1), LR(1), or LALR(1) if it is accepted respectively, by the SLR(0), SLR(1), LR(1), or LALR(1) parsing method
  - Every SLR(0) grammar is also SLR(1) grammar but not vice-versa
  - Every SLR(1) grammar is also LALR(1) grammar but not vice-versa
  - Every LALR(1) grammar is also LR(1) grammar but not vice-versa
- The expressions grammar ($E \rightarrow E + E \mid E * E \mid \text{id}$) is not accepted by any parsing method because it is ambiguous

  (without post-facto instrumentation of parsing tables using precedences and associativities)