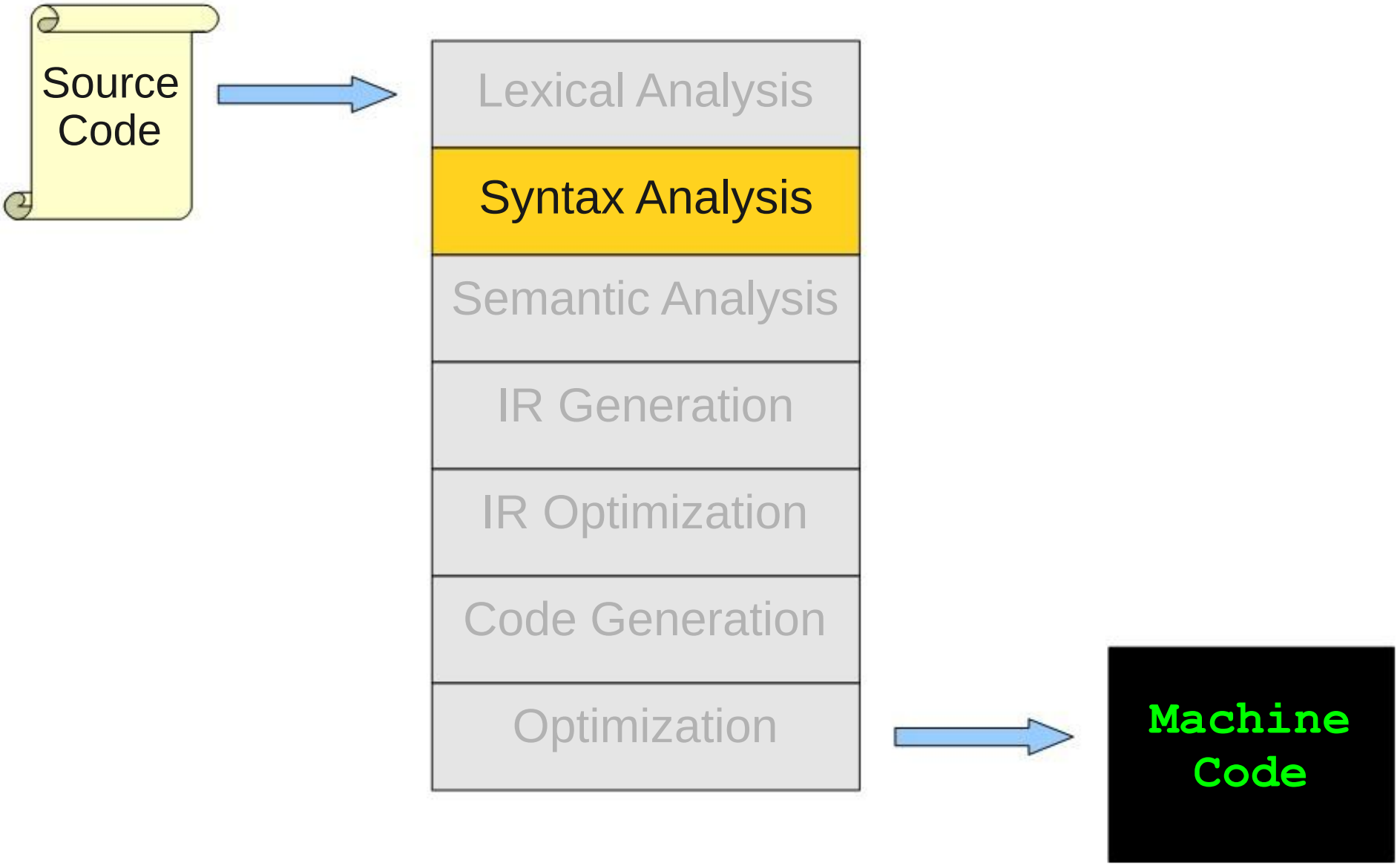
Bottom-Up Parsing III

Where We Are


Source Code
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.

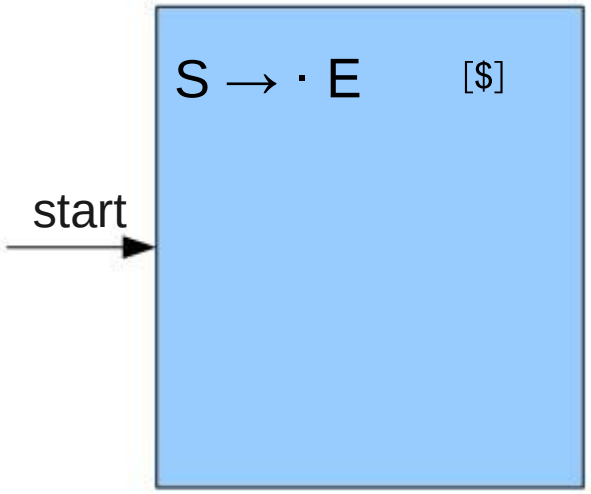
The “Lazy Merging” Technique

- **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.

S → E
E → L = R
E → R
L → **id**
L → ***R**
R → L

LALR(1) Construction

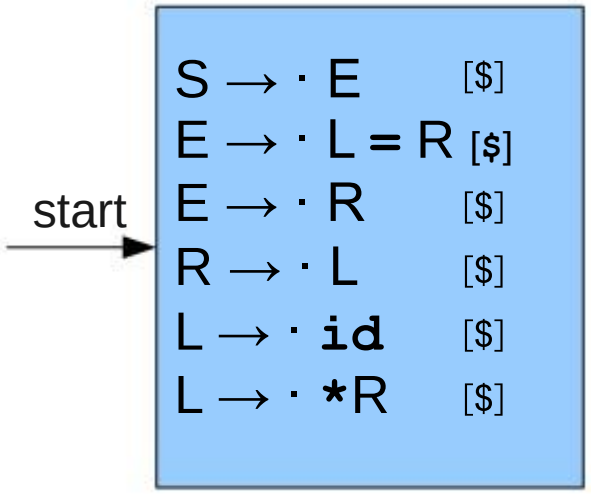
Lazy Merging



S → E
E → L = R
E → R
L → **id**
L → ***R**
R → L

LALR(1) Construction

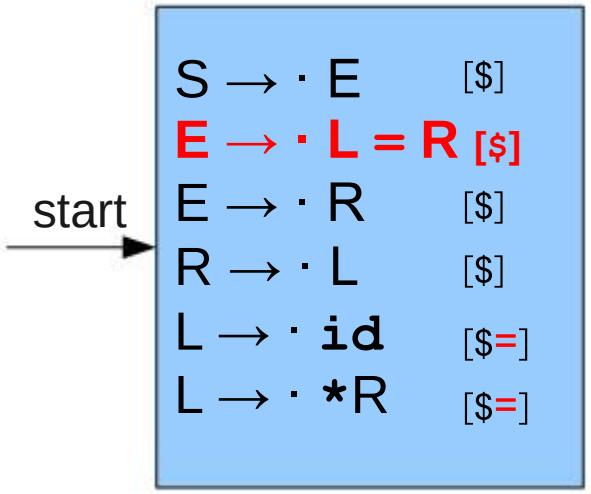
Lazy Merging



S → E
E → L = R
E → R
L → **id**
L → ***R**
R → L

LALR(1) Construction

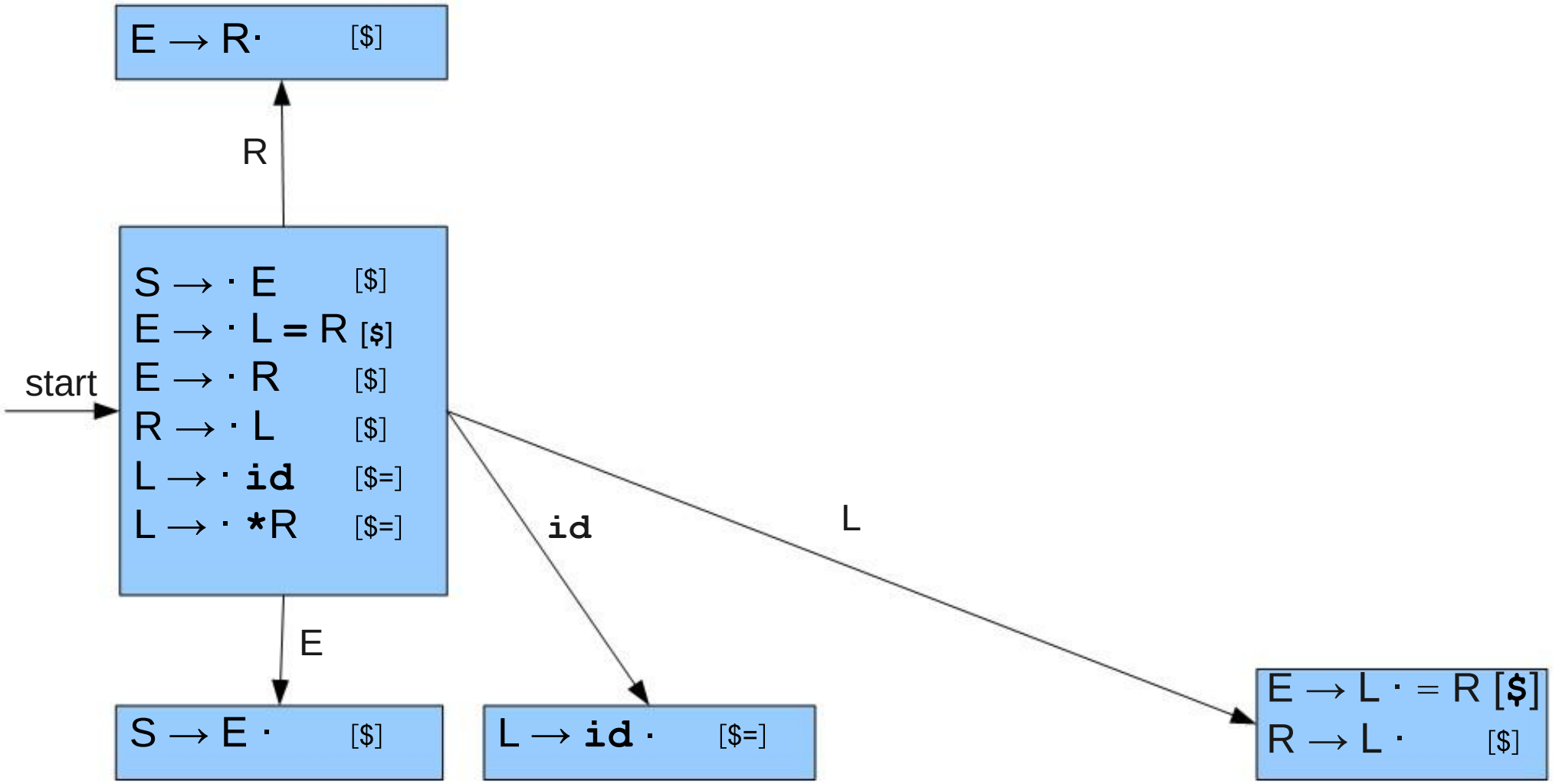
Lazy Merging



S → E
E → L = R
E → R
L → **id**
L → ***R**
R → L

LALR(1) Construction

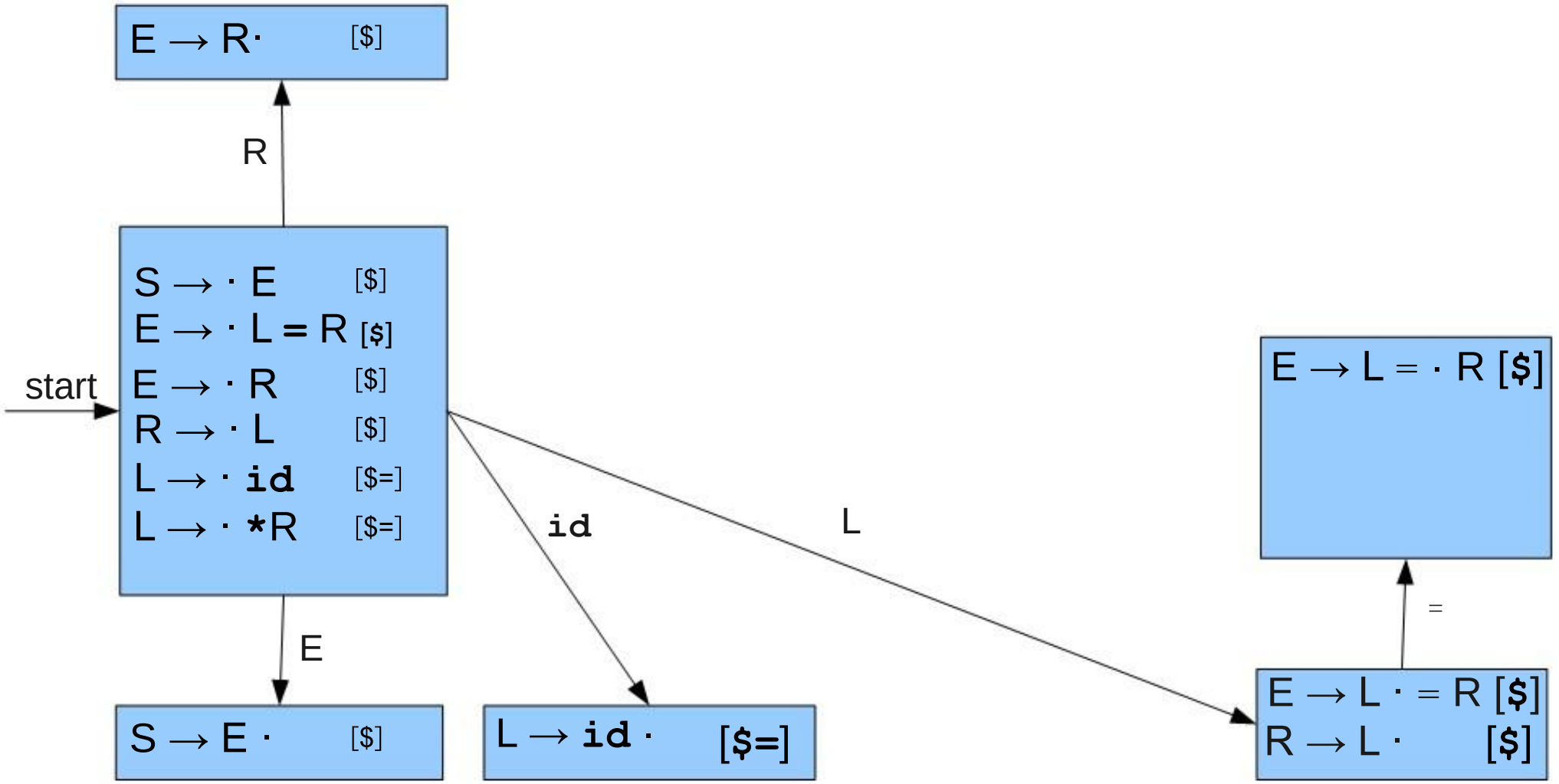
Lazy Merging



S → E
E → L = R
E → R
L → **id**
L → ***R**
R → L

LALR(1) Construction

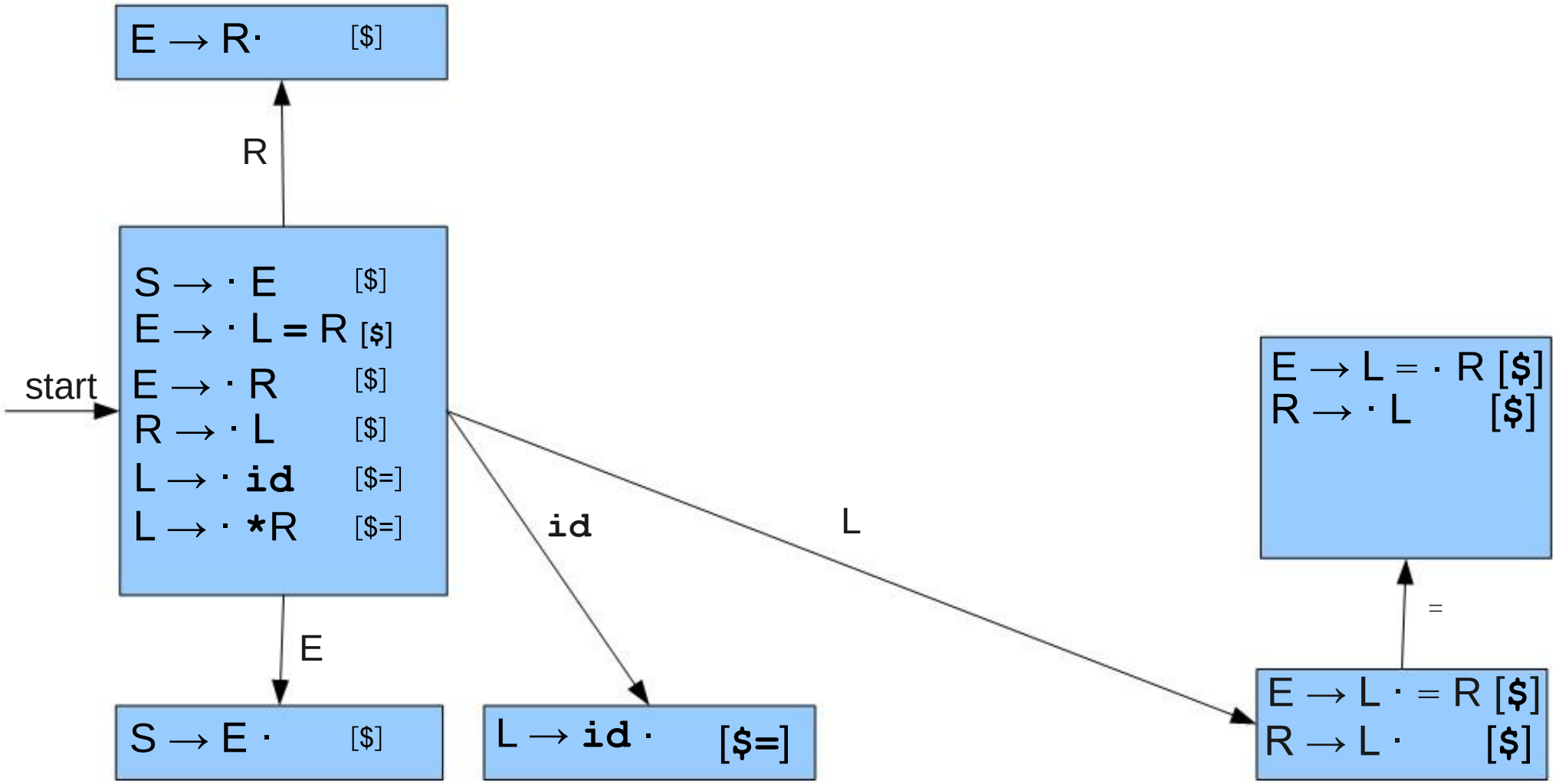
Lazy Merging



S → E
E → L = R
E → R
L → **id**
L → ***R**
R → L

LALR(1) Construction

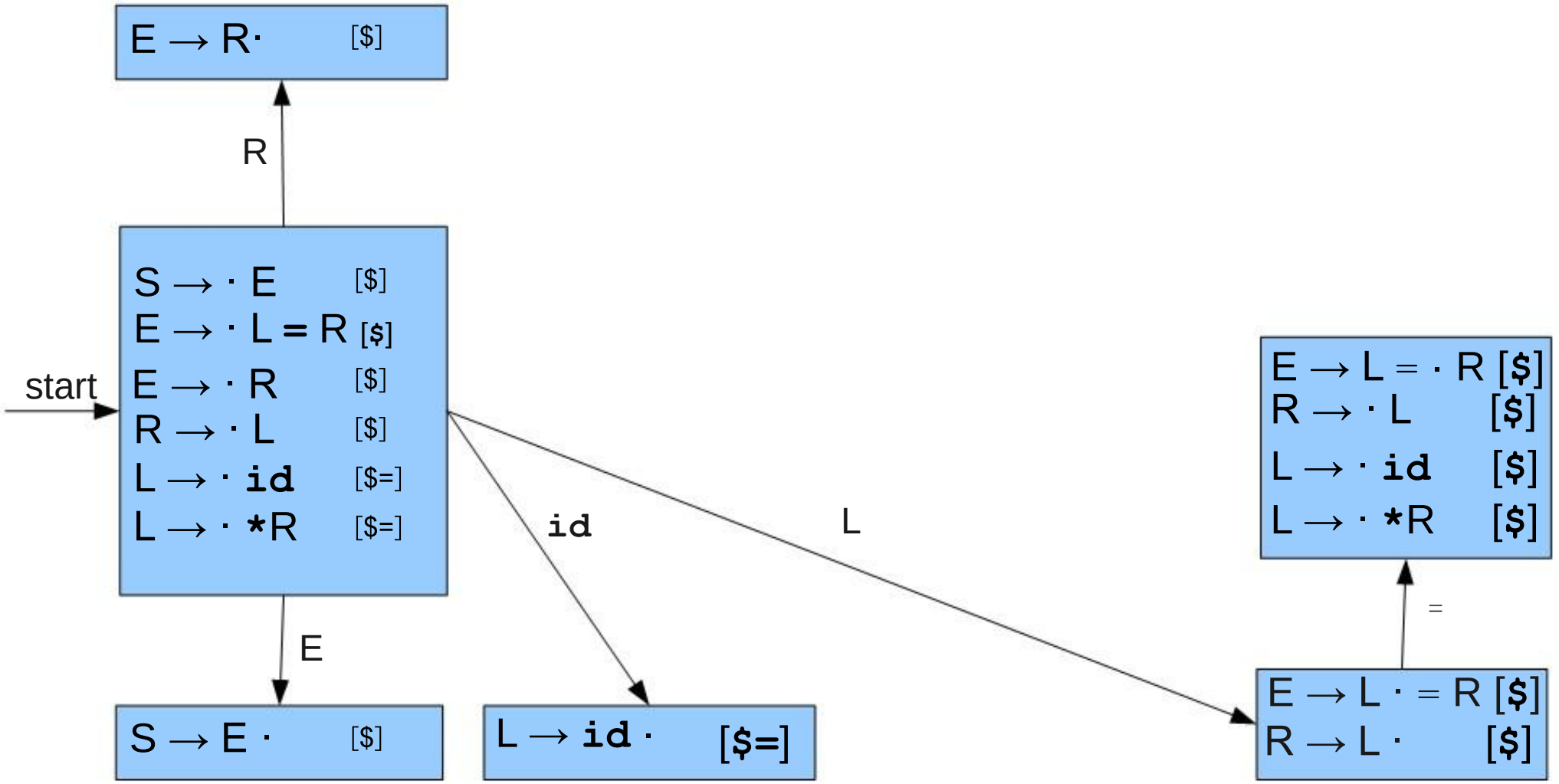
Lazy Merging



S → E
E → L = R
E → R
L → **id**
L → ***R**
R → L

LALR(1) Construction

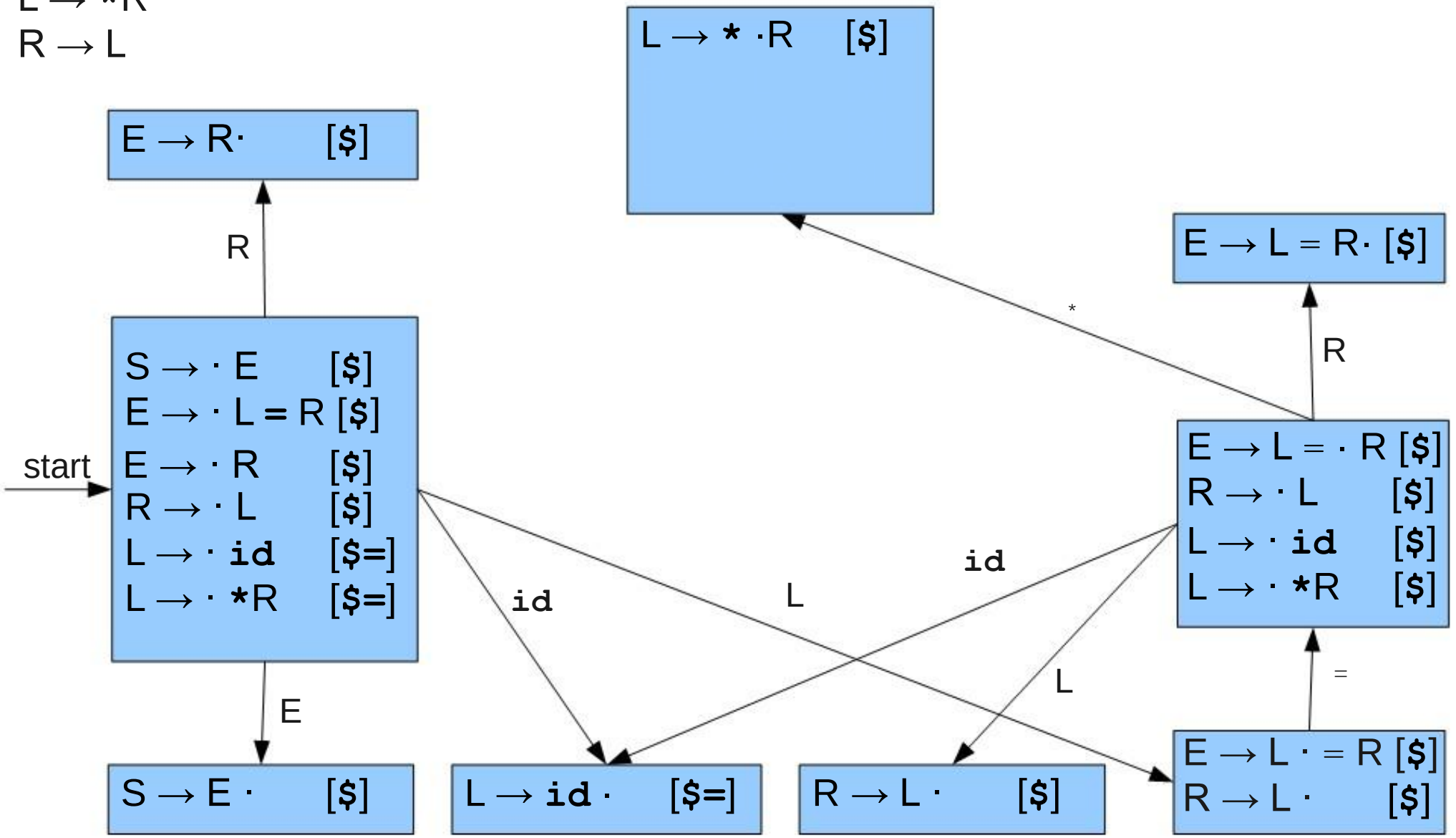
Lazy Merging



S → E
E → L = R
E → R
L → **id**
L → ***R**
R → L

LALR(1) Construction

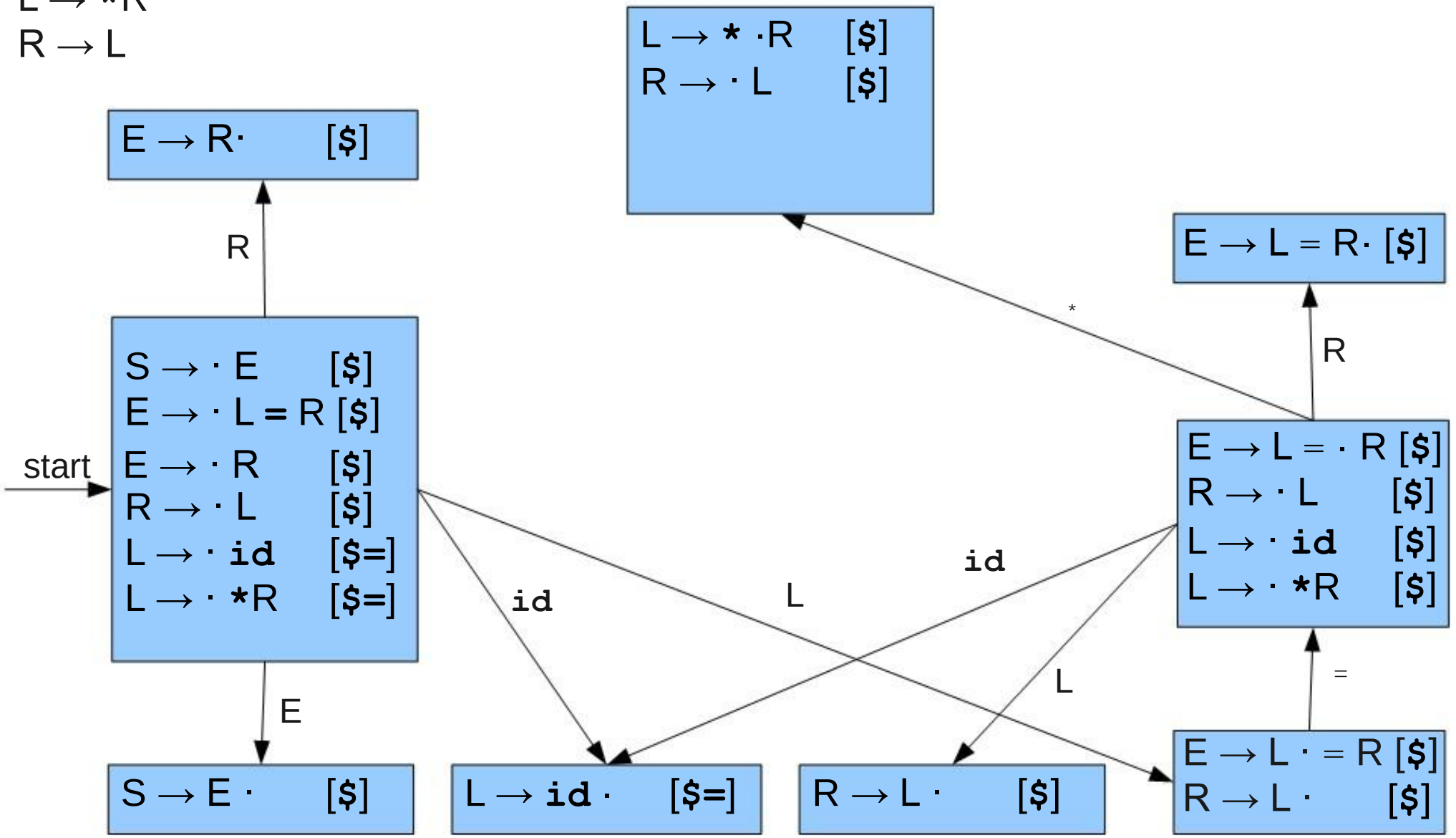
Lazy Merging



S → E
E → L = R
E → R
L → **id**
L → ***R**
R → L

LALR(1) Construction

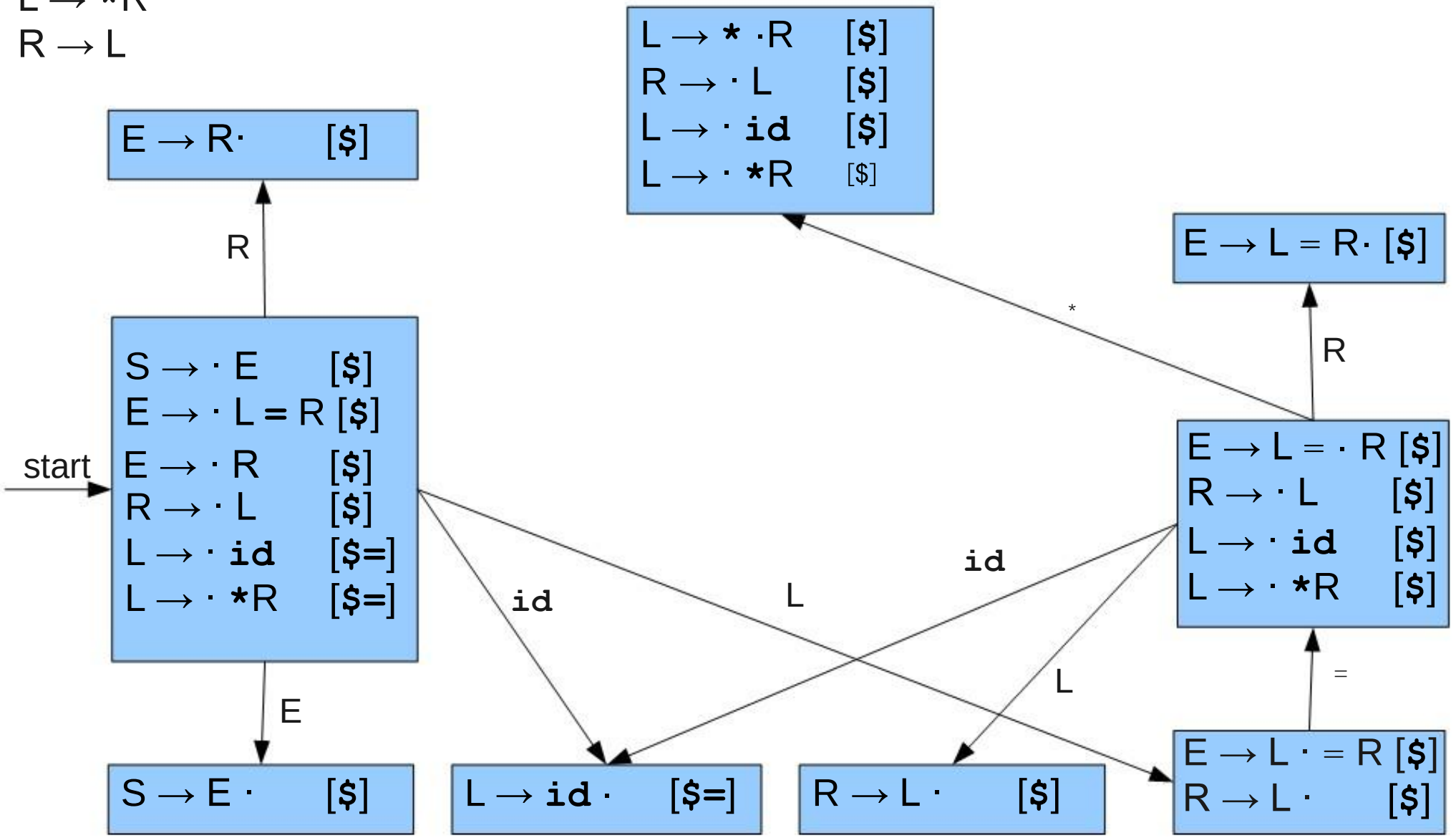
Lazy Merging



S → E
E → L = R
E → R
L → **id**
L → ***R**
R → L

LALR(1) Construction

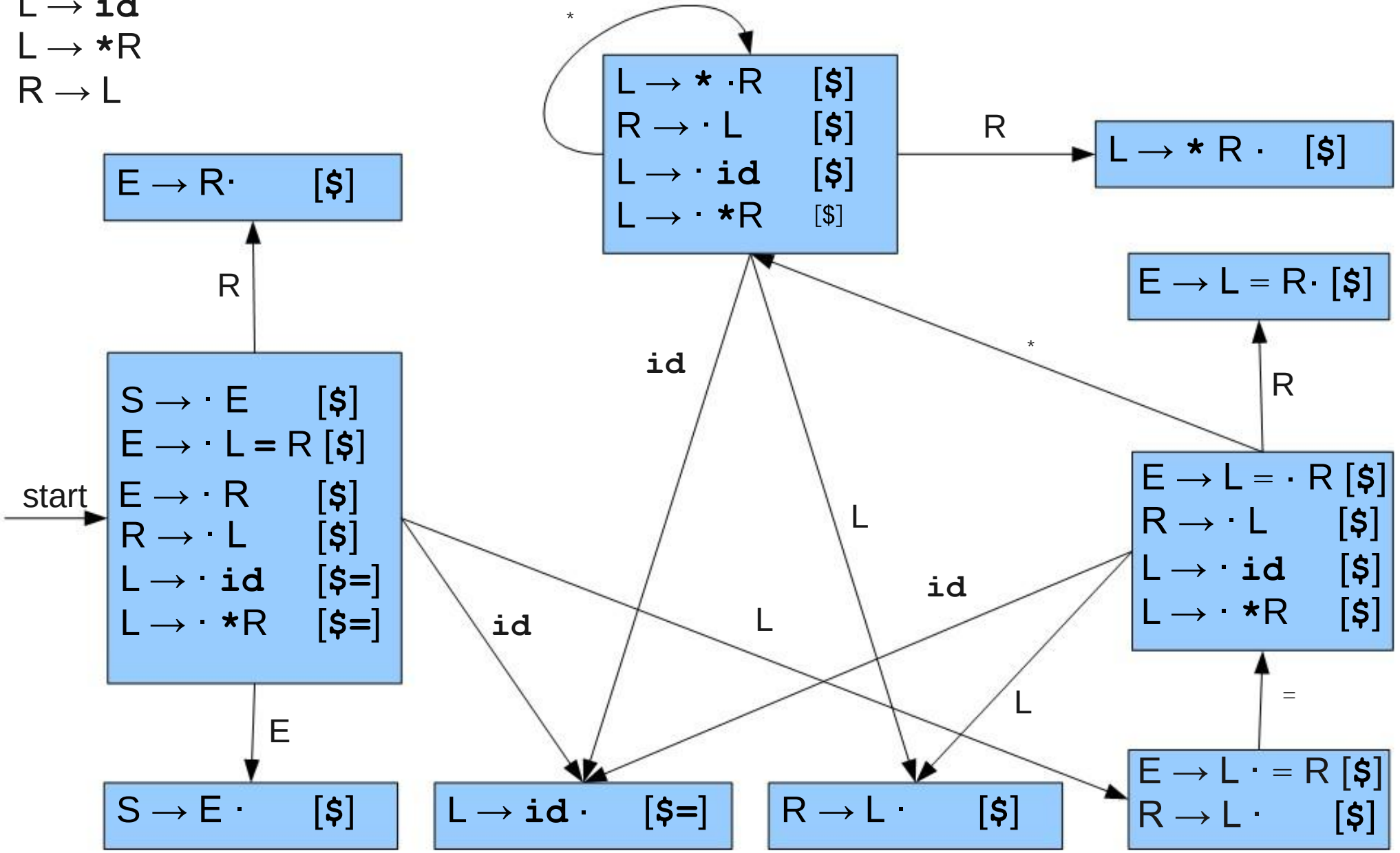
Lazy Merging



S → E
E → L = R
E → R
L → **id**
L → ***R**
R → L

LALR(1) Construction

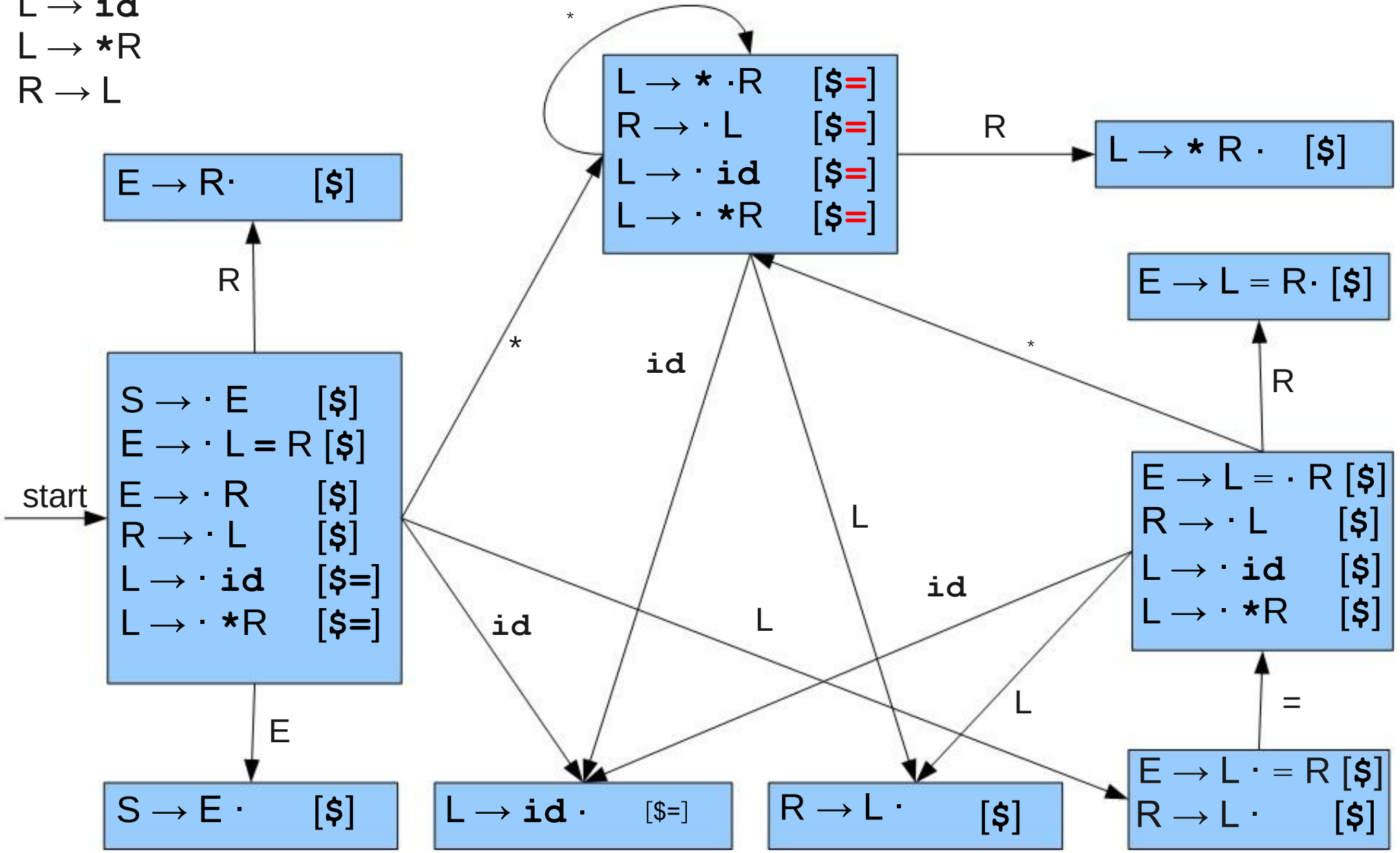
Lazy Merging



S → E
E → L = R
E → R
L → **id**
L → ***R**
R → L

LALR(1) Construction

Lazy Merging

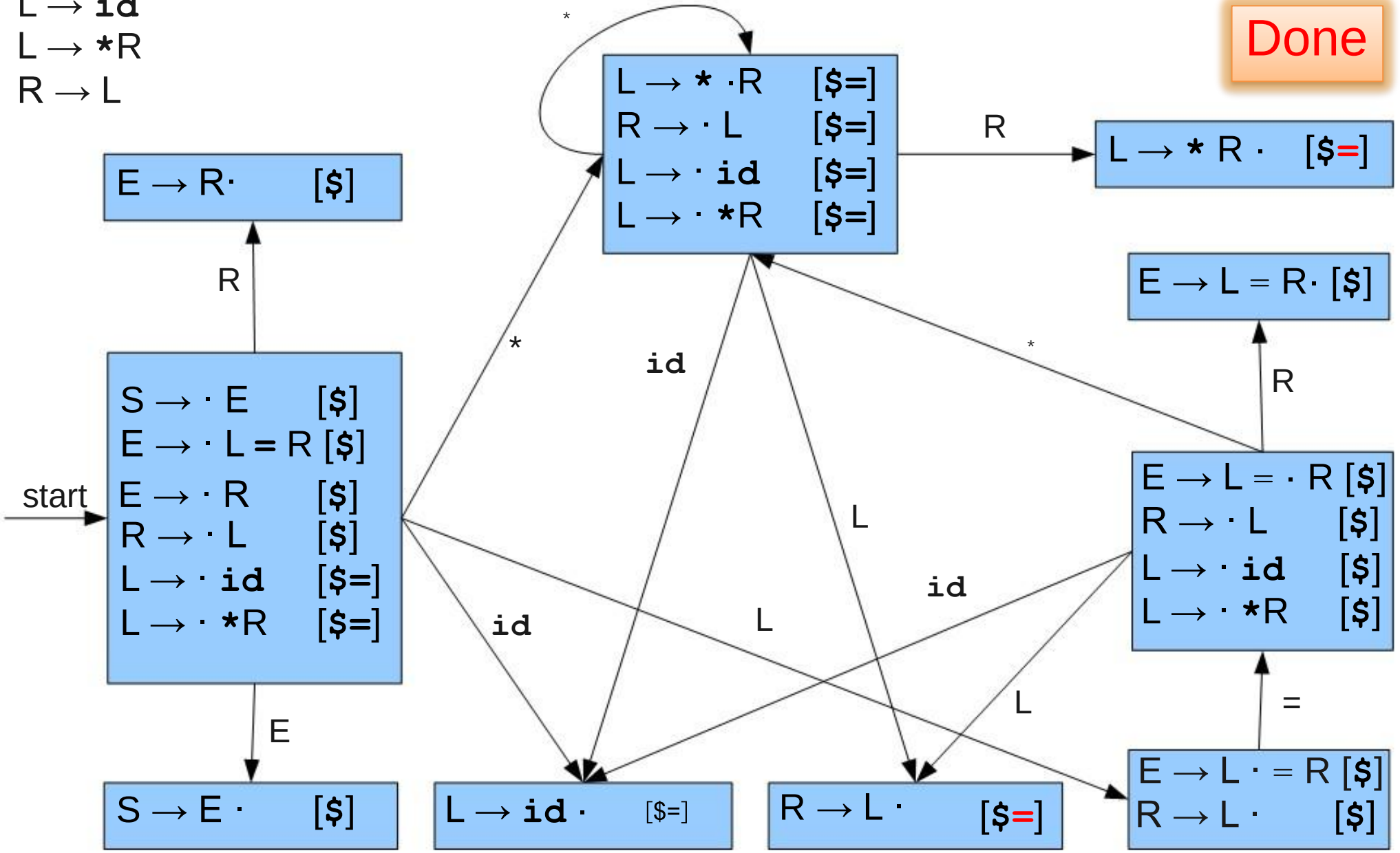


S → E
E → L = R
E → R
L → **id**
L → ***R**
R → L

LALR(1) Construction

Lazy Merging

Done



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.

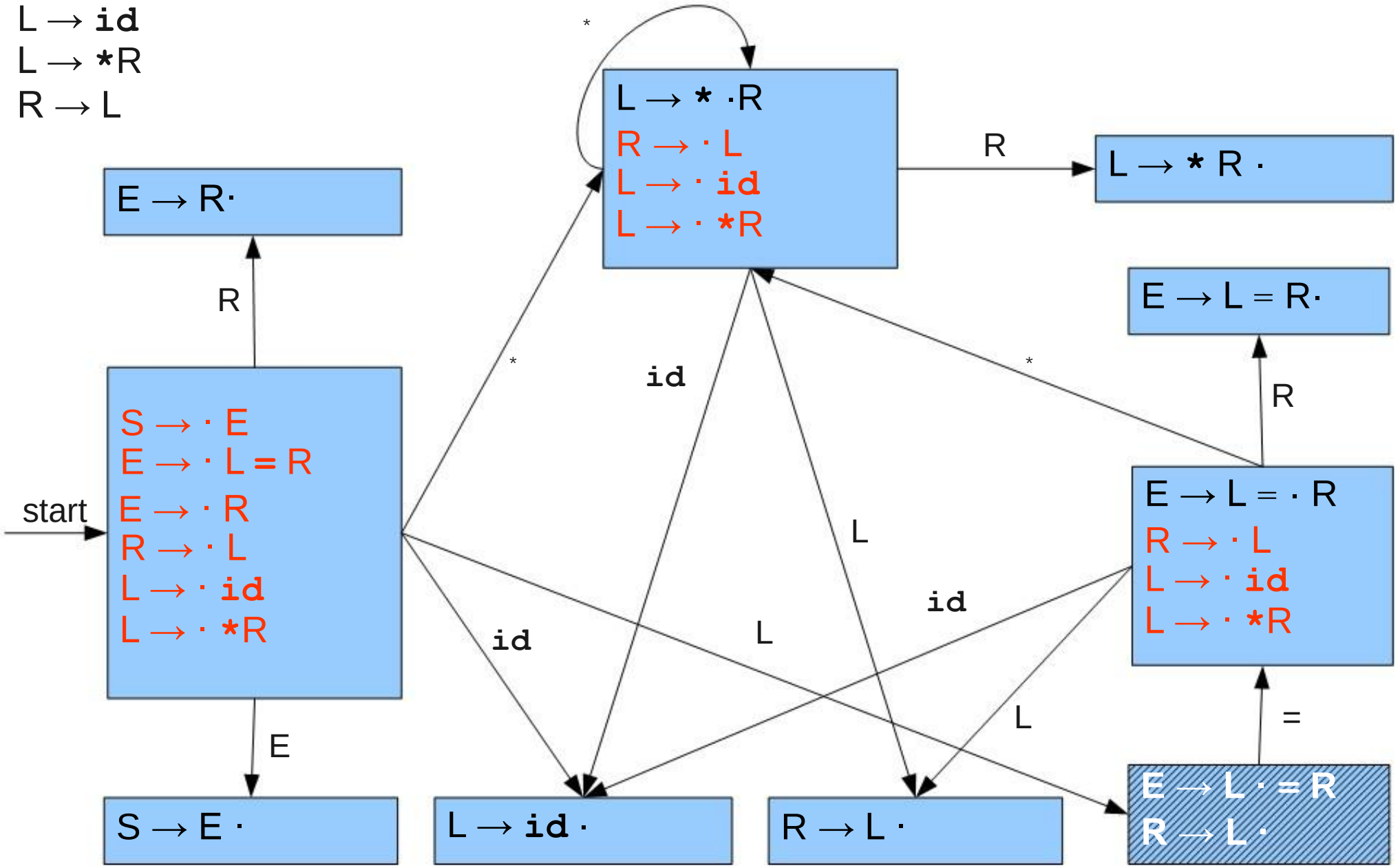
Second Technique: Fast LALR(1) Construction

- 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)

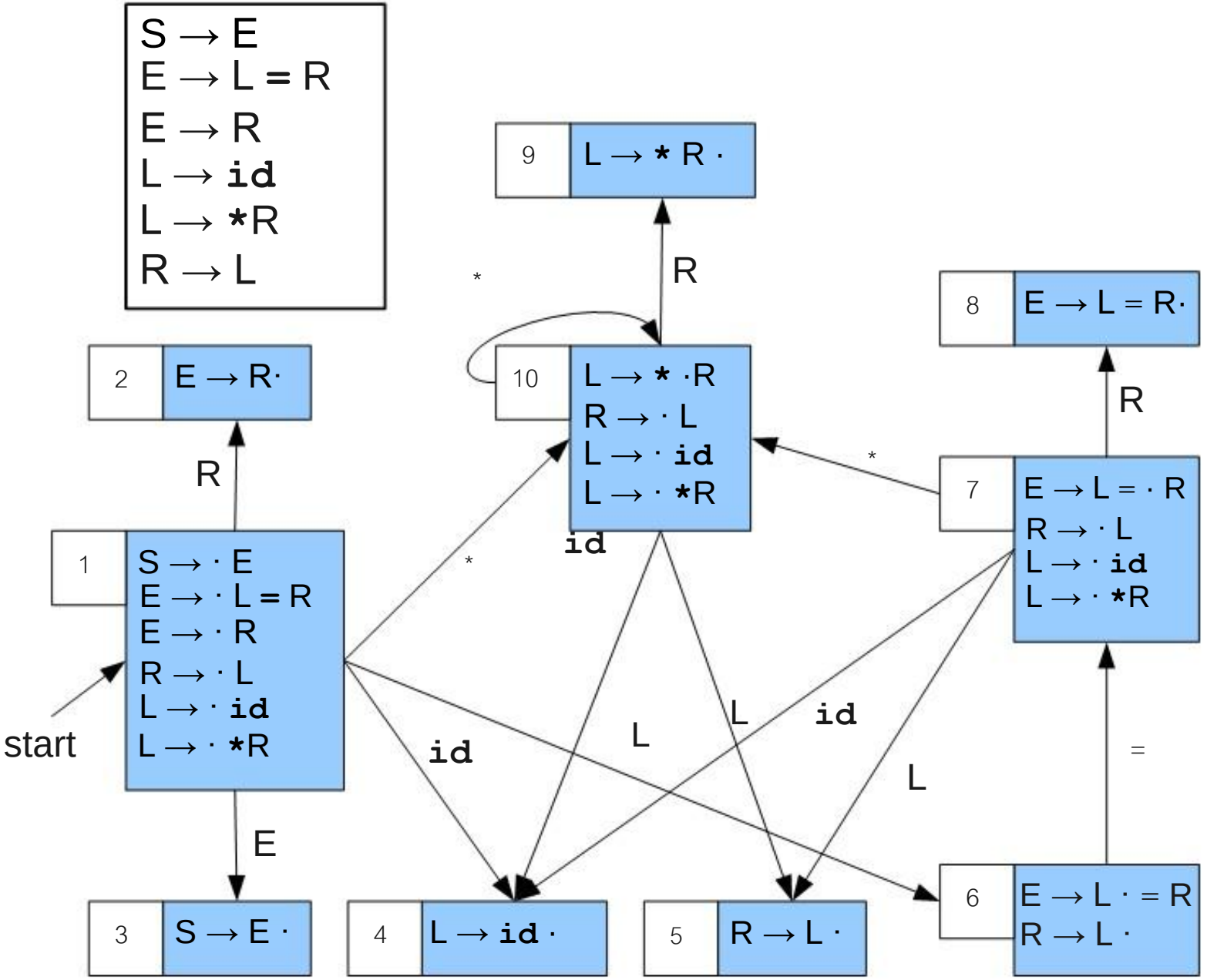
S → E
E → L = R
E → R
L → **id**
L → ***R**
R → L

An LR(0) Automaton



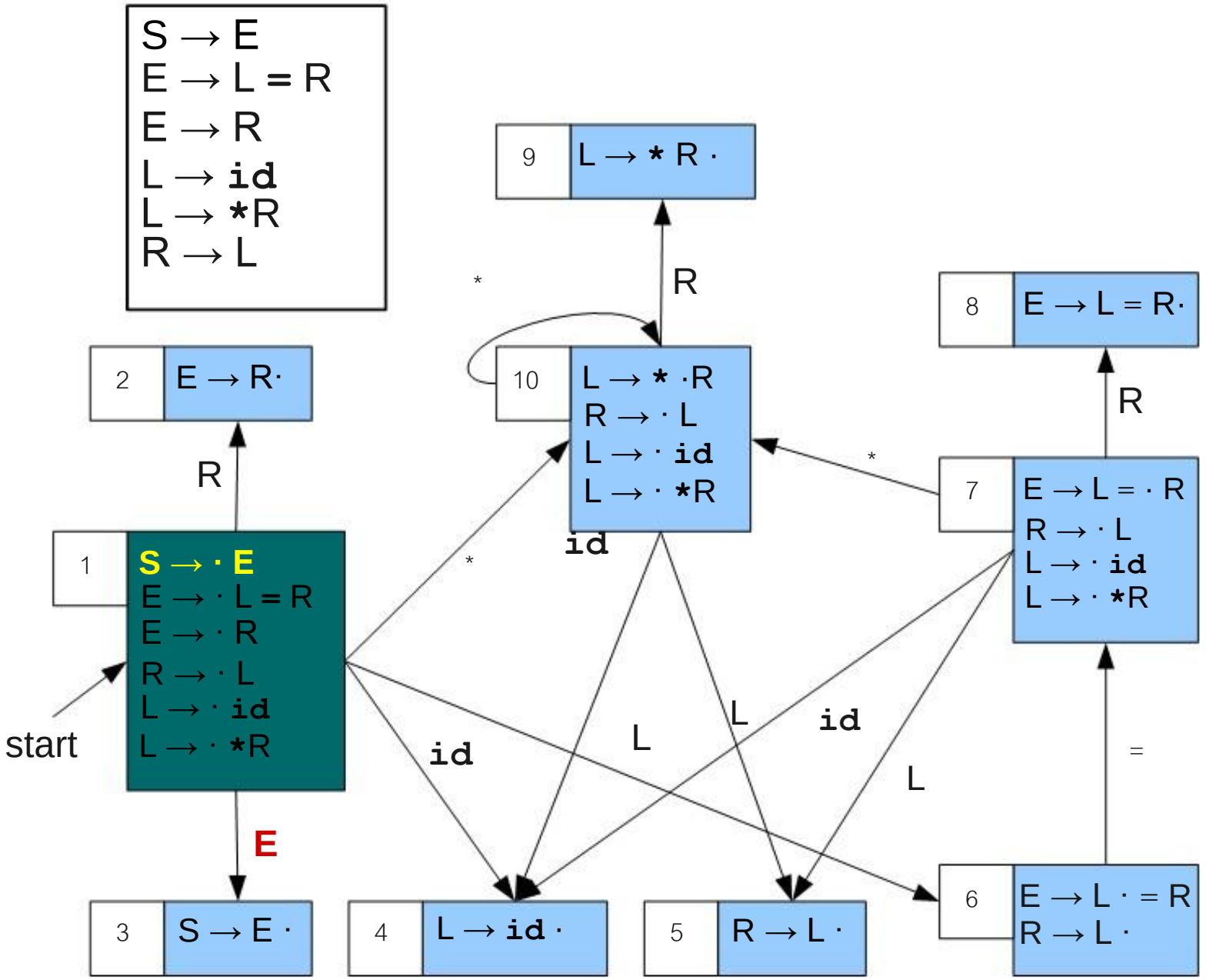
Note: We augment only productions which have a dot in front.

Augmenting the Grammar



Augmenting the Grammar

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

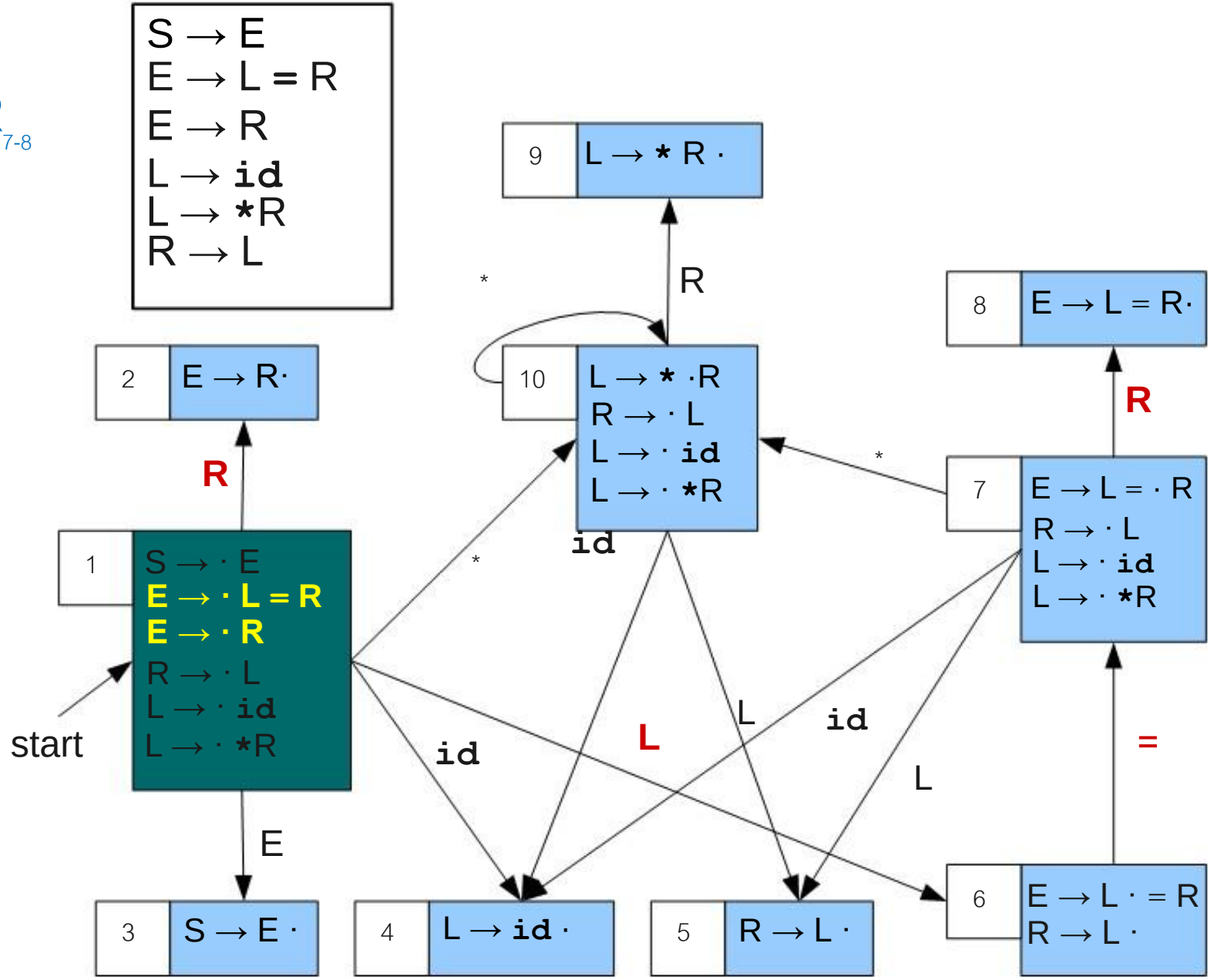


Augmenting the Grammar

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

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

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



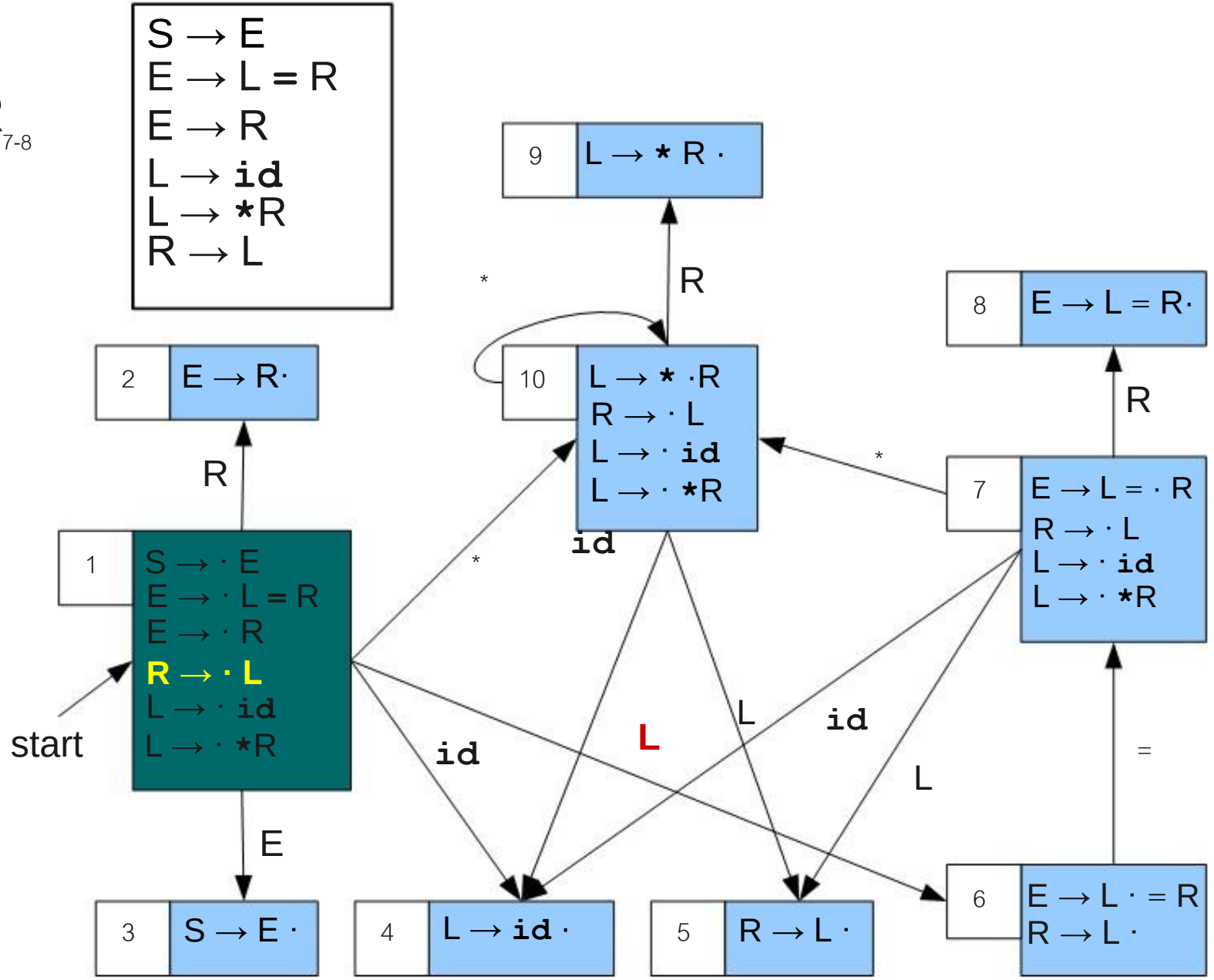
Augmenting the Grammar

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

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

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

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



Augmenting the Grammar

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

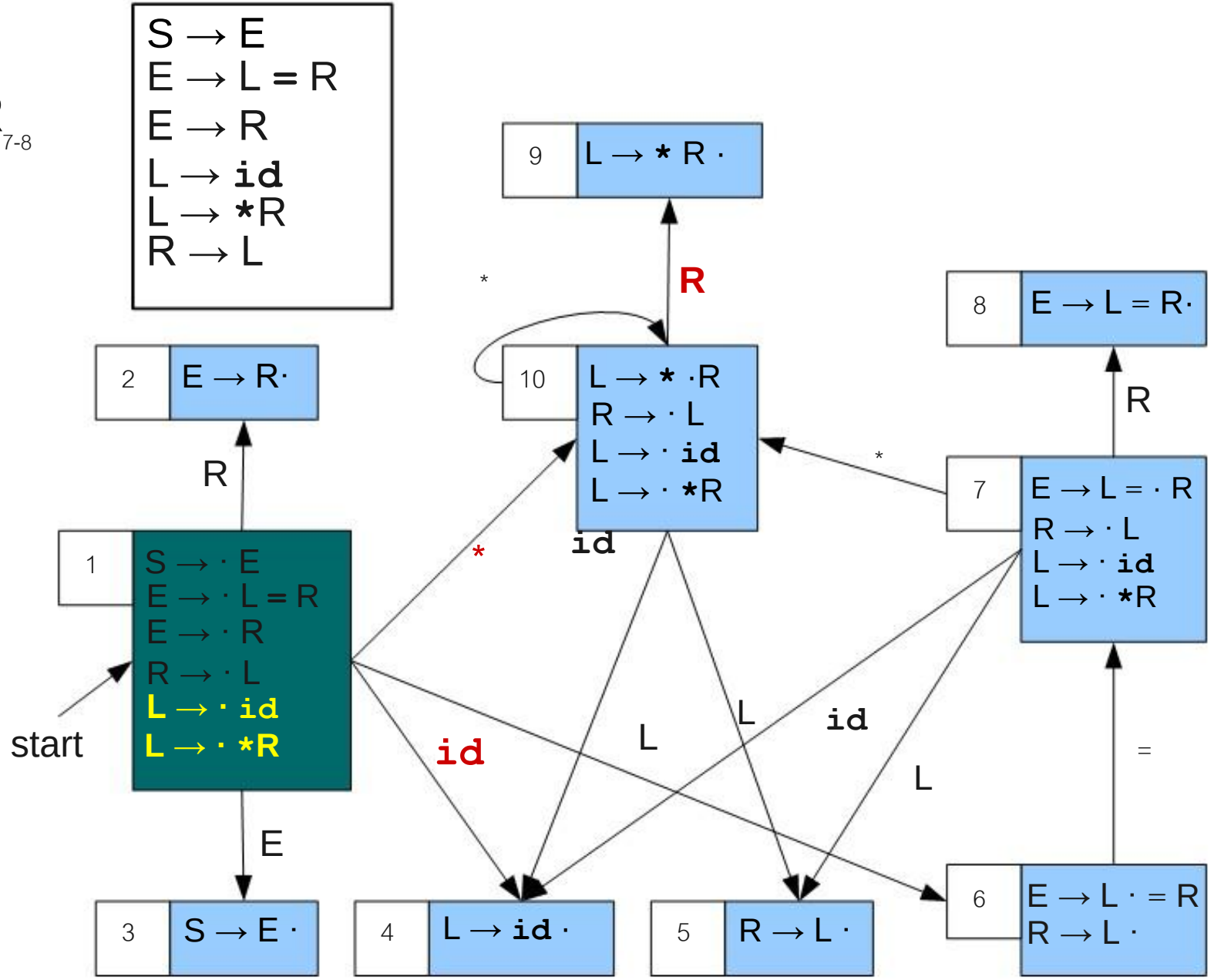
$E_{1\text{-}3} \rightarrow L_{1\text{-}6} = R_{7\text{-}8}$

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

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

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

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



Augmenting the Grammar

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

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

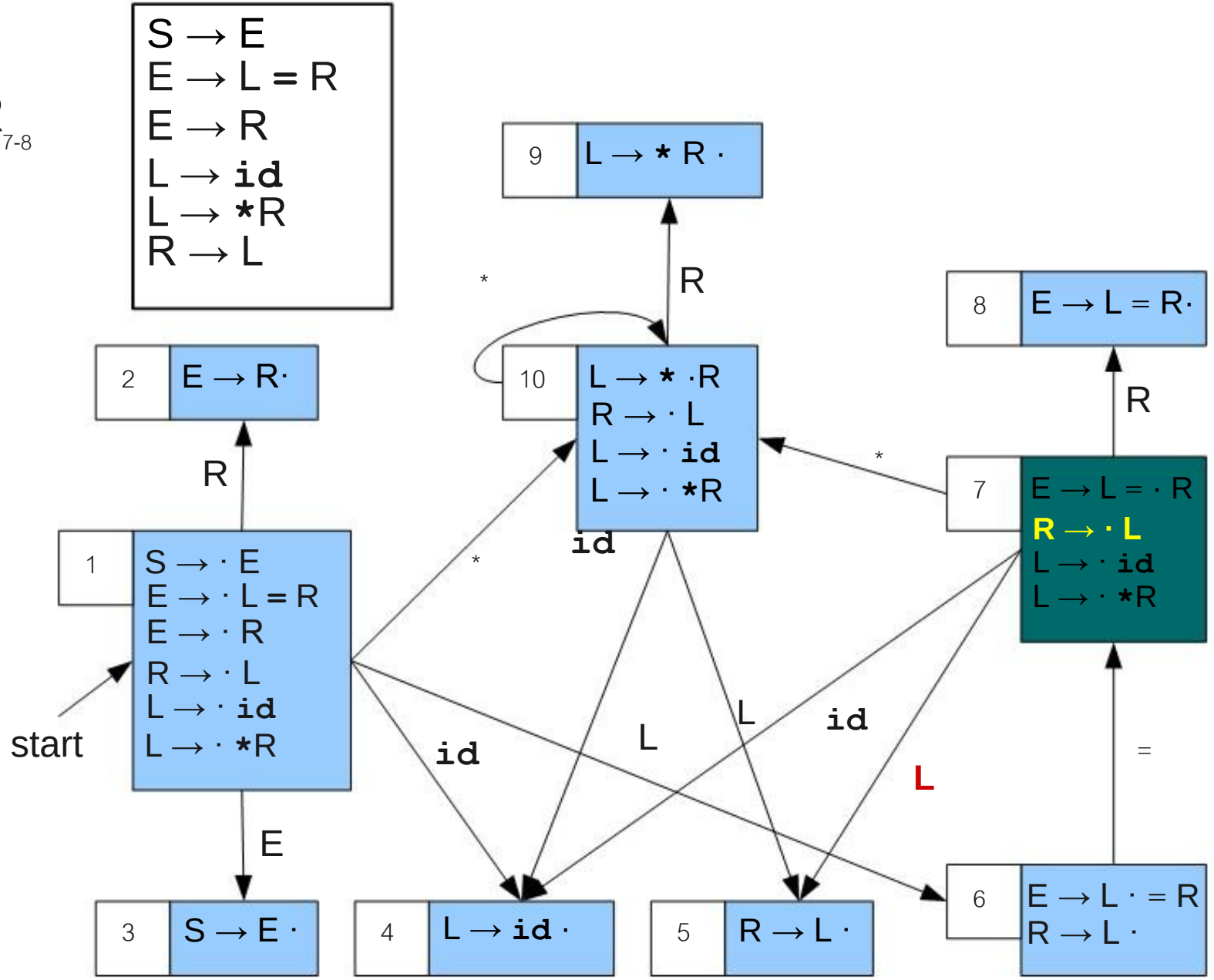
$E_{1\text{-}3} \rightarrow R_{1\text{-}2}$

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

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

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

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



Augmenting the Grammar

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

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

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

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

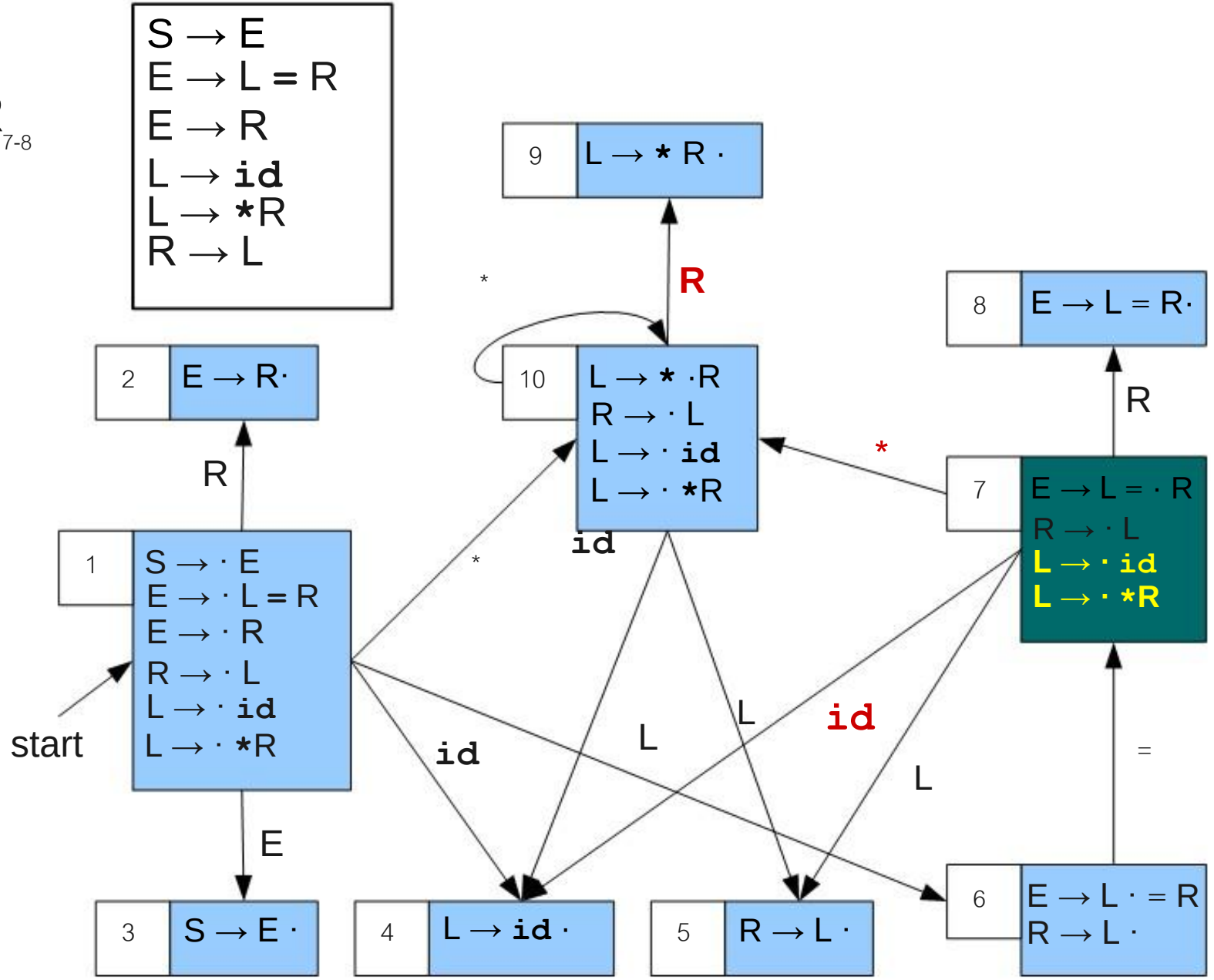
$L_{1\text{-}6} \rightarrow \mathbf{id}$

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

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

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

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



Augmenting the Grammar

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

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

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

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

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

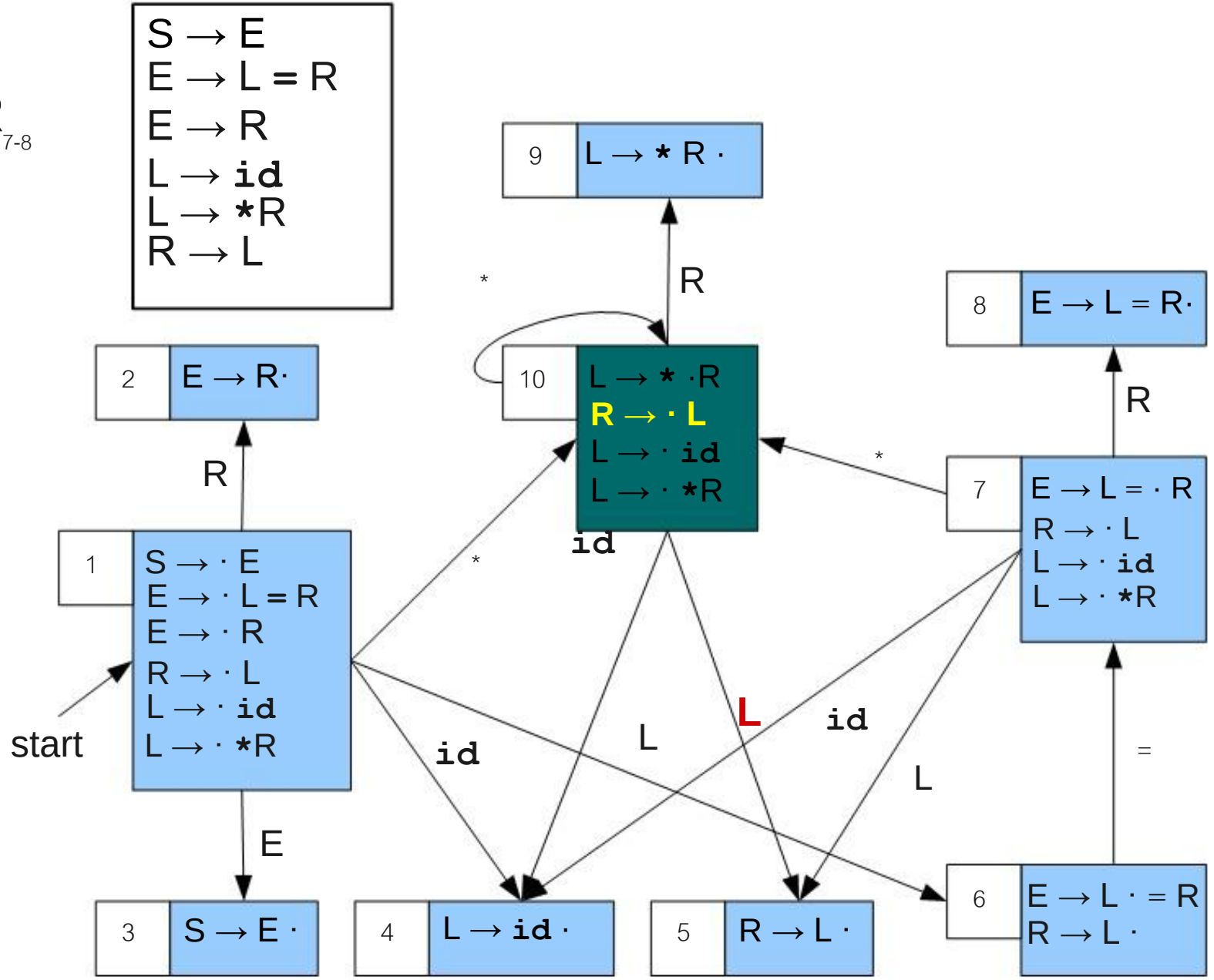
$L_{1\text{-}6} \rightarrow * R_{10\text{-}9}$

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

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

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

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



Augmenting the Grammar

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

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

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

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

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

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

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

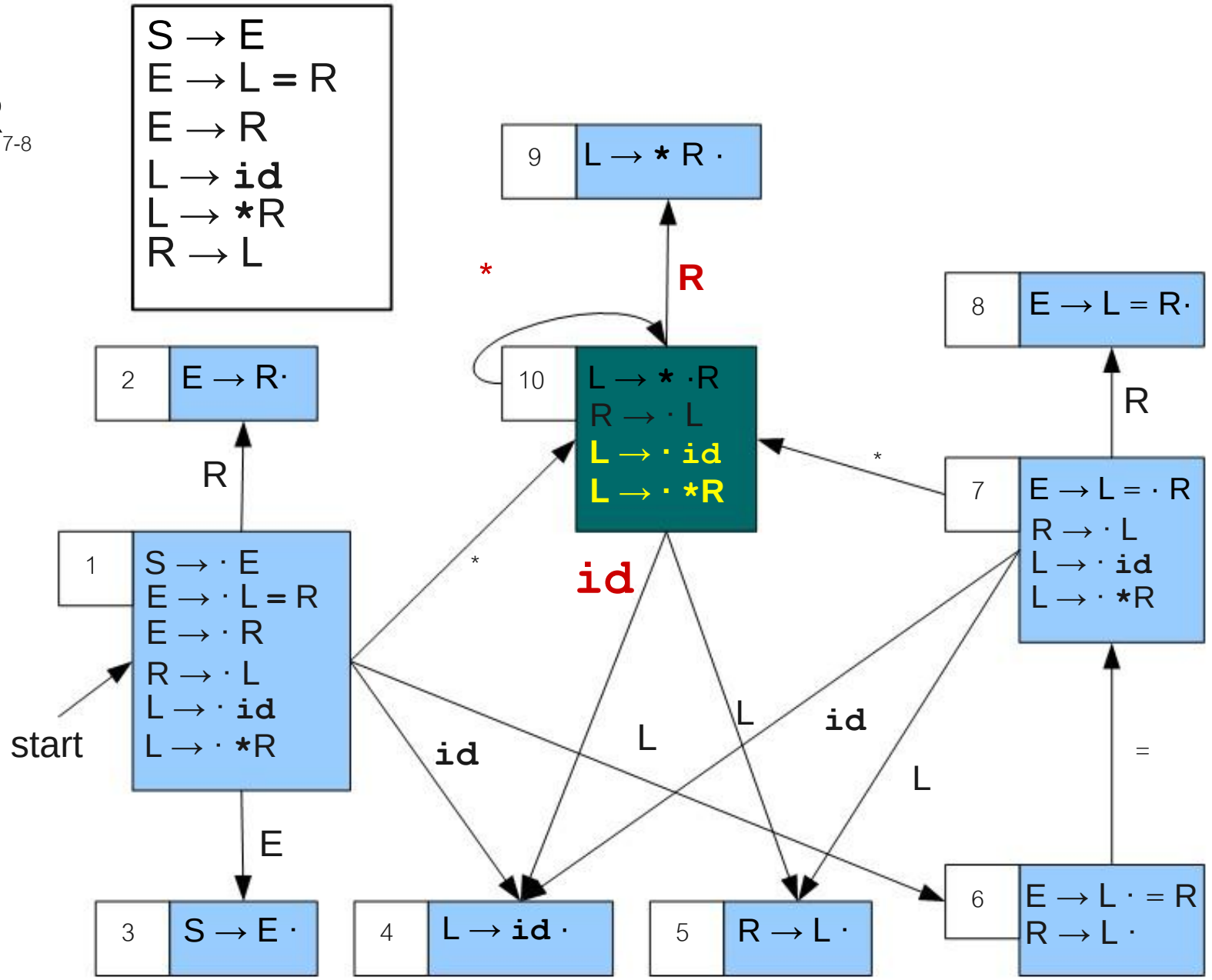
$L_{7\text{-}5} \rightarrow \mathbf{id}$

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

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

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

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



Augmenting the Grammar

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

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

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

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

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

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

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

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

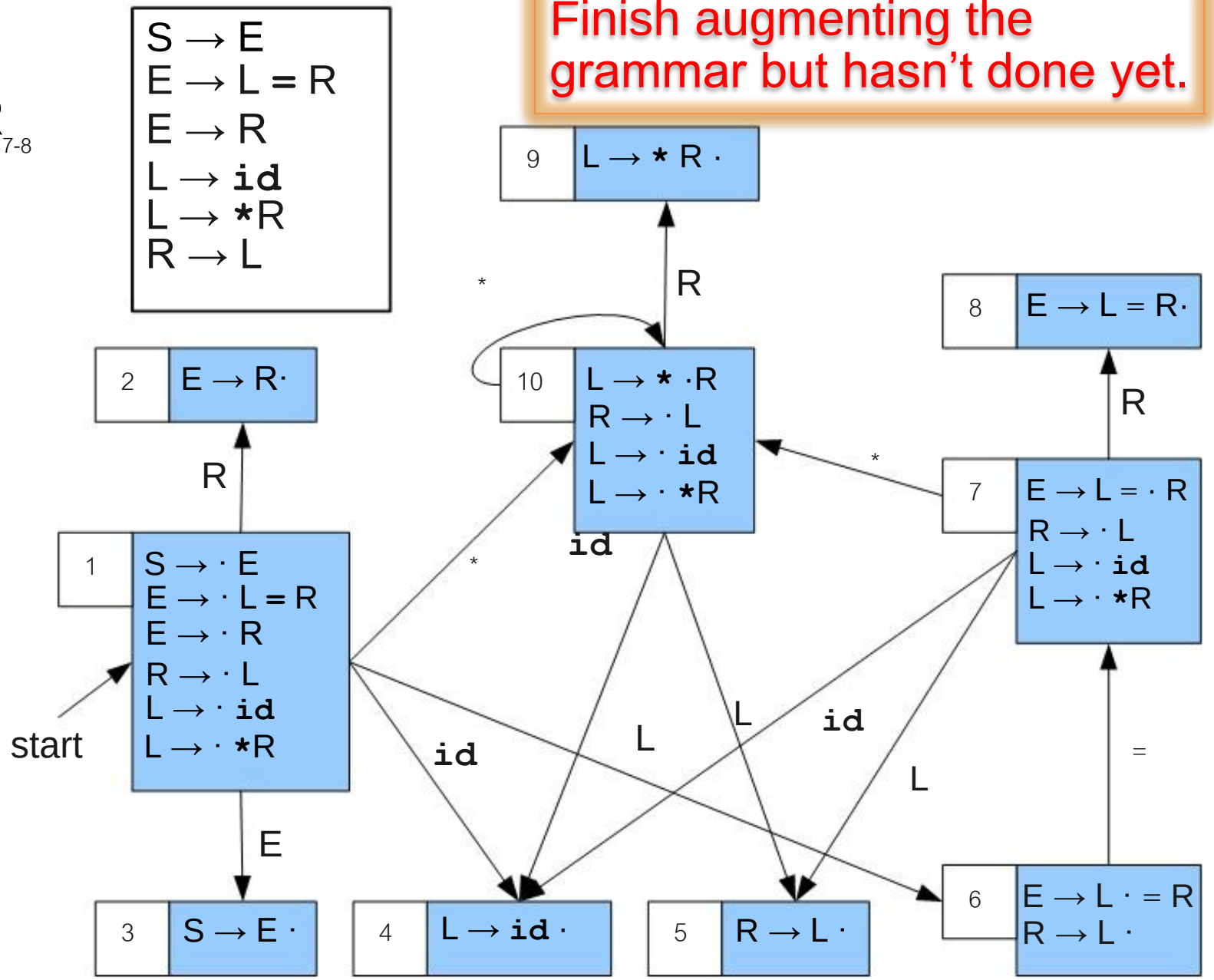
$L_{7\text{-}5} \rightarrow *R_{10\text{-}9}$

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

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

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

Finish augmenting the grammar but hasn't done yet.



Lookahead Sets

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

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

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

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

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

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

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

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

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

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

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

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

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

Lookahead Sets

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

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

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

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

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

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

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

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

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

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

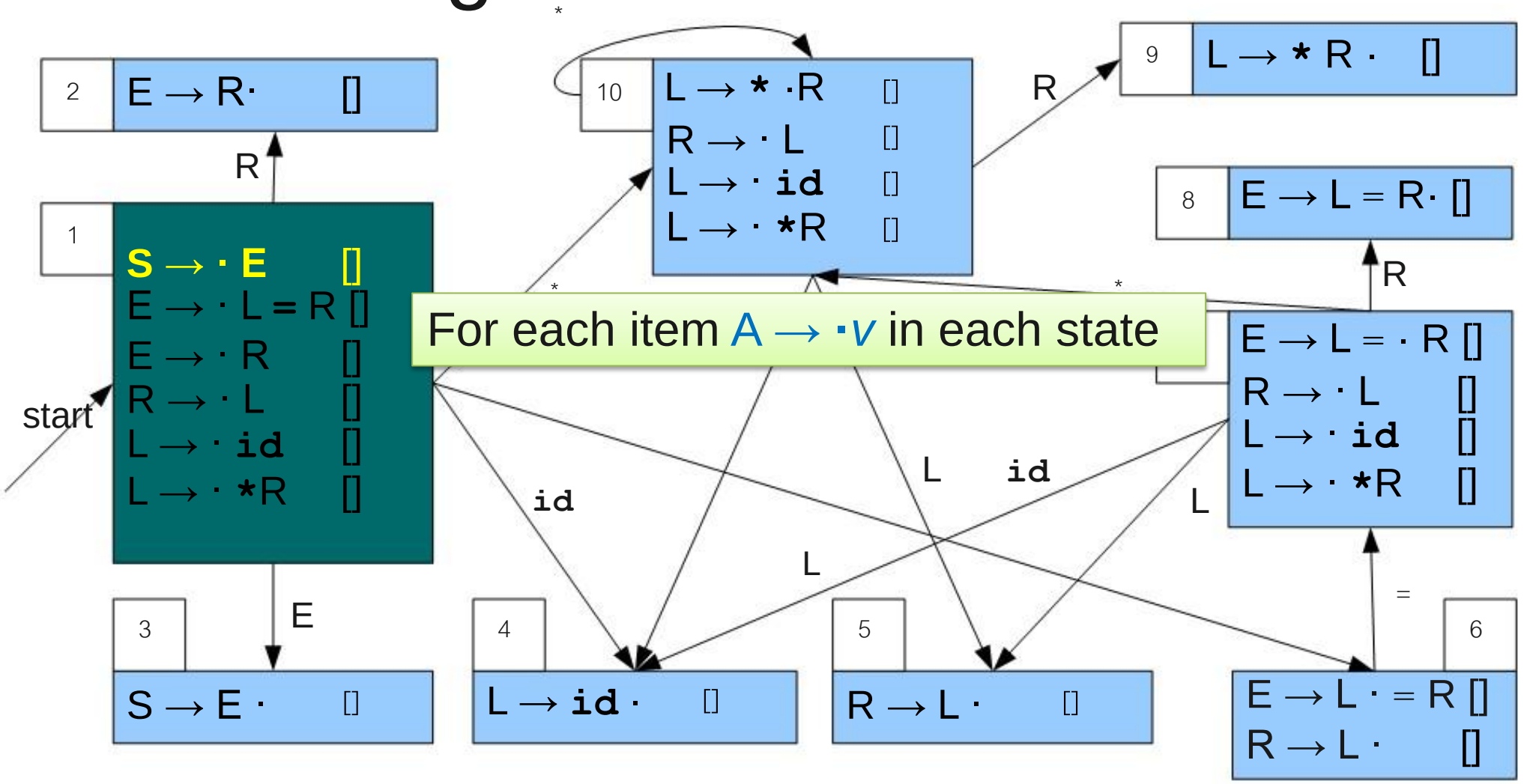
$L_{10\text{-}5} \rightarrow$ **id**

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

S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	= $	$	=	$	$	= $

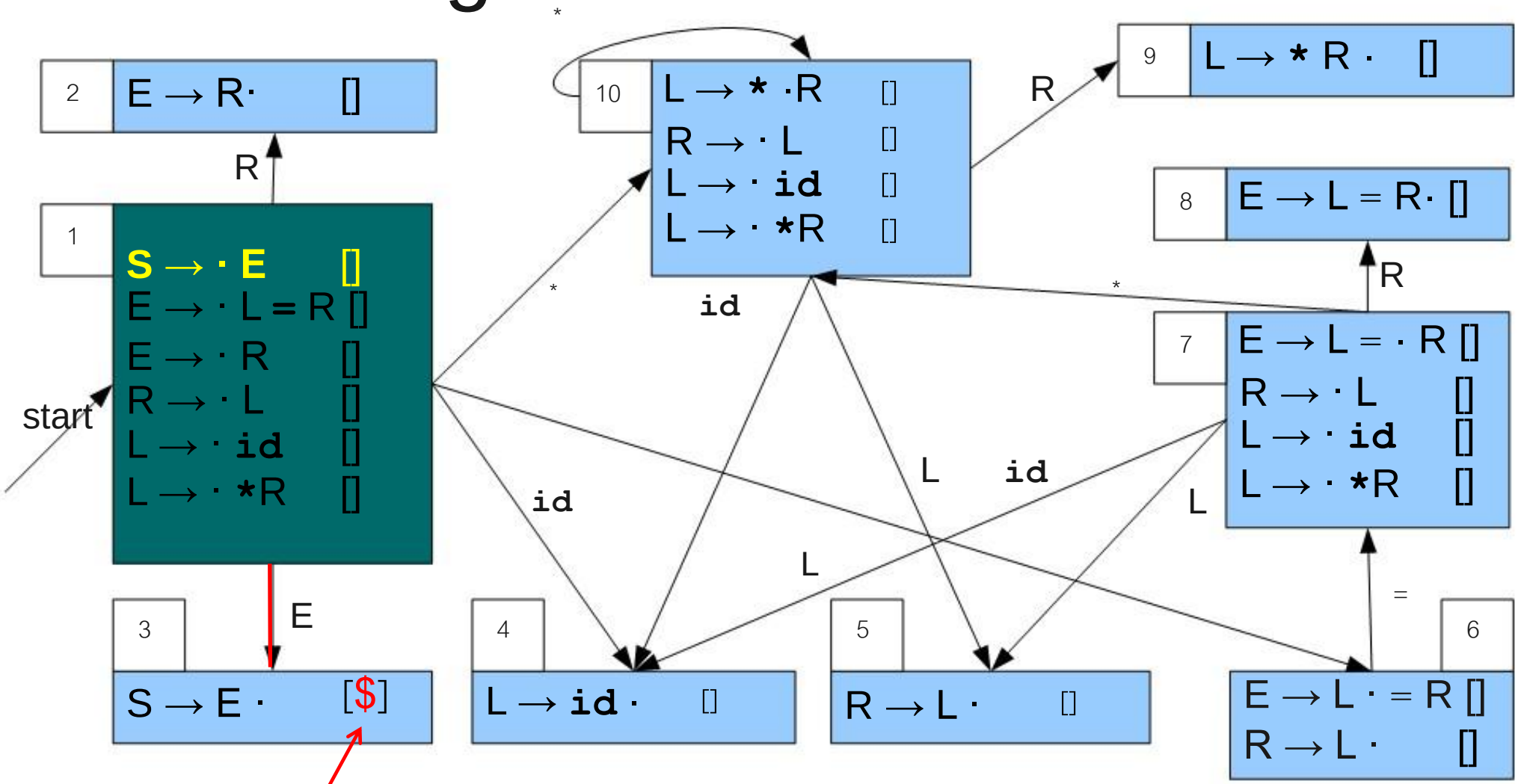
Lookahead Sets = Follow sets of augmented grammar.

Using Our LookAhead Sets



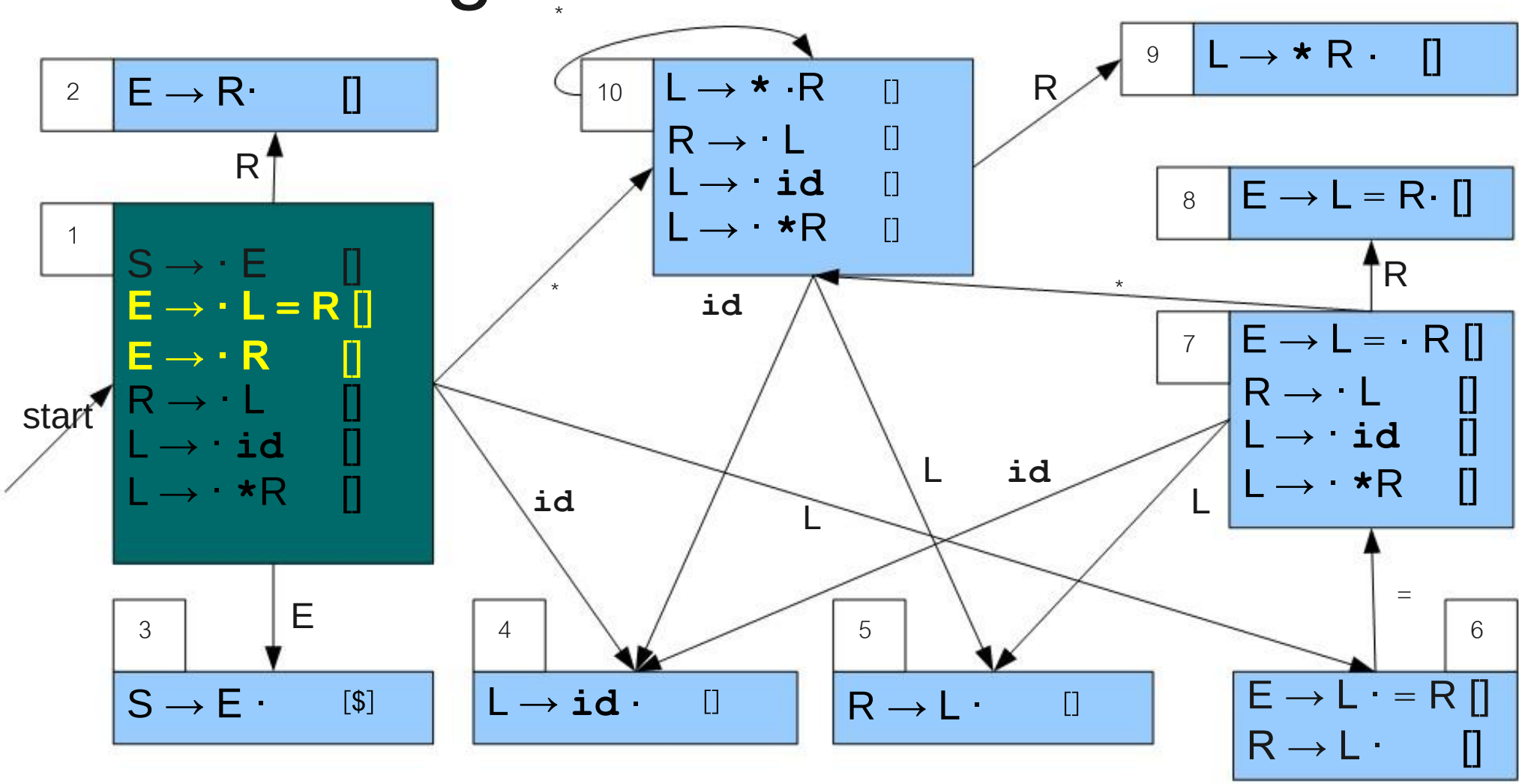
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=$	$	=	$	$	=

Using Our LookAhead Sets



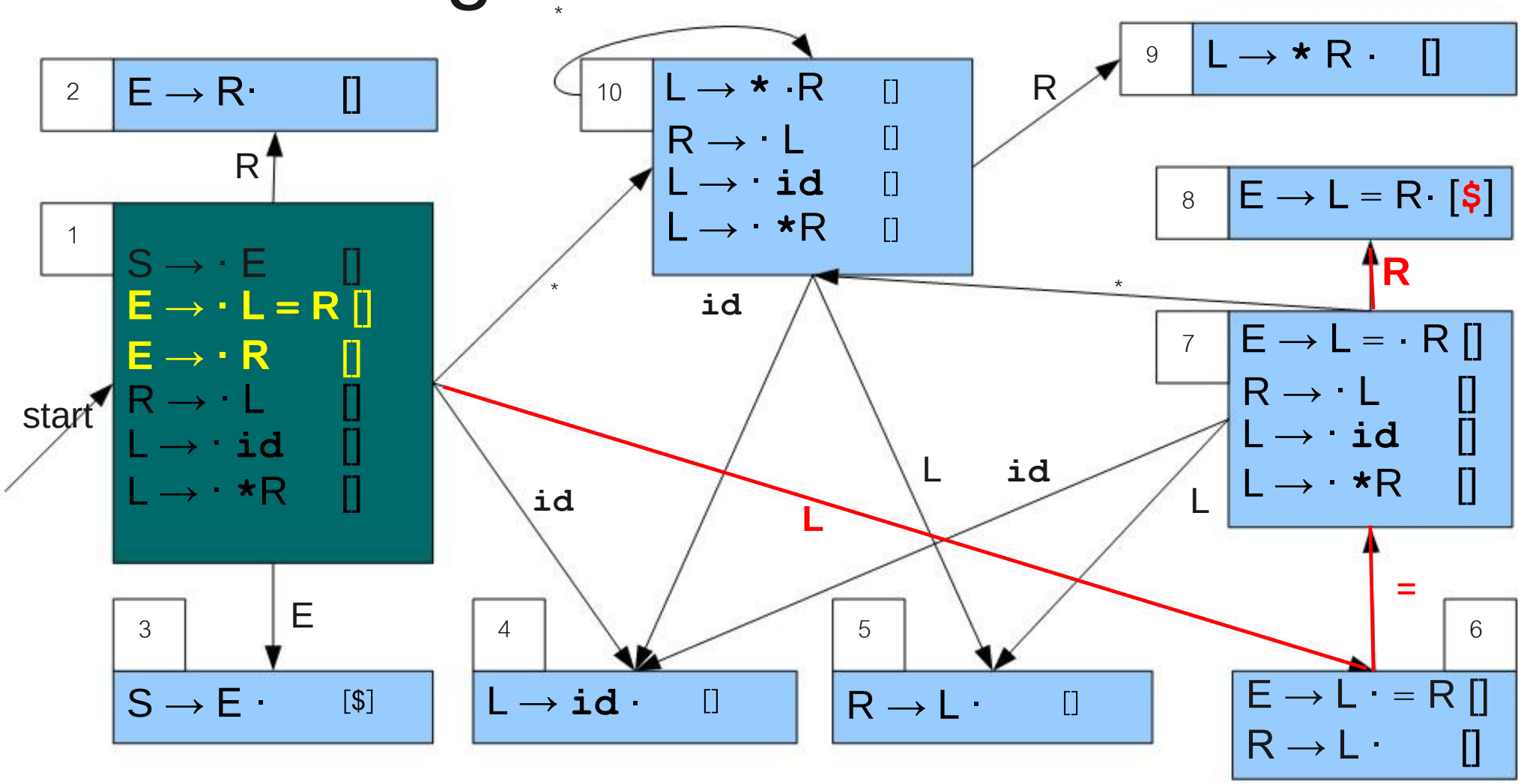
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=$	$	=	$	$	=

Using Our LookAhead Sets



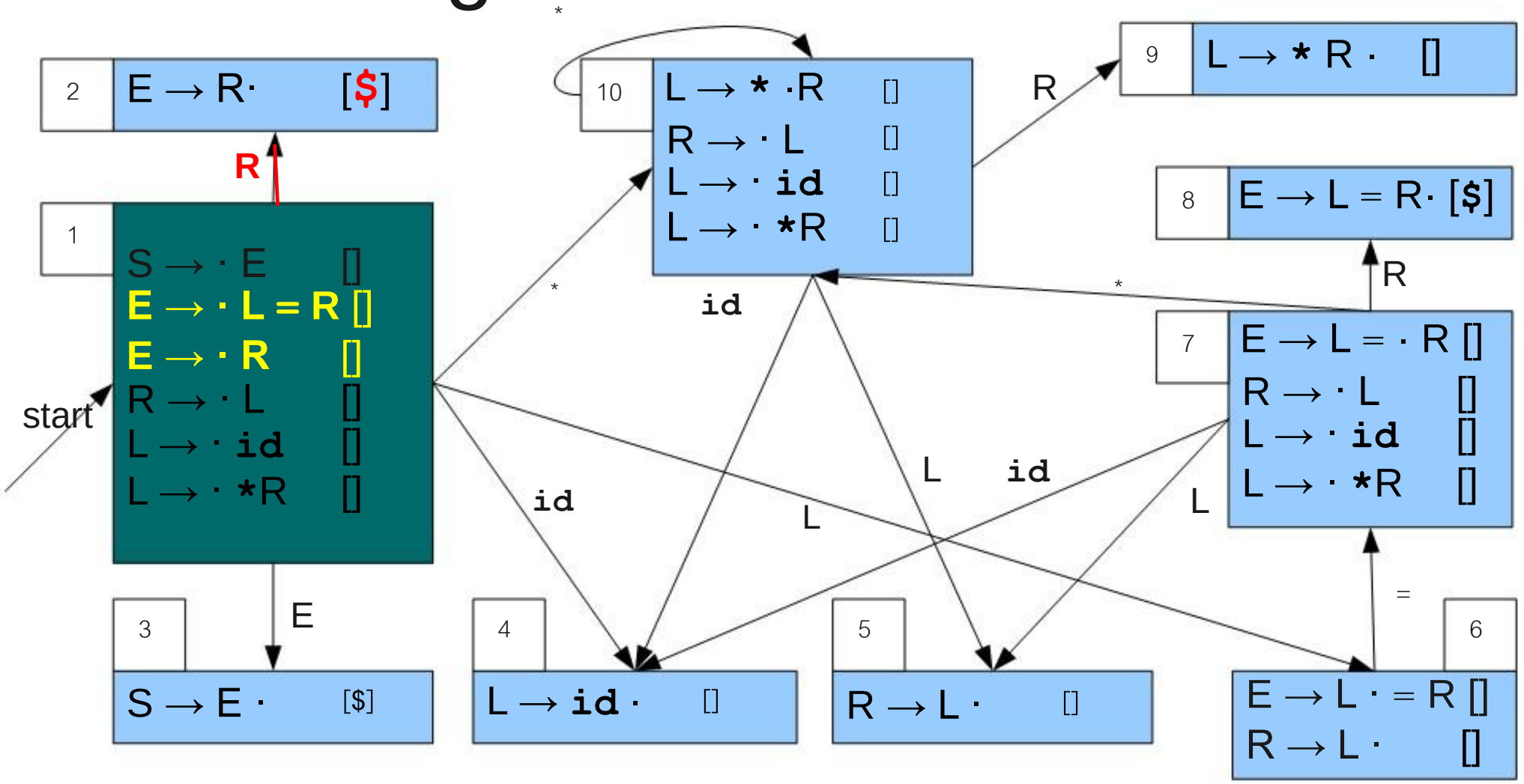
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=$	$	=	$	$	=

Using Our LookAhead Sets



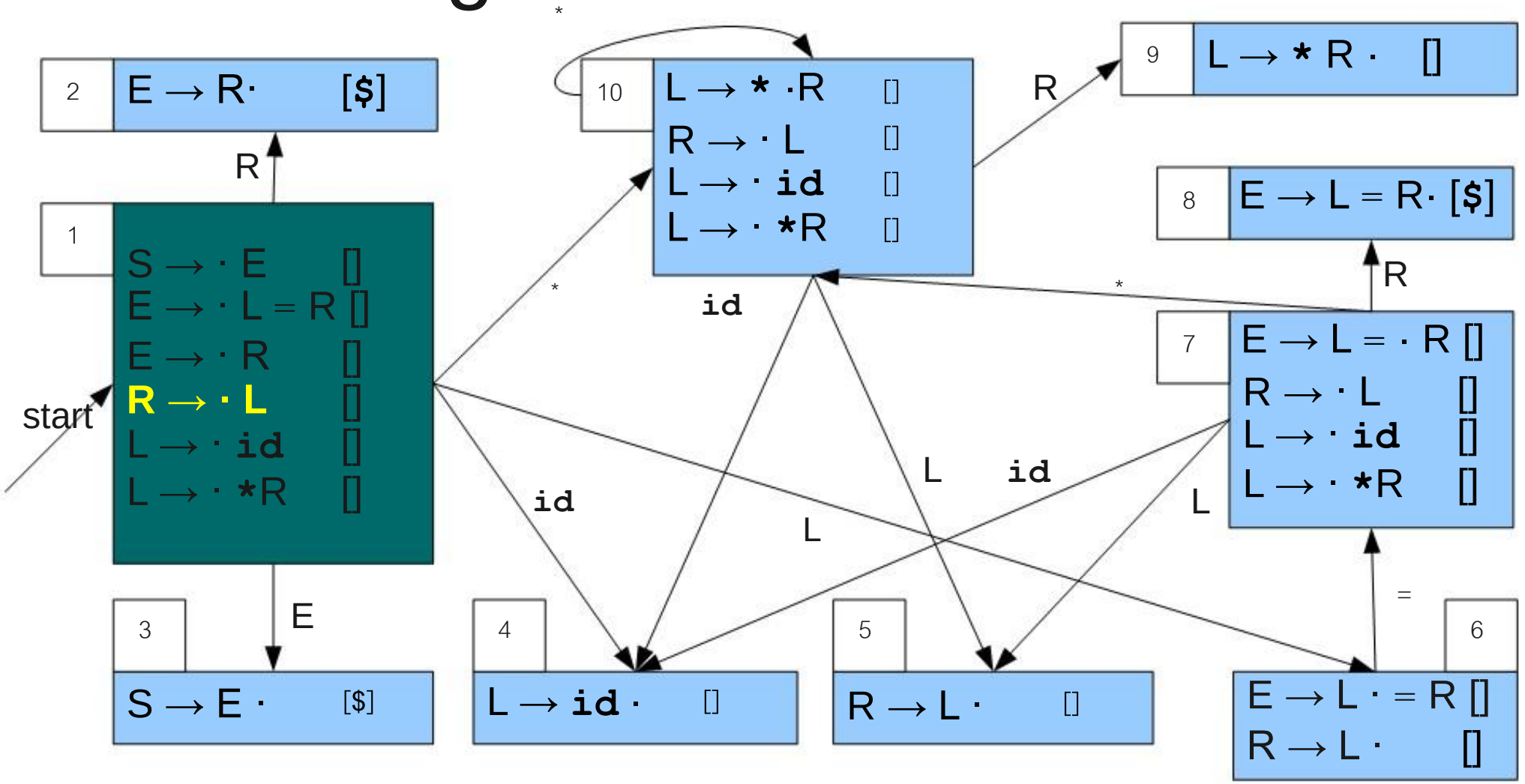
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=$	$	=	$	$	=

Using Our LookAhead Sets



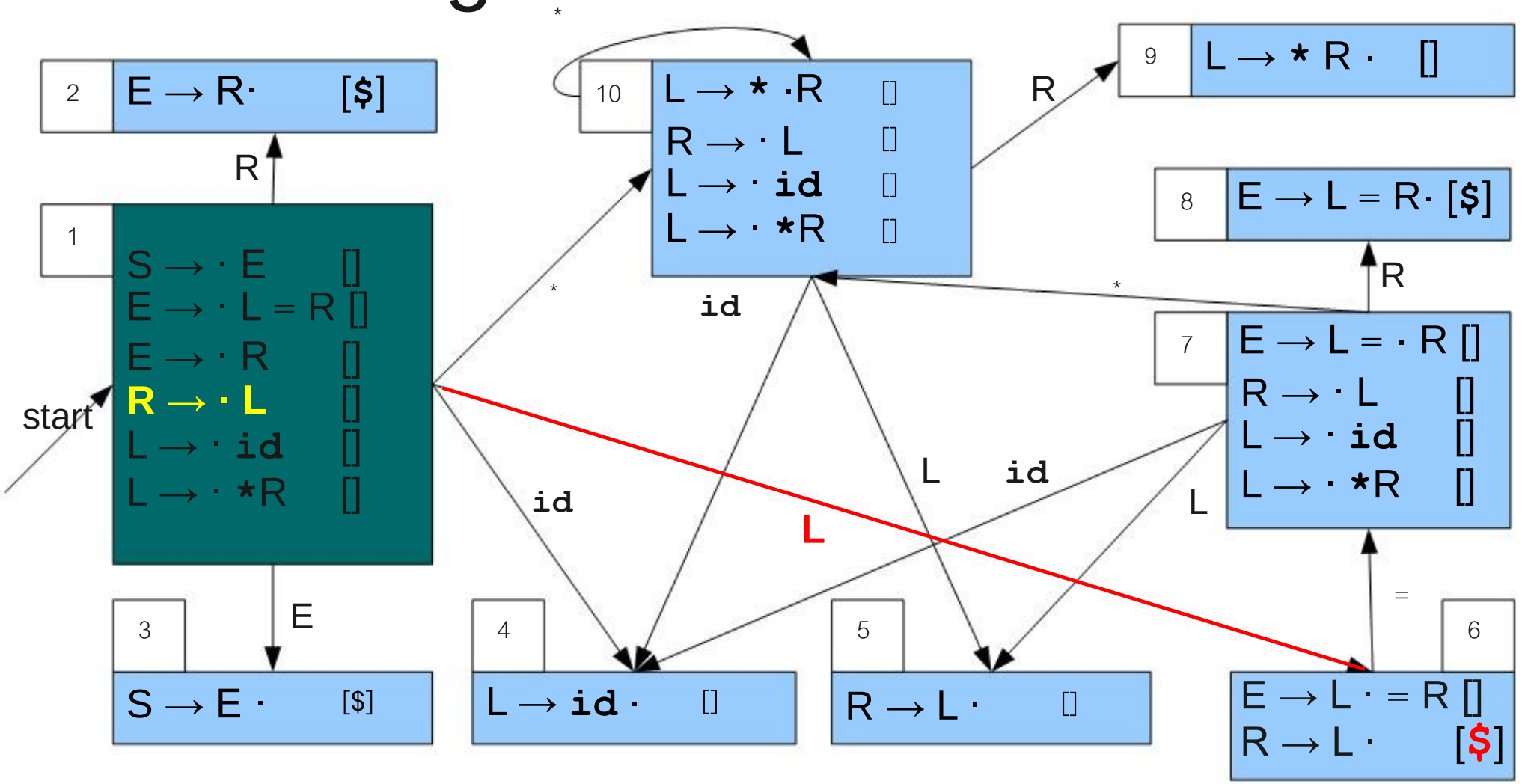
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=$	$	=	$	$	=

Using Our LookAhead Sets



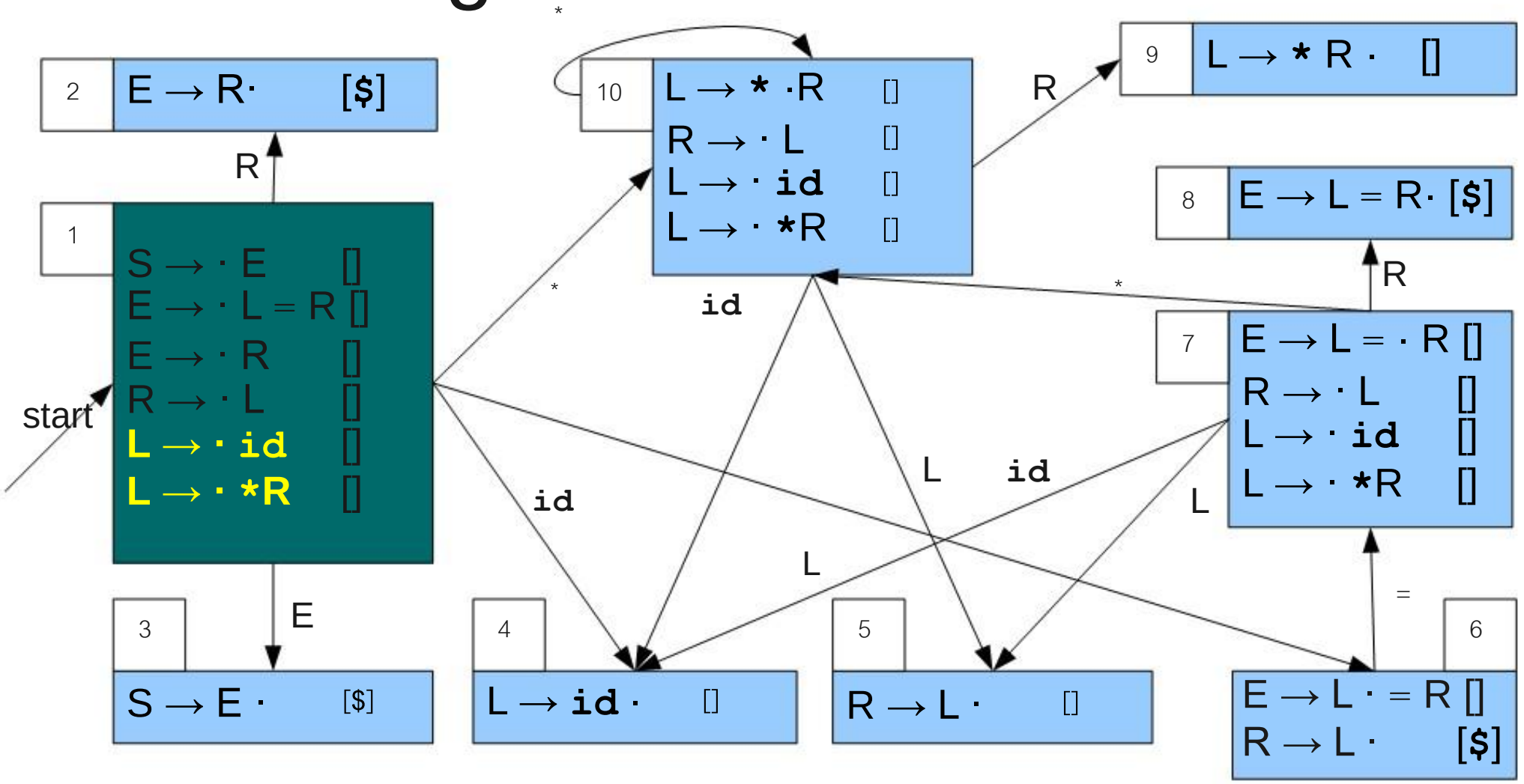
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=$	$	=	$	$	=

Using Our LookAhead Sets



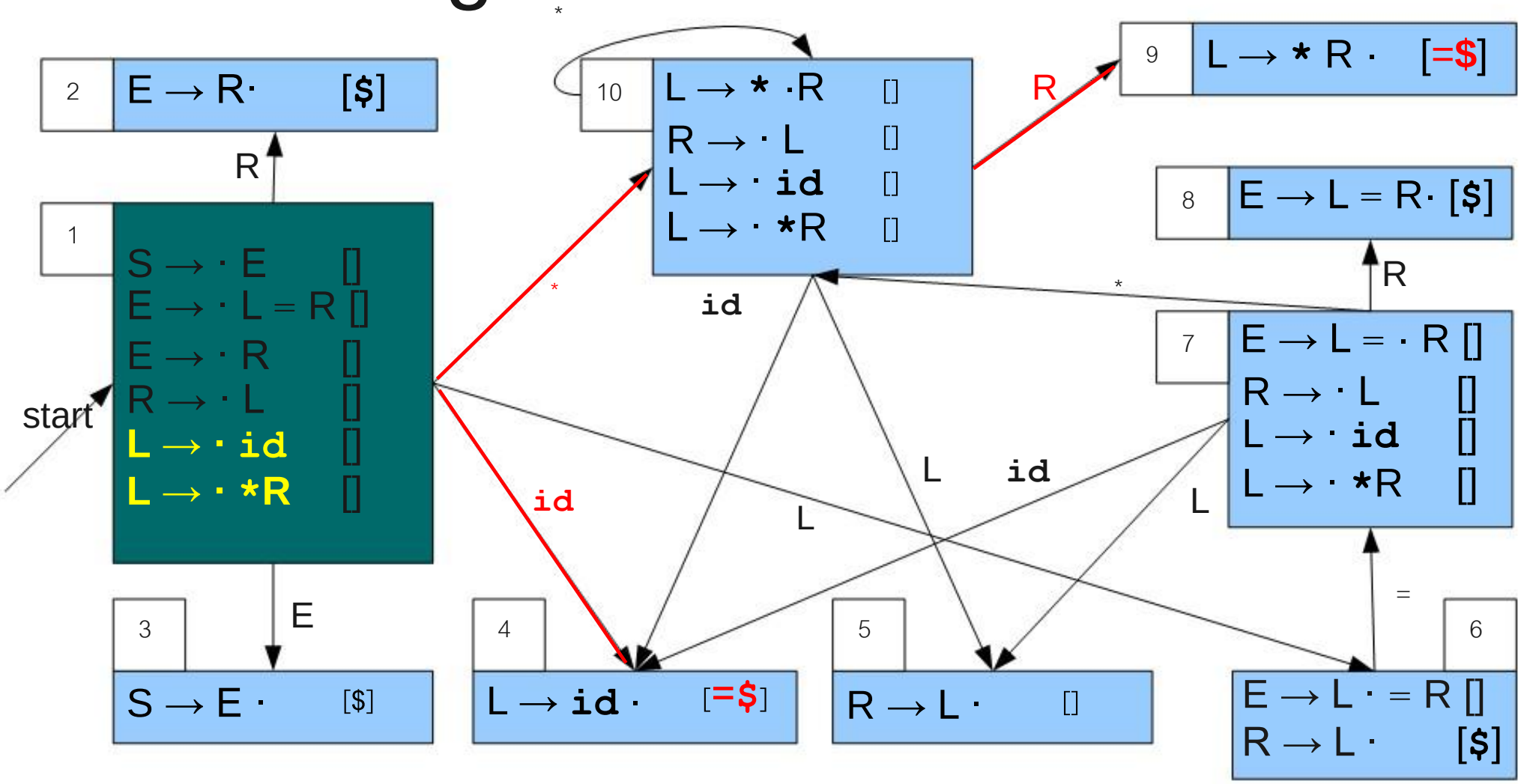
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=$	$	=	$	$	=

Using Our LookAhead Sets



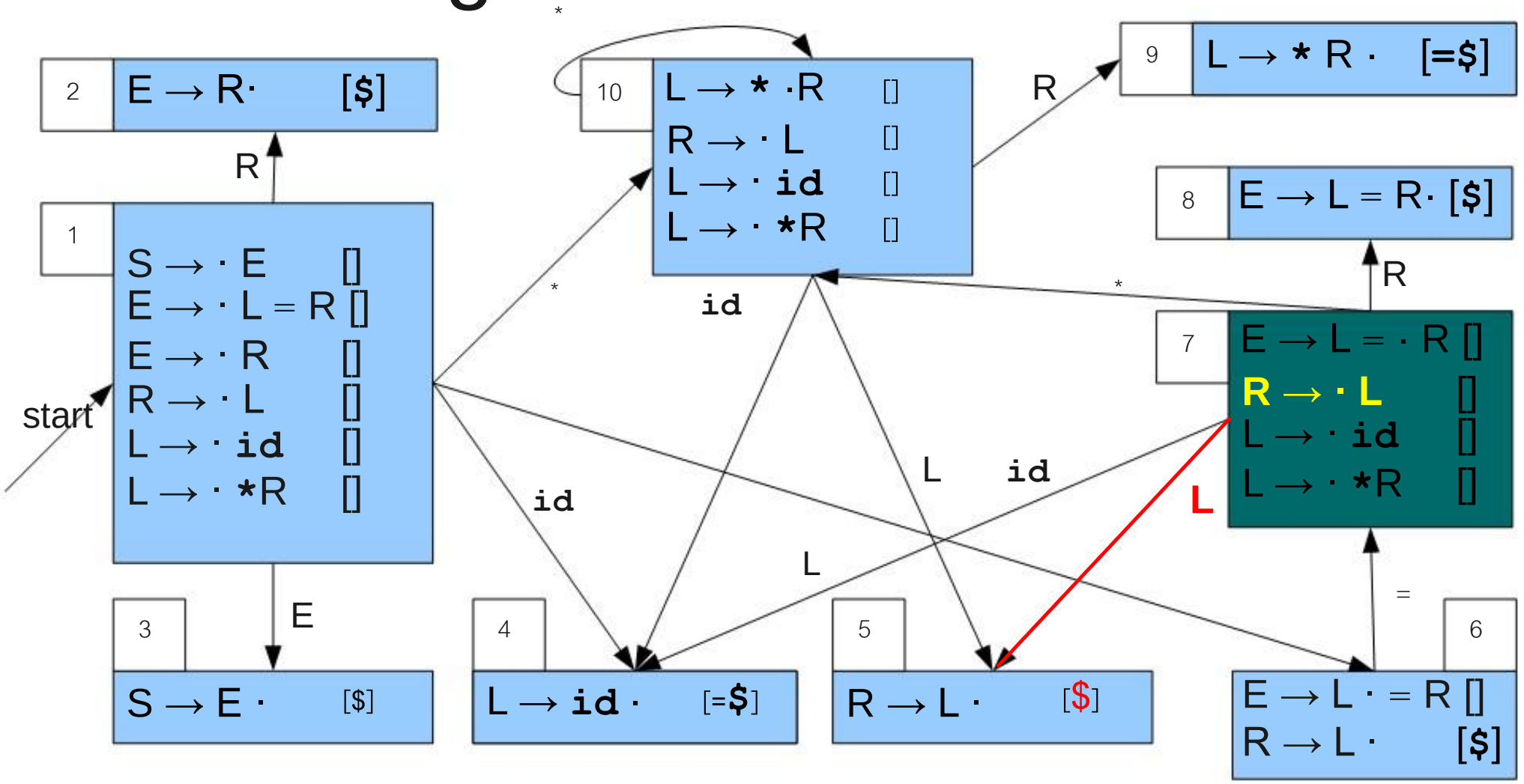
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=$	$	=	$	$	=

Using Our LookAhead Sets



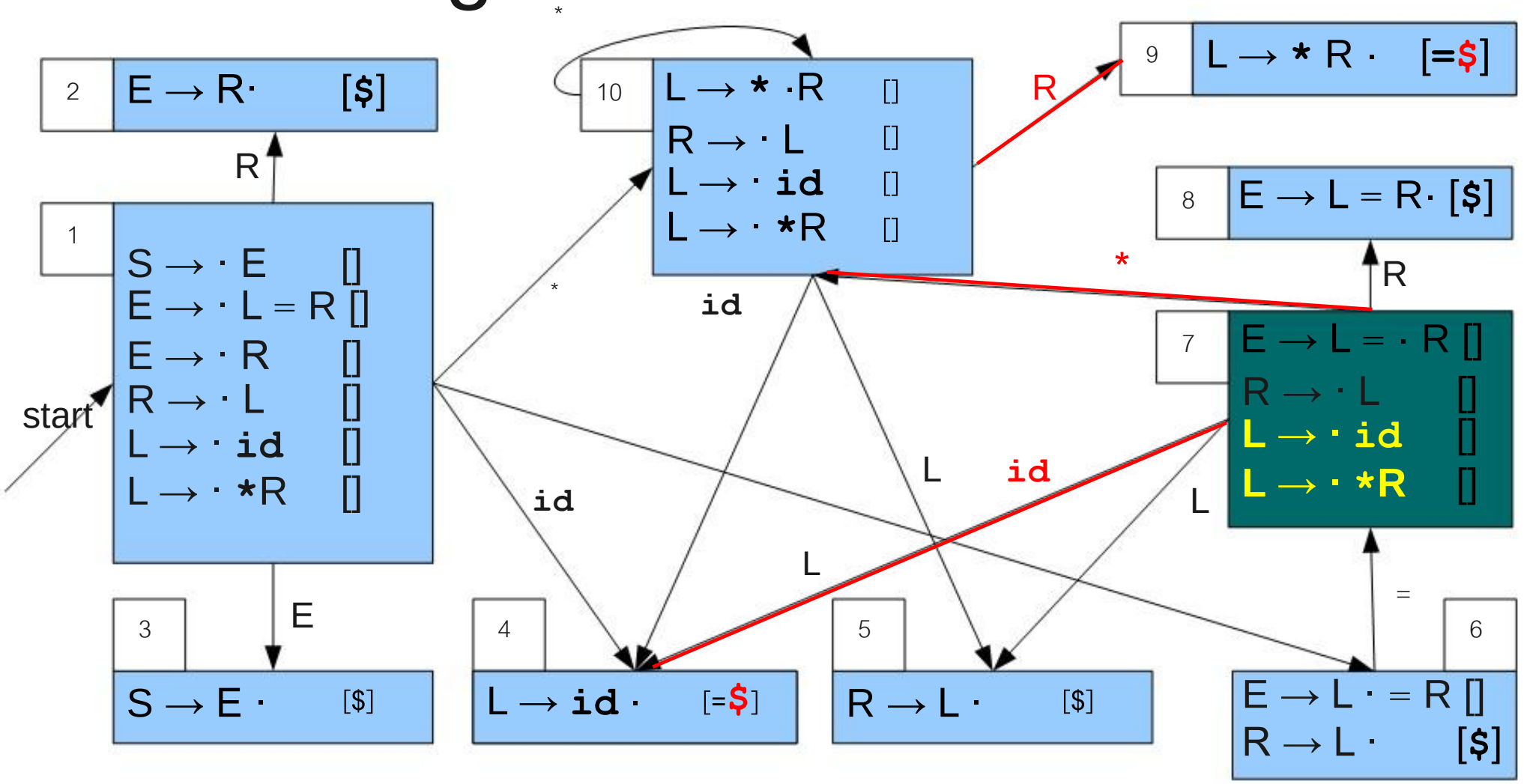
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=$	$	=	$	$	=

Using Our LookAhead Sets



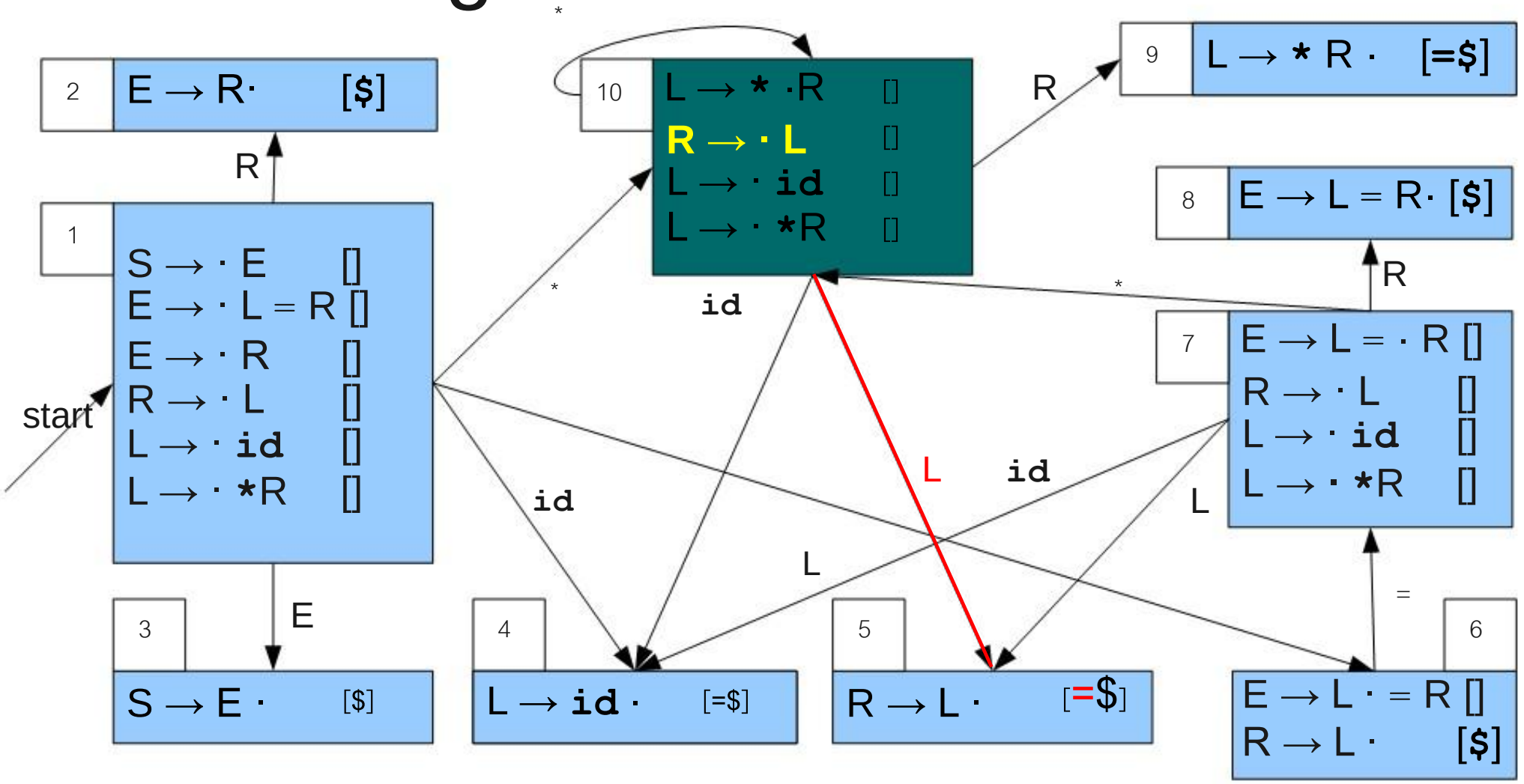
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=$	$	=	$	$	=

Using Our LookAhead Sets



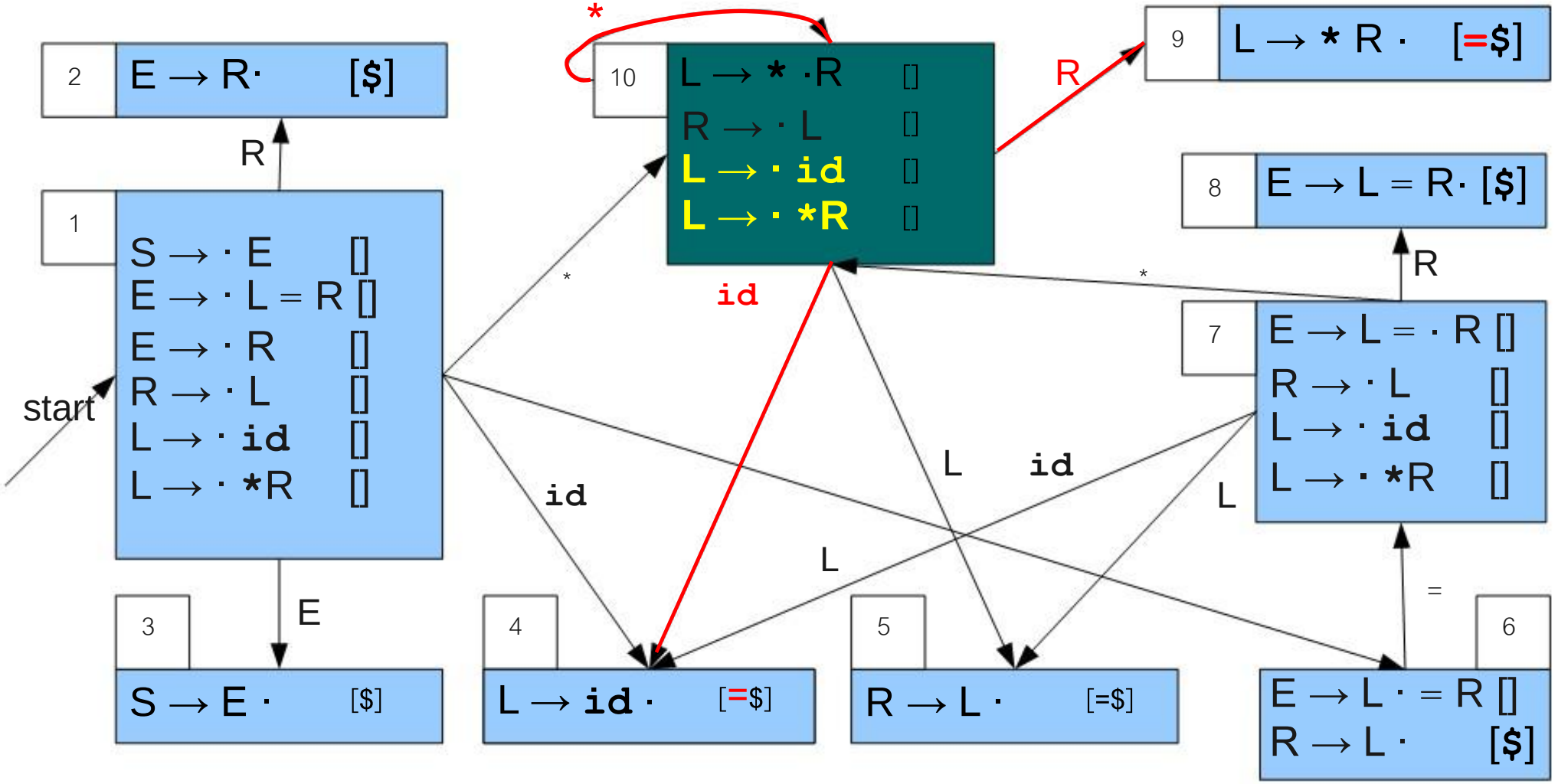
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=$	$	=	$	$	=

Using Our LookAhead Sets



S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=$	$	=	$	$	=

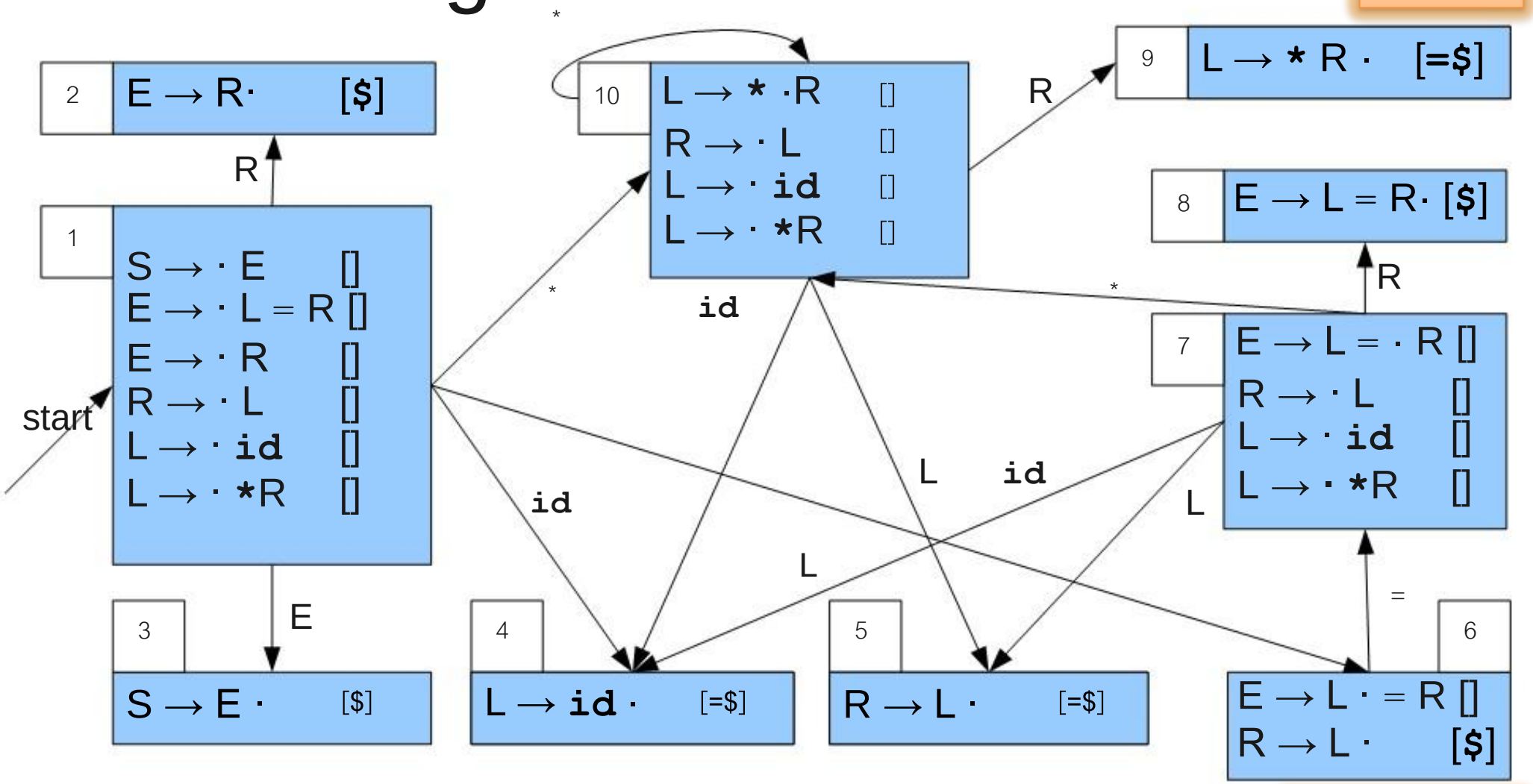
Using Our LookAhead Sets



S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=$	$	=	$	$	=

Using Our LookAhead Sets

Done



LALR(1)

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.

Error Handling

- 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?

Error Handling

E –> E + E | E * E | (E) | id

State on top of stack	Action					
	id	+	*	(	)	$
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	**e3**	r4	r4
4	s3	**e1**	**e1**	s2	**e2**	**e1**
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	**e3**	r3	r3	**e3**	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.

Error Handling

E –> E + E | E * E | (E) | id

State on top of stack	Action					
	id	+	*	(	)	$
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	**e3**	r4	r4
4	s3	**e1**	**e1**	s2	**e2**	**e1**
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	**e3**	r3	r3	**e3**	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.

Error Handling

E –> E + E | E * E | (E) | id

State on top of stack	Action					
	id	+	*	(	)	$
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	**e3**	r4	r4
4	s3	**e1**	**e1**	s2	**e2**	**e1**
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	**e3**	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.

Error Handling

E –> E + E | E * E | (E) | id

State on top of stack	Action					
	id	+	*	(	)	$
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	**e3**	r4	r4
4	s3	**e1**	**e1**	s2	**e2**	**e1**
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	**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.

Error Handling

Sample parsing on the erroneous input id +)

E –> E + E | E * E | (E) | id

State on top of stack	Action						Goto
	id	+	*	(	)	$	E
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	**e3**	r2	r2	**e3**	r2	r2	
9	**e3**	r3	r3	**e3**	r3	r3	

Stack	Input	Err Msg &Action
0	id+)$	

Error Handling

Sample parsing on the erroneous input id +)

E –> E + E | E * E | (E) | id

State on top of stack	Action						Goto
	id	+	*	(	)	$	E
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	**e3**	r2	r2	**e3**	r2	r2	
9	**e3**	r3	r3	**e3**	r3	r3	

Stack	Input	Err Msg &Action
0	id+)$	
0id3	+)$	

Error Handling

Sample parsing on the erroneous input id +)

E –> E + E | E * E | (E) | id

State on top of stack	Action						Goto
	id	+	*	(	)	$	E
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	**e3**	r2	r2	**e3**	r2	r2	
9	**e3**	r3	r3	**e3**	r3	r3	

Stack	Input	Err Msg &Action
0	id+)$	
0id3	+)$	
0E1	+)$	

Error Handling

Sample parsing on the erroneous input id +)

E –> E + E | E * E | (E) | id

State on top of stack	Action						Goto
	id	+	*	(	)	$	E
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	**e3**	r2	r2	**e3**	r2	r2	
9	**e3**	r3	r3	**e3**	r3	r3	

Stack	Input	Err Msg &Action
0	id+)$	
0id3	+)$	
0E1	+)$	
0E1+4	)$	-Unbalanced ‘)’ -Remove ‘)’

Error Handling

Sample parsing on the erroneous input id +)

E –> E + E | E * E | (E) | id

State on top of stack	Action						Goto
	id	+	*	(	)	$	E
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	**e3**	r2	r2	**e3**	r2	r2	
9	**e3**	r3	r3	**e3**	r3	r3	

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

Error Handling

Sample parsing on the erroneous input id +)

E –> E + E | E * E | (E) | id

State on top of stack	Action						Goto
	id	+	*	(	)	$	E
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	**e3**	r2	r2	**e3**	r2	r2	
9	**e3**	r3	r3	**e3**	r3	r3	

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

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

Error Handling

Sample parsing on the erroneous input id +)

E –> E + E | E * E | (E) | id

State on top of stack	Action						Goto
	id	+	*	(	)	$	E
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	**e3**	r2	r2	**e3**	r2	r2	
9	**e3**	r3	r3	**e3**	r3	r3	

Stack	Input	Err Msg &Action
0	id+)$	
0id3	+)$	
0E1	+)$	
0E1+4	)$	-Unbalanced ‘)’ -Remove ‘)’
0E1+4	$	-Missing opnd. - Push id 3
0E1+4id3	$	
0E1+4E7	$	

Error Handling

Sample parsing on the erroneous input id +)

E –> E + E | E * E | (E) | id

State on top of stack	Action						Goto
	id	+	*	(	)	$	E
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	**e3**	r2	r2	**e3**	r2	r2	
9	**e3**	r3	r3	**e3**	r3	r3	

Stack	Input	Err Msg &Action
0	id+)$	
0id3	+)$	
0E1	+)$	
0E1+4	)$	-Unbalanced ‘)’ -Remove ‘)’
0E1+4	$	-Missing opnd. - Push id 3
0E1+4id3	$	
0E1+4E7	$	
0E1	$	

Error Handling

Sample parsing on the erroneous input id +)

E –> E + E | E * E | (E) | id

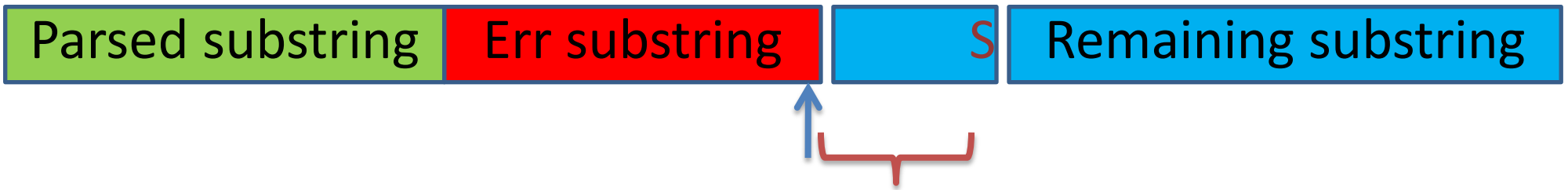
State on top of stack	Action						Goto
	id	+	*	(	)	$	E
0	s3	**e1**	**e1**	s2	**e2**	**e1**	1
1	**e3**	s4	s5	**e3**	**e2**	acc	
2	s3	**e1**	**e1**	s2	**e2**	**e1**	6
3	**e3**	r4	r4	**e3**	r4	r4	
4	s3	**e1**	**e1**	s2	**e2**	**e1**	7
5	s3	**e1**	**e1**	s2	**e2**	**e1**	8
6	**e3**	s4	s5	**e3**	s9	**e4**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

Stack	Input	Err Msg &Action
0	id+)$	
0id3	+)$	
0E1	+)$	
0E1+4	)$	-Unbalanced ‘)’ -Remove ‘)’
0E1+4	$	-Missing opnd. - Push id 3
0E1+4id3	$	
0E1+4E7	$	
0E1	$	

0E1	$	**Accept**

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.



Next Time

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