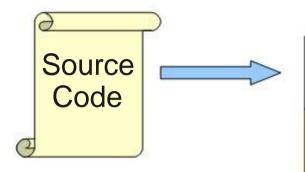
Bottom-Up Parsing III

Where We Are



Lexical Analysis

Syntax Analysis

Semantic Analysis

IR Generation

IR Optimization

Code Generation

Optimization



Machine Code

Constructing LALR(1) Automata

 It's not a good idea to build LALR(1) automata from LR(1) automata.

Why?

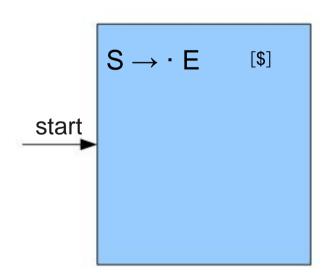
- LR(1) automata are impractically large.
- Are there more efficient methods for LALR(1) automata construction?
- Yes; we'll see two.

The "Lazy Merging" Technique

- Idea: Merge together LR(1) states as they're generated.
- Maintain a work list of states to process; begin with the initial LR(1) state.
- When adding a new state, if it has the same core as an old state, update the old state and put it back in the worklist.

 $\mathsf{R} \to \mathsf{L}$

LALR(1) Construction



```
S \rightarrow E

E \rightarrow L = R

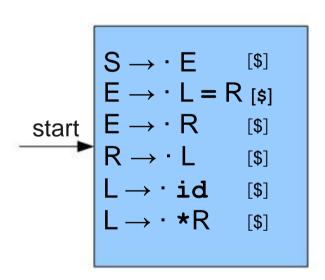
E \rightarrow R

L \rightarrow id

L \rightarrow *R
```

 $R \rightarrow L$

LALR(1) Construction



```
S \rightarrow E

E \rightarrow L = R

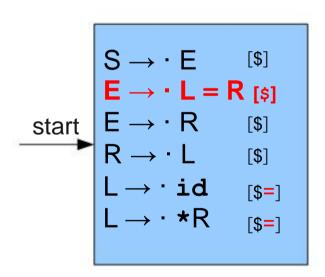
E \rightarrow R

L \rightarrow id

L \rightarrow *R
```

 $R \rightarrow L$

LALR(1) Construction

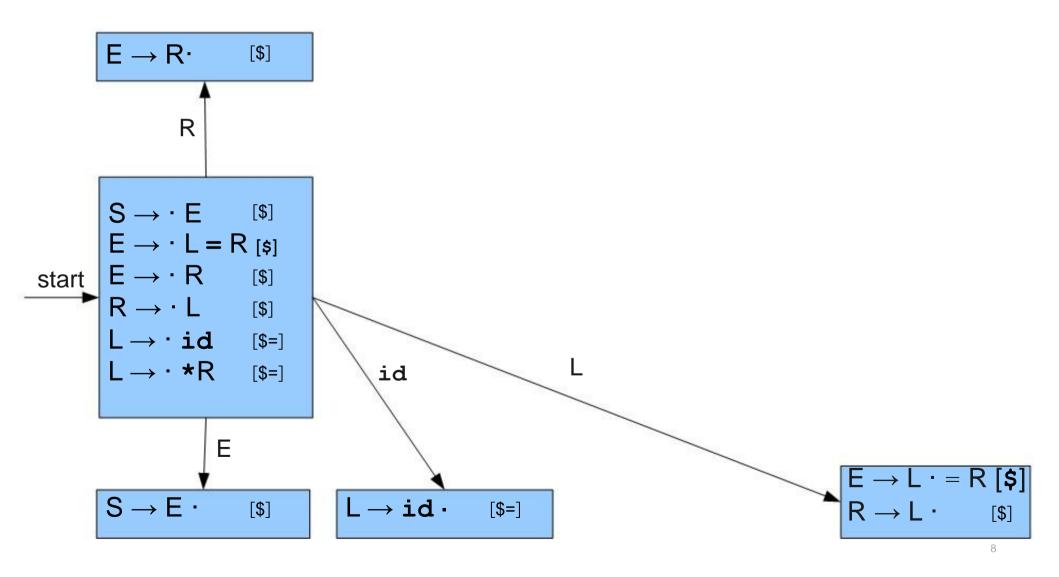


 $\begin{array}{l} S \rightarrow E \\ E \rightarrow L = R \\ E \rightarrow R \\ L \rightarrow \text{id} \end{array}$

 $L\to \star R$

 $R \rightarrow L$

LALR(1) Construction



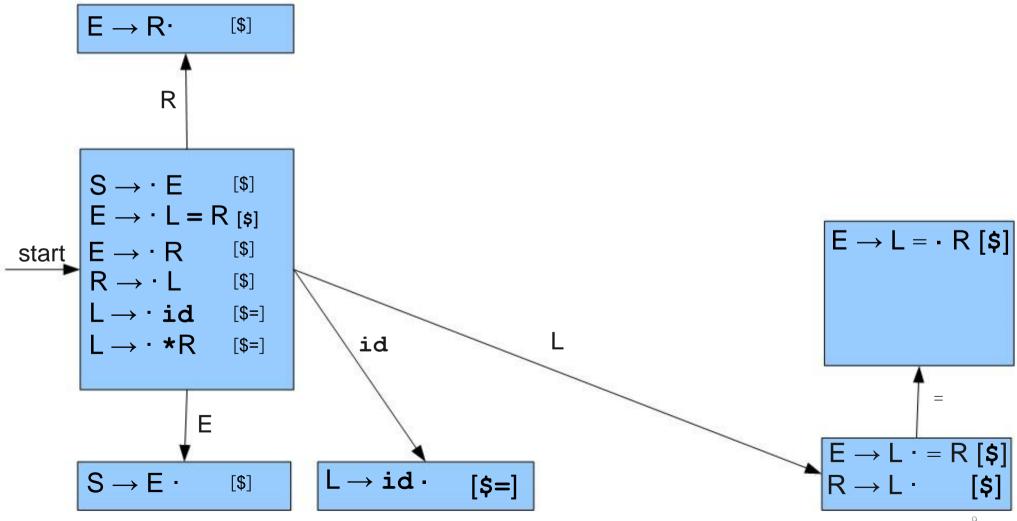
 $\begin{array}{l} S \rightarrow E \\ E \rightarrow L = R \\ E \rightarrow R \\ L \rightarrow \text{id} \end{array}$

 $L \rightarrow *R$

 $R \rightarrow L$

LALR(1) Construction

Lazy Merging



9

```
S \rightarrow E

E \rightarrow L = R

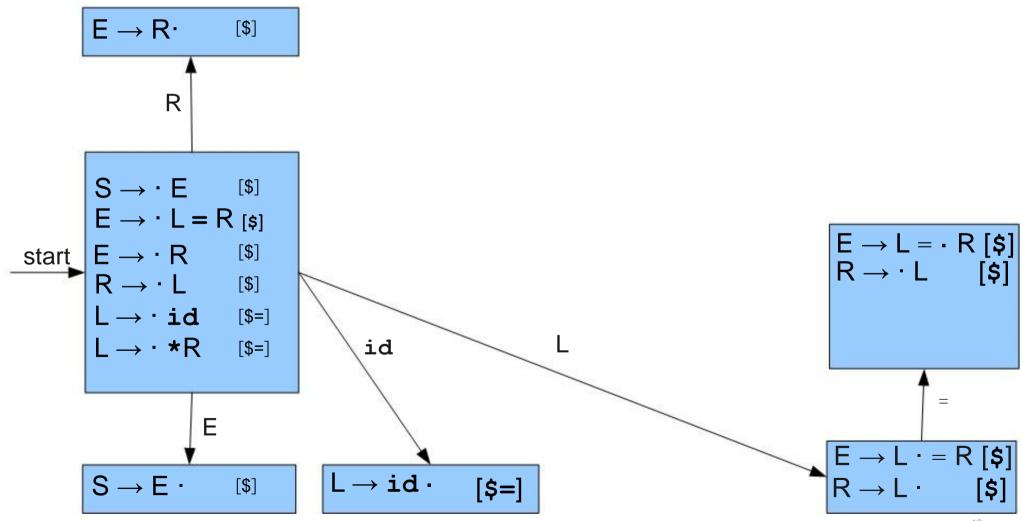
E \rightarrow R

L \rightarrow id

L \rightarrow *R
```

 $R \rightarrow L$

LALR(1) Construction



```
S \rightarrow E

E \rightarrow L = R

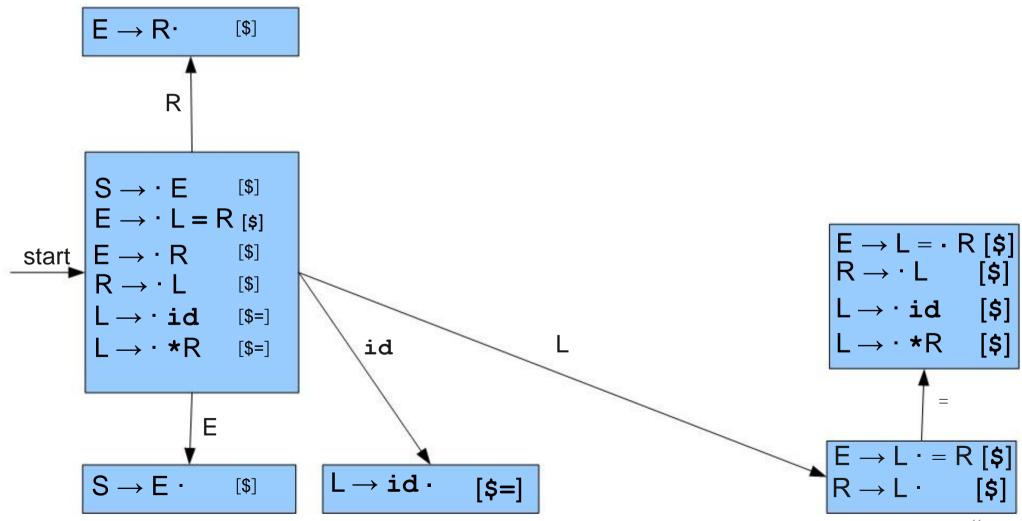
E \rightarrow R

L \rightarrow id

L \rightarrow *R
```

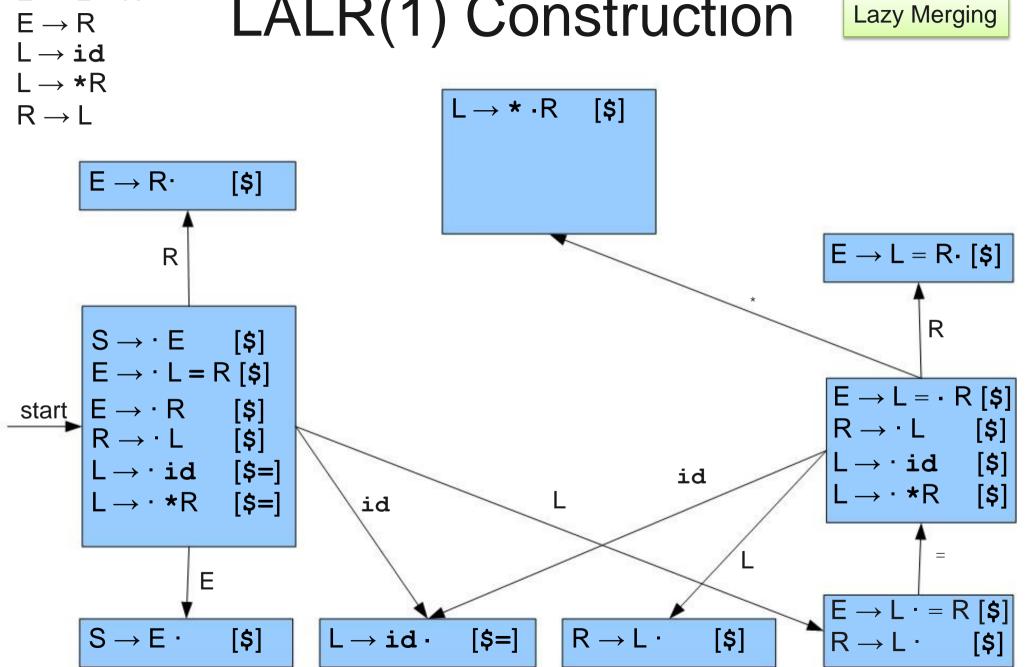
 $R \rightarrow L$

LALR(1) Construction



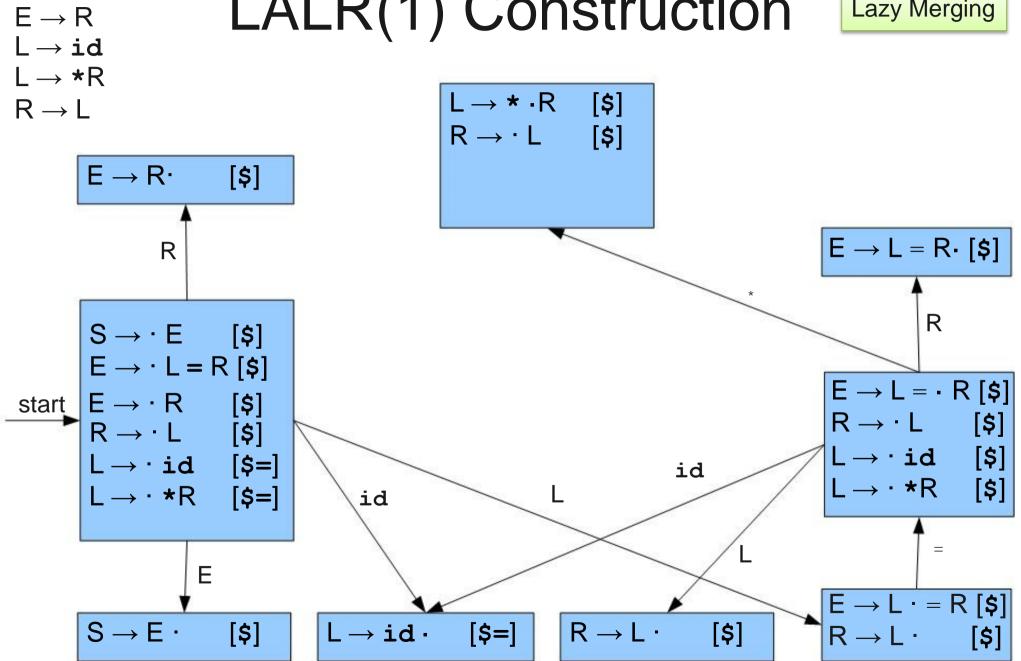
 $S \rightarrow E$ $E \rightarrow L = R$

LALR(1) Construction



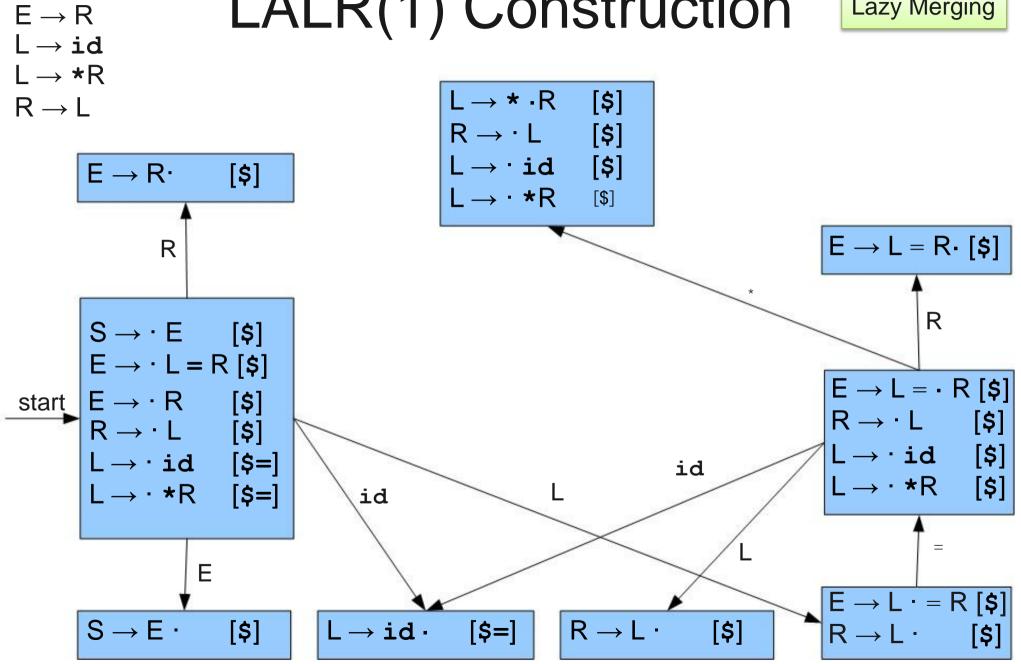
```
S \rightarrow E
E \rightarrow L = R
```

LALR(1) Construction



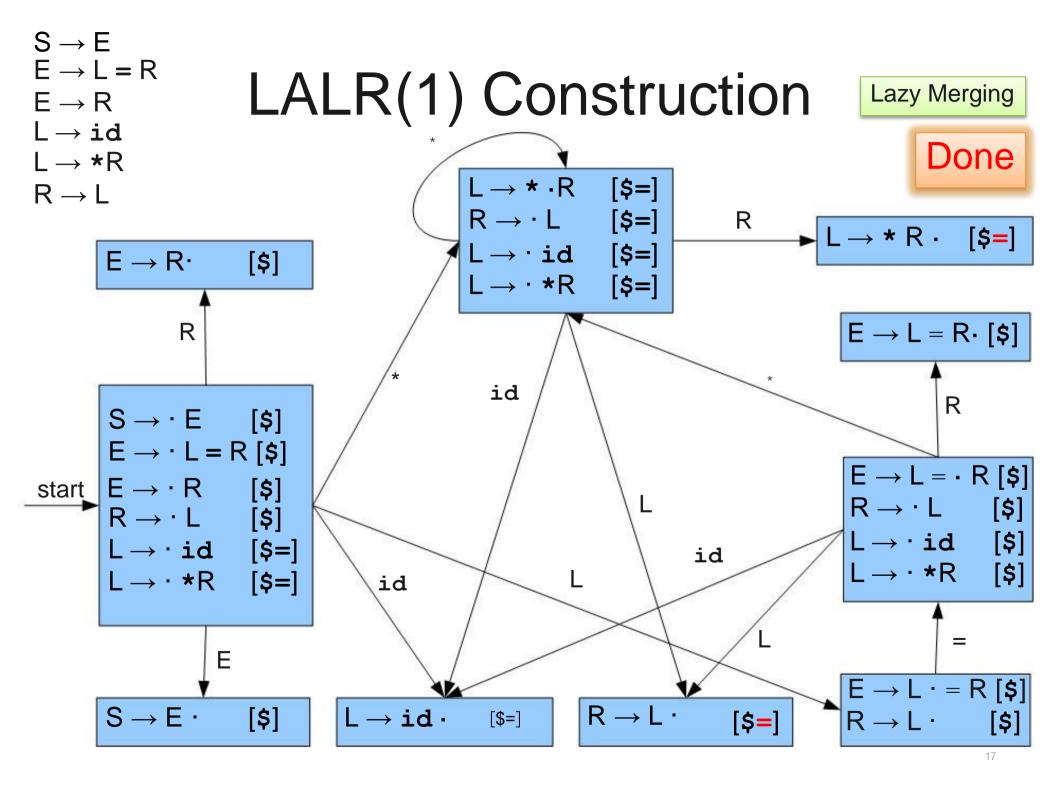
```
S \rightarrow E
E \rightarrow L = R
```

LALR(1) Construction



 $S \rightarrow E$ $E \rightarrow L = R$ LALR(1) Construction Lazy Merging $\mathsf{E} \to \mathsf{R}$ $L \to \text{id}$ $L \rightarrow *R$ L → * -R [\$] $R \rightarrow L$ $R \rightarrow L$ [\$] R $L \rightarrow *R$ [\$] [\$] $L \rightarrow \cdot id$ $E \rightarrow R$ [\$] $L \rightarrow *R$ [\$] $E \rightarrow L = R \cdot [\$]$ id R $S \rightarrow \cdot E$ [\$] $E \rightarrow L = R [\$]$ $E \rightarrow L = R$ $\mathsf{E} \to \mathsf{R}$ start [\$] $R \rightarrow \cdot L$ [\$] $R \rightarrow L$ [\$] [\$] $L \rightarrow \cdot id$ $L \rightarrow \cdot id$ [\$=] id $L \rightarrow *R$ [\$] L → • *R [\$=] id Ε $E \rightarrow L \cdot = R [\$]$ [\$] [\$=] $R \rightarrow L$ $S \rightarrow E$ [\$] $L \rightarrow id$ $R \rightarrow L$ [\$]

 $S \rightarrow E$ $E \rightarrow L = R$ LALR(1) Construction Lazy Merging $\mathsf{E} \to \mathsf{R}$ $L \to \text{id}$ $L \rightarrow *R$ $L \rightarrow * \cdot R$ [\$=] $R \rightarrow L$ $R \rightarrow L$ [\$=] R $L \rightarrow *R$ $L \rightarrow \cdot id$ [\$=] $E \rightarrow R$ [\$] $L \rightarrow \cdot *R$ [\$=] $E \rightarrow L = R \cdot [\$]$ id R $S \rightarrow \cdot E$ [\$] $E \rightarrow L = R [\$]$ $E \rightarrow L = R$ $\mathsf{E} \to \mathsf{R}$ start $R \rightarrow L$ [\$] $R \rightarrow \cdot L$ [\$] [\$] $L \rightarrow \cdot id$ $L \rightarrow \cdot id$ [\$=] id $L \rightarrow *R$ [\$] $L \rightarrow *R$ [\$=] id Ε $E \rightarrow L \cdot = R [\$]$ $R \rightarrow L$. [\$] $S \rightarrow E$ $L \rightarrow id$ [\$=] [\$] $R \rightarrow L$ [\$]



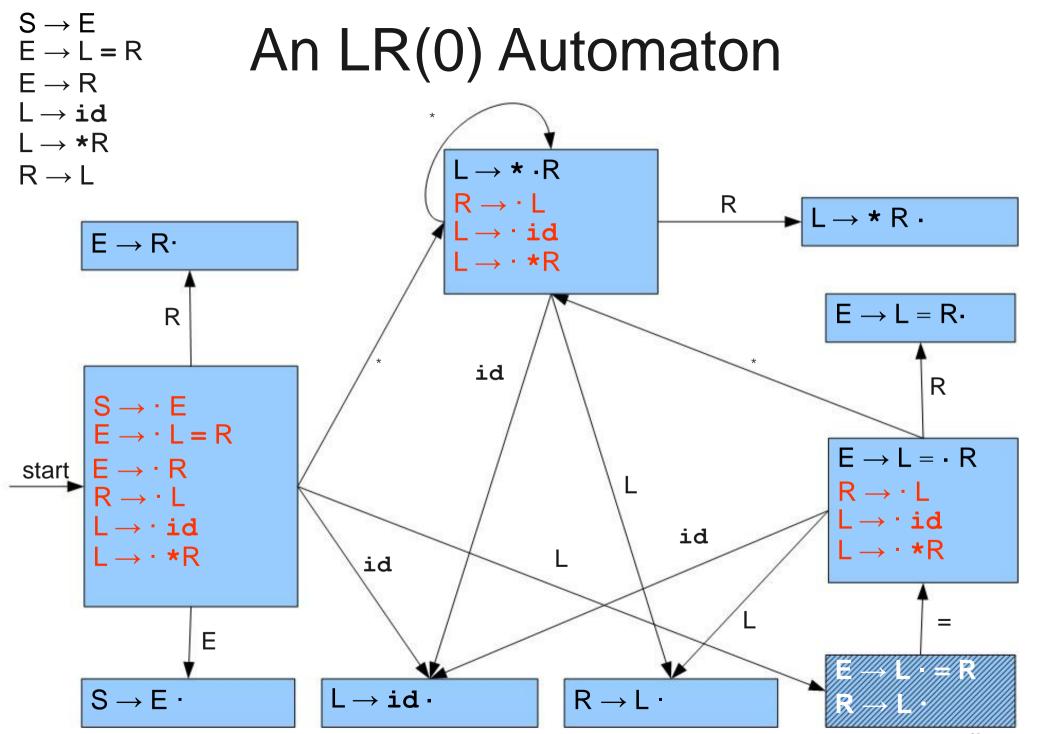
Analysis of Lazy Merging Algorithm

- Since we merge as we go, size of the partial automaton never exceeds size of overall automaton.
- However, this algorithm could be slow in practice.

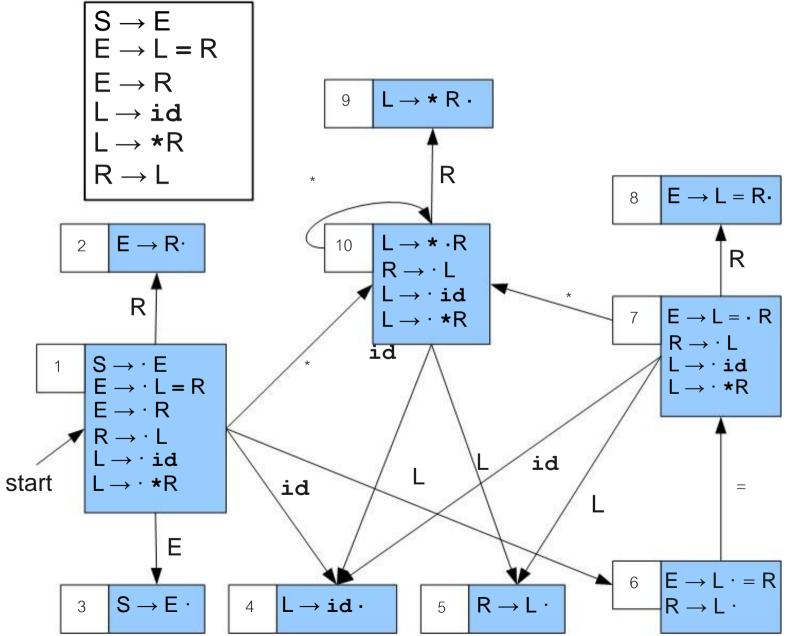
Second Technique: Fast LALR(1) Construction

- Fast and simple construction of LALR(1) lookaheads.
- Four steps:
 - Construct the LR(0) automaton for the grammar.
 - Construct the augmented grammar by replacing nonterminals with new nonterminals based on the LR(0) transitions.
 - Compute the FOLLOW sets for these nonterminals.
 - Propagate changes through the LR(0) automaton to create LOOKAHEAD sets by using the FOLLOW sets.

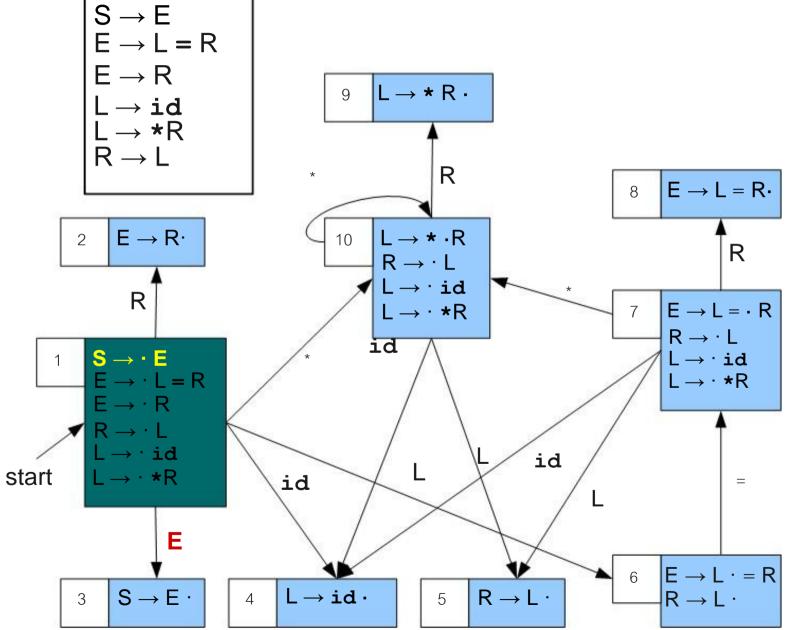
(Instead of using Follow sets of the original grammar for reduction like SLR(1), LALR(1) uses lookahead set which is the Follow sets of augmented grammar)

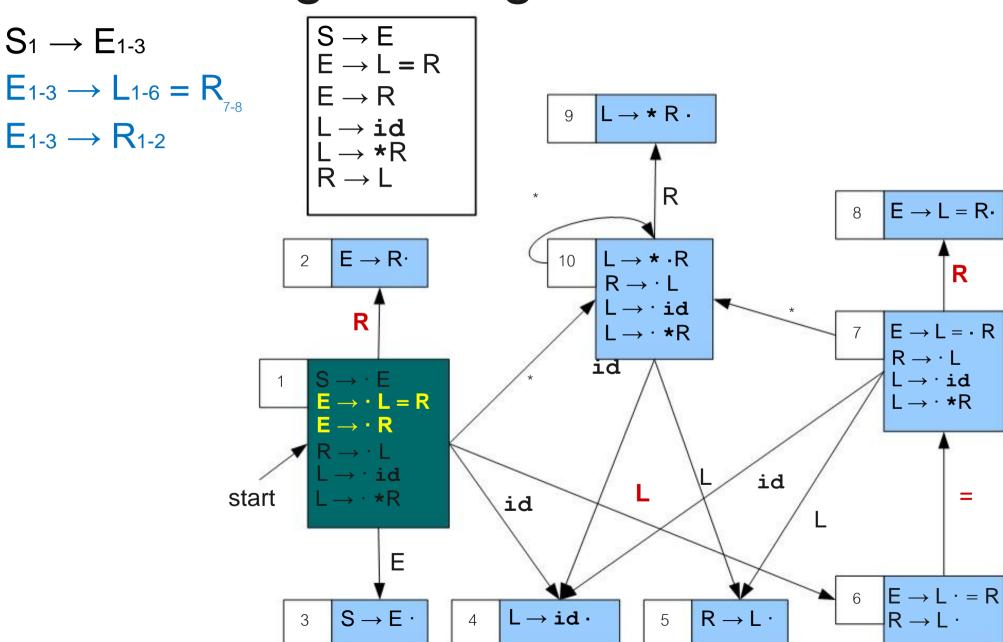


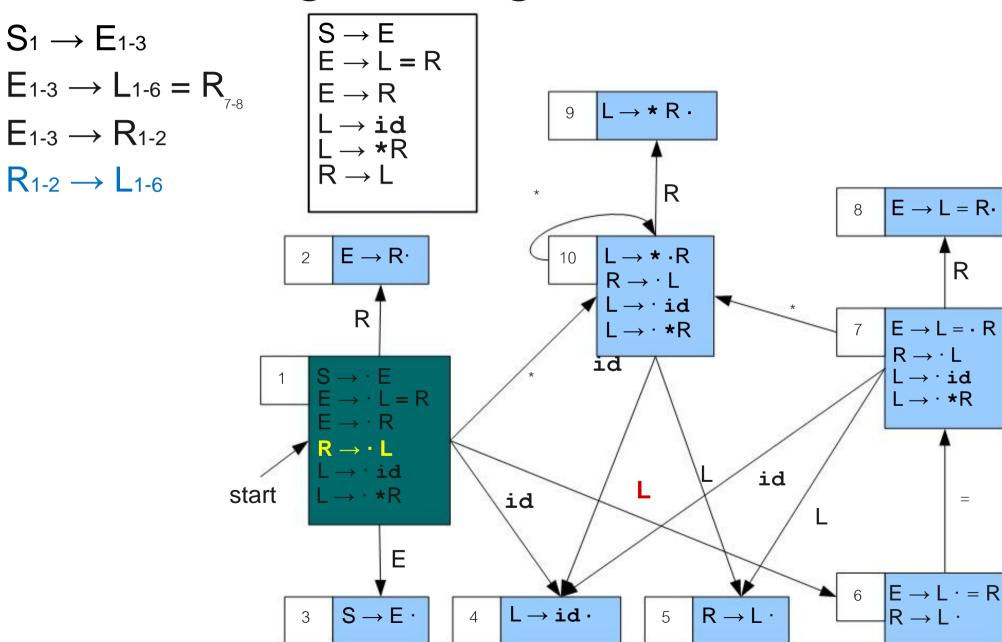
Note: We augment only productions which have a dot in front.

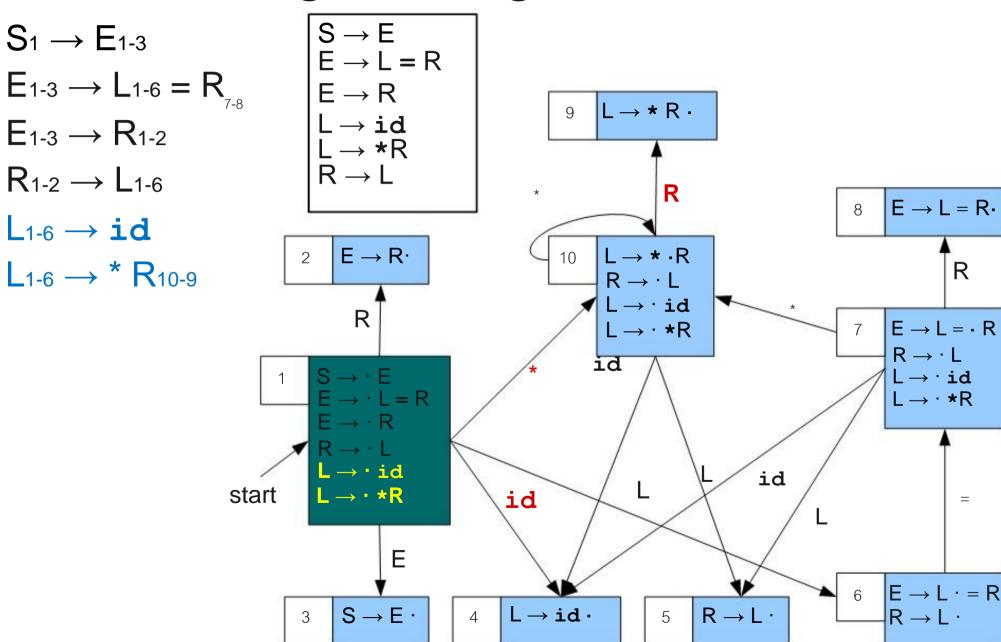


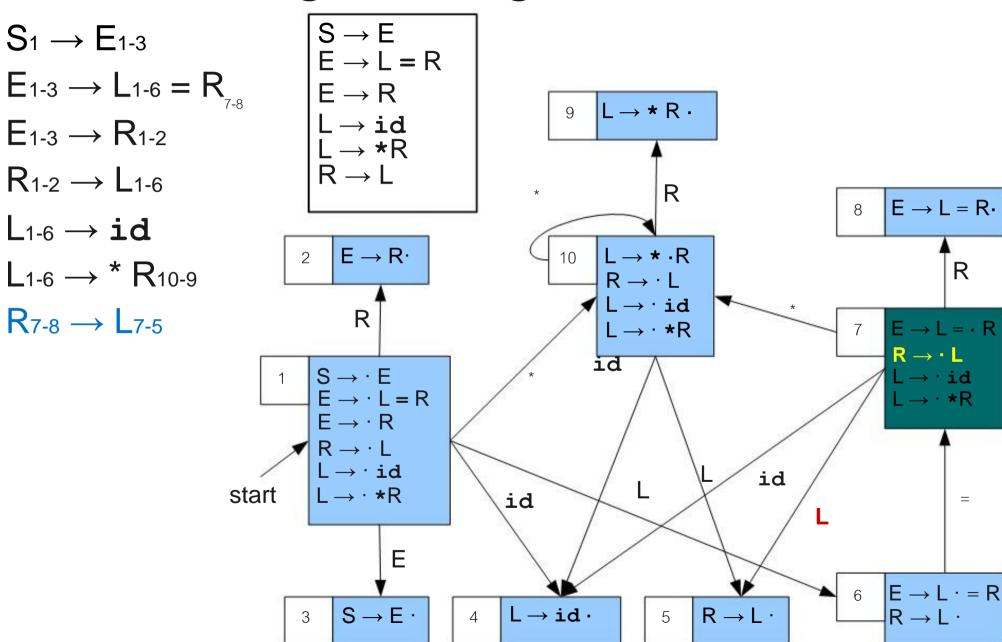
 $S_1 \rightarrow E_{1-3}$

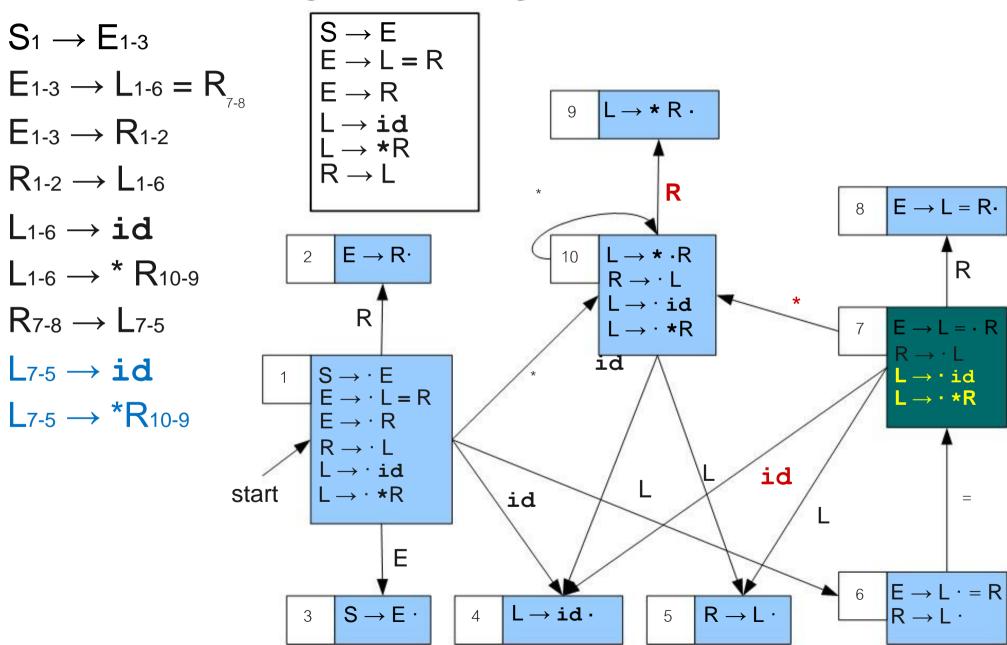


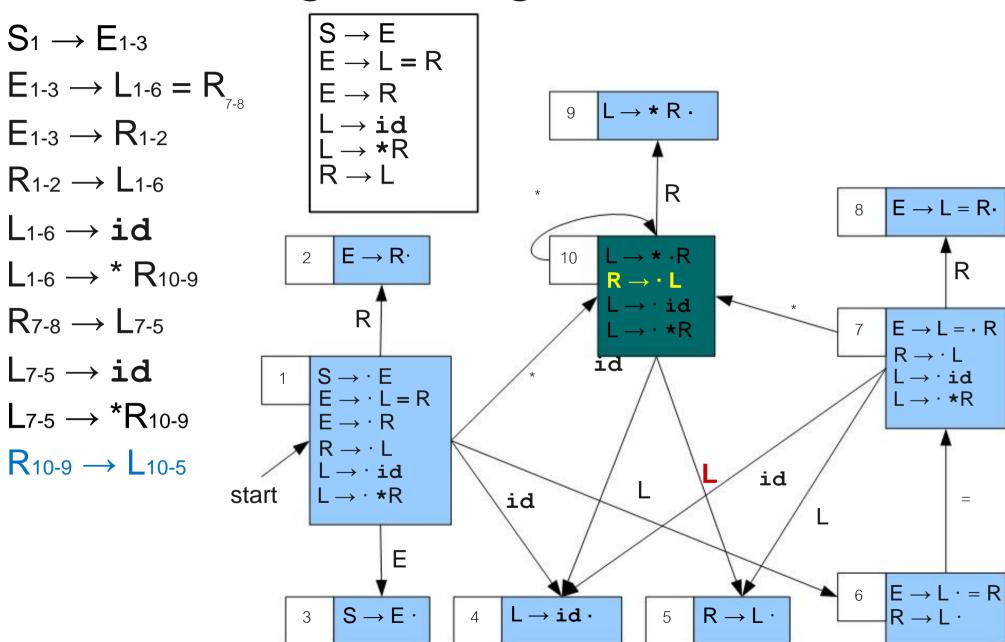


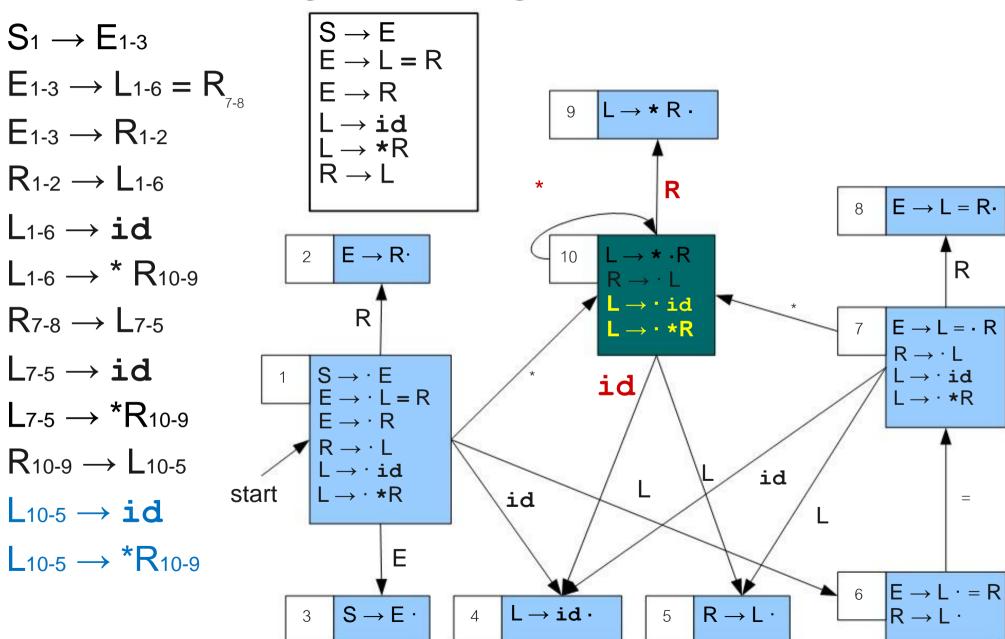










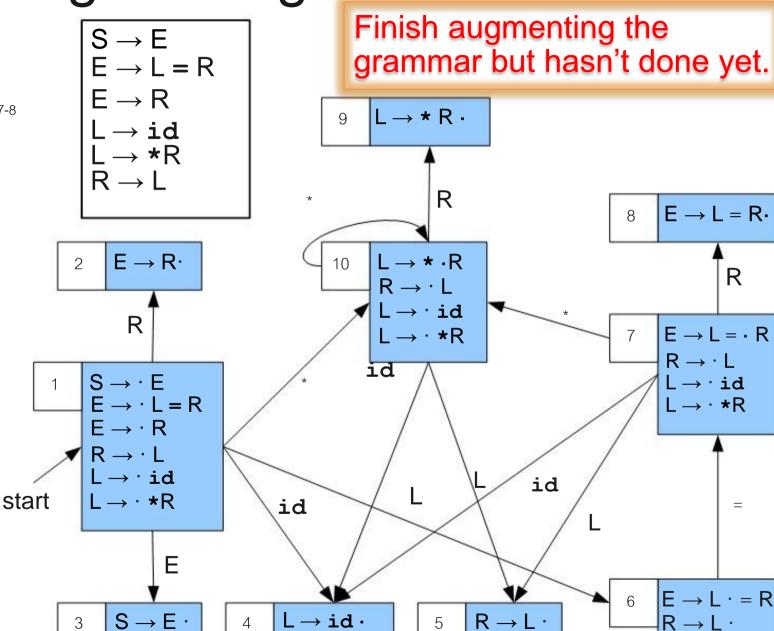


$$S_1 \rightarrow E_{1-3}$$

 $E_{1-3} \rightarrow L_{1-6} = R_{7-8}$
 $E_{1-3} \rightarrow R_{1-2}$
 $R_{1-2} \rightarrow L_{1-6}$
 $L_{1-6} \rightarrow \mathbf{id}$
 $L_{1-6} \rightarrow \mathbf{R}_{10-9}$
 $R_{7-8} \rightarrow L_{7-5}$
 $L_{7-5} \rightarrow \mathbf{id}$
 $L_{7-5} \rightarrow \mathbf{r}_{10-9}$
 $R_{10-9} \rightarrow L_{10-5}$

 $L_{10-5} \rightarrow id$

 $L_{10-5} \rightarrow *R_{10-9}$



Lookahead Sets

$$S_1 \rightarrow E_{1-3}$$

$$E_{1-3} \rightarrow L_{1-6} = R_{7-8}$$

$$E_{1-3} \rightarrow R_{1-2}$$

$$R_{1-2} \rightarrow L_{1-6}$$

$$L_{1-6} \rightarrow id$$

$$L_{1-6} \rightarrow * R_{10-9}$$

$$R_{7-8} \rightarrow L_{7-5}$$

$$L_{7-5} \rightarrow id$$

$$L_{7-5} \rightarrow *R_{10-9}$$

$$R_{10-9} \rightarrow L_{10-5}$$

$$L_{10-5} \rightarrow id$$

$$L_{10-5} \rightarrow *R_{10-9}$$

Now, we are going to build Lookahead sets from the FOLLOW sets of the augmented grammar.

Lookahead Sets

$$S_1 \rightarrow E_{1-3}$$

$$E_{1-3} \rightarrow L_{1-6} = R_{7-8}$$

$$E_{1-3} \rightarrow R_{1-2}$$

$$R_{1-2} \rightarrow L_{1-6}$$

$$L_{1-6} \rightarrow id$$

$$L_{1-6} \rightarrow * R_{10-9}$$

$$R_{7-8} \rightarrow L_{7-5}$$

$$L_{7-5} \rightarrow id$$

$$L_{7-5} \rightarrow *R_{10-9}$$

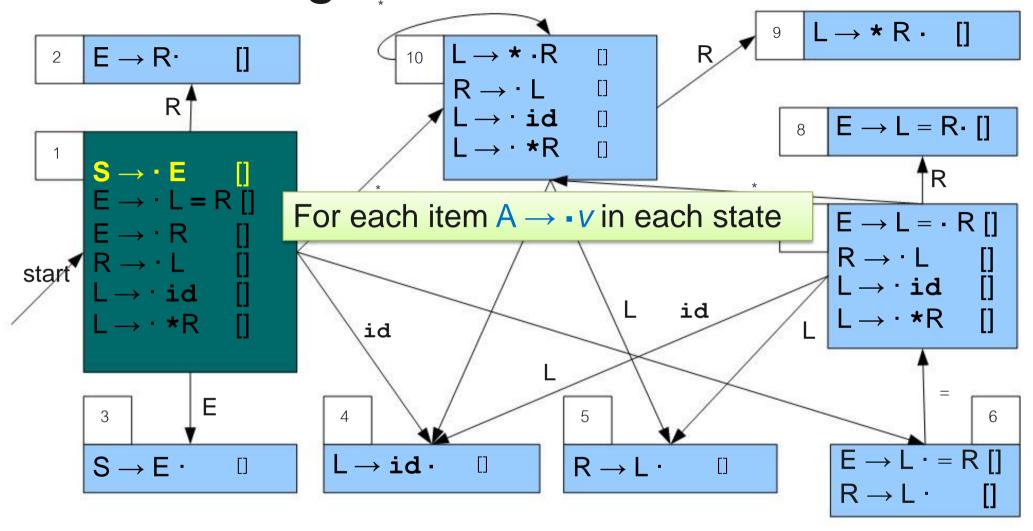
$$R_{10-9} \rightarrow L_{10-5}$$

$$L_{10-5} \rightarrow id$$

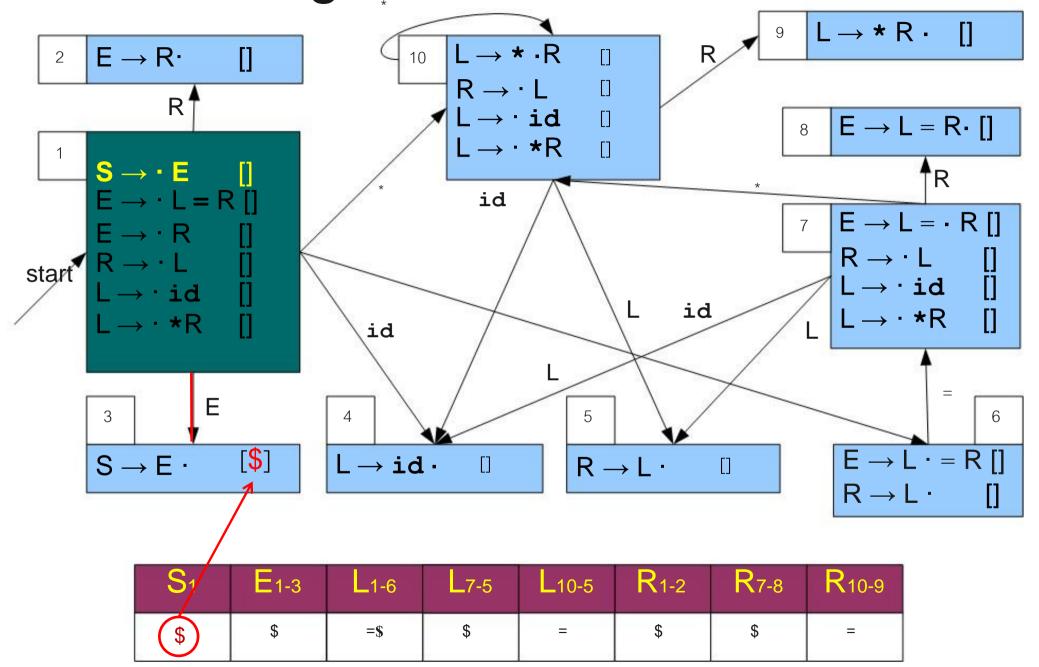
$$L_{10-5} \rightarrow *R_{10-9}$$

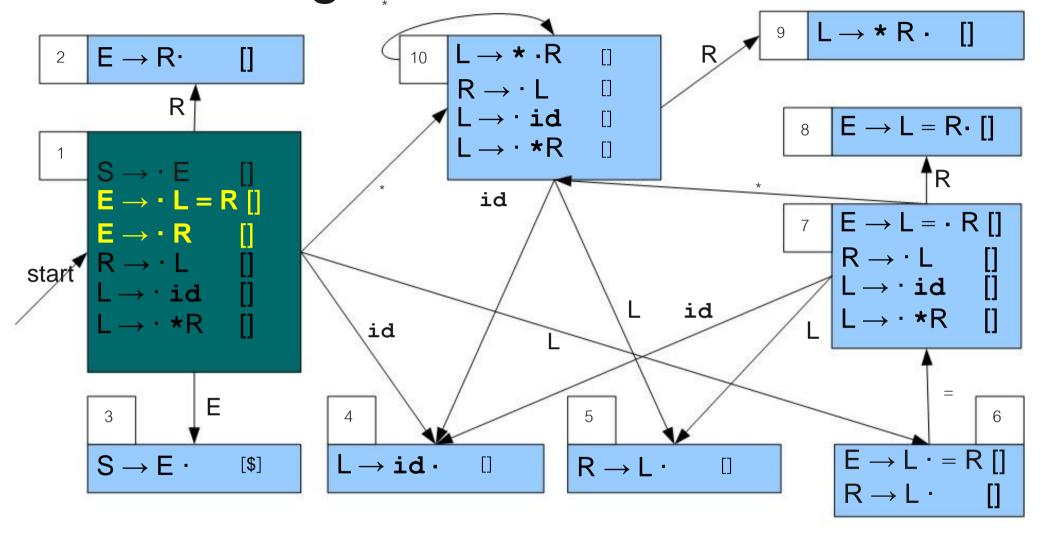
S ₁	E ₁₋₃	L ₁₋₆	7-5	L ₁₀₋₅	R ₁₋₂	R ₇₋₈	R ₁₀₋₉
\$	\$	= \$	\$	=	\$	\$	= \$

Lookahead Sets = Follow sets of augmented grammar.

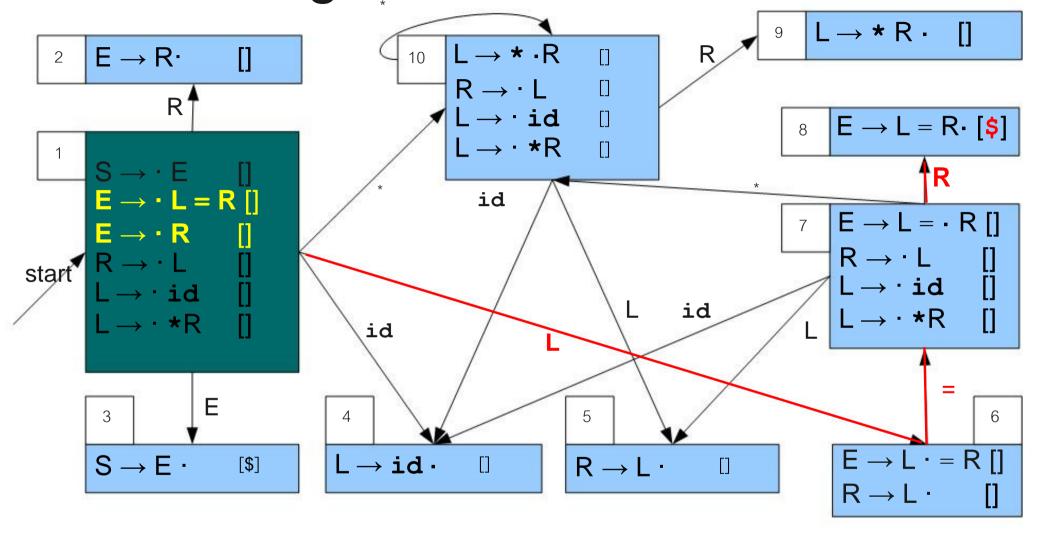


S ₁	E ₁₋₃	L1-6	L7-5	L ₁₀₋₅	R ₁₋₂	R ₇₋₈	R ₁₀₋₉
\$	\$	=\$	\$	=	\$	\$	=

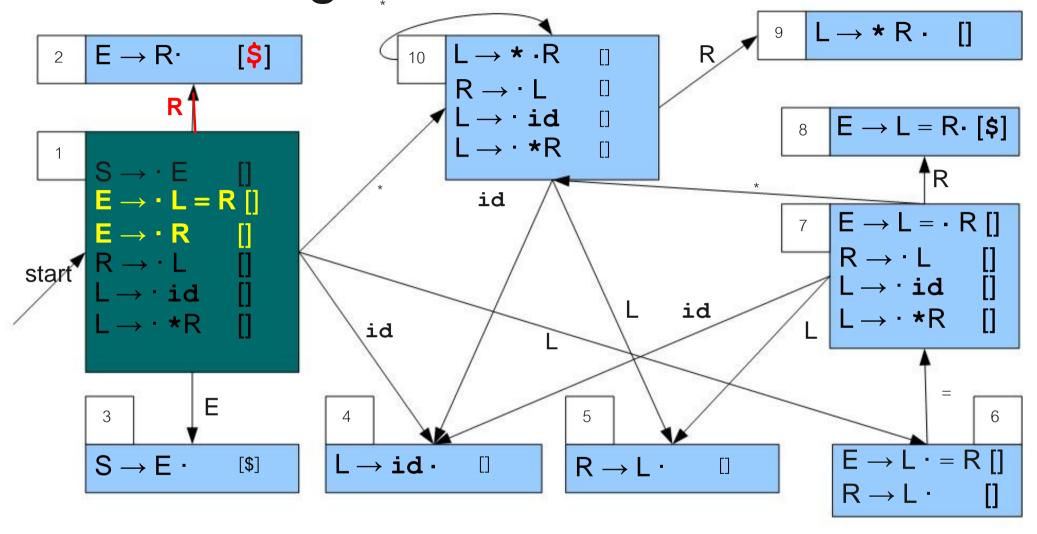




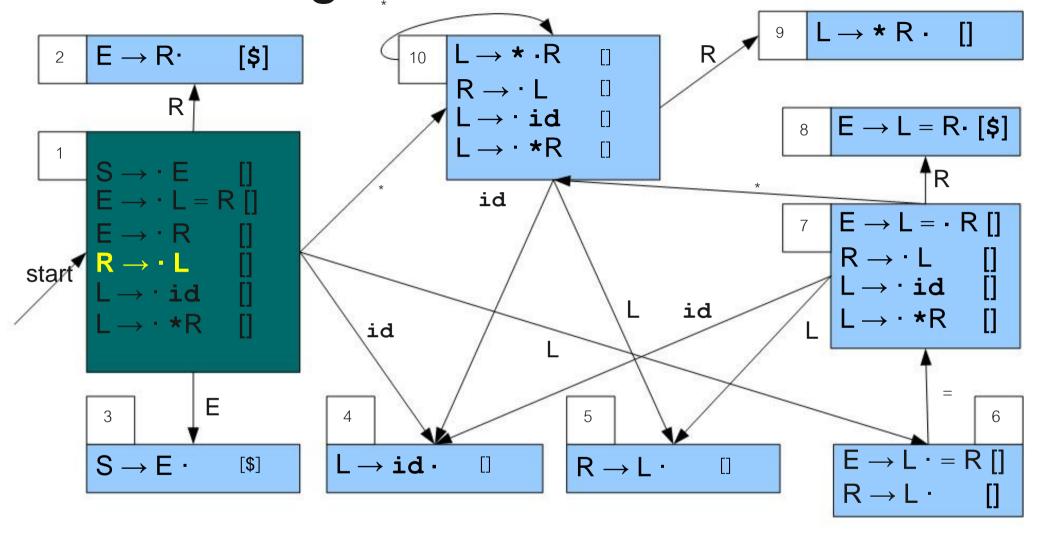
S ₁	E ₁₋₃	L1-6	L7-5	L10-5	R ₁₋₂	R ₇₋₈	R ₁₀₋₉
\$	\$	=\$	\$	=	\$	\$	=



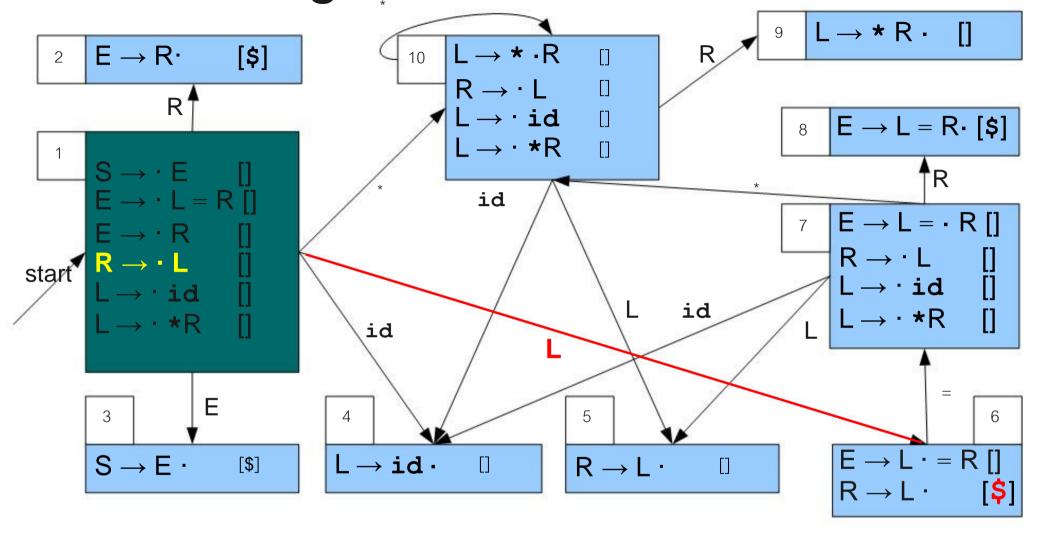
S ₁	E ₁₋₃	L1-6	L7-5	L10-5	R ₁₋₂	R ₇₋₈	R ₁₀₋₉
\$	\$	=\$	\$	=	\$	\$	=



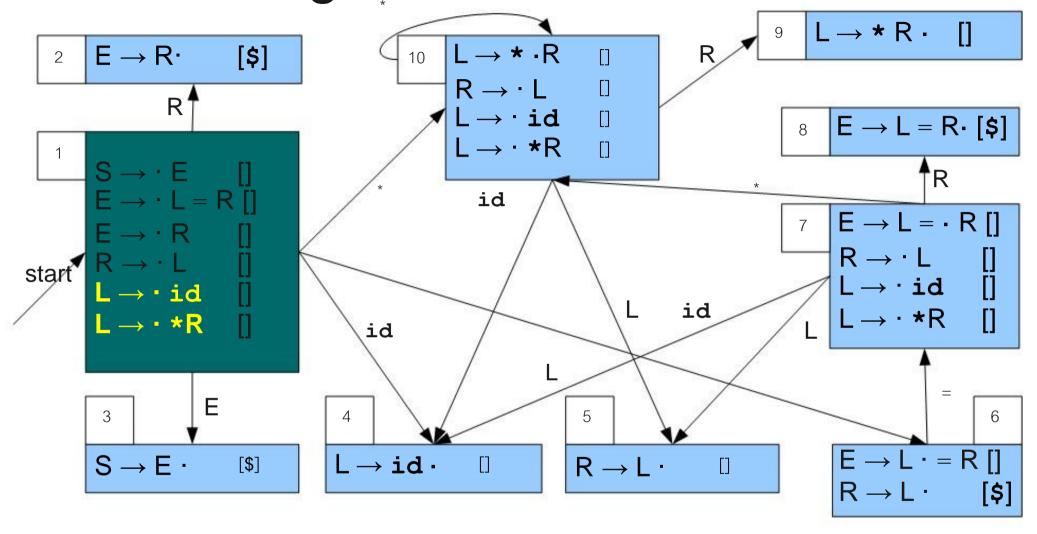
S ₁	E ₁₋₃	L1-6	L ₇₋₅	L ₁₀₋₅	R ₁₋₂	R ₇₋₈	R ₁₀₋₉
\$	\$	=\$	\$	=	\$	\$	=



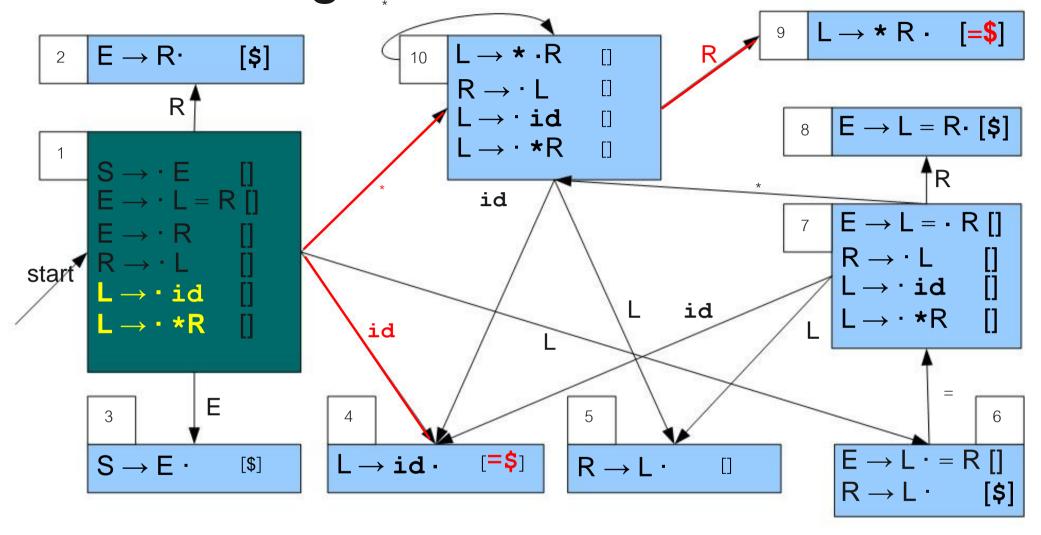
S ₁	E ₁₋₃	L ₁₋₆	L ₇₋₅	L ₁₀₋₅	R ₁₋₂	R ₇₋₈	R ₁₀₋₉
\$	\$	=\$	\$	=	\$	\$	=



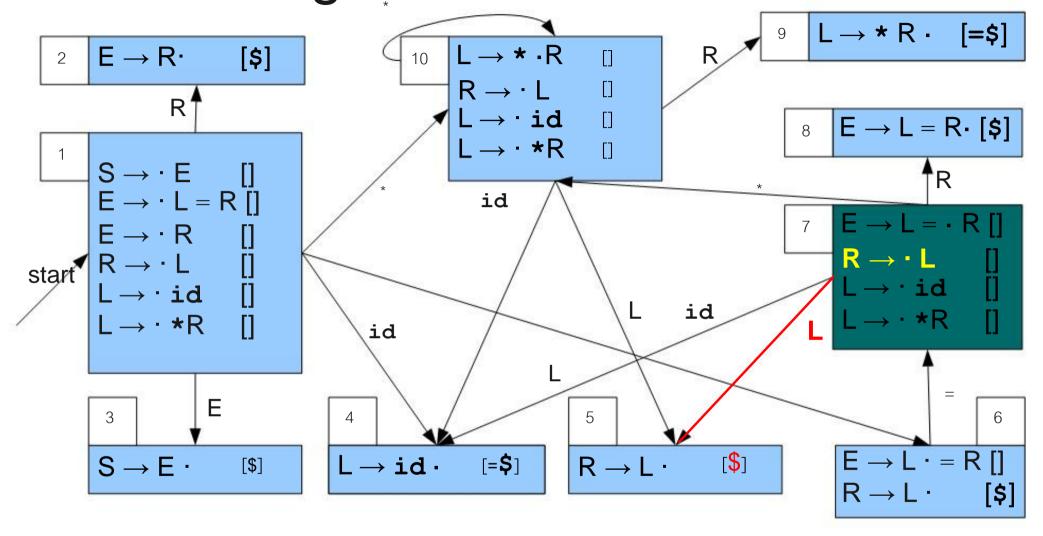
S ₁	E ₁₋₃	L1-6	L7-5	L ₁₀₋₅	R ₁₋₂	R ₇₋₈	R ₁₀₋₉
\$	\$	=\$	\$	=	\$	\$	=



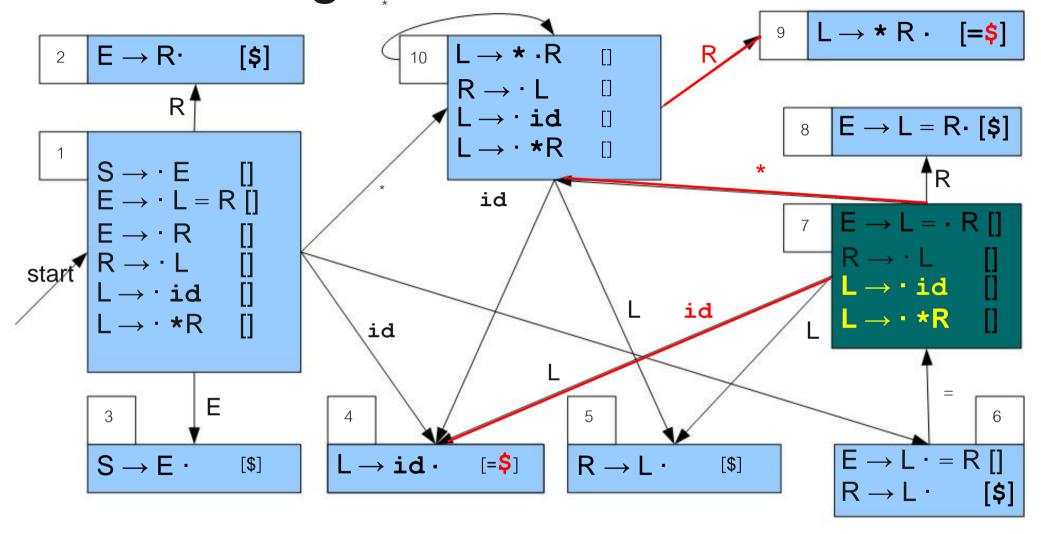
S ₁	E ₁₋₃	L1-6	L7-5	L10-5	R ₁₋₂	R ₇₋₈	R ₁₀₋₉
\$	\$	=\$	\$	=	\$	\$	=



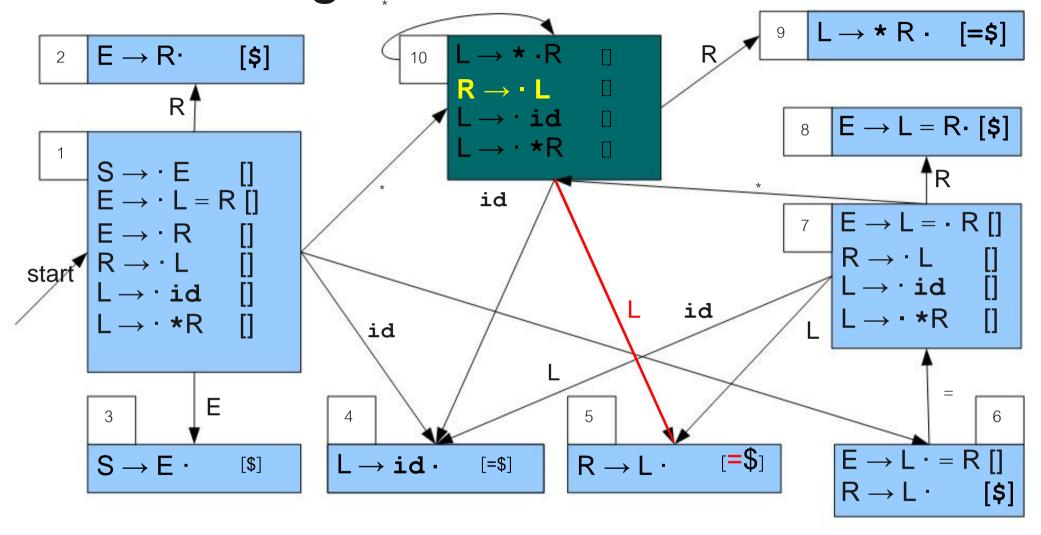
S ₁	E ₁₋₃	L1-6	L ₇₋₅	L ₁₀₋₅	R ₁₋₂	R ₇₋₈	R ₁₀₋₉
\$	\$	=\$	\$	=	\$	\$	=



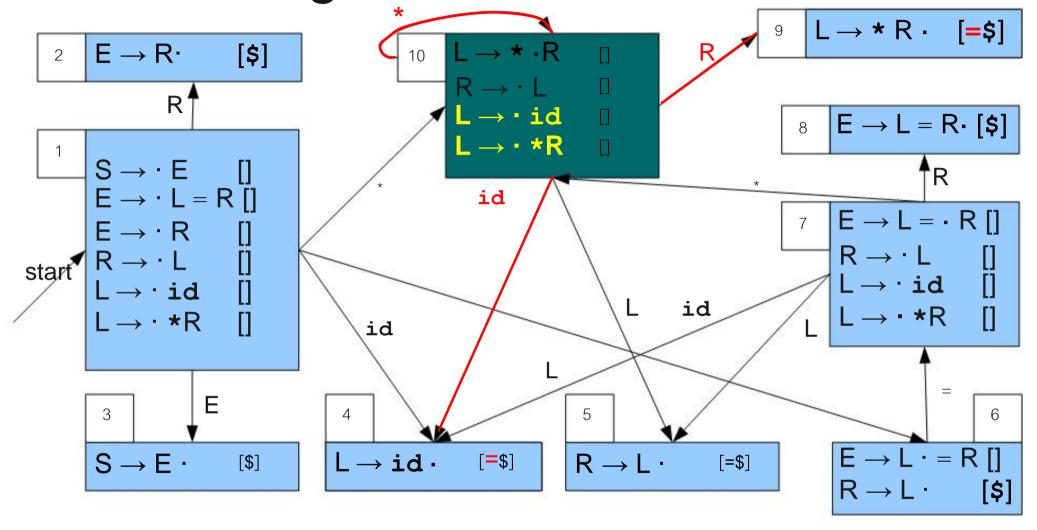
S ₁	E ₁₋₃	L1-6	L ₇₋₅	L ₁₀₋₅	R ₁₋₂	R ₇₋₈	R ₁₀₋₉
\$	\$	=\$	\$	=	\$	\$	=



S ₁	E ₁₋₃	L ₁₋₆	L ₇₋₅	L ₁₀₋₅	R ₁₋₂	R ₇₋₈	R ₁₀₋₉
\$	\$	=\$	\$	=	\$	\$	=

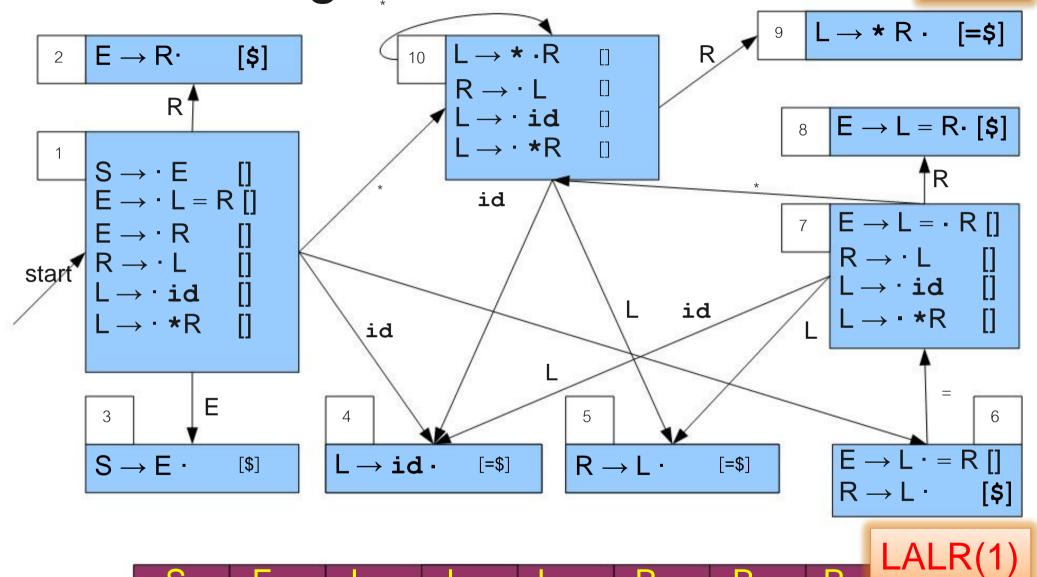


S ₁	E ₁₋₃	L1-6	L ₇₋₅	L ₁₀₋₅	R ₁₋₂	R ₇₋₈	R ₁₀₋₉
\$	\$	=\$	\$	=	\$	\$	=



S ₁	E ₁₋₃	L ₁₋₆	L7-5	L10-5	R ₁₋₂	R ₇₋₈	R ₁₀₋₉
\$	\$	=\$	\$	=	\$	\$	=





S ₁	E ₁₋₃	L ₁₋₆	L ₇₋₅	L ₁₀₋₅	R ₁₋₂	R ₇₋₈	R ₁₀₋₉
\$	\$	=\$	\$	=	\$	\$	=

Summary of LALR(1)

- Along with LL(k), one of the most popular parsing algorithms in use today.
- LALR(1) is produced by the bison parser generator; rarely generated by hand.
- Can handle most, but not all, LR(1) languages.

- What should the parser do when it encounters an error?
- Could just say "syntax error," but we'd like more detailed messages.
- How do we resume parsing after an error?

State on				Action		
top of	id	+	*	()	\$
stack						
0	s3	e 1	e 1	s2	e2	e 1
1	е3	s4	s5	e 3	e2	acc
2	s3	e 1	e 1	s2	e2	e 1
3	е3	r4	r4	е3	r4	r4
4	s3	e 1	e 1	s2	e2	e 1
5	s3	e 1	e1	s2	e2	e 1
6	е3	s4	s5	е3	s9	e 4
7	e3	r1	s5	e3	r1	r1
8	е3	r2	r2	e3	r2	r2
9	е3	r3	r3	е3	r3	r3

- e1 : expect to see the beginning of expression, i.e., id, '('. E.g. 1 + +
- Error message: missing operand
- Recover by pretending that we've seen id and then push id and state 3 on the stack.

E -> E + E | E * E | (E) | id

State on				Action		
top of	id	+	*	()	\$
stack						
0	s3	e1	e1	s2	e2	e1
1	e3	s4	s5	e3	e2	acc
2	s3	e1	e1	s2	e2	e1
3	e3	r4	r4	е3	r4	r4
4	s3	e1	e1	s2	e2	e1
5	s3	e1	e1	s2	e2	e1
6	е3	s4	s5	е3	s9	e4
7	e3	r1	s5	e3	r1	r1
8	e3	r2	r2	e 3	r2	r2
9	е3	r3	r3	е3	r3	r3

- e2: expect to see the beginning of the new expression. E.g. 1+2)
- Error message: Unbalanced right parenthesis.
- Recover by removing the right parenthesis from the current input.

State on				Action		
top of stack	id	+	*	()	\$
0	s3	e1	e1	s2	e2	e1
1	e3	s4	s5	e3	e2	acc
2	s3	e1	e1	s2	e2	e1
3	e 3	r4	r4	e3	r4	r4
4	s3	e1	e1	s2	e2	e 1
5	s3	e1	e1	s2	e2	e1
6	e3	s4	s5	e3	s9	e4
7	e3	r1	s5	e3	r1	r1
8	e3	r2	r2	e3	r2	r2
9	e 3	r3	r3	e3	r3	r3

- e3: expect to see the operator. E.g. id+id id, id(, etc.
- Error message: Missing operator.
- Recover by pushing operator + and state 4 on the stack.

E -> E + E | E * E | (E) | id

State on				Action		
top of	id	+	*	()	\$
stack						
0	s3	e1	e1	s2	e2	e1
1	е3	s4	s5	e3	e2	acc
2	s3	e1	e1	s2	e2	e1
3	e3	r4	r4	e 3	r4	r4
4	s3	e1	e1	s2	e2	e1
5	s3	e1	e1	s2	e2	e1
6	е3	s4	s5	е3	s9	e4
7	e3	r1	s5	e 3	r1	r1
8	е3	r2	r2	е3	r2	r2
9	e3	r3	r3	e3	r3	r3

- e4: expect to see the operator or the right parenthesis.
 E.g. (1+2 \$
- Error message: Missing right parenthesis.
- Recover by pushing) and state 9 on the stack.

Sample parsing on the erroneous input id +)

State on				Action			Goto		_	
top of stack	id	+	*	()	\$	Е	Stack	Input	Err Msg &Action
0	s3	e1	e1	s2	e2	e1	1			4 , 100.011
1	e3	s4	s5	е3	e2	acc		0	id+)\$	
2	s3	e1	e1	s2	e2	e1	6			
3	е3	r4	r4	e3	r4	r4				
4	s3	e1	e1	s2	e2	e1	7			
5	s3	e1	e1	s2	e2	e1	8			
6	е3	s4	s5	e3	s9	e4				
7	е3	r1	s5	е3	r1	r1				
8	е3	r2	r2	е3	r2	r2				
9	е3	r3	r3	e3	r3	r3				

Sample parsing on the erroneous input id +)

top of stack id	State on				Action			Goto	<u> </u>		
0 s3 e1 e1 s2 e2 e1 1		id	+	*	()	\$	Е	Stack	Input	
2	0	s3	e1	e1	s2	e2	e1	1			QACIOII
3	1	1					acc		0	id+)\$	
4 s3 e1 e1 s2 e2 e1 7 5 s3 e1 e1 s2 e2 e1 8 6 e3 s4 s5 e3 s9 e4 7 e3 r1 s5 e3 r1 r1 8 e3 r2 r2 e3 r2 r2		s3	e1	e1	s2	e2	e1	6	0: 10	٠, ١, ٢,	
5 s3 e1 e1 s2 e2 e1 8 6 e3 s4 s5 e3 s9 e4 7 e3 r1 s5 e3 r1 r1 8 e3 r2 r2 e3 r2 r2	3	е3	r4	r4	e3	r4	r4		Uld3	+)\$	
6 e3 s4 s5 e3 s9 e4 7 e3 r1 s5 e3 r1 r1 8 e3 r2 r2 e3 r2 r2	4	s3	e1	e1	s2	e2	e1	7			
7 e3 r1 s5 e3 r1 r1 8 e3 r2 r2 r2	5	s3	e1	e1	s2	e2	e1	8			
8 e3 r2 r2 e3 r2 r2	6	e3	s4	s5	e3	s9	e4				
	7	е3	r1	s5	e3	r1	r1				
9 e3 r3 r3 e3 r3 r3	8	е3	r2	r2	e3	r2	r2				
	9	е3	r3	r3	e3	r3	r3				

Sample parsing on the erroneous input id +)

top of id	l -	*		,						
		-		()	\$	Е	Stack	Input	Err Msg &Action
0 s3	В е	1 e	1 :	s2	e2	e1	1			C/ (CCIOII
1 e3	3 s	4 s	5 є	e3	e2	acc		0	id+)\$	
2 s3	В е	1 e	1 :	s2	e2	e1	6	0.10	٠, ١, ٢	
3 e 3	3 r	4 r4	ļ - є	€3	r4	r4		0id3	+)\$	
4 s3	В е	1 e	1 :	s2	e2	e1	7	OE1	+)\$	
5 s3	в е	1 e	1 :	s2	e2	e1	8	OLL	⊤ /→	
6 e 3	3 s	4 s	5 e	€3	s9	e4				
7 e3	3 r	1 s	5 6	e3	r1	r1				
8 e 3	3 r	2 r2	2 €	e3	r2	r2				
9 e 3	3 r	3 r3	3 6	€3	r3	r3				

Sample parsing on the erroneous input id +)

State on				Action			Goto			
top of stack	id	+	*	()	\$	E	Stack	Input	Err Msg &Action
0	s3	e1	e1	s2	e2	e1	1		_	3.0 130.311
1	e3	s4	s5	e3	e2	acc		0	id+)\$	
2	s3	e1	e1	s2	e2	e1	6	0: 12	. \ \ \	
3	e3	r4	r4	e3	r4	r4		0id3	+)\$	
4	s3	e1	e1	s2	e2	e1	7	0E1	+)\$	
5	s3	e1	e1	s2	e2	e1	8	OLI	י וי	
6	e3	s4	s5	e3	s9	e4		0E1+4)\$	-Unbalanced ')'
7	е3	r1	s5	e3	r1	r1		022 1	74	,
8	e3	r2	r2	e 3	r2	r2				-Remove ')'
9	е3	r3	r3	е3	r3	r3				

Sample parsing on the erroneous input id +)

State on				Action			Goto
top of	id	+	*	()	\$	Е
stack							
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	e3	r4	r4	e3	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	e3	r1	s5	e3	r1	r1	
8	е3	r2	r2	e3	r2	r2	
9	e3	r3	r3	е3	r3	r3	

Stack	Input	Err Msg &Action
0	id+)\$	
0id3	+)\$	
0E1	+)\$	
0E1+4)\$	-Unbalanced ')' -Remove ')'
0E1+4	\$	-Missing opnd Push id 3

Sample parsing on the erroneous input id +)

 $E \rightarrow E + E \mid E * E \mid (E) \mid id$

State on				Action			Goto			
top of stack	id	+	*	()	\$	E	Stack	Input	Err Msg &Action
0	s3	e1	e1	s2	e2	e1	1			G/ 1011011
1	e3	s4	s5	e3	e2	acc		0	id+)\$	
2	s3	e1	e1	s2	e2	e1	6	0.40	٠, ١, ٢	
3	е3	r4	r4	e3	r4	r4		0id3	+)\$	
4	s3	e1	e1	s2	e2	e1	7	0E1	+)\$	
5	s3	e1	e1	s2	e2	e1	8	OLI	⊤ /⊋	
6	е3	s4	s5	e3	s9	e4		0E1+4)\$	-Unbalanced ')'
7	е3	r1	s5	e3	r1	r1			/ Ψ	•
8	е3	r2	r2	e3	r2	r2				-Remove ')'
9	е3	r3	r3	e3	r3	r3		0E1 + /	<mark>ረ</mark>	N 4: :
								0E1+4	\$	-Missing opnd.- Push id 3

Pretend by pushing Id and state 3 on the top of stack

0E1+4id3) \$

Sample parsing on the erroneous input id +)

 $E \rightarrow E + E \mid E * E \mid (E) \mid id$

State on				Action			Goto	`		
top of stack	id	+	*	()	\$	E	Stack	Input	Err Msg &Action
0	s3	e1	e1	s2	e2	e1	1			C (7 1001011
1	e3	s4	s5	e3	e2	acc		0	id+)\$	
2	s3	e1	e1	s2	e2	e1	6	0:42	٠,١٥	
3	e3	r4	r4	e3	r4	r4		0id3	+)\$	
4	s3	e1	e1	s2	e2	e1	7	0E1	+)\$	
5	s3	e1	e1	s2	e2	e1	8	OLI	י וי	
6	e3	s4	s5	e3	s9	e4		0E1+4)\$	-Unbalanced ')'
7	е3	r1	s5	e3	r1	r1		0221	74	•
8	e3	r2	r2	e3	r2	r2				-Remove ')'
9	e3	r3	r3	е3	r3	r3		0E1	ć	N 4::
								0E1+4	P	-Missing opnd.- Push id 3

0E1+4id3 \$

0E1+4E7

Sample parsing on the erroneous input id +)

State on				Action			Goto			
top of stack	id	+	*	()	\$	E	Stack	Input	Err Msg &Action
0	s3	e1	e1	s2	e2	e1	1		_	
1	e3	s4	s5	e3	e2	acc		0	id+)\$	
2	s3	e1	e1	s2	e2	e1	6	0: 10	٠ ١ ٨	
3	e3	r4	r4	e3	r4	r4		0id3	+)\$	
4	s3	e1	e1	s2	e2	e1	7	0E1	+)\$	
5	s3	e1	e1	s2	e2	e1	8	OLI	7)7	
6	e3	s4	s5	e 3	s9	e4		0E1+4)\$	-Unbalanced ')'
7	е3	r1	s5	e3	r1	r1			74	•
8	e3	r2	r2	e 3	r2	r2				-Remove ')'
9	e3	r3	r3	е3	r3	r3		0E1+4	Ċ	N 4: :
								UEI+4	\$	-Missing opnd.- Push id 3

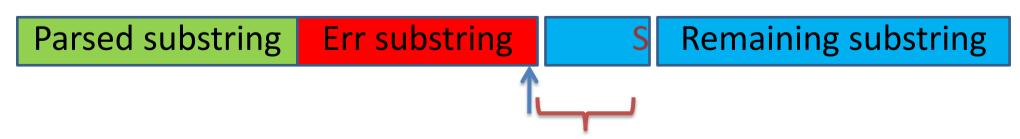
		-Remove ')'
	0E1+4	\$ -Missing opnd. - Push id 3
0E	1+4id3	\$
0E	1+4E7	\$
	0E1	\$

Sample parsing on the erroneous input id +)

State on	Action						Goto	Goto			
top of stack	id	+	*	()	\$	E	Stack	Input	Err Msg &Action	
0	s3	e1	e1	s2	e2	e1	1	0	• • • •		
1	e3	s4	s5	е3	e2	acc		0	id+)\$		
2	s3	e1	e1	s2	e2	e 1	6	0id3	+)\$		
3	e3	r4	r4	е3	r4	r4		ulus	⊤ /⊋		
4	s3	e1	e1	s2	e2	e1	7	0E1	+)\$		
5	s3	e1	e1	s2	e2	e1	8	OLI	. 14		
6	е3	s4	s5	e3	s9	e4		0E1+4)\$	-Unbalanced ')'	
7	е3	r1	s5	e3	r1	r1			,	'	
8	е3	r2	r2	e3	r2	r2				-Remove ')'	
9	e3	r3	r3	e3	r3	r3	<u> </u>	0E1+4	\$	-Missing opnd. - Push id 3	
							0E	1+4id3	\$		
0E:	1	\$ Accept				0E	1+4E7	\$			
		·		•				- 0E1	\$		

Panic Mode Error Recovery

- Previous error recovery is called "Phrase-level Recovery". It is a local correction on the remaining input.
- Another most popular error recovery is "Panic Mode"
- On discovering an error, the parser discards input symbols until one of a designated set of synchronizing tokens is found.
- Synchronizing tokens are usually delimiters, such as; or }.
 The compiler designers must select sync. tokens appropriately.
- This technique employed by bison and many other parser generators.



Discards input symbols until we've found sync. token S

Next Time

Semantic Analysis

- Overview of semantic analysis.
- Scoping and symbol tables.
- Introduction to types and type-checking.