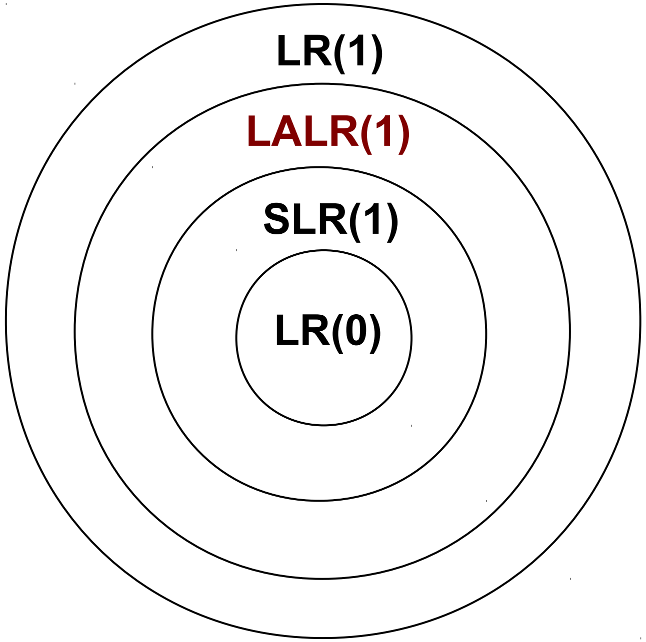
Bottom-up Parsing

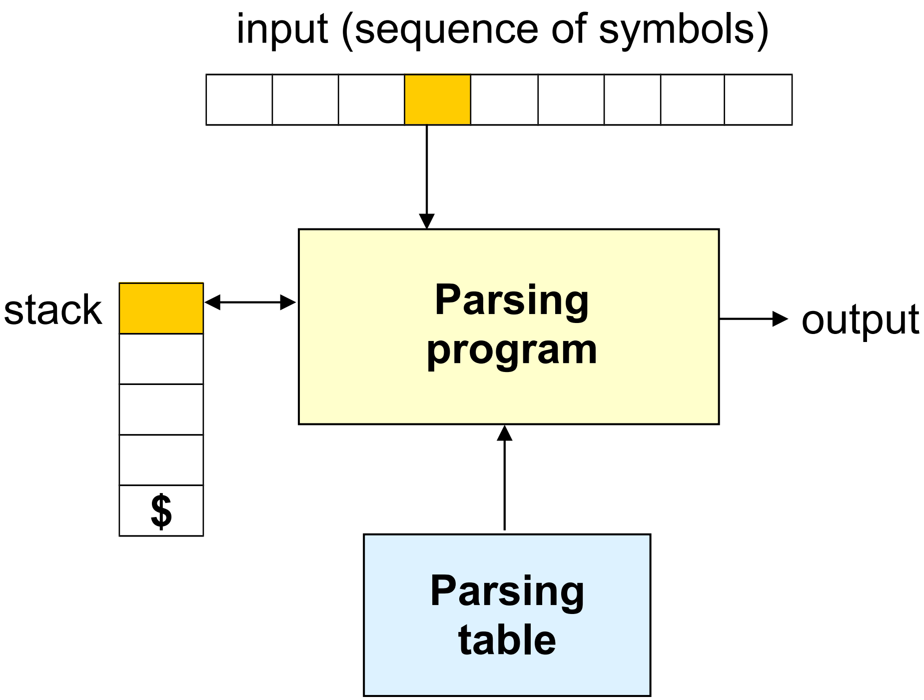
- Classification:



- Each class marked by parser **P** corresponds to the set of grammars which can be analyzed by **P**

Bottom-up Parsing (ii)

- Architecture similar to LL(1) parsing



Overview of parsing:

Stack	*Input*	*Action*
$	*string***$**	
...	...	...
$S'	**$**	accept

axiom of "extended" G

- Possible actions (besides **accept**):

  1. **Shift** current terminal from the front of the input to the top of the stack

  2. **Reduce** a string α at the top of the stack to a nonterminal **A**, given the production $\mathbf{A} \rightarrow \alpha$

Bottom-up Parsing (iii)

- For technical reasons, G extended with new
  - axiom: S'
  - production: $S' \to S$

1\. $S' \to S$
$S \to$ **(** S **)** $S \mid \varepsilon$

string = **( )**

Stack	*Input*	*Action*
$	()**$**	shift
$(	)**$**	$S \to \varepsilon$
$(S	)**$**	shift
$(S)	**$**	$S \to \varepsilon$
$(S)S	**$**	$S \to$ **(** S **)** S
$S	**$**	$S' \to S$
$S'	**$**	accept

2\. $E' \to E$
$E \to E$ **+ n** | **n**

string = **n + n**

Stack	*Input*	*Action*
$	n+n**$**	shift
$n	+n**$**	$E \to$ **n**
$E	+n**$**	shift
$E+	n**$**	shift
$E+n	**$**	$E \to E$ **+ n**
$E	**$**	$E' \to E$
$E'	**$**	accept

Bottom-up Parsing (iv)

	Stack	*Input*	*Action*
1	$	()$	shift
2	$(	)$	$S \to \varepsilon$
3	$(S	)$	shift
4	$(S)	$	$S \to \varepsilon$
5	$(S)S	$	$S \to$ **(** S **)** S
6	$S	$	$S' \to S$
7	$S'	$	accept

(A)

	Stack	*Input*	*Action*
1	$	n+n$	shift
2	$n	+n$	$E \to$ **n**
3	$E	+n$	shift
4	$E+	n$	shift
5	$E+n	$	$E \to E$ **+ n**
6	$E	$	$E' \to E$
7	$E'	$	accept

(B)

Notes:

1. To choose the action, need to "look" below the top of the stack (internal prospection, unlike LL(1))
   Example: (A) steps 5, 6: same top, but different reductions!

2. Arbitrary prospection within the stack: not a problem because stack built by the parser!

3. Action: depends not only on the stack but also on the current terminal
   Example: (B) steps 3, 6: same stack content, but different actions!

4. [Reductions] = Tracing in reverse order of a right canonical derivation
   - (A): $S' \Rightarrow S \Rightarrow (S)S \Rightarrow (S) \Rightarrow ()$
   - (B): $E' \Rightarrow E \Rightarrow E+n \Rightarrow n+n$

5. Stack+input = right sentential form $\to$ list of symbols on the stack $\equiv$ **viable prefix** of right sent. form

Bottom-up Parsing (v)

- Parser technique: shift of symbols from input to stack until possible a reduction <u>corresponding to the previous sentential form</u>

- Hence: α on top of the stack = necessary / <u>insufficient</u> condition for reduction $A \rightarrow \alpha$

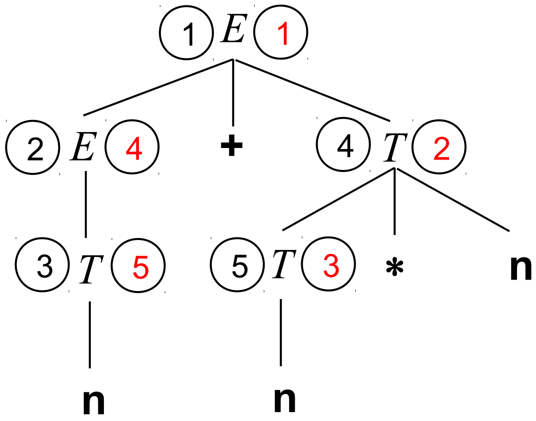
<u>Example</u>: (A): $S \rightarrow \varepsilon$ ⟹ $\alpha = \varepsilon$ is always on top of the stack!
Only reducible when stack+input = previous sentential form!

Step 3: **$**(S)**$** $S \rightarrow \varepsilon$ ⟹ **$**(SS)**$** ⟹ <u>**not**</u> a sentential form!

Bottom-up Parsing (vi)

$E \rightarrow E \; \mathbf{+} \; T \mid T$
$T \rightarrow T \; \mathbf{*} \; \mathbf{n} \mid \mathbf{n}$

n + n ∗ n



$E \Rightarrow E \; \mathbf{+} \; T \Rightarrow E \; \mathbf{+} \; T \; \mathbf{*} \; \mathbf{n} \Rightarrow E \; \mathbf{+} \; \mathbf{n} \; \mathbf{*} \; \mathbf{n} \Rightarrow T \; \mathbf{+} \; \mathbf{n} \; \mathbf{*} \; \mathbf{n} \Rightarrow \mathbf{n} \; \mathbf{+} \; \mathbf{n} \; \mathbf{*} \; \mathbf{n}$

Stack	*Input*	*Action*
$	n+n∗n$	shift
$n	+n∗n$	$T \rightarrow \mathbf{n}$
$T	+n∗n$	$E \rightarrow T$
$E	+n∗n$	shift
$E+	n∗n$	shift
$E+n	∗n$	$T \rightarrow \mathbf{n}$
$E+T	∗n$	shift
$E+T∗	n$	shift
$E+T∗ n	$	$T \rightarrow T \; \mathbf{*} \; \mathbf{n}$
$E+T	$	$E \rightarrow E \; \mathbf{+} \; T$
$E	$	accept

- Shift ≅ advance in input
- Reduction ≅ inverse derivation

LR(0) Parsing

- **LR(0) item** of G ≡ Production of G with a specified position in the RHS

  Intuitively: "contextualized" production $E \to E \,.\, + \mathbf{n}$

  Generically: $A \to \alpha,\ \ \alpha = \beta\gamma\ :\ \ A \to \beta.\gamma$

$S' \to S$
$S \to (\,S\,)\,S \mid \varepsilon$

⟹

$S' \to .\,S$
$S' \to S\,.$
$S \to .\,(\,S\,)\,S$
$S \to (\,.\,S\,)\,S$
$S \to (\,S\,.\,)\,S$
$S \to (\,S\,)\,.\,S$
$S \to (\,S\,)\,S\,.$
$S \to\ .$

$E' \to E$
$E \to E + \mathbf{n} \mid \mathbf{n}$

⟹

$E' \to .\,E$
$E' \to E\,.$
$E \to .\,E + \mathbf{n}$
$E \to E\,.+ \mathbf{n}$
$E \to E +.\,\mathbf{n}$
$E \to E + \mathbf{n}\,.$
$E \to .\,\mathbf{n}$
$E \to \mathbf{n}\,.$

- Intuitively: Item = representation of the recognition state of the RHS of a production

State
- general case: $A \to \beta.\gamma$
  - β: already analyzed ⟹ β on top of the stack !
  - γ: to be scanned (present in concrete form in a prefix of the input to be analyzed)
- specific cases
  - $A \to .\,\alpha$ ≡ **Initial item**: we start to recognize A by $A \to \alpha$
  - $A \to \alpha\,.$ ≡ **Complete item**: α on top of the stack: $A \to \alpha$ = candidate to reduction

LR(0) Parsing (ii)

- LR(0) items organized in **NFA of items** = (Σ, S, T, s_0)
  - Σ = { grammar symbols }
  - LR(0) items = states of an NFA maintaining the state of recognition of a shift/reduce parser
  - Transitions = ?

$$A \to \beta.X\delta \xrightarrow{\;X\;} A \to \beta X.\delta$$

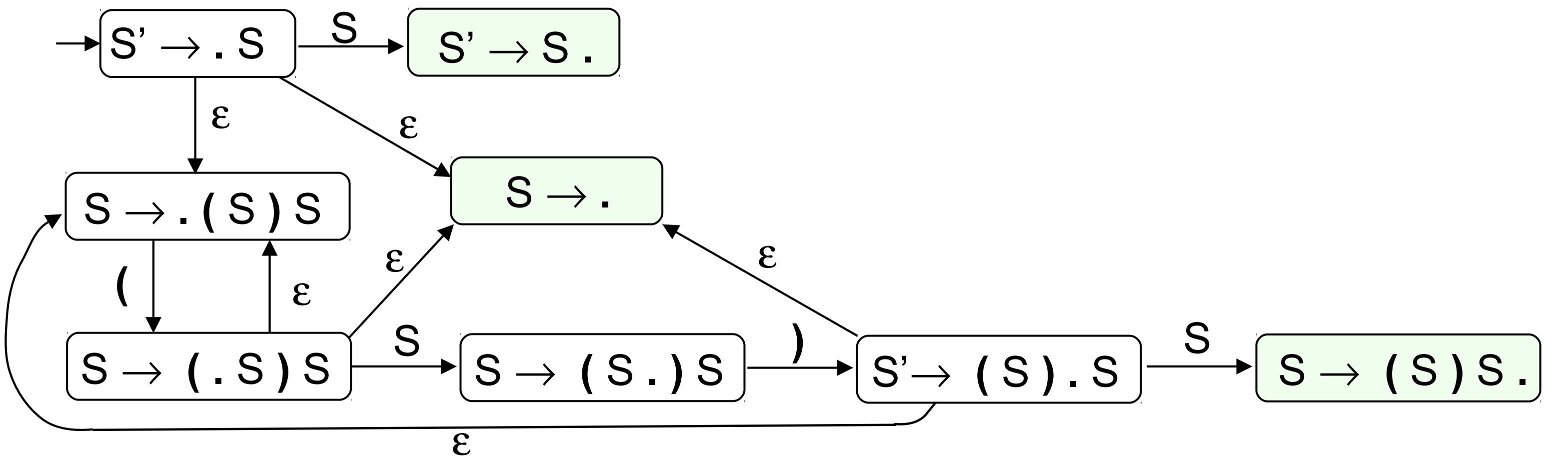
Possibilities: X
- terminal: shift of X on the stack
- nonterminal:
  - virtual shift of X on the stack, but following a reduction $X \to \eta$: must be preceded by a recognition of η
  - $.\,\eta$ = initial state of such recognition

$\Rightarrow$ $\forall$ alternative $X \to \eta$: $A \to \beta.X\delta \xrightarrow{\;\varepsilon\;} X \to .\eta$ (additional)

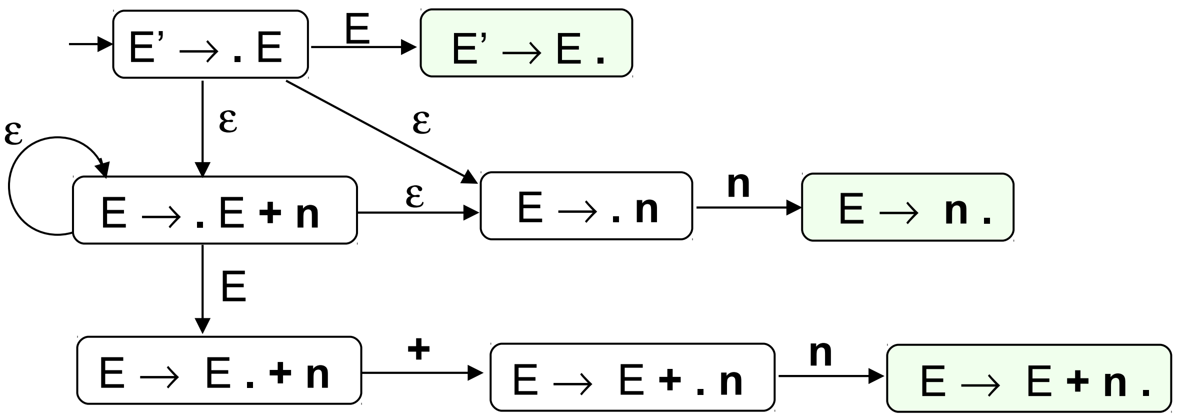
- Initial state? In theory: $S \to .\alpha$, but since $\exists \neq$ alternatives $\Rightarrow$ $S' \to .S$
- $\nexists$ final states: aim of the automaton
  - to maintain the state of bottom-up parsing
  - <u>not</u> recognition of strings!

Examples of LR(0) Parsing

1. $S' \rightarrow S$
   $S \rightarrow \mathbf{(}\ S\ \mathbf{)}\ S \mid \varepsilon$

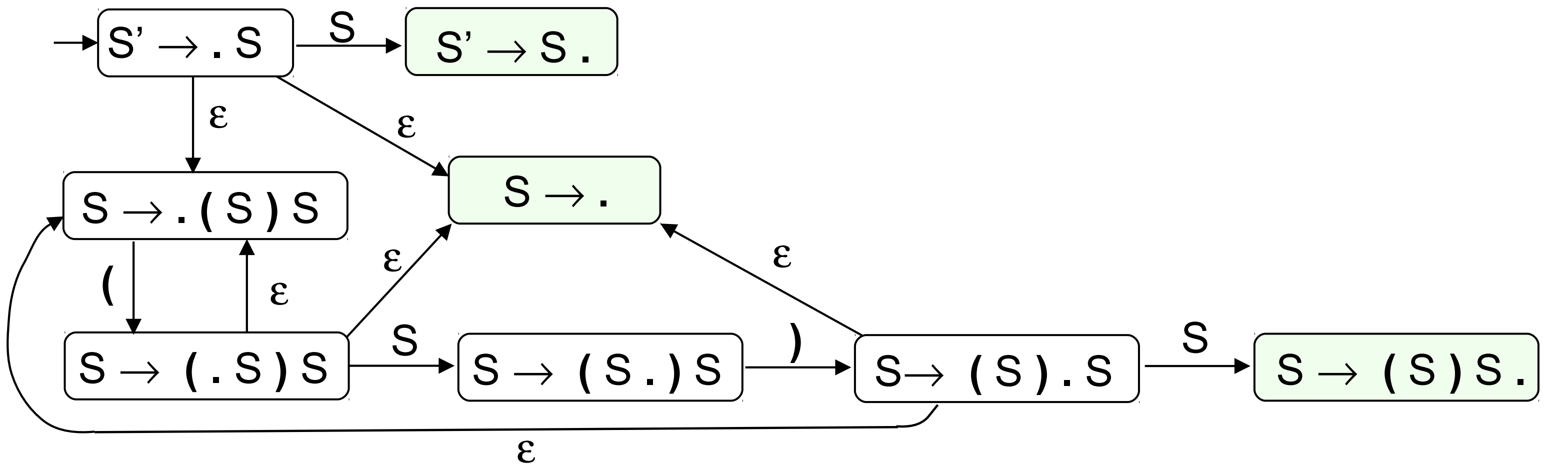


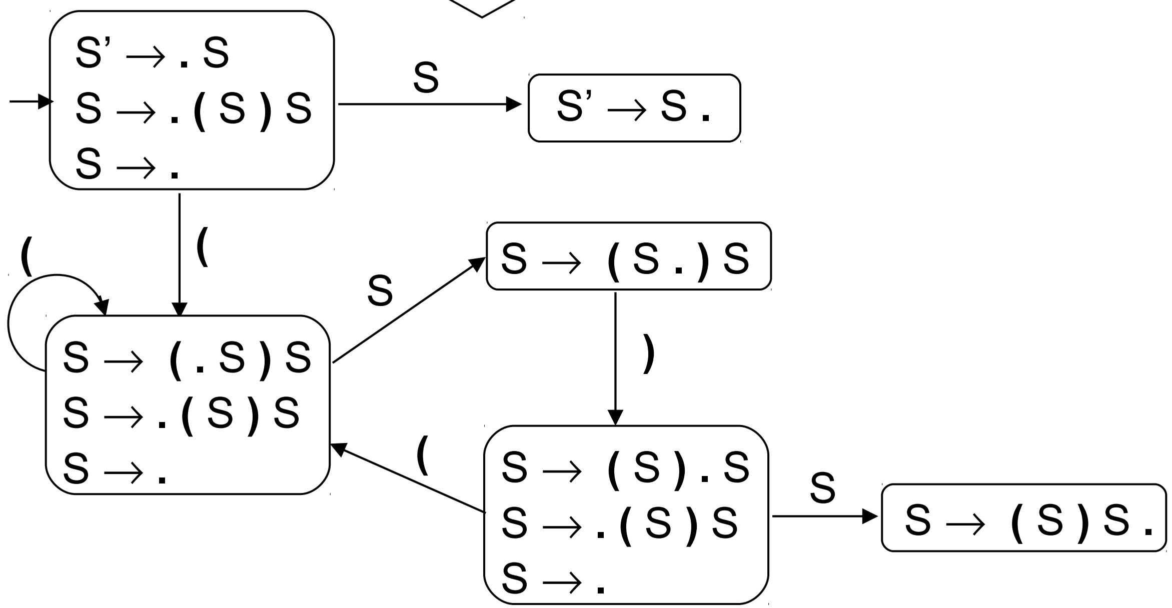
2. $E' \rightarrow E$
   $E \rightarrow E\ \mathbf{+}\ \mathbf{n} \mid \mathbf{n}$



Transformation NFA → DFA

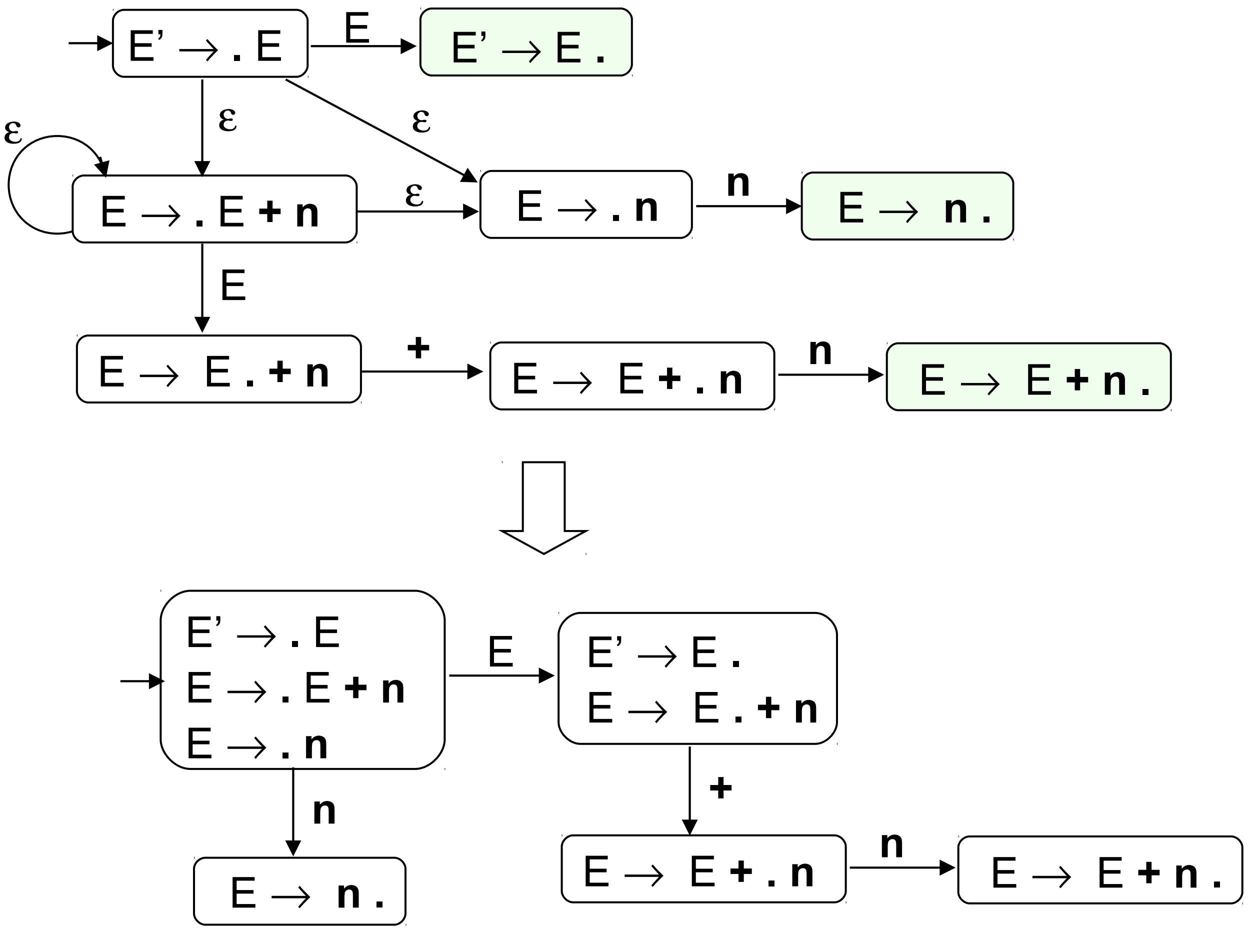
1.





Transformation NFA → DFA (ii)

2.



sufficient to identify the state

- Within state: distinction
  - **kernel items** ≡ { states reached by non-empty transitions (or initial state) }
  - **closure items** ≡ { states reached by ε-closure }

LR(0) Parsing Algorithm

<u>Note</u>: Need to maintain within the stack the information on the state too $\Rightarrow$

state
symbol

(pairs)

<u>Example</u>:

\$0	n+n\$
\$0n2	+n\$

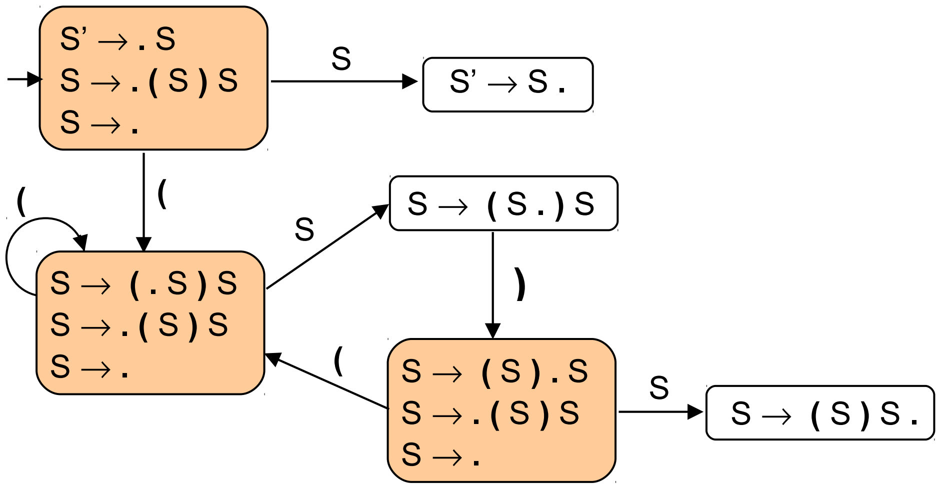
stack := \$0; *lookahead* := first input symbol;
repeat
  s := Top(stack); /* s is a state */
  if A $\rightarrow$ β.X$\delta \in$ s **and** Terminal(X) **then**
    Shift *lookahead* on the stack;
shift item
    if A' $\rightarrow$ β'. X'δ' $\in$ s **and** Terminal(**X'**) **and X'** = Top(stack) **then**
      Push(s'), where s $\xrightarrow{X'}$ s' is a transition in the DFA
    else Error()
  end-if;
reduce item
  if A $\rightarrow$ η . $\in$ s **then**
    Reduce A $\rightarrow$ η;
    if A $\rightarrow$ η = S' $\rightarrow$ S **then**
      if *lookahead* = **\$ then** Accept **else** Error()
    else
      Remove η with its states from the stack; /* η is on top of the stack by construction */
      s' := Top(stack); /* B $\rightarrow$ θ . A $\delta \in$ s' */
      Push(A); Push(s''), where s' $\xrightarrow{A}$ s'' is a transition in the DFA
  end-if
until acceptance or error.

LR(0) Grammars

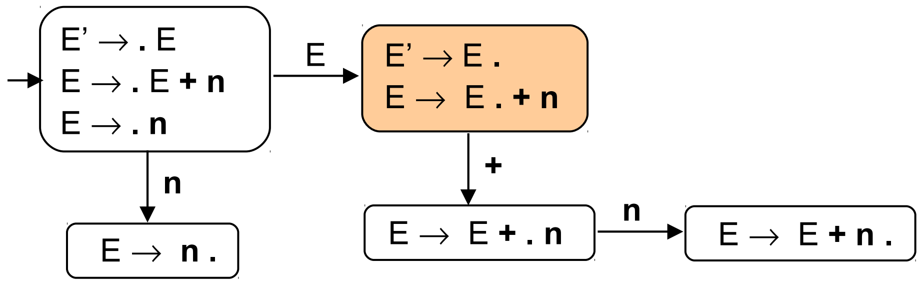
<u>Def</u>: G is LR(0) if the actions of the algorithm are unambiguous, that is,
$\forall$ state of the DFA:

$\nexists$ conflict
- **shift/reduce**: $s \not\supseteq \{ A \rightarrow \alpha.,\ B \rightarrow \delta.a\gamma \}$
- **reduce/reduce**: $s \not\supseteq \{ A \rightarrow \alpha.,\ B \rightarrow \beta. \}$

$S' \rightarrow S$
$S \rightarrow \textbf{(}\ S\ \textbf{)}\ S \mid \varepsilon$

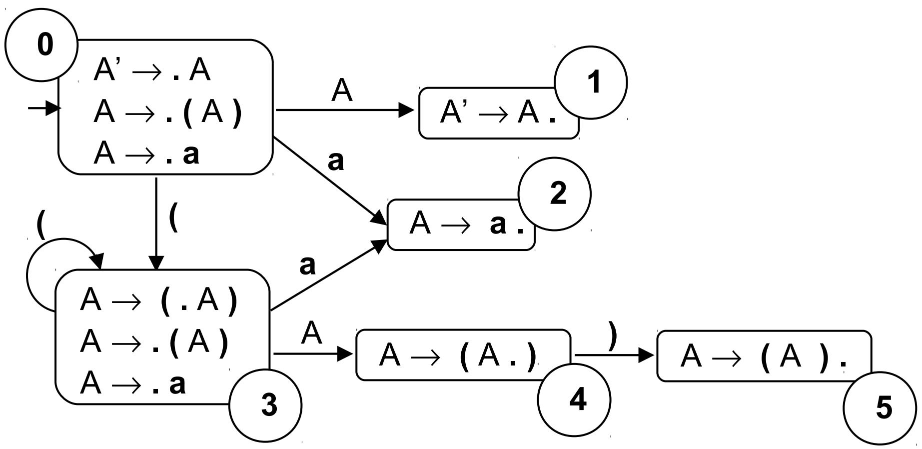


$E' \rightarrow E$
$E \rightarrow E\ \textbf{+}\ \textbf{n} \mid \textbf{n}$



LR(0) Grammars (ii)

$A' \rightarrow A$
$A \rightarrow$ **(** A **)** | **a**

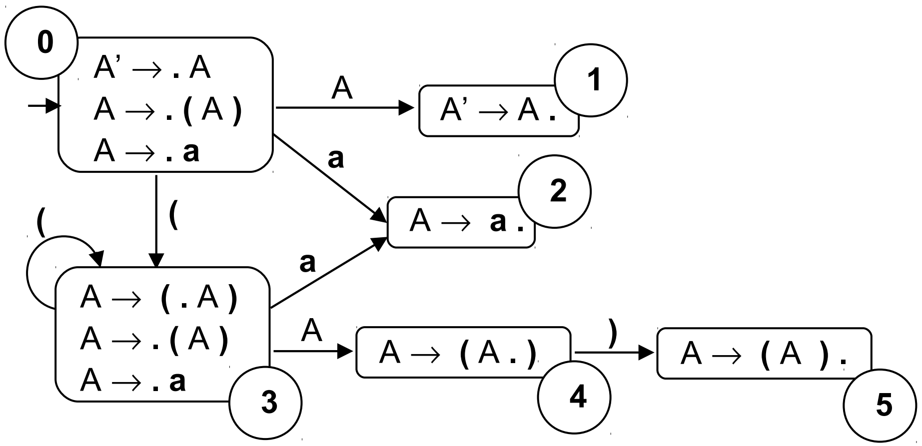


((a))

	Stack	*Input*	*Action*
1	$0	((a))**$**	shift
2	$0 (3	(a))**$**	shift
3	$0 (3 (3	a))**$**	shift
4	$0 (3 (3 a2	))**$**	$A \rightarrow$ **a**
5	$0 (3 (3 A4	))**$**	shift
6	$0 (3 (3 A4)5	)**$**	$A \rightarrow$ **(** A **)**
7	$0 (3 A4	)**$**	shift
8	$0 (3 A4)5	**$**	$A \rightarrow$ **(** A **)**
9	$0 A1	**$**	accept

LR(0) Parsing Table

LR(0) algorithm: table-driven (automaton extended with actions $\rightarrow$ parsing table)



State	*Action*	*Production*	*Input*			*Goto*
			(	**a**	**)**	*A*
0	shift		3	2		1
1	reduce	$A' \rightarrow A$				
2	reduce	$A \rightarrow$ **a**				
3	shift		3	2		4
4	shift				5	
5	reduce	$A \rightarrow (A)$				

SLR(1) Parsing

- Sufficiently powerful for almost all constructs of PLs in use
- <u>Idea</u>: Exploitation of the next input symbol to decide which action to perform:

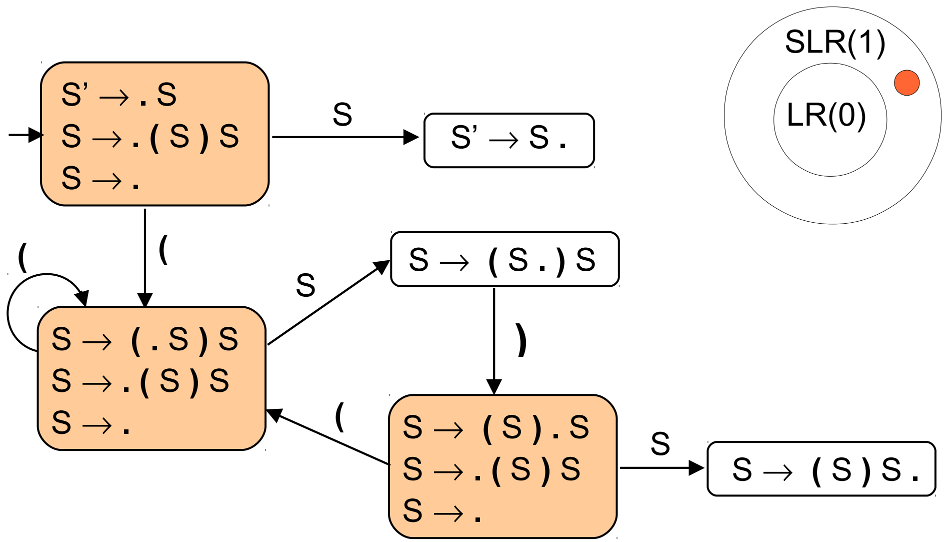
In 2 ways:
- before the shift
- before the reduction: $FOLLOW(A)$: to decide whether to reduce $A \rightarrow \alpha$

stack := \$0; *lookahead* := first input symbol;
repeat
  s := Top(stack); /* s is a state */
  if $A \rightarrow \beta\, .\, X\, \delta \in$ s **and** Terminal(X) **and** X = *lookahead* **then**
    Shift *lookahead* on the stack;
    Push(s'), where $s \xrightarrow{X} s'$ is a transition in the DFA
  else if $A \rightarrow \eta\, . \in$ s **and** *lookahead* $\in FOLLOW(A)$ **then**
    Reduce $A \rightarrow \eta$;
    if $A \rightarrow \eta = S' \rightarrow S$ **then**
      Accept /* *lookahead* = **\$**, since $FOLLOW(A)$ = { **\$** } */
    else
      Remove η with its states from the stack; /* η is on top of the stack by construction */
      s' := Top(stack); /* $B \rightarrow \theta\, .\, A\, \delta \in$ s' */
      Push(A); Push(s''), where $s' \xrightarrow{A} s''$ is a transition in the DFA
  else Error()
until acceptance or error.

SLR(1) Grammars

<u>Def</u>: G is SLR(1) if $\forall$ s of the DFA (unambiguous actions):

1. $\forall\ A \rightarrow \alpha.a\beta \in$ s, Terminal(a) ($\nexists\ B \rightarrow \gamma. \in$ s (a $\in$ *FOLLOW*(B)));
2. $\forall\ A \rightarrow \alpha. \in$ s, $\forall\ B \rightarrow \beta. \in$ s (*FOLLOW*(A) $\cap$ *FOLLOW*(B) = $\emptyset$).

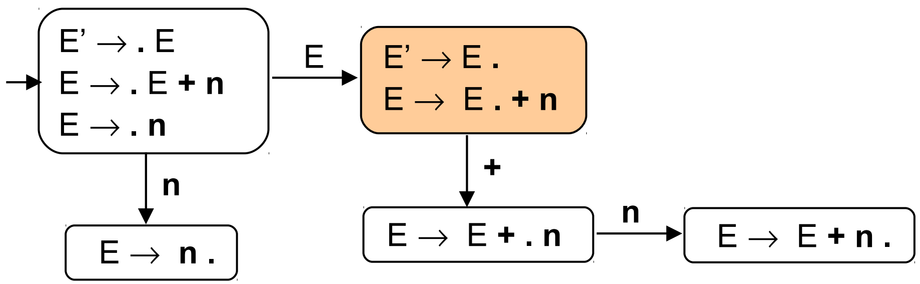


$S' \rightarrow S$
$S \rightarrow$ **(** S **)** $S \mid \varepsilon$

FOLLOW(*S'*) = { **$** }
FOLLOW(*S*) = { **$**, **)** }

$\Downarrow$

($\notin$ { **$**, **)** }



$E' \rightarrow E$
$E \rightarrow E$ **+ n** | **n**

FOLLOW(*E'*) = { **$** }
FOLLOW(*E*) = { **+**, **$** }

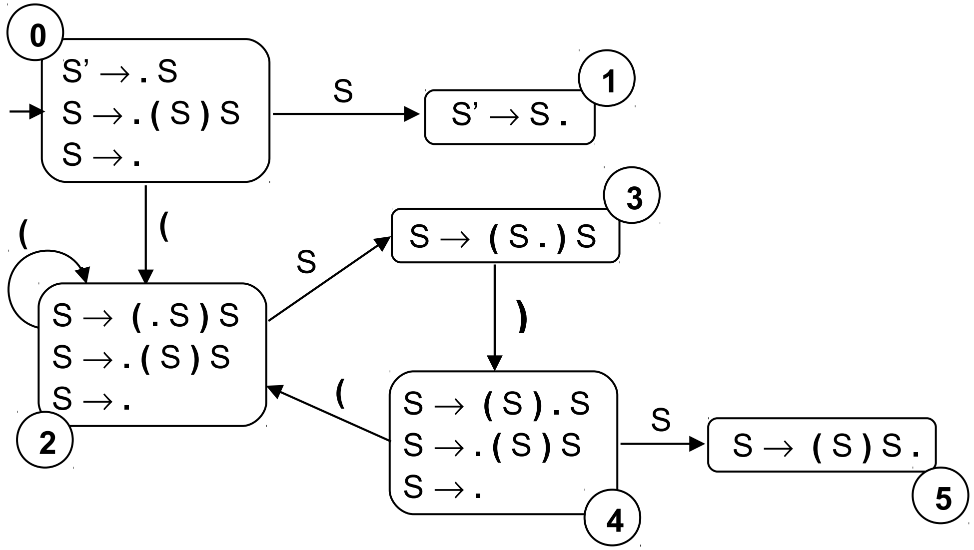
$\Downarrow$

+ $\notin$ { **$** }

SLR(1) Grammars (ii)

$S' \rightarrow S$
$S \rightarrow \mathbf{(}\ S\ \mathbf{)}\ S \mid \varepsilon$

$FOLLOW(S) = \{\ \$,\ )\ \}$



polimorphic

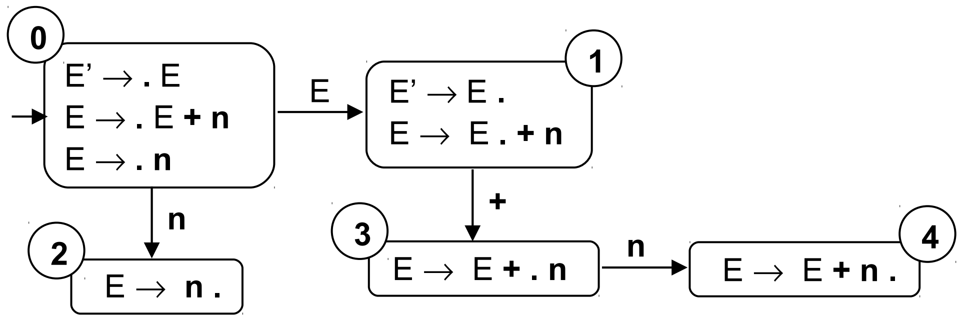
State	Input			Goto
	(	)	$	S
0	s2	$S \rightarrow \varepsilon$	$S \rightarrow \varepsilon$	1
1			accept	
2	s2	$S \rightarrow \varepsilon$	$S \rightarrow \varepsilon$	3
3		s4		
4	s2	$S \rightarrow \varepsilon$	$S \rightarrow \varepsilon$	5
5		$S \rightarrow (\ S\ )\ S$	$S \rightarrow (\ S\ )\ S$	

( ) ( )

Stack	*Input*	*Action*
$0	()()$	shift
$0 (2	)()$	$S \rightarrow \varepsilon$
$0 (2 S3	)()$	shift
$0 (2 S3)4	()$	shift
$0 (2 S3)4 (2	)$	$S \rightarrow \varepsilon$
$0 (2 S3)4 (2 S3	)$	shift
$0 (2 S3)4 (2 S3)4	$	$S \rightarrow \varepsilon$
$0 (2 S3)4 (2 S3)4 S5	$	$S \rightarrow (\ S\ )\ S$
$0 (2 S3)4 S5	$	$S \rightarrow (\ S\ )\ S$
$0 S1	$	accept

SLR(1) Grammars (iii)

$E' \rightarrow E$
$E \rightarrow E$ **+ n** | **n**



$FOLLOW(E')$ = { **$** }
$FOLLOW(E)$ = { **+**, **$** }

n + n + n

State	*Input*			*Goto*
	n	**+**	**$**	*E*
0	s2			1
1		s3	accept	
2		$E \rightarrow$ **n**	$E \rightarrow$ **n**	
3	s4			
4		$E \rightarrow E$ **+ n**	$E \rightarrow E$ **+ n**	

Stack	*Input*	*Action*
$0	n+n+n**$**	shift
$0 n2	+n+n**$**	$E \rightarrow$ **n**
$0 E1	+n+n**$**	shift
$0 E1 +3	n+n**$**	shift
$0 E1 +3 n4	+n**$**	$E \rightarrow E$ **+ n**
$0 E1	+n**$**	shift
$0 E1 +3	n**$**	shift
$0 E1 +3 n4	**$**	$E \rightarrow E$ **+ n**
$0 E1	**$**	accept

Disambiguating Rules for Parsing Conflicts

- Conflict
  - shift/reduce $\rightarrow$ Chosen the <u>shift</u>
  - reduce/reduce $\rightarrow$ Error in the design of G?

- <u>Example</u>: shift/reduce conflict

$$
\begin{aligned}
&stat \rightarrow if\text{-}stat \mid \textbf{other}\\
&if\text{-}stat \rightarrow \textbf{if}\ expr\ \textbf{then}\ stat \mid\\
&\qquad\qquad \textbf{if}\ expr\ \textbf{then}\ stat\ \textbf{else}\ stat\\
&expr \rightarrow \textbf{true} \mid \textbf{false}
\end{aligned}
$$

$\Rightarrow$ G ambiguous $\rightarrow$ must $\exists$ conflict somewhere!

$\Downarrow$ abstraction (removal of *expr* and **then**)

$$
\begin{aligned}
&S' \rightarrow S\\
&S \rightarrow I \mid \textbf{other}\\
&I \rightarrow \textbf{if}\ S \mid \textbf{if}\ S\ \textbf{else}\ S
\end{aligned}
$$

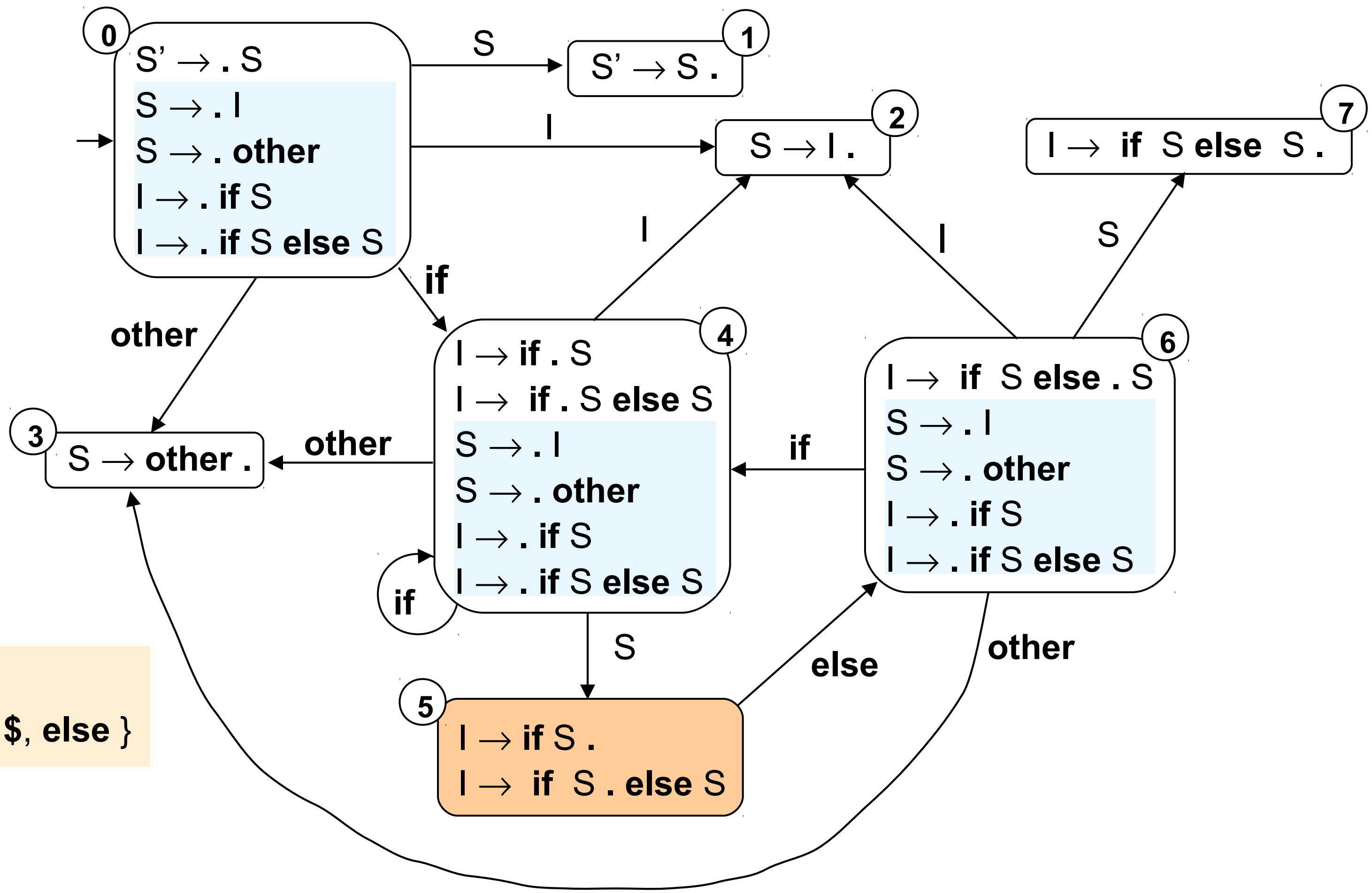
$\Rightarrow$

$FOLLOW(S') = \{ \textbf{\$} \}$

$FOLLOW(S) = FOLLOW(I) = \{ \textbf{\$}, \textbf{else} \}$

Disambiguating Rules for Parsing Conflicts (ii)

$S' \rightarrow S$
$S \rightarrow I$ | **other**
$I \rightarrow$ **if** S | **if** S **else** S



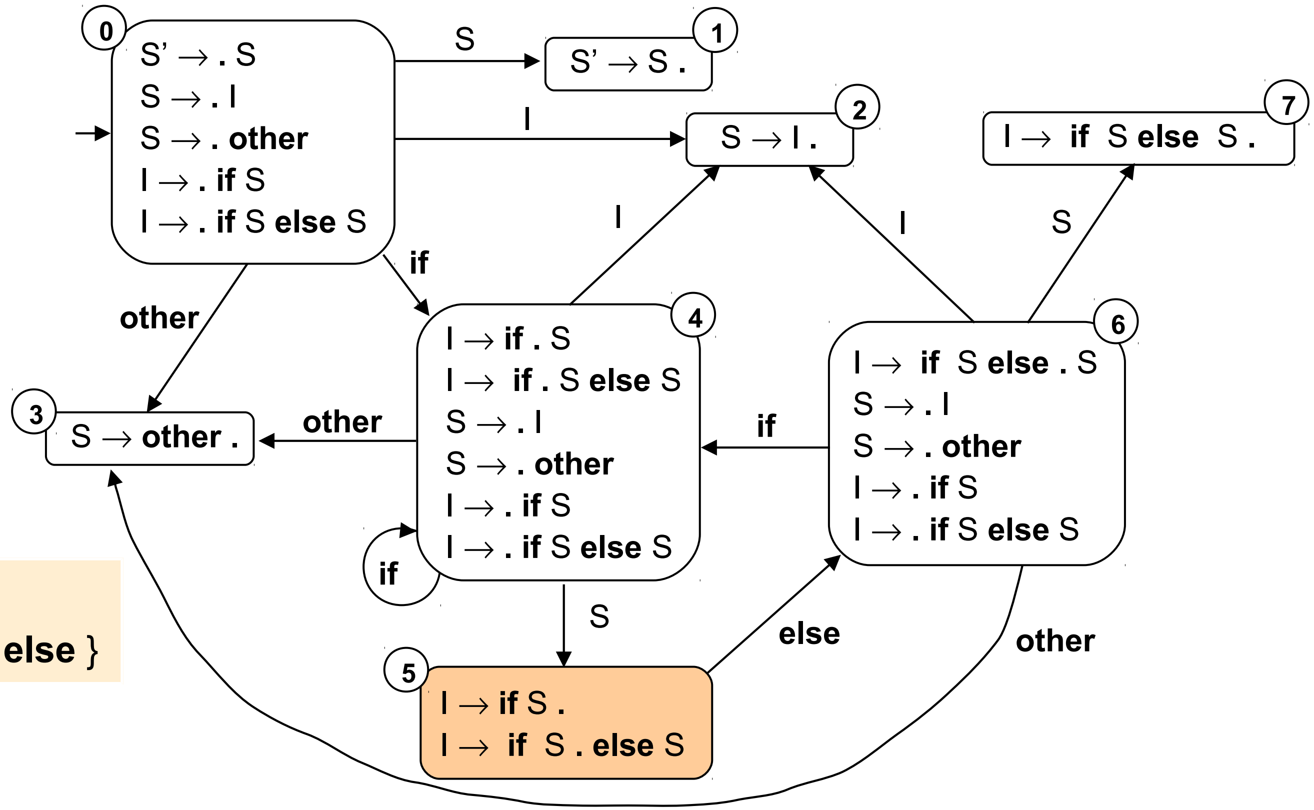
$FOLLOW(S') = \{ \$ \}$

$FOLLOW(S) = FOLLOW(I) = \{ \$, \textbf{else} \}$

- State 5 ⟨ Reduction on input ∈ { **$**, **else** } / Shift on input = **else** ⟹ Shift/reduce conflict on **else** ! ⟹ chosen the shift

Disambiguating Rules for Parsing Conflicts (iii)

$S' \to S$
$S \to I \mid$ **other**
$I \to$ **if** $S \mid$ **if** S **else** S



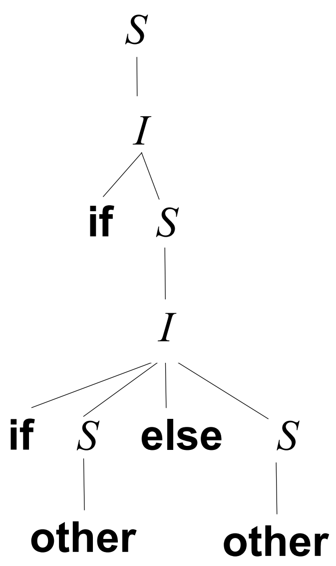
$FOLLOW(S') = \{ \$ \}$
$FOLLOW(S) = FOLLOW(I) = \{ \$,$ **else** $\}$

State	*Input*				*Goto*	
	if	**else**	**other**	**$**	*S*	*I*
0	s4		s3		1	2
1				accept		
2		$S \to I$		$S \to I$		
3		$S \to$ **other**		$S \to$ **other**		
4	s4		s3		5	2
5		s6		$I \to$ **if** S		
6	s4		s3		7	2
7		$I \to$ **if** S **else** S		$I \to$ **if** S **else** S		

Disambiguating Rules for Parsing Conflicts (iv)

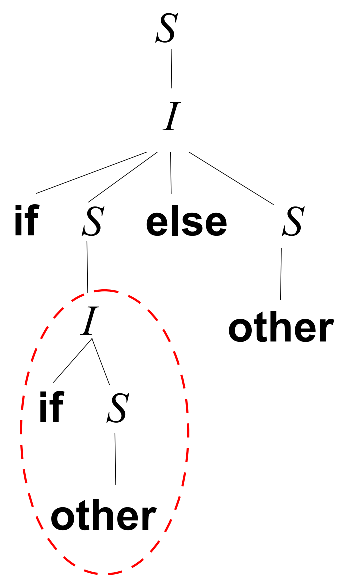
$S' \rightarrow S$
$S \rightarrow I$ | **other**
$I \rightarrow$ **if** S | **if** S **else** S

if if other else other



Stack	*Input*	*Action*
\$0	if if other else other \$	shift
\$0 if 4	if other else other \$	shift
\$0 if 4 if 4	other else other \$	shift
\$0 if 4 if 4 other 3	else other \$	$S \rightarrow$ **other**
\$0 if 4 if 4 S 5	else other \$	shift
\$0 if 4 if 4 S 5 else 6	other \$	shift
\$0 if 4 if 4 S 5 else 6 other 3	\$	$S \rightarrow$ **other**
\$0 if 4 if 4 S 5 else 6 S 7	\$	$I \rightarrow$ **if** S **else** S
\$0 if 4 I 2	\$	$S \rightarrow I$
\$0 if 4 S 5	\$	$I \rightarrow$ **if** S
\$0 I 2	\$	$S \rightarrow I$
\$0 S 1	\$	accept

State	*Input*				*Goto*	
	if	**else**	**other**	**\$**	S	I
0	s4		s3		1	2
1				accept		
2		$S \rightarrow I$		$S \rightarrow I$		
3		$S \rightarrow$ **other**		$S \rightarrow$ **other**		
4	s4		s3		5	2
5		s6		$I \rightarrow$ **if** S		
6	s4		s3		7	2
7		$I \rightarrow$ **if** S **else** S		$I \rightarrow$ **if** S **else** S		



Limits of SLR(1) Parsing

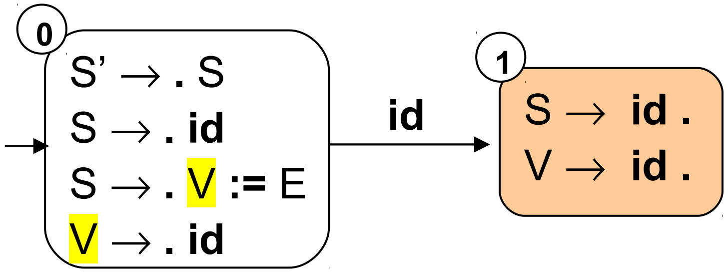
$stat \rightarrow\ call\text{-}stat \mid assign\text{-}stat$
$call\text{-}stat \rightarrow$ **id**
$assign\text{-}stat \rightarrow var$ **:=** $expr$
$var \rightarrow var$ **[** $expr$ **]** | **id**
$expr \rightarrow var$ | **num**

⇒ Simplification + abstraction ⇒

$S' \rightarrow\ S$
$S \rightarrow$ **id** | V **:=** E
$V \rightarrow$ **id**
$E \rightarrow V$ | **num**

$\nexists$
- *call-stat*
- *assign-stat*
- *var* **[** *expr* **]**

$FOLLOW(S') = \{$ **\$** $\}$
$FOLLOW(S) = \{$ **\$** $\}$
$FOLLOW(E) = \{$ **\$** $\}$
$FOLLOW(V) = \{$ **\$**, **:=** $\}$



⇒ $FOLLOW(S) \cap FOLLOW(V) = \{ \$ \} \neq \varnothing$

<u>Note</u>: Actually, the reduce/reduce conflict is a false problem, caused by the myopia (low discrimination power) of SLR(1), since, <u>within context of state 1</u>, V <u>cannot</u> be followed by **\$**, but only by **:=**

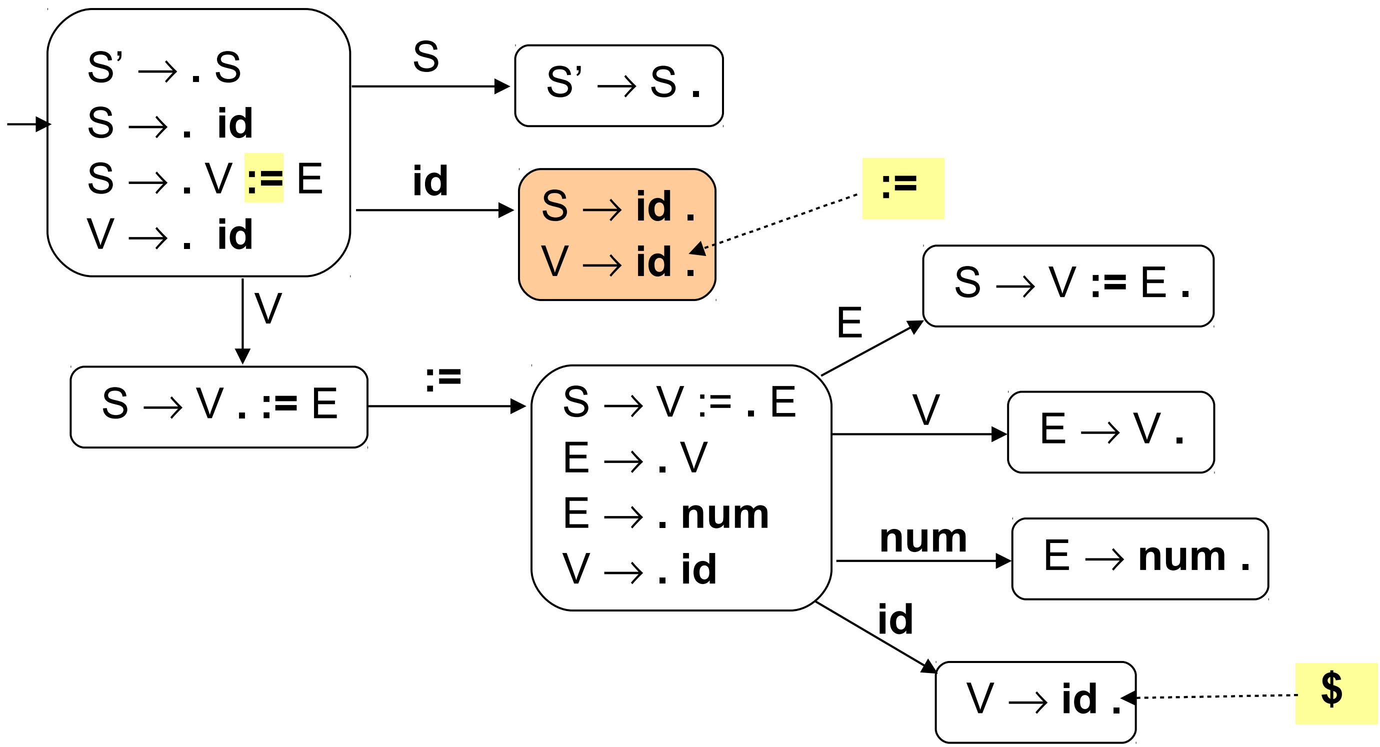
⇓

need for <u>contextual</u> prospection!

Limits of SLR(1) Parsing (ii)

$S' \rightarrow S$
$S \rightarrow$ **id** | V **:=** E
$V \rightarrow$ **id**
$E \rightarrow V$ | **num**

$FOLLOW(S') = \{ \$ \}$
$FOLLOW(S) = \{ \$ \}$
$FOLLOW(E) = \{ \$ \}$
$FOLLOW(V) = \{ \$, := \}$



Note: Reduce item **A → η** in a state: not followed by all symbols in *FOLLOW*(**A**)

LR(1) Parsing

- In general, LR(1) too complex $\rightarrow$ LALR(1) : maintains $\left\langle \begin{array}{l} \text{most power of LR(1)} \\ \text{efficiency of SLR(1)} \end{array} \right.$

- Pb of SLR(1): Applies lookahead symbols <u>after</u> constructing the DFA $\rightarrow$ context-free!

- LR(1): Incorporates lookahead symbols within construction of DFA $\rightarrow$ context-sensitive prospection!

- <u>Def</u>: **LR(1) item** of G $\equiv$ pair (LR(0) item, Lookahead symbol) = $[\, A \rightarrow \alpha \,.\, \beta,\ a \,]$

  context-sensitive!

- <u>Def</u>: **LR(1) transition**:

  1. $[\, A \rightarrow \alpha \,.\, X\gamma,\ a \,] \xrightarrow{X} [\, A \rightarrow \alpha X \,.\, \gamma,\ a \,]$ (X = grammar symbol)

  2. $[\, A \rightarrow \alpha \,.\, B\gamma,\ a \,] \xrightarrow{\varepsilon} [\, B \rightarrow .\, \beta,\ b \,]$ $\forall \left\langle \begin{array}{l} \text{production } B \rightarrow \beta \\ \text{symbol } b \in \mathit{FIRST}(\gamma a) \end{array} \right.$

- <u>Notes</u>:

  1. Better prospection of LR(1) wrt SLR(1) owing to: $\mathit{FIRST}(\gamma a) \subseteq \mathit{FOLLOW}(B)$

  2. Initial state of NFA of LR(1) items = $[\, S' \rightarrow .\, S,\ \$ \,]$

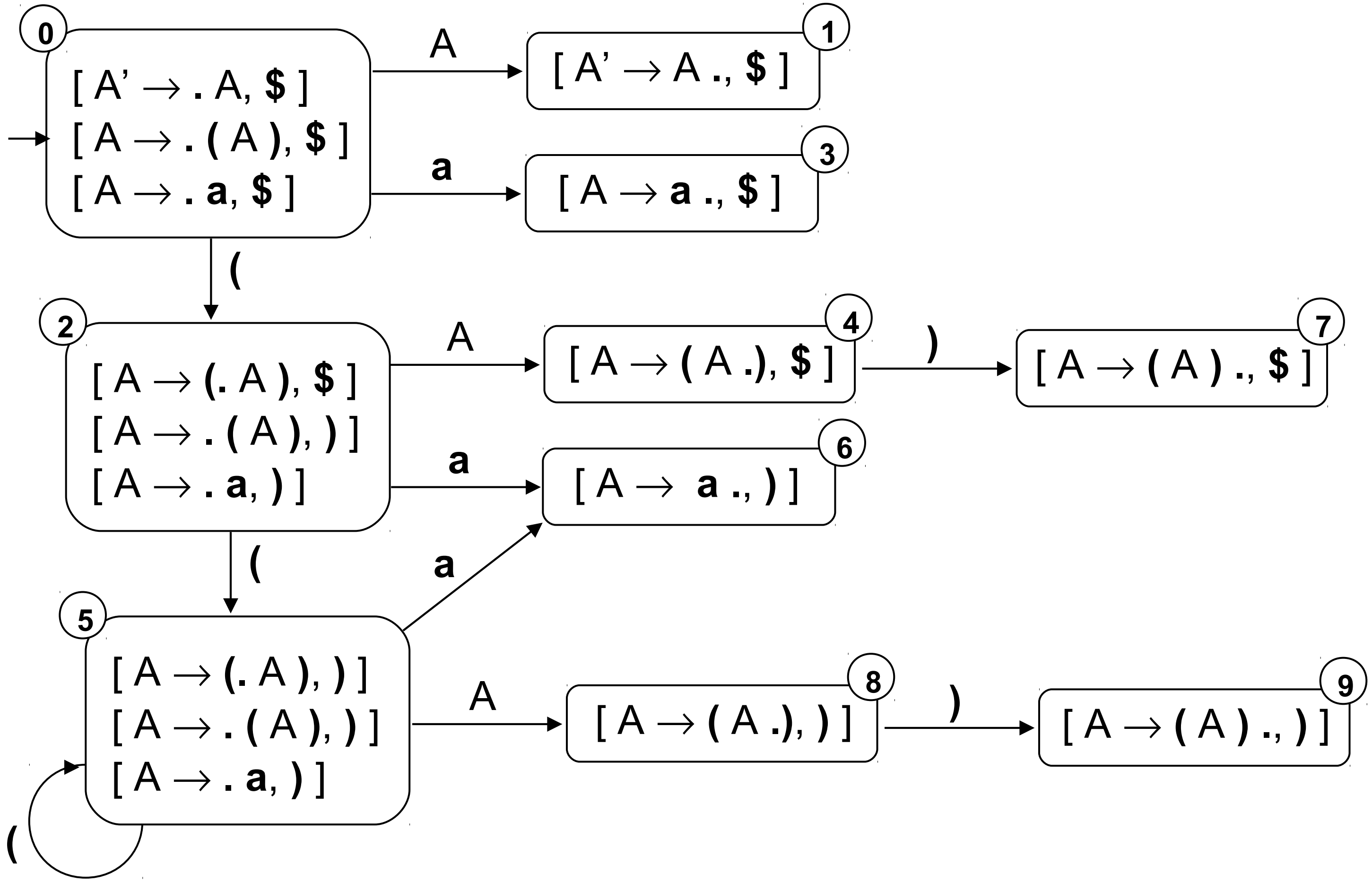
  3. $\gamma = \varepsilon \implies [\, A \rightarrow \alpha \,.\, B,\ a \,] \xrightarrow{\varepsilon} [\, B \rightarrow .\, \beta,\ a \,]$

  $[\, S' \rightarrow .\, S,\ \$ \,] \xrightarrow{\varepsilon} [\, S \rightarrow \ldots,\ \$ \,]$

LR(1) Parsing (ii)

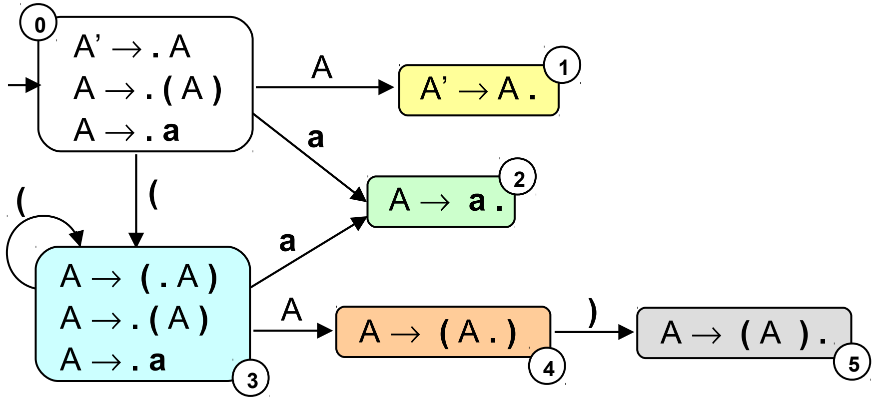
Attention focused on closure items → lookahead ∈ *FIRST*(γ a)

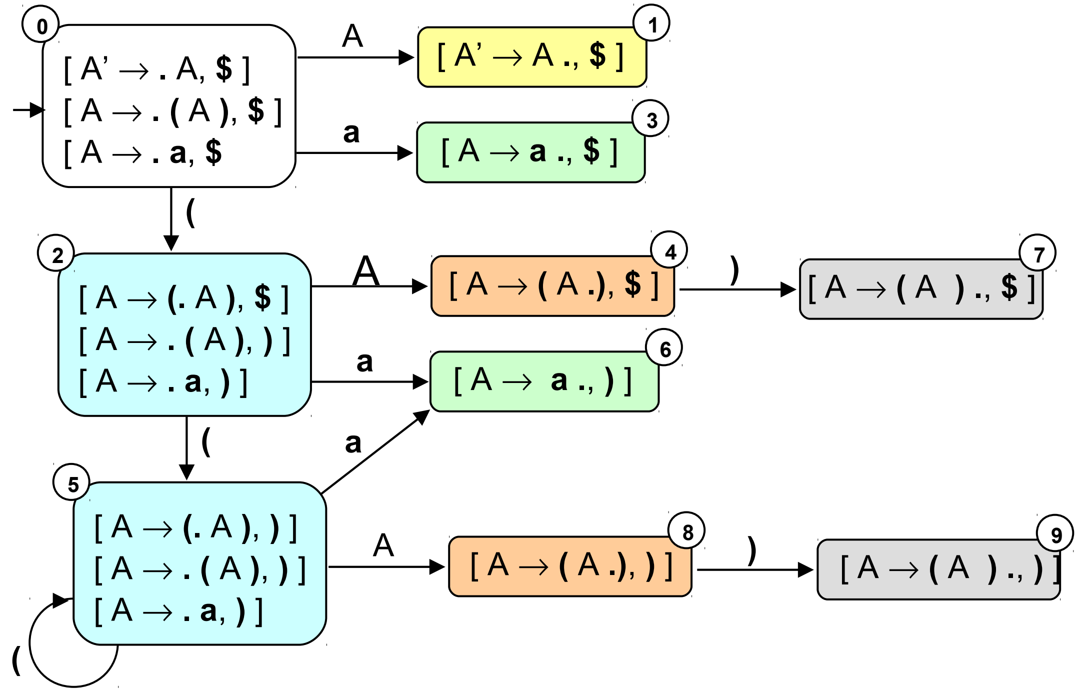
$A' \to A$
$A \to \mathbf{(}\ A\ \mathbf{)} \mid \mathbf{a}$



LR(1) Parsing (iii)

- <u>Note</u>: 10 states instead of 6 in LR(0) DFA → in general: even a difference of an order of magnitude! (hypertrophy of LR(1) DFA)





- Correspondence:

LR(0) state	*LR(1) states*
0	0
1	1
2	3, 6
3	2, 5
4	4, 8
5	7, 9

LR(1) Parsing Algorithm

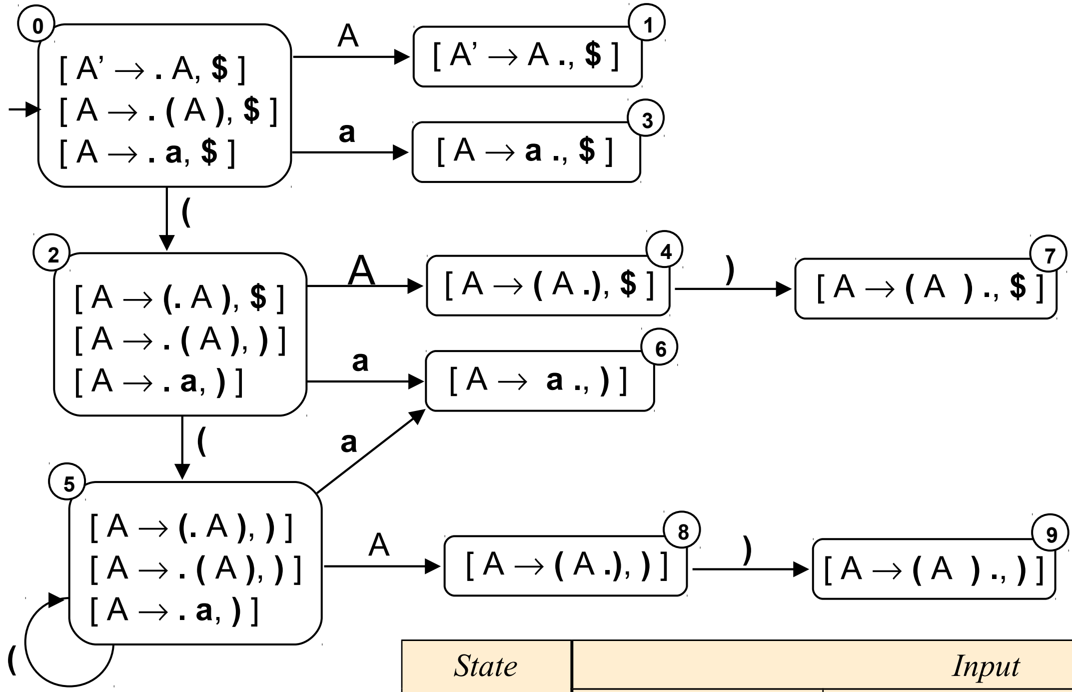
stack := \$0; *lookahead* := first input symbol;
repeat
  s := Top(stack);
  if $[A \rightarrow \beta.X\delta, a] \in s$ **and** Terminal(X) **and** X = *lookahead* **then**
    Shift *lookahead* on the stack;
    Push(s'), where $s \xrightarrow{X} s'$ is a transition in DFA
  else if $[A \rightarrow \eta., a] \in s$ **and** *lookahead* = a **then**
    Reduce $A \rightarrow \eta$;
    if $A \rightarrow \eta = S' \rightarrow S$ **then**
      Accept
    else
      Remove η with its states from the stack;   /* η is on top of the stack by construction */
      s' := Top(stack);
      Push(A); Push(s''), where $s' \xrightarrow{A} s''$ is a transition in DFA
  else Error()
until acceptance o error.

Def: G is LR(1) if $\forall$ s of DFA (no conflicts):

1. $\forall [A \rightarrow \alpha.X\beta, a] \in s$, Terminal(X) $([B \rightarrow \gamma., X] \notin s)$;
2. $\neg([A \rightarrow \alpha., a] \in s, [B \rightarrow \beta., a] \in s)$.

LR(1) Parsing Table

Invariance of morphology of parsing table $\rightarrow$ reduction in correspondence of the symbols indicated in LR(1) items



$A' \rightarrow A$

$A \rightarrow$ **(** A **)** | **a**

$FOLLOW(A') = \{ \$ \}$

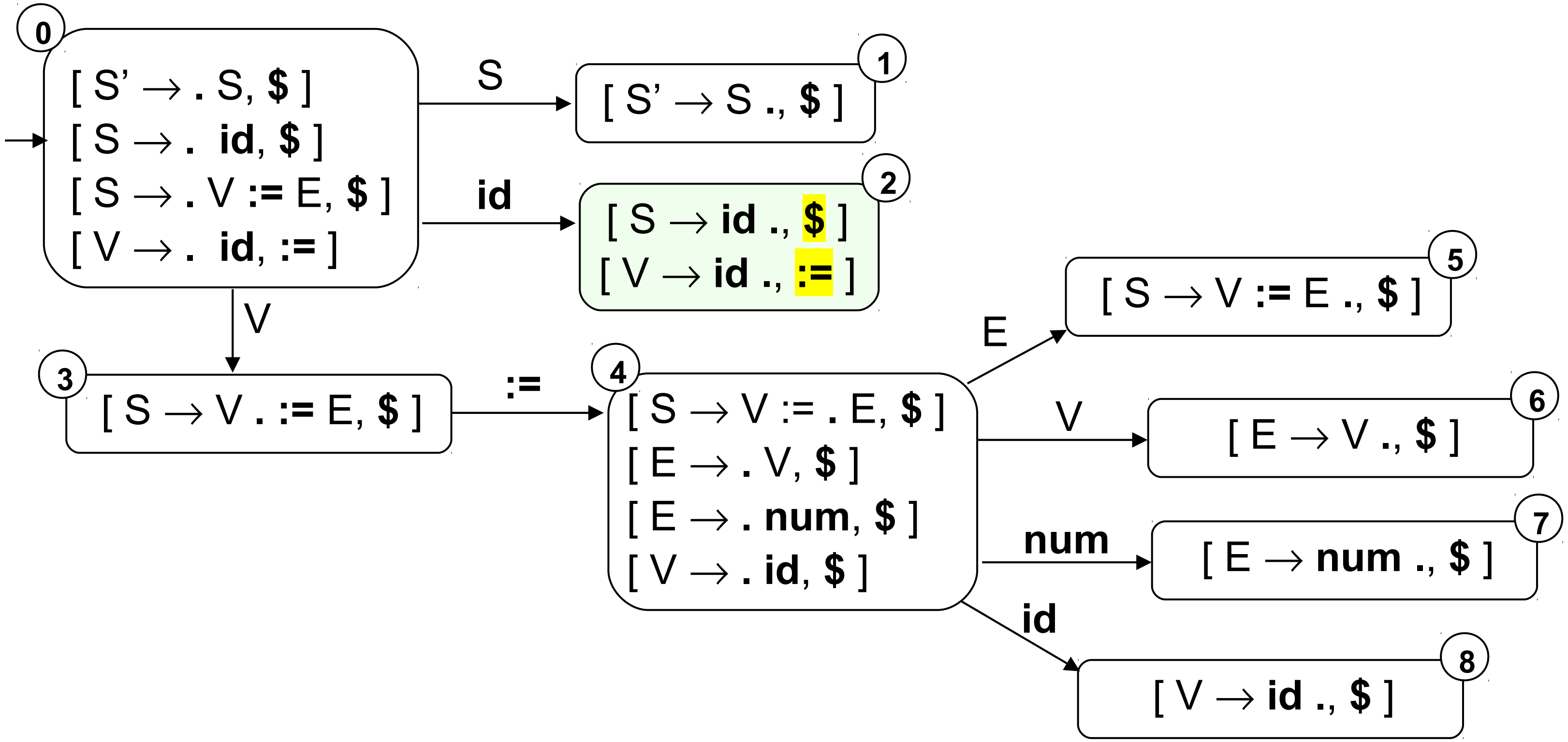
$FOLLOW(A) = \{ \$,) \}$

State	*Input*				*Goto*
	(	**a**	**)**	**$**	*A*
0	s2	s3			1
1				accept	
2	s5	s6			4
3				$A \rightarrow$ **a**	
4			s7		
5	s5	s6			8
6			$A \rightarrow$ **a**		
7				$A \rightarrow (A)$	
8			s9		
9			$A \rightarrow (A)$		

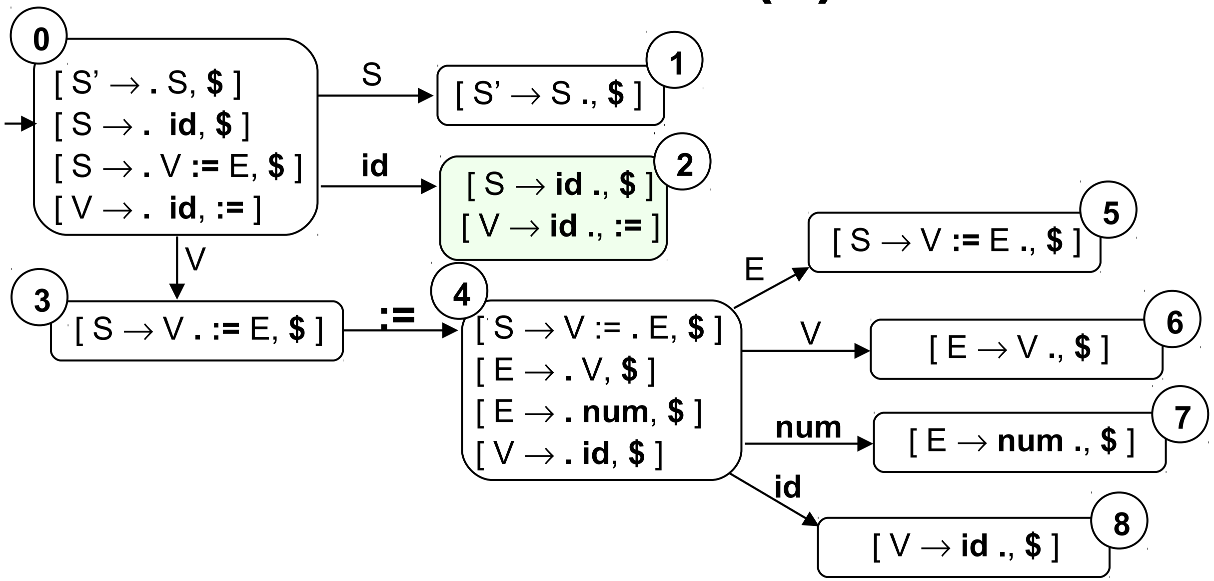
LR(1) Grammar

$S' \rightarrow S$
$S \rightarrow$ **id** | V **:=** E
$V \rightarrow$ **id**
$E \rightarrow V$ | **num**

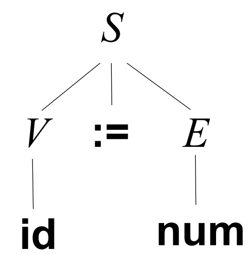
: <u>not</u> SLR(1) but LR(1) !



LR(1) Grammar (ii)



FOLLOW(*S'*) = { **$** }
FOLLOW(*S*) = { **$** }
FOLLOW(*E*) = { **$** }
FOLLOW(*V*) = { **$**, **:=** }



id := num

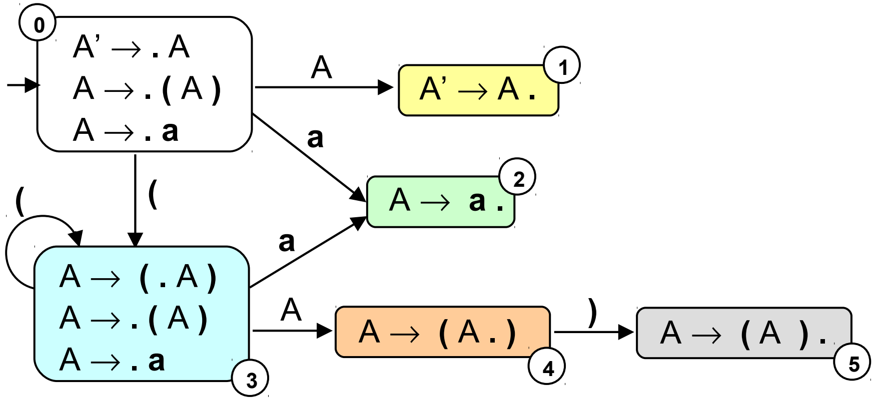
State	*Input*				*Goto*		
	id	**:=**	**num**	**$**	*S*	*V*	*E*
0	s2				1	3	
1				accept			
2		*V* → **id**		*S* → **id**			
3		s4					
4	s8		s7			6	5
5				*S* → *V* **:=** *E*			
6				*E* → *V*			
7				*E* → **num**			
8				*V* → **id**			

Stack	*Input*	*Action*
$0	id := num **$**	shift
$0 id 2	:= num **$**	*V* → **id**
$0 V 3	:= num **$**	shift
$0 V 3 := 4	num **$**	shift
$0 V 3 := 4 num 7	**$**	*E* → **num**
$0 V 3 := 4 E 5	**$**	*S* → *V* **:=** *E*
$0 S 1	**$**	accept

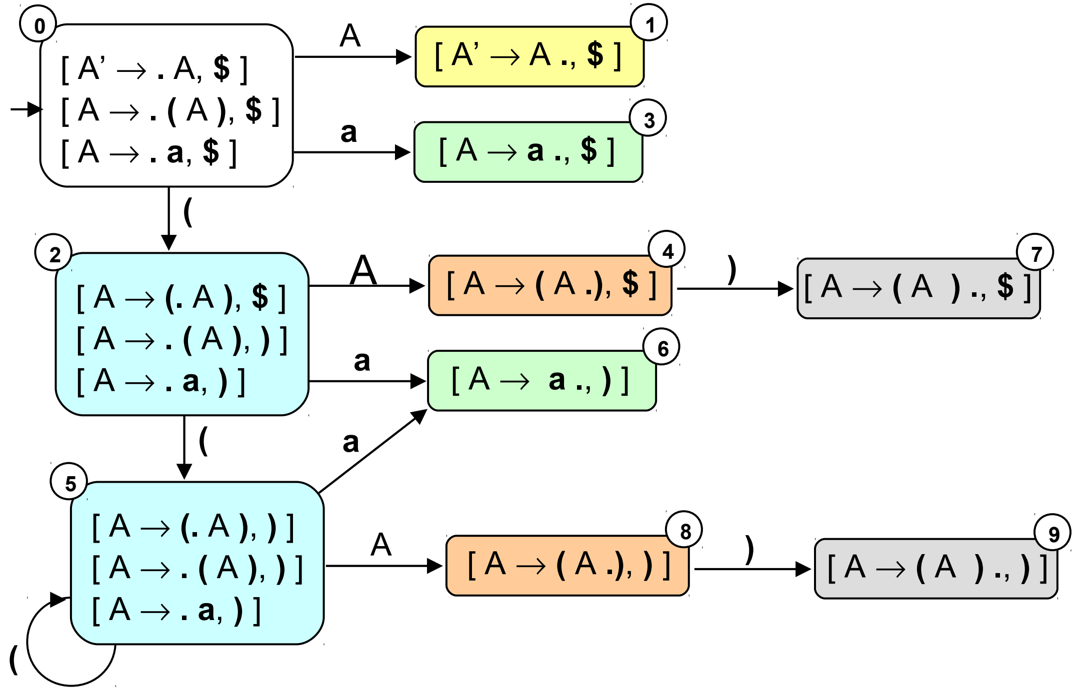
LALR(1) Parsing

<u>Note</u>: Often, hypertrophy of DFA of LR(1) items caused by different states that share the set of LR(0) items, but differ in the lookahead symbol

LR(0)



LR(1)



Factorization of the <u>core</u> part (LR(0)) of the state → aggregation of lookahead symbols

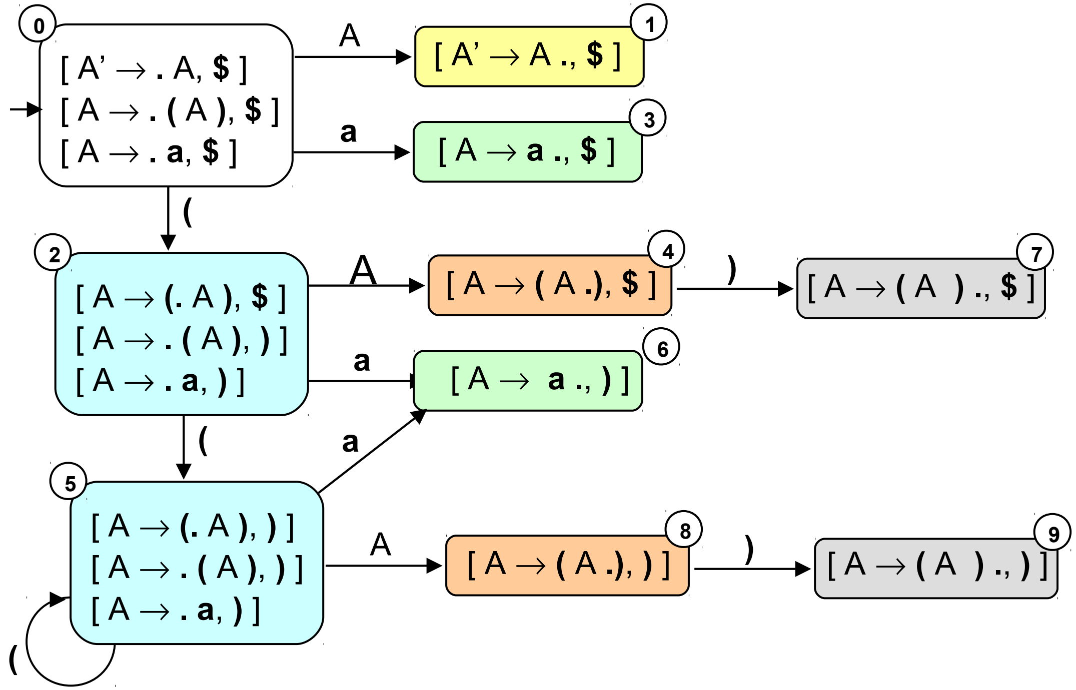
⇩

DFA(LALR(1)) = DFA(LR(0)) with the exception of the (new) lookahead part

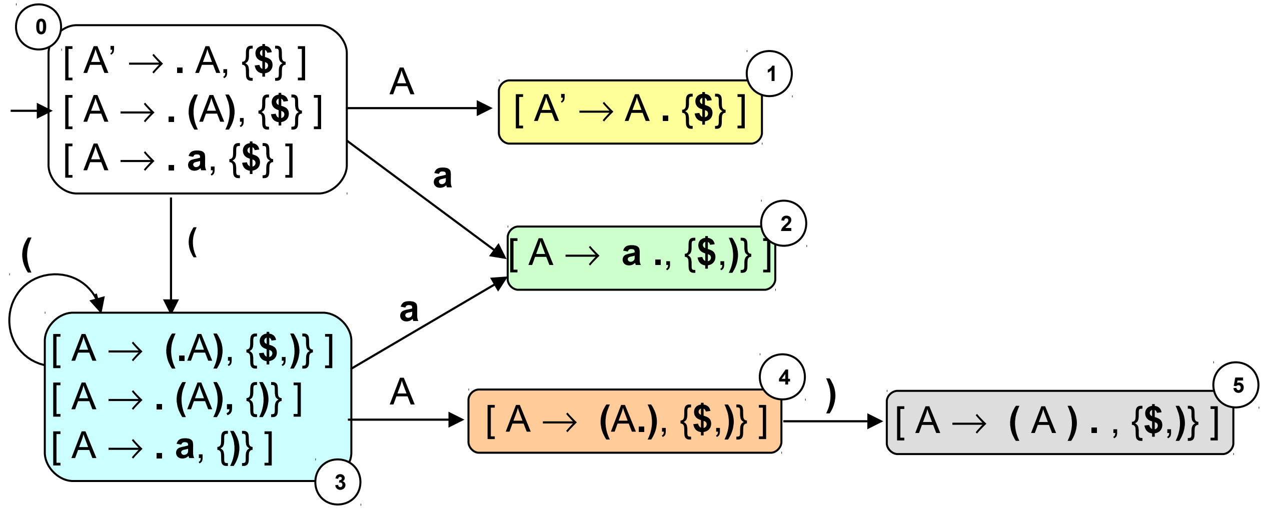
Possible to specify DFA of **LALR(1) items** ≡ $[\,A \rightarrow \alpha\,.\,\beta,\ \{ a_1, a_2, \ldots, a_n \}\,]$

LALR(1) Parsing (ii)

LR(1)



LALR(1)



LALR(1) Parsing Algorithm

```
stack := $0;  lookahead := first input symbol;
repeat
   s := Top(stack);
   if [A → β.Xδ, Λ] ∈ s and Terminal(X) and X = lookahead then
      Shift lookahead on the stack;
      Push(s'), where s –X→ s' is a transition in DFA
   else if  [A → η., Λ] ∈ s  and lookahead ∈ Λ then
      Reduce A → η;
      if A → η = S' → S then
        Accept
      else
         Remove η with its states from the stack;      /* η is on top of the stack by construction */
         s' := Top(stack);
         Push(A); Push(s''), where s' –A→ s'' is a transition in DFA
   else Error()
until acceptance or error.
```

Def: G is LALR(1) if $\forall$ s of DFA:

1. $\forall$ [$A \to \alpha.X\beta, \Lambda$] $\in$ s, Terminal(X) ($\neg$([$B \to \gamma., \Lambda'$] $\in$ s, $X \in \Lambda'$));
2. $\neg$ ([$A \to \alpha., \Lambda$] $\in$ s, [$B \to \beta., \Lambda'$] $\in$ s, $\Lambda \cap \Lambda' \neq \emptyset$).

Notes and Properties

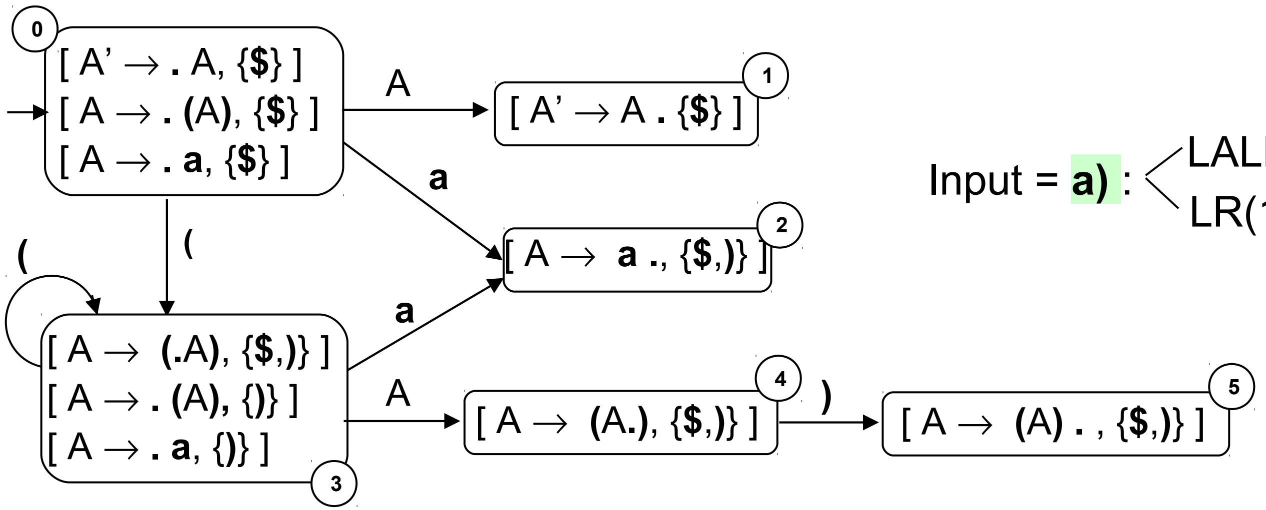
1. May ∃ conflicts in LALR(1) which ∄ in LR(1) (<u>rare</u> in practice)

2. G is LR(1) → LALR(1) parsing table (which could include conflicts) <u>cannot</u> have shift/reduce conflicts

$S \rightarrow$ **id** | V **:=** E
$V \rightarrow$ **id**
$E \rightarrow V$ | **num**

not SLR(1) but LALR(1) → DFA(LR(1)) = DFA(LALR(1)) (not factorizable)

3. If G is LALR(1) ⇒ G also LR(1): difference wrt LR(1) parsing = possible some spurious reductions before error declaration



Input = **a)** : LALR(1): error <u>after</u> reduction A→**a**
LR(1): error <u>before</u> reduction A→**a**

4. Possible direct construction of LALR(1) DFA starting from LR(0) DFA