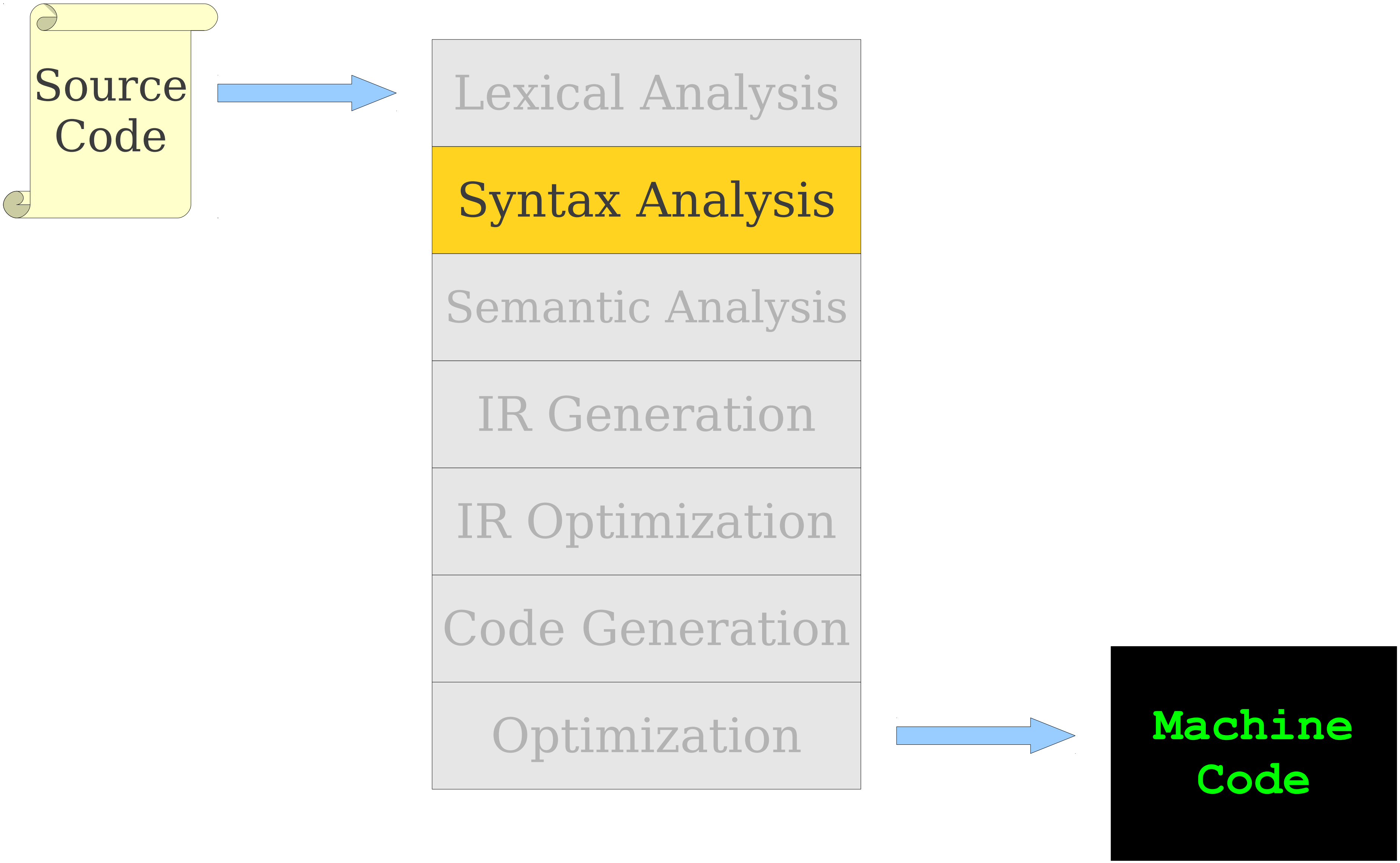
Bottom-Up Parsing, Part III

Announcements

- Written Assignment 2 out, due next Wednesday, July 18th at 5:00PM.
  - Explore the theory behind these parsing algorithms.
  - **Start early;** this problem set is longer and more involved than the previous one.

Where We Are



Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code

LR Conflicts

- A **shift/reduce conflict** is an error where a shift/reduce parser cannot tell whether to shift a token or perform a reduction.
  - Often happens when two productions overlap.
- A **reduce/reduce conflict** is an error where a shift/reduce parser cannot tell which of many reductions to perform.
  - Often the result of ambiguous grammars.
- A grammar whose handle-finding automaton contains a shift/reduce conflict or a reduce/reduce conflict is not LR(0).

What Conflicts Mean

- Recall: our automaton was constructed by looking for viable prefixes.
- Each accepting state represents a point where the handle might occur.
- A **shift/reduce** conflict is a state where the handle might occur, but we might actually need to keep searching.
- A **reduce/reduce** conflict is a state where we know we have found the handle, but can't tell which reduction to apply.

Why LR(0) is Weak

- LR(0) only accepts languages where the handle can be found with no **right context**.
- Our shift/reduce parser only looks to the left of the handle, not to the right.
- How do we exploit the tokens after a possible handle to determine what to do?

A Powerful Parser: **LR(1)**

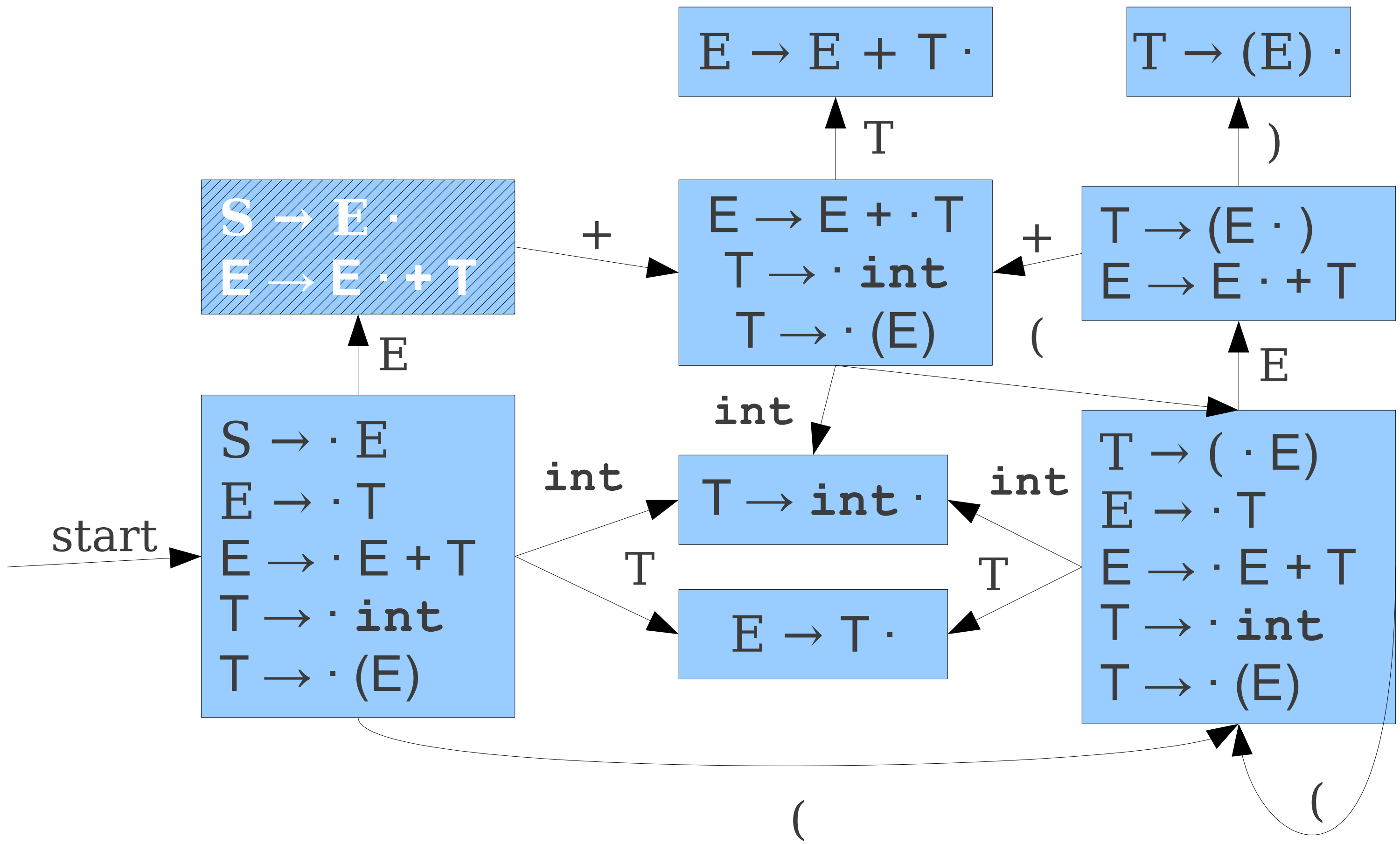
- Bottom-up predictive parsing with
  - **L**: **L**eft-to-right scan
  - **R**: **R**ightmost derivation
  - (**1**): One token lookahead
- *Substantially* more powerful than the other methods we've covered so far (more on that later).
- Tries to more intelligently find handles by using a lookahead token at each step.

LR(1) Parsing: The Intuition

$\mathbf{S} \rightarrow \mathbf{E}$
$\mathbf{E} \rightarrow \mathbf{T}$
$\mathbf{E} \rightarrow \mathbf{E}$ **`+`** $\mathbf{T}$
$\mathbf{T} \rightarrow$ **`int`**
$\mathbf{T} \rightarrow$ **`(`**$\mathbf{E}$**`)`**

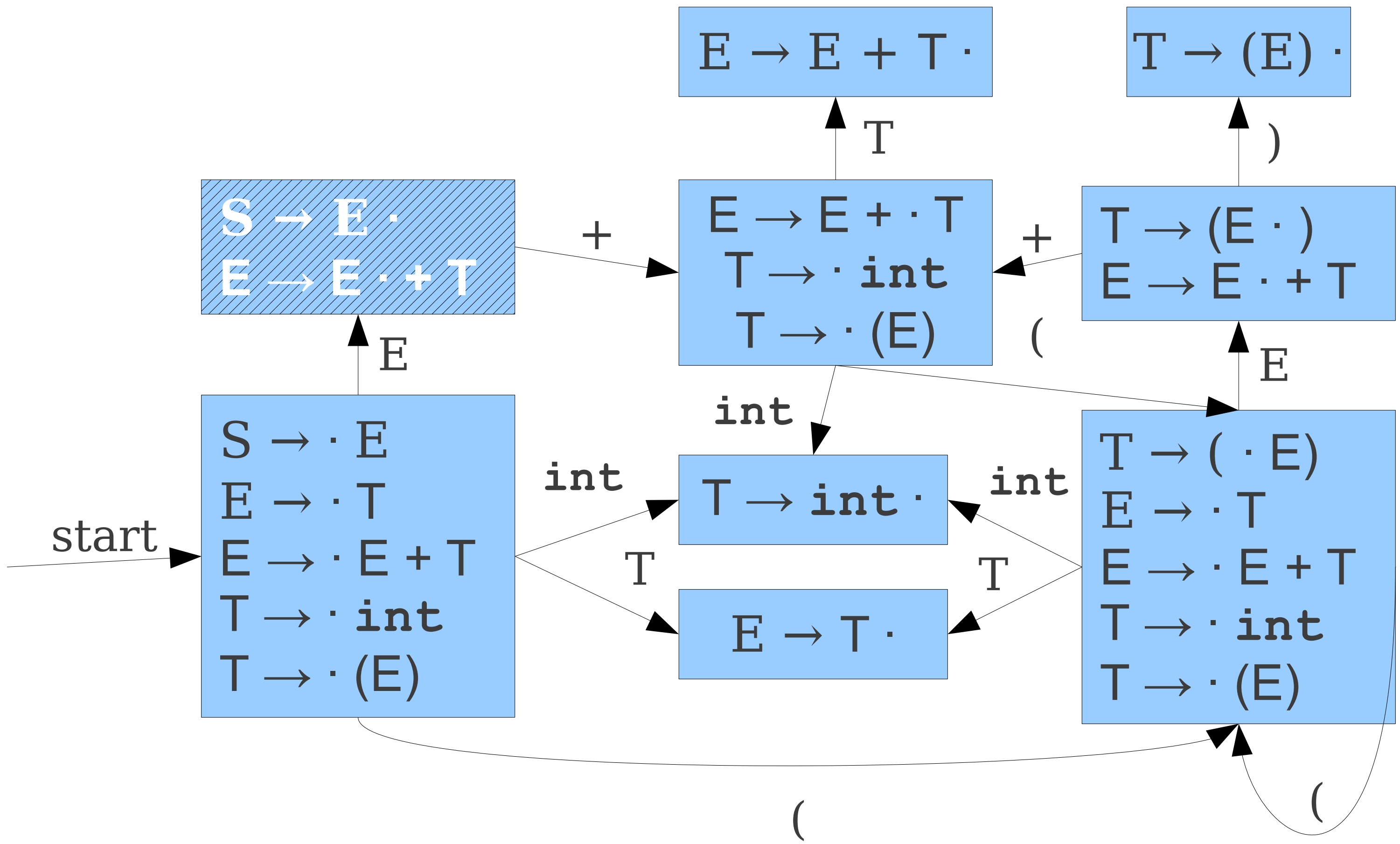
LR(1) Parsing: The Intuition

$S \to E$
$E \to T$
$E \to E + T$
$T \to$ int
$T \to$ (E)



LR(1) Parsing: The Intuition

S → **E**
E → **T**
E → **E** **+** **T**
T → **`int`**
T → **(E)**

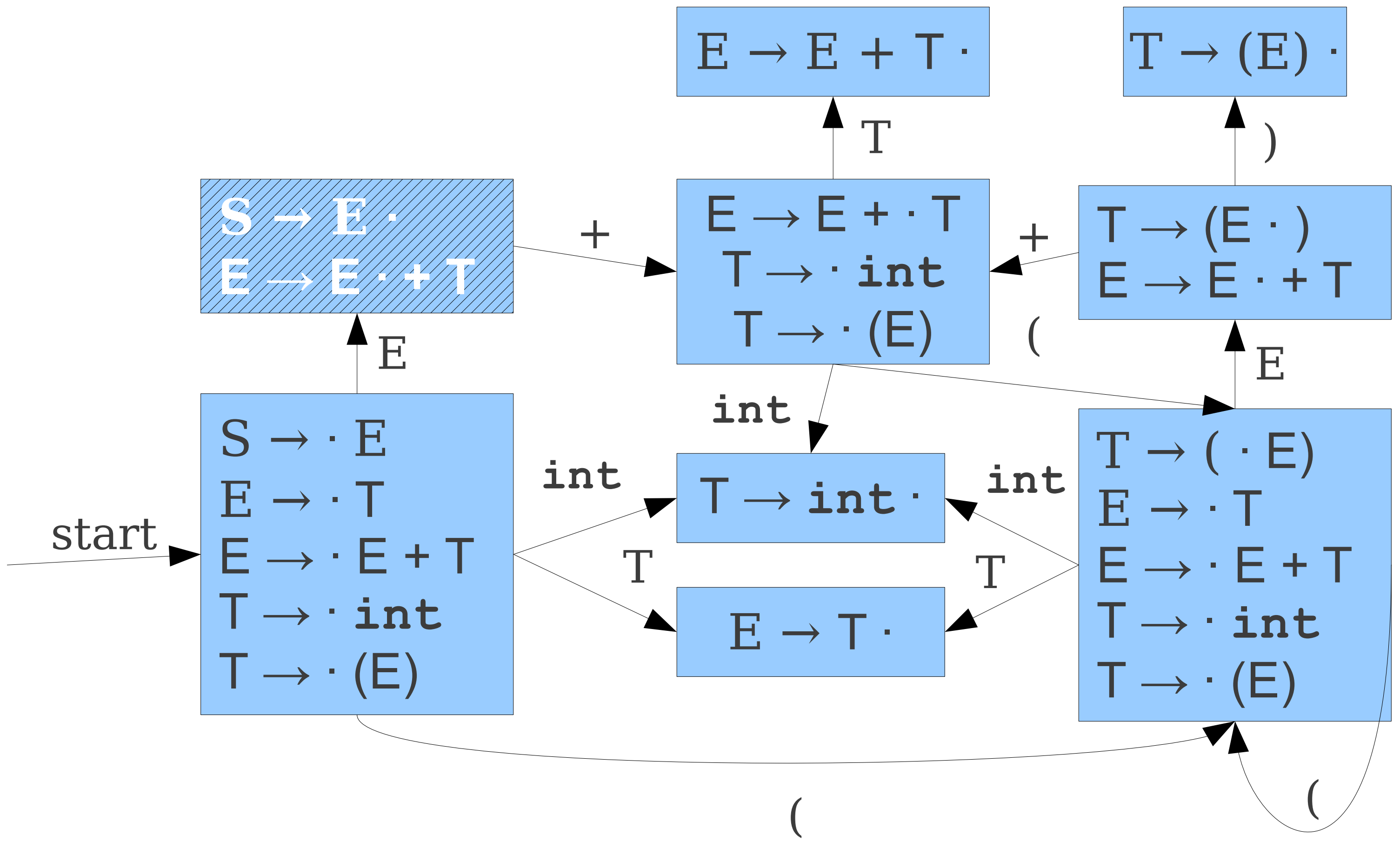


int	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → **E**
E → **T**
E → **E + T**
T → **int**
T → **(E)**

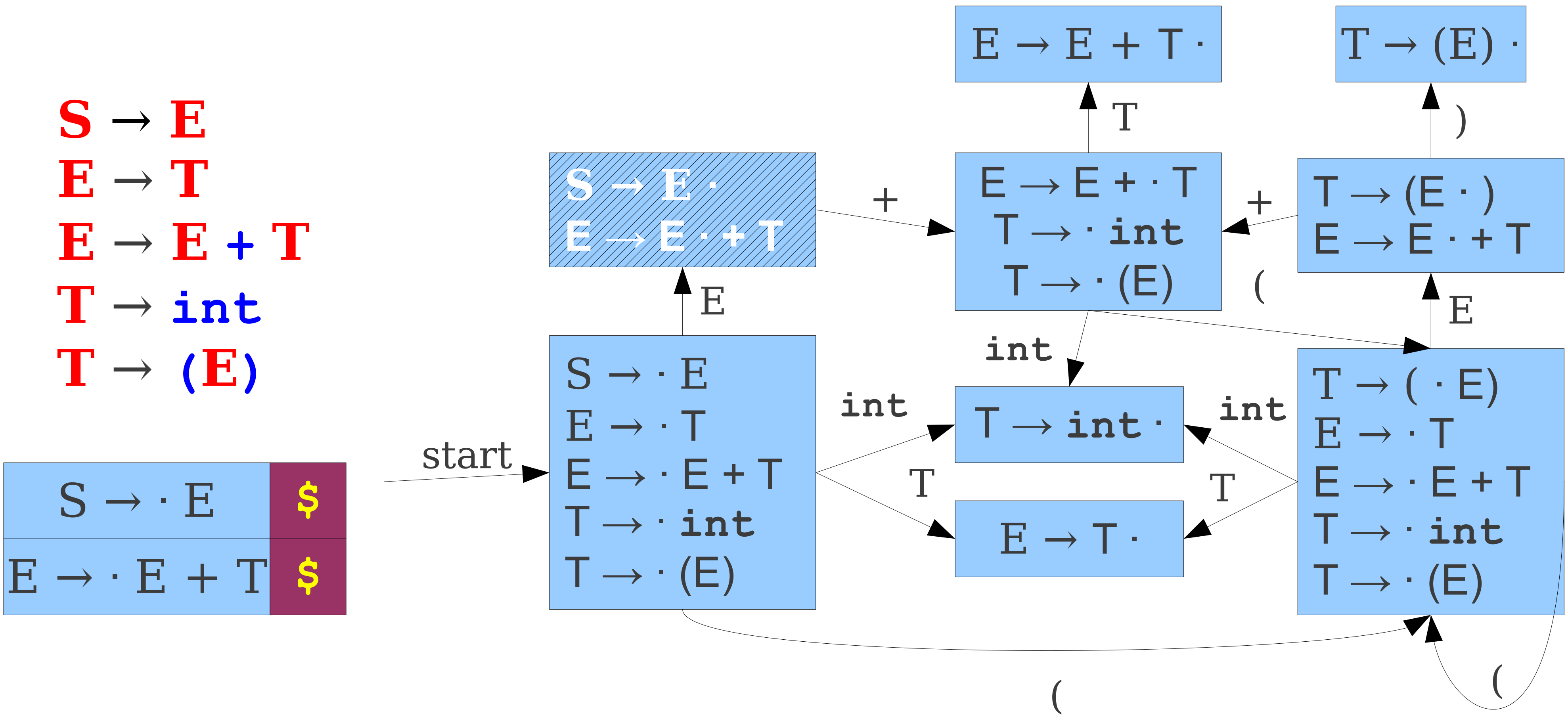
S → · E	$



int	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

$S \to E$
$E \to T$
$E \to E + T$
$T \to$ **int**
$T \to (E)$

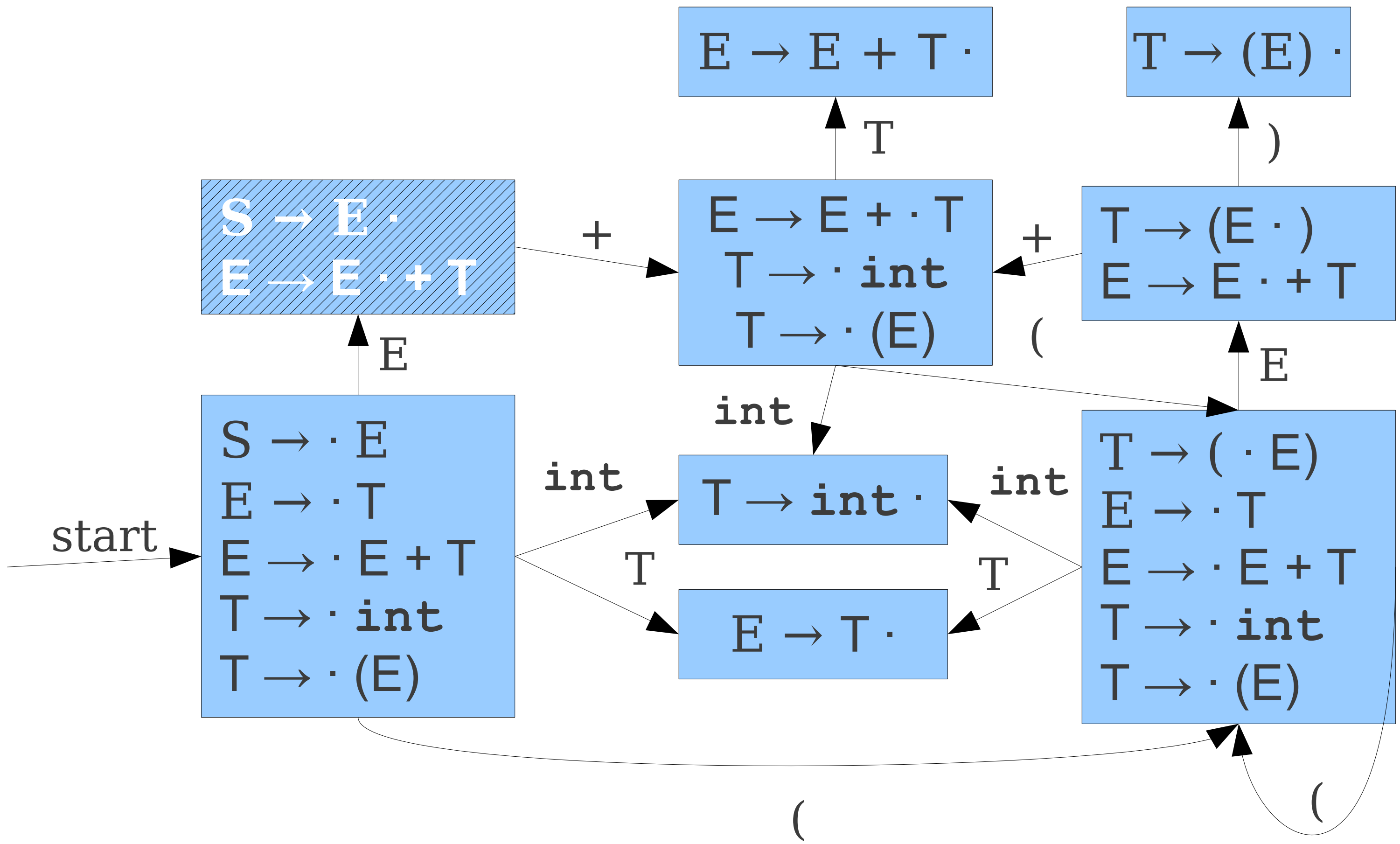


int	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → **E**
E → **T**
E → **E** **+** **T**
T → **int**
T → **(E)**

S → · E	$
E → · E + T	$
E → · T	+

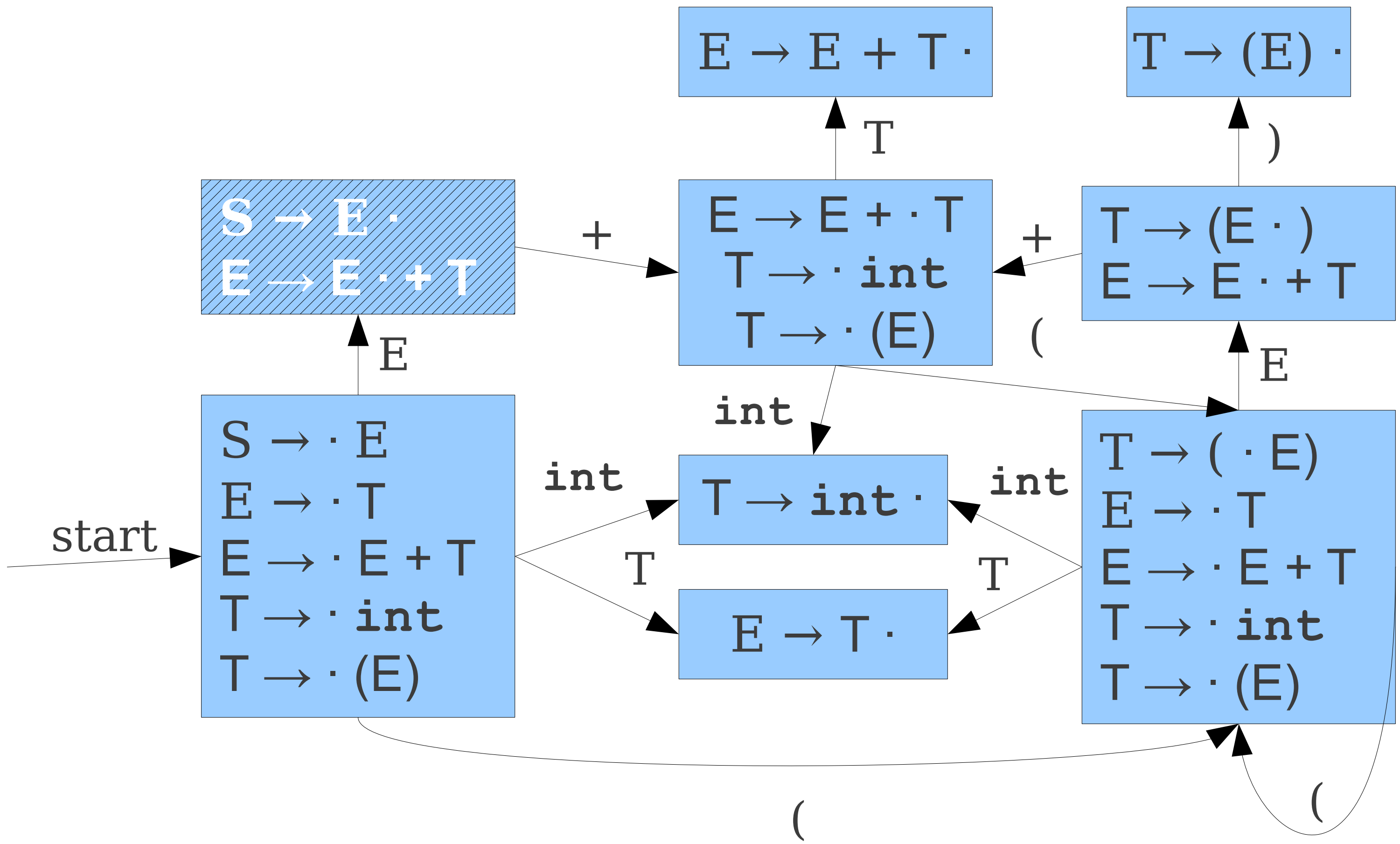


int	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → E
E → T
E → E + T
T → int
T → (E)

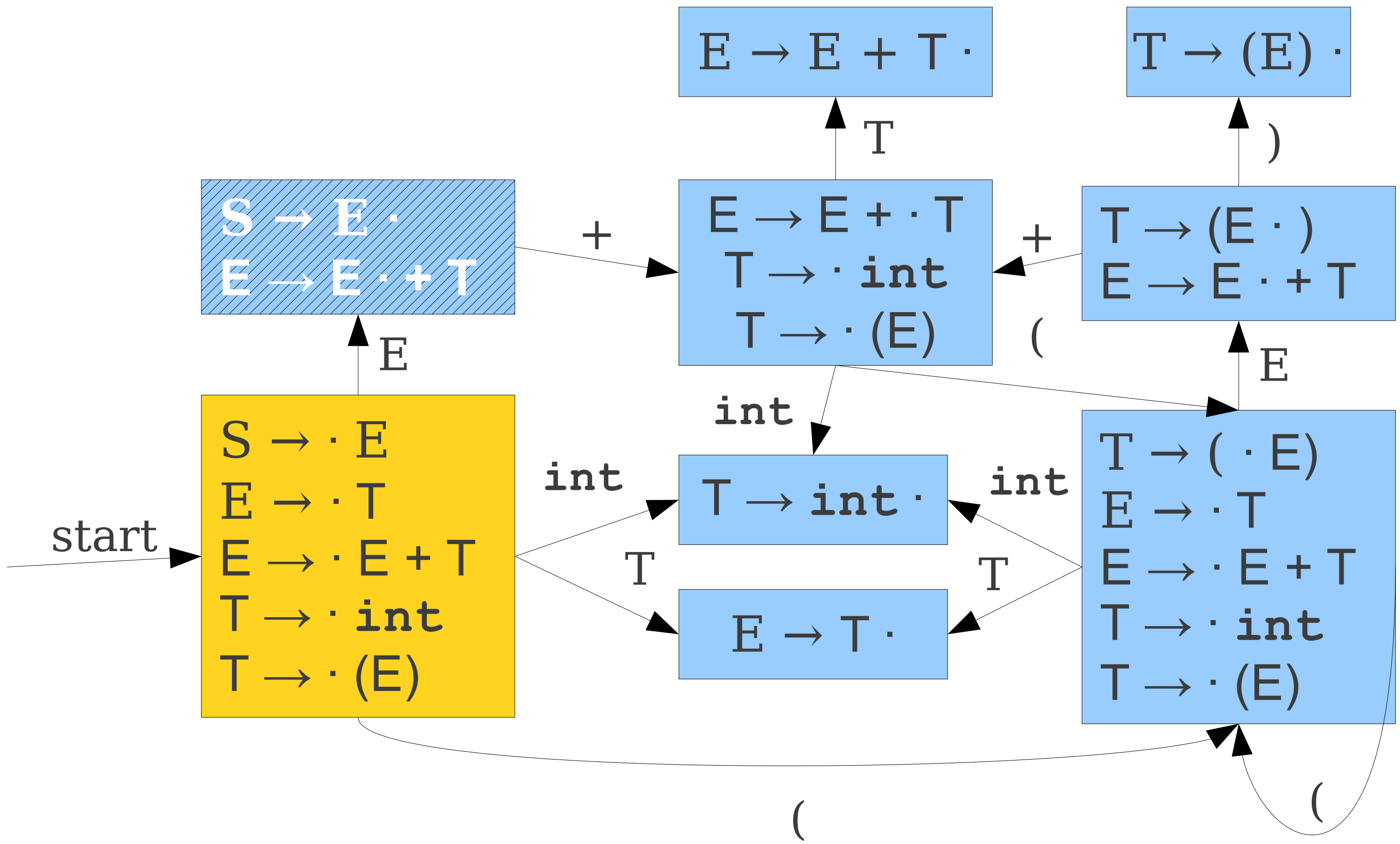
S → · E	$
E → · E + T	$
E → · T	+
T → · int	+



int	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → **E**
E → **T**
E → **E + T**
T → **`int`**
T → **(E)**

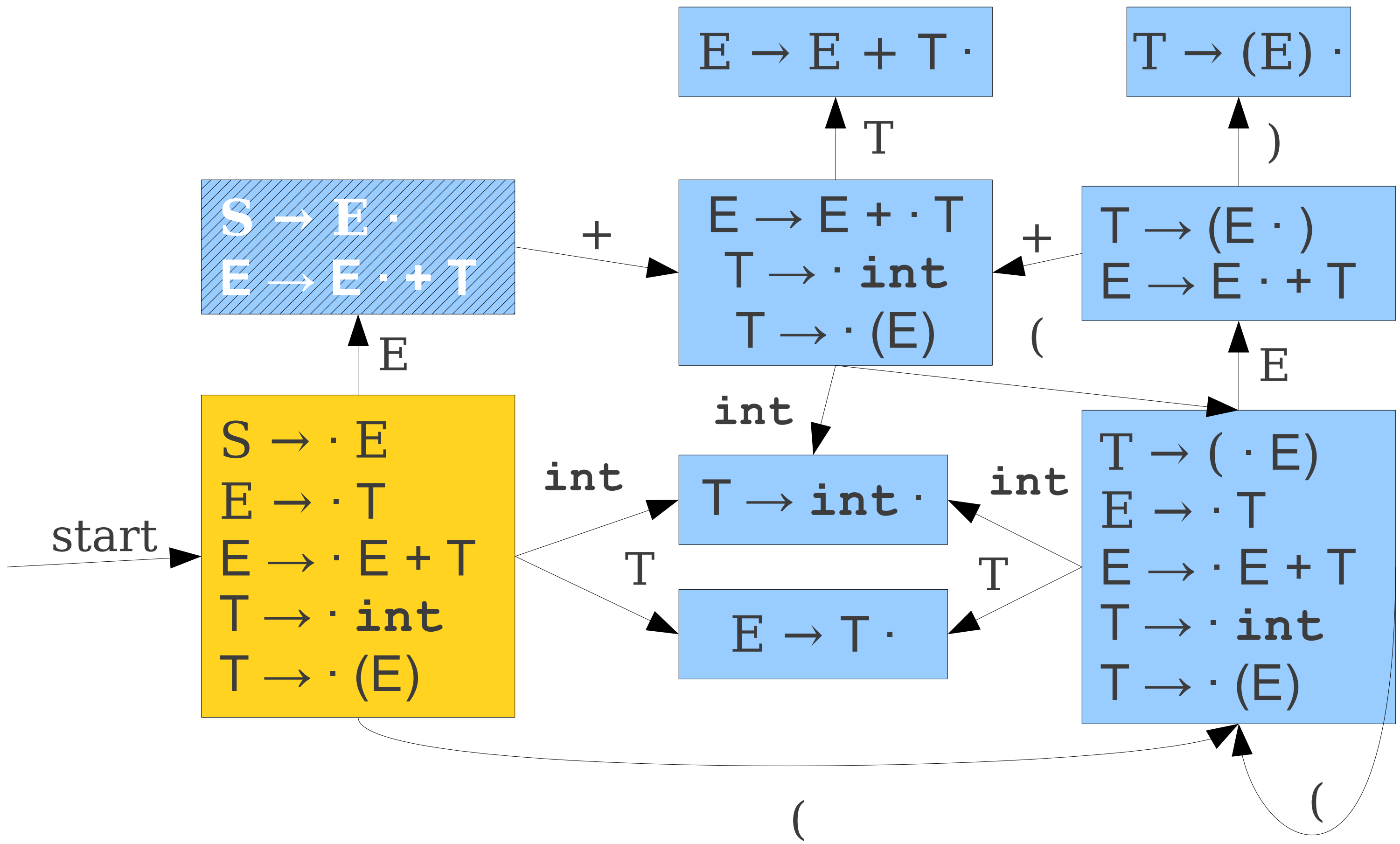


S → · E	$
E → · E + T	$
E → · T	+
T → · `int`	+

`int`	+	(	`int`	+	`int`	+	`int`	)	$

LR(1) Parsing: The Intuition

S → **E**
E → **T**
E → **E** + **T**
T → `int`
T → `(`**E**`)`



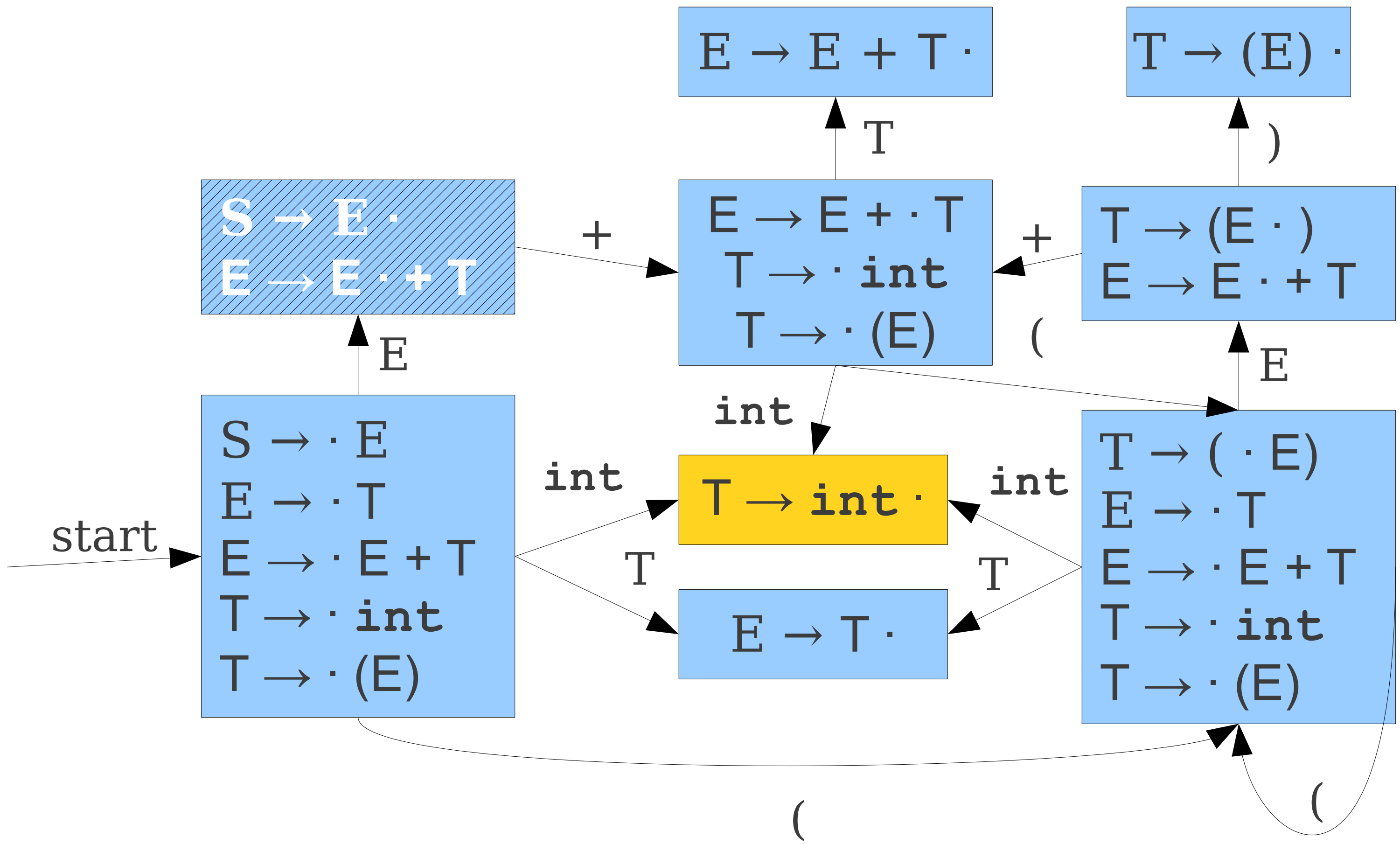
S → · E	$
E → · E + T	$
E → · T	+
T → · `int`	+

`int`	`+`	`(`	`int`	`+`	`int`	`+`	`int`	`)`	`$`

LR(1) Parsing: The Intuition

S → **E**
E → **T**
E → **E + T**
T → **int**
T → **(E)**

S → · E	$
E → · E + T	$
E → · T	+
T → · int	+

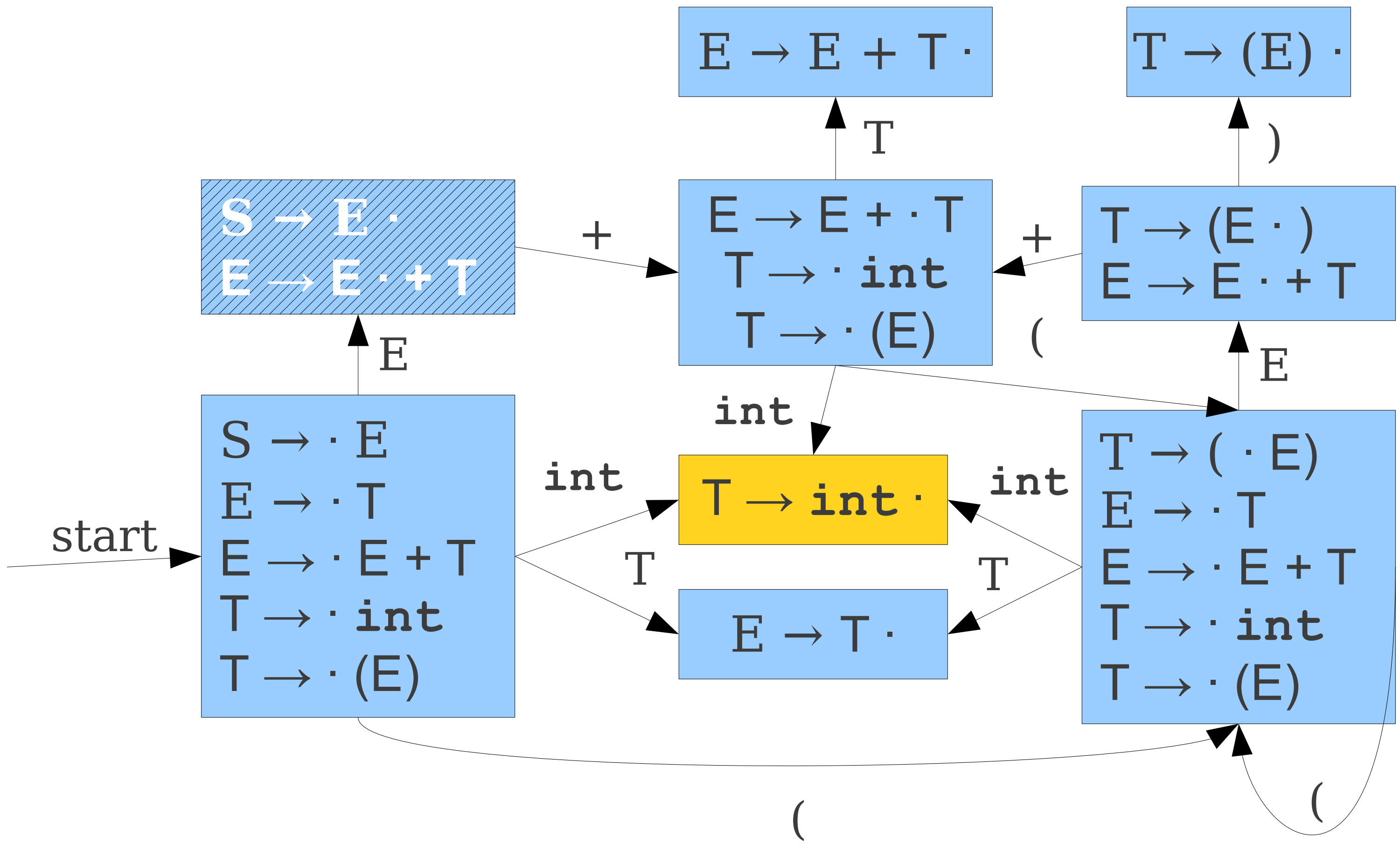


int	|	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → **E**
E → **T**
E → **E + T**
T → **int**
T → **(E)**

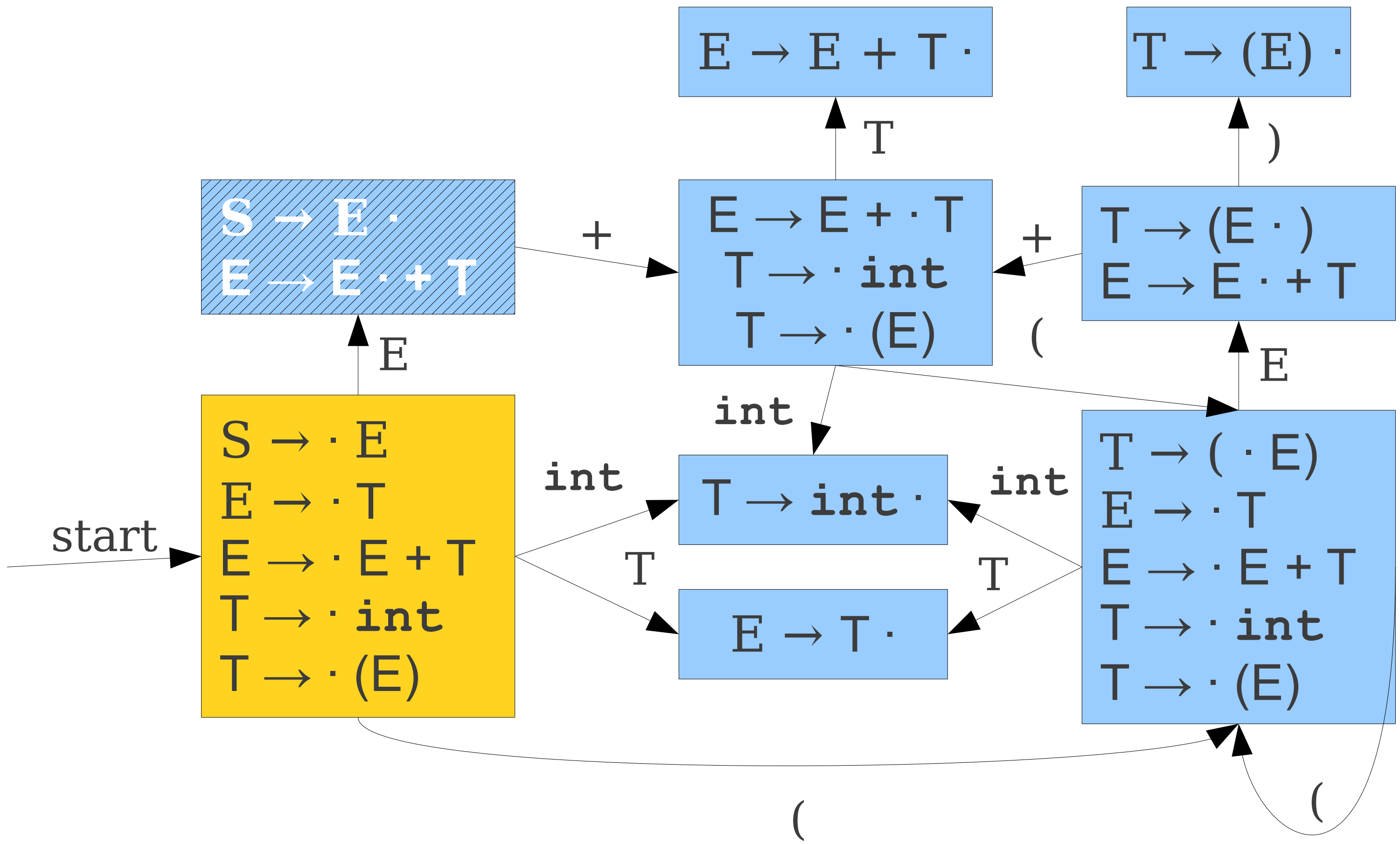
S → · E	$
E → · E + T	$
E → · T	+
T → int ·	+



int	|	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → **E**
E → **T**
E → **E + T**
T → **`int`**
T → **(E)**



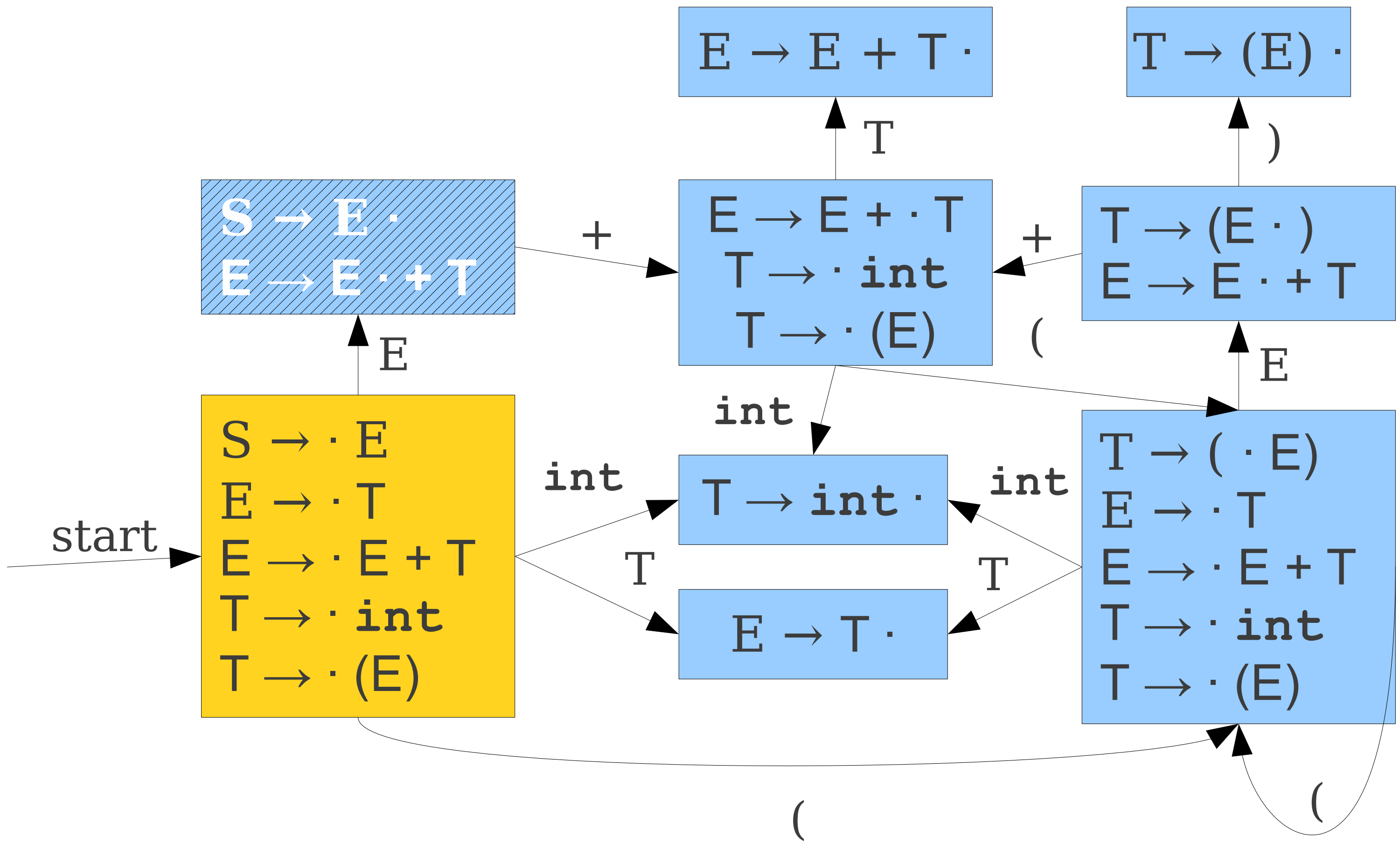
S → · E	$
E → · E + T	$
E → · T	+

+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → E
E → T
E → E + T
T → `int`
T → (E)

S → · E	$
E → · E + T	$
E → · T	+

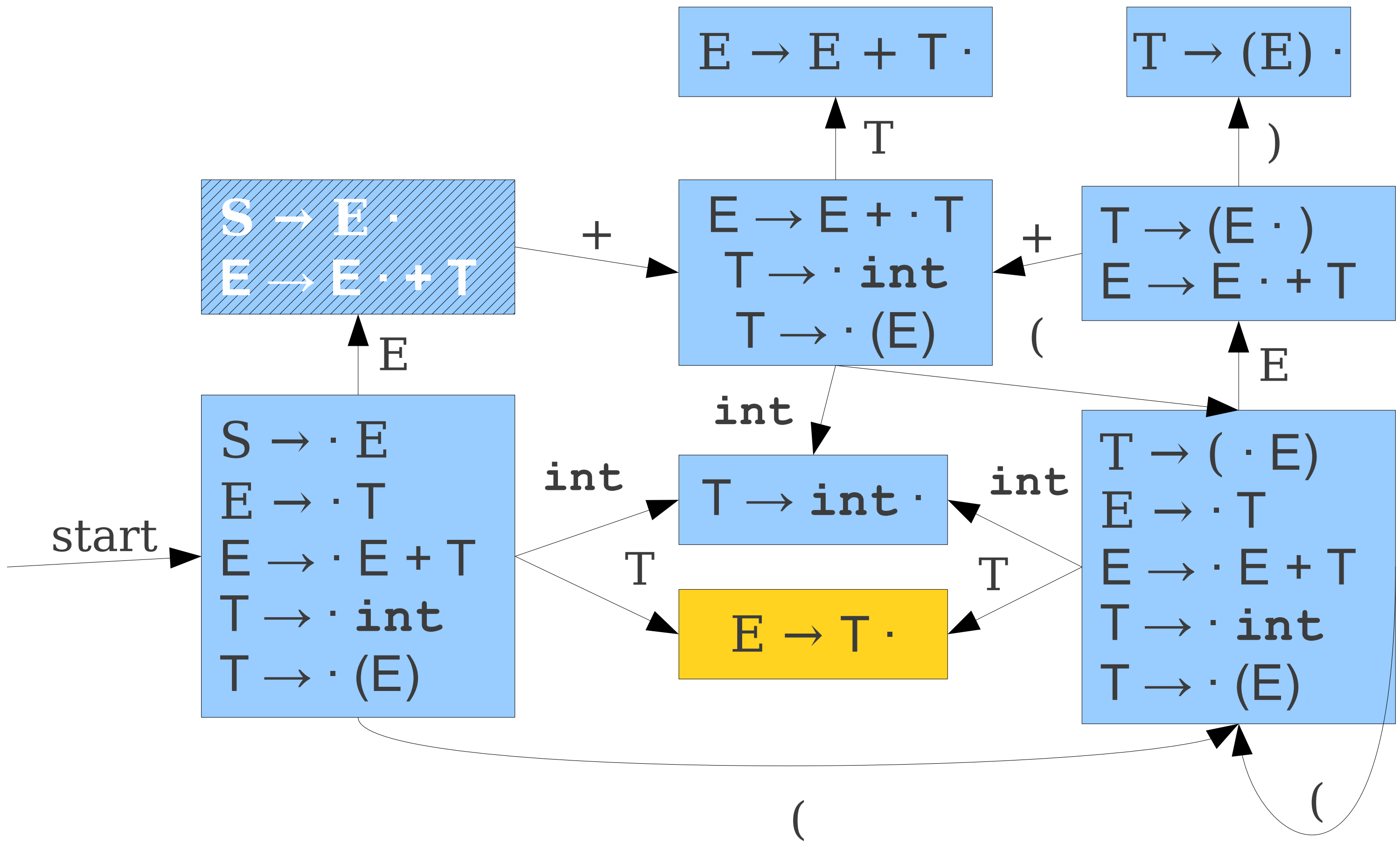


T	+	(	`int`	+	`int`	+	`int`	)	$

LR(1) Parsing: The Intuition

$S \to E$
$E \to T$
$E \to E + T$
$T \to$ **int**
$T \to (E)$

$S \to \cdot E$	\$
$E \to \cdot E + T$	\$
$E \to T \cdot$	+

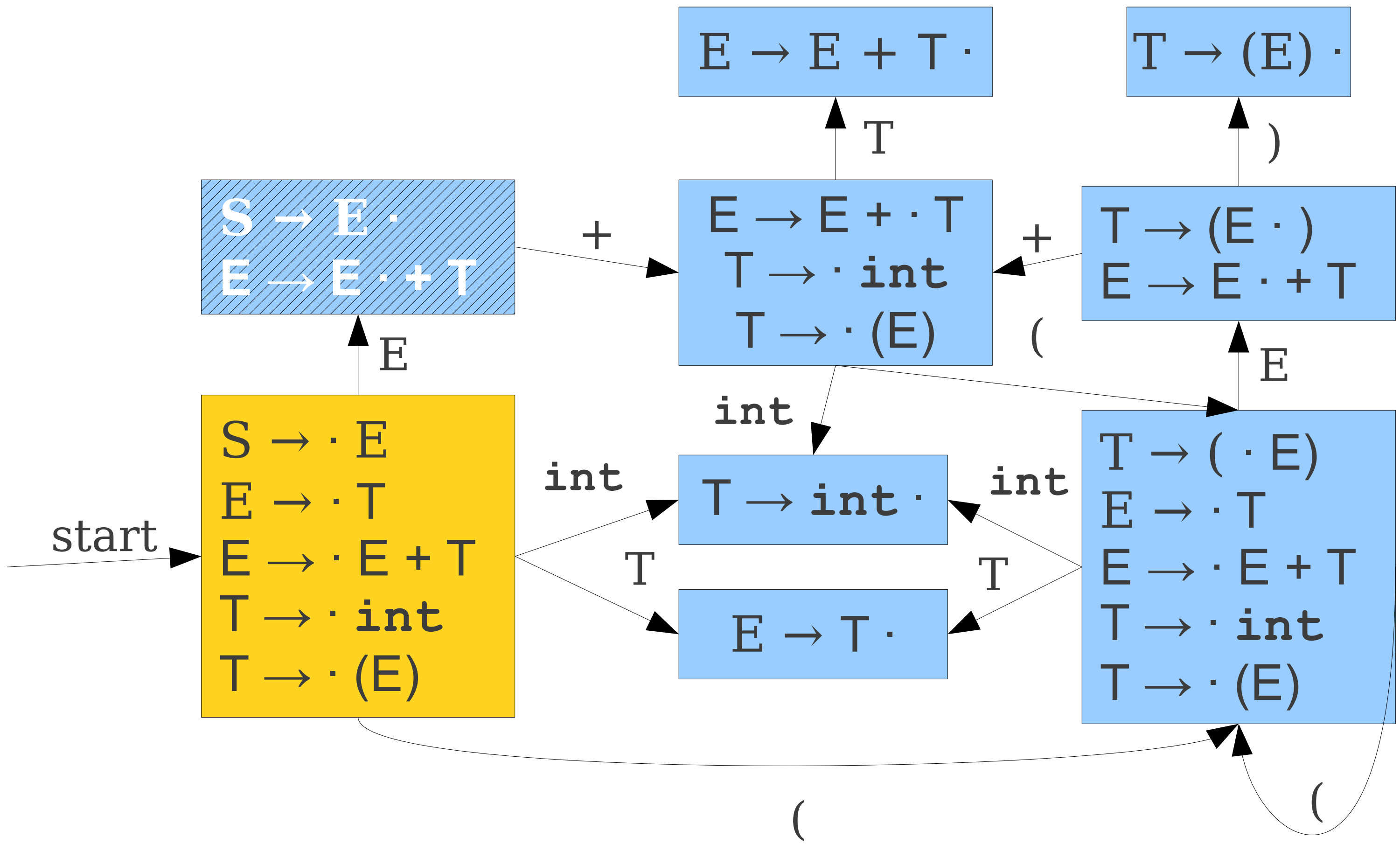


T		+	(	int	+	int	+	int	)	\$

LR(1) Parsing: The Intuition

$S \to E$
$E \to T$
$E \to E + T$
$T \to$ **int**
$T \to$ **(E)**

$S \to \cdot E$	$
$E \to \cdot E + T$	$

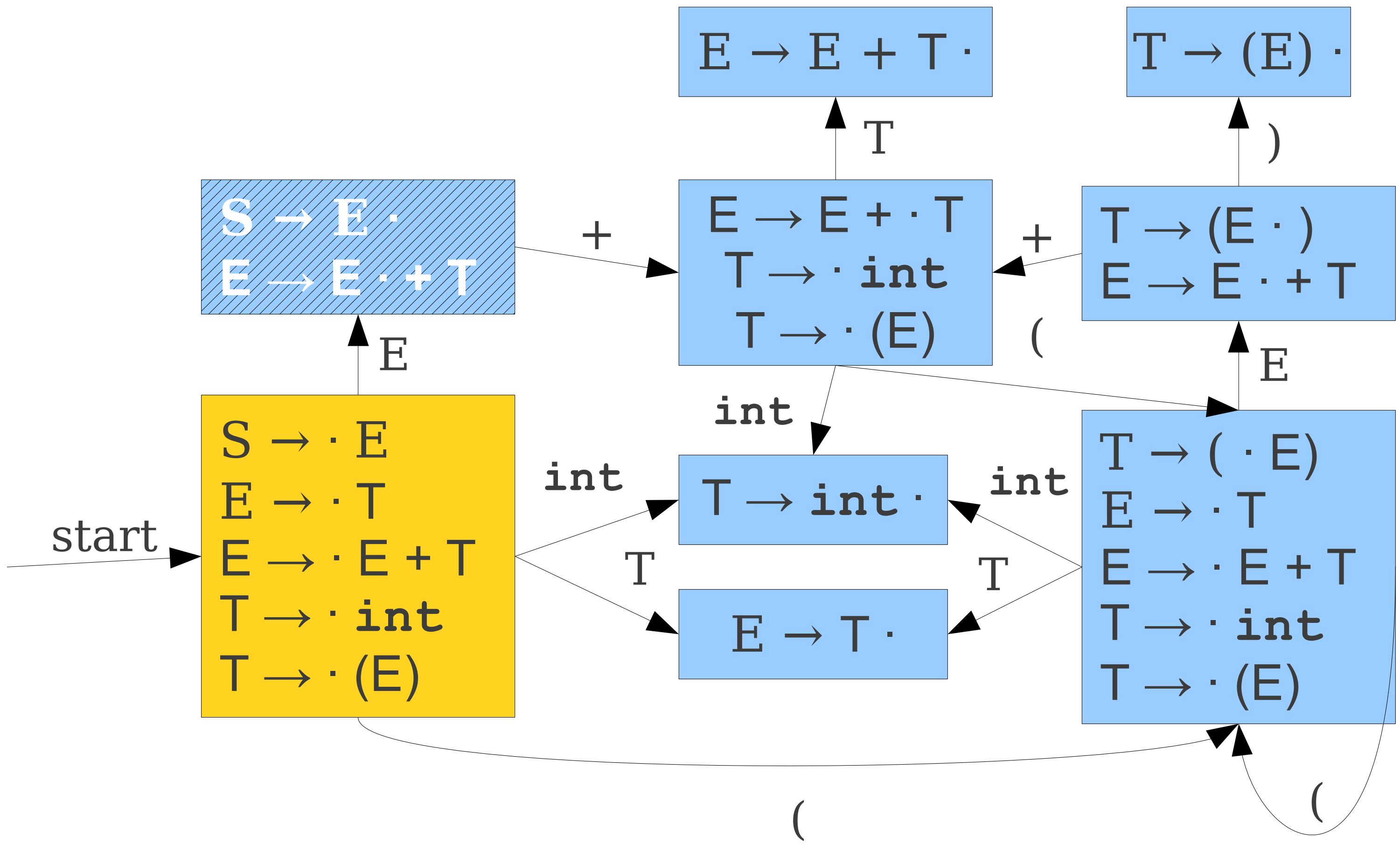


+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → **E**
E → **T**
E → **E + T**
T → **`int`**
T → **(E)**

S → · E	$
E → · E + T	$

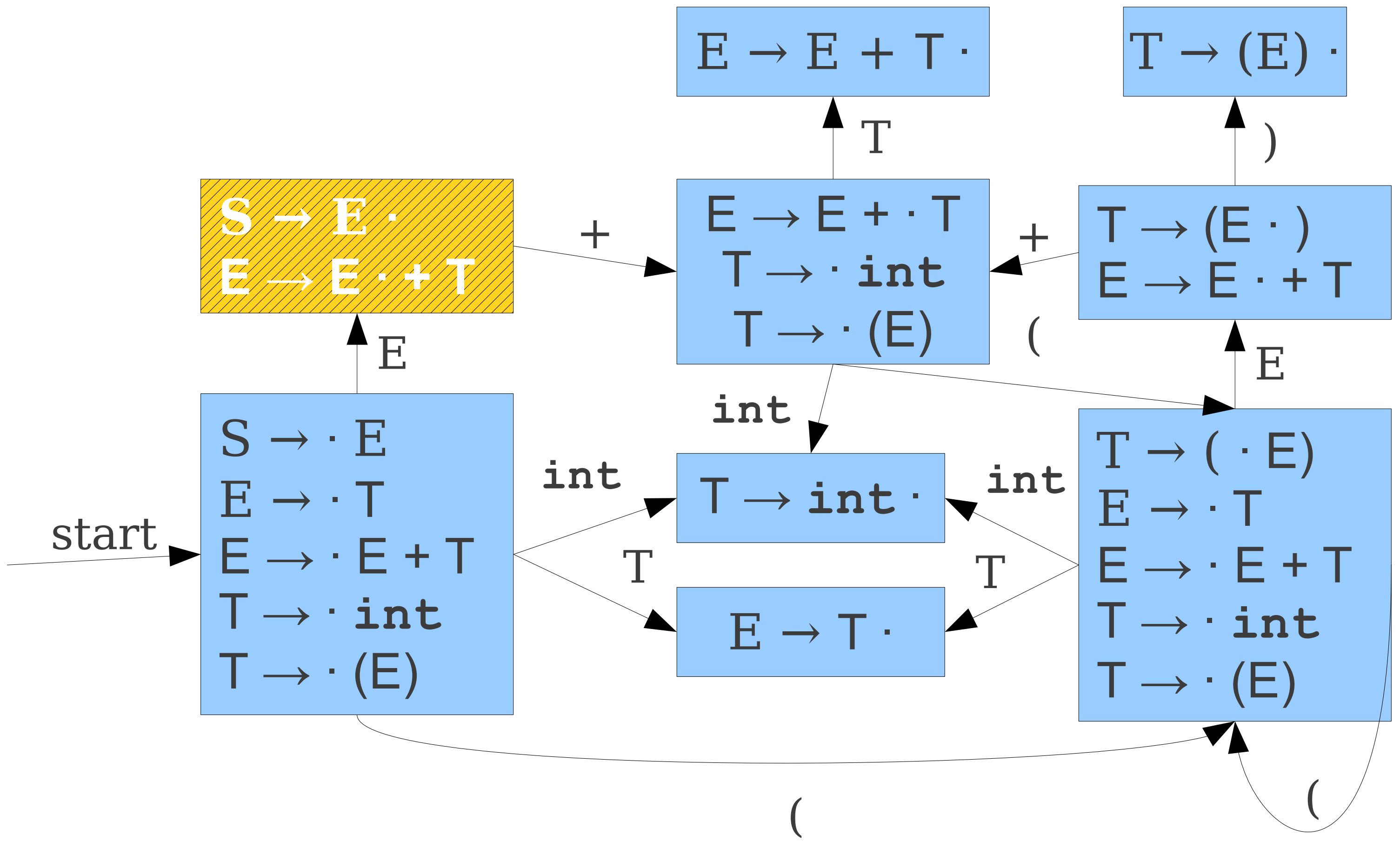


E	ǀ	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

$S \to E$
$E \to T$
$E \to E + T$
$T \to$ **int**
$T \to (E)$

S → · E	$
E → E · + T	$

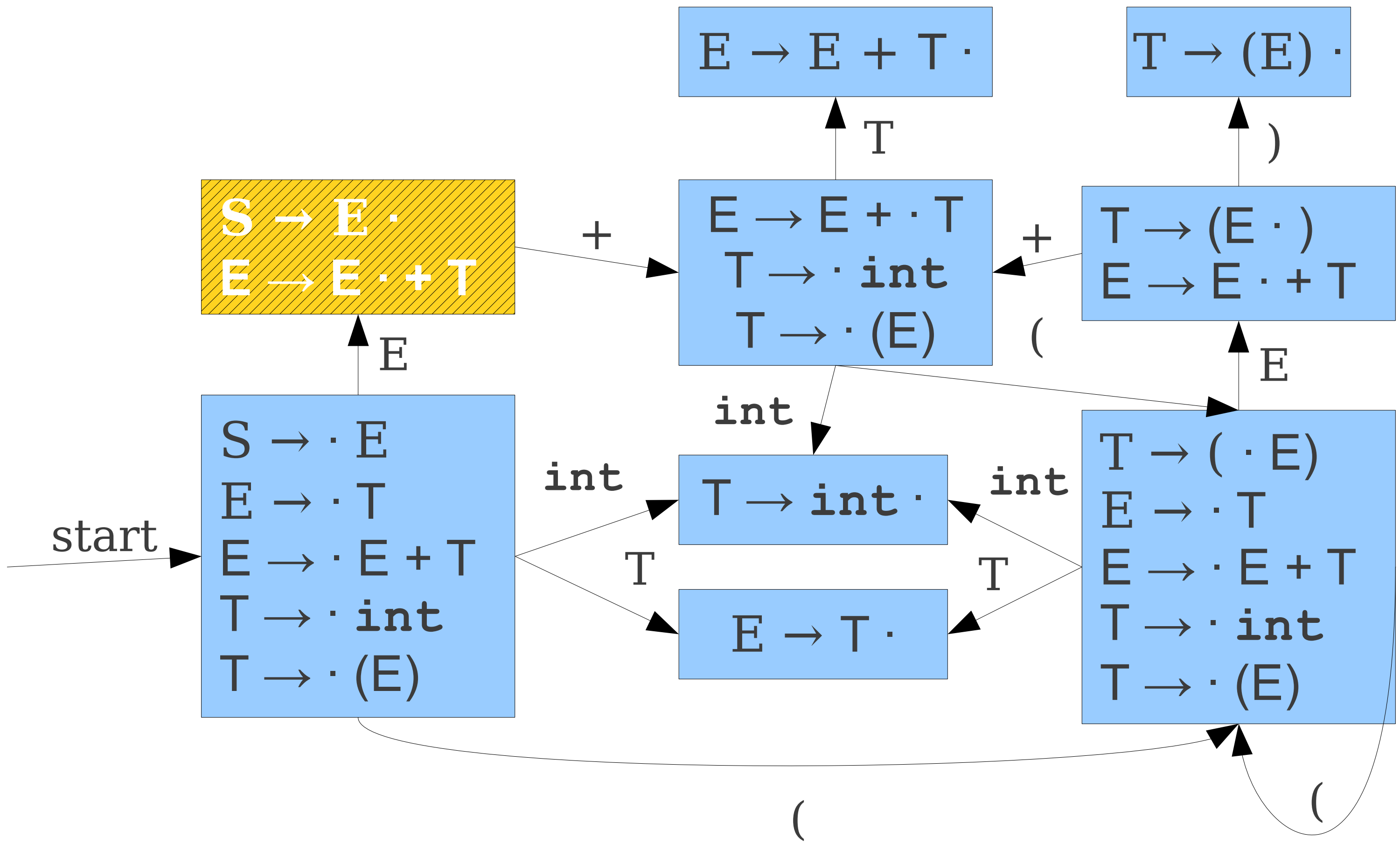


E		+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → **E**
E → **T**
E → **E** + **T**
T → `int`
T → (**E**)

S → · E	$
E → E · + T	$

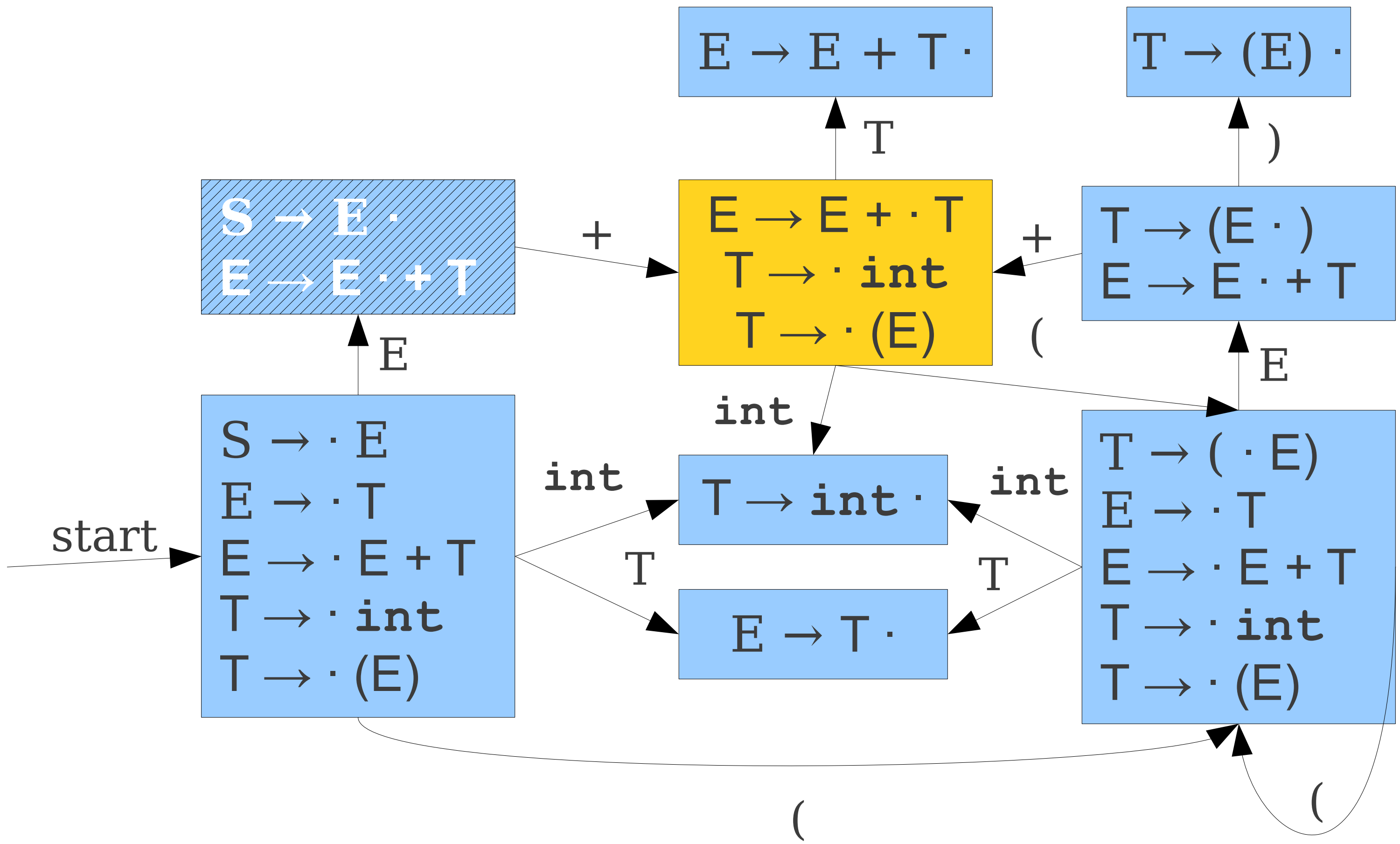


E		+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → **E**
E → **T**
E → **E + T**
T → **int**
T → **(E)**

S → · E	$
E → E · + T	$



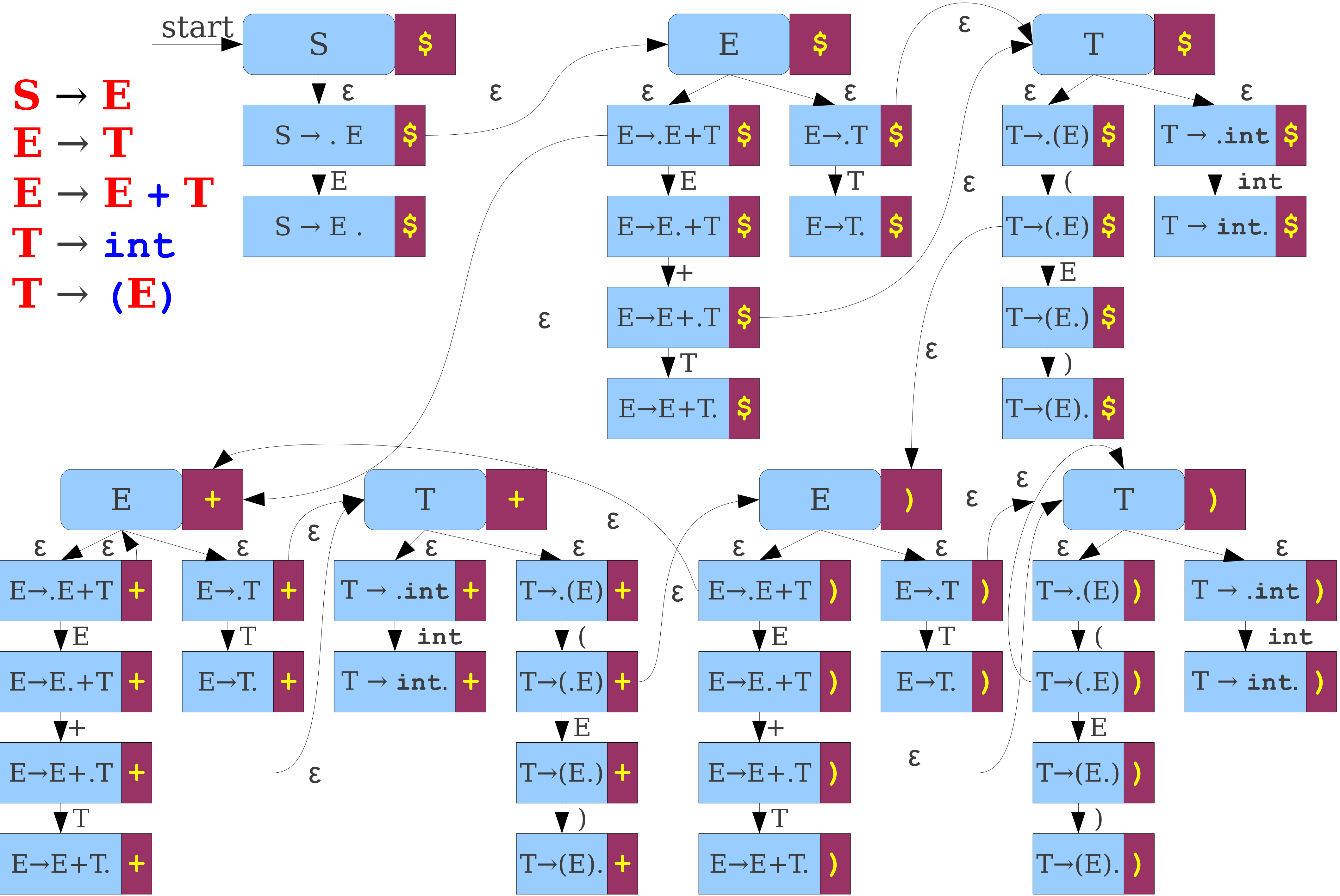
E	+	|	(	int	+	int	+	int	)	$

The Intuition behind LR(1)

- Guess which series of productions we are reversing.
- Use this information to maintain information about what lookahead to expect.
- When deciding whether to shift or reduce, use lookahead to disambiguate.

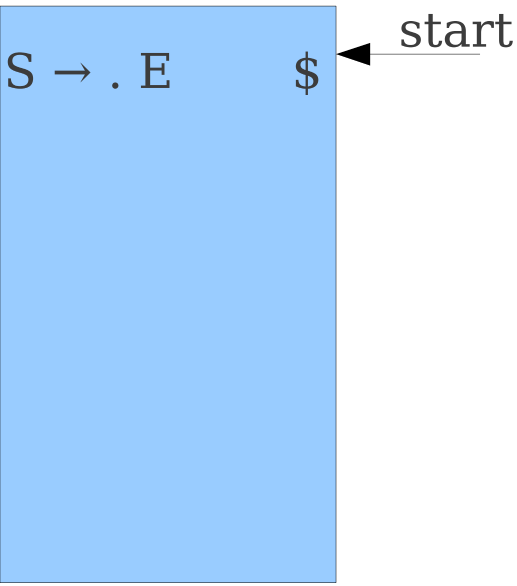
LR(1) Automata

$S \rightarrow E$
$E \rightarrow T$
$E \rightarrow E + T$
$T \rightarrow$ `int`
$T \rightarrow (E)$



Deterministic LR(1) Automata

Deterministic LR(1) Automata



Deterministic LR(1) Automata

start

$S \rightarrow .\ E \qquad \$$
$E \rightarrow .\ T \qquad \$$
$E \rightarrow .\ E + T \quad \$$

Deterministic LR(1) Automata

start

S → . E	$
E → . T	$
E → . E + T	$
E → . T	+
E → . E + T	+

Deterministic LR(1) Automata

start

$S \to . E$	$\$$
$E \to . T$	$\$$
$E \to . E + T$	$\$$
$E \to . T$	$+$
$E \to . E + T$	$+$
$T \to . int$	$\$$
$T \to . (E)$	$\$$

Deterministic LR(1) Automata

start

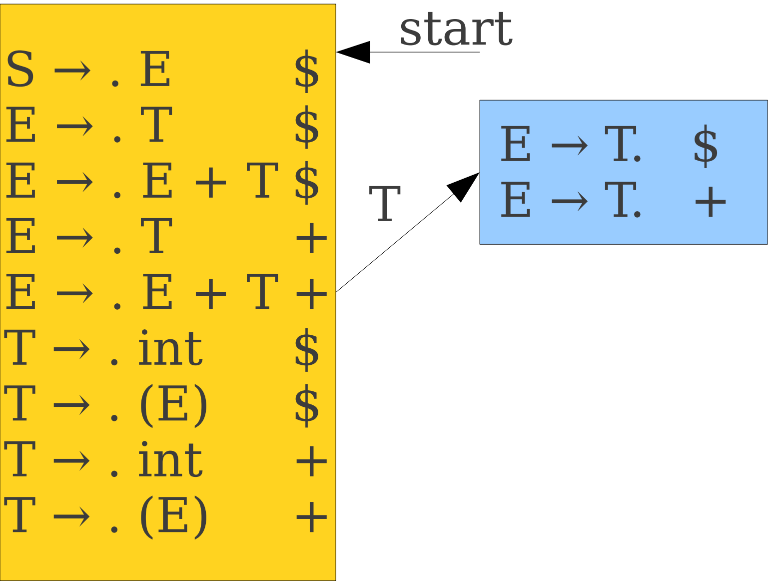
S → . E	$
E → . T	$
E → . E + T	$
E → . T	+
E → . E + T	+
T → . int	$
T → . (E)	$
T → . int	+
T → . (E)	+

Deterministic LR(1) Automata

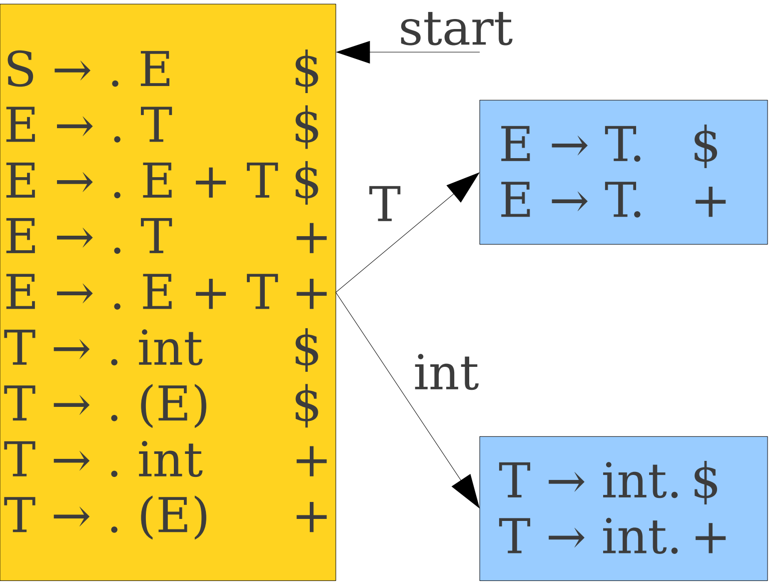
start

S → . E	$
E → . T	$
E → . E + T	$
E → . T	+
E → . E + T	+
T → . int	$
T → . (E)	$
T → . int	+
T → . (E)	+

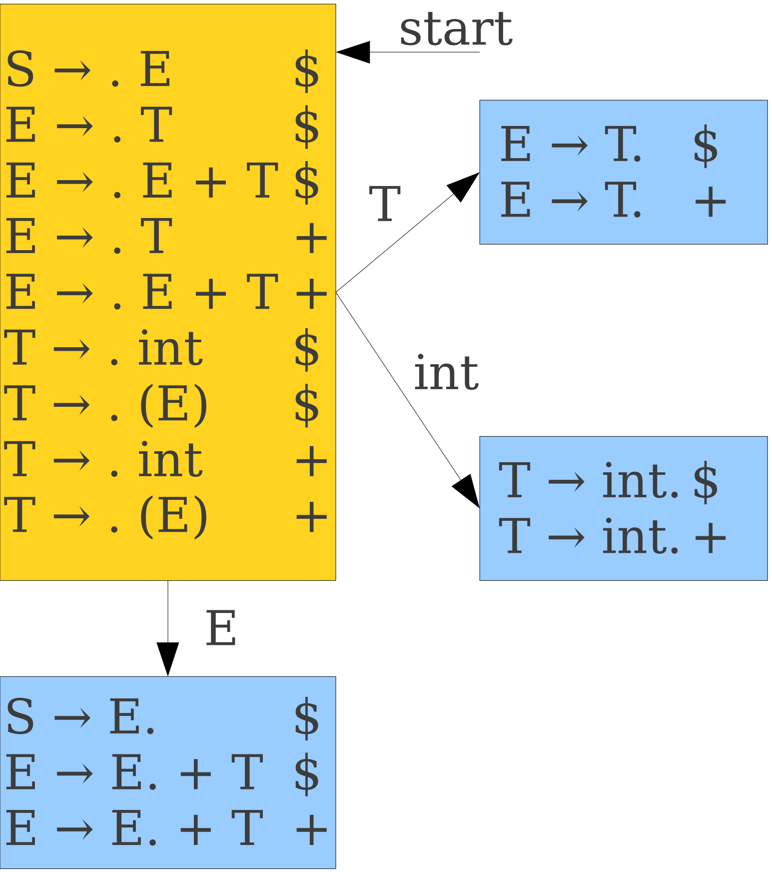
Deterministic LR(1) Automata



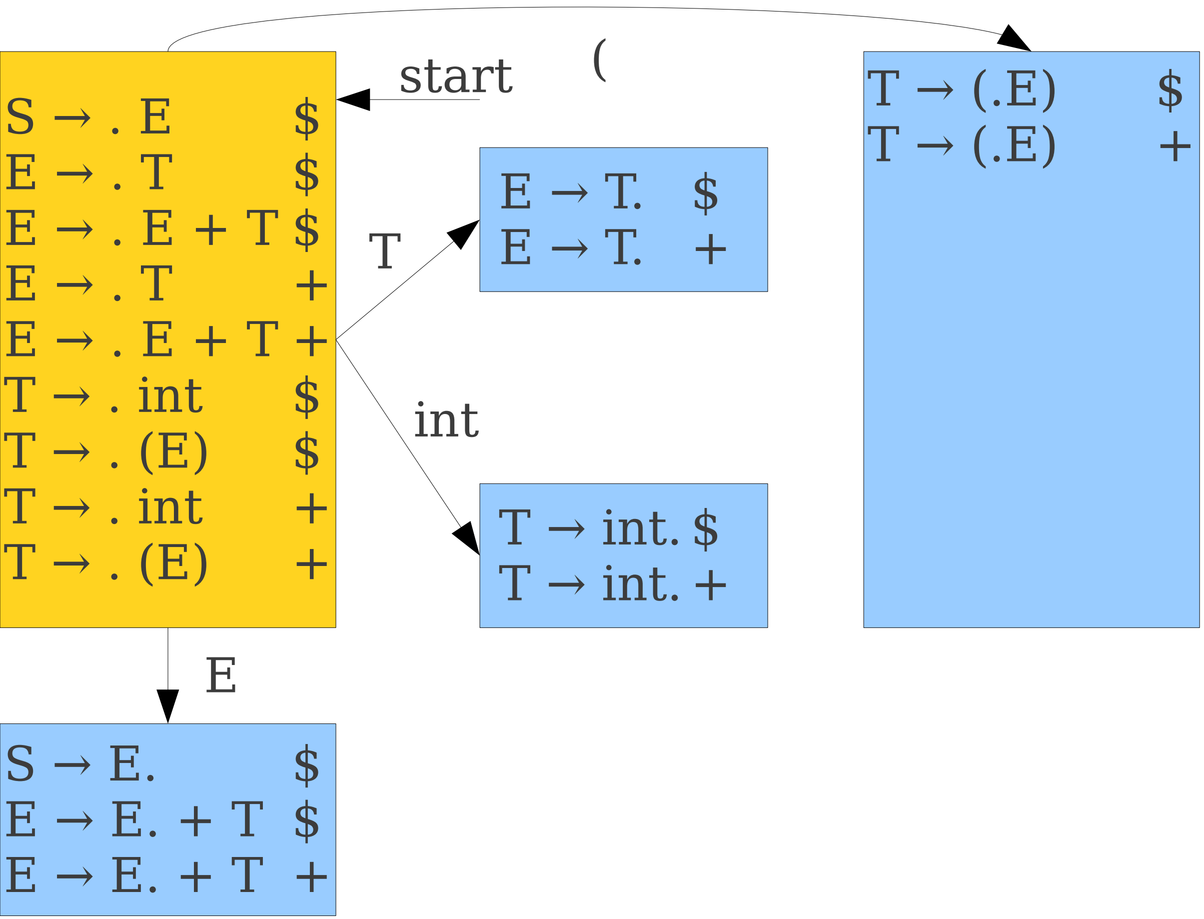
Deterministic LR(1) Automata



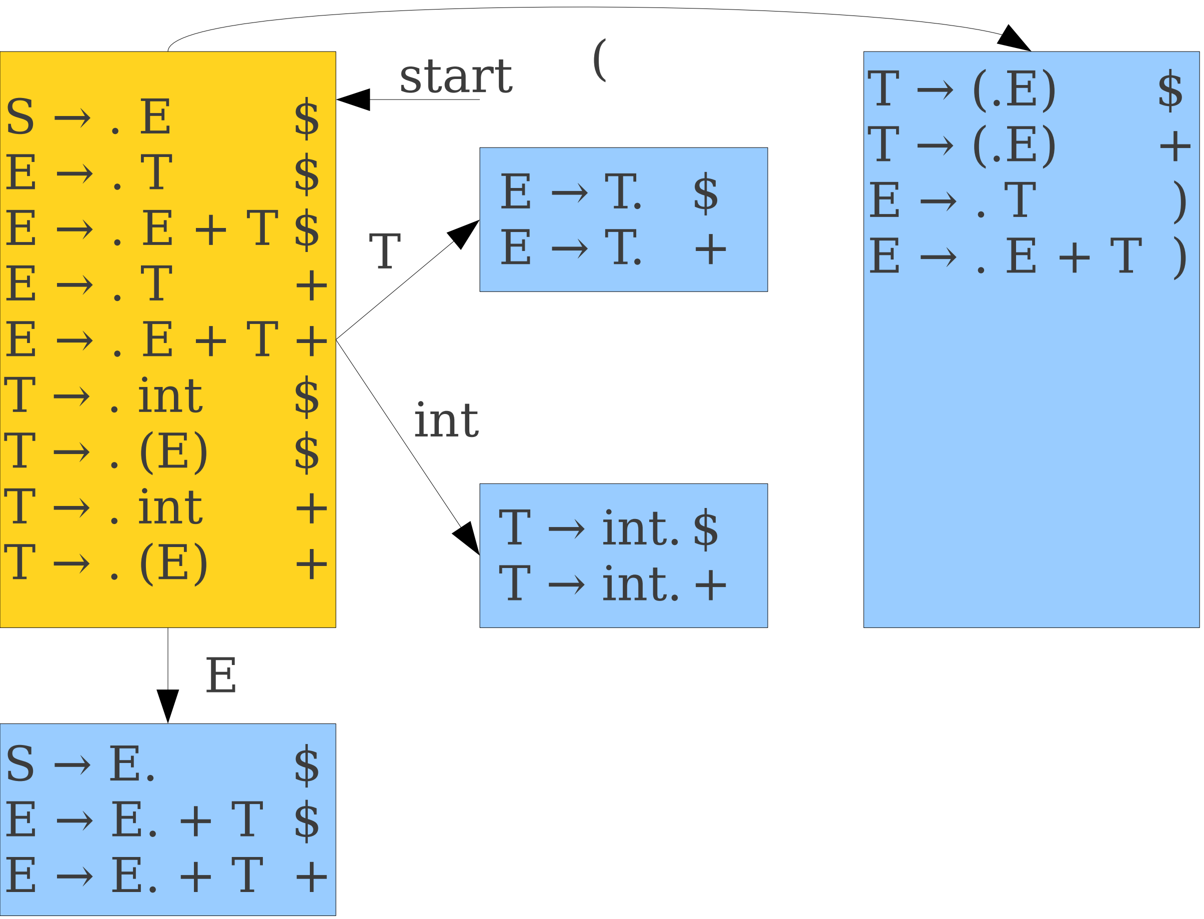
Deterministic LR(1) Automata



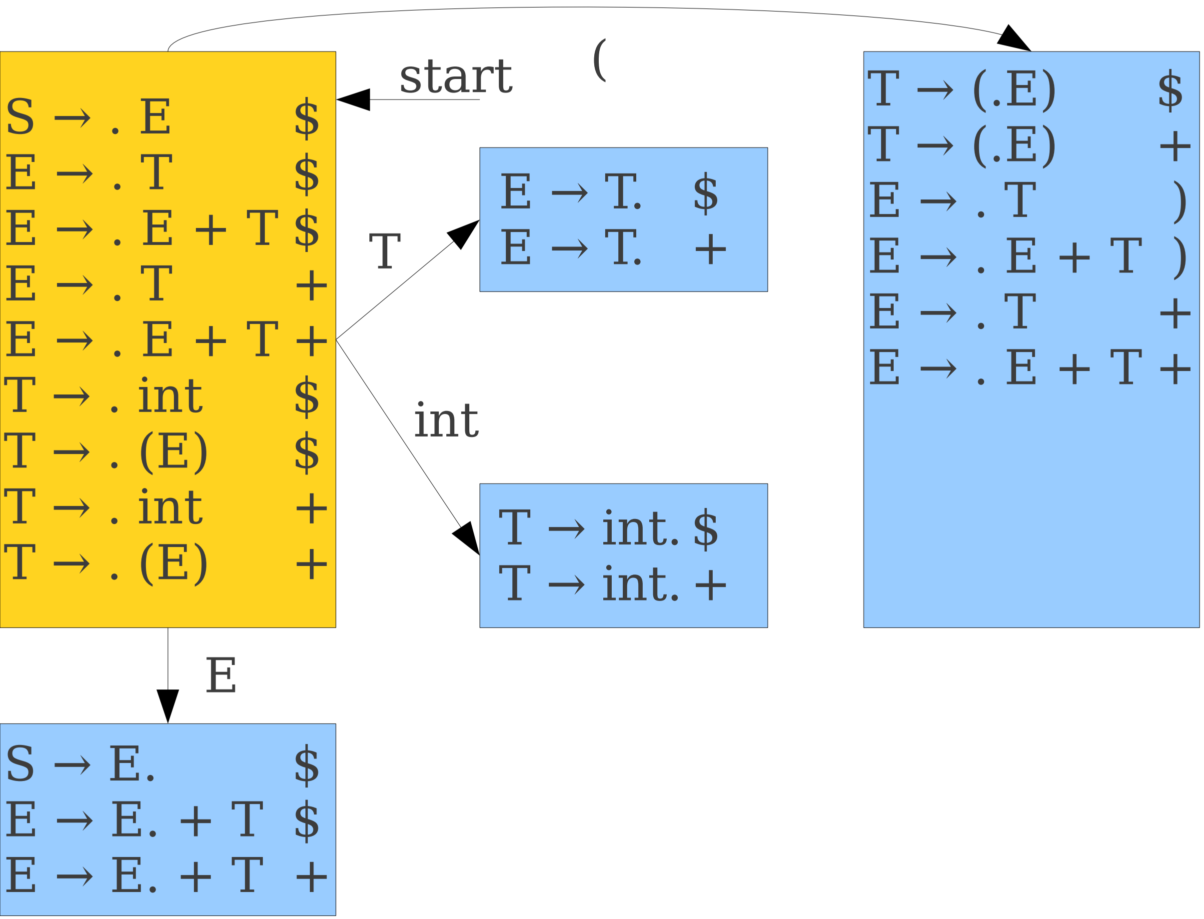
Deterministic LR(1) Automata



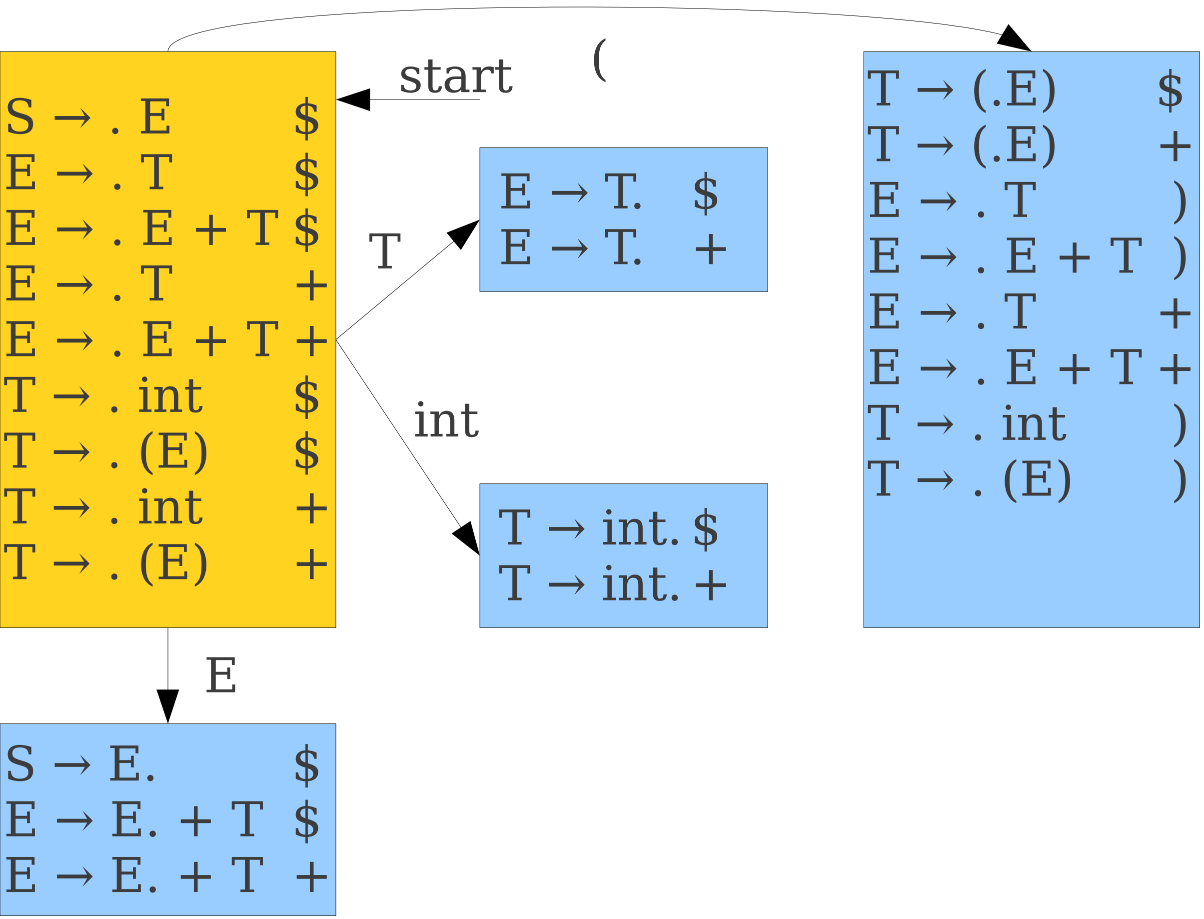
Deterministic LR(1) Automata



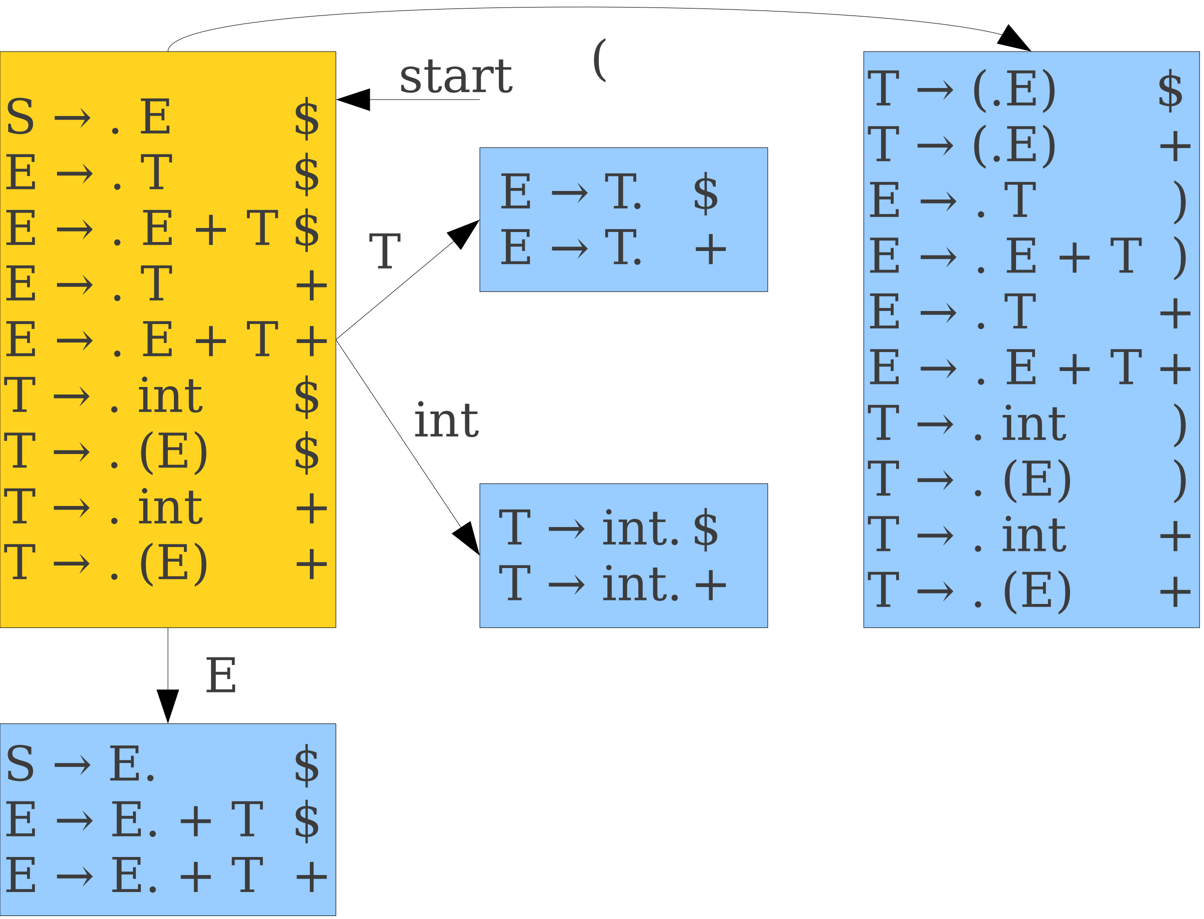
Deterministic LR(1) Automata



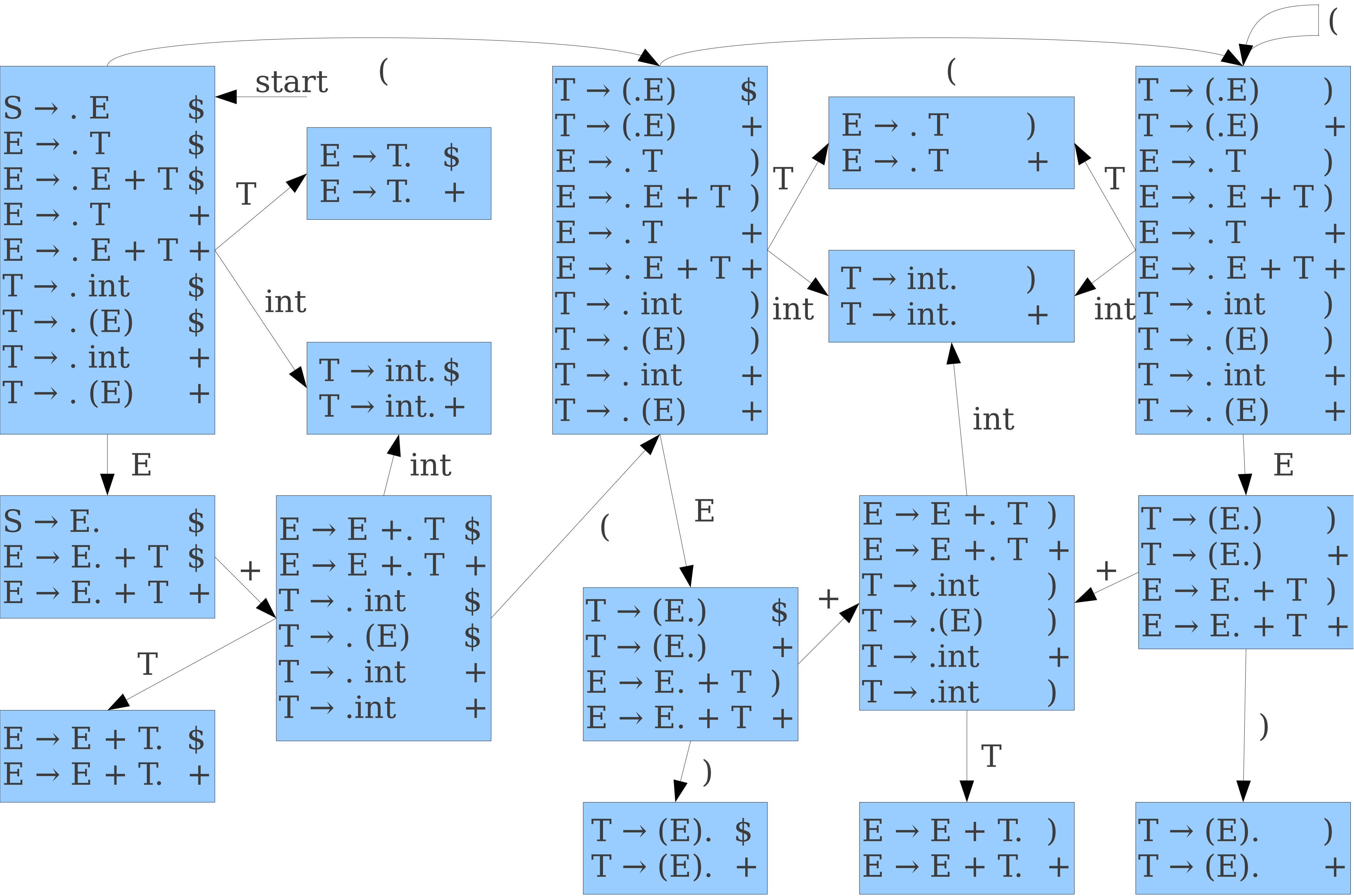
Deterministic LR(1) Automata



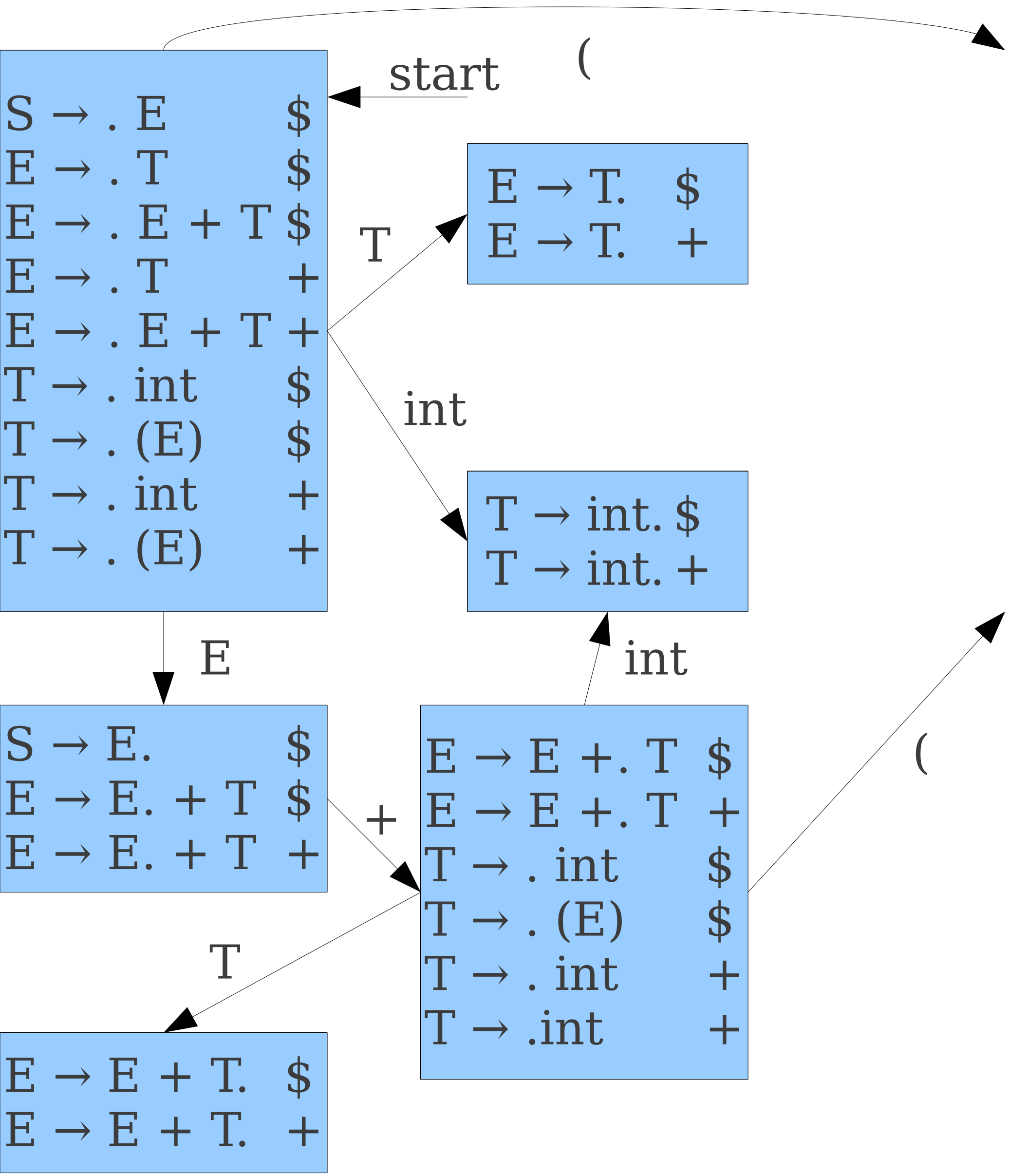
Deterministic LR(1) Automata



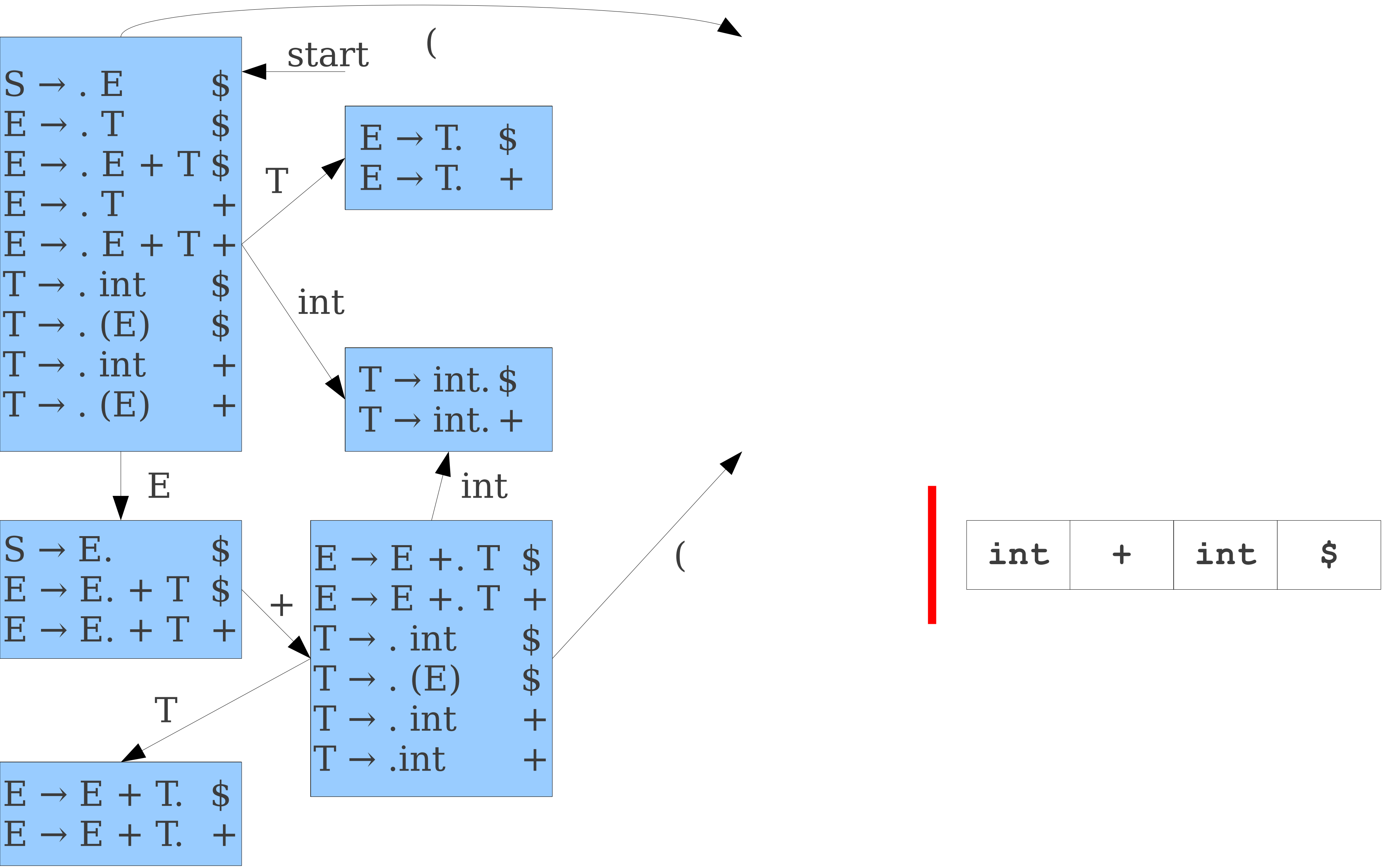
Deterministic LR(1) Automata



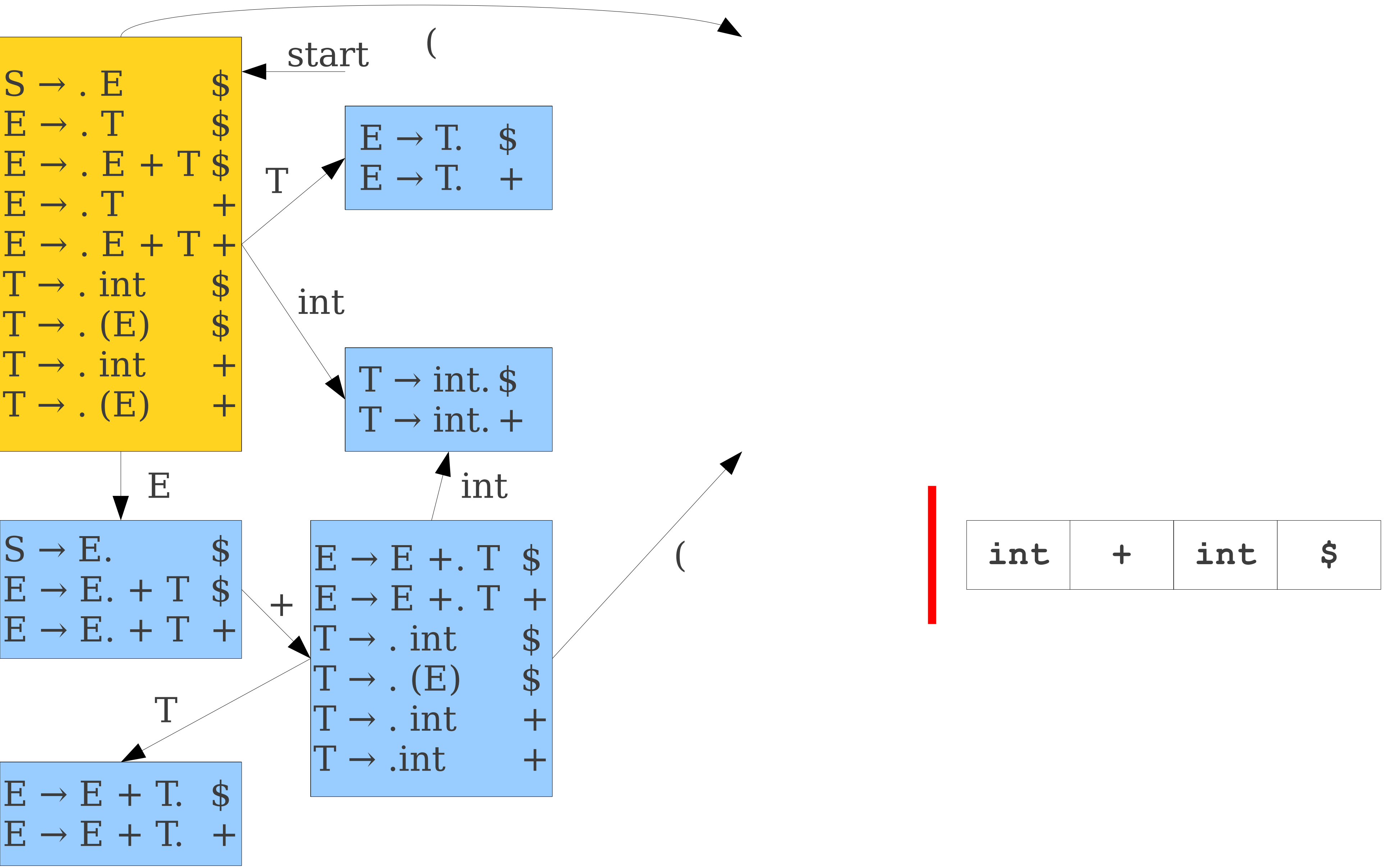
Deterministic LR(1) Automata



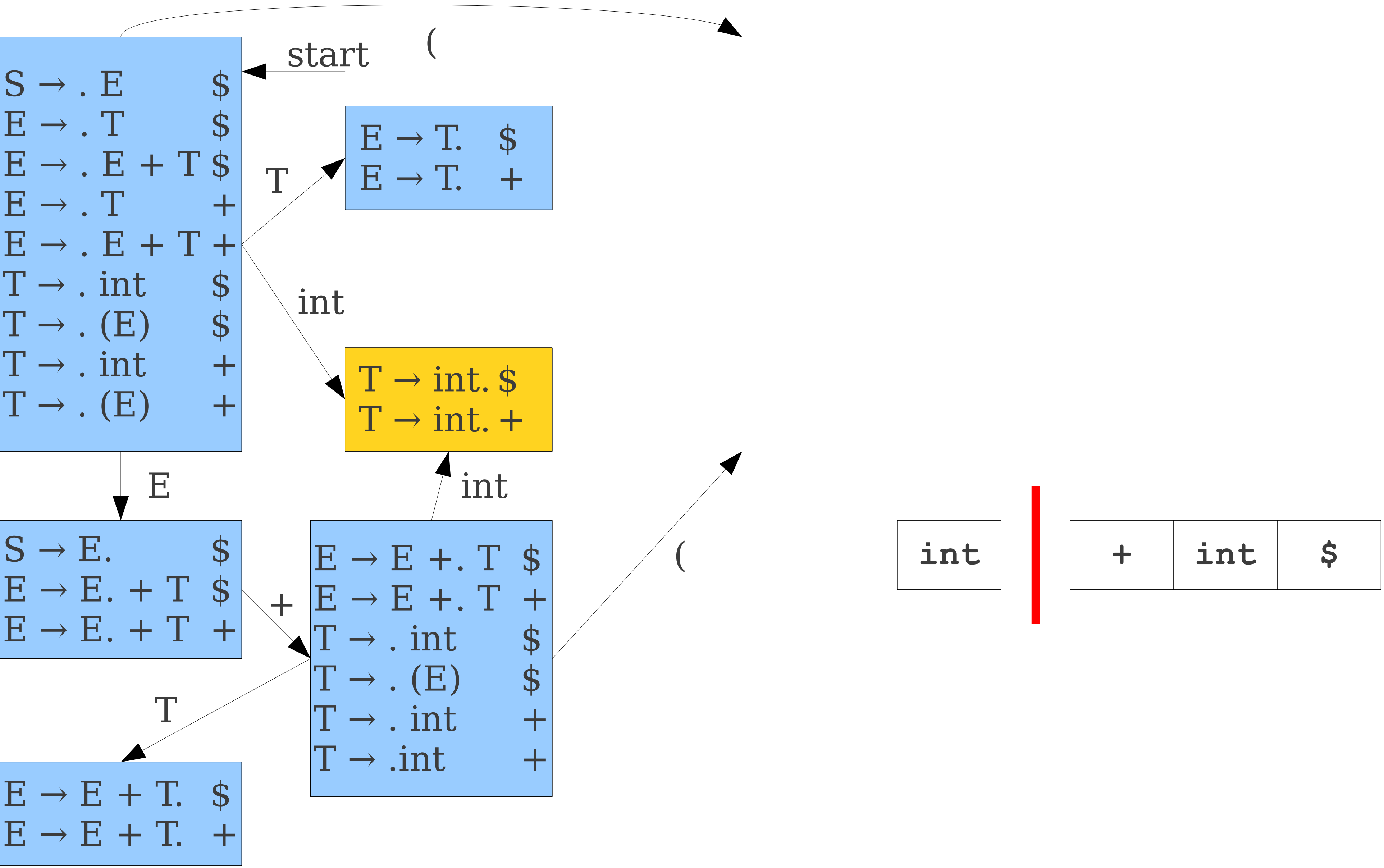
Deterministic LR(1) Automata



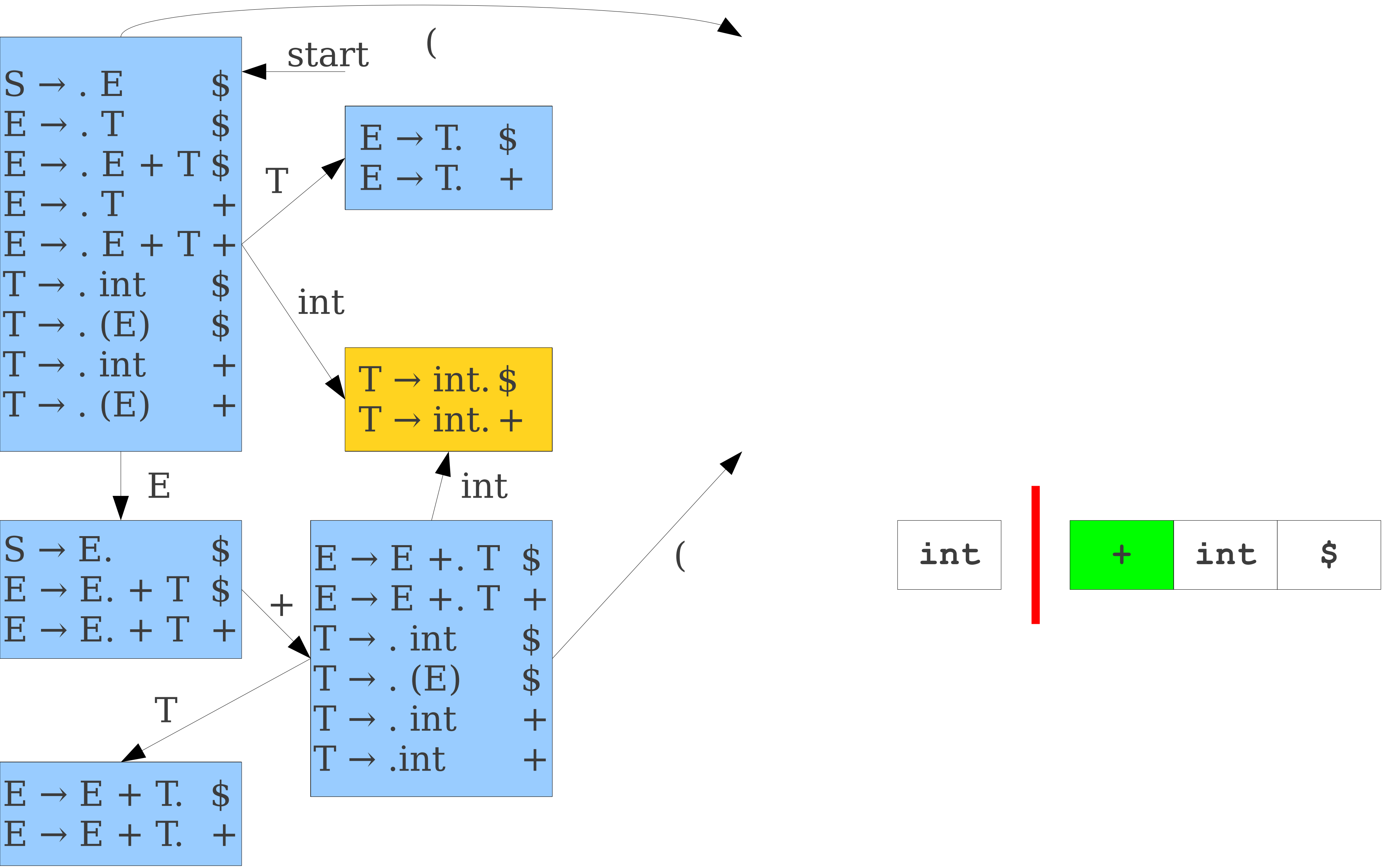
Deterministic LR(1) Automata



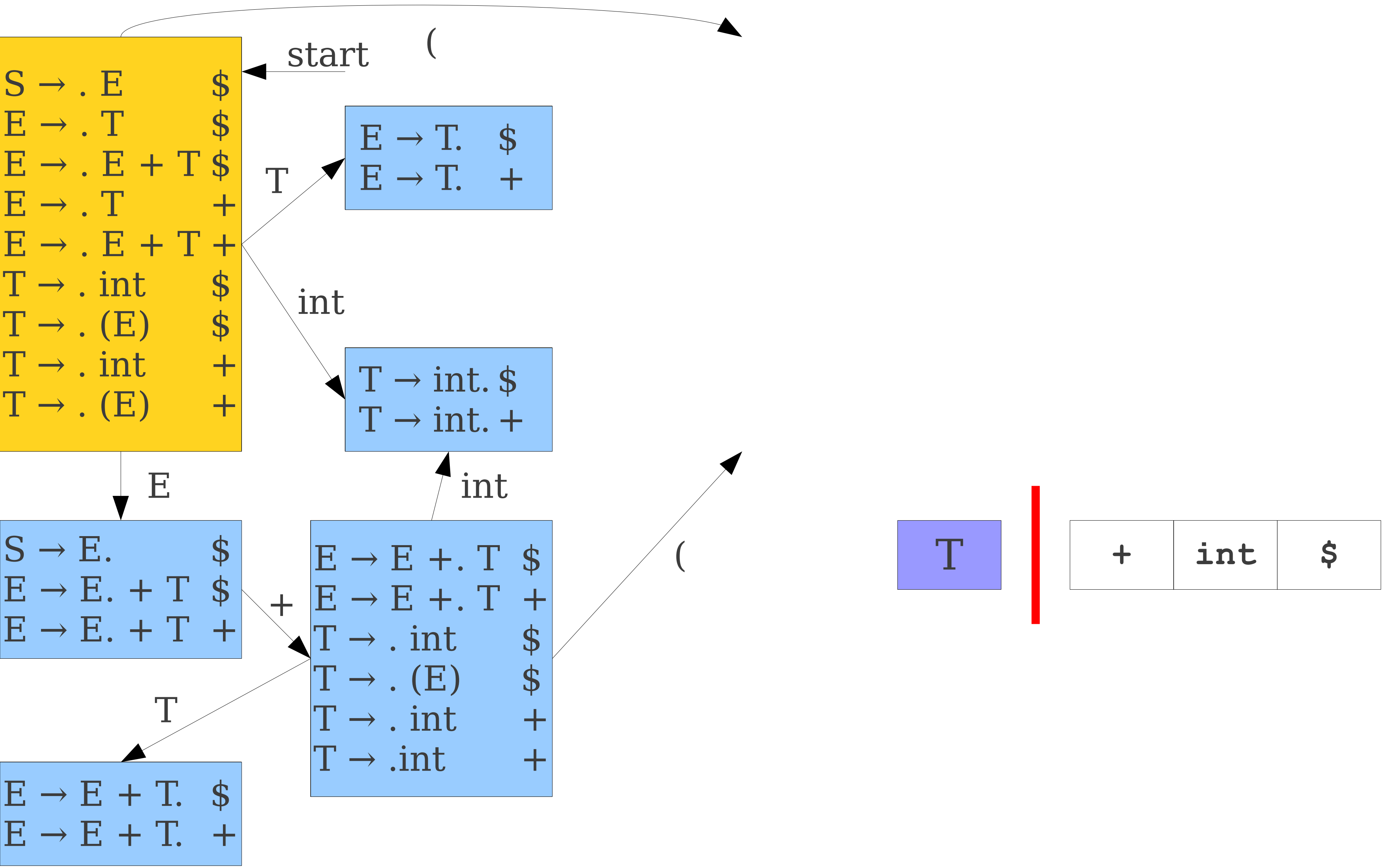
Deterministic LR(1) Automata



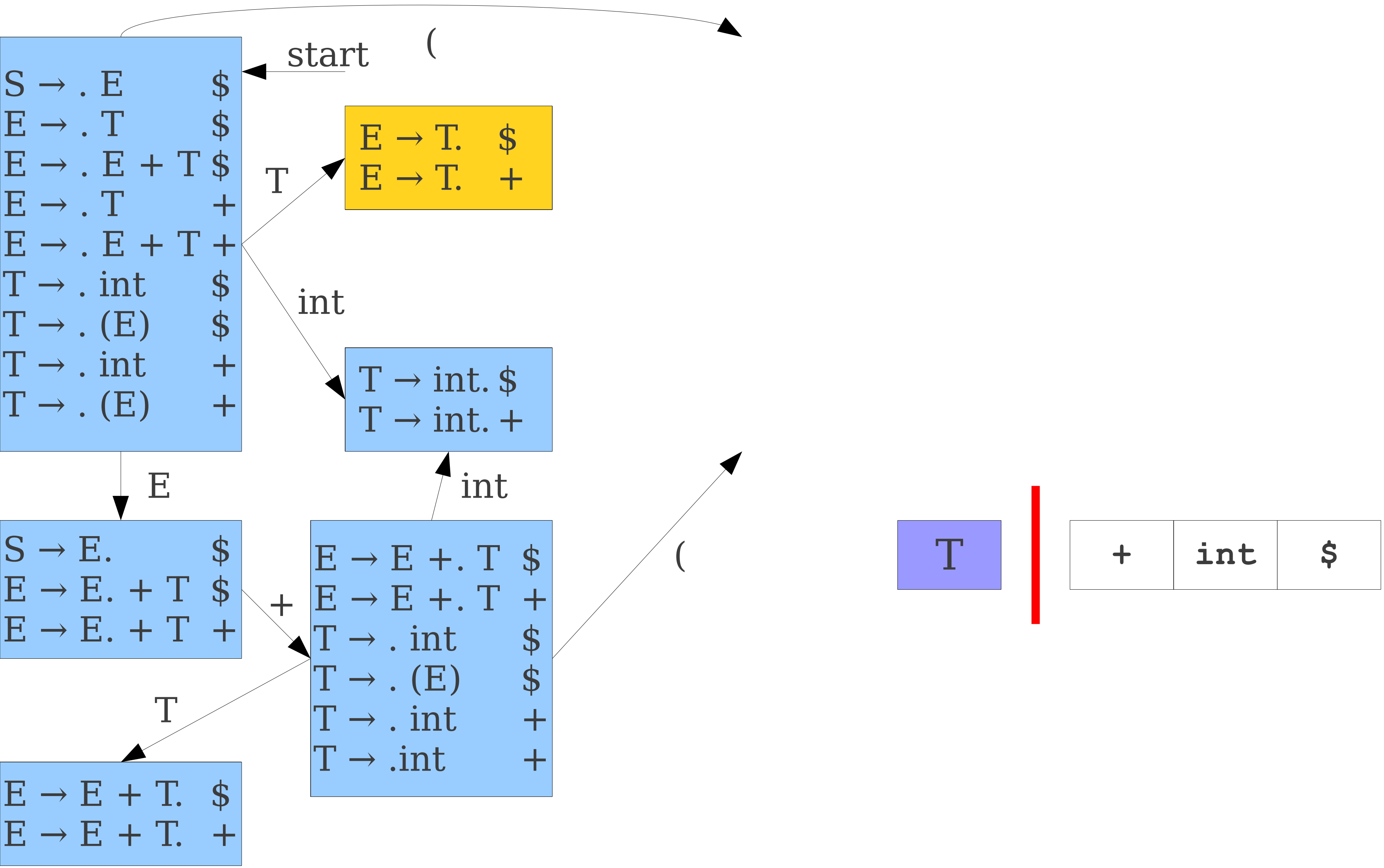
Deterministic LR(1) Automata



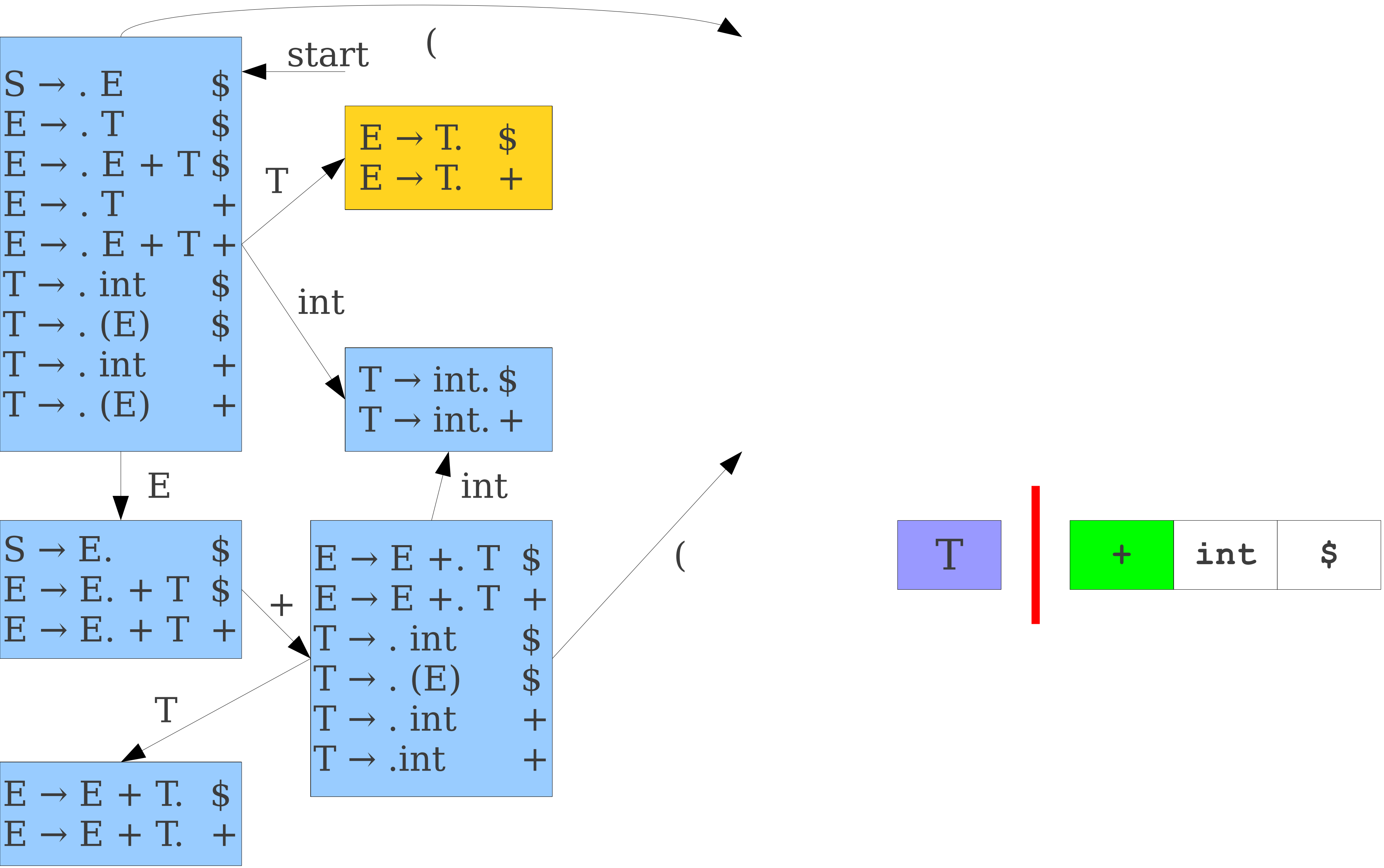
Deterministic LR(1) Automata



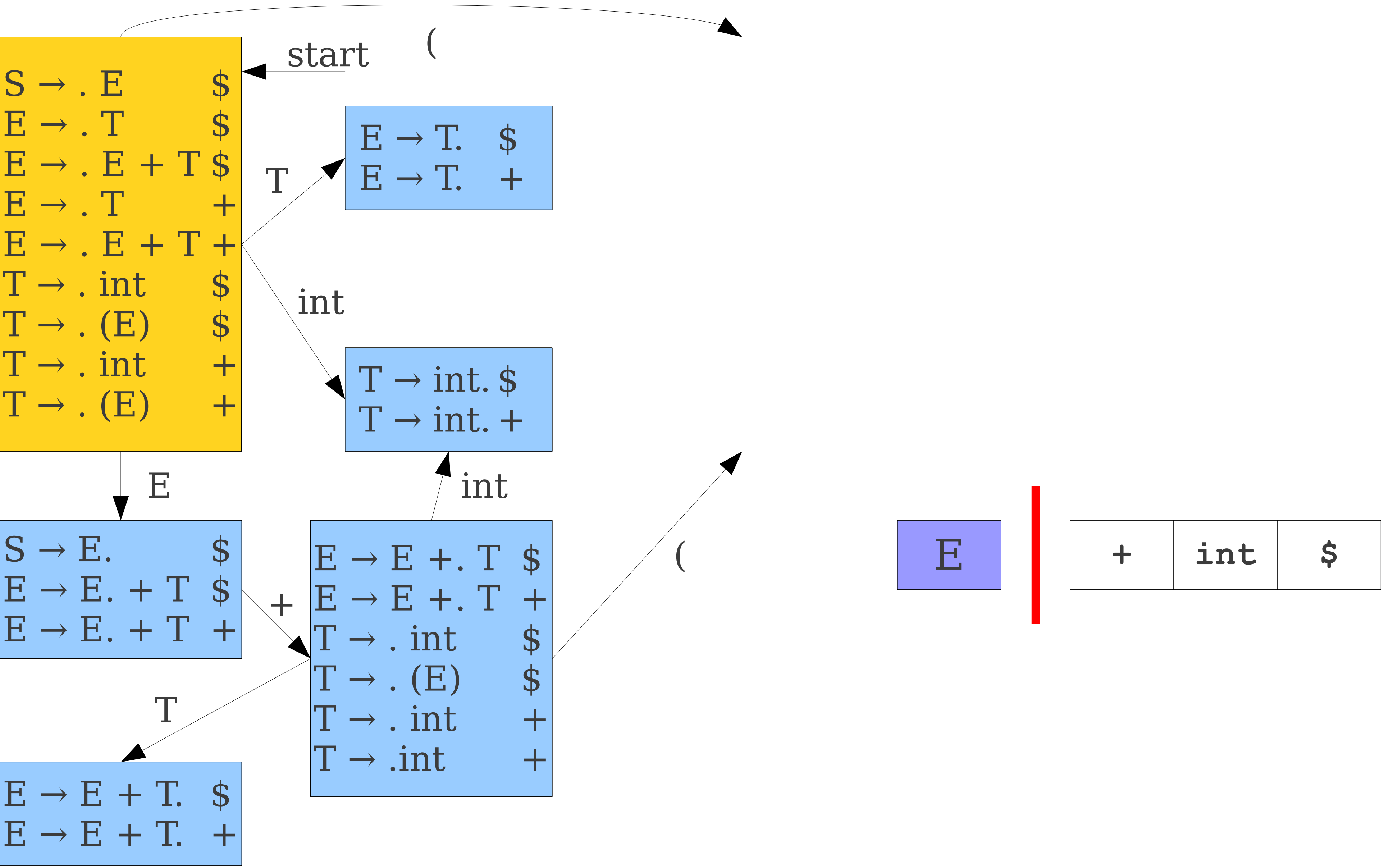
Deterministic LR(1) Automata



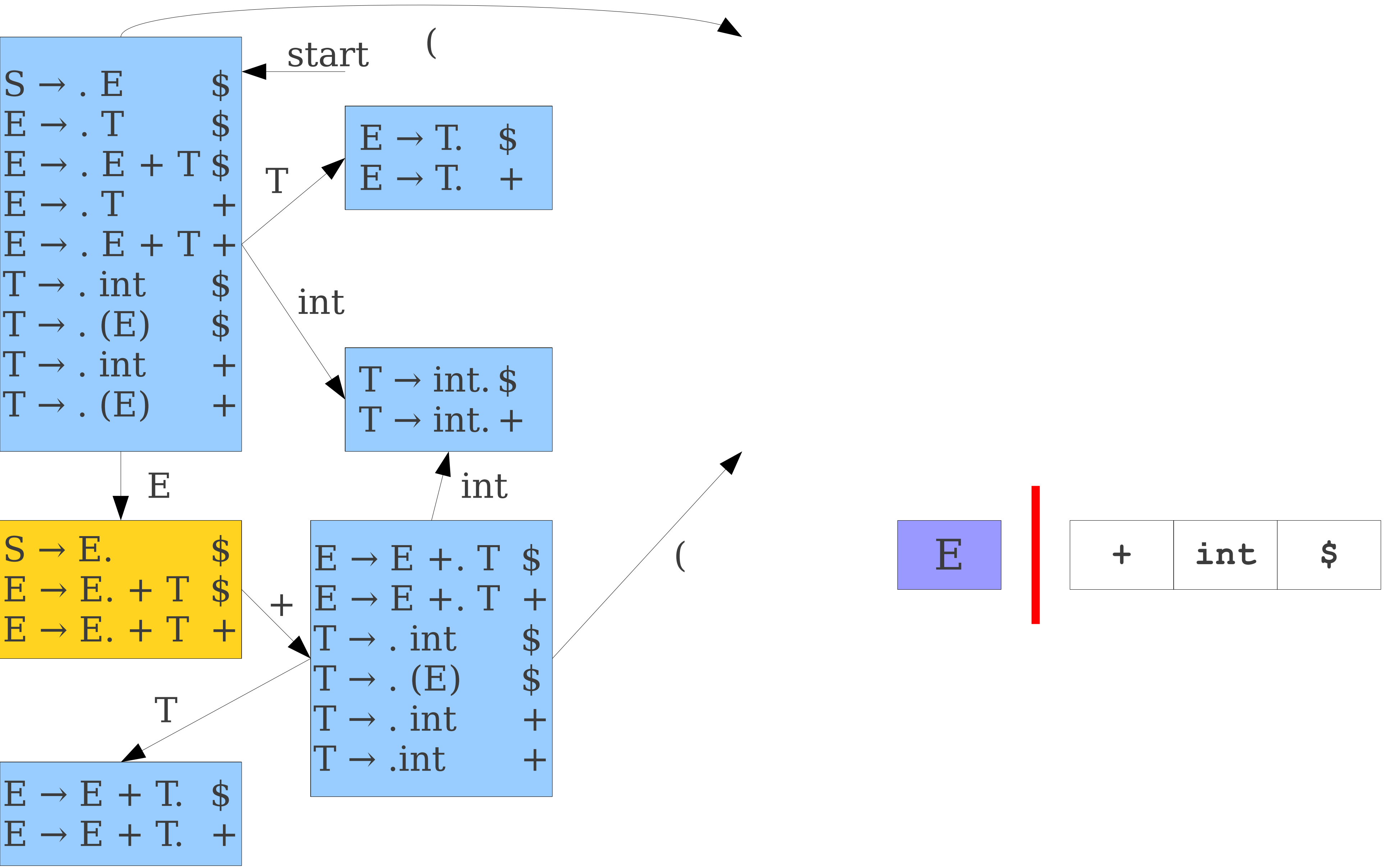
Deterministic LR(1) Automata



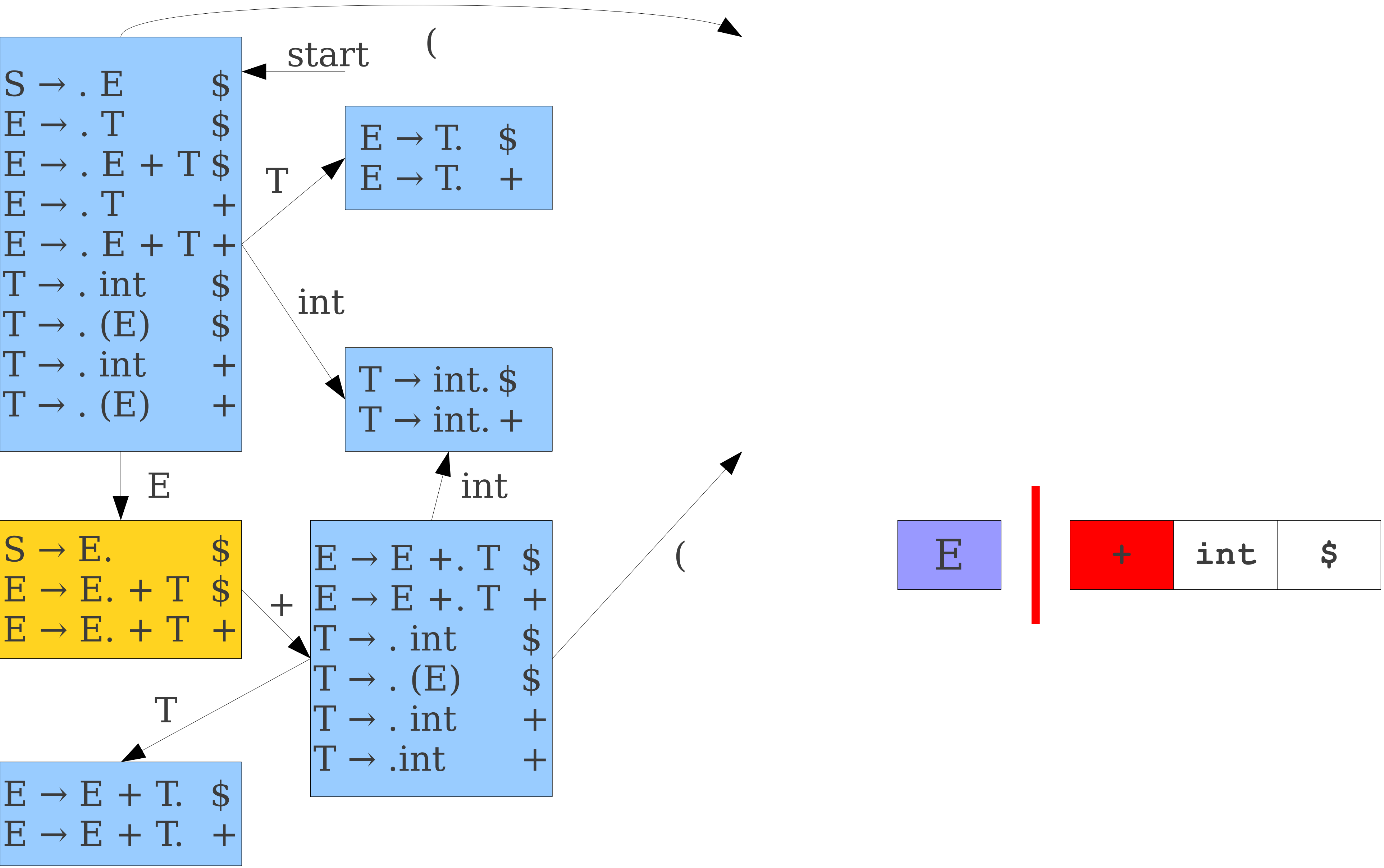
Deterministic LR(1) Automata



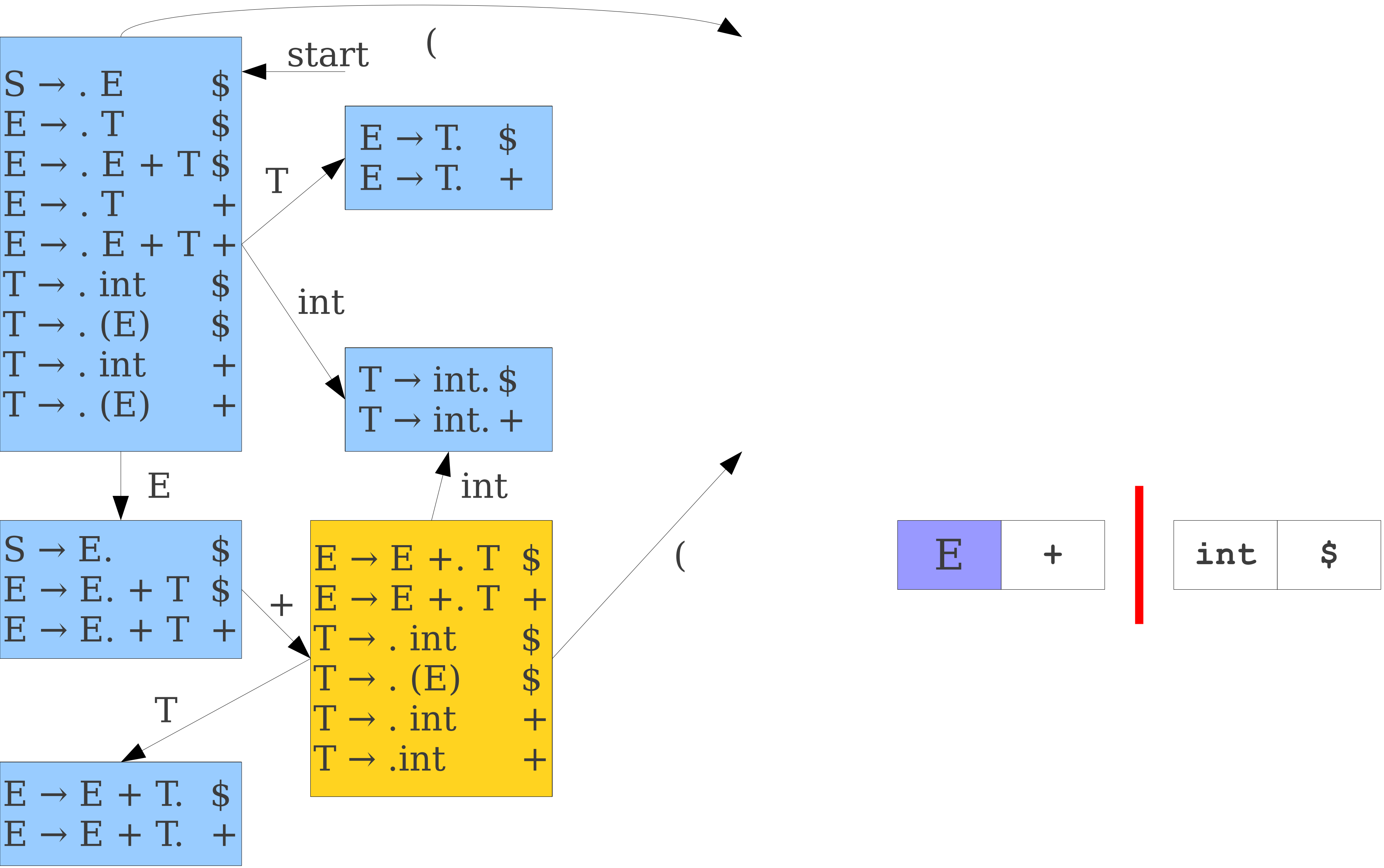
Deterministic LR(1) Automata



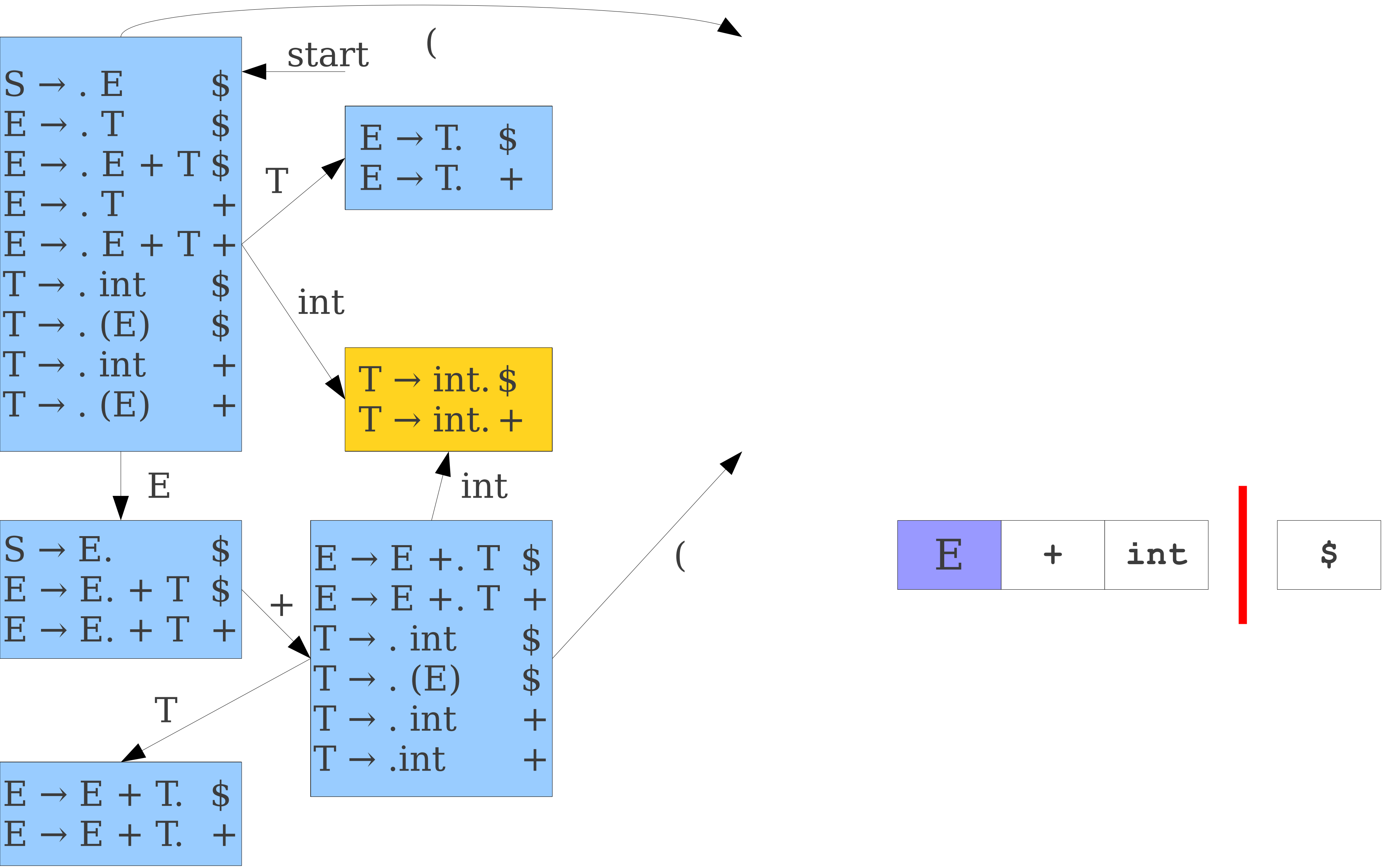
Deterministic LR(1) Automata



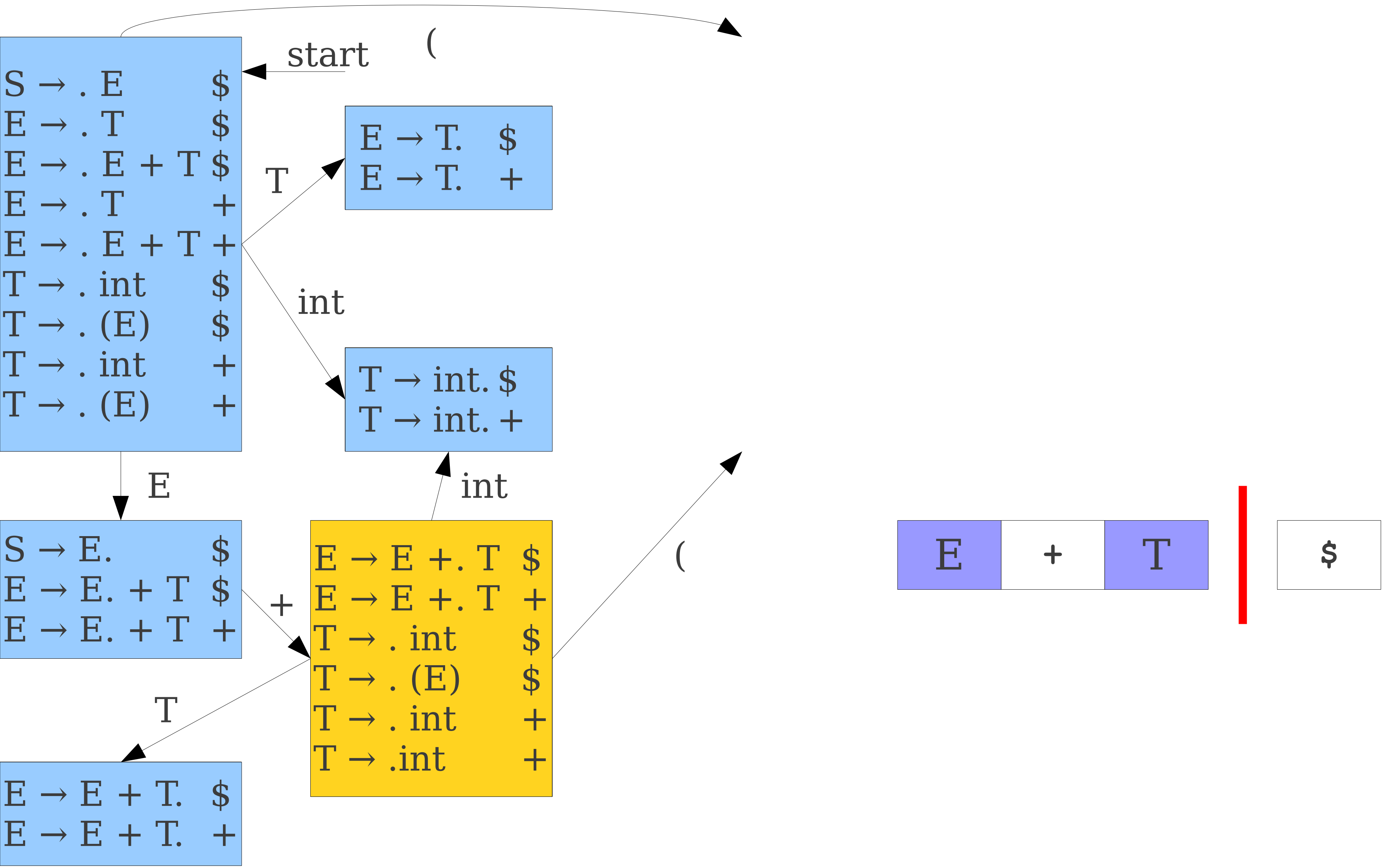
Deterministic LR(1) Automata



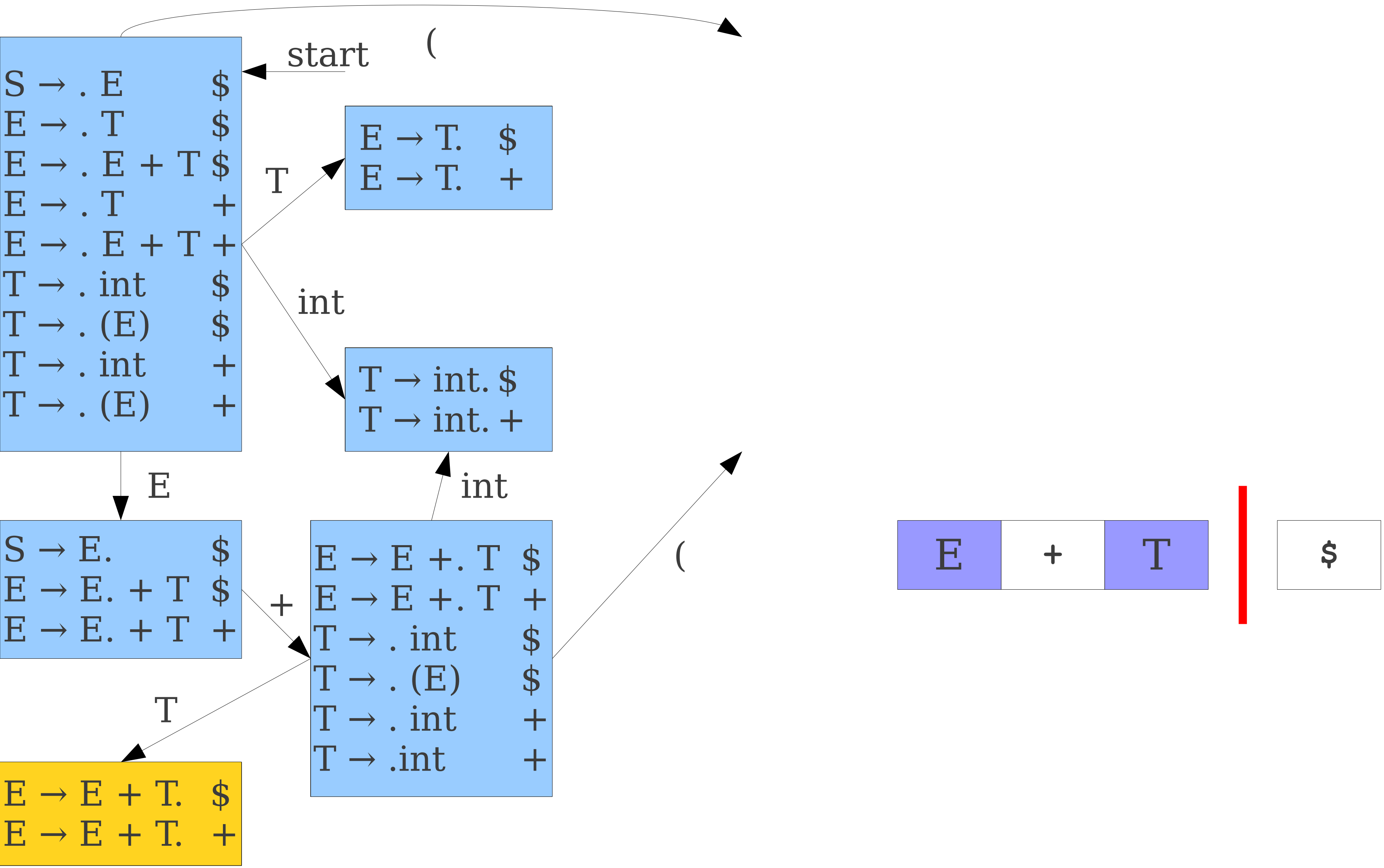
Deterministic LR(1) Automata



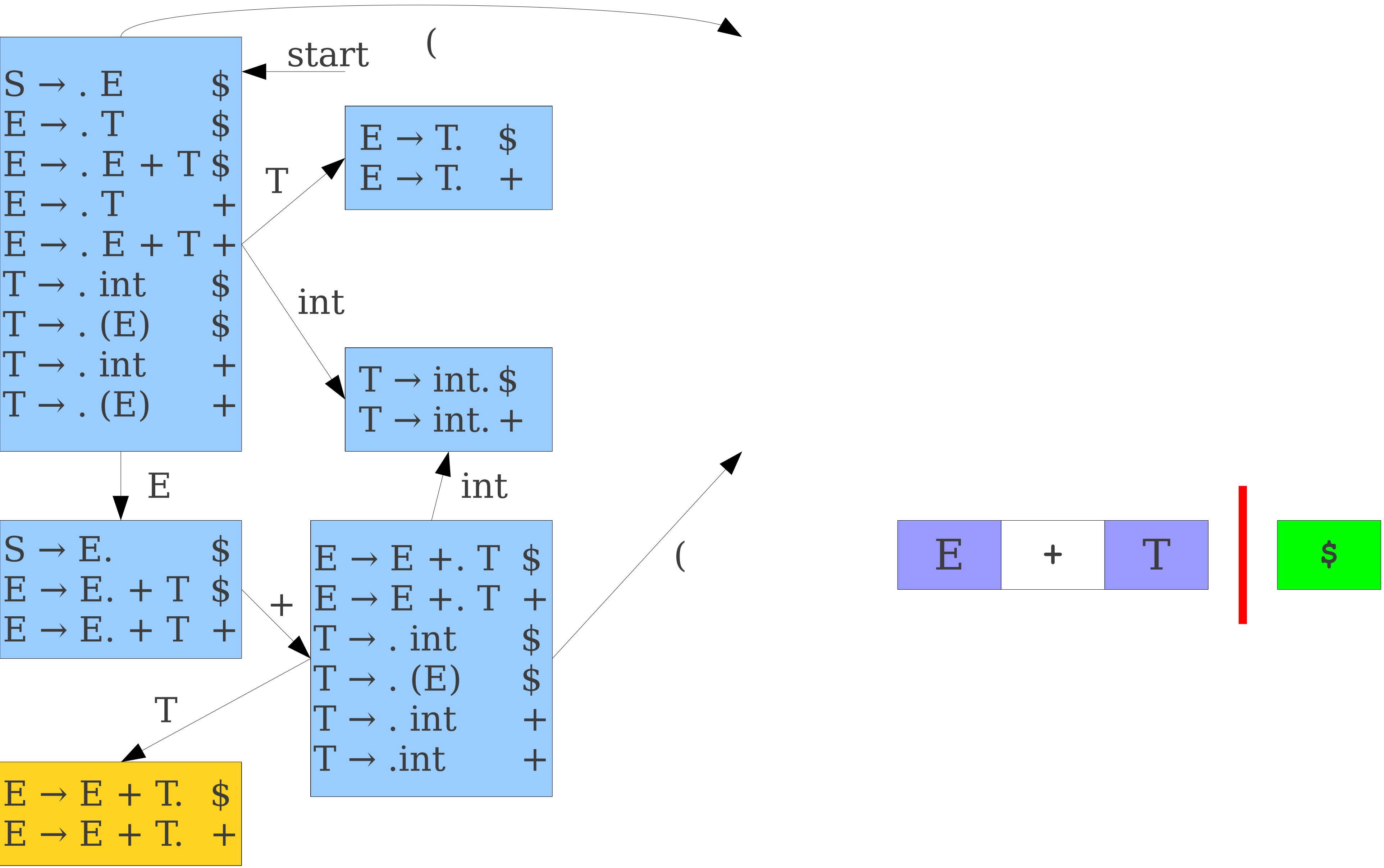
Deterministic LR(1) Automata



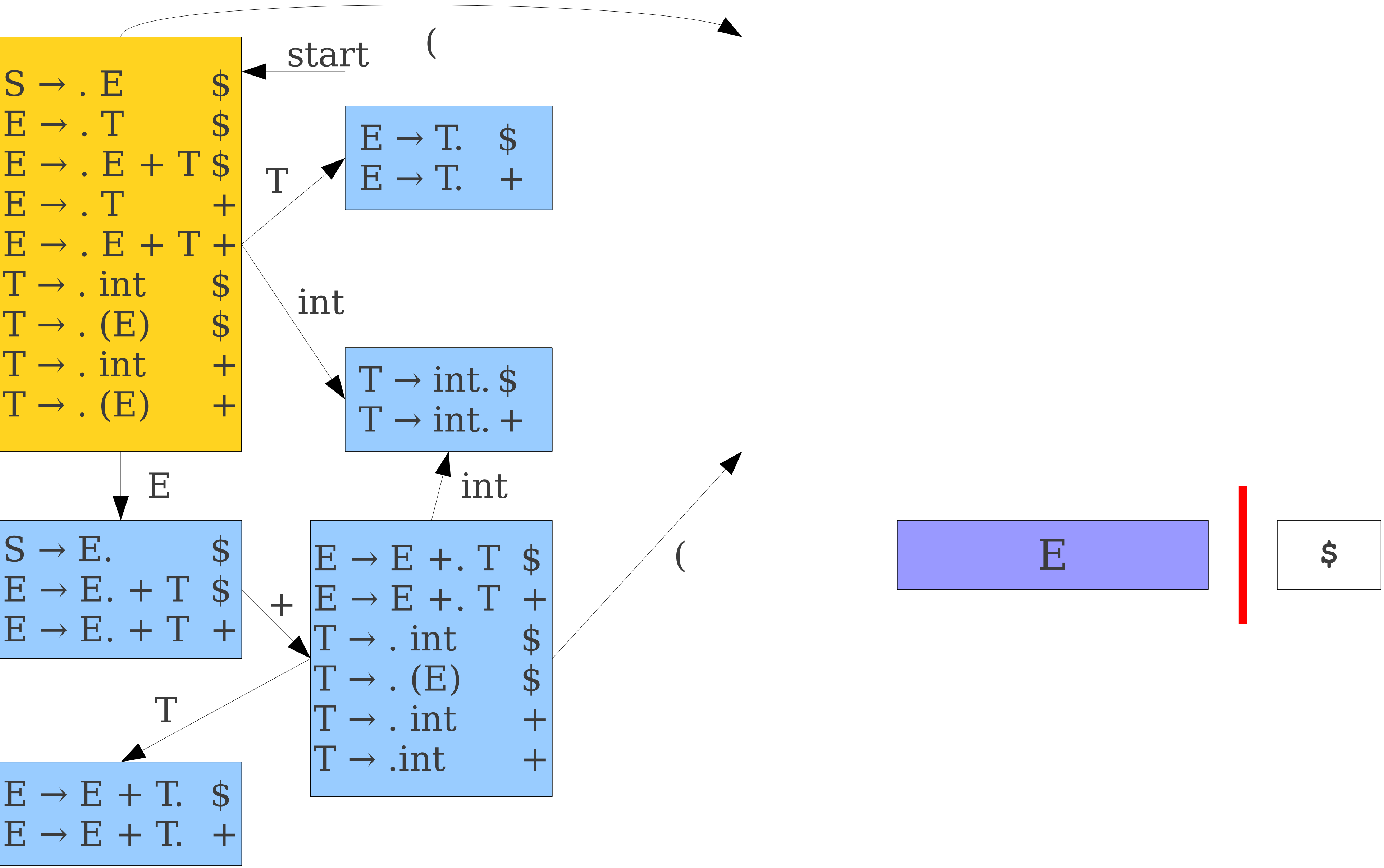
Deterministic LR(1) Automata



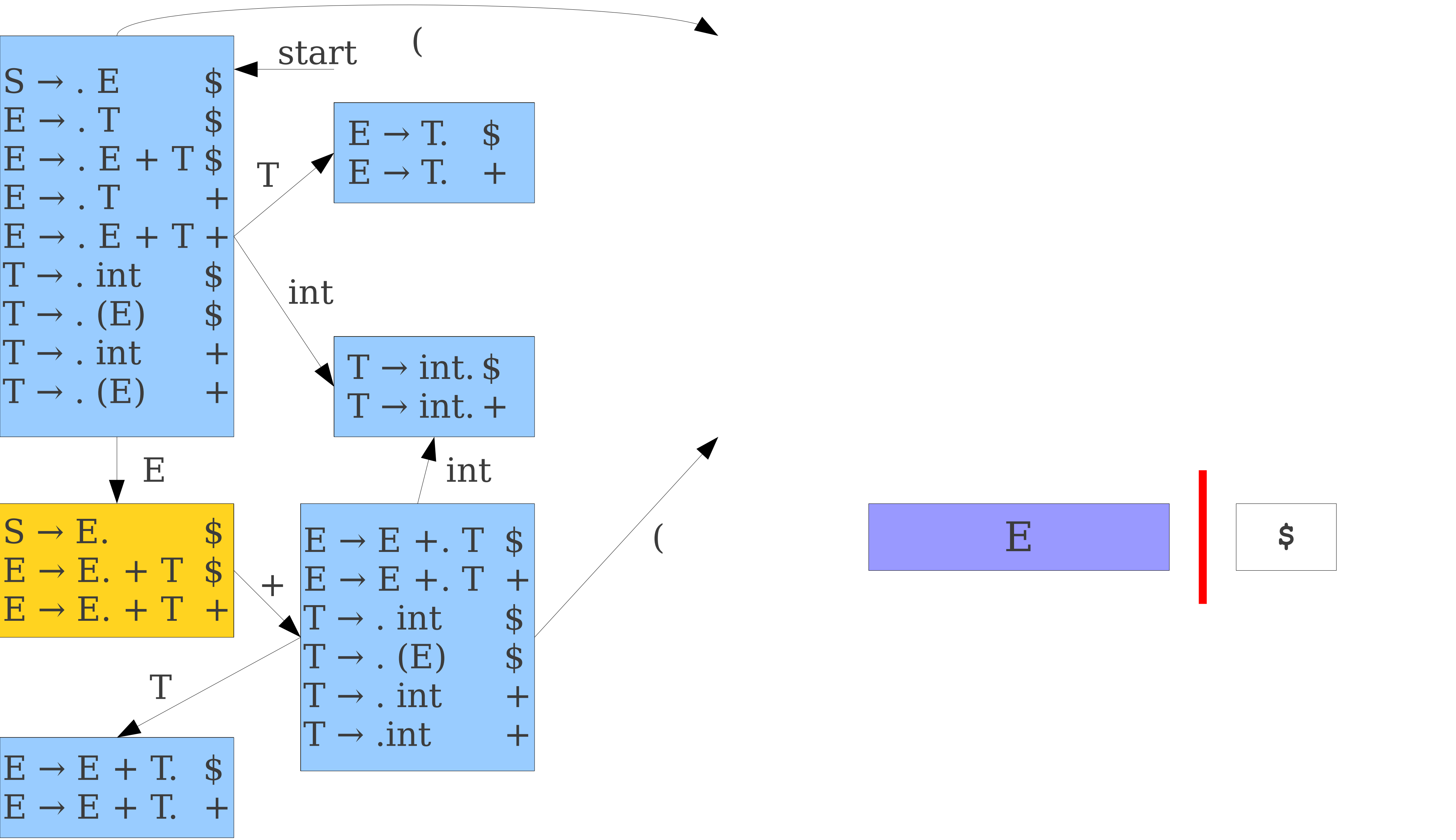
Deterministic LR(1) Automata



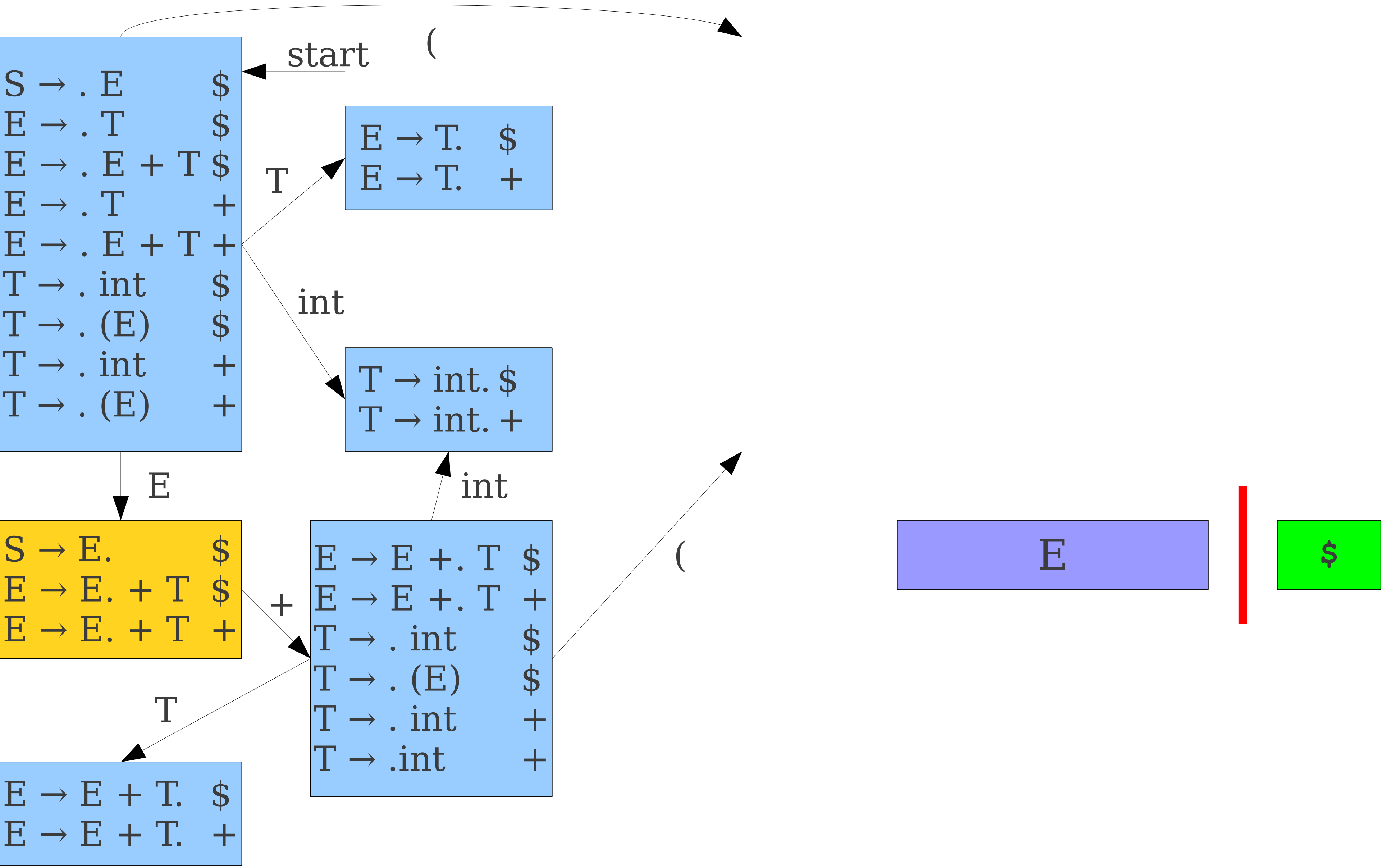
Deterministic LR(1) Automata



Deterministic LR(1) Automata



Deterministic LR(1) Automata



Representing LR(1) Automata

- As with LR(0), use **action** and **goto** tables.
- **goto** table defined as before; encodes transition table as map from (state, token) to states.
- **action** table maps pairs (state, lookahead) to actions.
- Commonly combined into a single **action/goto** table.

Constructing LR(1) Parse Tables

- For each state X:
  - If there is a production **A** → ***ω*** · [`t`], set **action[**X**,** `t`**] = reduce A** → ***ω***.
  - If there is the special production **S** → **E** · [`$`], where **S** is the start symbol, set **action[**X**,** `t`**] = accept**.
  - If there is a transition out of X on symbol `t`, set **action[**X**,** `t`**] = shift**.
- Set all other actions to **error**.
- If any table entry contains two or more actions, the grammar is not LR(1).

$\mathbf{S} \rightarrow \mathbf{E}$ (1)

$\mathbf{E} \rightarrow \mathbf{T}$ (2)

$\mathbf{E} \rightarrow \mathbf{E} + \mathbf{T}$ (3)

$\mathbf{T} \rightarrow$ `int` (4)

$\mathbf{T} \rightarrow$ `(E)` (5)

	int	(	)	+	$	T	E
1	s5					s4	s2
2				s6	ACCEPT		
3				r3	r3		
4				r2	r2		
5				r5	r5		
6	s5	s7				s3	
7	s10	s14				s10	s8
8			s9	s12			
9				r5	r5		
10			r2	r2			
11			r4	r4			
12	s11					s13	
13			r3	r3			
14	s11		s14			s10	s15
15			s16	s12			
16			r5	r5			

The LR(1) Parsing Algorithm

- Begin with an empty stack and the input set to ω`$`, where ω is the string to parse. Set **state** to the initial state.
- Repeat the following:
  - Let the next symbol of input be `t`.
  - If **action[state, `t`]** is **shift**, then shift the input and set **state = goto[state, `t`]**.
  - If **action[state, `t`]** is **reduce A** → ω:
    - Pop $|\omega|$ symbols off the stack; replace them with **A**.
    - Let the state atop the stack be **top-state**.
    - Set **state = goto[top-state**, **A]**
  - If **action[state, `t`]** is **accept**, then the parse is done.
  - If **action[state, `t`]** is **error**, report an error.

The Power of LR(1)

- Any LR(0) grammar is LR(1).
- Any LL(1) grammar is LR(1).
- Any deterministic CFL (a CFL parseable by a *deterministic pushdown automaton*) has an LR(1) grammar.
- Any LL(k) *language* is LR(1), though individual LL(k) *grammars* might not be.
- Any LR(k) *language* is LR(1), though individual LR(k) *grammars* might not be.

LR(1) Automata are **Huge**

- In a grammar with *n* terminals, could in theory be $O(2^n)$ times as large as the LR(0) automaton.

  复制

  - Replicate each state with all $O(2^n)$ possible lookaheads.
- LR(1) tables for practical programming languages can have hundreds of thousands or even *millions* of states.
- Consequently, LR(1) parsers are rarely used in practice.

Is there a way to get the power of LR(1) without the huge table size?

Why is LR(1) so powerful?

- Intuitively, for two reasons:
- **Lookahead makes handle-finding easier**.
  - The LR(0) automaton says whether there could be a handle later on based on no right context.
  - The LR(1) automaton can predict whether it needs to reduce based on more information.
- **More states encode more information**.
  - LR(1) lookaheads are very good because there's a greater number of states to be in.
- **Goal**: Incorporate lookahead without increasing the number of states.

合并，嵌入

Revisiting Shift/Reduce Conflicts

- A shift/reduce conflict is a state that looks like this:

  $$\mathbf{A} \rightarrow \boldsymbol{\omega} \cdot$$

  $$\mathbf{B} \rightarrow \boldsymbol{\alpha} \cdot \boldsymbol{\beta}$$

- In LR(0), this is simply not allowed.
- In LR(1), this can be avoided by using lookahead to determine whether to shift or reduce.
- Can we get some of the lookahead power of LR(1) without the huge tables?

SLR(1)

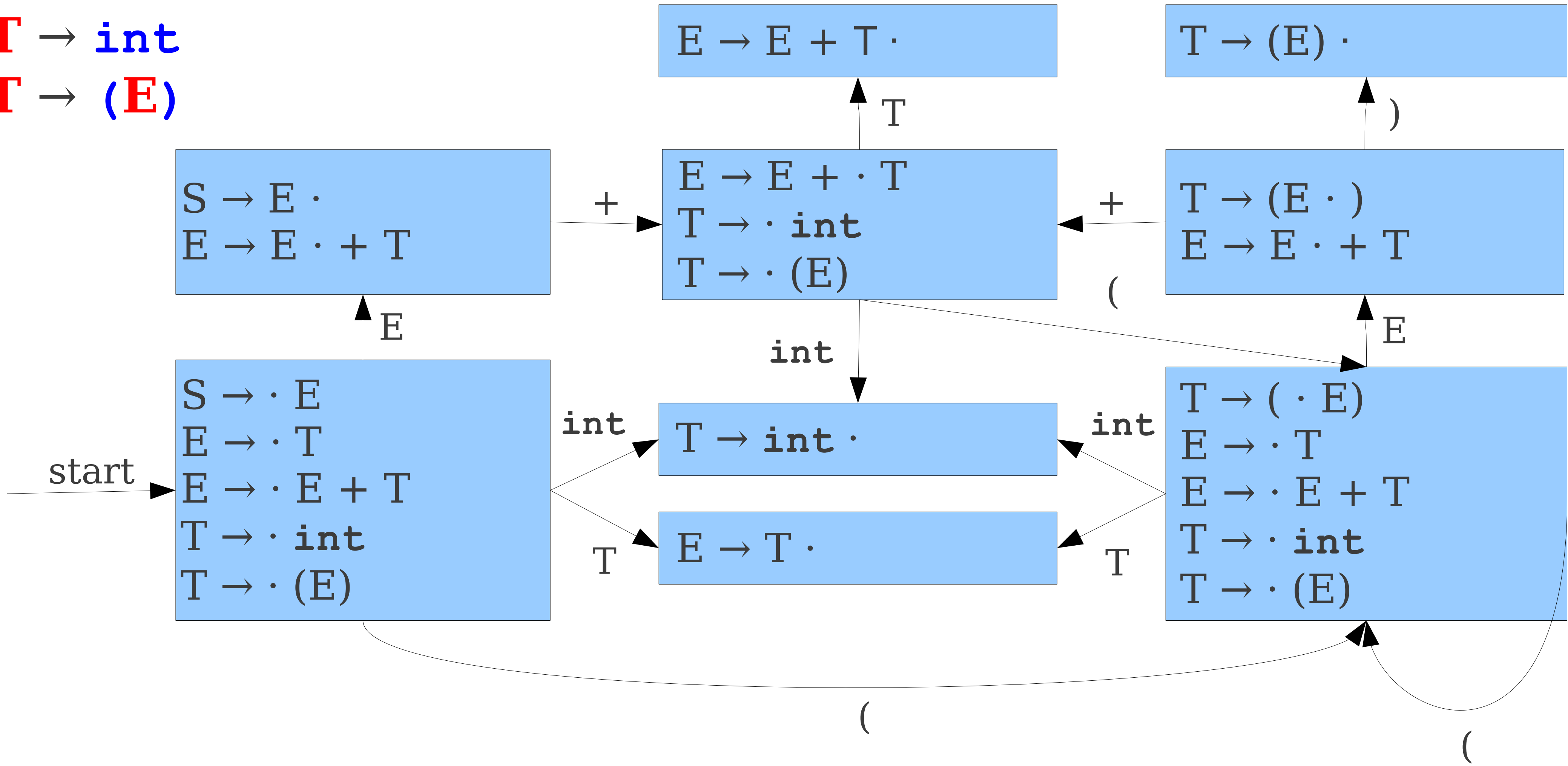
- **Simple LR(1)** 小小的改动
- Minor modification to LR(0) automaton that uses lookahead to avoid shift/reduce conflicts.
- Idea: Only reduce **A** → $\boldsymbol{\omega}$ if the next token `t` is in FOLLOW(**A**).
- Automaton identical to LR(0) automaton; only change is when we choose to reduce.

$S \to E$
$E \to T$
$E \to E + T$
$T \to$ `int`
$T \to (E)$

SLR(1) Parsing

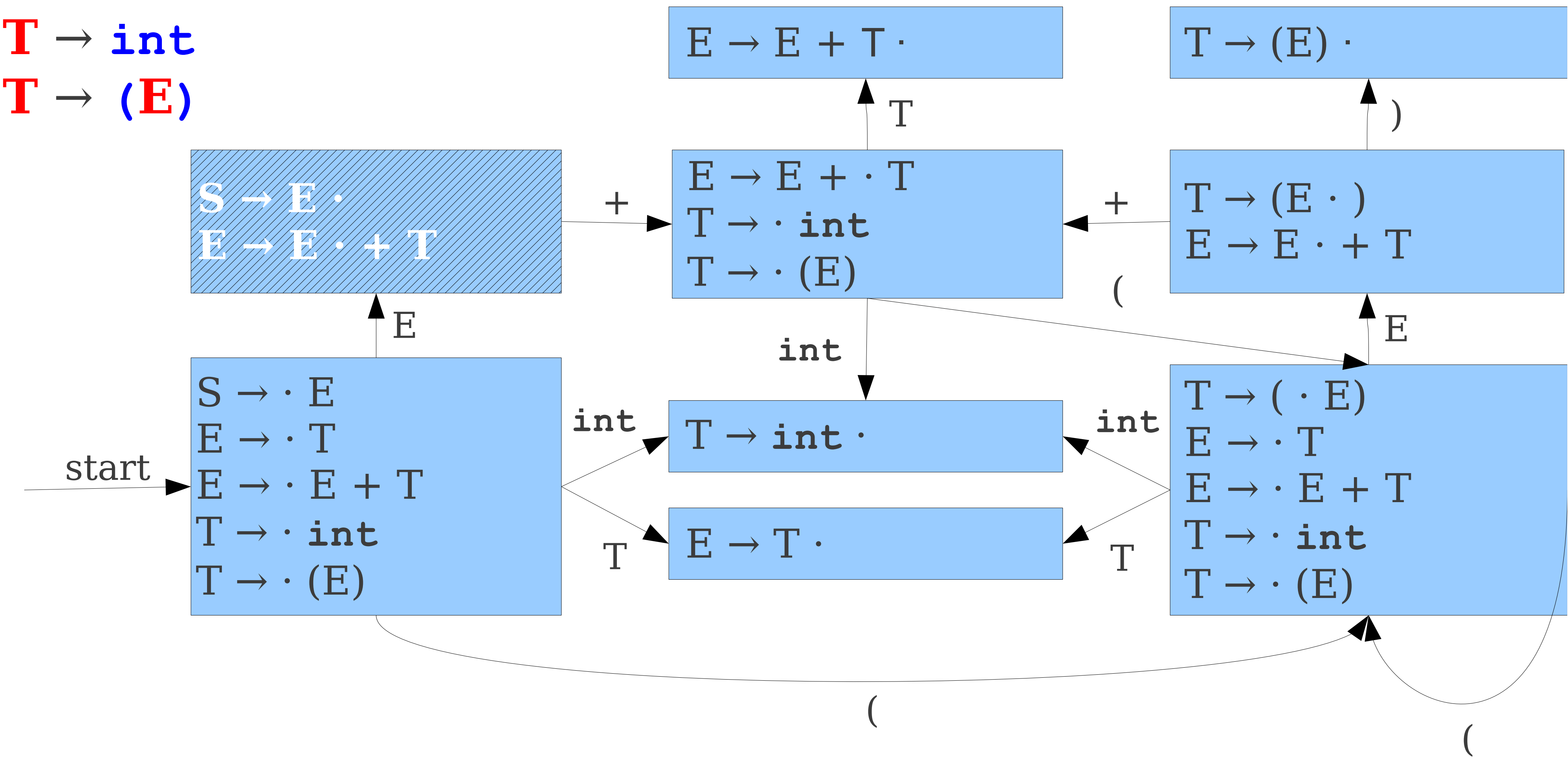
$S \to E$
$E \to T$
$E \to E + T$
$T \to$ `int`
$T \to$ (E)

SLR(1) Parsing



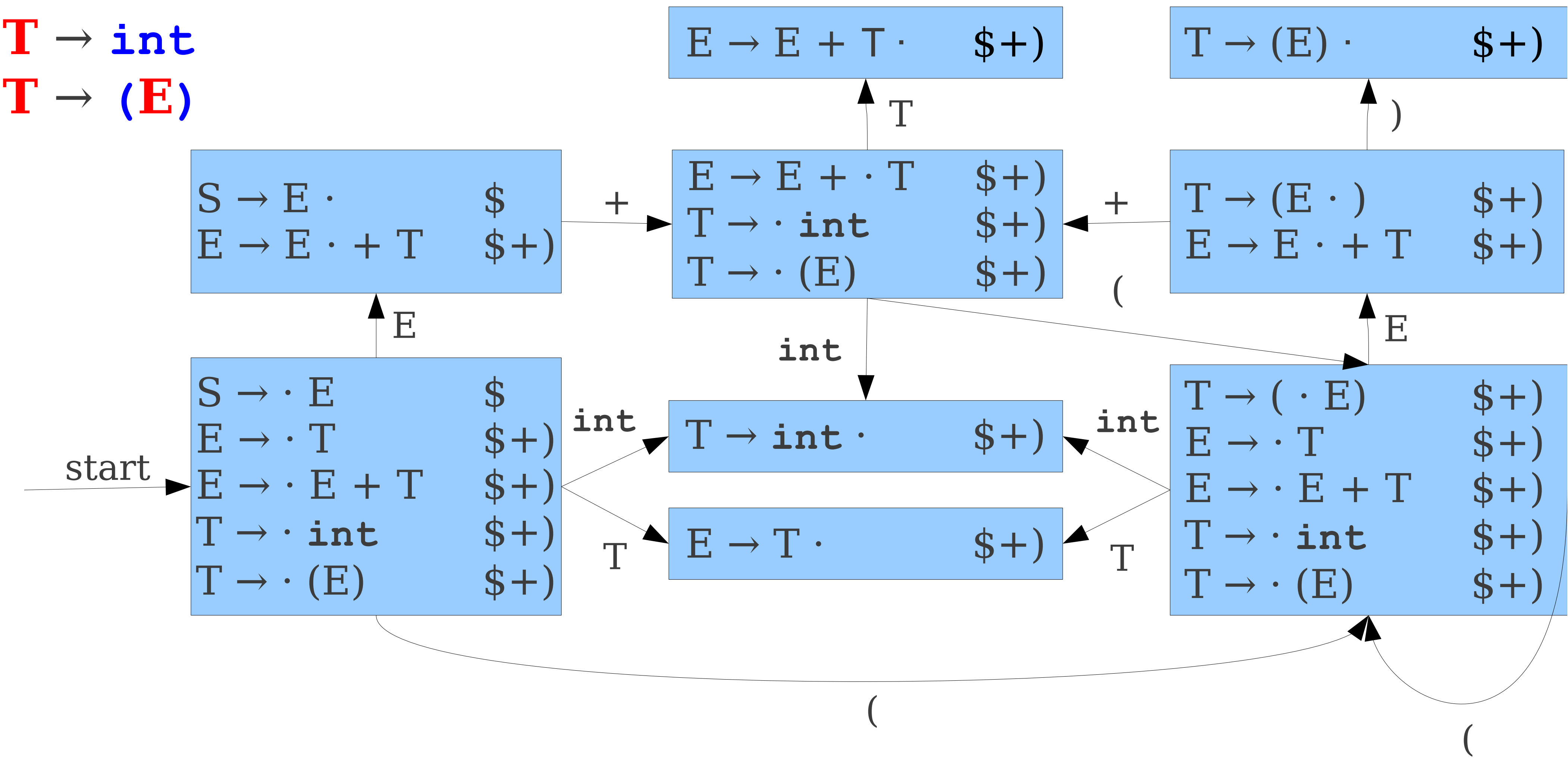
$S \rightarrow E$
$E \rightarrow T$
$E \rightarrow E + T$
$T \rightarrow$ **int**
$T \rightarrow (E)$

SLR(1) Parsing



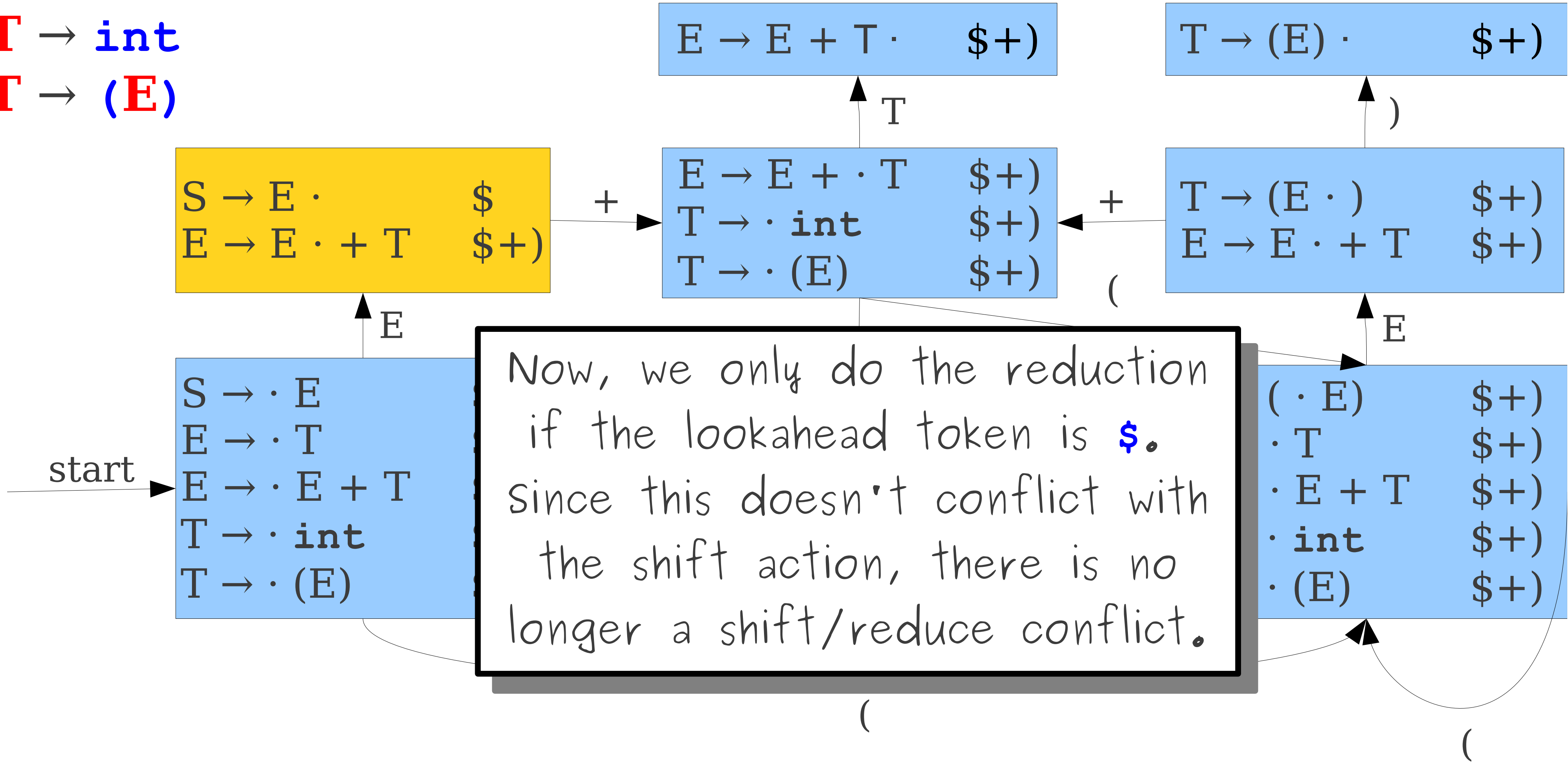
S → **E**
E → **T**
E → **E + T**
T → **`int`**
T → **(E)**

SLR(1) Parsing



$S \rightarrow E$
$E \rightarrow T$
$E \rightarrow E + T$
$T \rightarrow$ **int**
$T \rightarrow (E)$

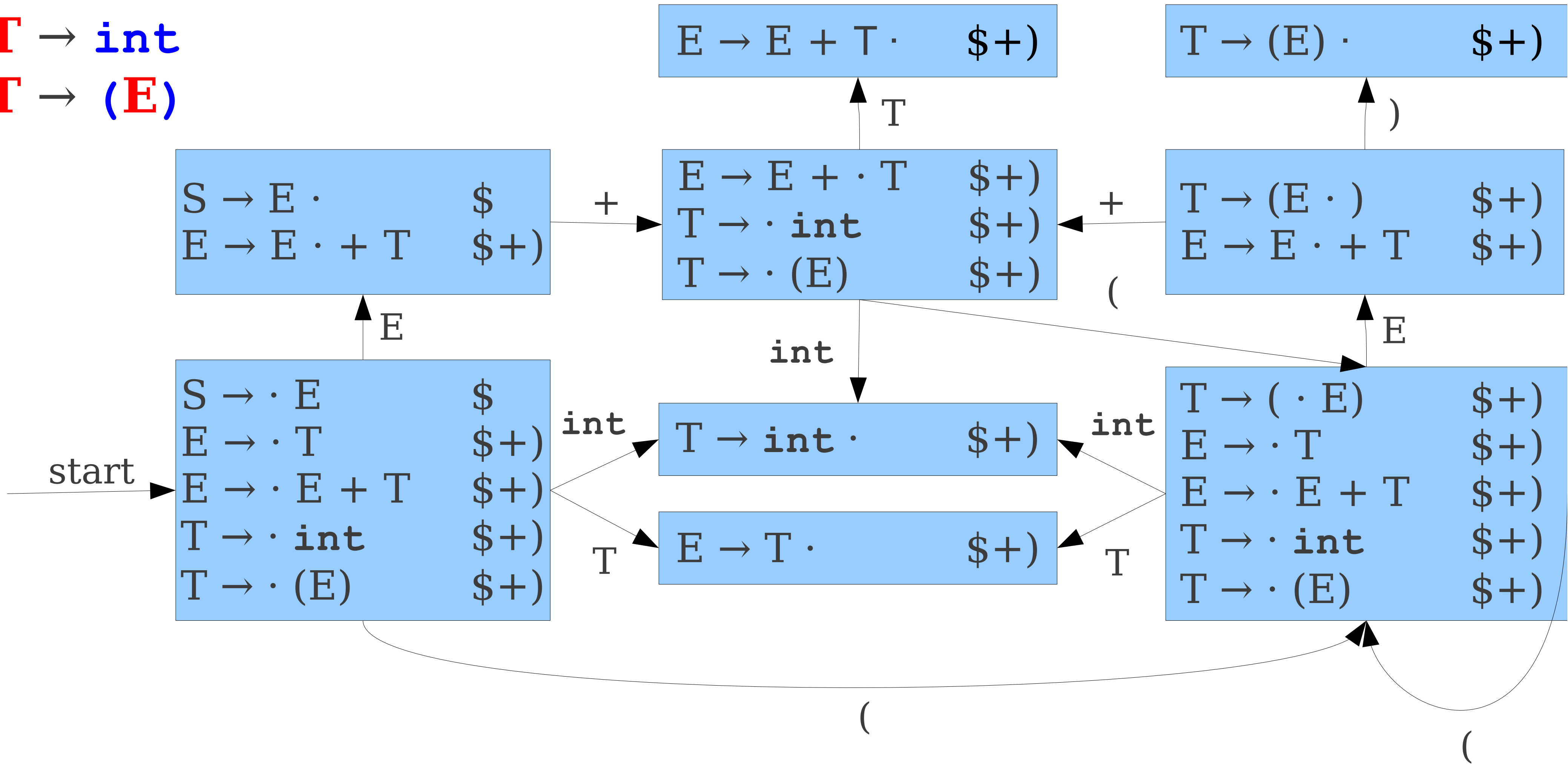
SLR(1) Parsing



Now, we only do the reduction if the lookahead token is **$**. Since this doesn't conflict with the shift action, there is no longer a shift/reduce conflict.

S → **E**
E → **T**
E → **E + T**
T → **`int`**
T → **(E)**

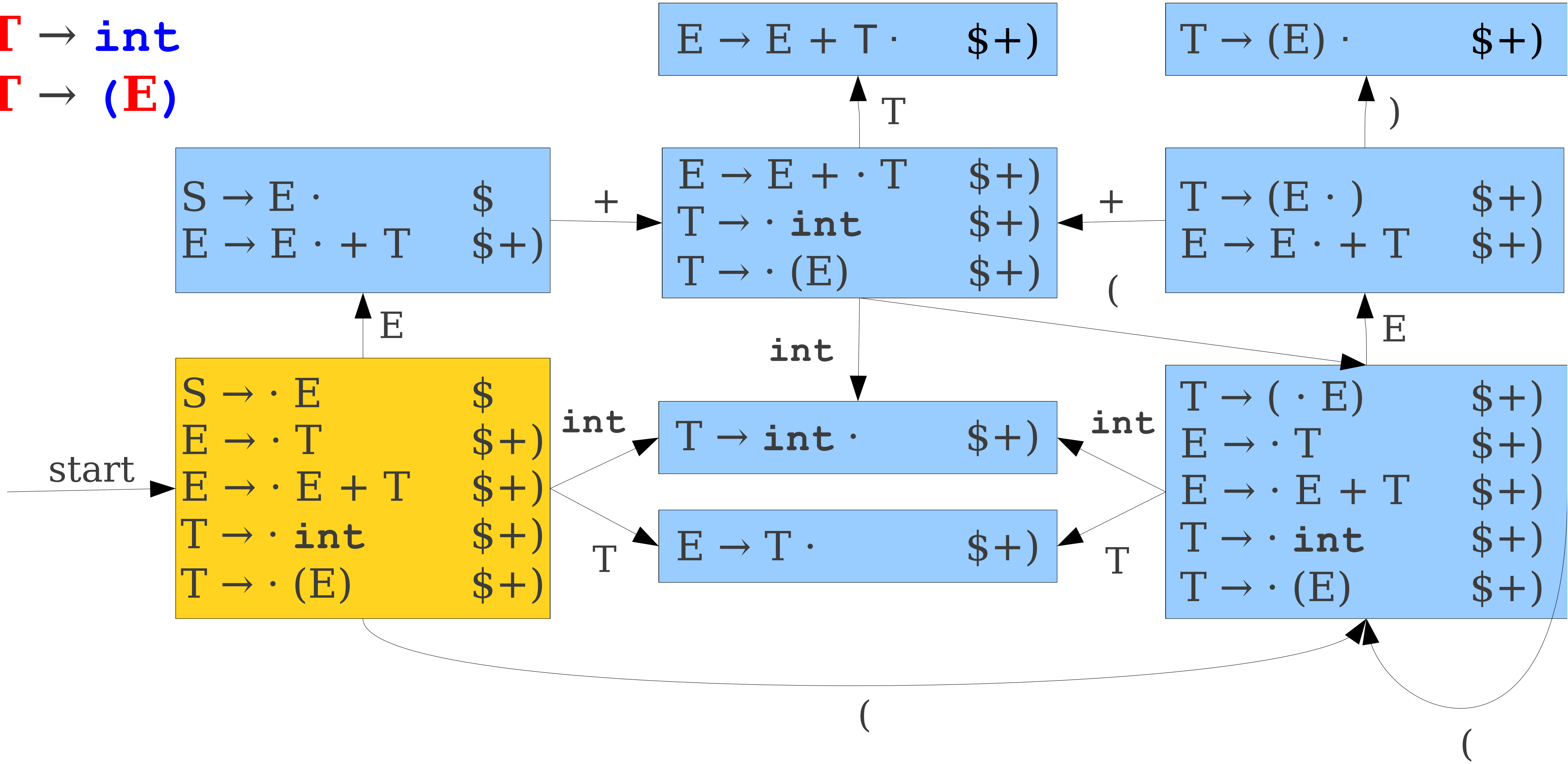
SLR(1) Parsing



int	+	(	int	+	int	+	int	)	$

S → E
E → T
E → E + T
T → int
T → (E)

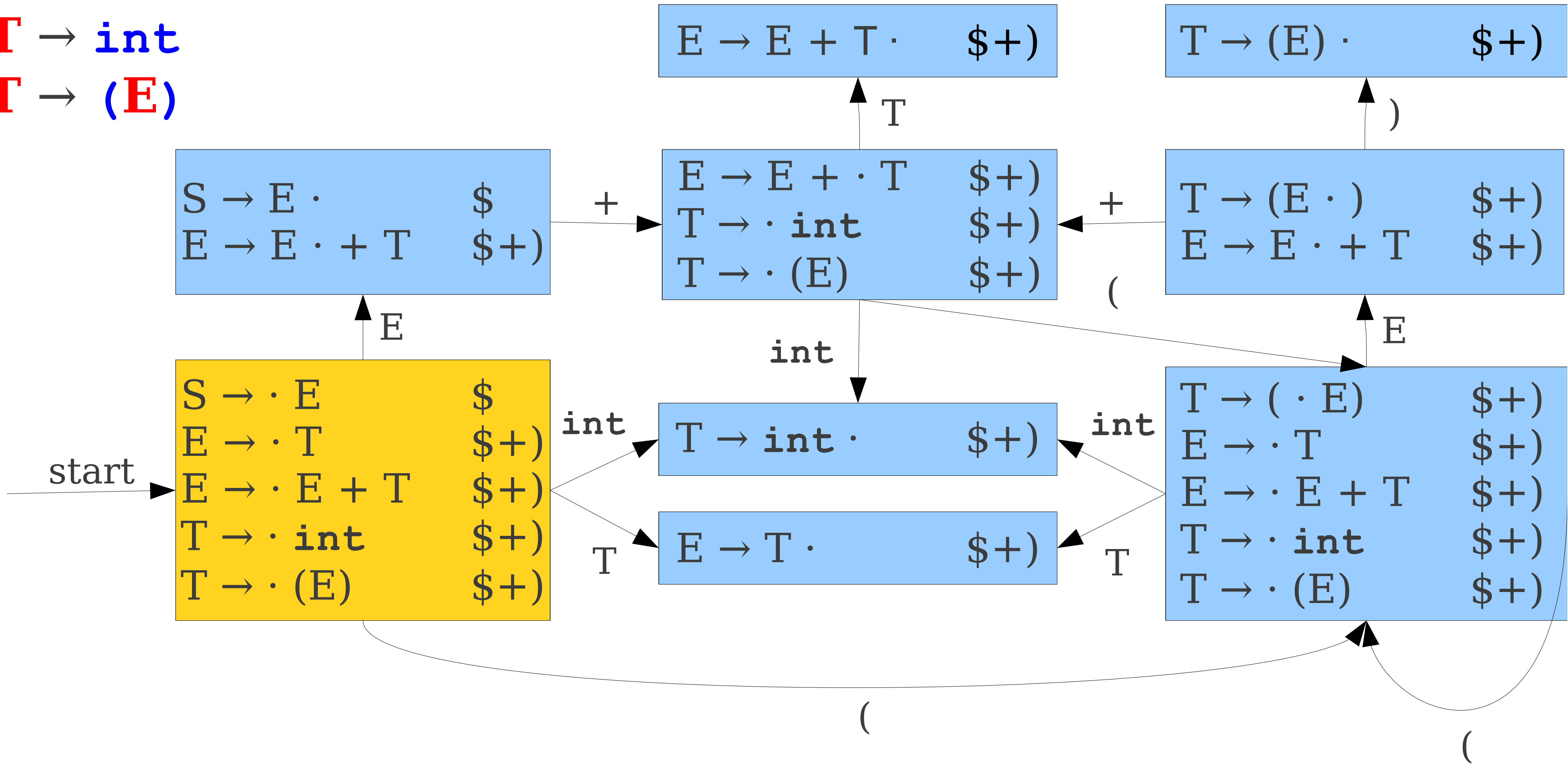
SLR(1) Parsing



int	+	(	int	+	int	+	int	)	$

$S \rightarrow E$
$E \rightarrow T$
$E \rightarrow E + T$
$T \rightarrow$ `int`
$T \rightarrow (E)$

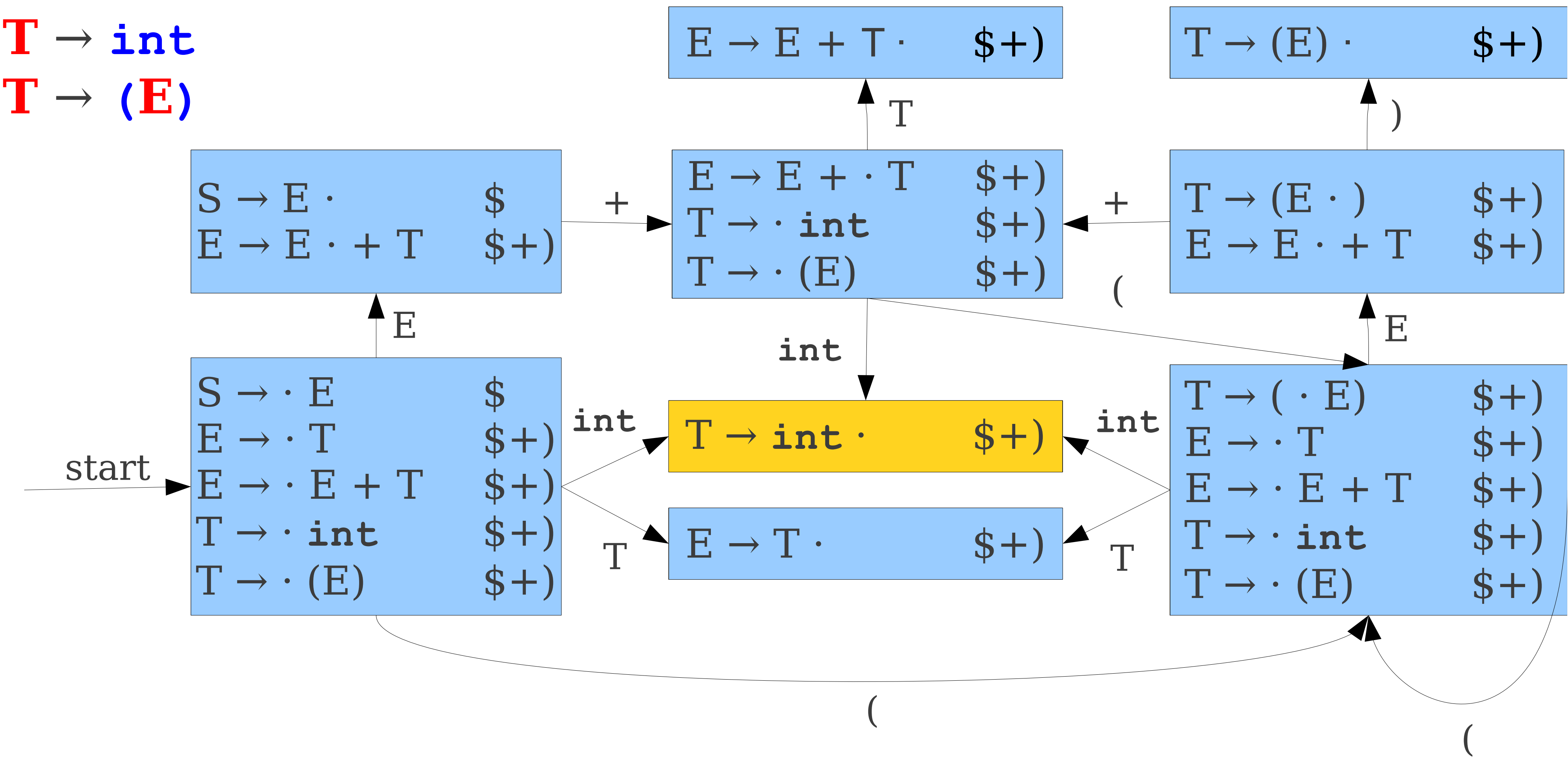
SLR(1) Parsing



int	|	+	(	int	+	int	+	int	)	$

$S \rightarrow E$
$E \rightarrow T$
$E \rightarrow E + T$
$T \rightarrow$ **int**
$T \rightarrow (E)$

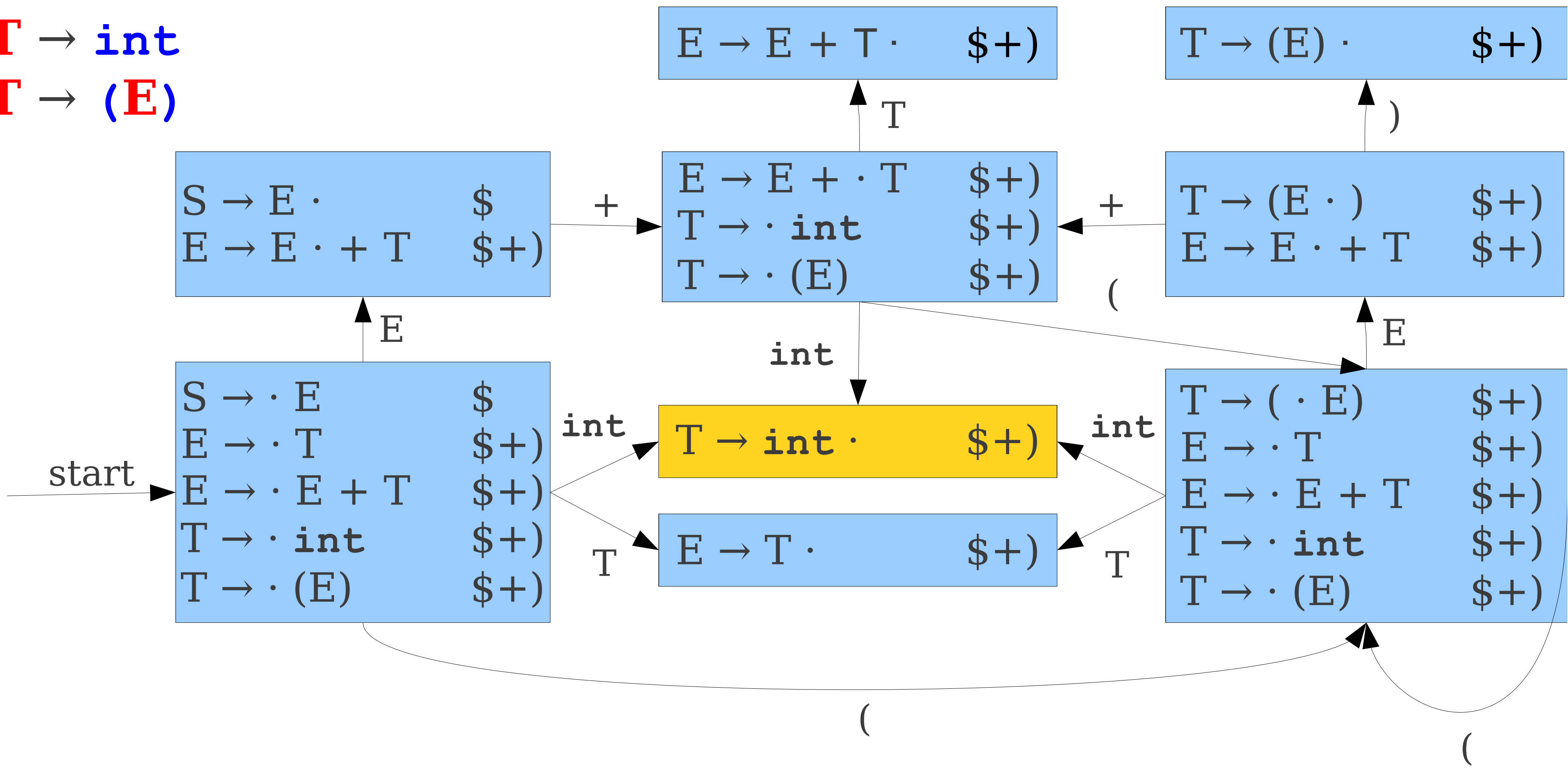
SLR(1) Parsing



int		+	(	int	+	int	+	int	)	$

$S \to E$
$E \to T$
$E \to E + T$
$T \to \texttt{int}$
$T \to (E)$

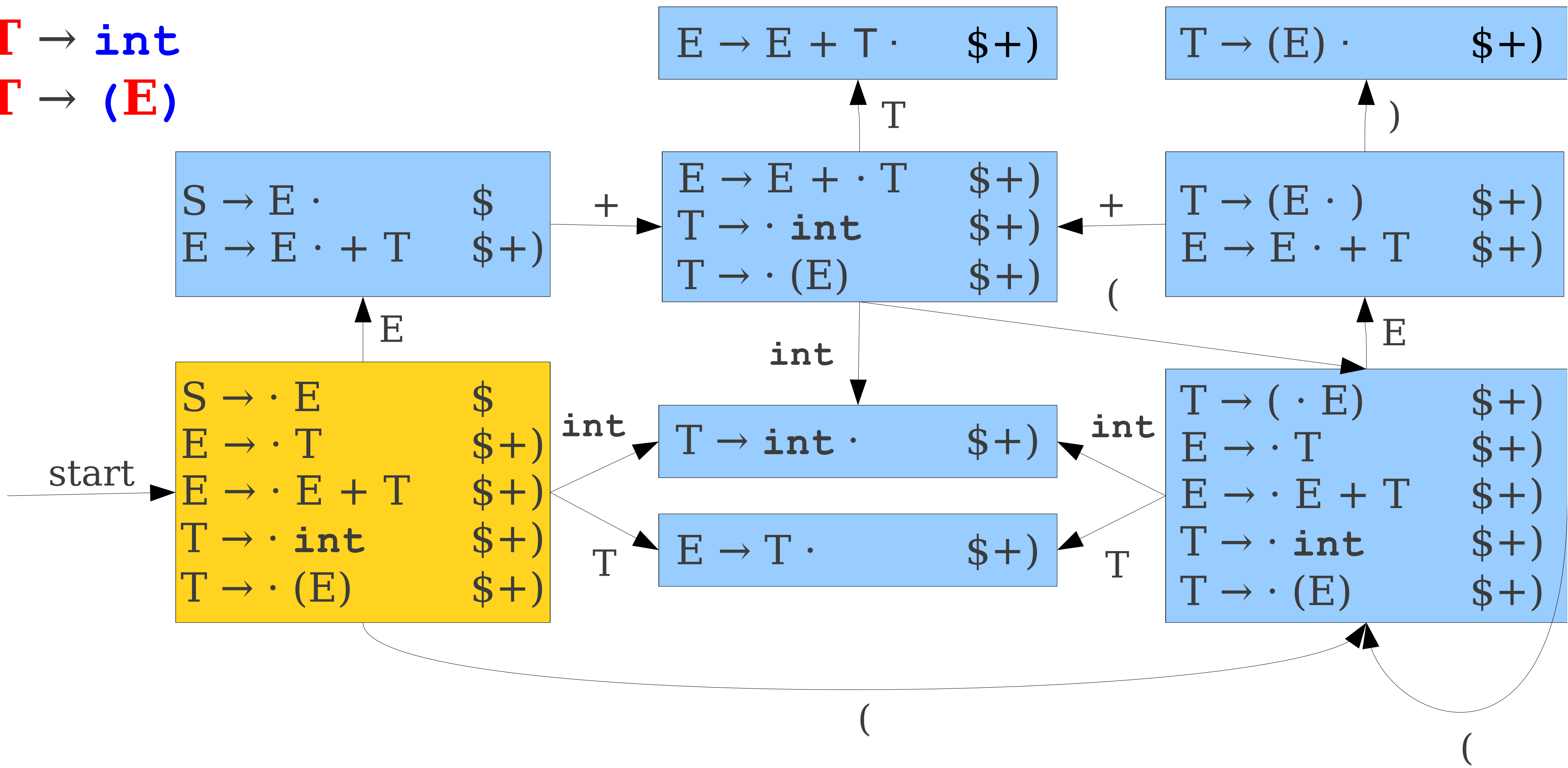
SLR(1) Parsing



int	|	+	(	int	+	int	+	int	)	$

$S \rightarrow E$
$E \rightarrow T$
$E \rightarrow E + T$
$T \rightarrow$ **int**
$T \rightarrow (E)$

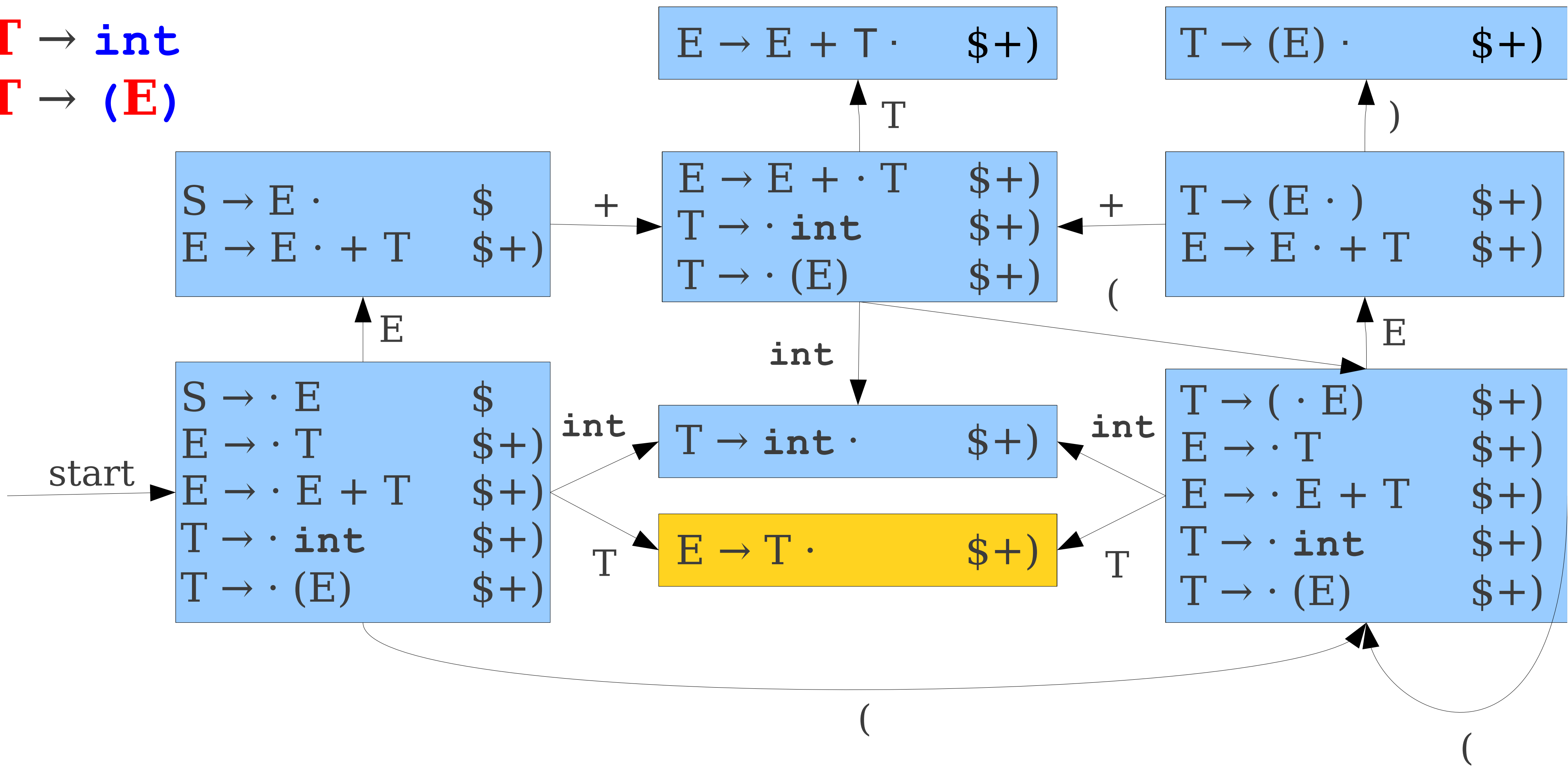
SLR(1) Parsing



T		+	(	int	+	int	+	int	)	$

S → E
E → T
E → E + T
T → int
T → (E)

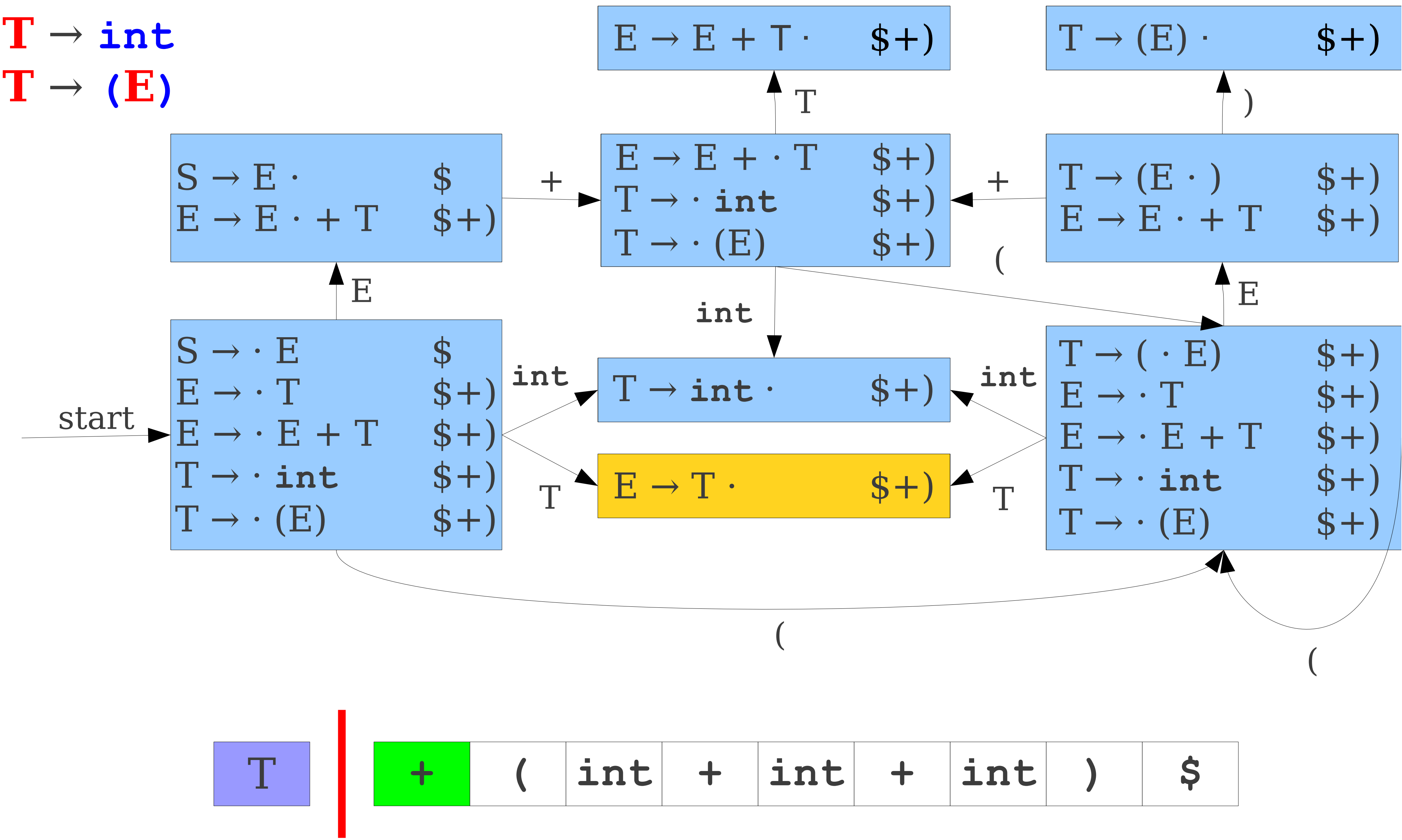
SLR(1) Parsing



T		+	(	int	+	int	+	int	)	$

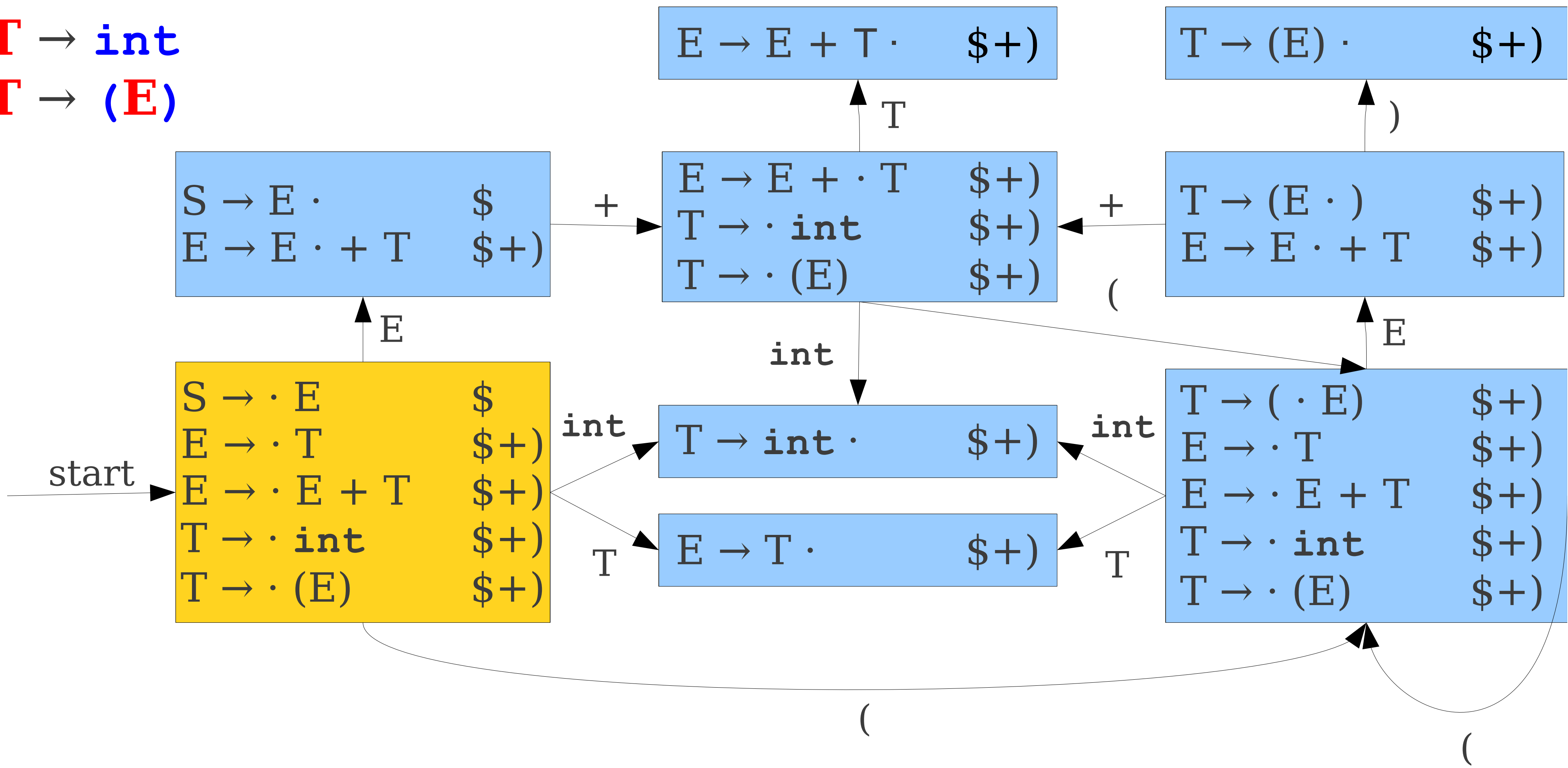
$S \rightarrow E$
$E \rightarrow T$
$E \rightarrow E + T$
$T \rightarrow$ int
$T \rightarrow (E)$

SLR(1) Parsing



S → **E**
E → **T**
E → **E + T**
T → **int**
T → **(E)**

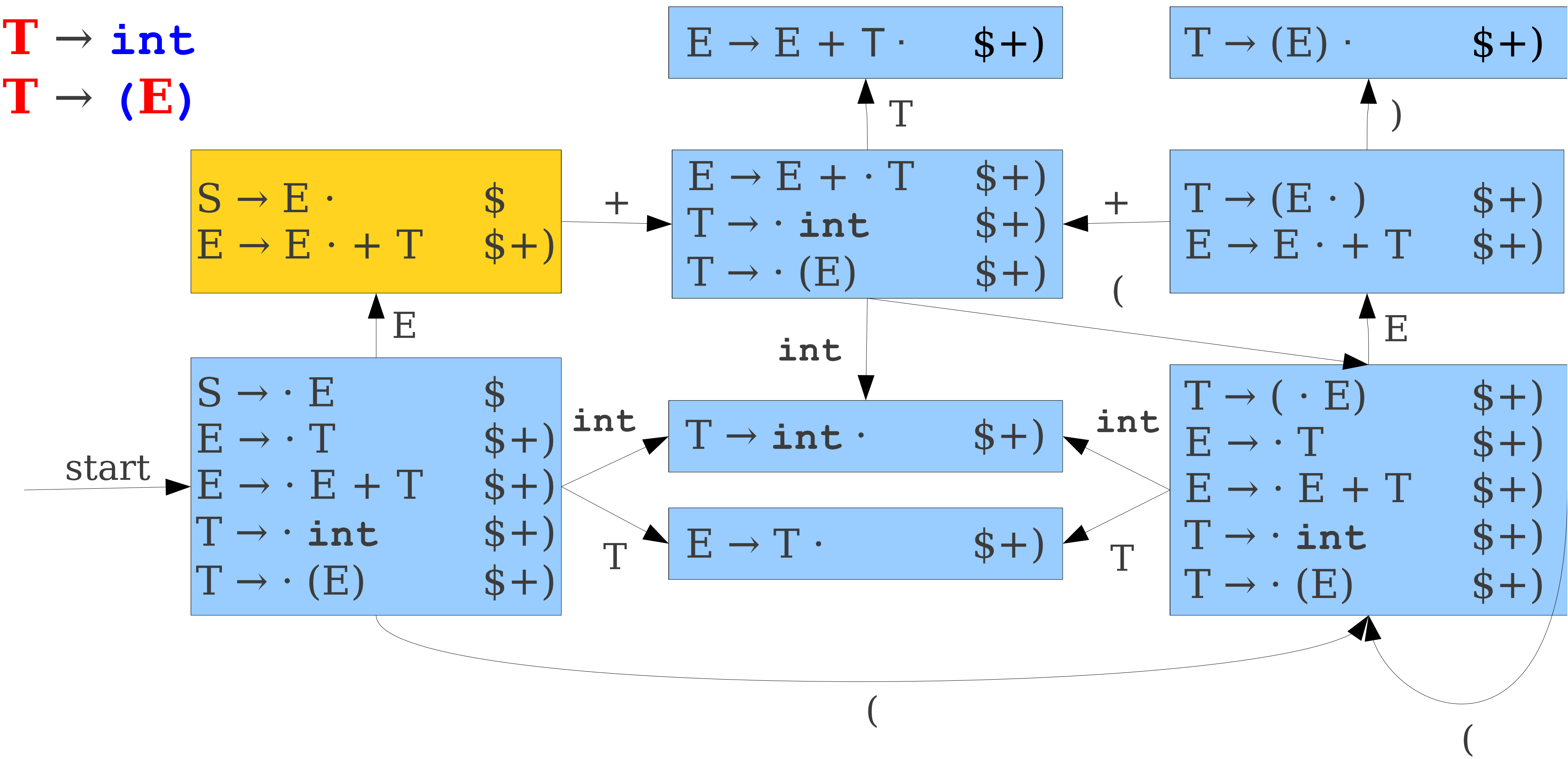
SLR(1) Parsing



E	|	+	(	int	+	int	+	int	)	$

S → E
E → T
E → E + T
T → int
T → (E)

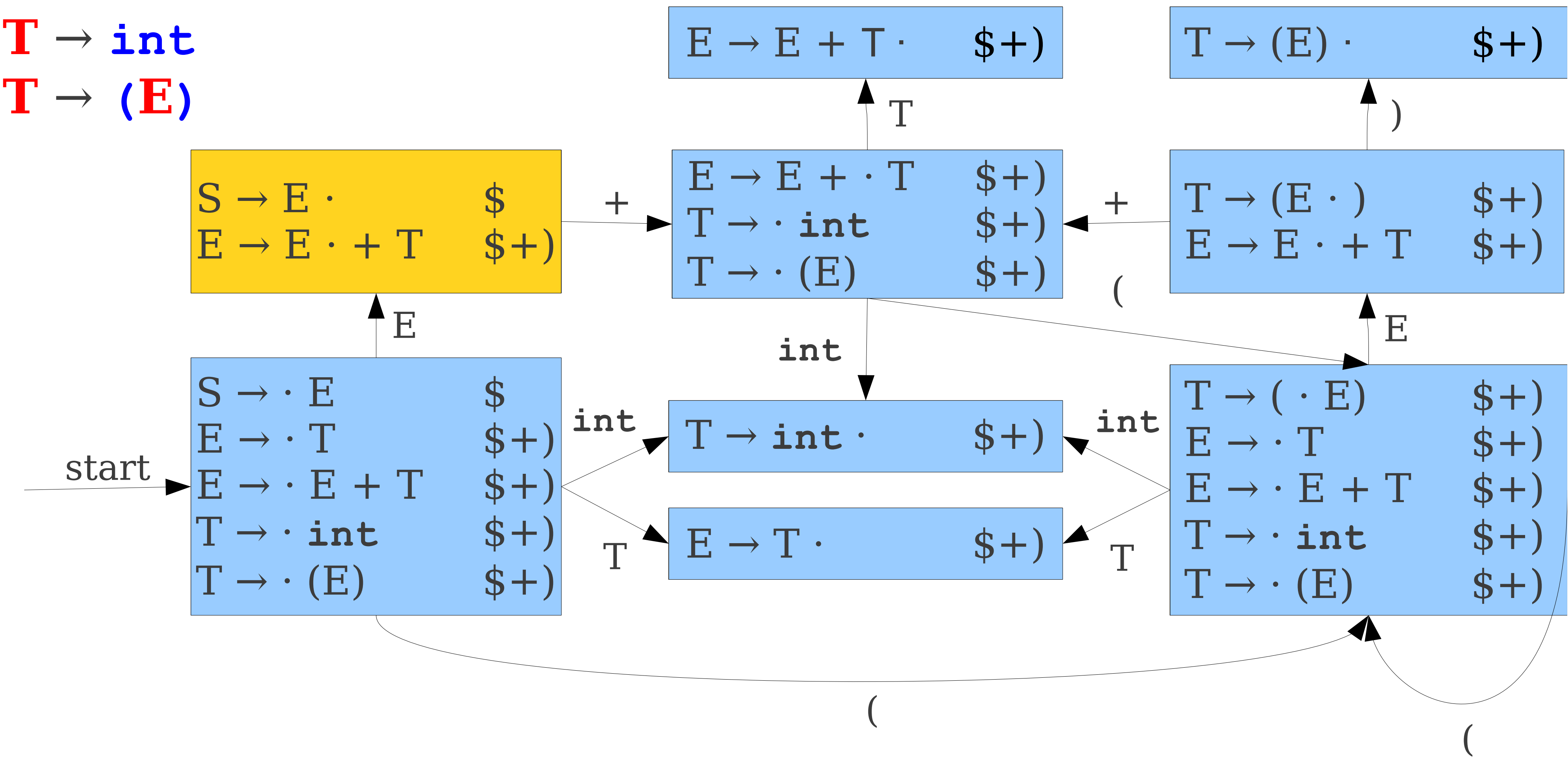
SLR(1) Parsing



E	+	(	int	+	int	+	int	)	$

S → E
E → T
E → E + T
T → int
T → (E)

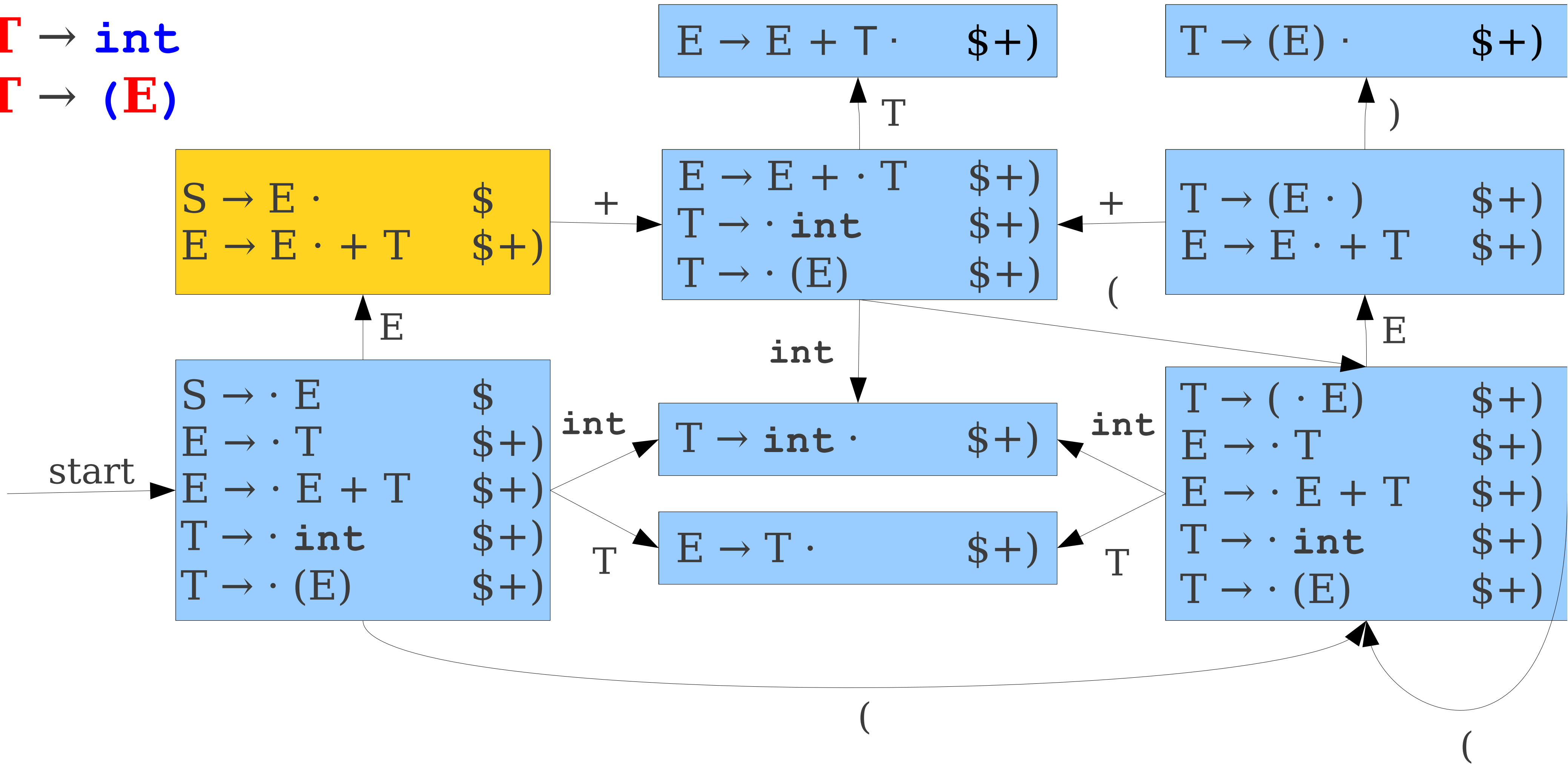
SLR(1) Parsing



E	+	(	int	+	int	+	int	)	$

$S \rightarrow E$
$E \rightarrow T$
$E \rightarrow E + T$
$T \rightarrow$ **int**
$T \rightarrow (E)$

SLR(1) Parsing

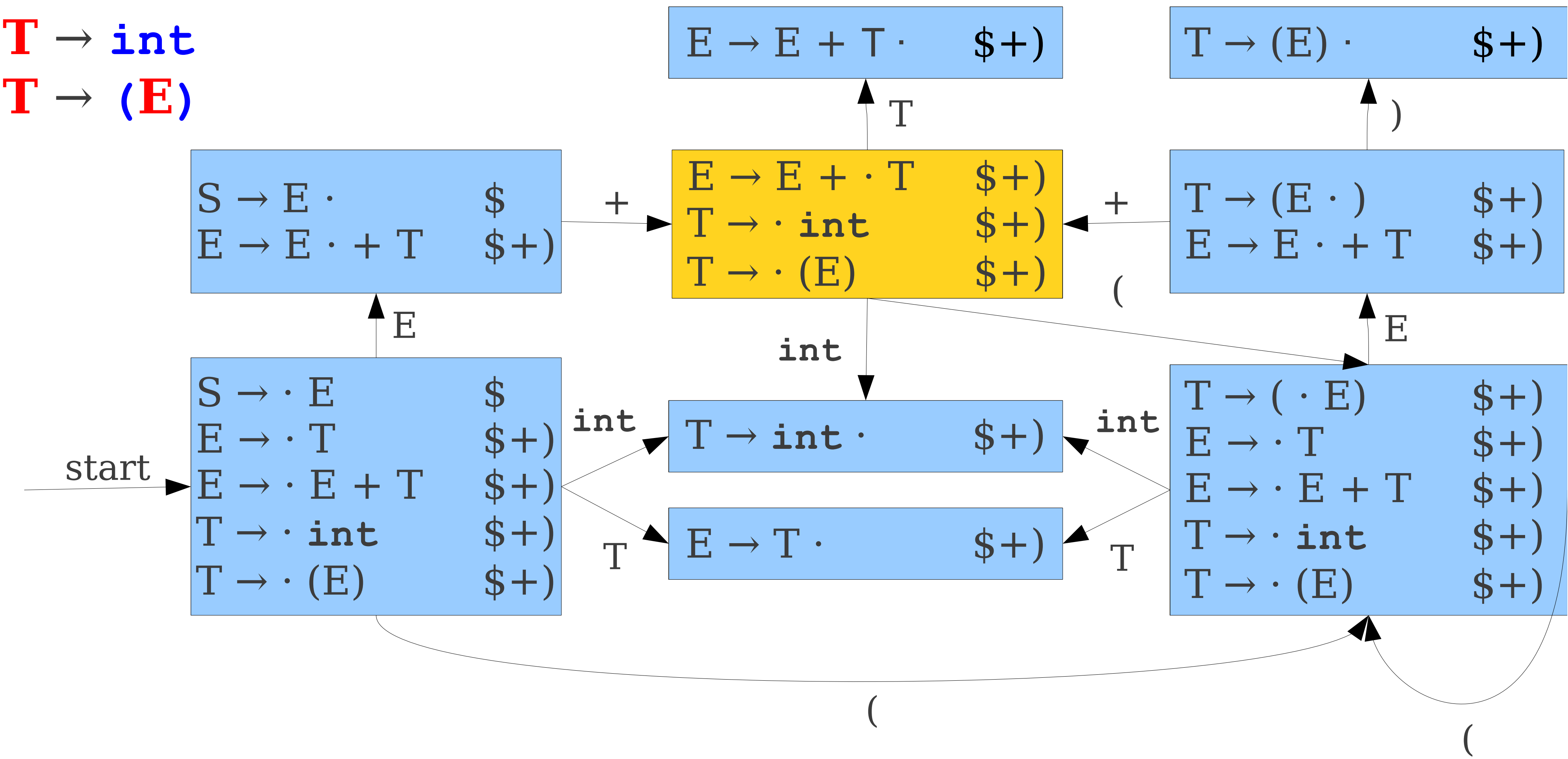


E	+

(	int	+	int	+	int	)	$

$S \to E$
$E \to T$
$E \to E + T$
$T \to \texttt{int}$
$T \to (E)$

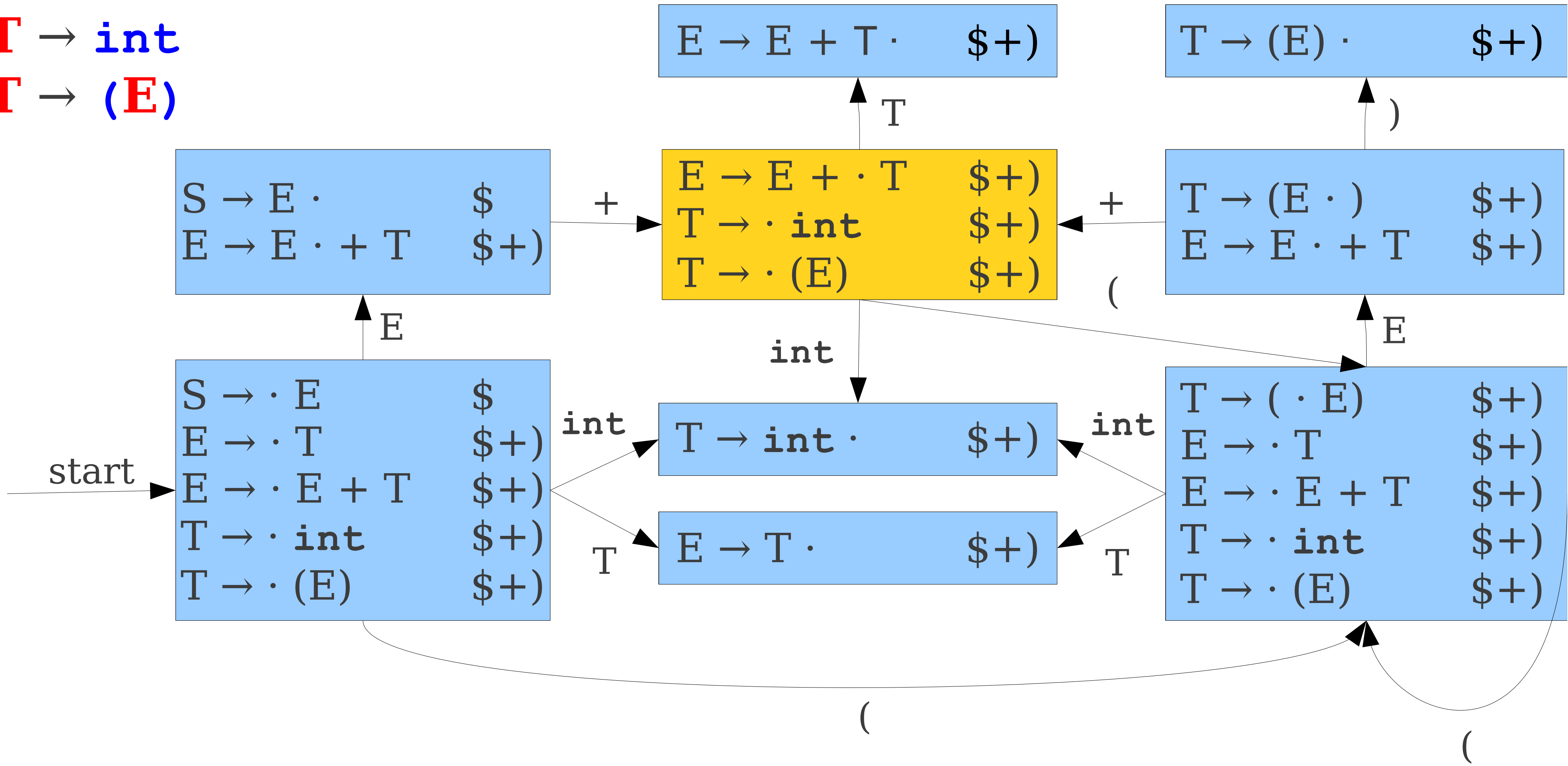
SLR(1) Parsing



E	+		(	int	+	int	+	int	)	$

$S \to E$
$E \to T$
$E \to E + T$
$T \to \texttt{int}$
$T \to (E)$

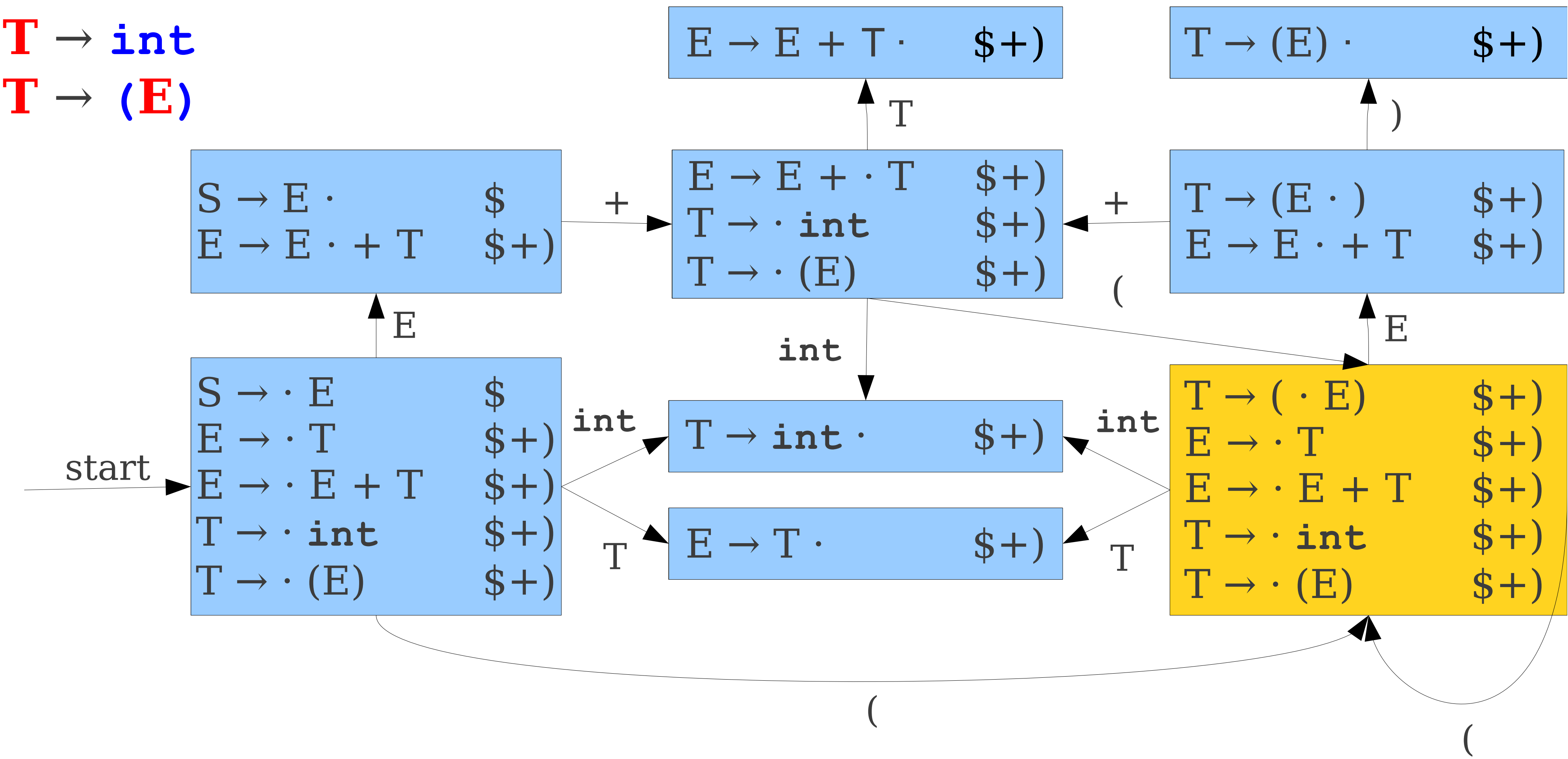
SLR(1) Parsing



E	+	(	int	+	int	+	int	)	$

S → **E**
E → **T**
E → **E** **+** **T**
T → **`int`**
T → **(E)**

SLR(1) Parsing



E	+	(

int	+	int	+	int	)	$

Analysis of SLR(1)

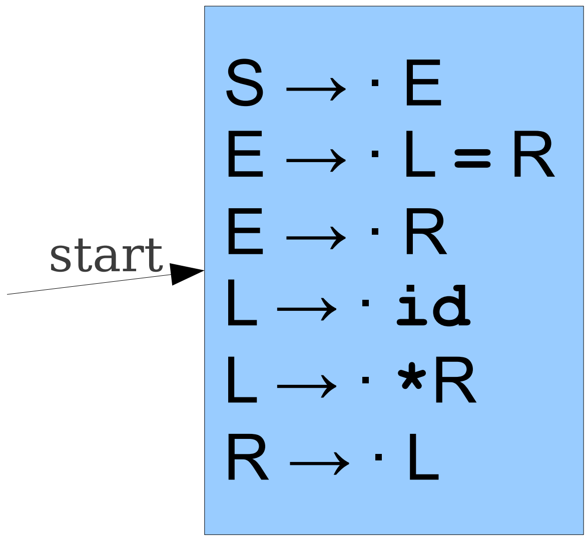
- Exploits lookahead in a small space.
  - Small automaton – same number of states as in as LR(0).
  - Works on many more grammars than LR(0)
- Too weak for most grammars: lose context from not having extra states.

The Limits of SLR(1)

$\mathbf{S} \rightarrow \mathbf{E}$
$\mathbf{E} \rightarrow \mathbf{L}$ `=` $\mathbf{R}$
$\mathbf{E} \rightarrow \mathbf{R}$
$\mathbf{L} \rightarrow$ `id`
$\mathbf{L} \rightarrow$ `*`$\mathbf{R}$
$\mathbf{R} \rightarrow \mathbf{L}$

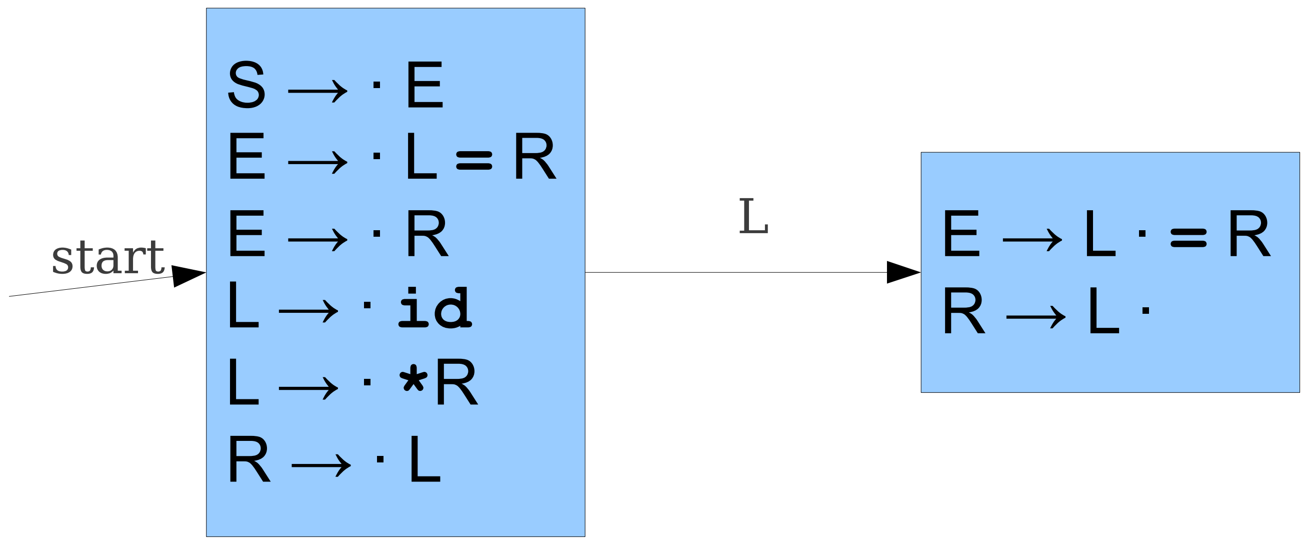
The Limits of SLR(1)

$S \to E$
$E \to L = R$
$E \to R$
$L \to \texttt{id}$
$L \to {*}R$
$R \to L$



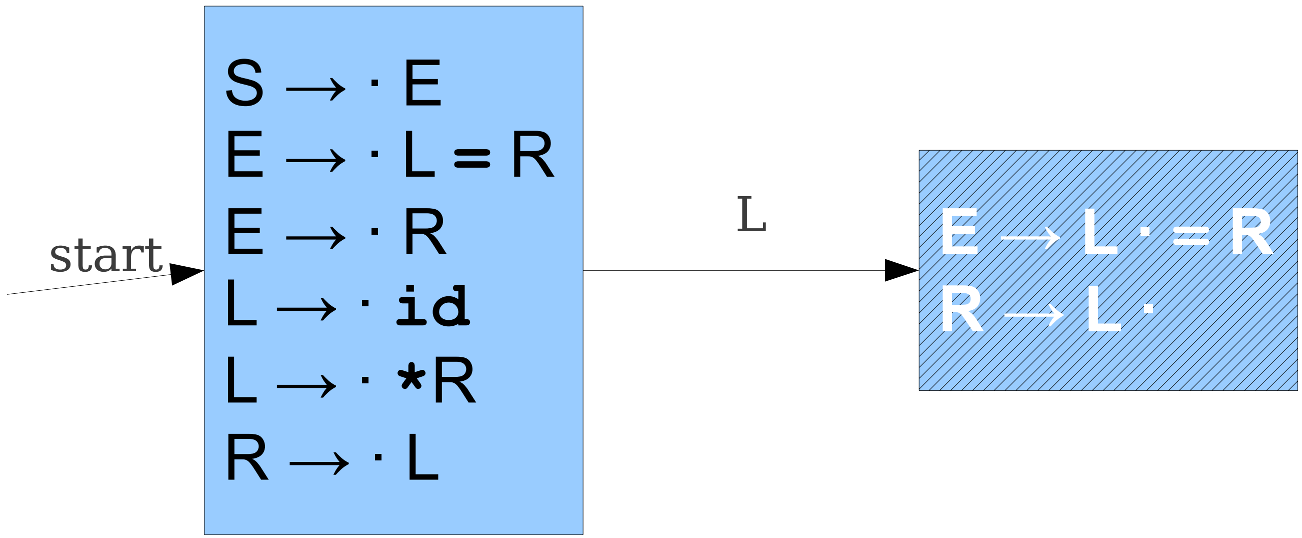
The Limits of SLR(1)

$S \rightarrow E$
$E \rightarrow L = R$
$E \rightarrow R$
$L \rightarrow \texttt{id}$
$L \rightarrow \texttt{*}R$
$R \rightarrow L$



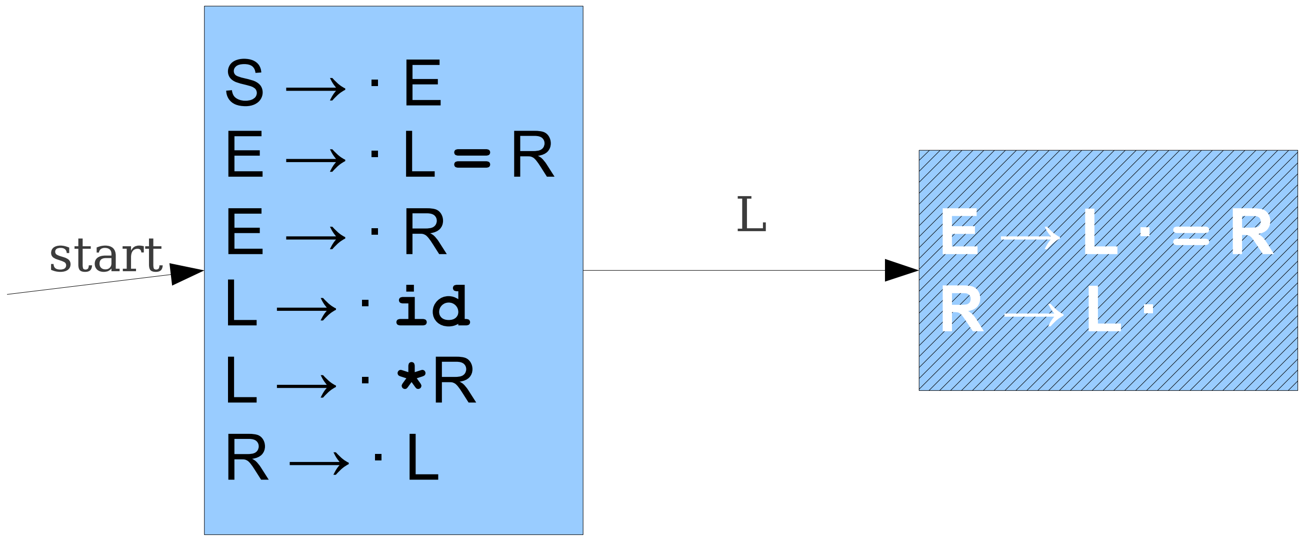
The Limits of SLR(1)

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



The Limits of SLR(1)

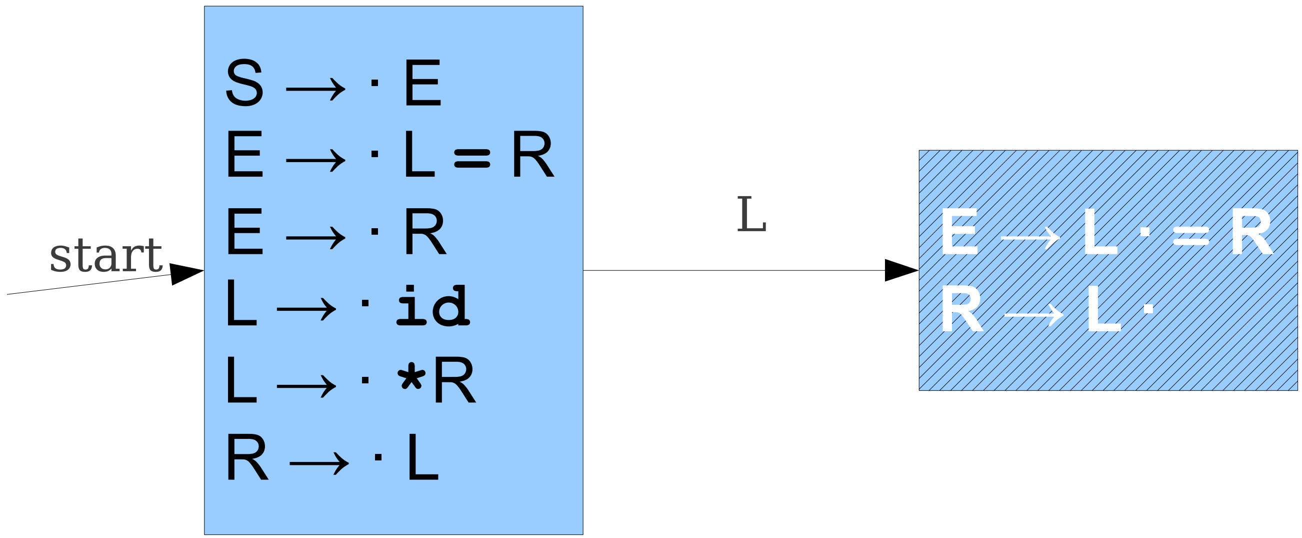
S → **E**
E → **L** `=` **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**



E → **L** · `=` **R** tells us to shift on seeing `=`
R → **L** · tells us to reduce on FOLLOW(**R**).

The Limits of SLR(1)

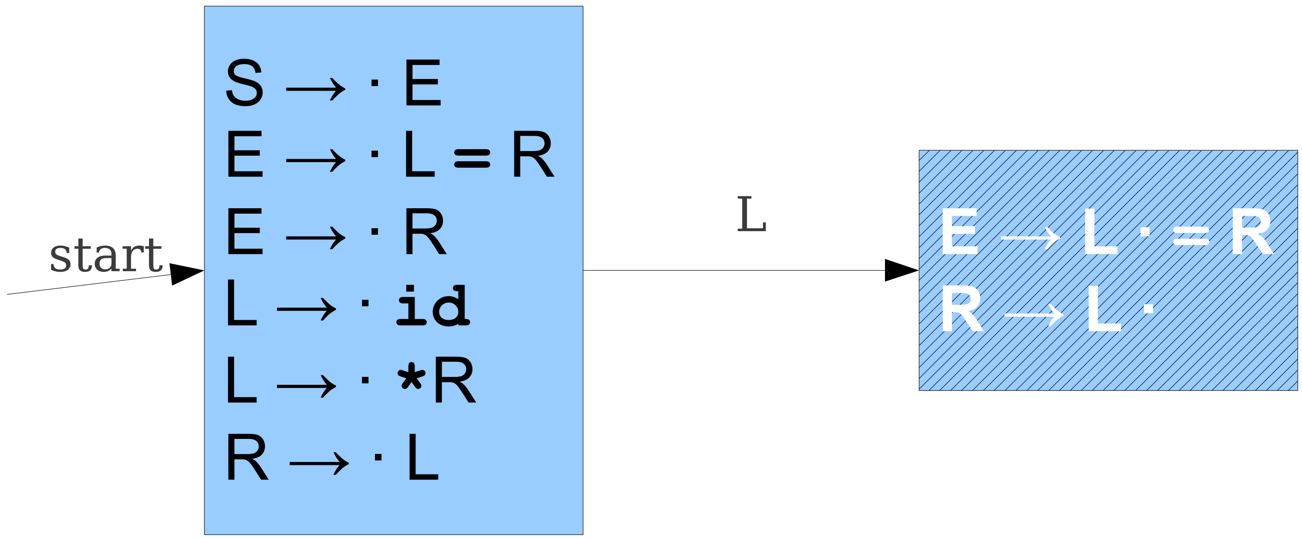
$S \to E$
$E \to L = R$
$E \to R$
$L \to \mathtt{id}$
$L \to *R$
$R \to L$



$E \to L \cdot = R$ tells us to shift on seeing =
$R \to L \cdot$ tells us to reduce on FOLLOW(R).

The Limits of SLR(1)

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



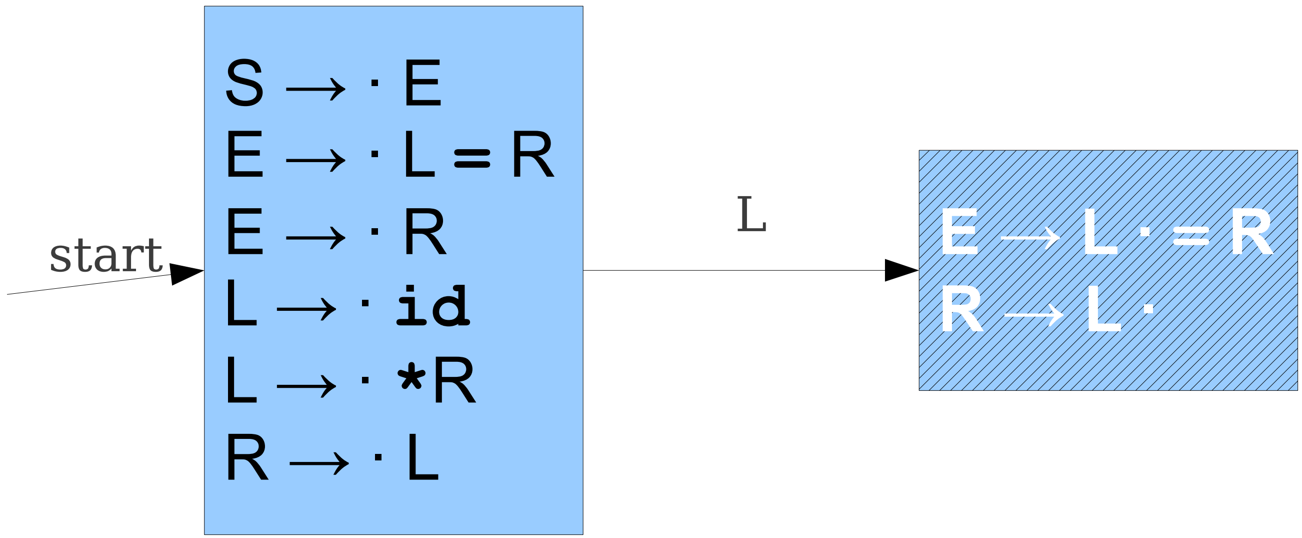
$E \rightarrow L \cdot = R$ tells us to shift on seeing $=$

$R \rightarrow L \cdot$ tells us to reduce on FOLLOW(R).

$= \in$ FOLLOW(R).

The Limits of SLR(1)

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



E → L · = R tells us to shift on seeing **=**
R → L · tells us to reduce on FOLLOW(**R**).

= ∈ FOLLOW(**R**).

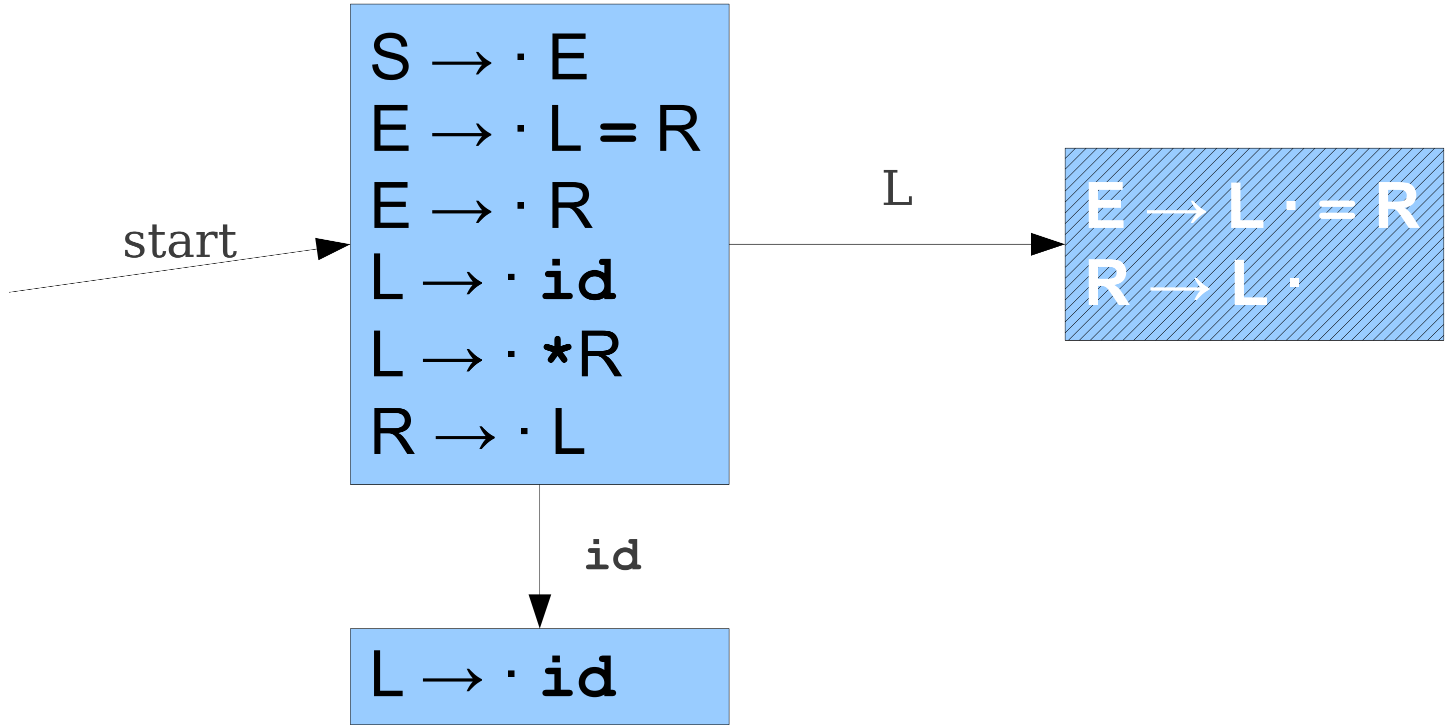
We have a conflict!

Why is SLR(1) Weak?

- With LR(1), incredible contextual information. 惊人的
  - Lookaheads at each state only possible after applying the productions that could get us there.
- With SLR(1), *minimal* context.
  - FOLLOW(**A**) means "what could follow **A** *somewhere* in the grammar?," even if in a particular state **A** couldn't possibly have that symbol after it.

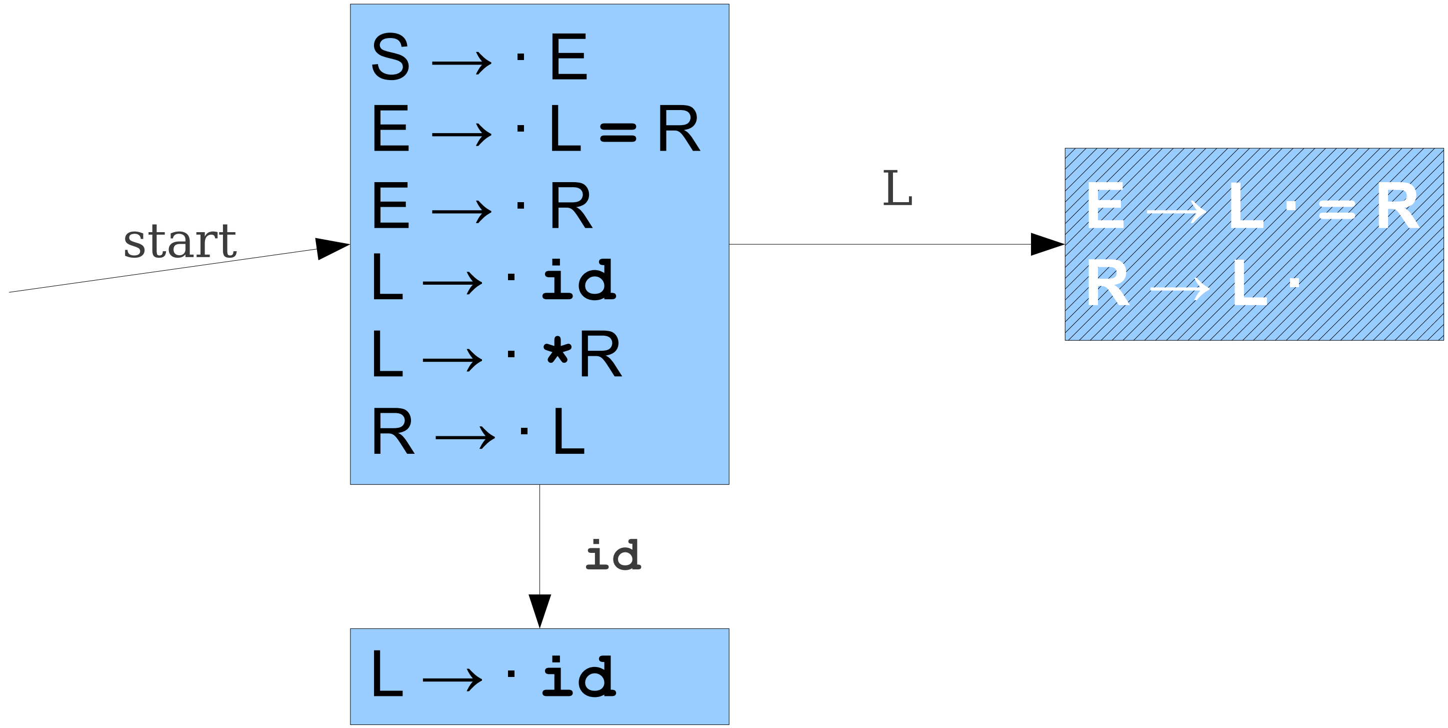
A Lack of Context

$S \to E$
$E \to L = R$
$E \to R$
$L \to \texttt{id}$
$L \to *R$
$R \to L$



A Lack of Context

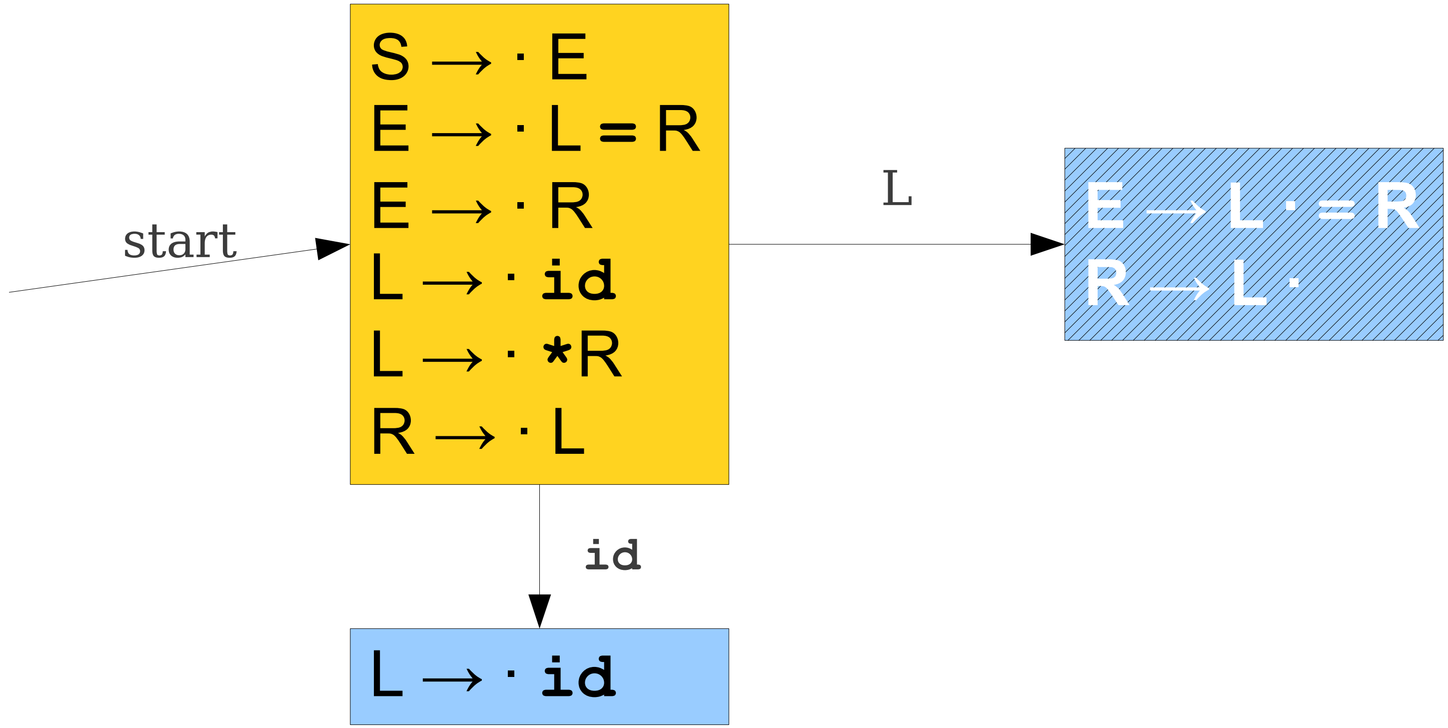
S → **E**
E → **L** `=` **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**



`id`	`=`	`*`	`id`

A Lack of Context

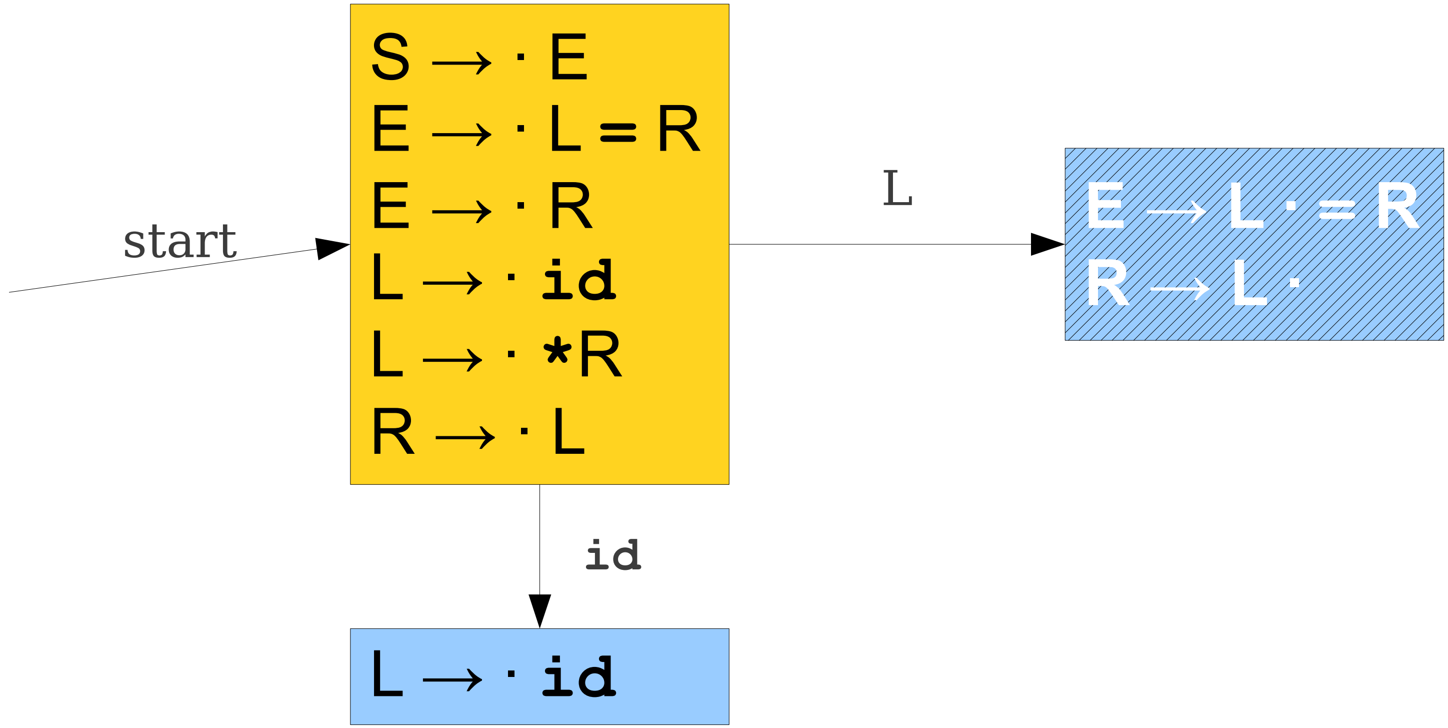
S → **E**
E → **L** `=` **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**

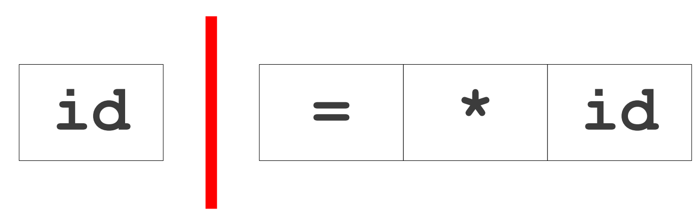


`id`	`=`	`*`	`id`

A Lack of Context

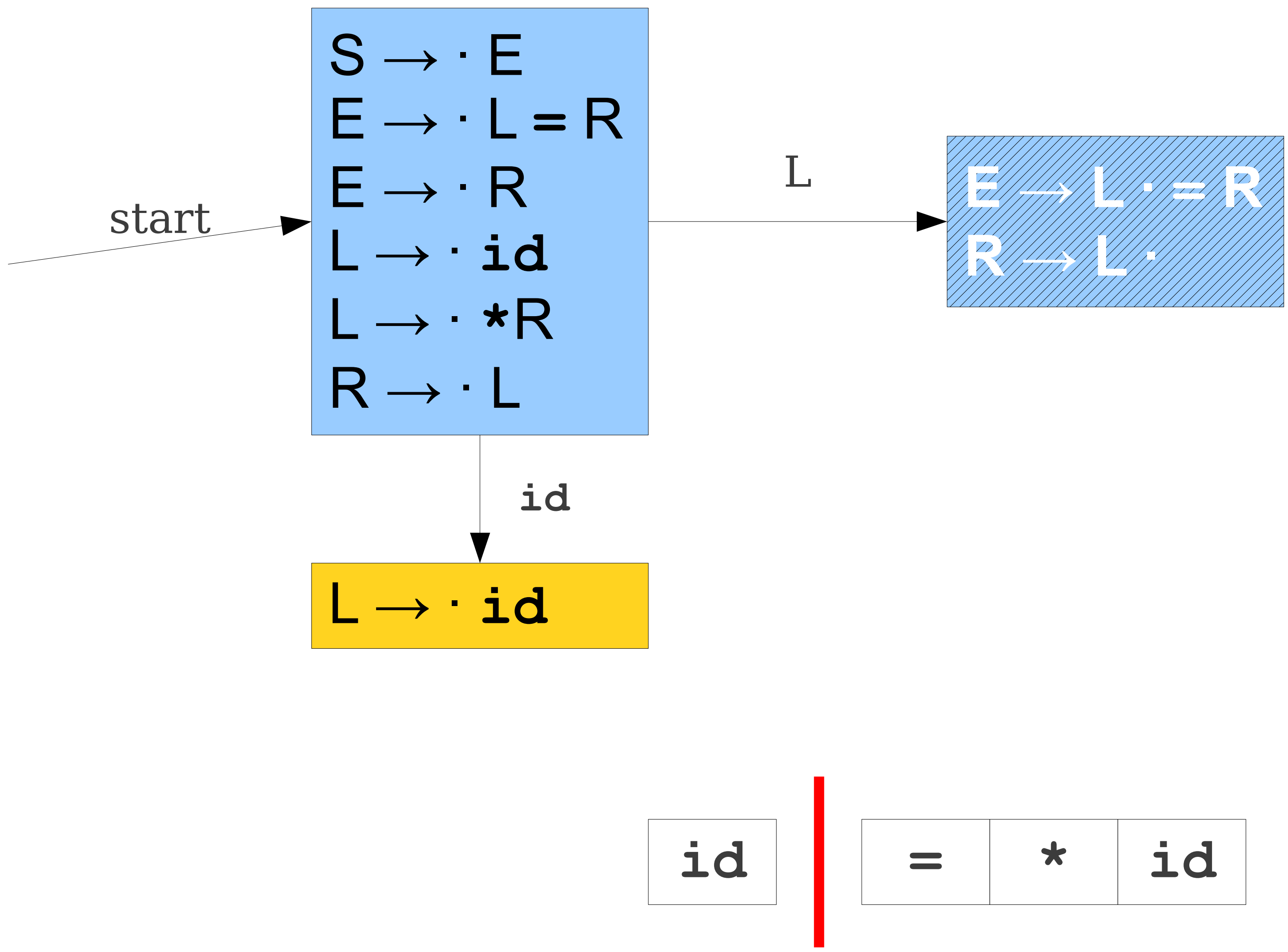
S → **E**
E → **L** `=` **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**





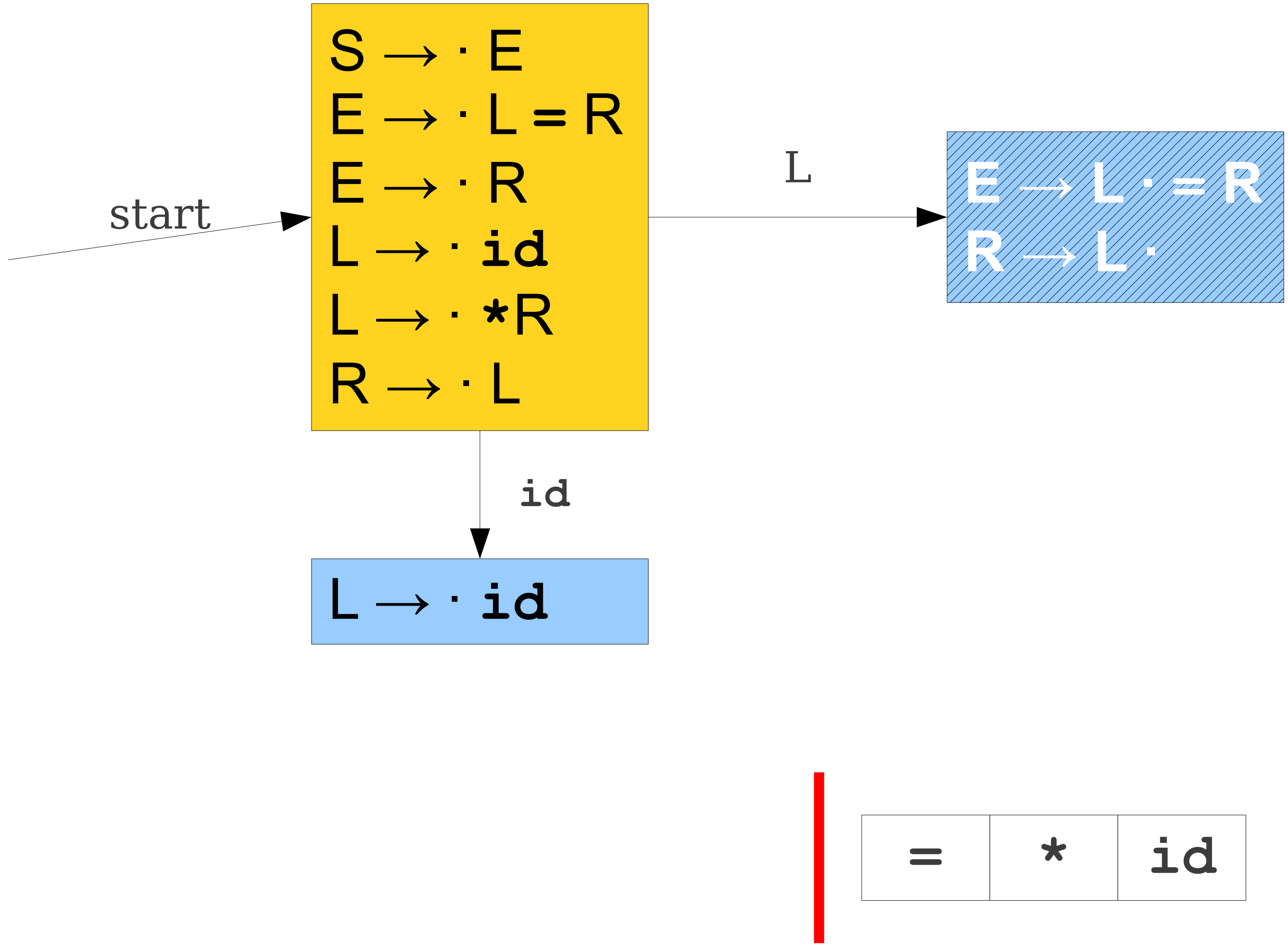
A Lack of Context

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



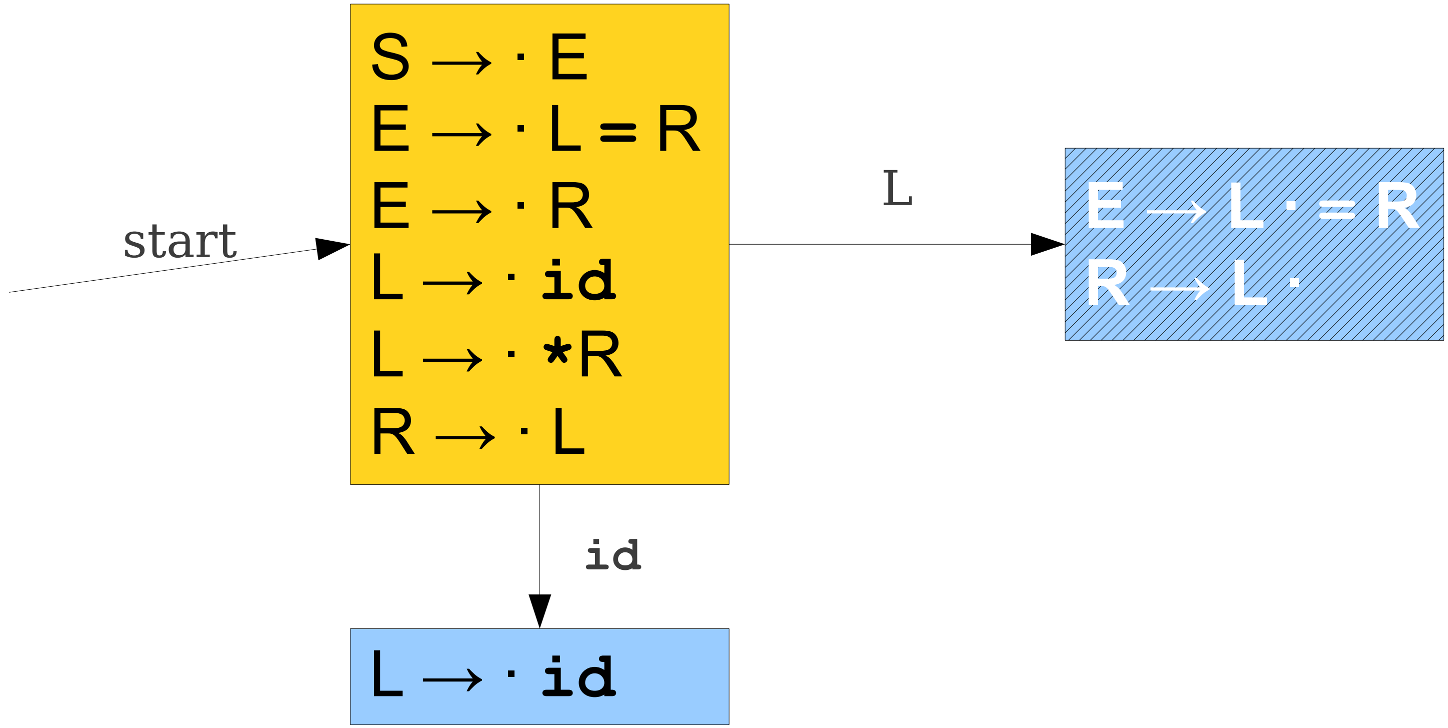
A Lack of Context

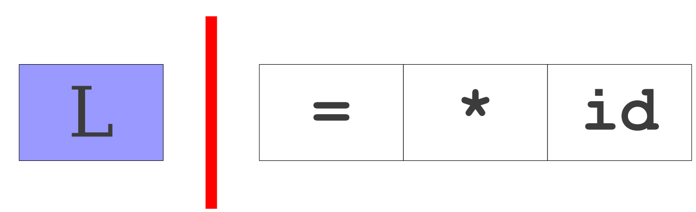
S → E
E → L = R
E → R
L → id
L → *R
R → L



A Lack of Context

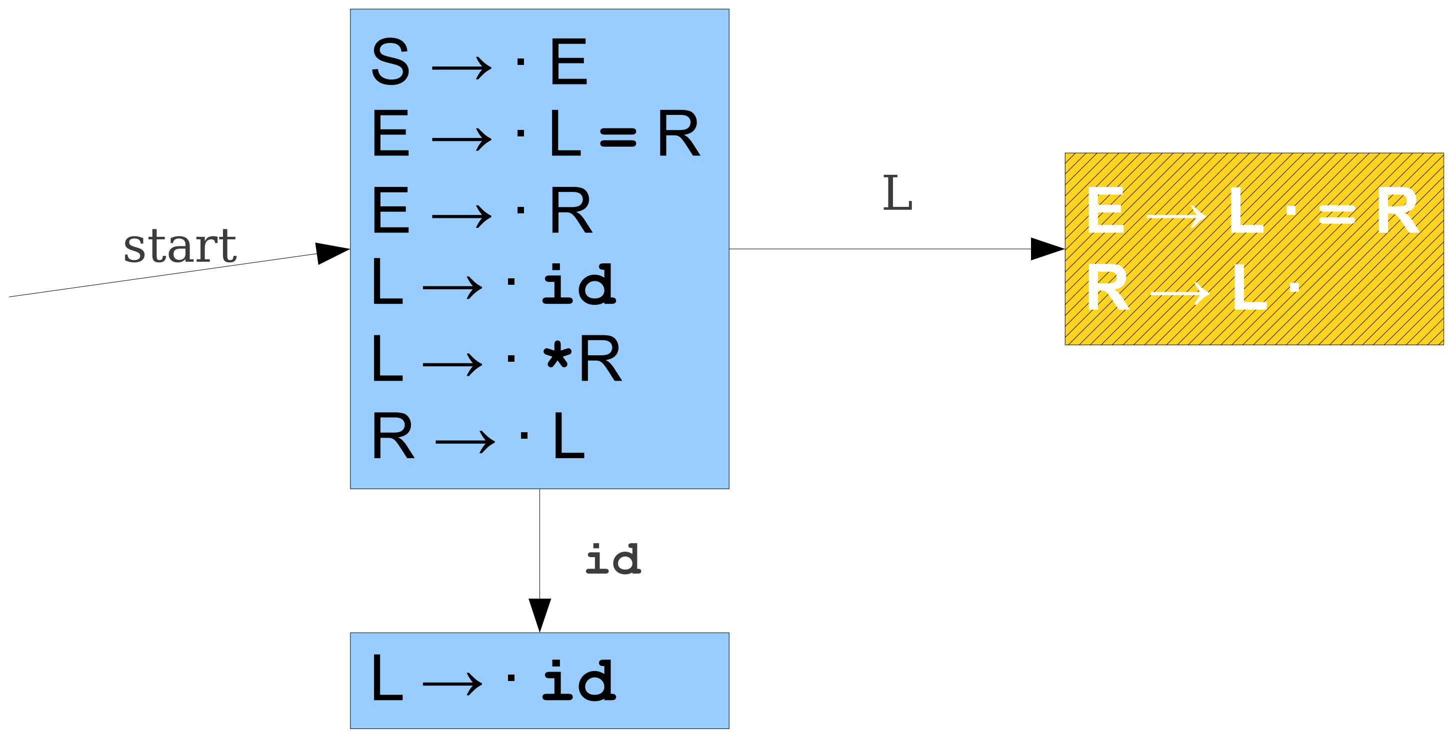
S → **E**
E → **L** `=` **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**

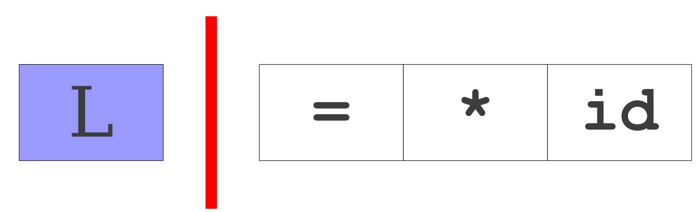




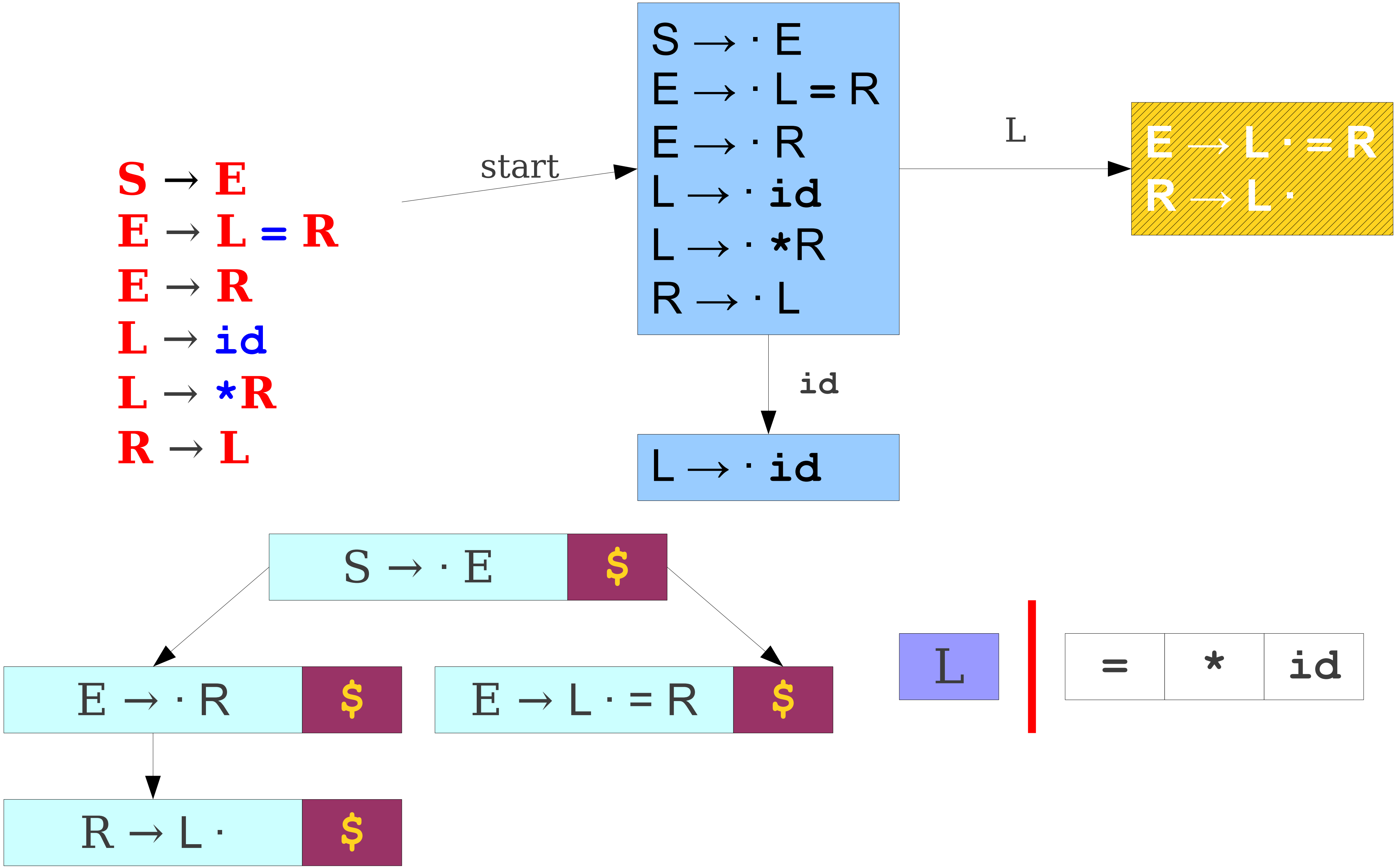
A Lack of Context

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



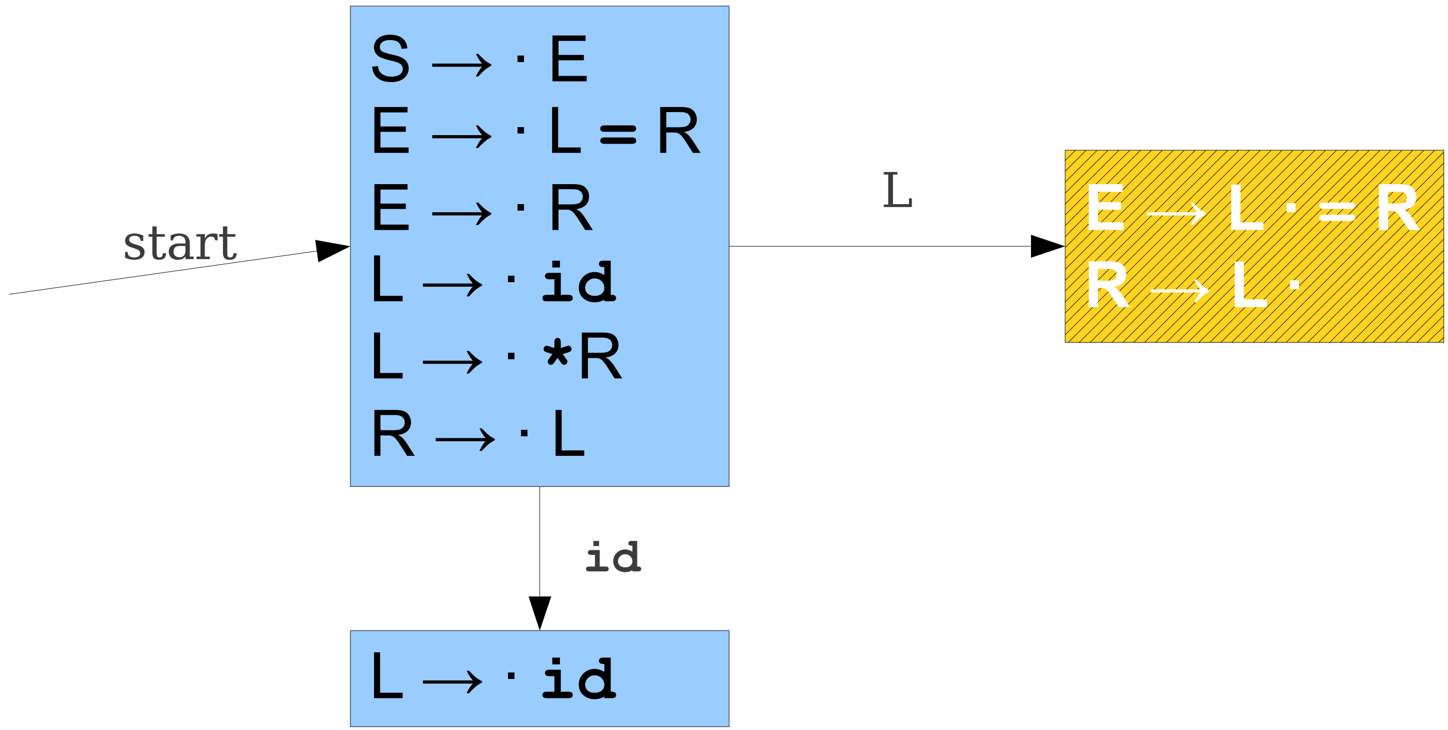


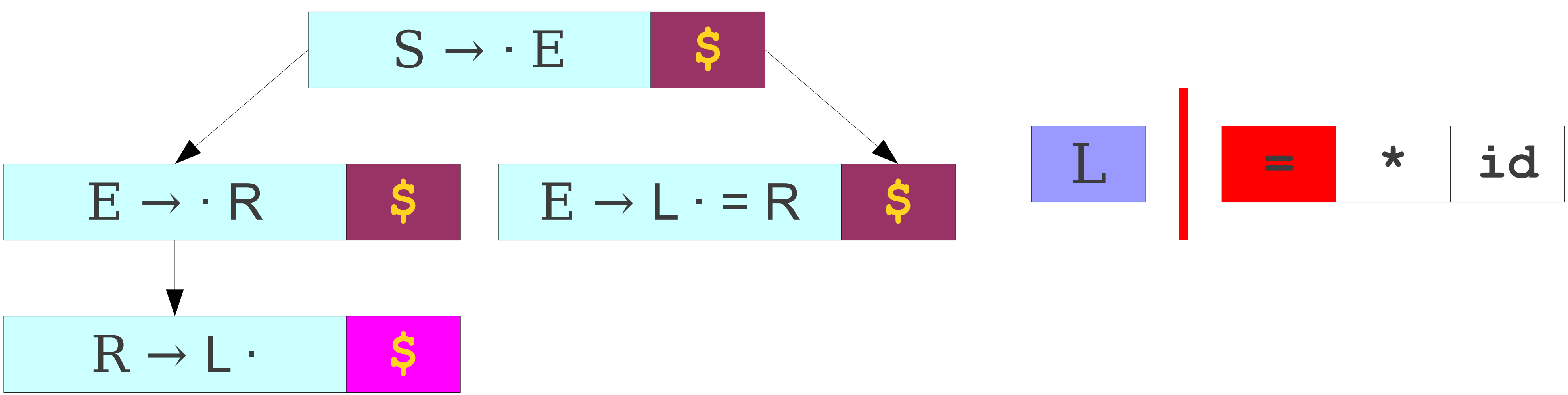
A Lack of Context



A Lack of Context

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



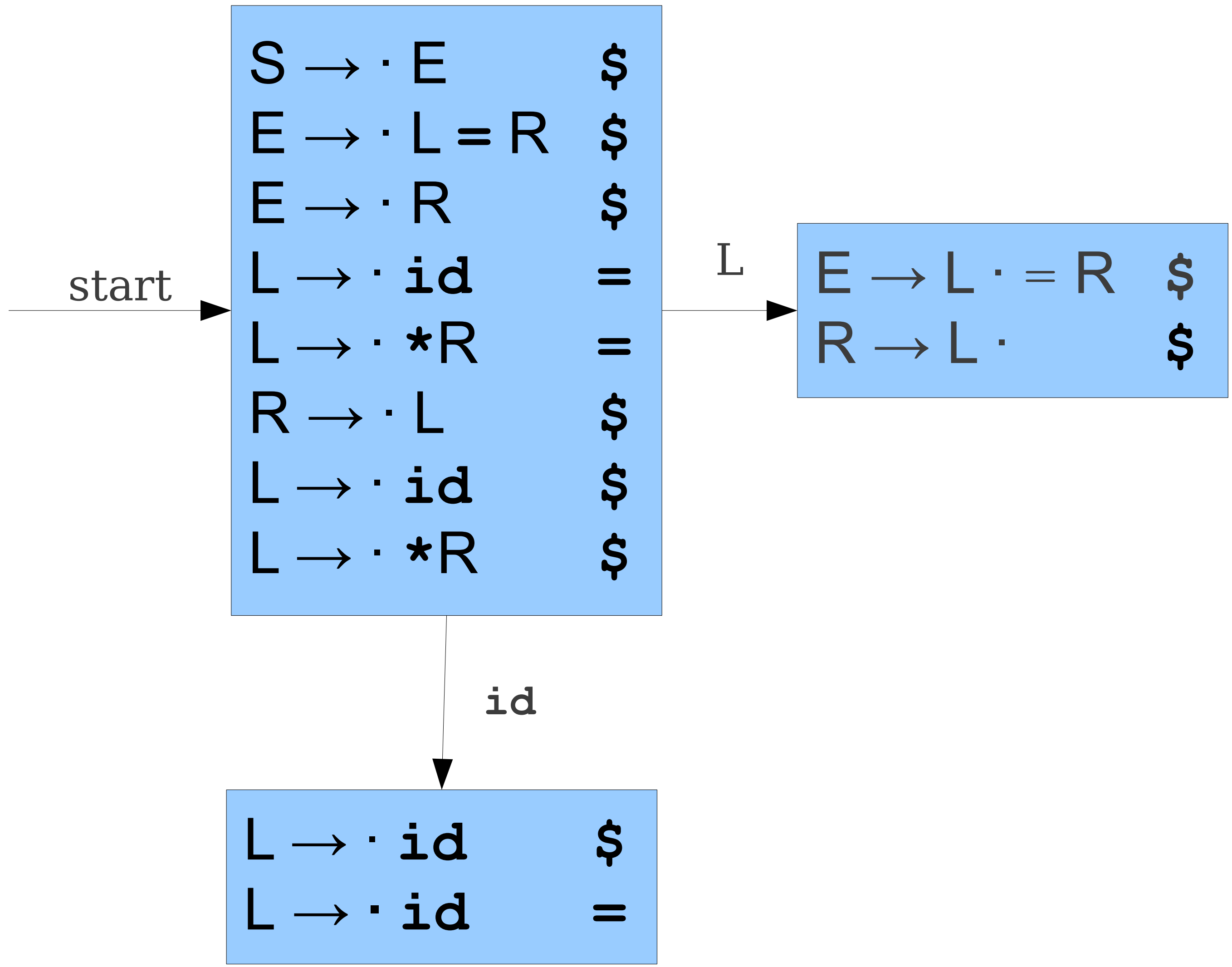


For Reference: LR(1) States

$S \to E$
$E \to L = R$
$E \to R$
$L \to id$
$L \to {*}R$
$R \to L$

For Reference: LR(1) States

$S \to E$
$E \to L = R$
$E \to R$
$L \to \texttt{id}$
$L \to {*}R$
$R \to L$



LR(1) and SLR(1)

- SLR(1) is weak because it has no contextual information.
- LR(1) is impractical because its contextual information makes the automaton too big.
- Can we retain the LR(1) automaton's contextual information without all its states?

Review of LR(1)

- Each state in an LR(1) automaton is a combination of an LR(0) state and lookahead information.
- Two LR(1) items have the same **core** if they are identical except for lookahead.

$T \to (\cdot E)$	**$**
$E \to \cdot E + T$	)
$E \to \cdot T$	)
$T \to \cdot$**int**	)
$T \to \cdot(E)$	)

$T \to (\cdot E)$	)
$E \to \cdot E + T$	)
$E \to \cdot T$	)
$T \to \cdot$**int**	)
$T \to \cdot(E)$	)

A Surprisingly Powerful Idea

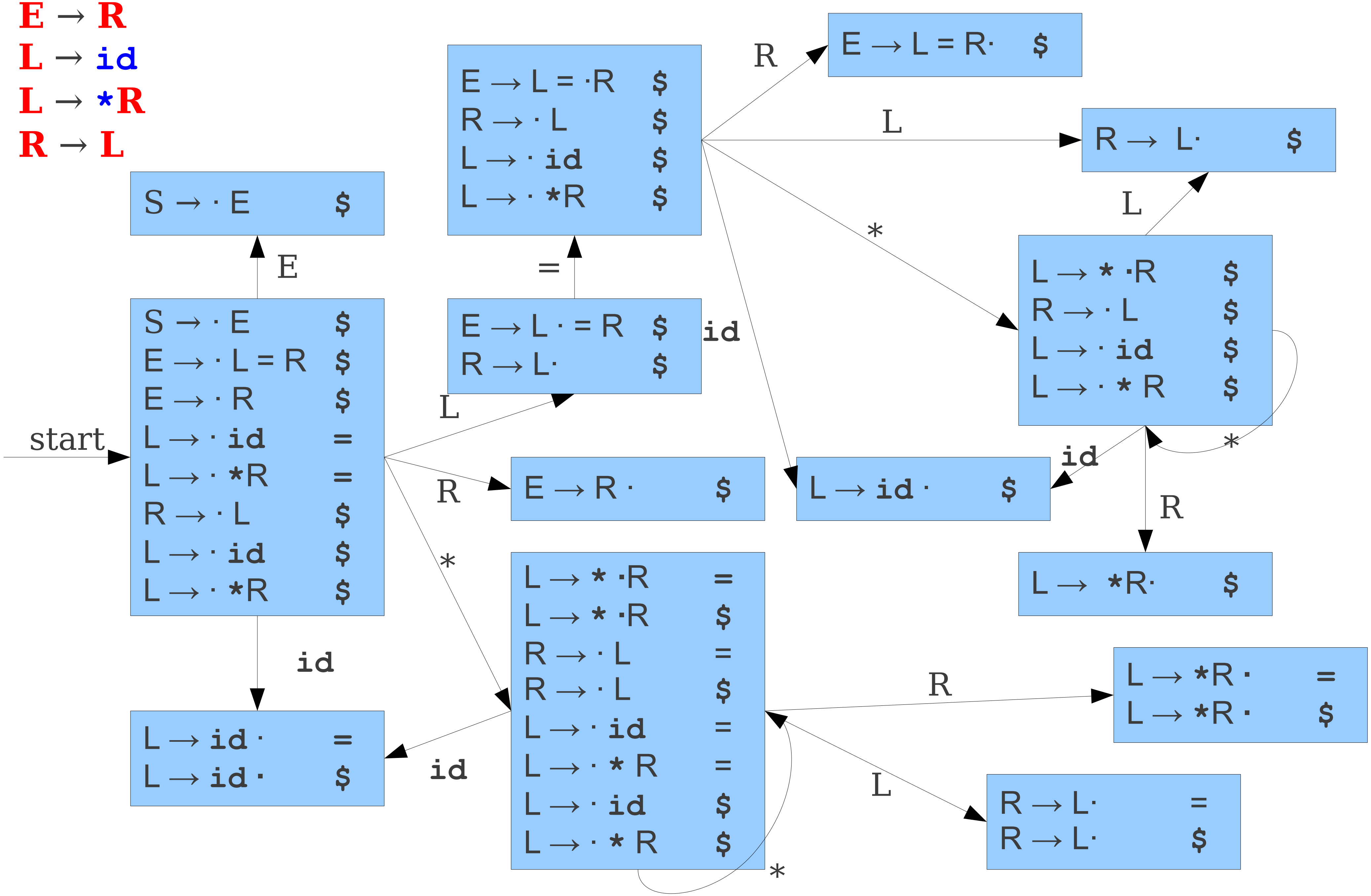
- In an LR(1) automaton, we have multiple states with the same core but different lookahead.
- What if we merge all these states together?
- This is called **LALR(1)**
  - **Lookahead(1) LR(0)**

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

From LR(1) to LALR(1)

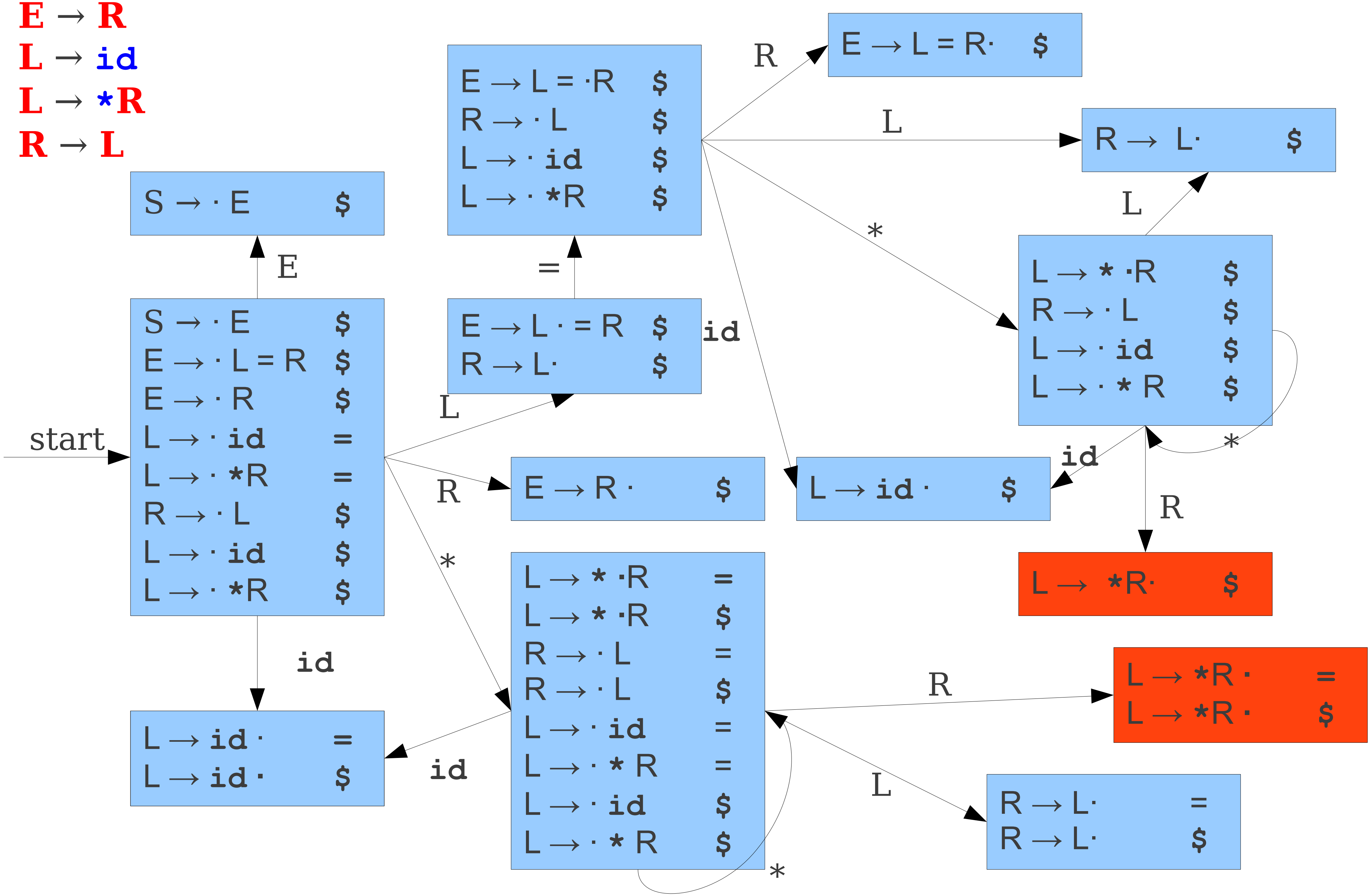
From LR(1) to LALR(1)

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



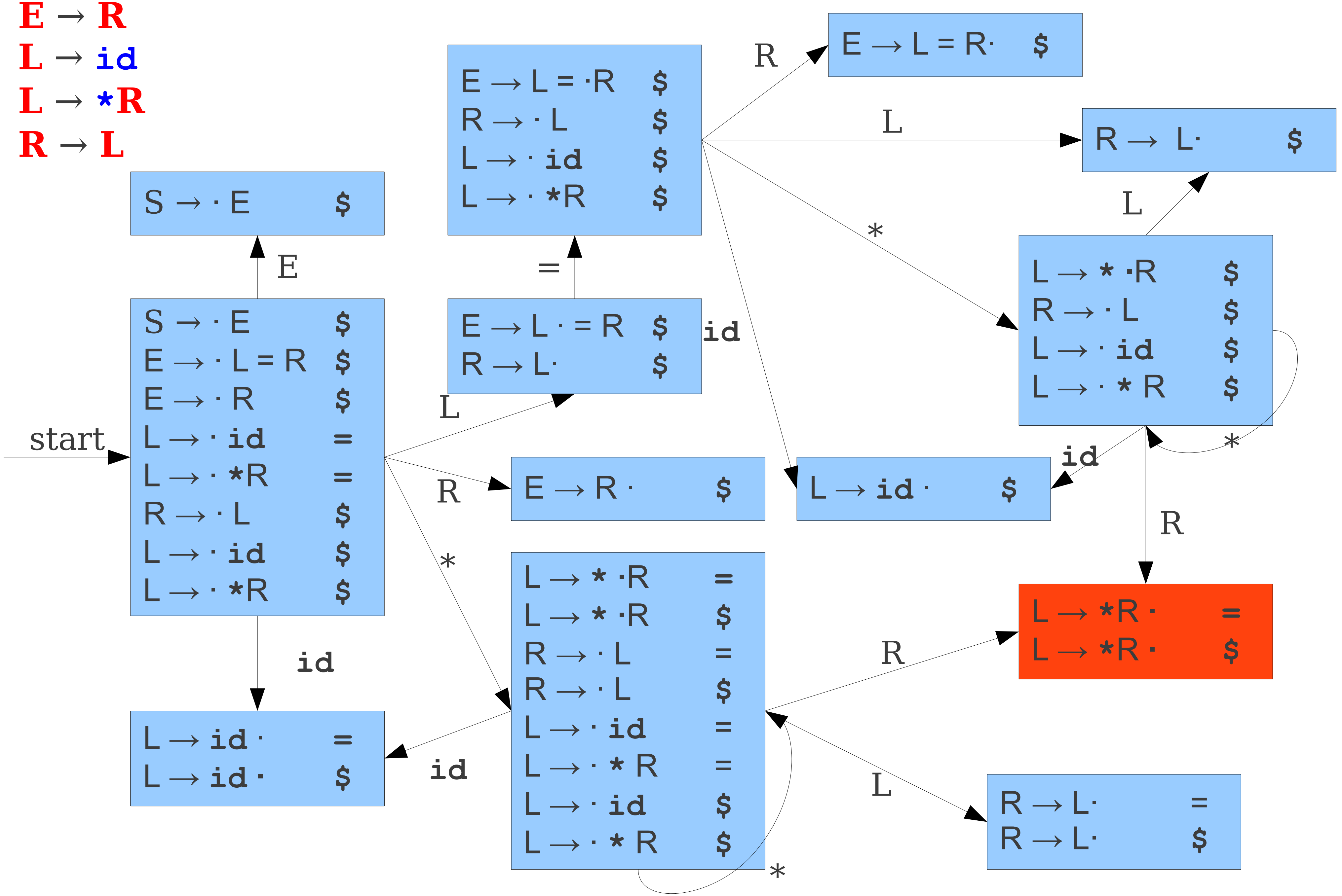
From LR(1) to LALR(1)

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



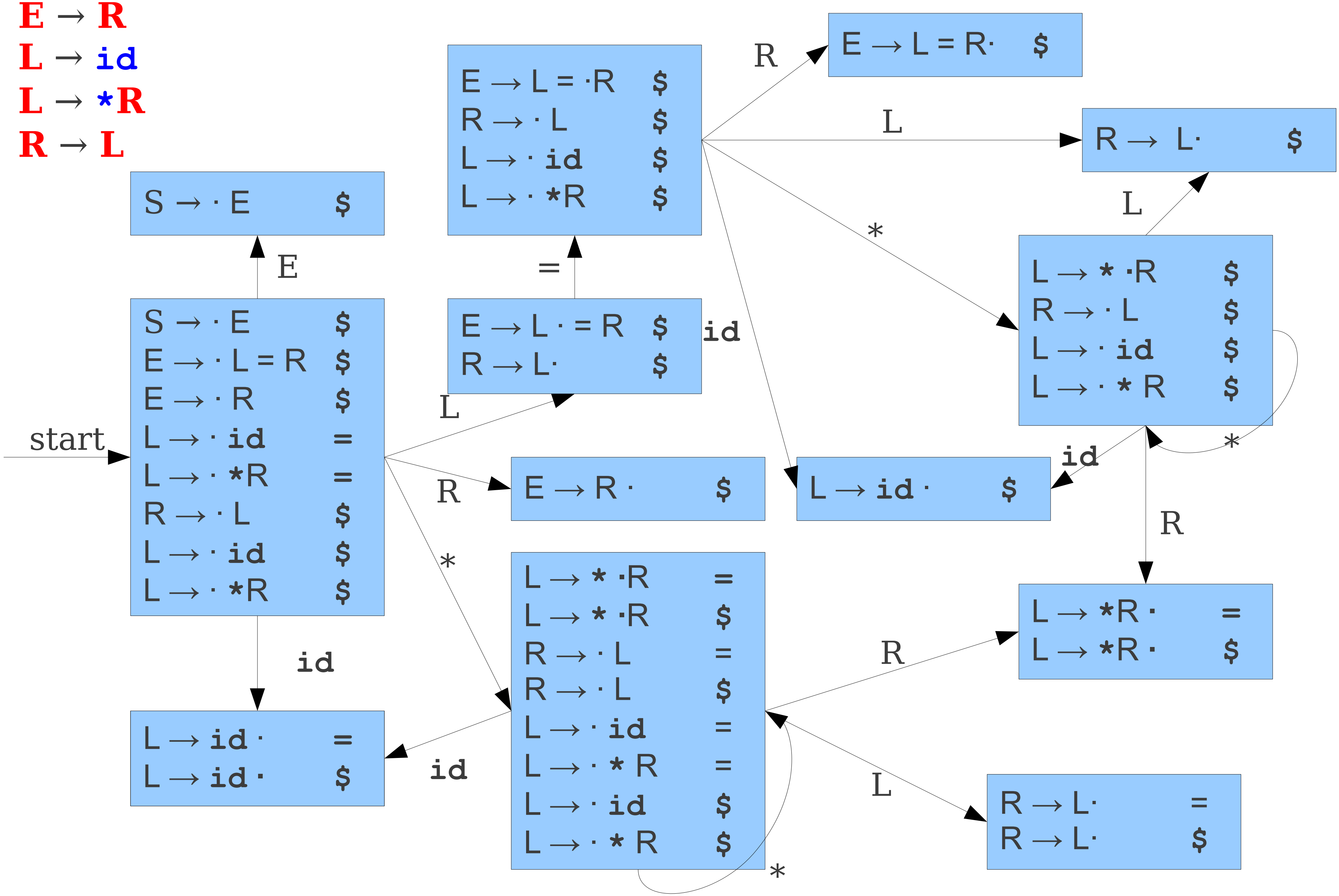
S → E
E → L = R
E → R
L → id
L → *R
R → L

From LR(1) to LALR(1)



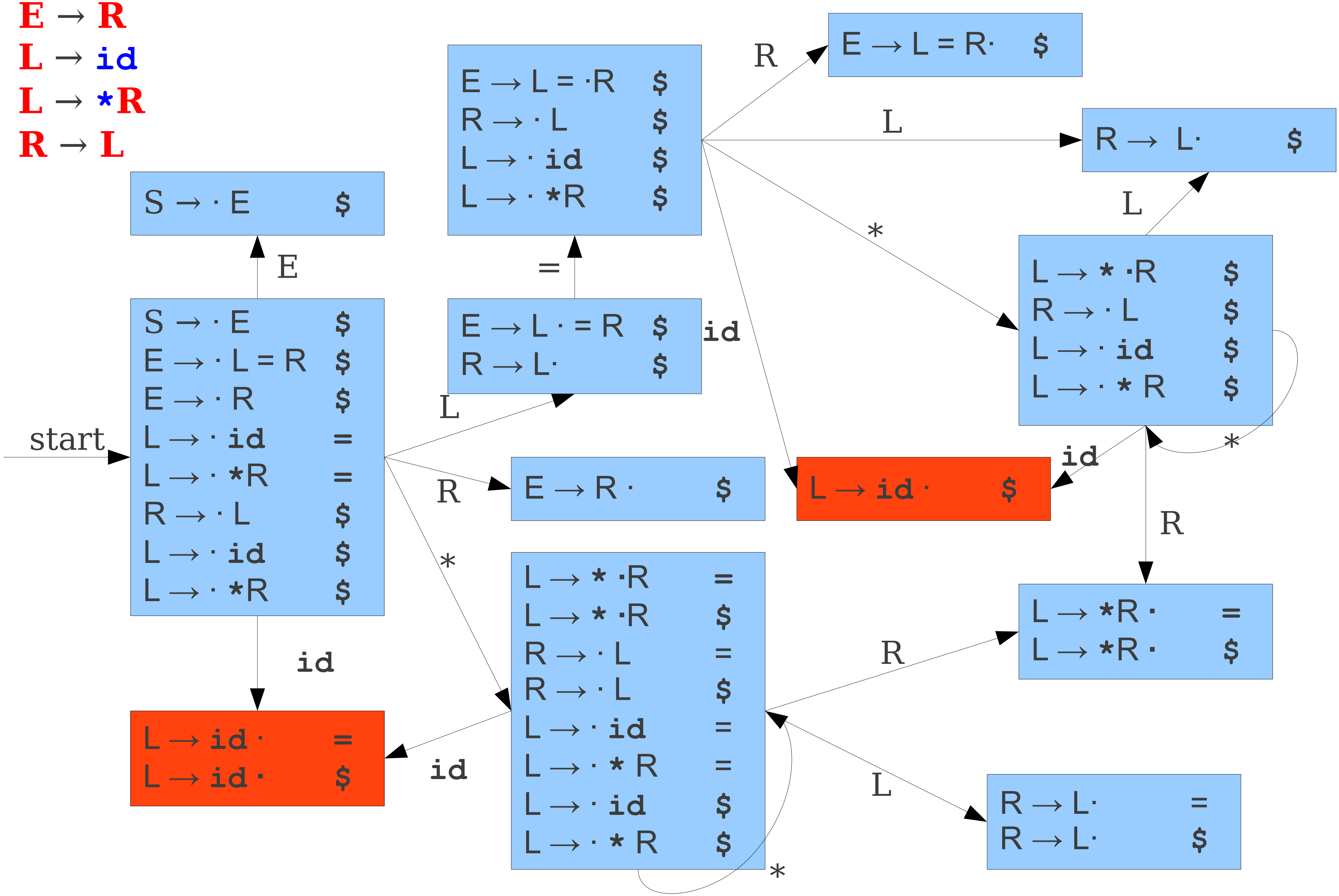
From LR(1) to LALR(1)

$S \to E$
$E \to L = R$
$E \to R$
$L \to$ **id**
$L \to$ ***R**
$R \to L$



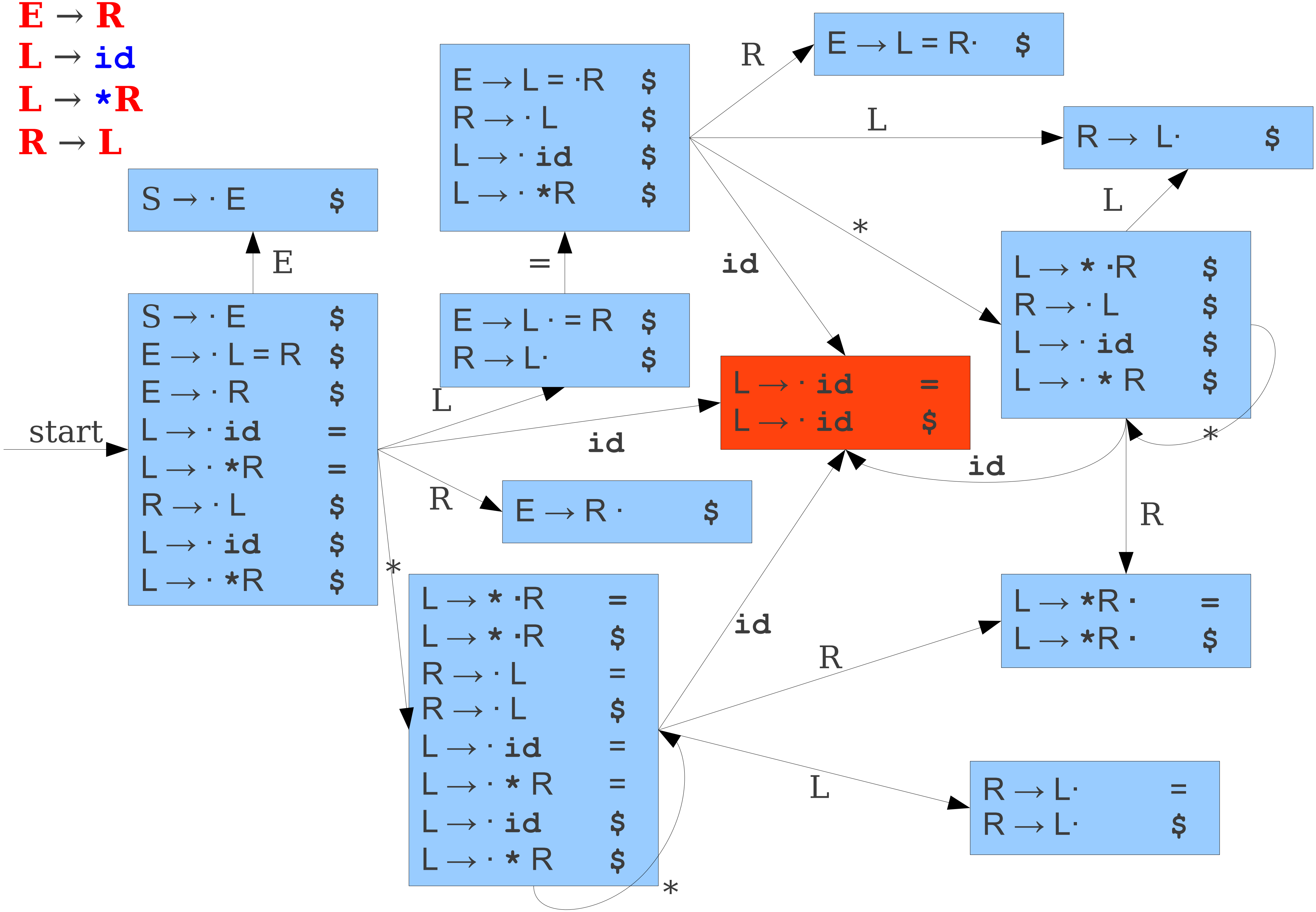
$S \to E$
$E \to L = R$
$E \to R$
$L \to \mathbf{id}$
$L \to *R$
$R \to L$

From LR(1) to LALR(1)



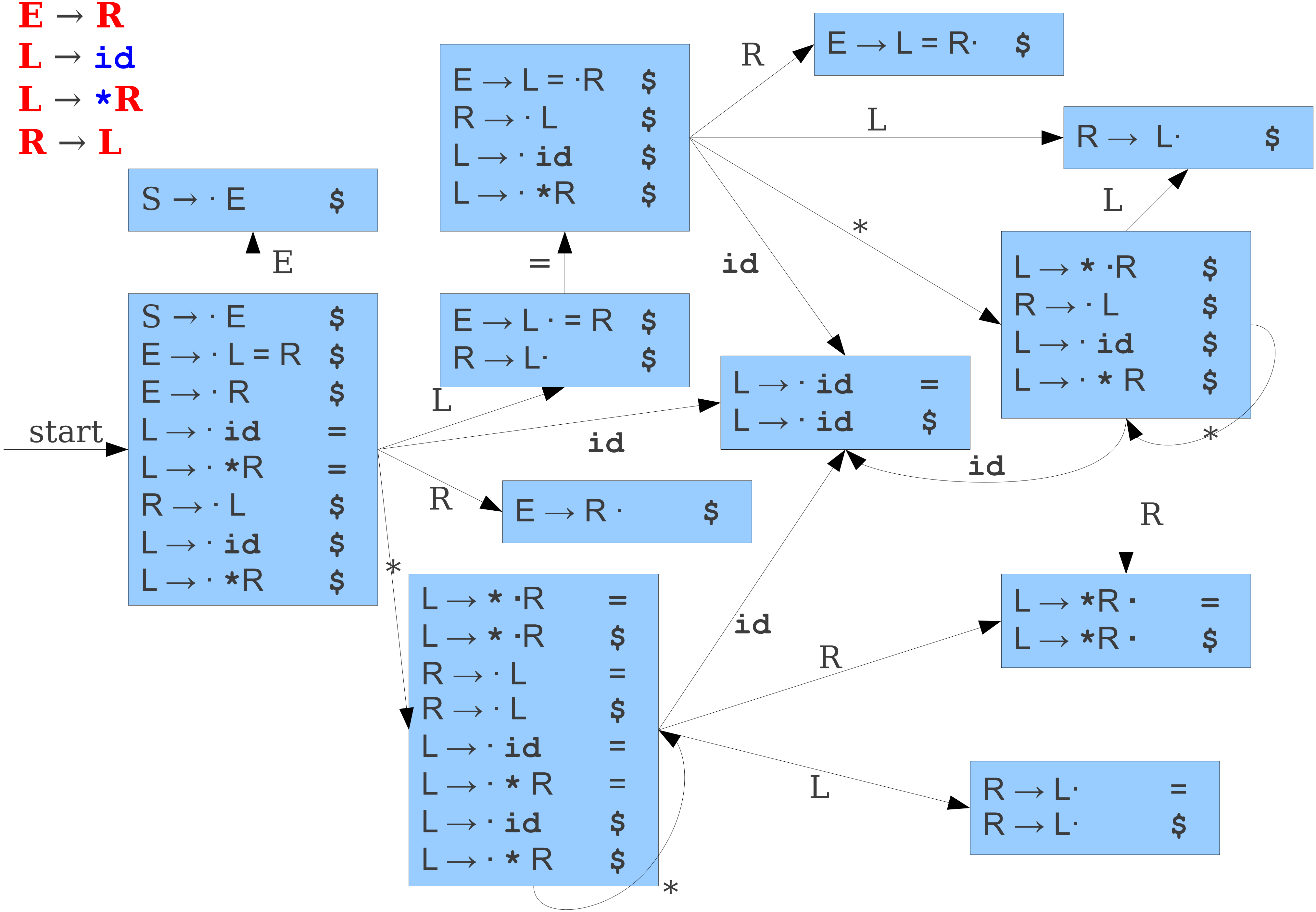
From LR(1) to LALR(1)

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



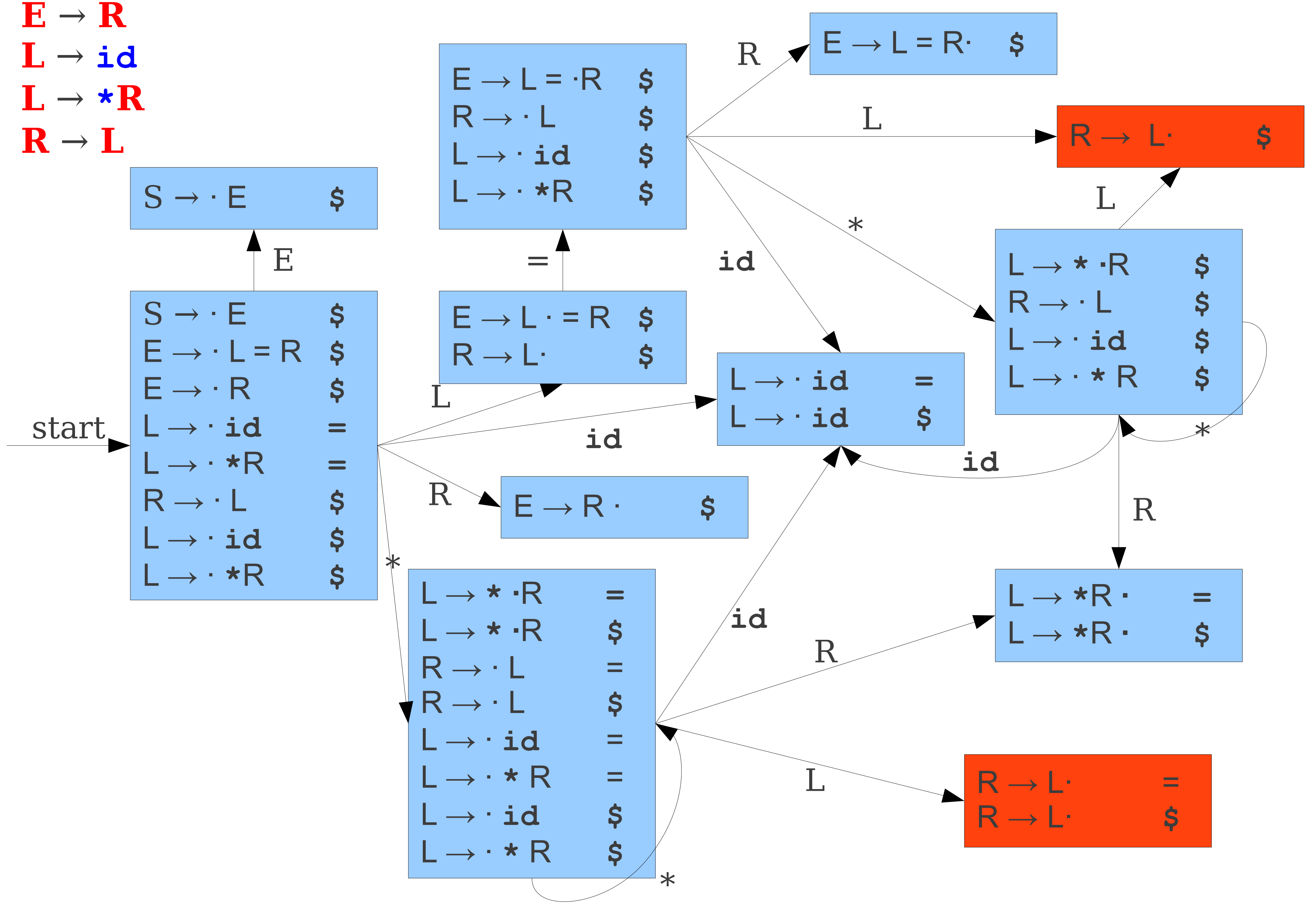
From LR(1) to LALR(1)

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



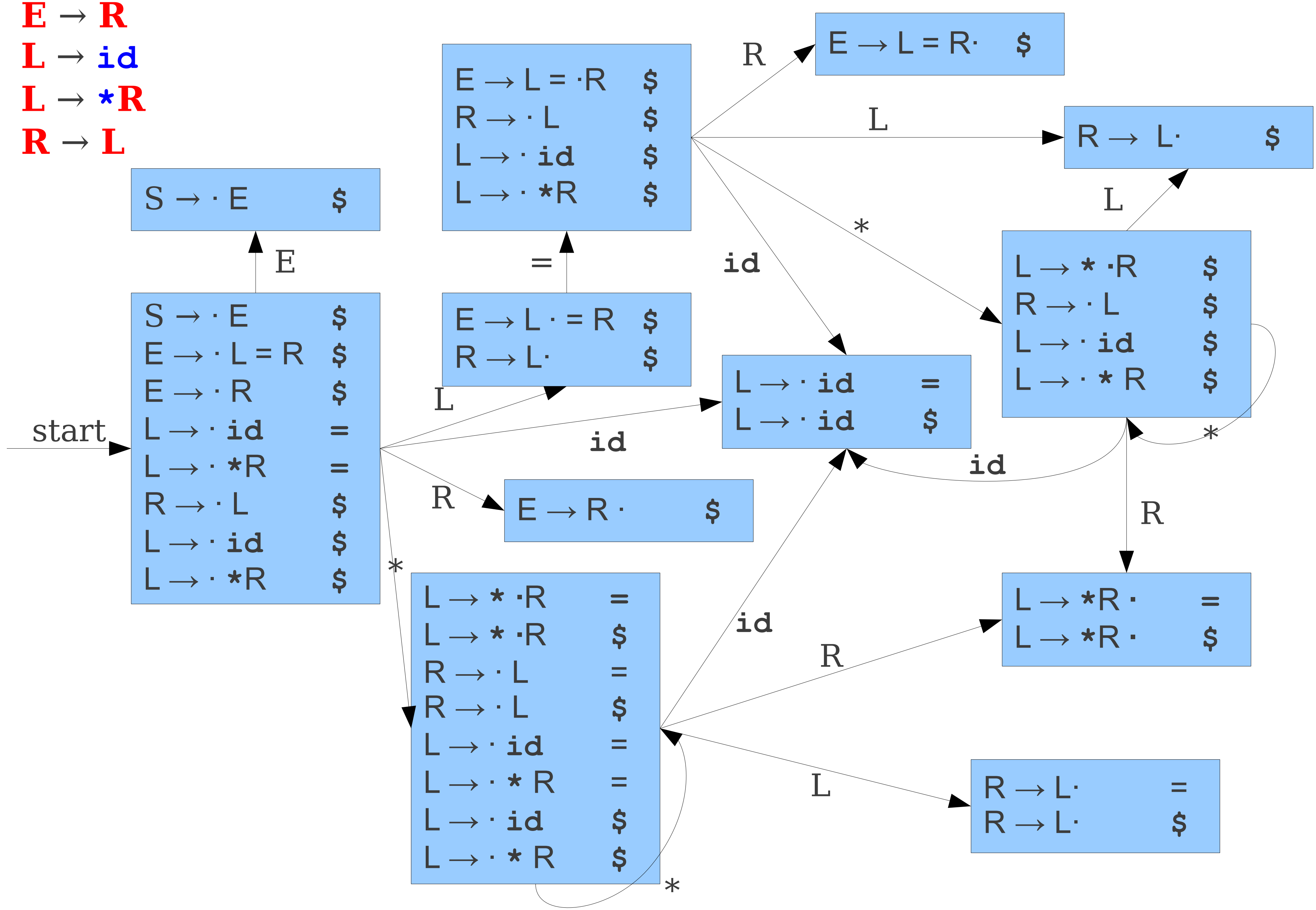
From LR(1) to LALR(1)

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



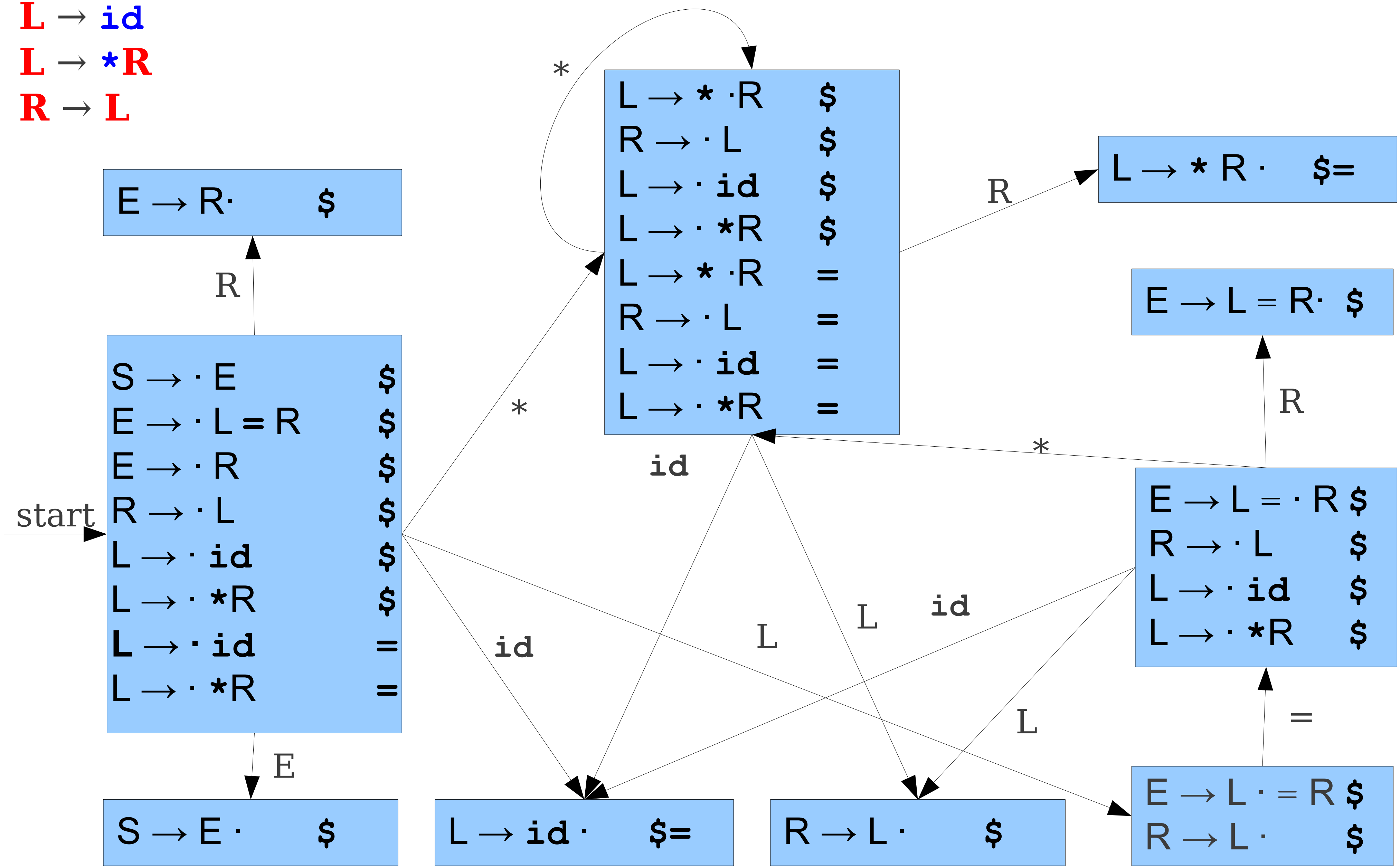
From LR(1) to LALR(1)

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



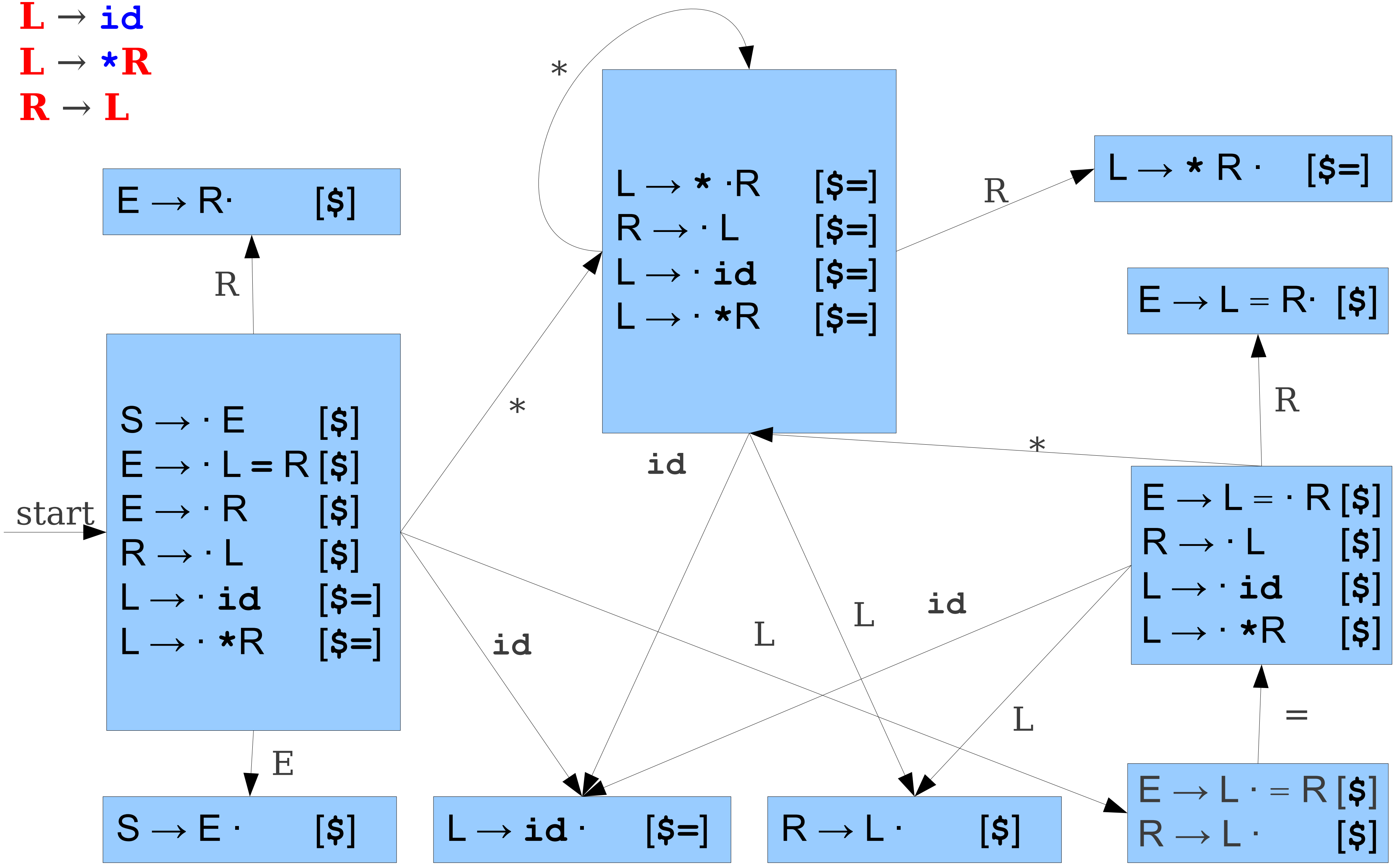
From LR(1) to LALR(1)

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



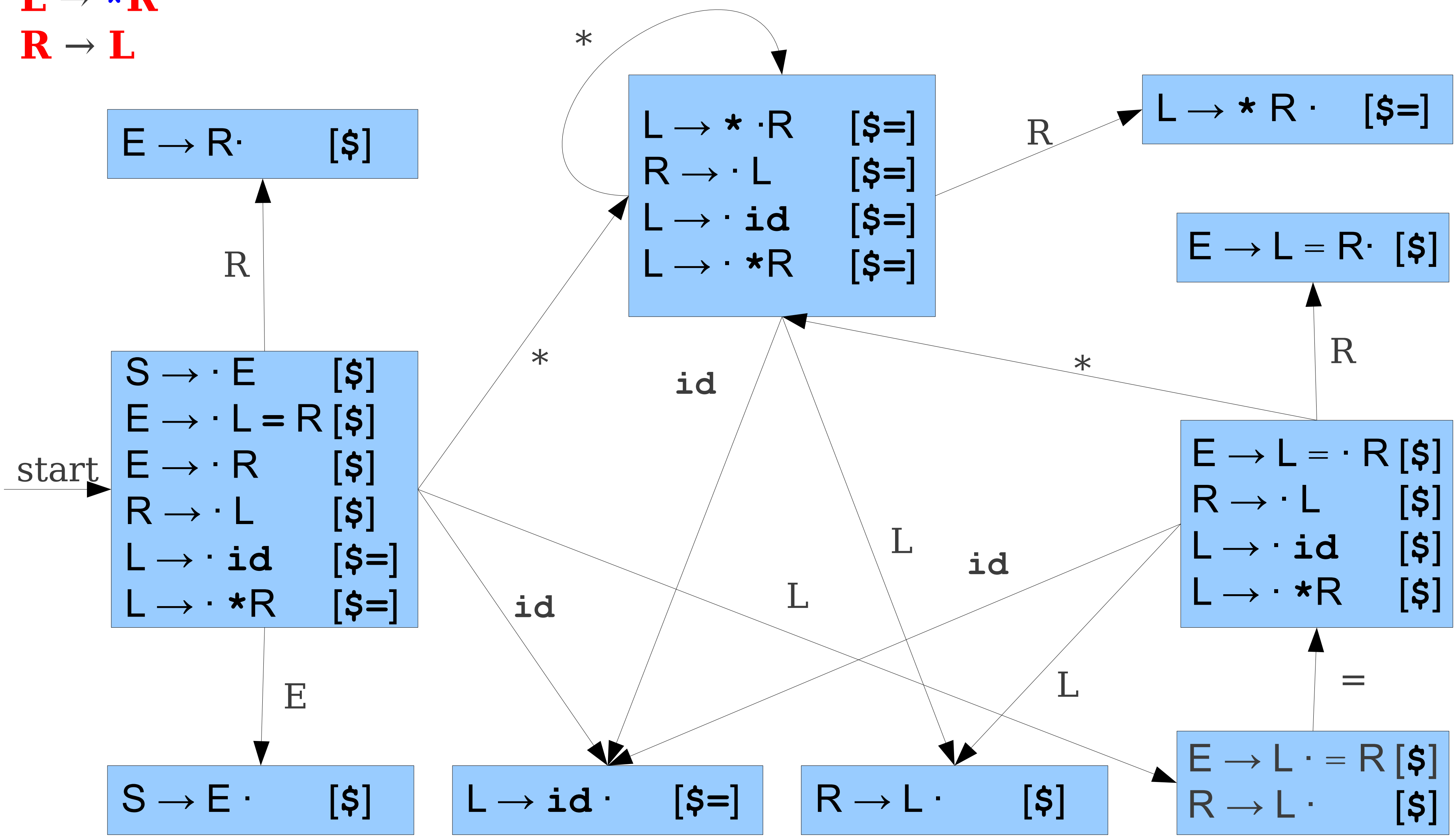
From LR(1) to LALR(1)

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



From LR(1) to LALR(1)

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



Advantages of LALR(1)

- Maintains context.
  - Lookup sets based on the fine-grained LR(1) automaton.
  - Each state's lookup relevant only for that state.
- Keeps automaton small.
  - Resulting automaton has same size as LR(0) automaton.

LALR(1) is Powerful

- Every LR(0) grammar is LALR(1).
- Every SLR(1) grammar is LALR(1)
- *Most* (but not all) LR(1) grammars are LALR(1).

LALR(1) isn't LR(1)

- Merging LR(1) states cannot introduce a shift/reduce conflict.
- **Why?**
- Since the items have the same core, a shift/reduce conflict in a LALR(1) state would have to also exist in one of the LR(1) states it was merged from.
- Merging LR(1) states **can** introduce a reduce/reduce conflict.
- Often these conflicts appear without any good reason; this is one limitation of LALR(1).

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

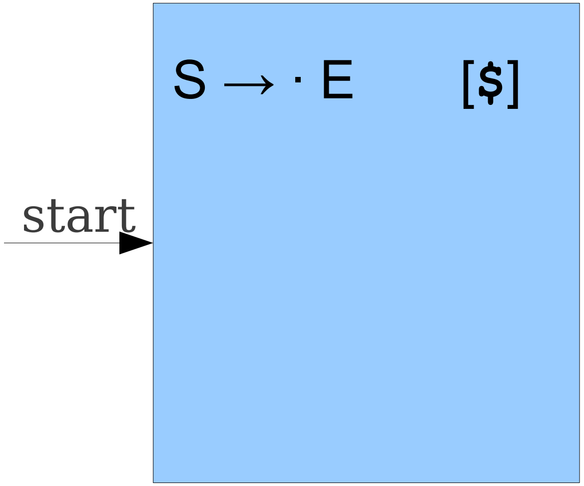
- Idea: Merge together LR(1) states as they're generated.
- Maintain a worklist of states to process; begin with the initial LR(1) state.
- When adding a new state, if it has the same core as an old state, update the old state and put it back in the worklist.

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

LALR(1) Construction

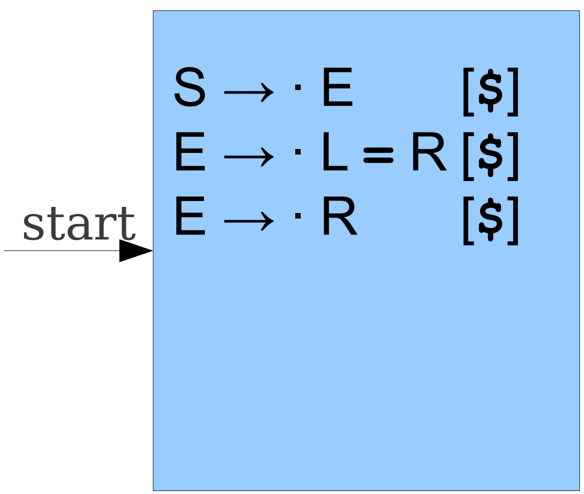
S → **E**
E → **L** `=` **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**

LALR(1) Construction



$S \to E$
$E \to L = R$
$E \to R$
$L \to id$
$L \to *R$
$R \to L$

LALR(1) Construction



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

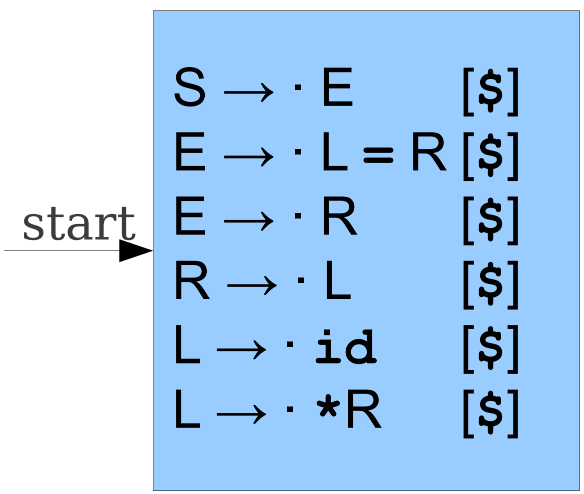
LALR(1) Construction

start

$S \rightarrow \cdot\, E \quad [\$]$
$E \rightarrow \cdot\, L = R \quad [\$]$
$E \rightarrow \cdot\, R \quad [\$]$
$R \rightarrow \cdot\, L \quad [\$]$

$S \to E$
$E \to L$ **=** R
$E \to R$
$L \to$ **id**
$L \to$ *****R
$R \to L$

LALR(1) Construction



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

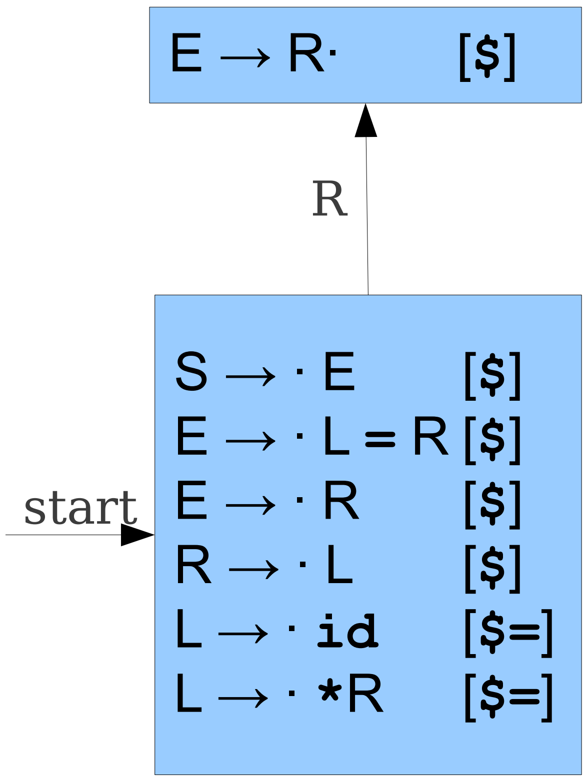
LALR(1) Construction

start

S → · E [$]
E → · L = R [$]
E → · R [$]
R → · L [$]
L → · id [$=]
L → · *R [$=]

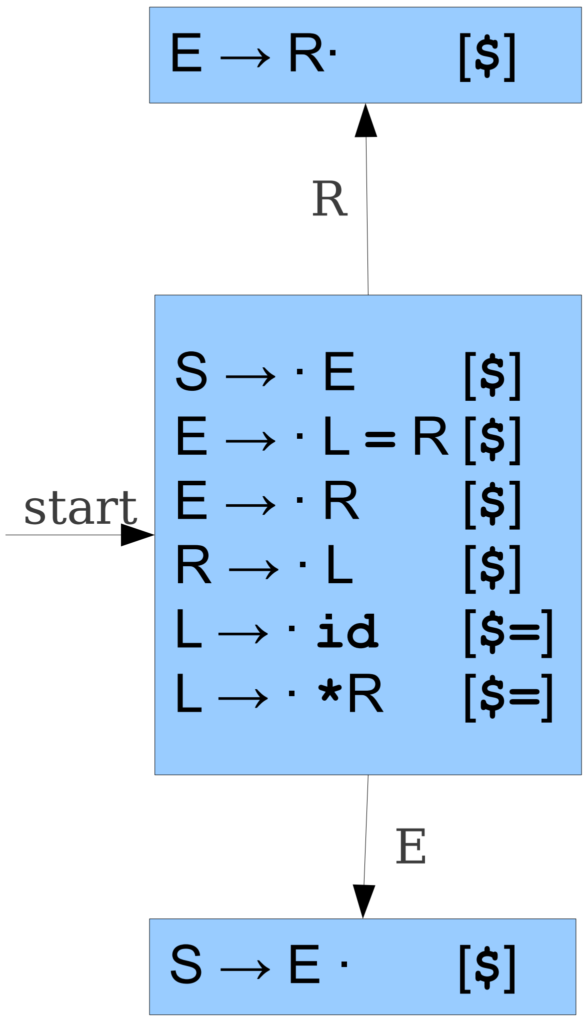
S → E
E → L = R
E → R
L → id
L → *R
R → L

LALR(1) Construction



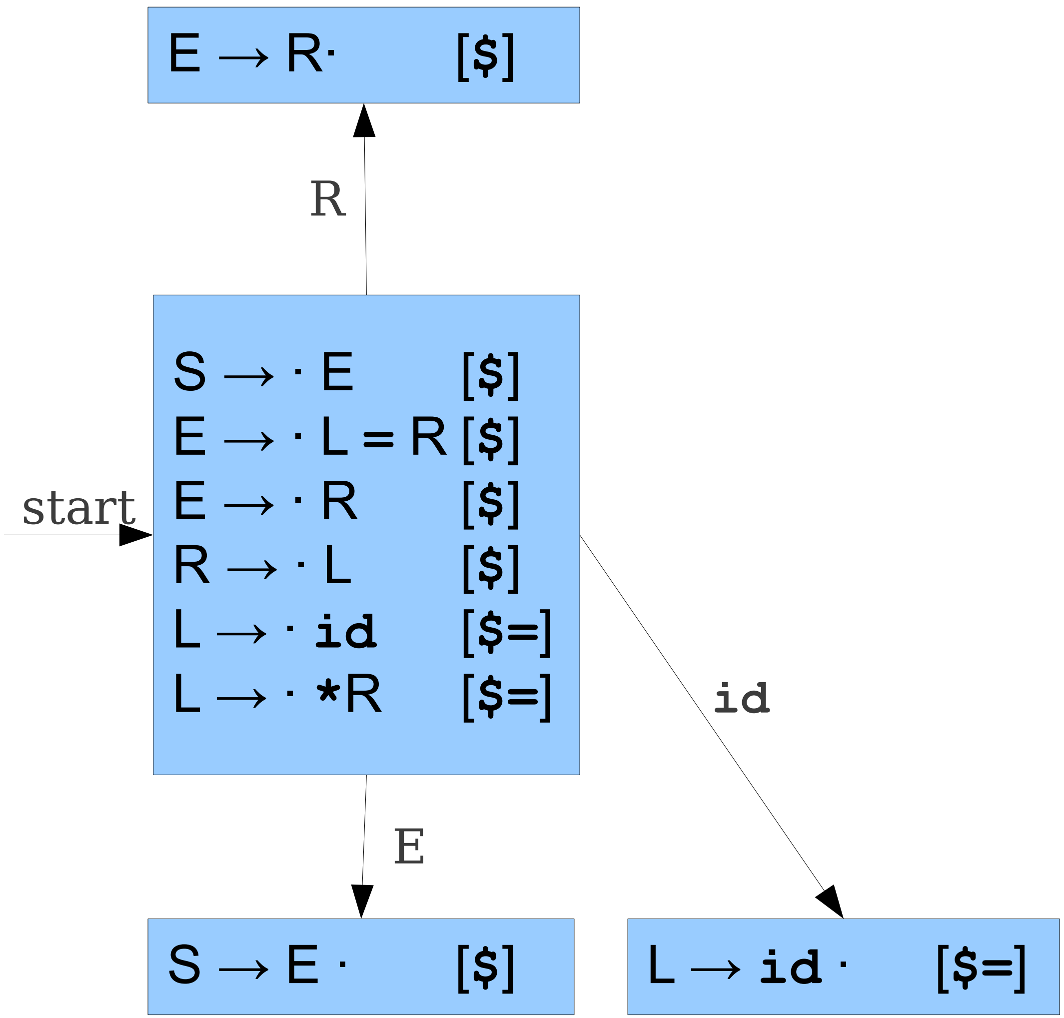
LALR(1) Construction

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



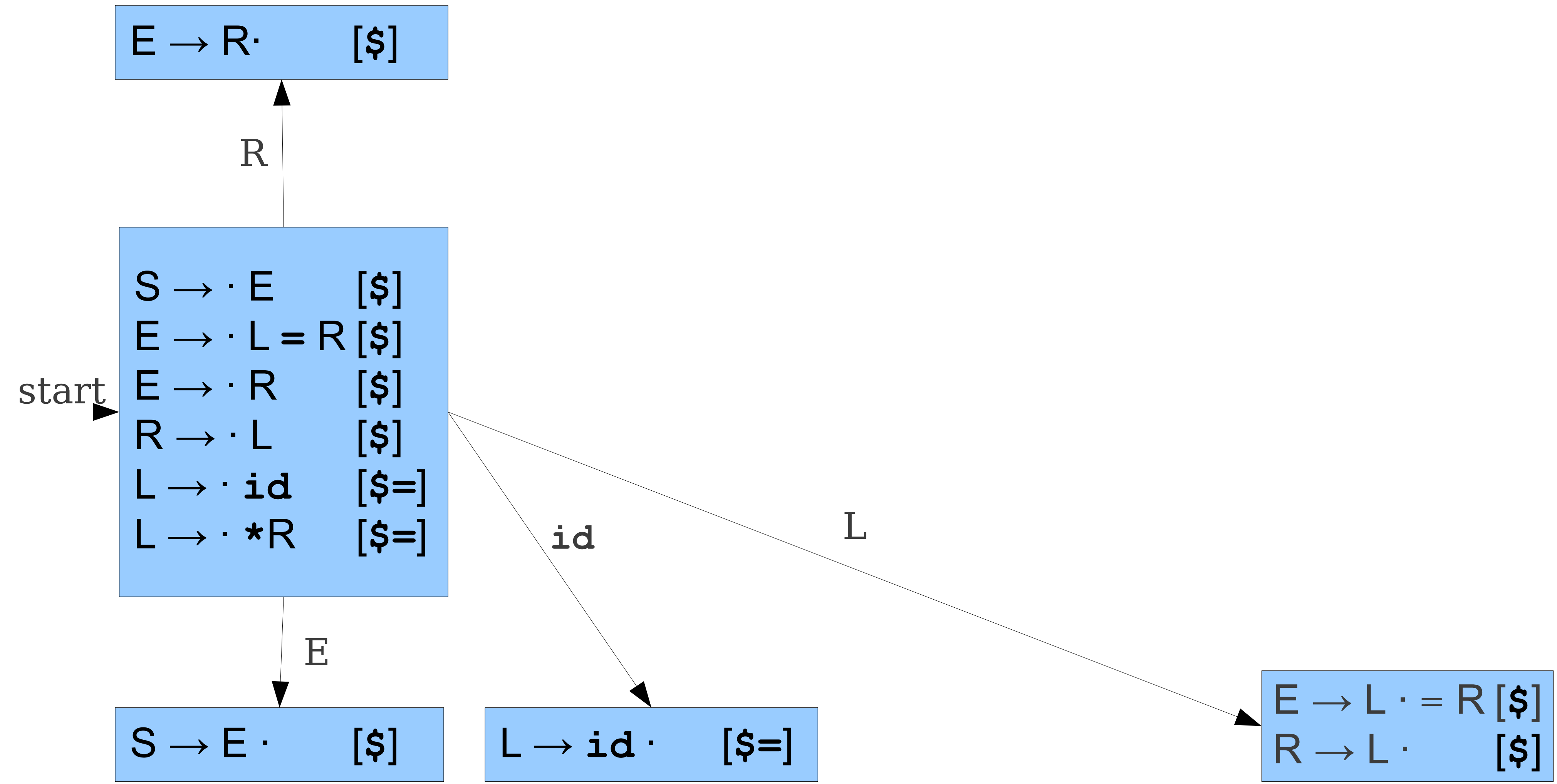
S → **E**
E → **L** `=` **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**

LALR(1) Construction



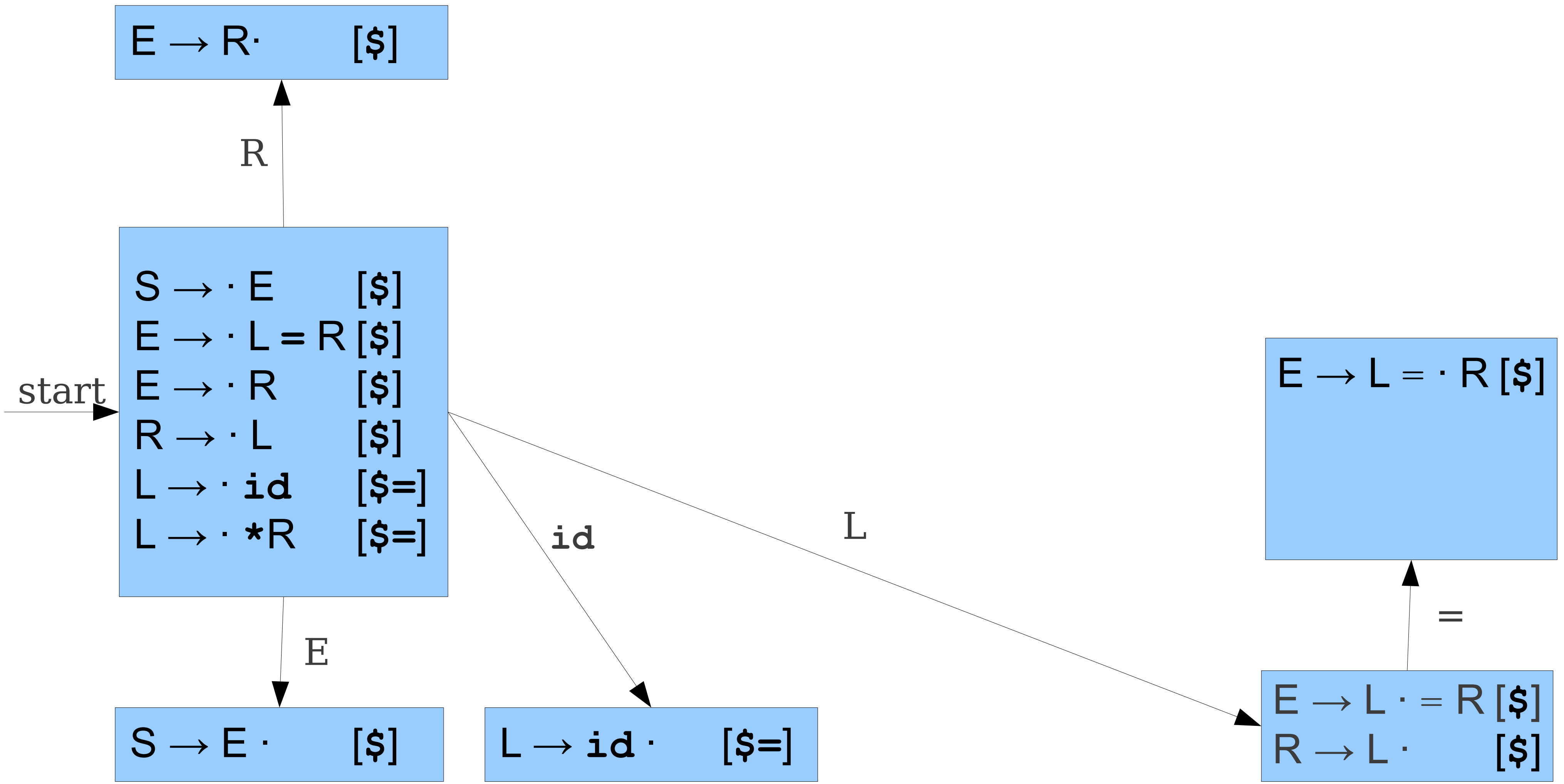
$S \to E$
$E \to L = R$
$E \to R$
$L \to \mathbf{id}$
$L \to *R$
$R \to L$

LALR(1) Construction



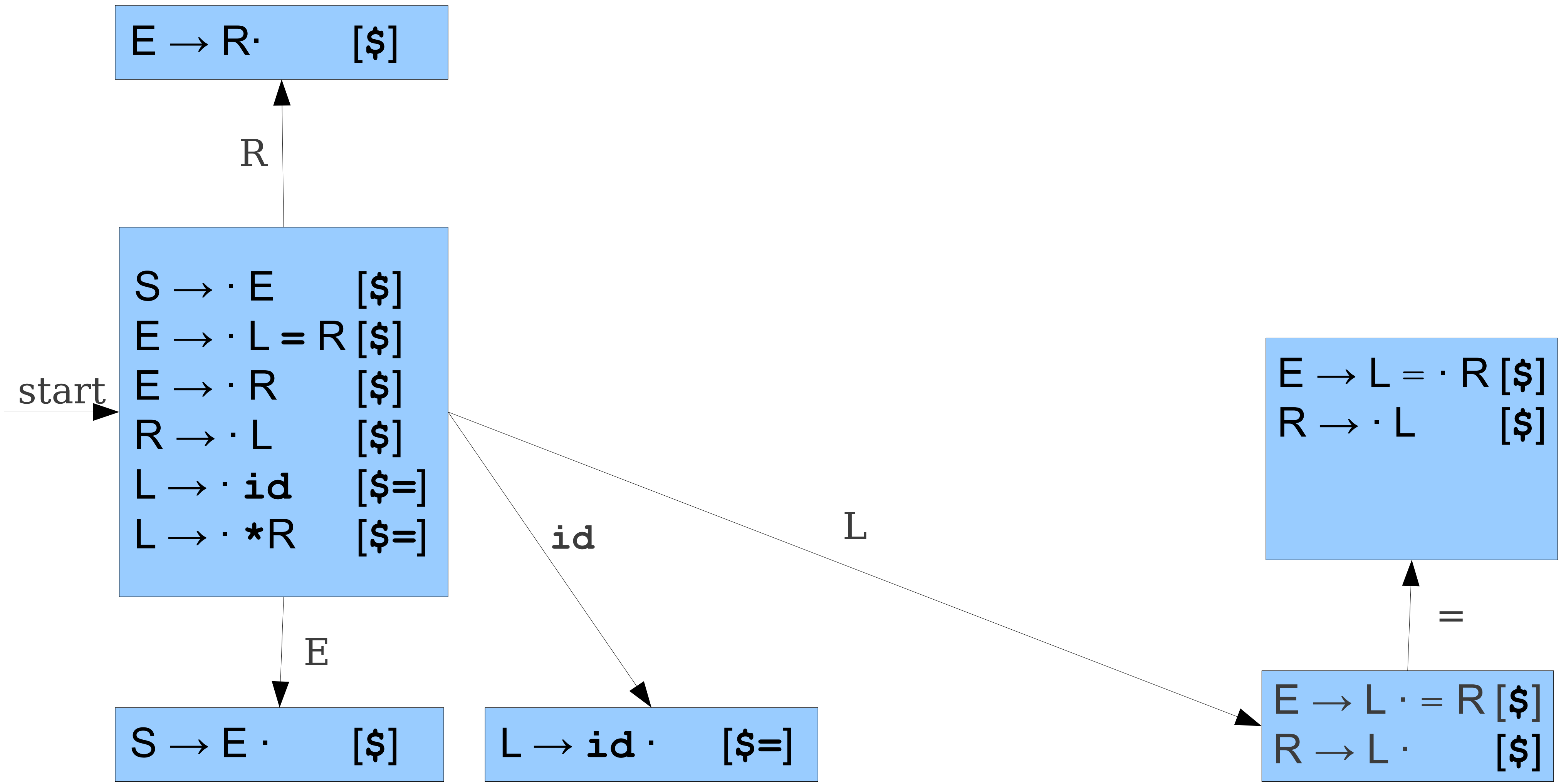
$S \to E$
$E \to L = R$
$E \to R$
$L \to \mathtt{id}$
$L \to *R$
$R \to L$

LALR(1) Construction



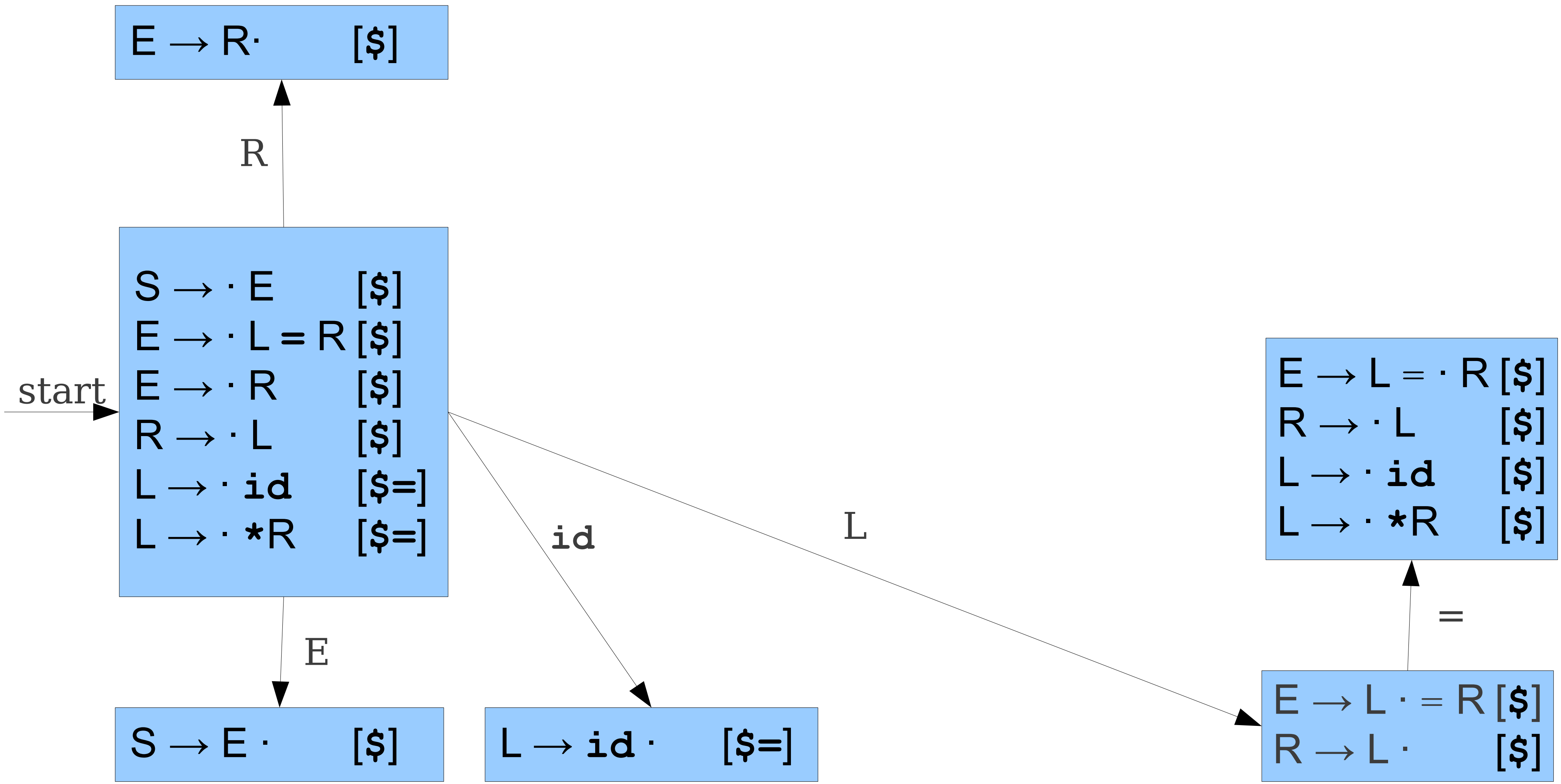
S → E
E → L = R
E → R
L → id
L → *R
R → L

LALR(1) Construction



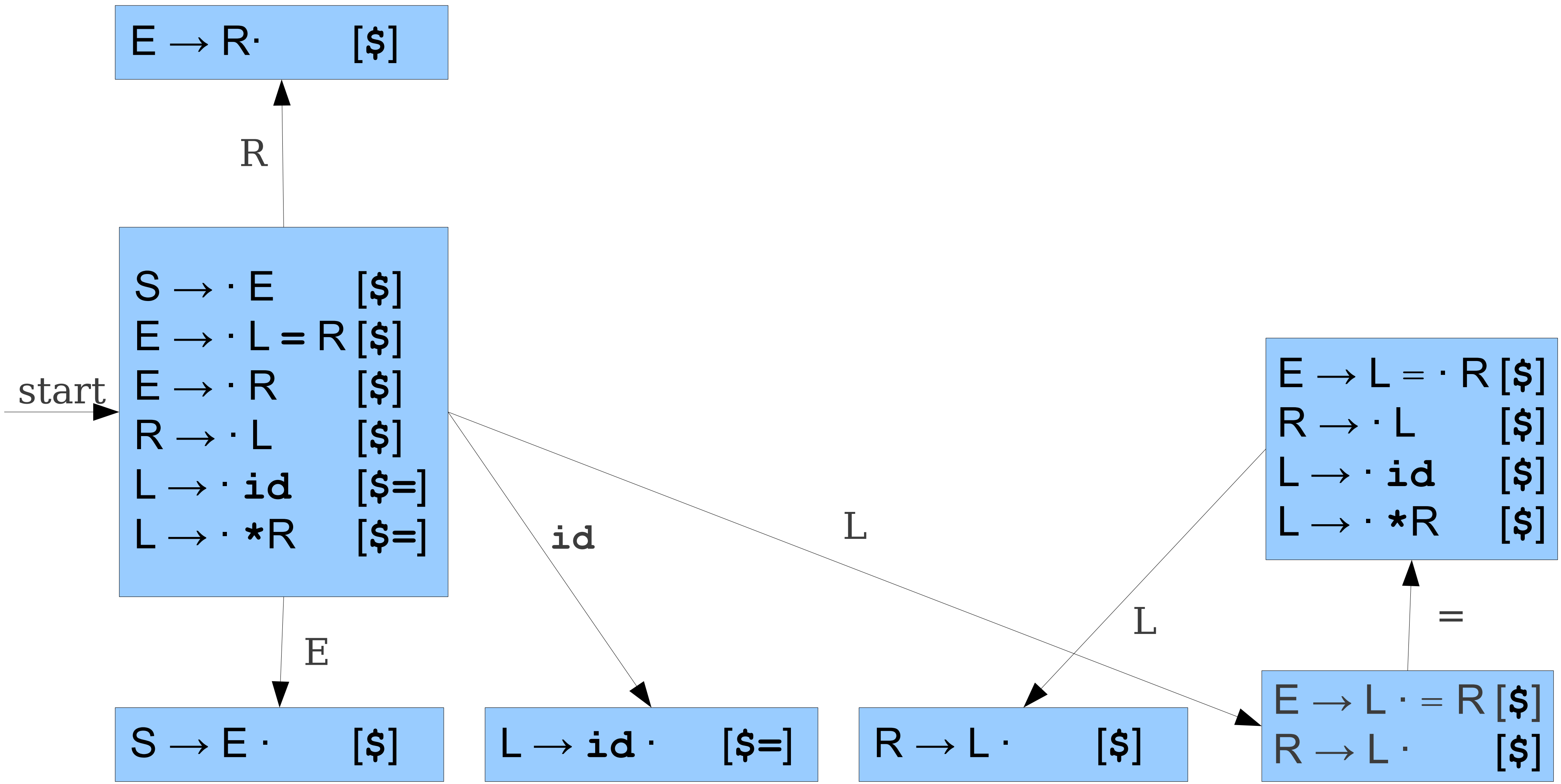
S → E
E → L = R
E → R
L → id
L → *R
R → L

LALR(1) Construction



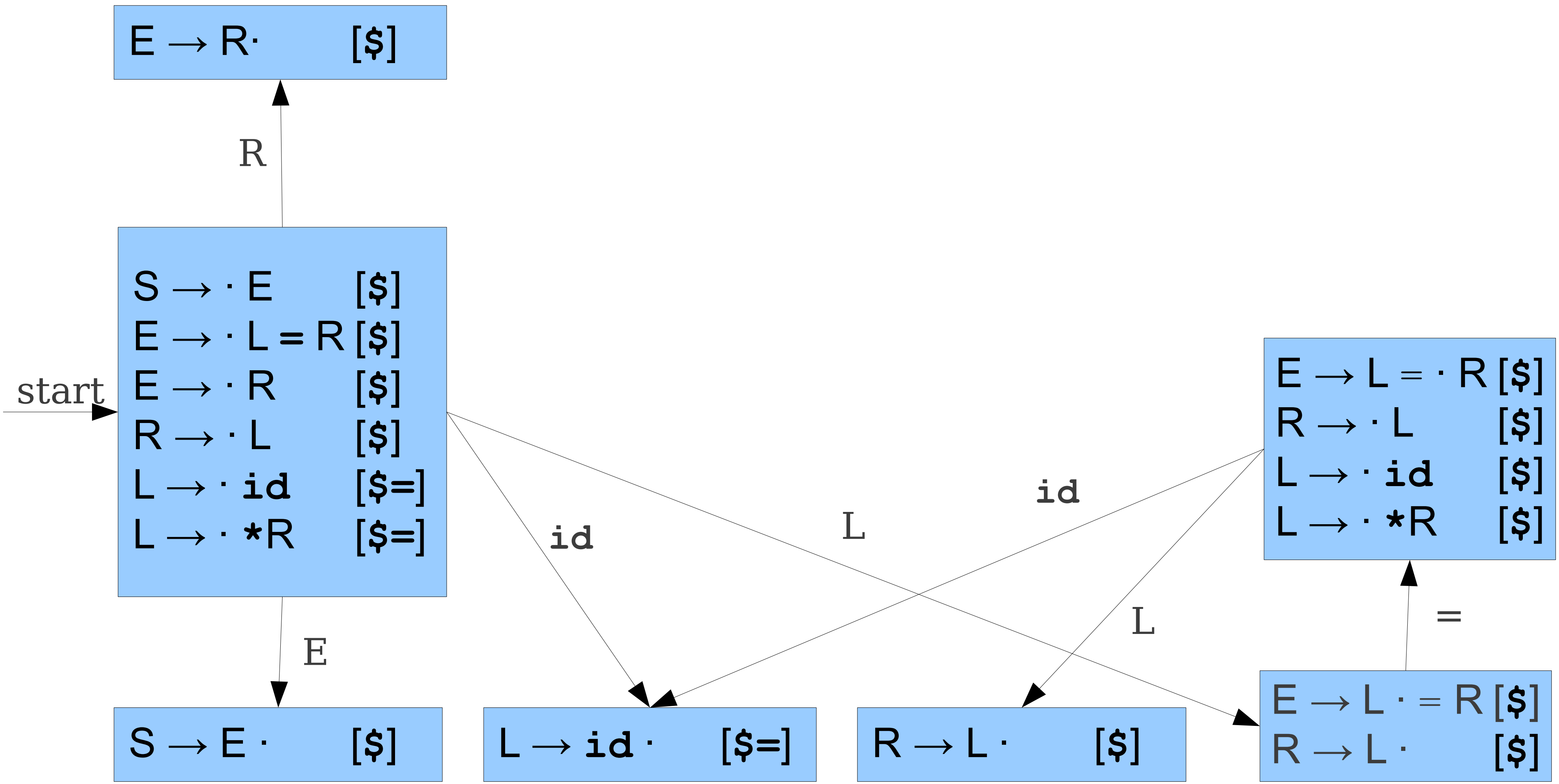
S → E
E → L = R
E → R
L → id
L → *R
R → L

LALR(1) Construction



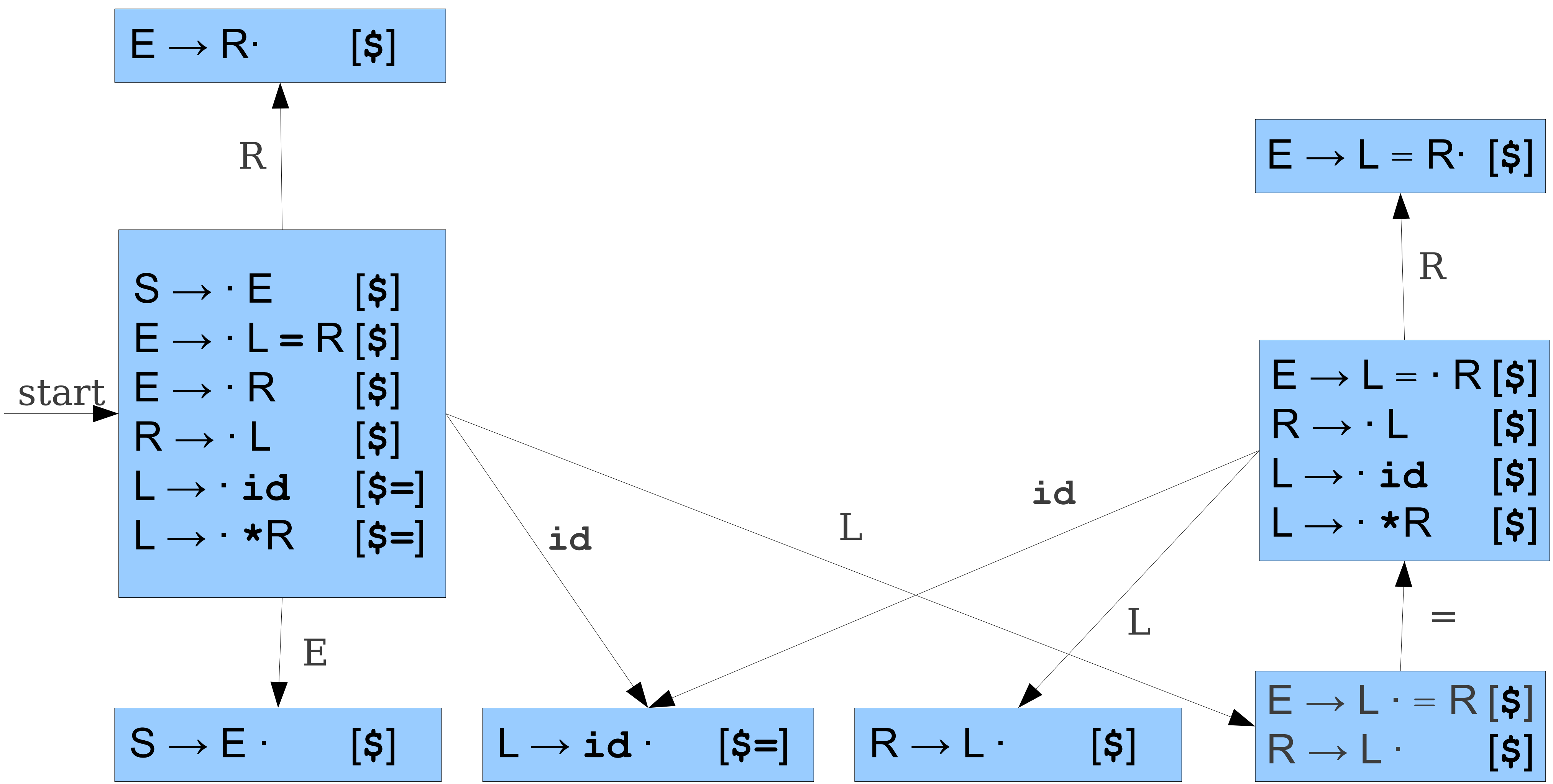
S → E
E → L = R
E → R
L → id
L → *R
R → L

LALR(1) Construction



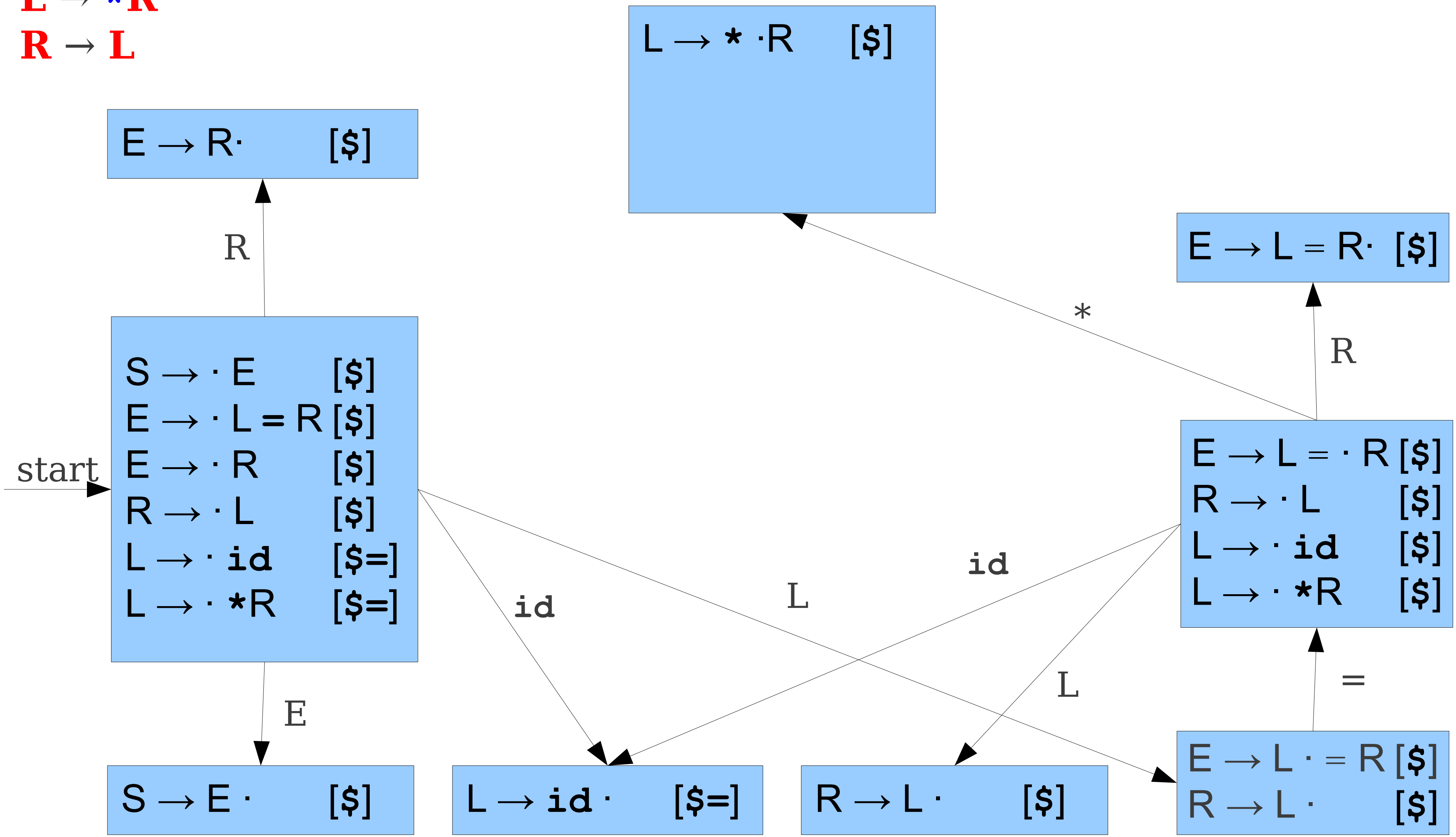
$S \to E$
$E \to L = R$
$E \to R$
$L \to \texttt{id}$
$L \to *R$
$R \to L$

LALR(1) Construction



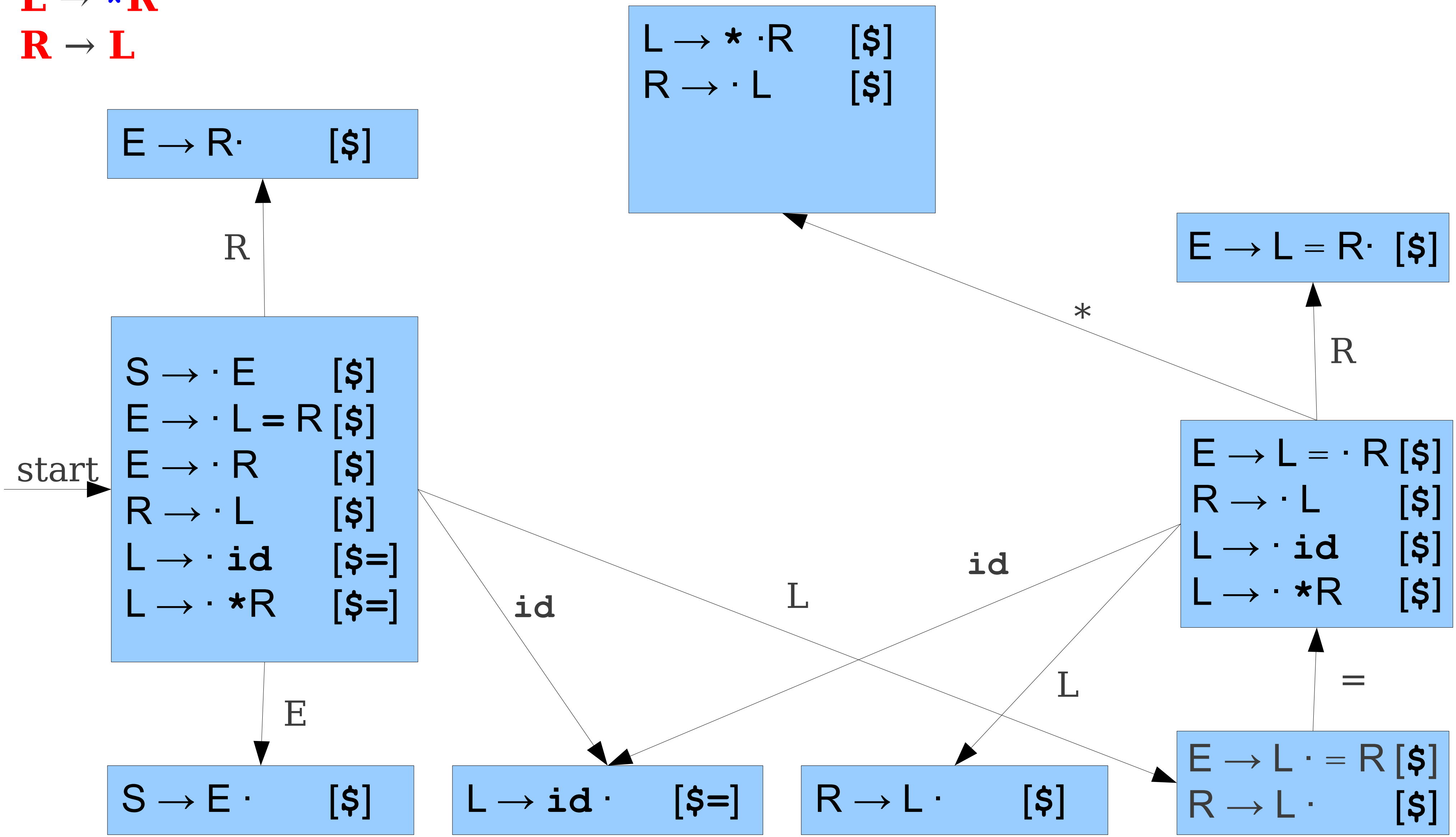
$S \to E$
$E \to L = R$
$E \to R$
$L \to \texttt{id}$
$L \to {*}R$
$R \to L$

LALR(1) Construction



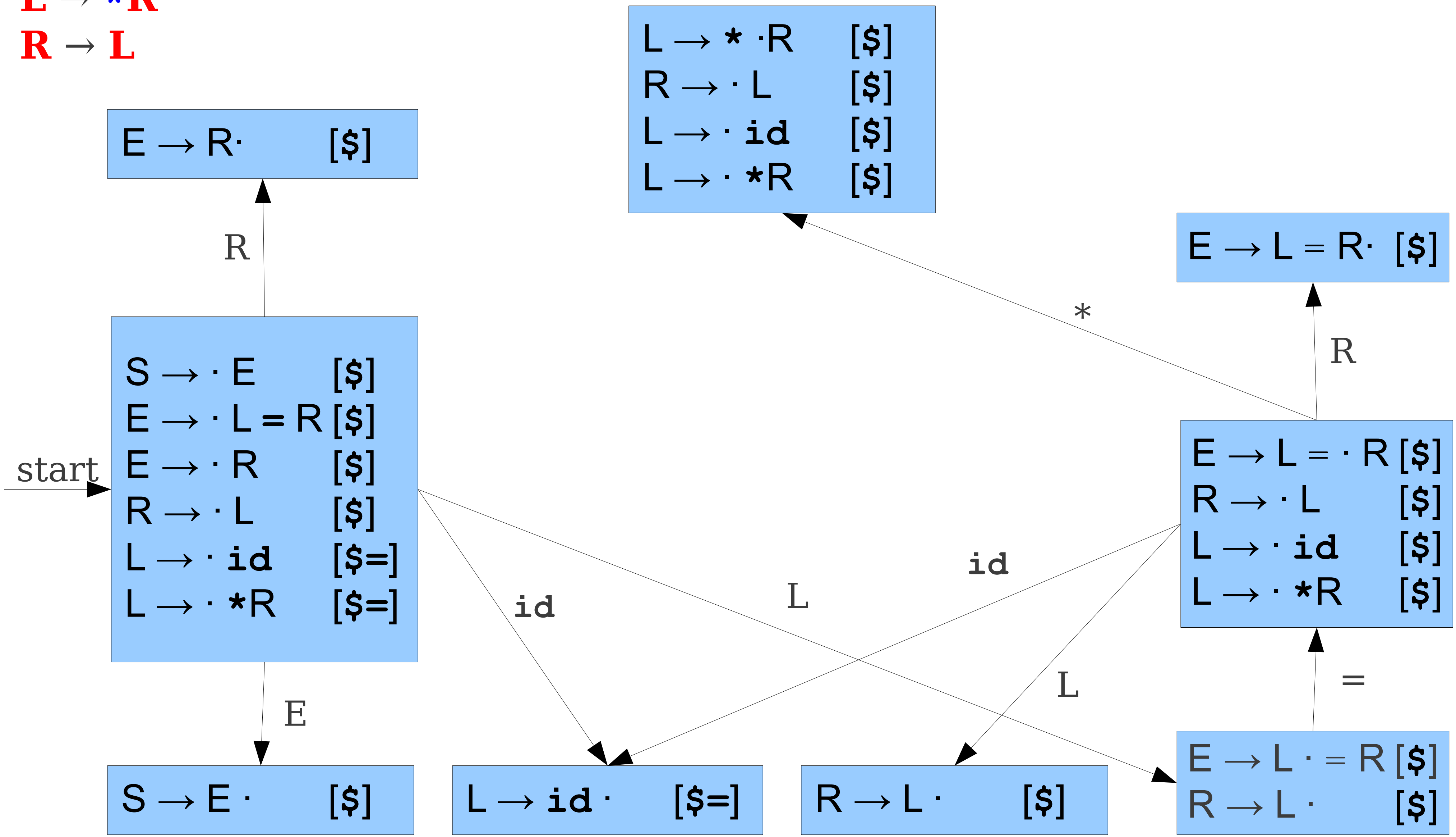
LALR(1) Construction

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



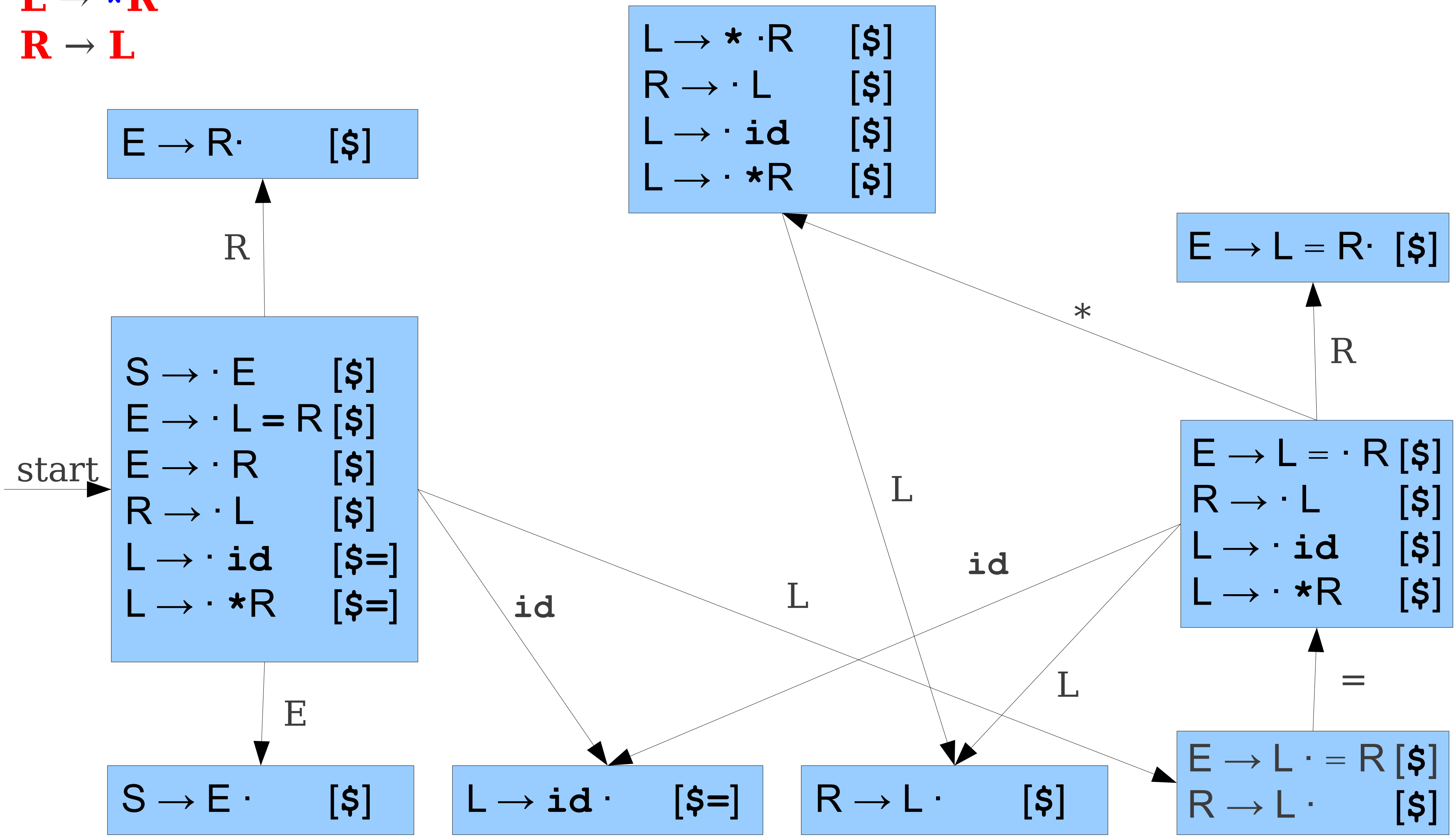
$S \rightarrow E$
$E \rightarrow L = R$
$E \rightarrow R$
$L \rightarrow$ **id**
$L \rightarrow$ ***R**
$R \rightarrow L$

LALR(1) Construction



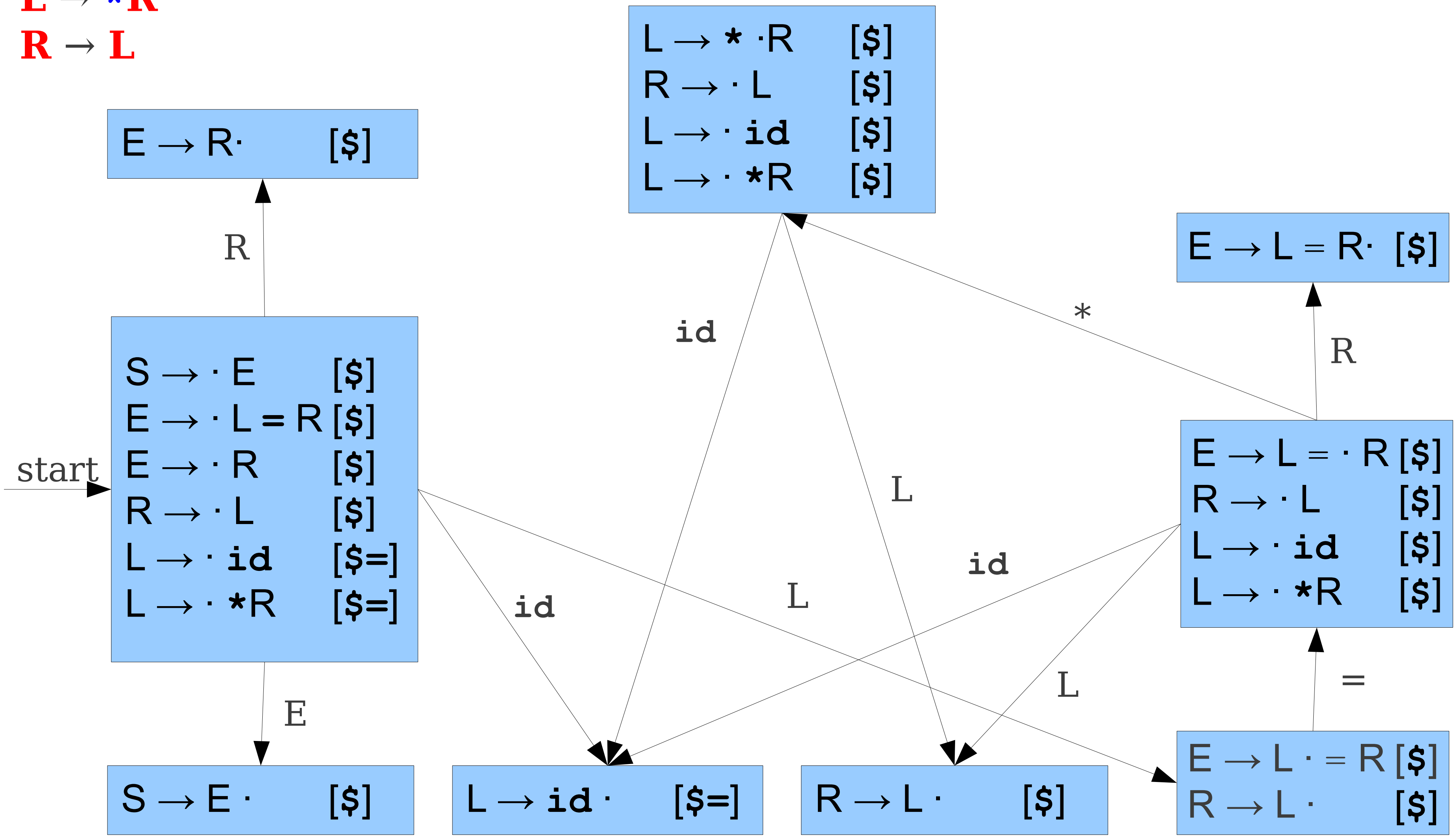
S → E
E → L = R
E → R
L → id
L → *R
R → L

LALR(1) Construction



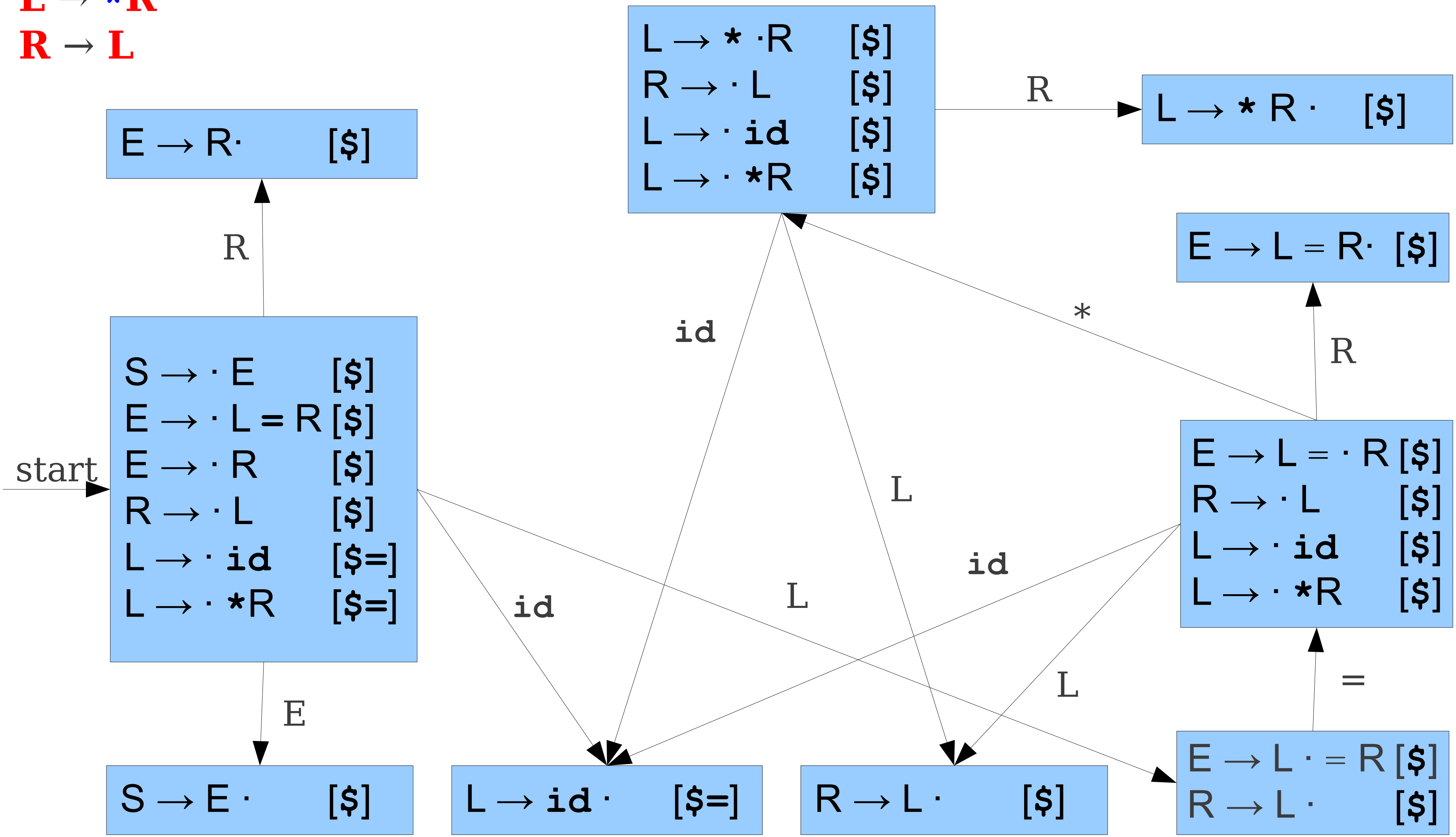
$S \rightarrow E$
$E \rightarrow L = R$
$E \rightarrow R$
$L \rightarrow$ **id**
$L \rightarrow$ ***R**
$R \rightarrow L$

LALR(1) Construction



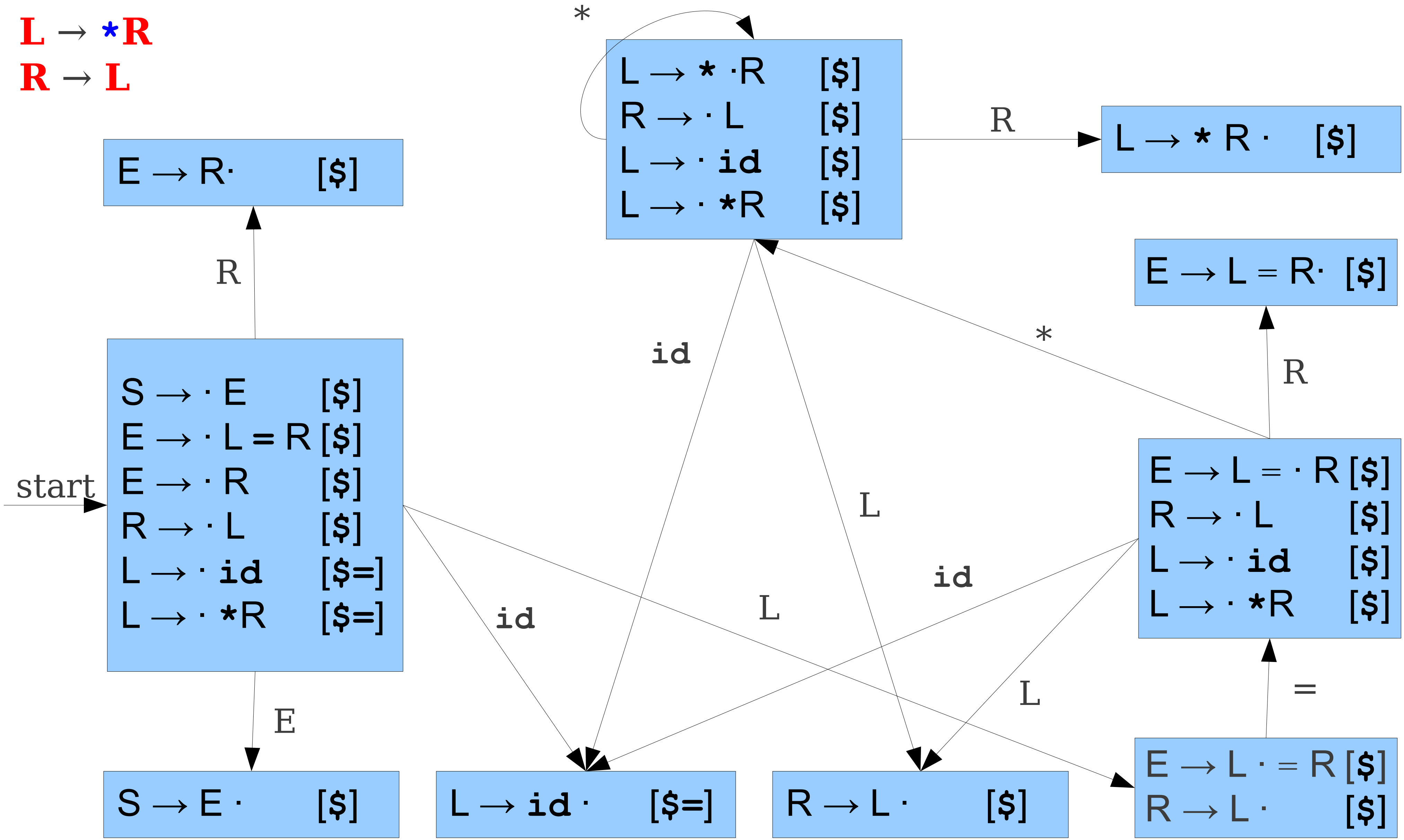
S → E
E → L = R
E → R
L → id
L → *R
R → L

LALR(1) Construction



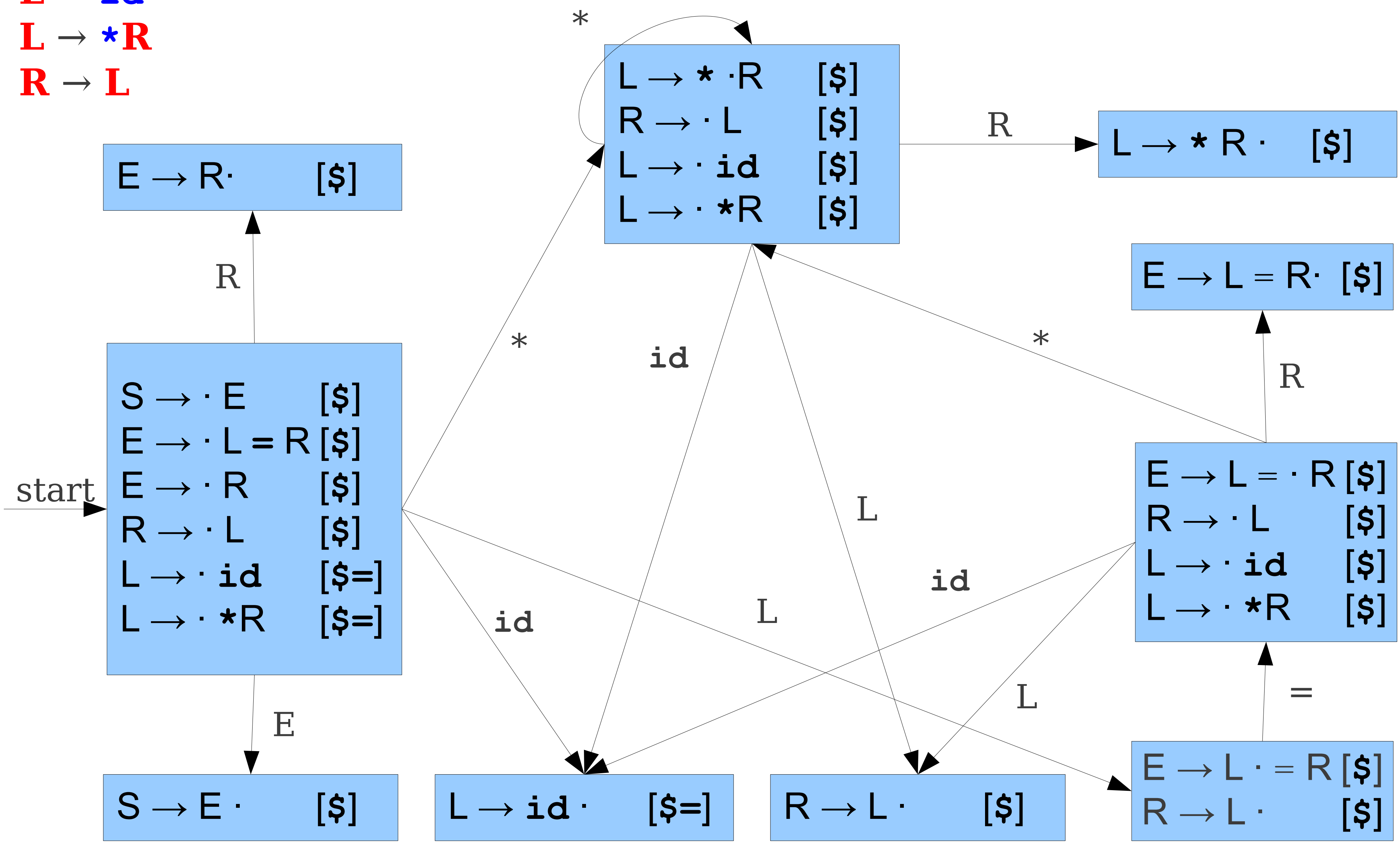
$S \to E$
$E \to L = R$
$E \to R$
$L \to$ **id**
$L \to *R$
$R \to L$

LALR(1) Construction



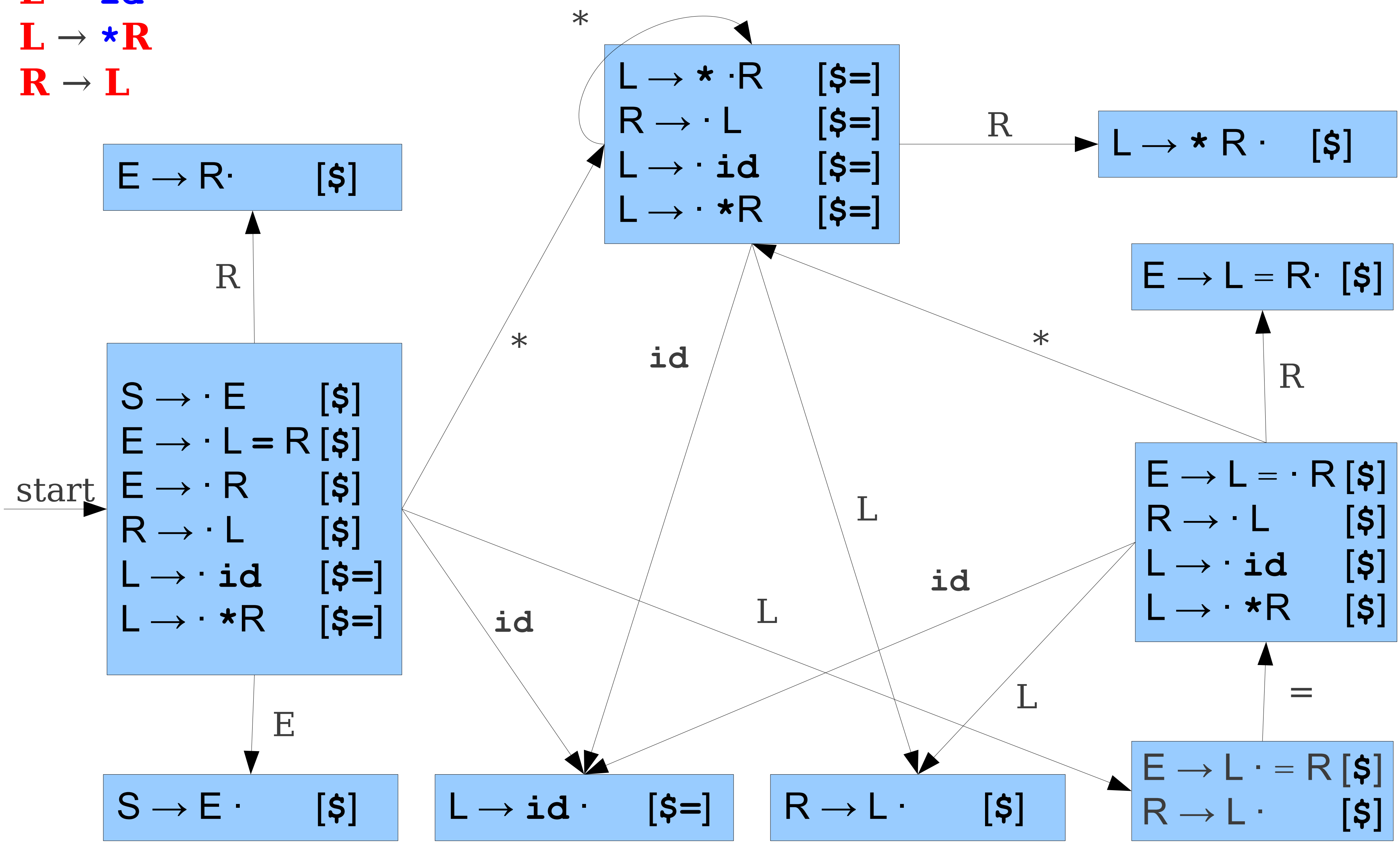
$S \to E$
$E \to L = R$
$E \to R$
$L \to$ id
$L \to$ *R
$R \to L$

LALR(1) Construction



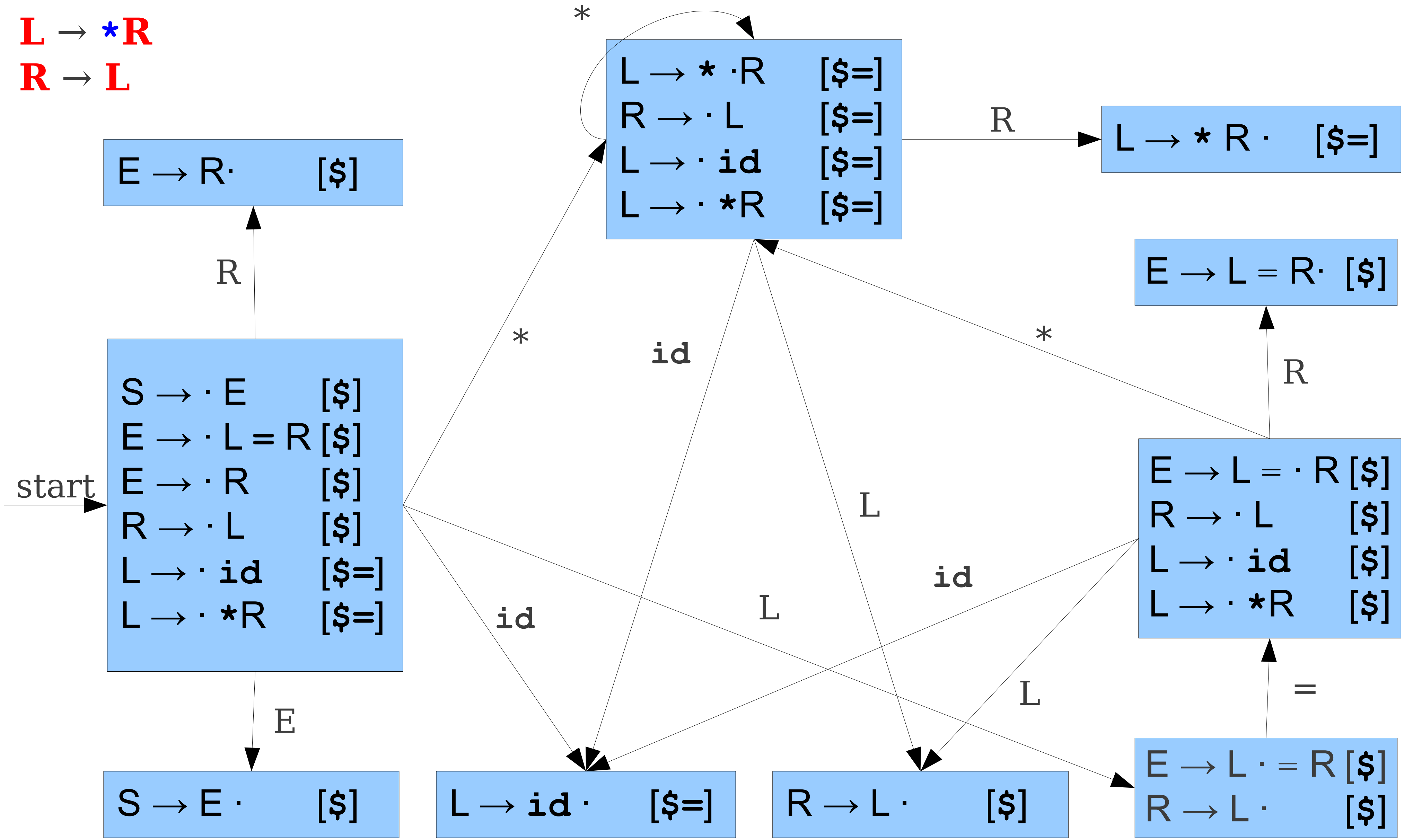
$S \rightarrow E$
$E \rightarrow L = R$
$E \rightarrow R$
$L \rightarrow$ **id**
$L \rightarrow$ *****R
$R \rightarrow L$

LALR(1) Construction



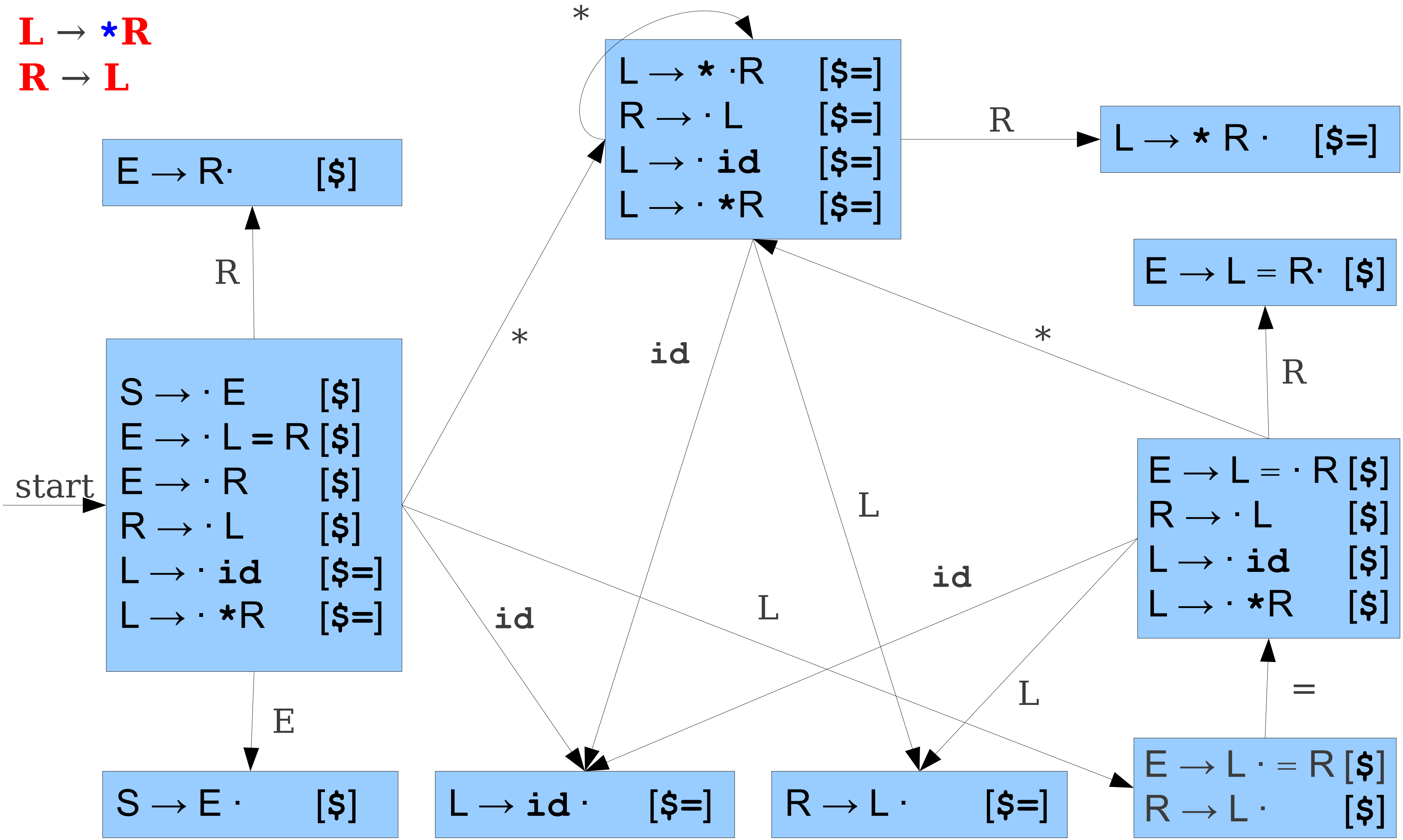
S → E
E → L = R
E → R
L → id
L → *R
R → L

LALR(1) Construction



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

LALR(1) Construction



Analysis of our Algorithm

- Since we merge as we go, size of the partial automaton never exceeds size of overall automaton.
- However, this algorithm could be very slow in practice.
  - We might still have to generate all the LR(1) states, even if they immediately get merged.
- This can be very slow.

SLR uses FOLLOW sets

- Recall: FOLLOW(**A**) is the set of terminals that can follow **A** in a derivation:

  FOLLOW(**A**) = { **`t`** | **S** ⇒* $\boldsymbol{\alpha}$**A**`t`$\boldsymbol{\omega}$ }

- SLR is LR(0), with reductions augmented using FOLLOW sets.
- This is too weak for two reasons:
  - It ignores context (what state we're in).
  - It ignores which reduction we're doing.

LALR uses LA sets

- Given an LR(0) state q and a production **A** → ***γ***, the **lookahead set** LA(q, **A** → ***γ***) is defined as

  LA(q, **A** → ***γ***) = { `t` | **S** ⇒* ***α*****A**`t`***ω*** and ***αγ*** reaches q }

- Here, "***αγ*** reaches q" means that the LR(0) automaton, when run on ***αγ***, reaches state q.
- Intuitively, if we're in some state q and are going to reduce **A** to ***γ***, LA(q, **A** → ***γ***) is the set of terminals that could actually follow **A** at this point, given that we're reducing **A** → ***γ***.
- Much more precise than FOLLOW sets.

LA and FOLLOW

- The **lookahead set** LA(q, $\mathbf{A} \to \boldsymbol{\gamma}$) is defined as

  LA(q, $\mathbf{A} \to \boldsymbol{\gamma}$) = $\{ \mathtt{t} \mid \mathbf{S} \Rightarrow^* \boldsymbol{\alpha}\mathbf{A}\mathtt{t}\boldsymbol{\omega}$ and $\boldsymbol{\alpha\gamma}$ reaches $q \}$

- The **follow set** FOLLOW($\mathbf{A}$) is defined as

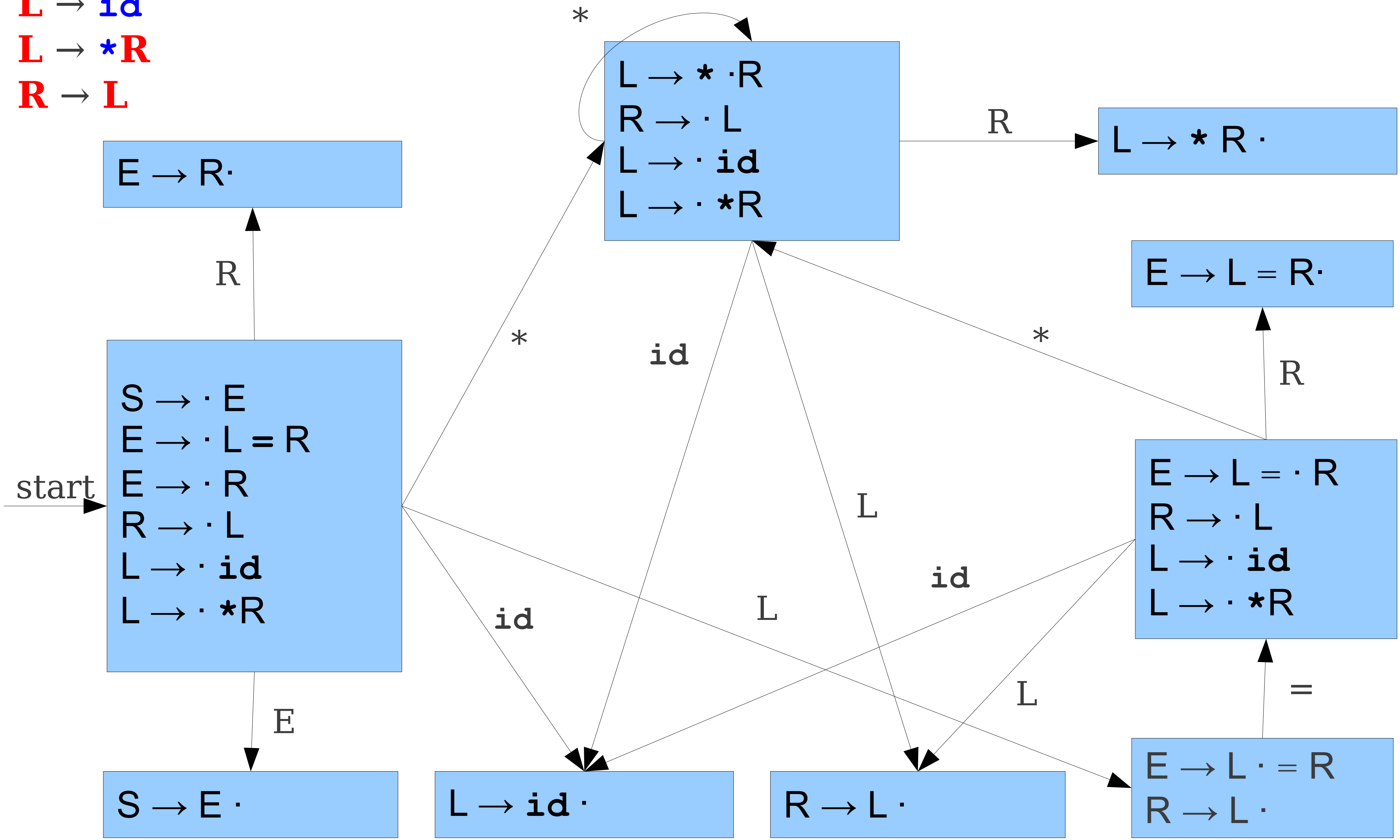
  FOLLOW($\mathbf{A}$) = $\{ \mathtt{t} \mid \mathbf{S} \Rightarrow^* \boldsymbol{\alpha}\mathbf{A}\mathtt{t}\boldsymbol{\omega} \}$

- Note that LA(q, $\mathbf{A} \to \boldsymbol{\gamma}$) $\subseteq$ FOLLOW($\mathbf{A}$); that is, LA sets are “more precise” than FOLLOW sets.

- If we can compute LA from FOLLOW, we can construct a LALR(1) parser efficiently.

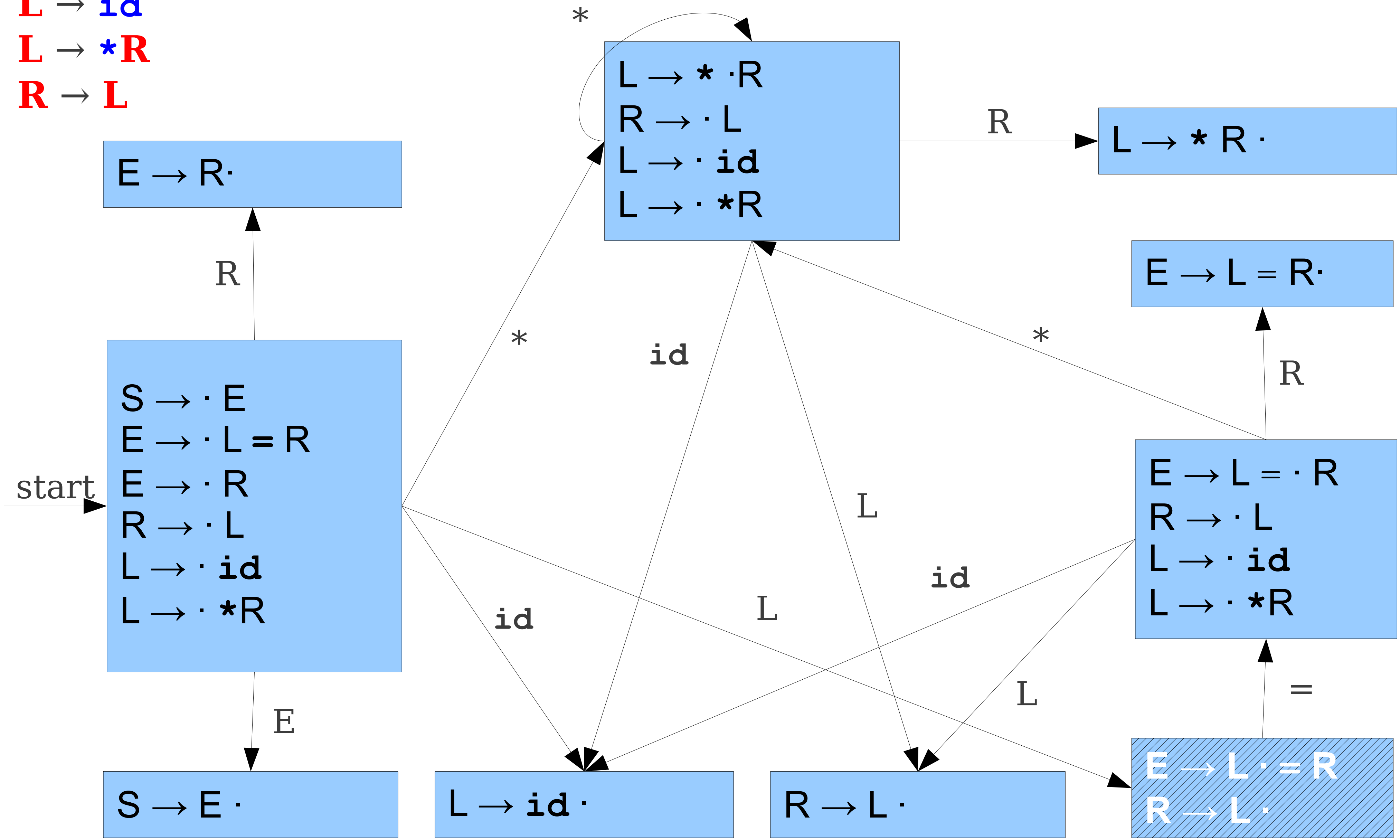
An LR(0) Automaton

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



$S \to E$
$E \to L = R$
$E \to R$
$L \to$ id
$L \to *R$
$R \to L$

An LR(0) Automaton

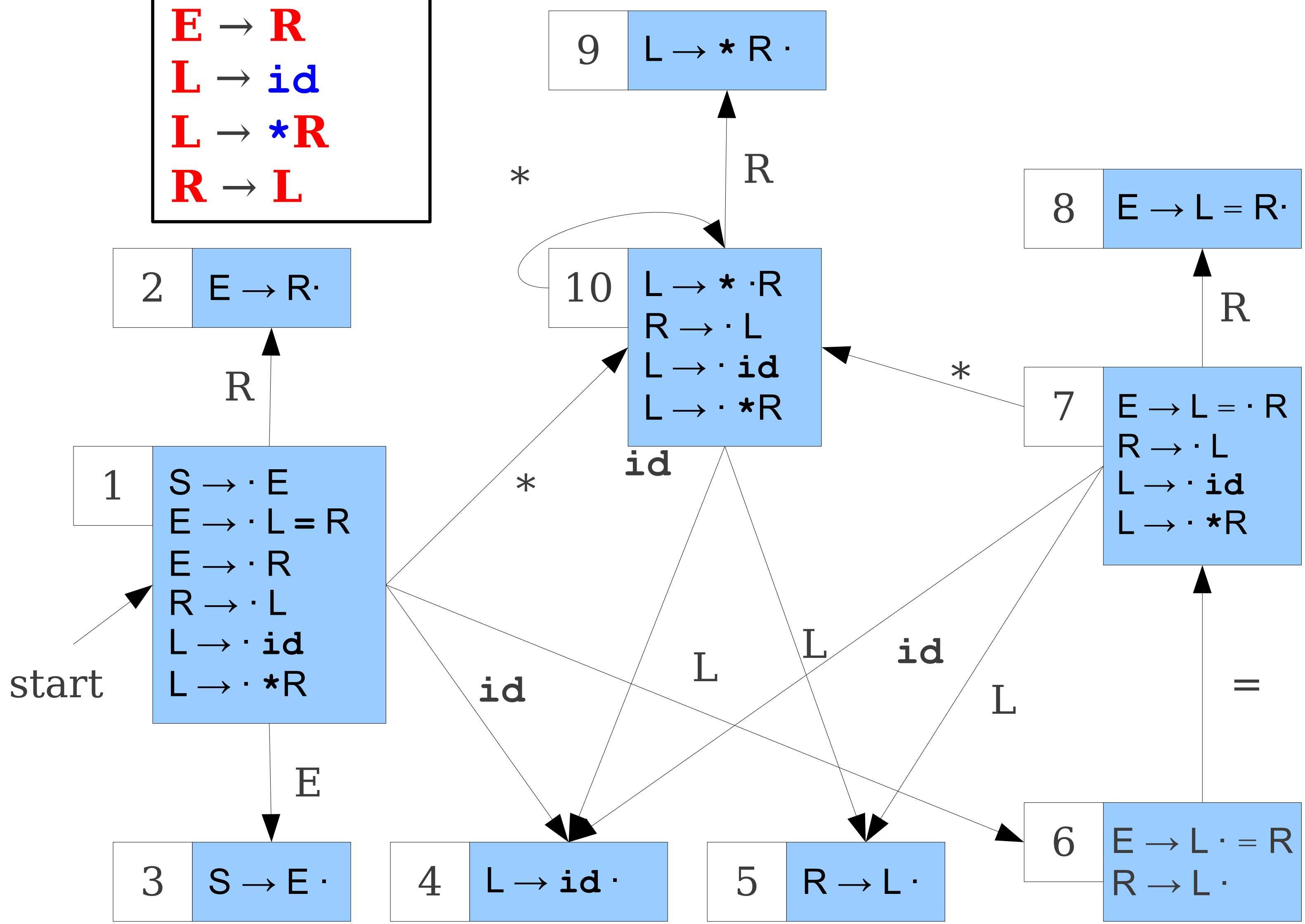


What if we used the LR(0) automaton to add context to the grammar?

Prepare for one of the most beautiful constructions of the quarter…

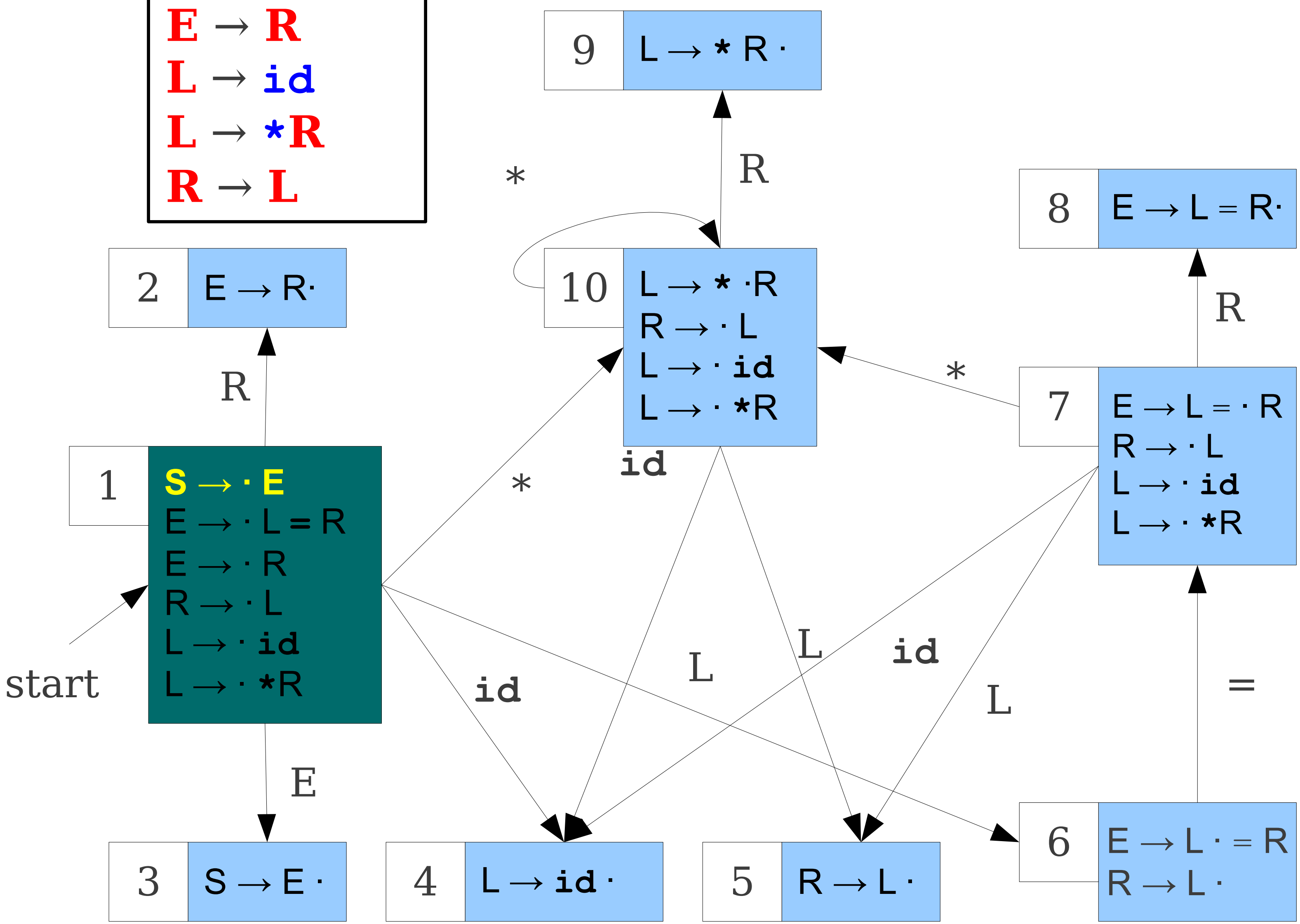
Augmenting the Grammar

$S \to E$
$E \to L$ **=** R
$E \to R$
$L \to$ **id**
$L \to$ *****R
$R \to L$



Augmenting the Grammar

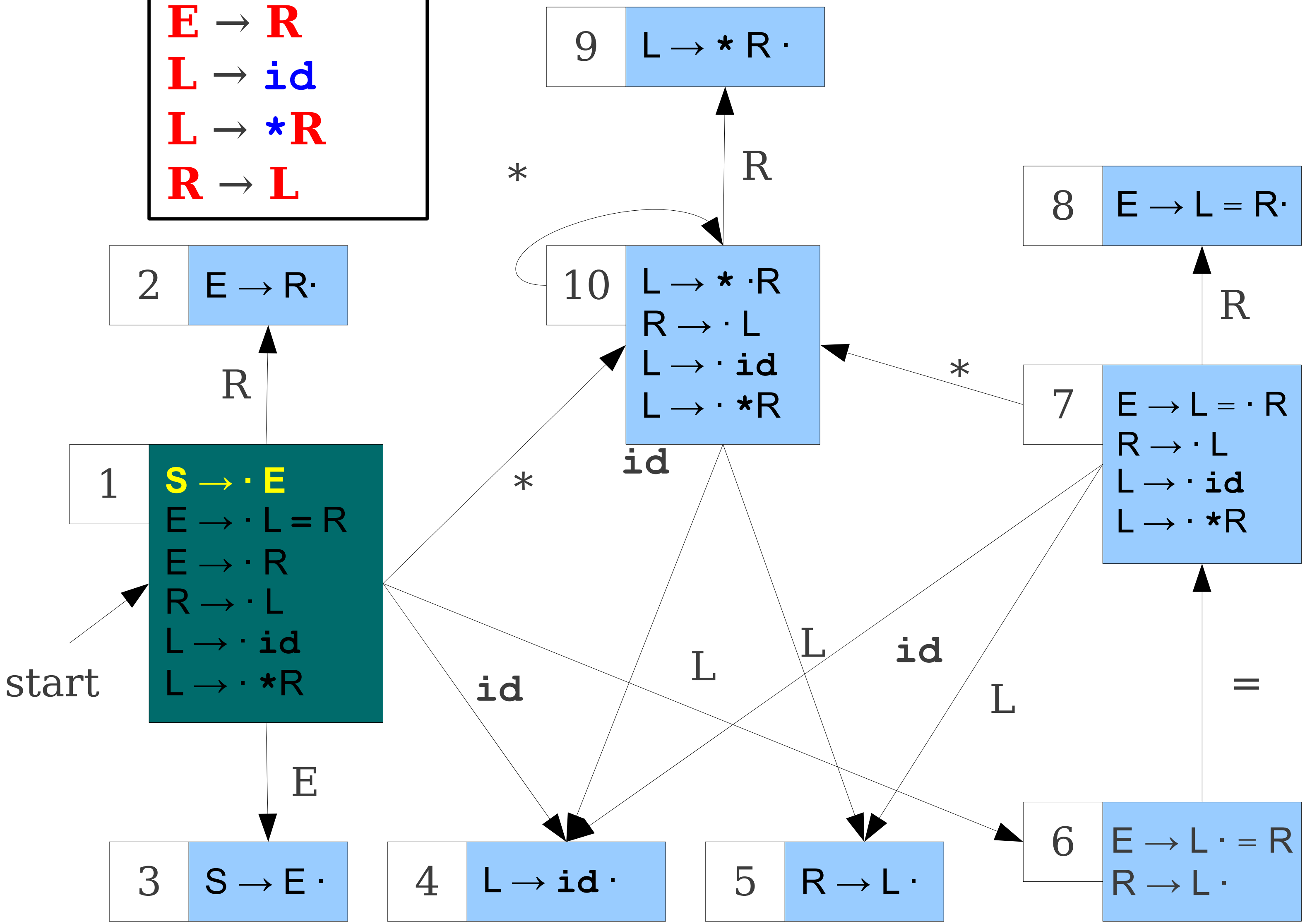
$S \rightarrow E$
$E \rightarrow L = R$
$E \rightarrow R$
$L \rightarrow \texttt{id}$
$L \rightarrow *R$
$R \rightarrow L$



Augmenting the Grammar

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

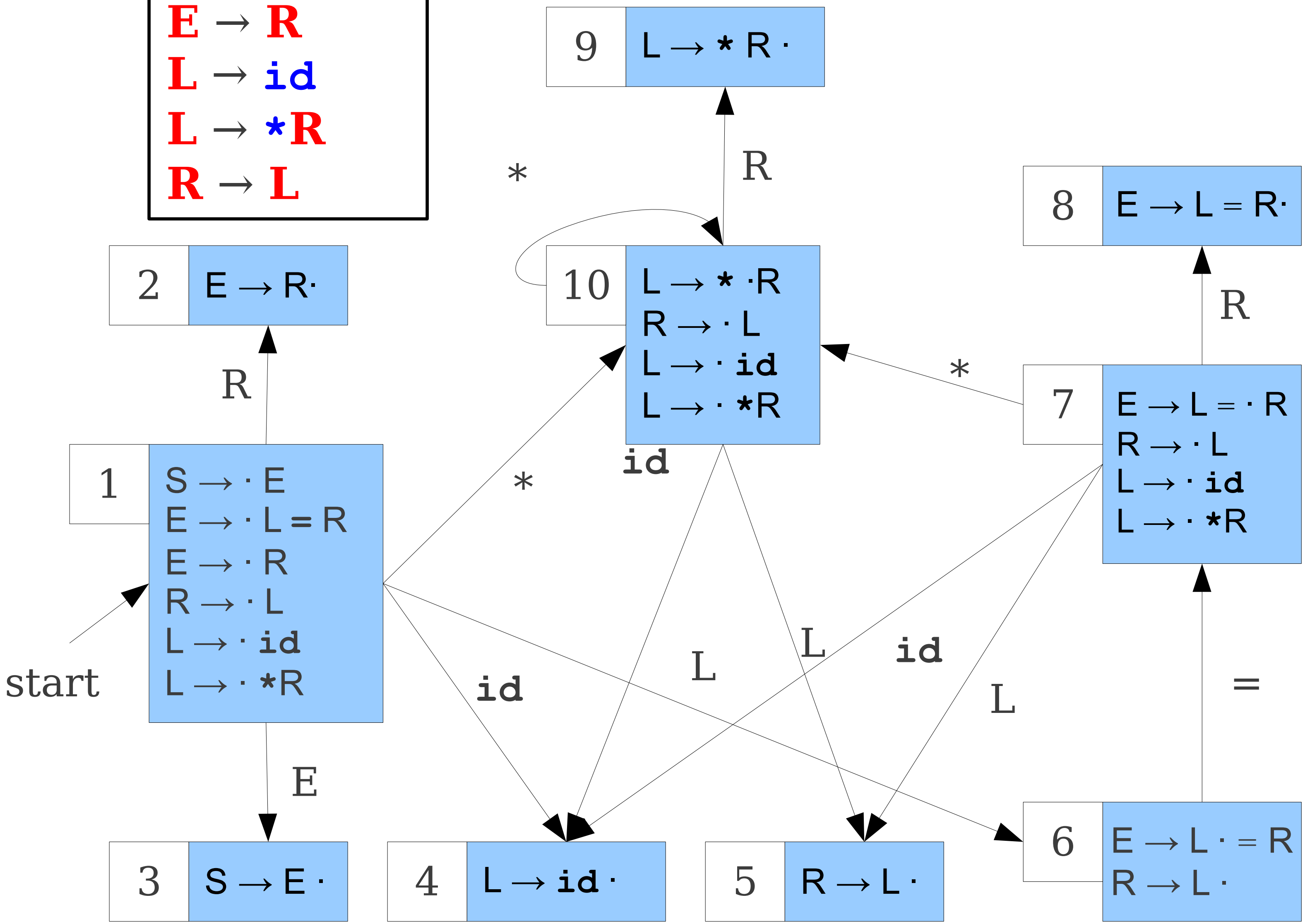
S → **E**
E → **L** **=** **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**



Augmenting the Grammar

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

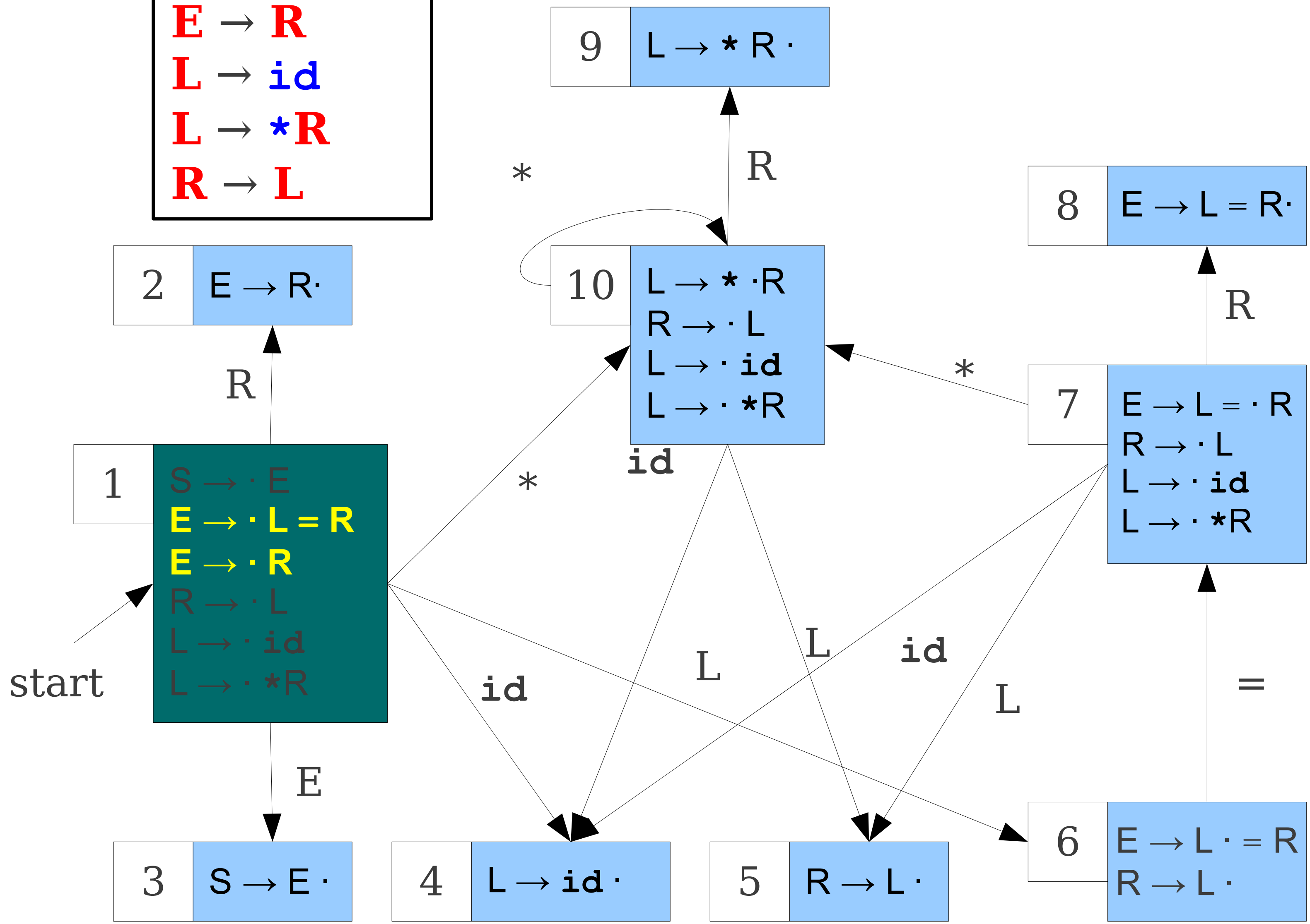
S → **E**
E → **L** = **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**



Augmenting the Grammar

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

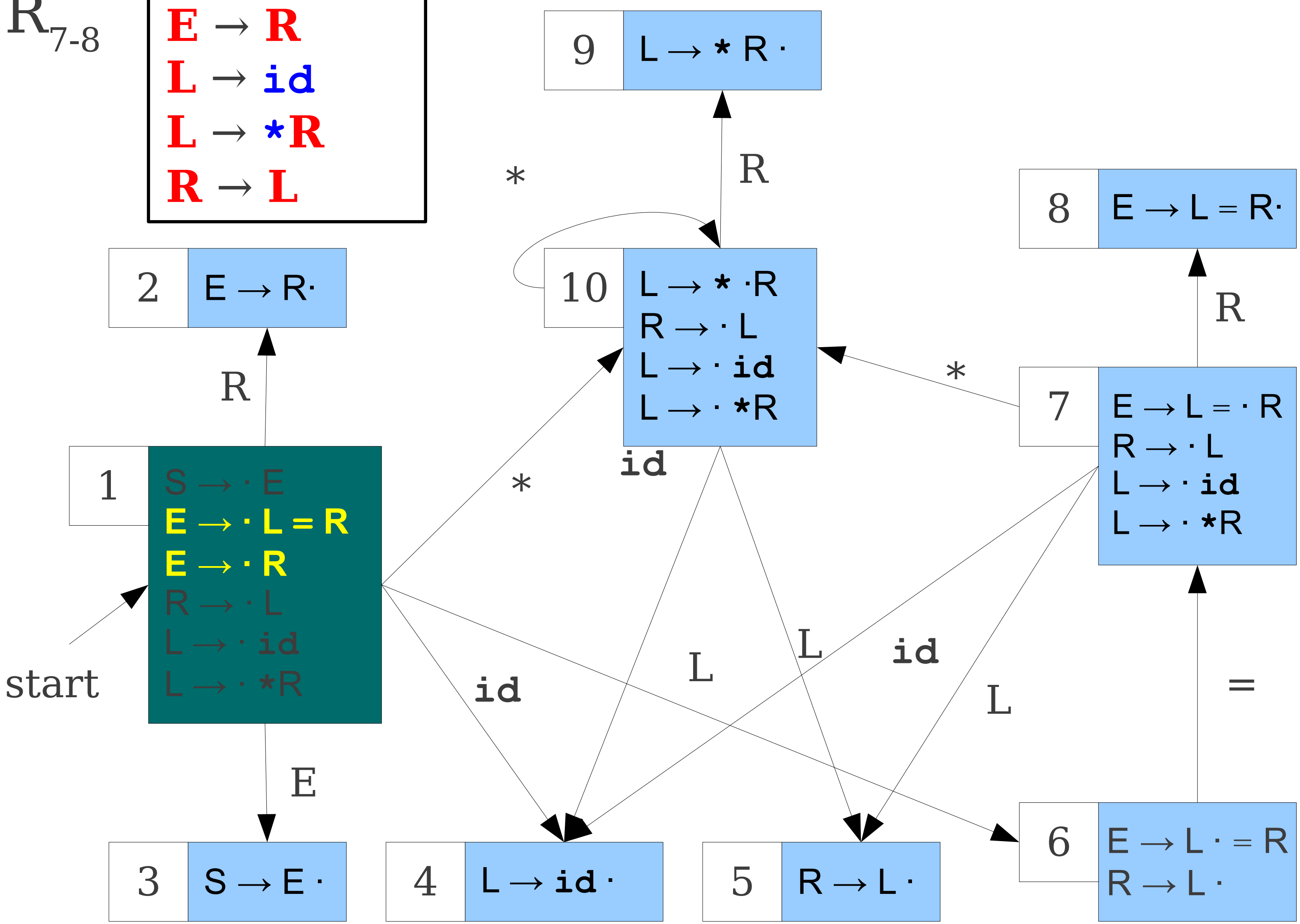
S → **E**
E → **L** `=` **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**



Augmenting the Grammar

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

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



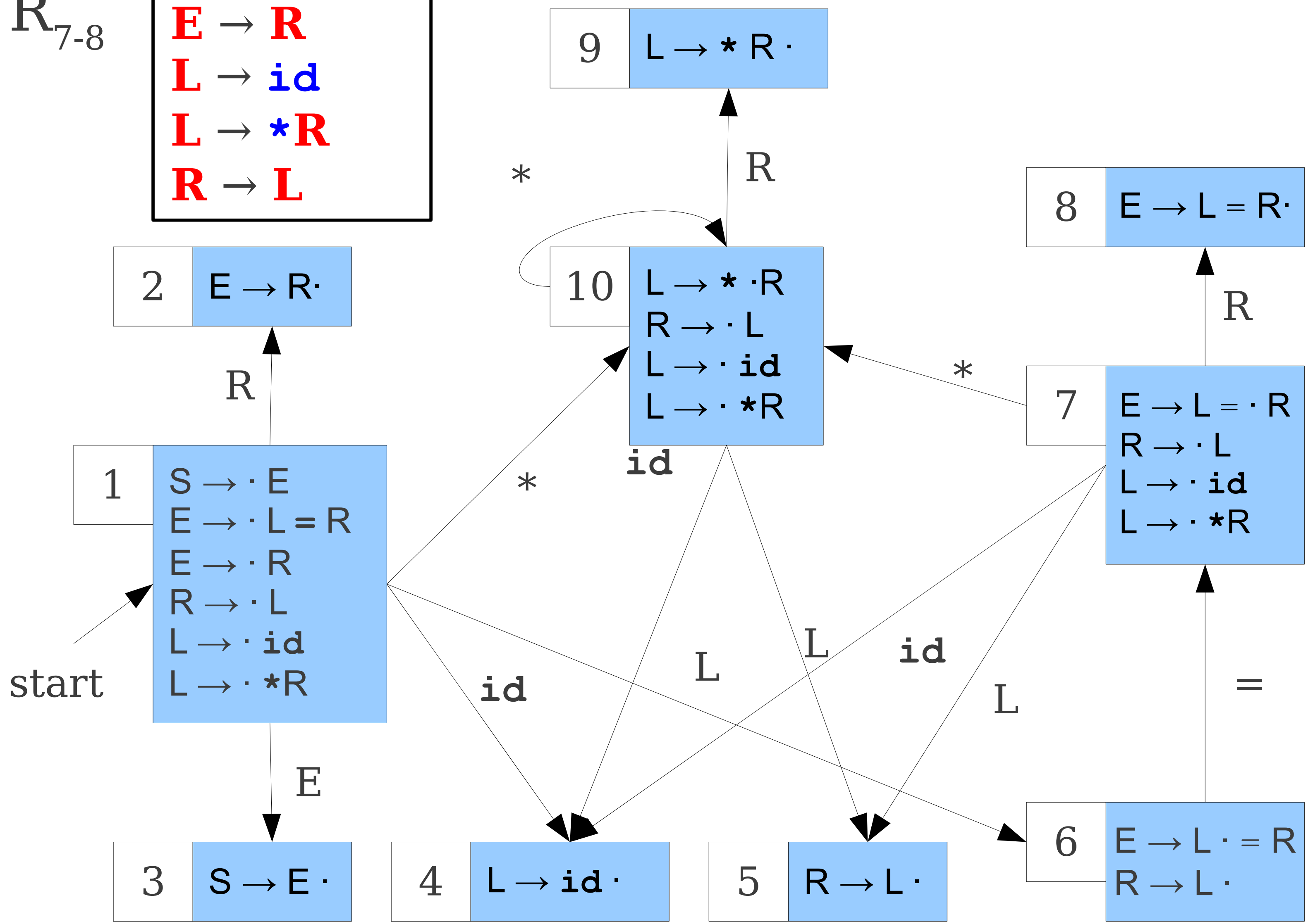
Augmenting the Grammar

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

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

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

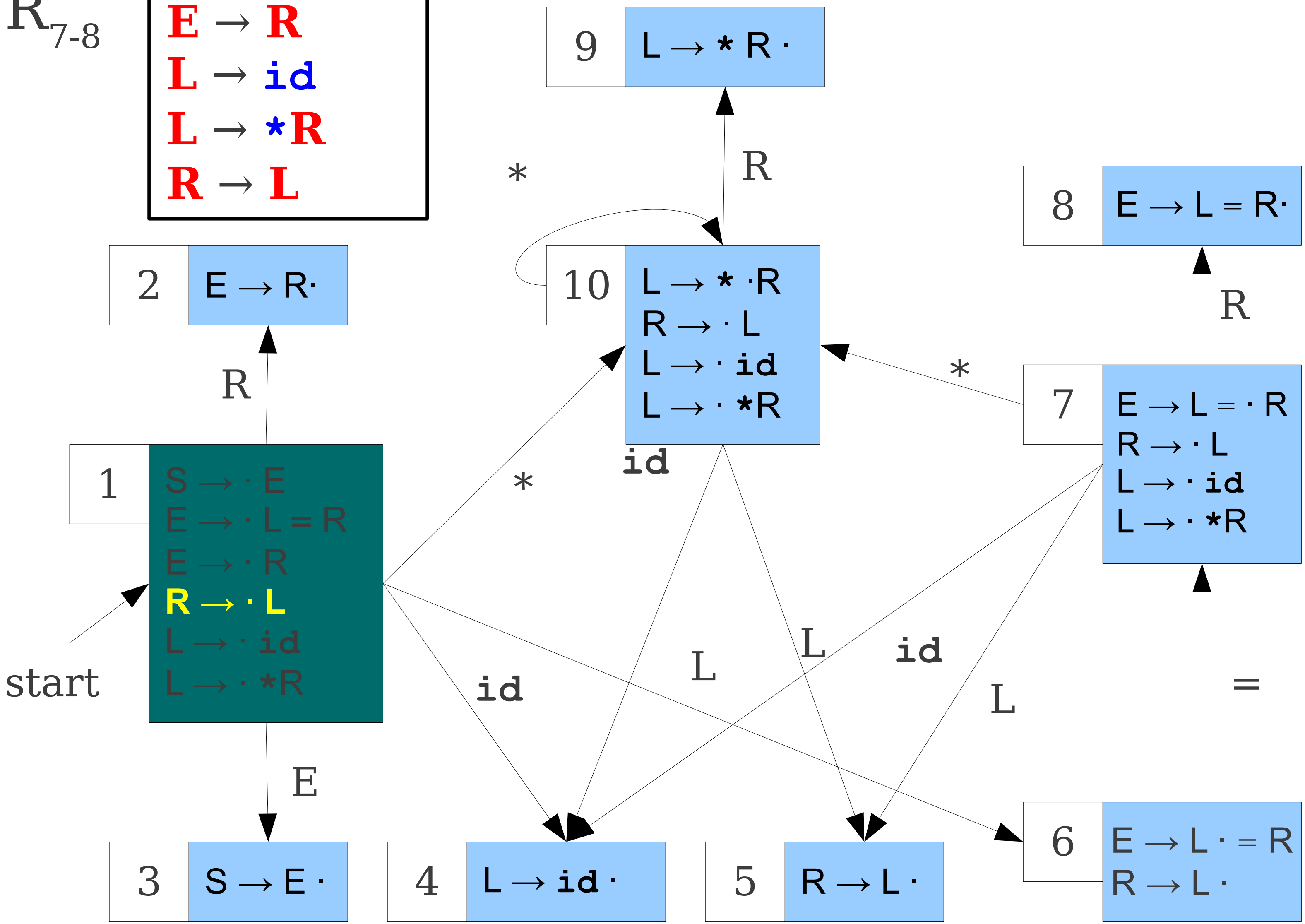
S → **E**
E → **L** **=** **R**
E → **R**
L → **id**
L → ***R**
R → **L**



Augmenting the Grammar

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

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



Augmenting the Grammar

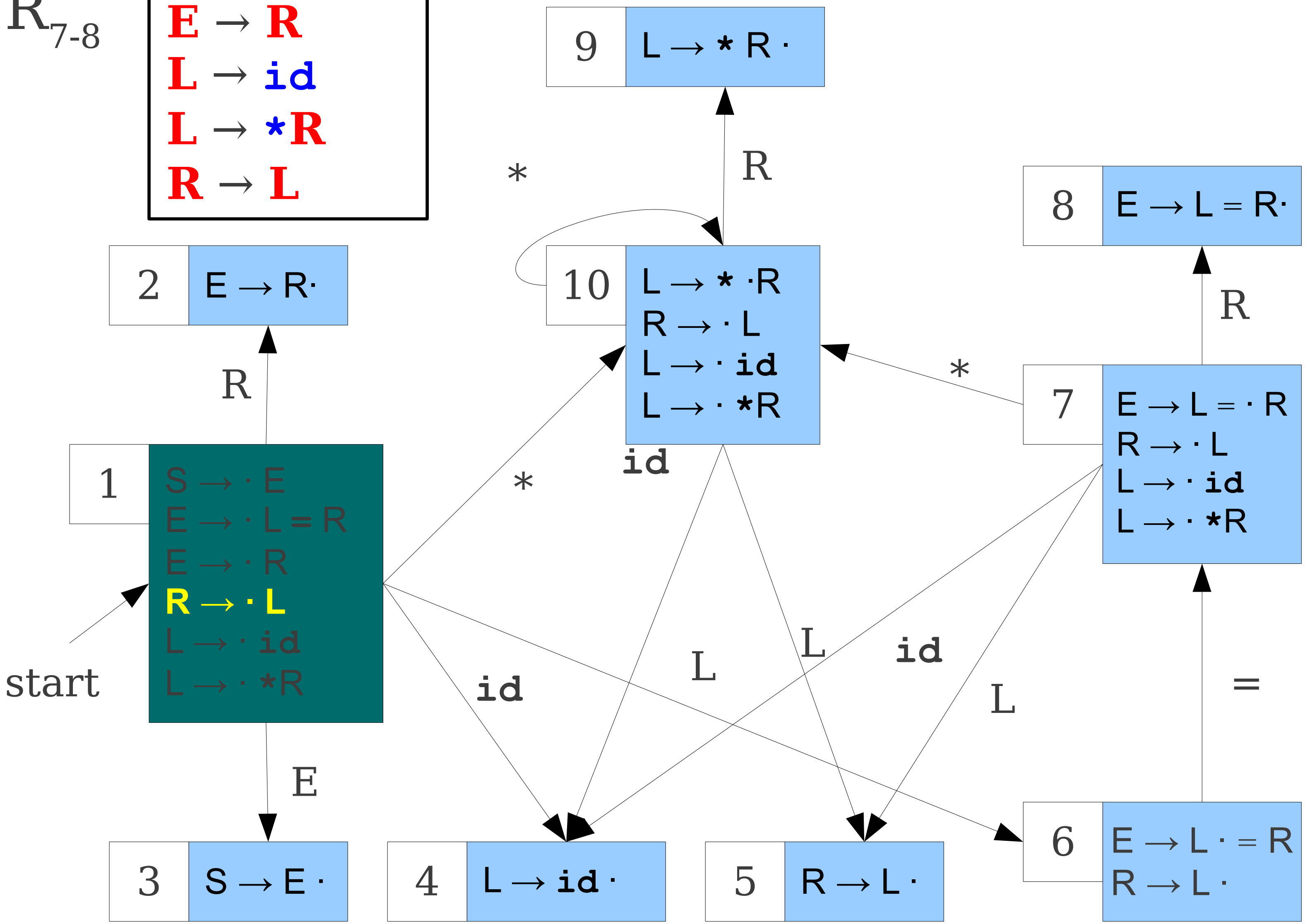
$S_1 \rightarrow E_{1\text{-}3}$

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

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

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

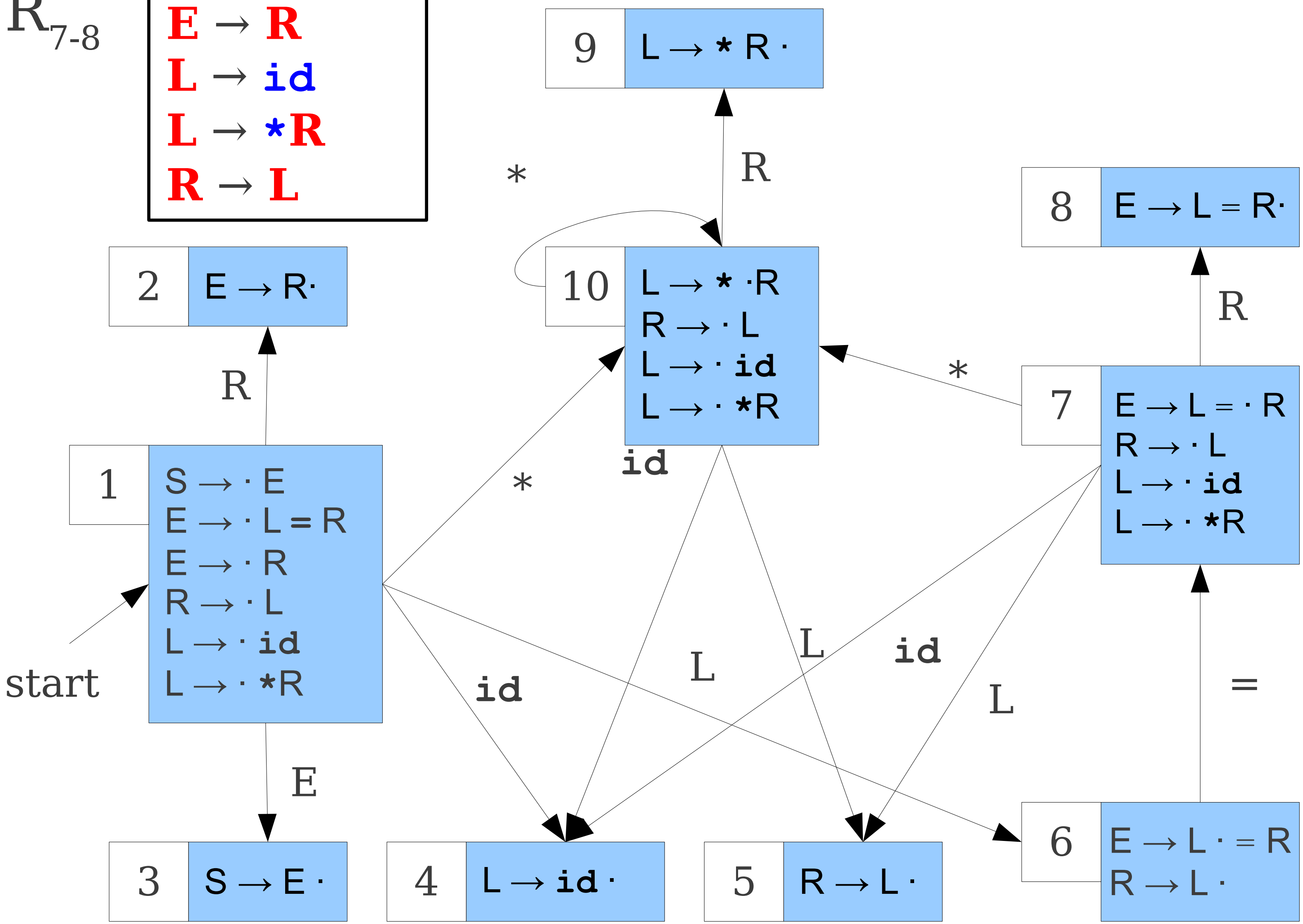
S → **E**
E → **L** **=** **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**



Augmenting the Grammar

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

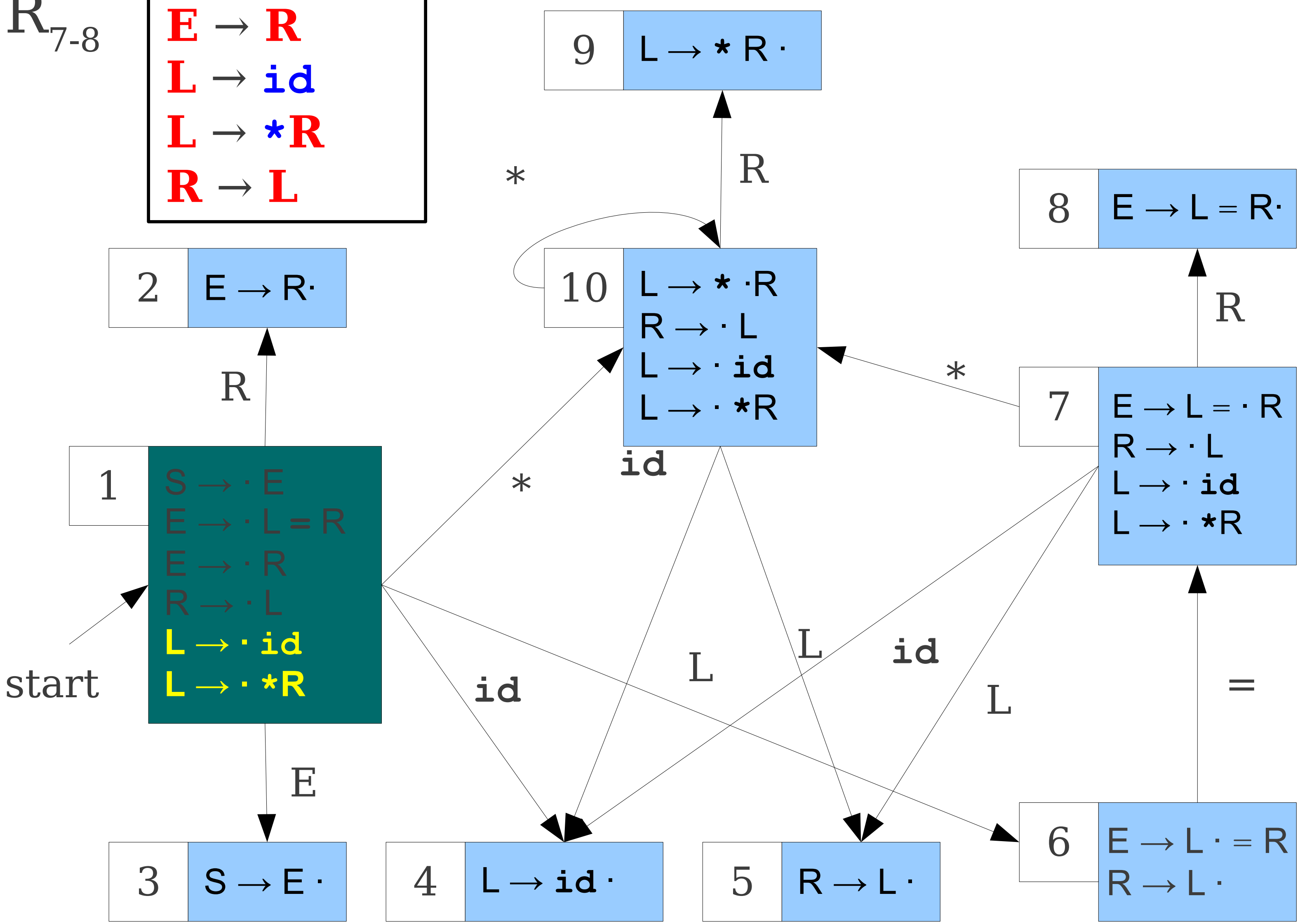
S → **E**
E → **L** `=` **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**



Augmenting the Grammar

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

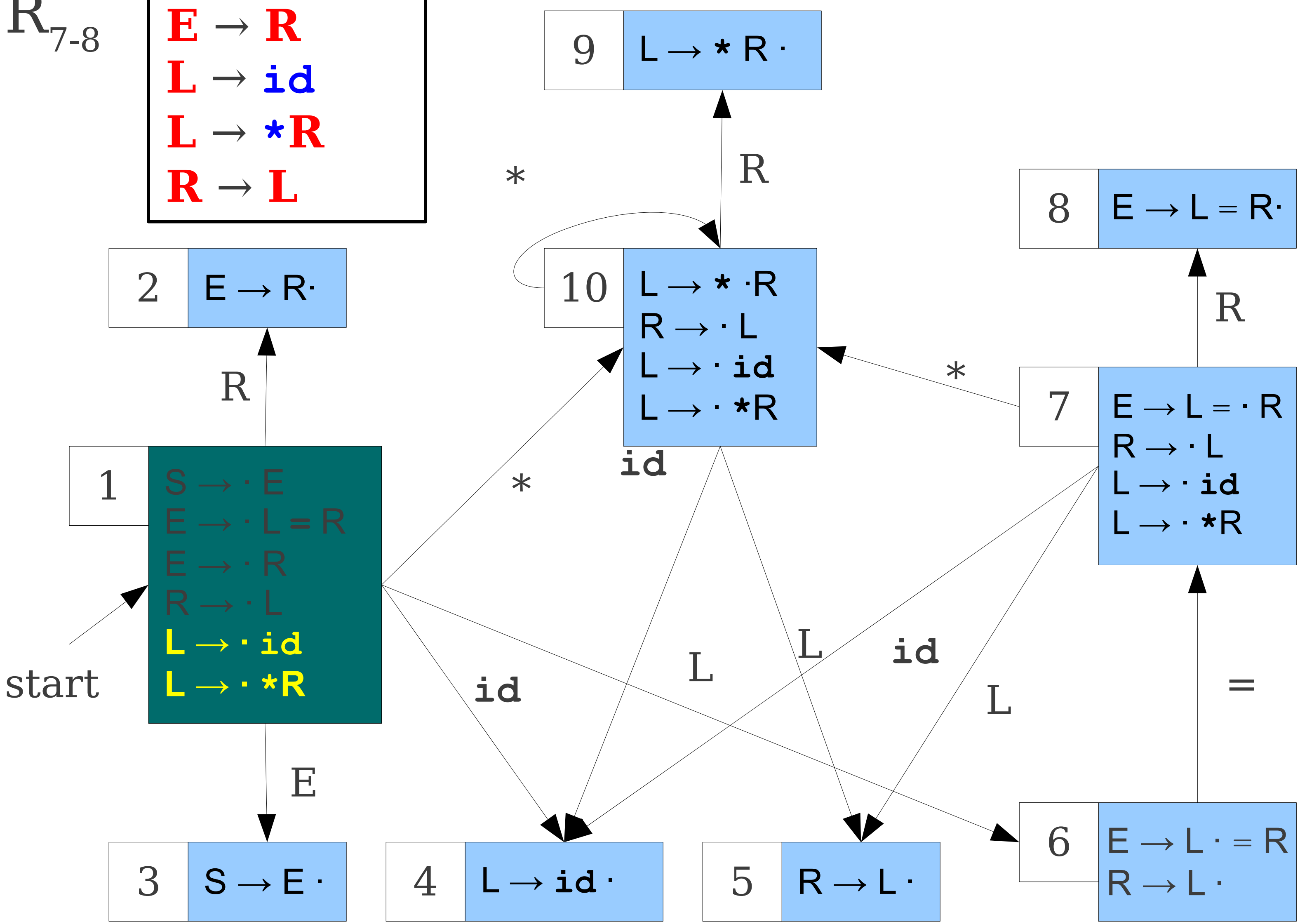
S → **E**
E → **L** = **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**



Augmenting the Grammar

$S_1 \rightarrow E_{1\text{-}3}$
$E_{1\text{-}3} \rightarrow L_{1\text{-}6} = R_{7\text{-}8}$
$E_{1\text{-}3} \rightarrow R_{1\text{-}2}$
$R_{1\text{-}2} \rightarrow L_{1\text{-}6}$
$L_{1\text{-}6} \rightarrow \textbf{id}$
$L_{1\text{-}6} \rightarrow * R_{10\text{-}9}$

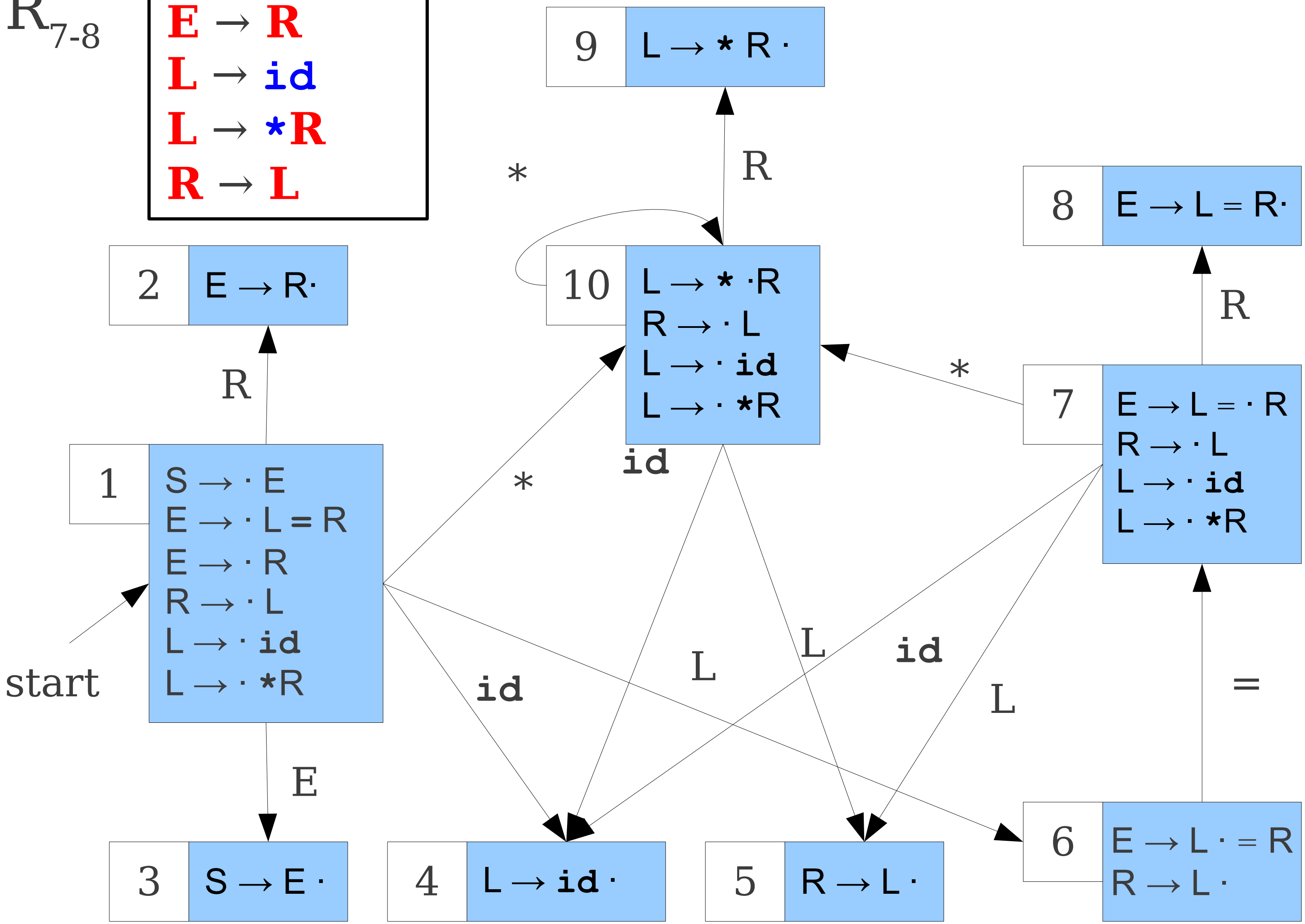
S → **E**
E → **L** = **R**
E → **R**
L → `id`
L → ***R**
R → **L**



Augmenting the Grammar

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

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



Augmenting the Grammar

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

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

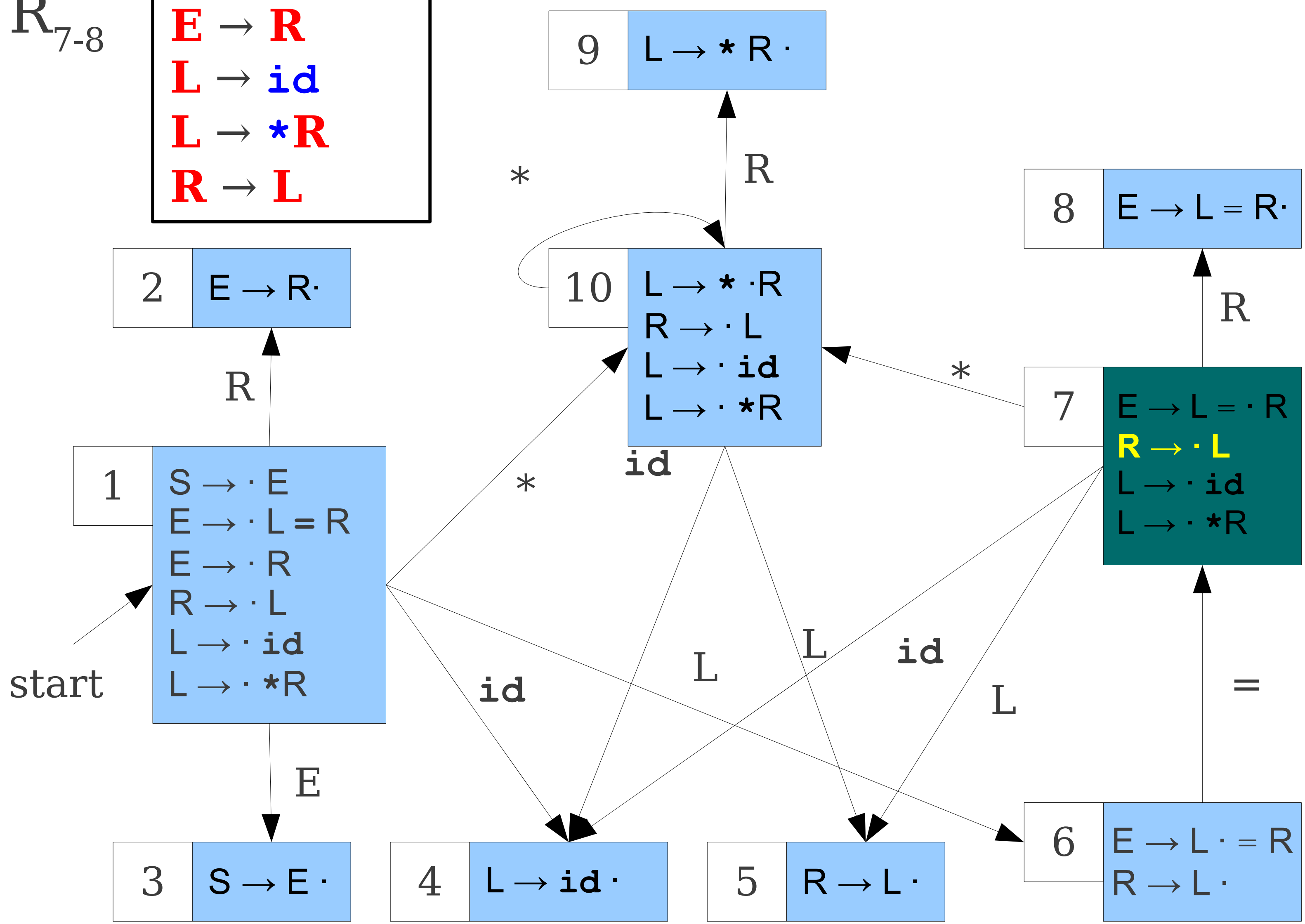
$E_{1\text{-}3} \rightarrow R_{1\text{-}2}$

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

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

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

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



Augmenting the Grammar

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

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

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

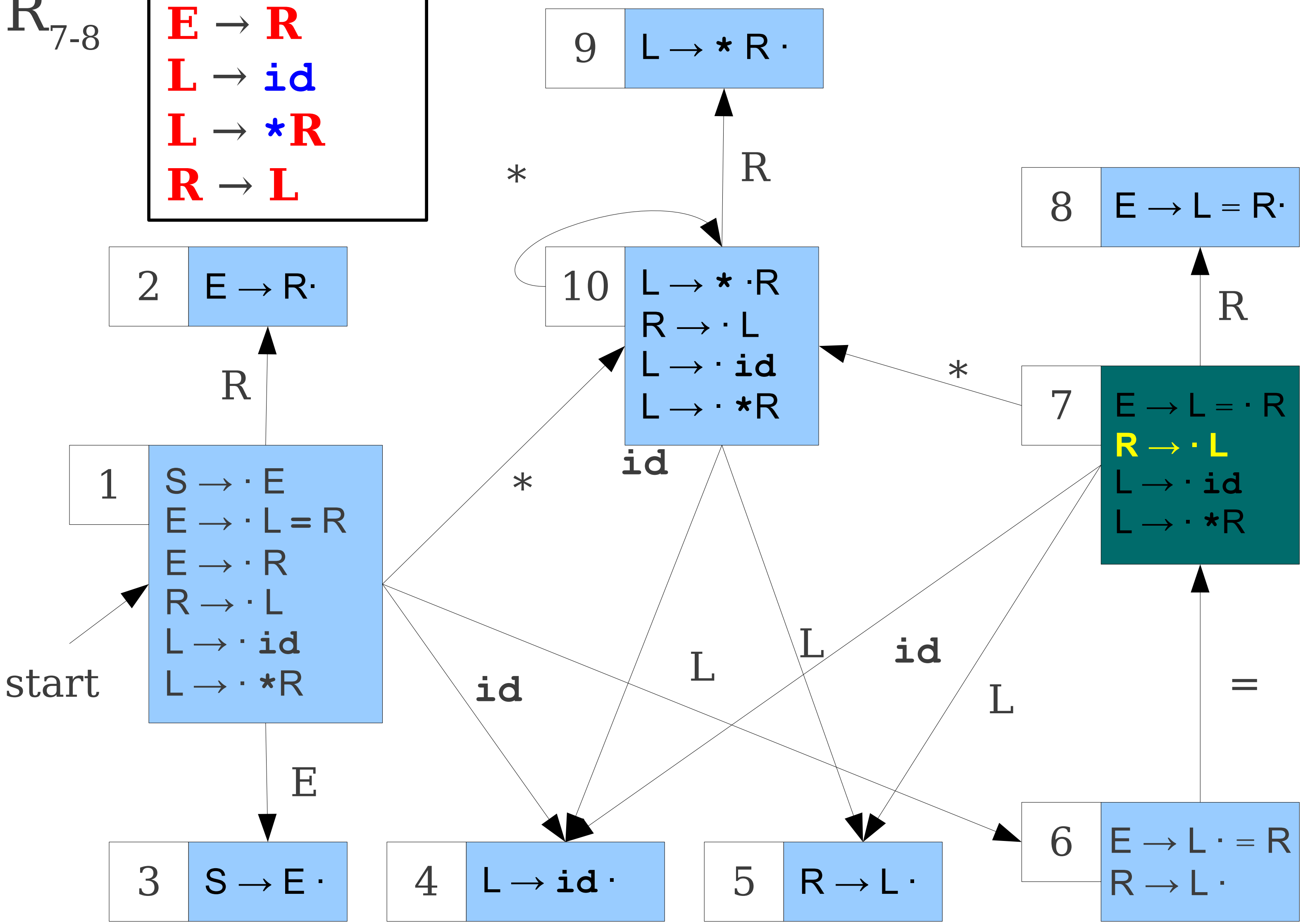
$R_{1\text{-}2} \to L_{1\text{-}6}$

$L_{1\text{-}6} \to \textbf{id}$

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

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

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



Augmenting the Grammar

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

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

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

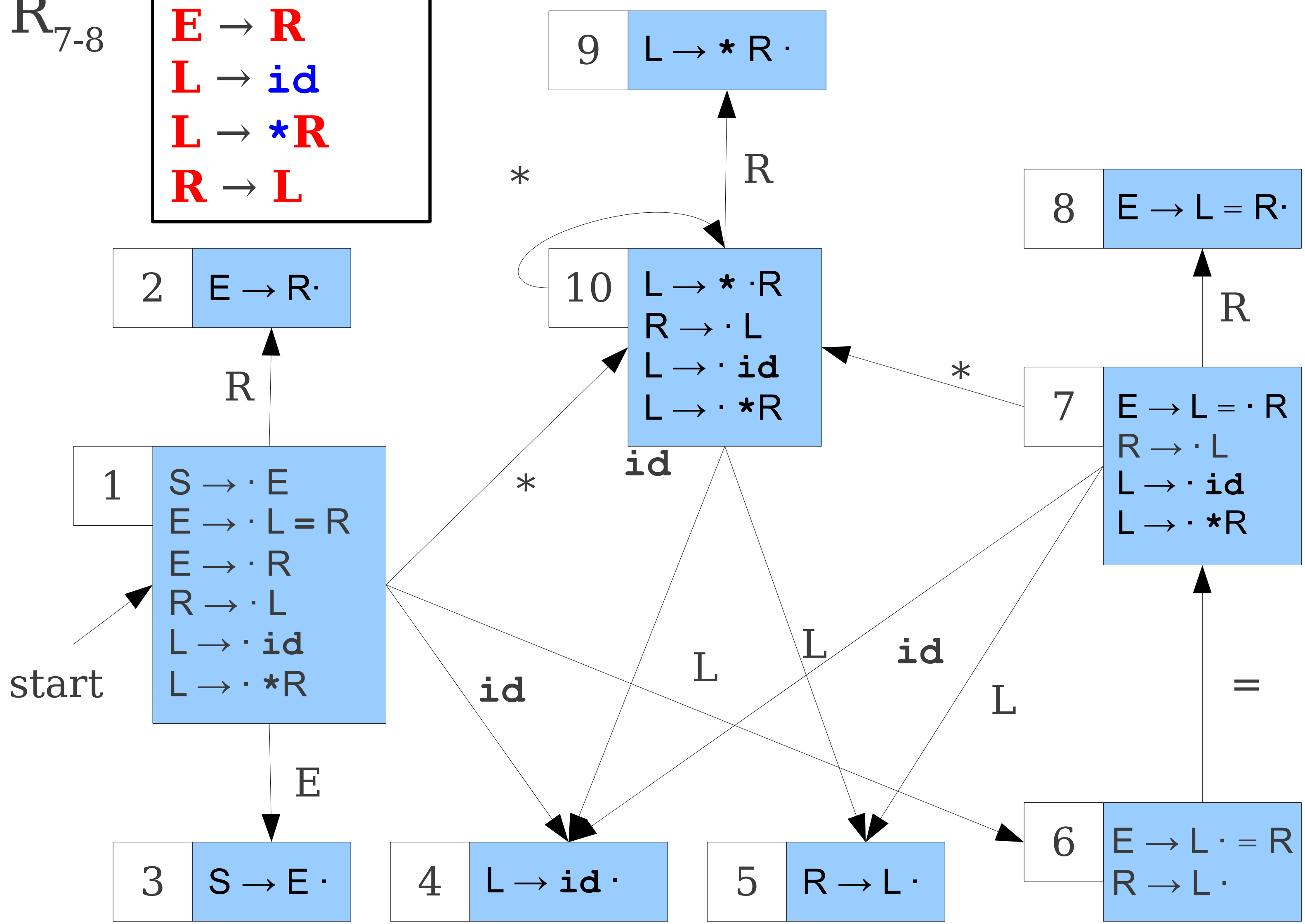
$R_{1\text{-}2} \rightarrow L_{1\text{-}6}$

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

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

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

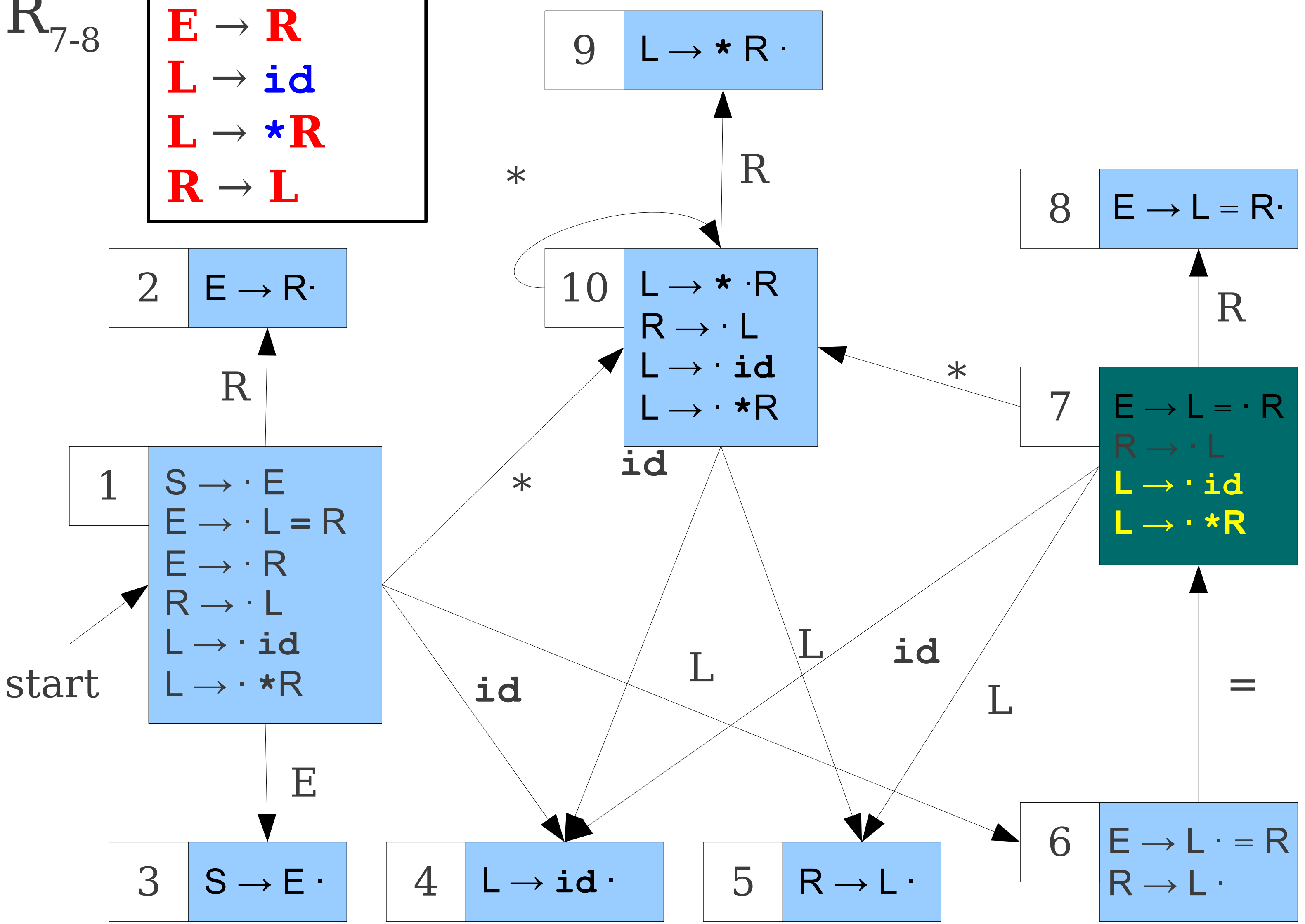
S → **E**
E → **L = R**
E → **R**
L → **id**
L → ***R**
R → **L**



Augmenting the Grammar

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

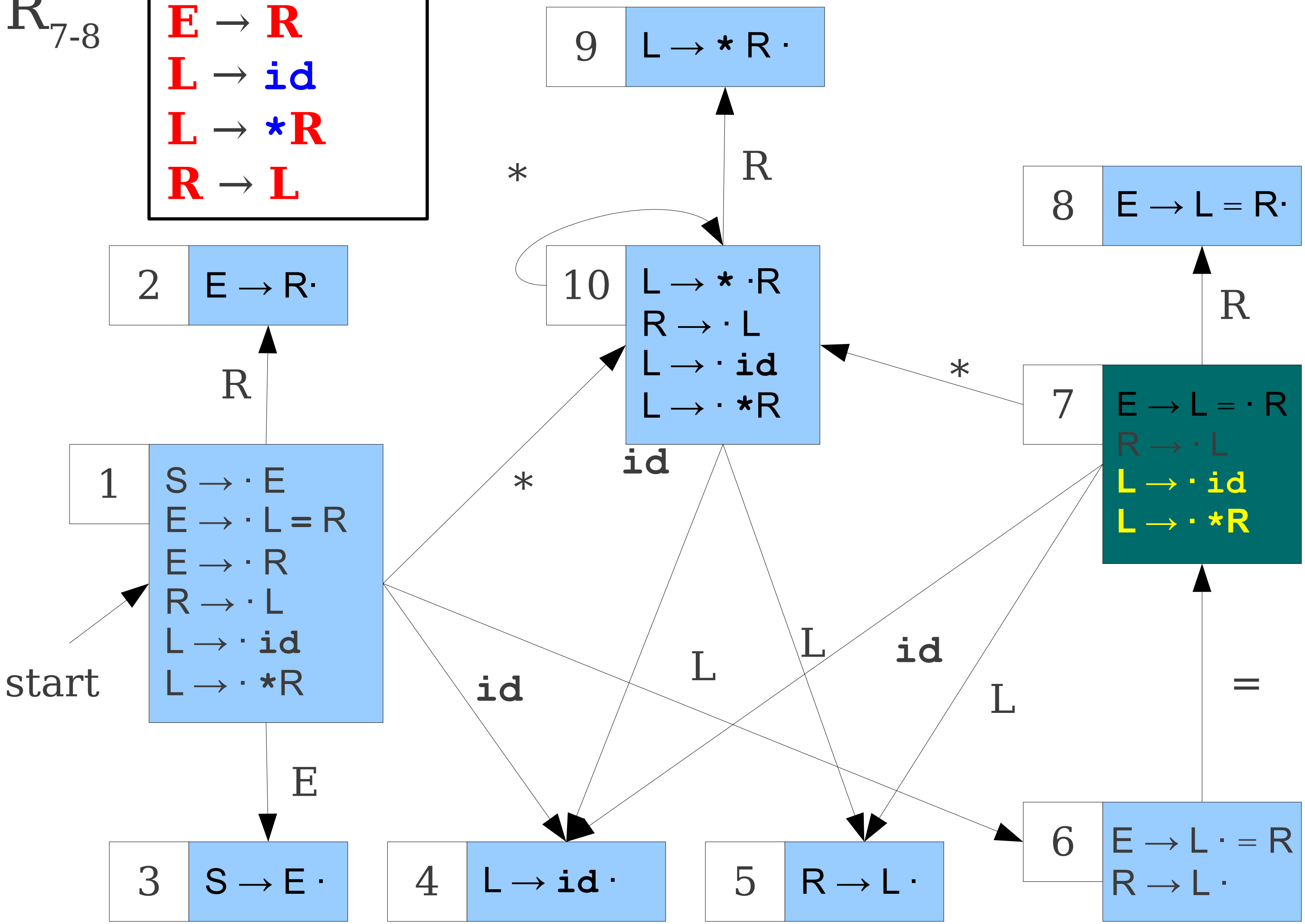
S → **E**
E → **L** = **R**
E → **R**
L → `id`
L → *__R__
R → **L**



Augmenting the Grammar

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

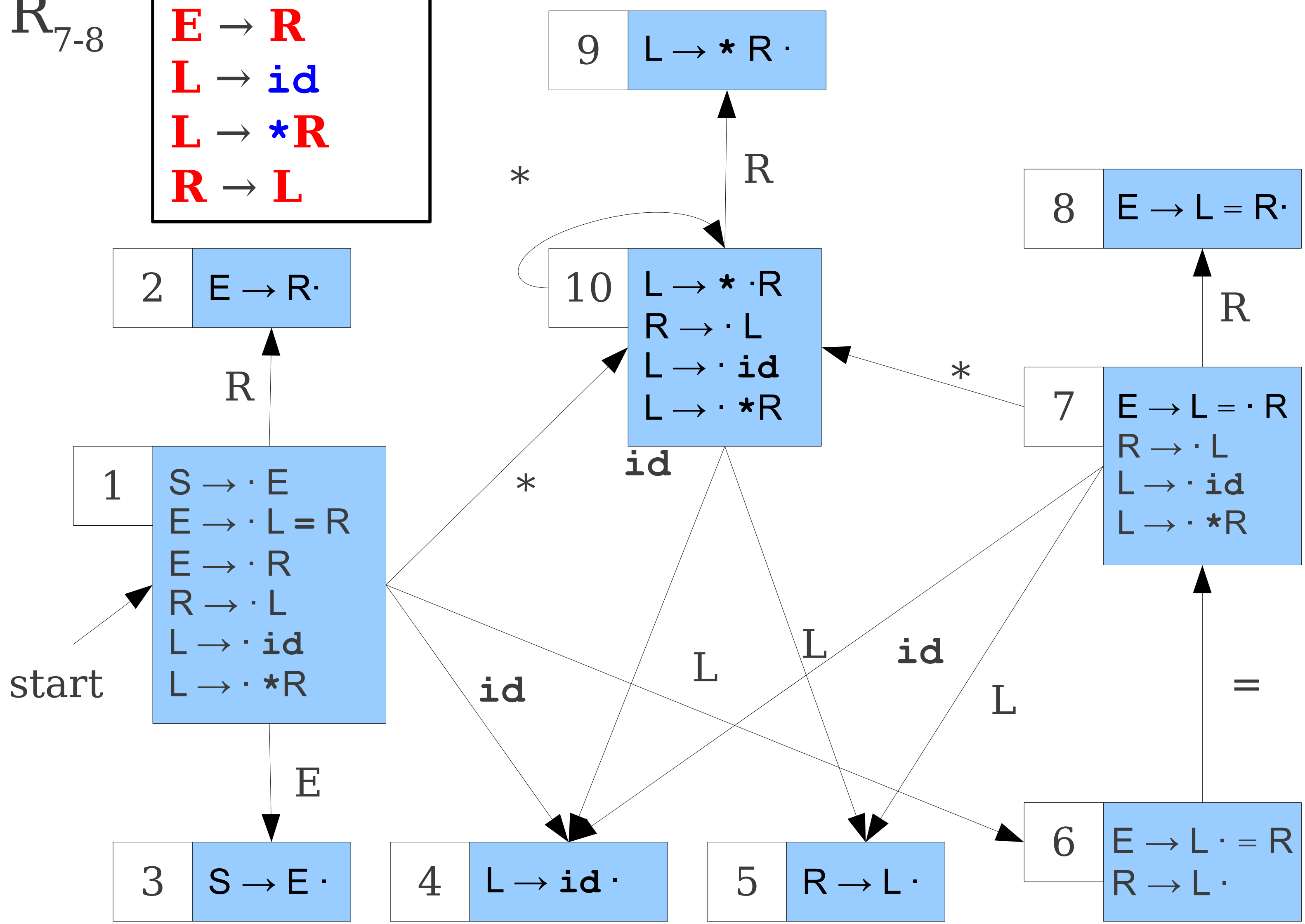
S → **E**
E → **L** = **R**
E → **R**
L → id
L → ***R**
R → **L**



Augmenting the Grammar

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

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



Augmenting the Grammar

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

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

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

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

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

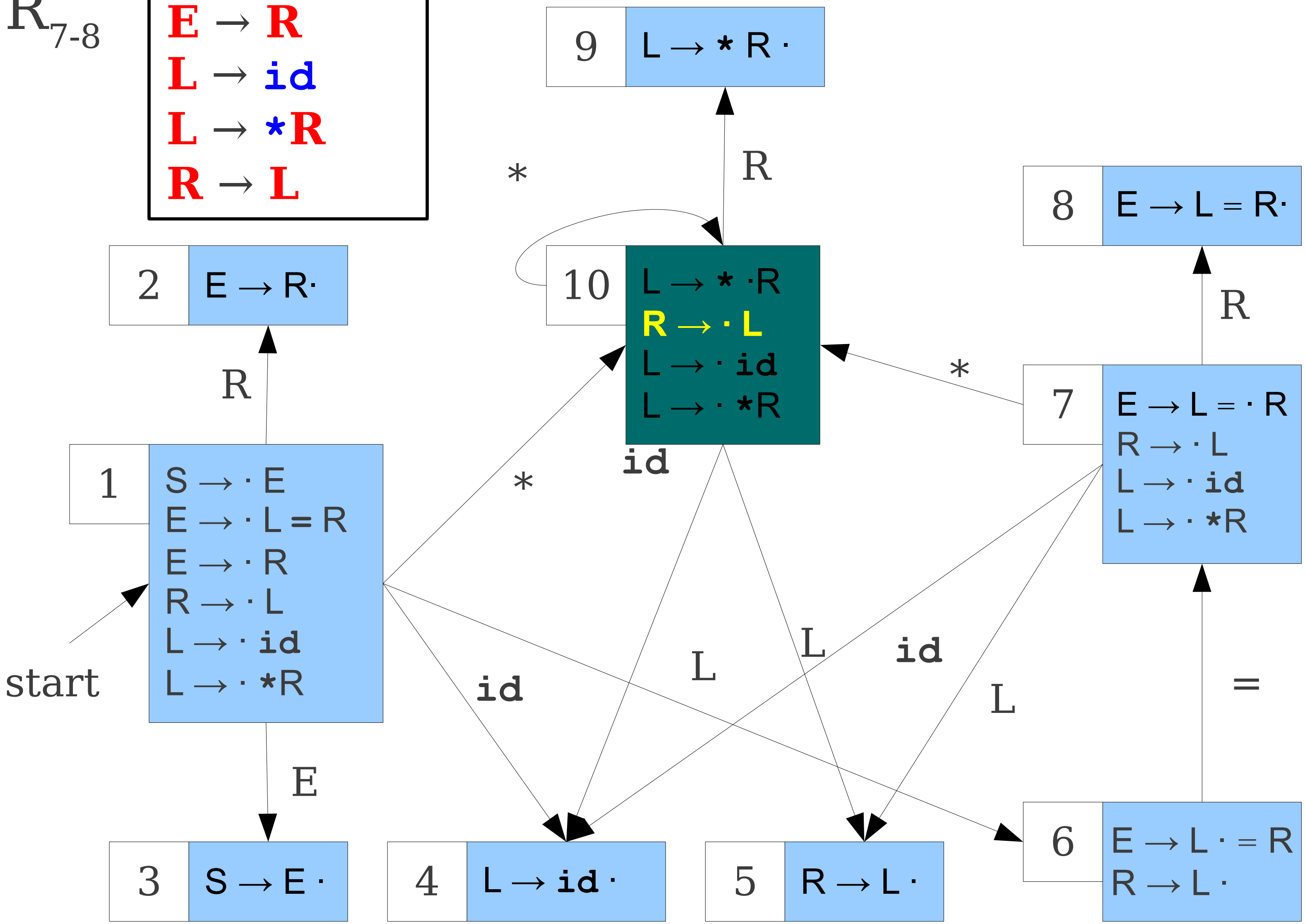
$L_{1\text{-}6} \rightarrow * R_{10\text{-}9}$

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

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

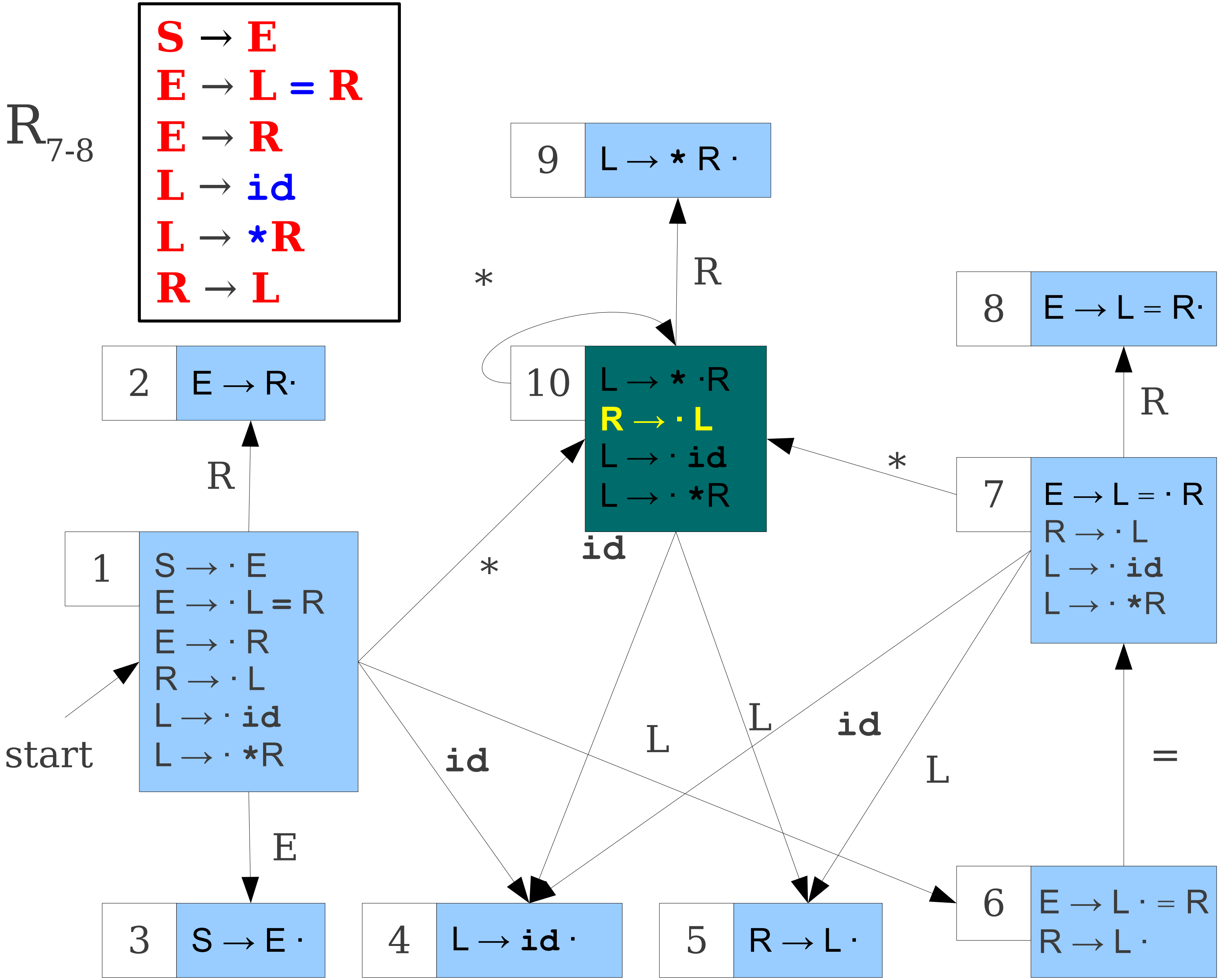
$L_{7\text{-}5} \rightarrow *R_{10\text{-}9}$

$S \rightarrow E$
$E \rightarrow L = R$
$E \rightarrow R$
$L \rightarrow \texttt{id}$
$L \rightarrow *R$
$R \rightarrow L$



Augmenting the Grammar

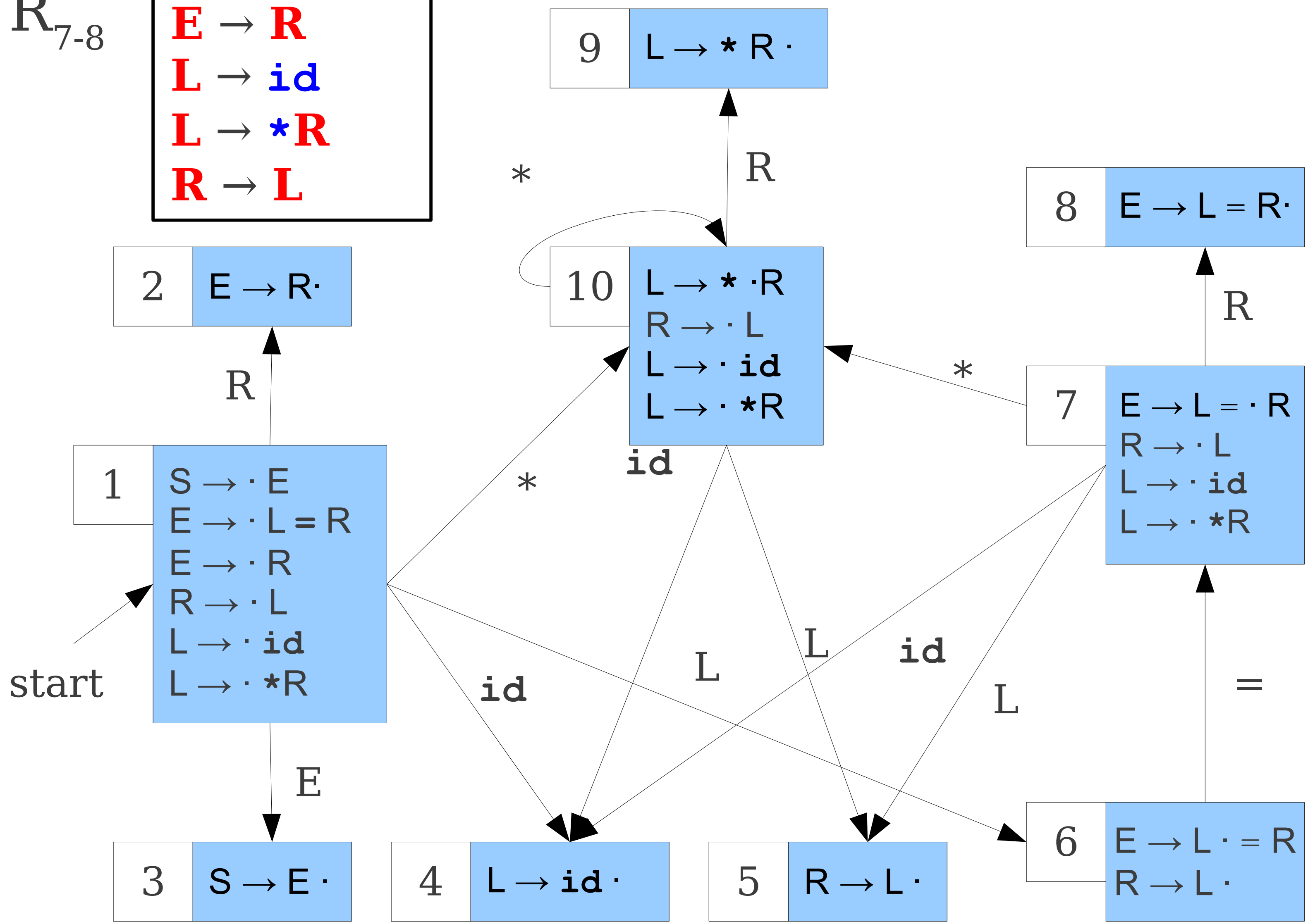
$S_1 \rightarrow E_{1\text{-}3}$
$E_{1\text{-}3} \rightarrow L_{1\text{-}6} = R_{7\text{-}8}$
$E_{1\text{-}3} \rightarrow R_{1\text{-}2}$
$R_{1\text{-}2} \rightarrow L_{1\text{-}6}$
$L_{1\text{-}6} \rightarrow$ **id**
$L_{1\text{-}6} \rightarrow * R_{10\text{-}9}$
$R_{7\text{-}8} \rightarrow L_{7\text{-}5}$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



Augmenting the Grammar

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

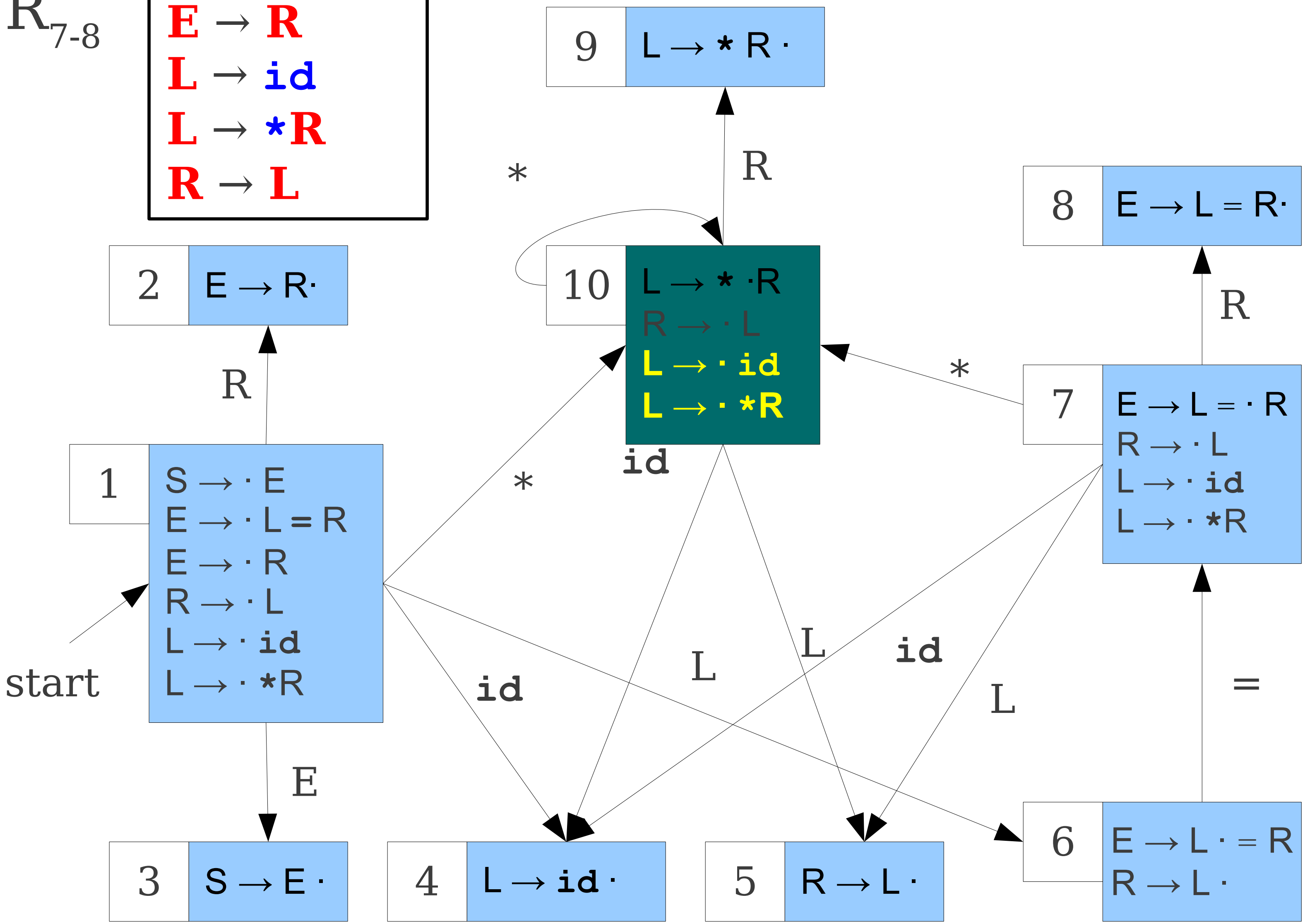
$$\begin{aligned}
&\mathbf{S} \to \mathbf{E}\\
&\mathbf{E} \to \mathbf{L} = \mathbf{R}\\
&\mathbf{E} \to \mathbf{R}\\
&\mathbf{L} \to \texttt{id}\\
&\mathbf{L} \to *\mathbf{R}\\
&\mathbf{R} \to \mathbf{L}
\end{aligned}$$



Augmenting the Grammar

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

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



Augmenting the Grammar

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

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

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

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

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

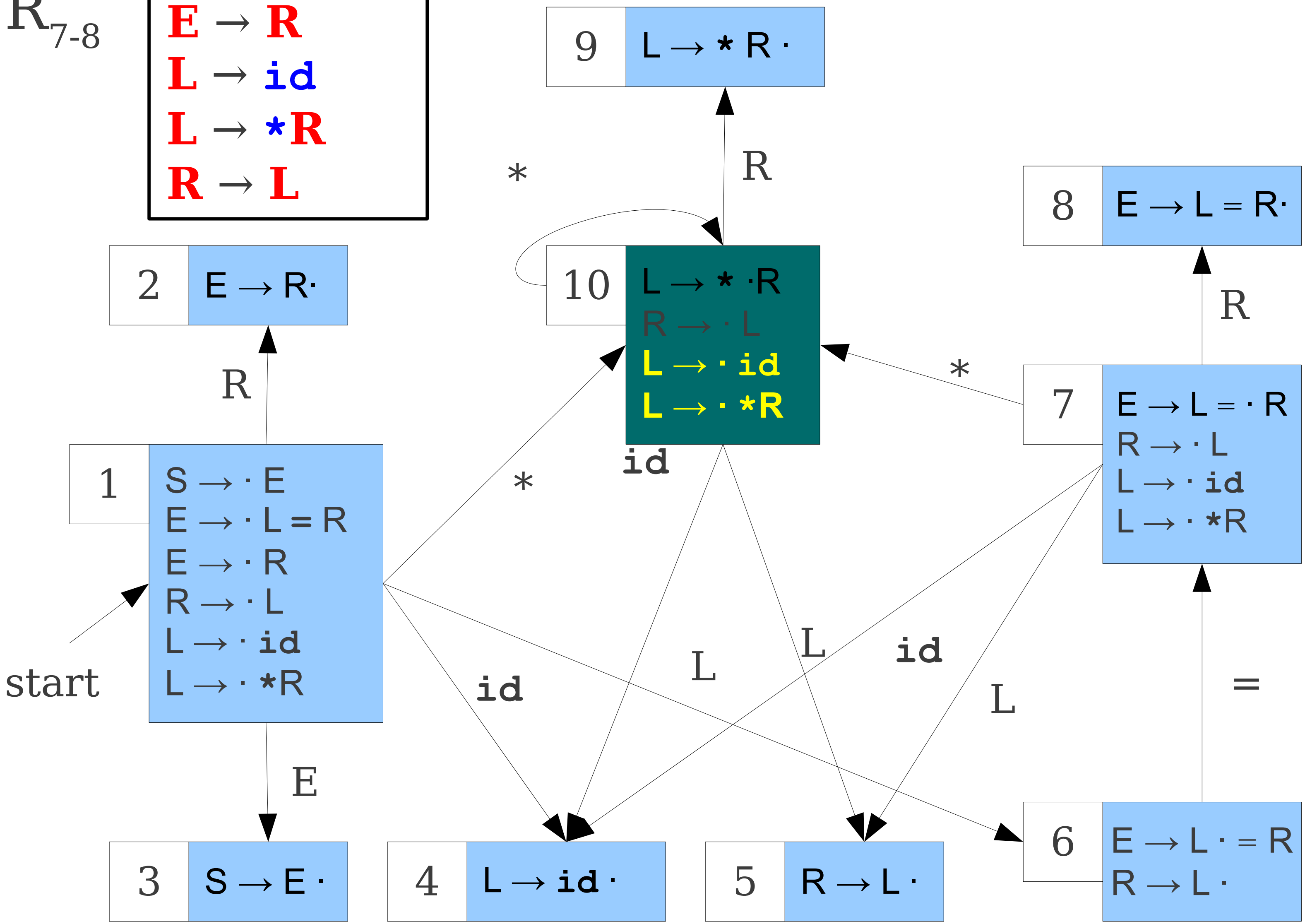
$L_{7\text{-}5} \rightarrow *R_{10\text{-}9}$

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

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

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

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



Augmenting the Grammar

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

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

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

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

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

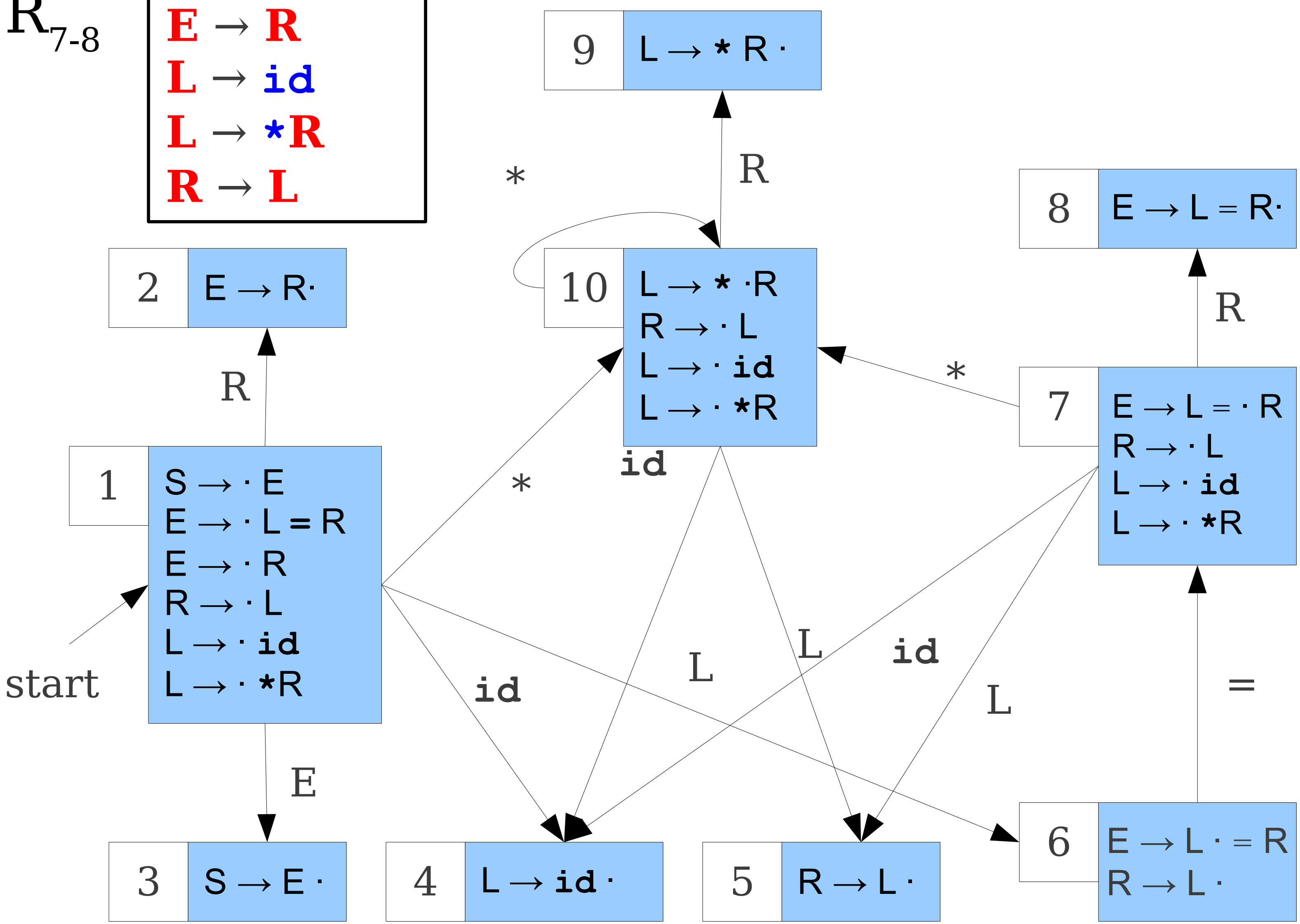
$L_{7\text{-}5} \rightarrow *R_{10\text{-}9}$

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

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

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

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



Constructing Augmented Grammars

- For each item $\mathbf{A} \rightarrow \cdot\boldsymbol{\omega}$ in some state q:
  - Trace out the path $\boldsymbol{\omega}$ takes through the LR(0) automaton starting at q.
  - Replace each nonterminal in $\boldsymbol{\omega}$ with a nonterminal annotated with the state transitioned to and from by the edge labeled with that nonterminal.
  - Replace $\mathbf{A}$ with a nonterminal annotated with the start and end state of the transition on $\mathbf{A}$ out of q.
- Result is a larger grammar with more precise productions.

Why is this Grammar Useful?

- At a high-level, separates out the nonterminals based on their context.
- This makes the FOLLOW sets more precise for their nonterminals.
- In fact, the FOLLOW sets are surprisingly precise.

Augmented FOLLOW 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`**

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

Augmented FOLLOW 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**

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

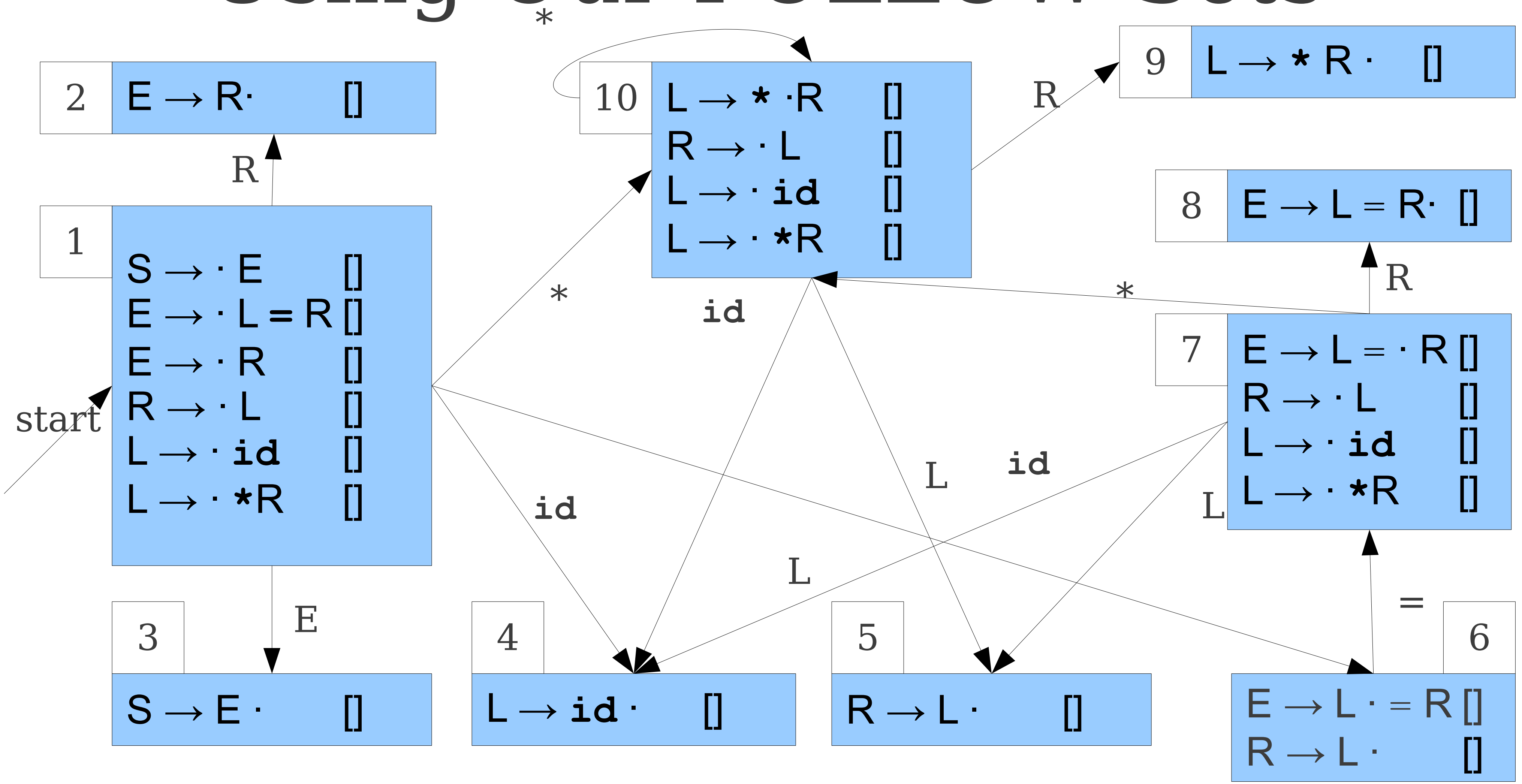
$R_{10\text{-}9} \rightarrow L_{10\text{-}5}$

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

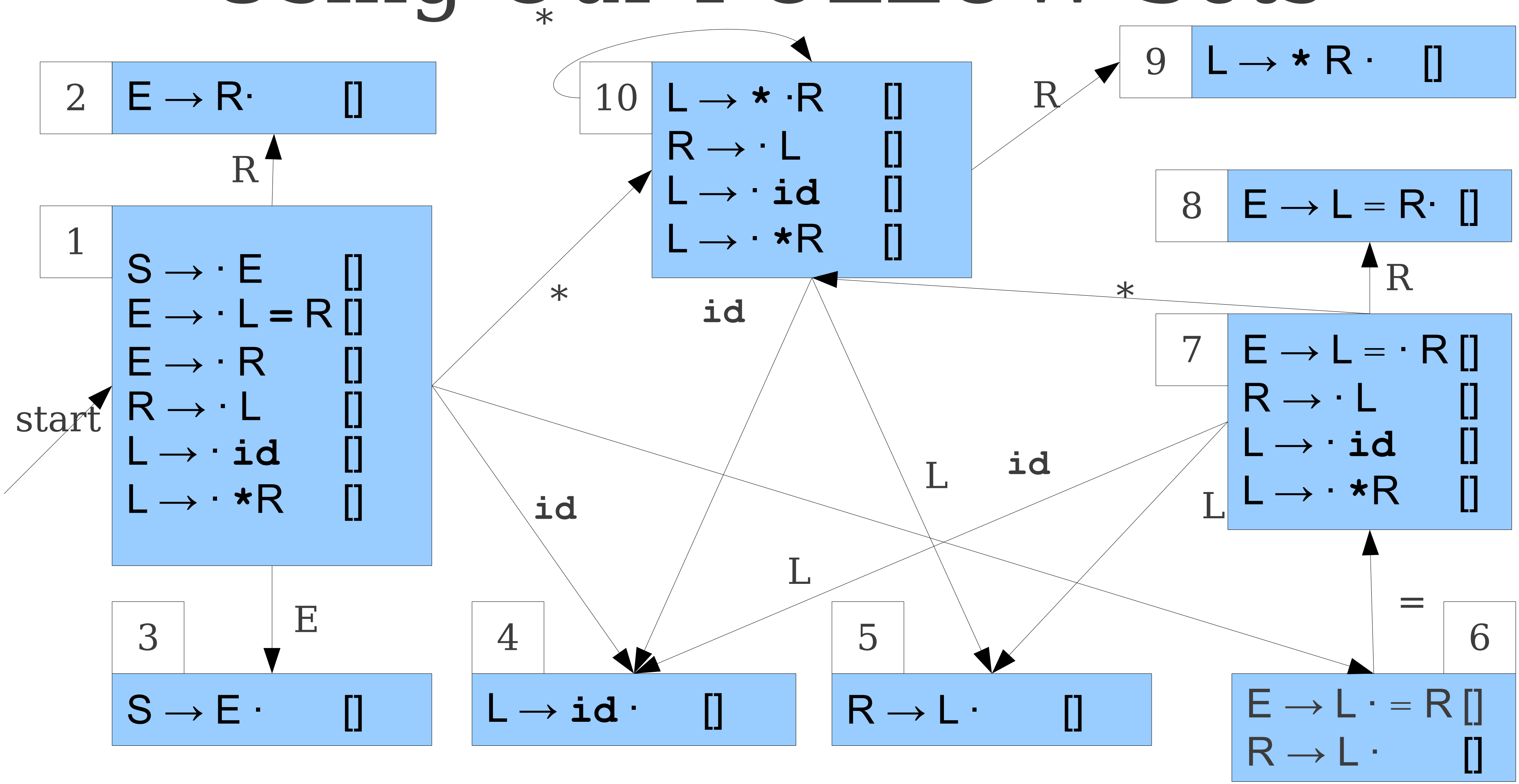
$L_{10\text{-}5} \rightarrow *R_{10\text{-}9}$

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

Using Our FOLLOW Sets

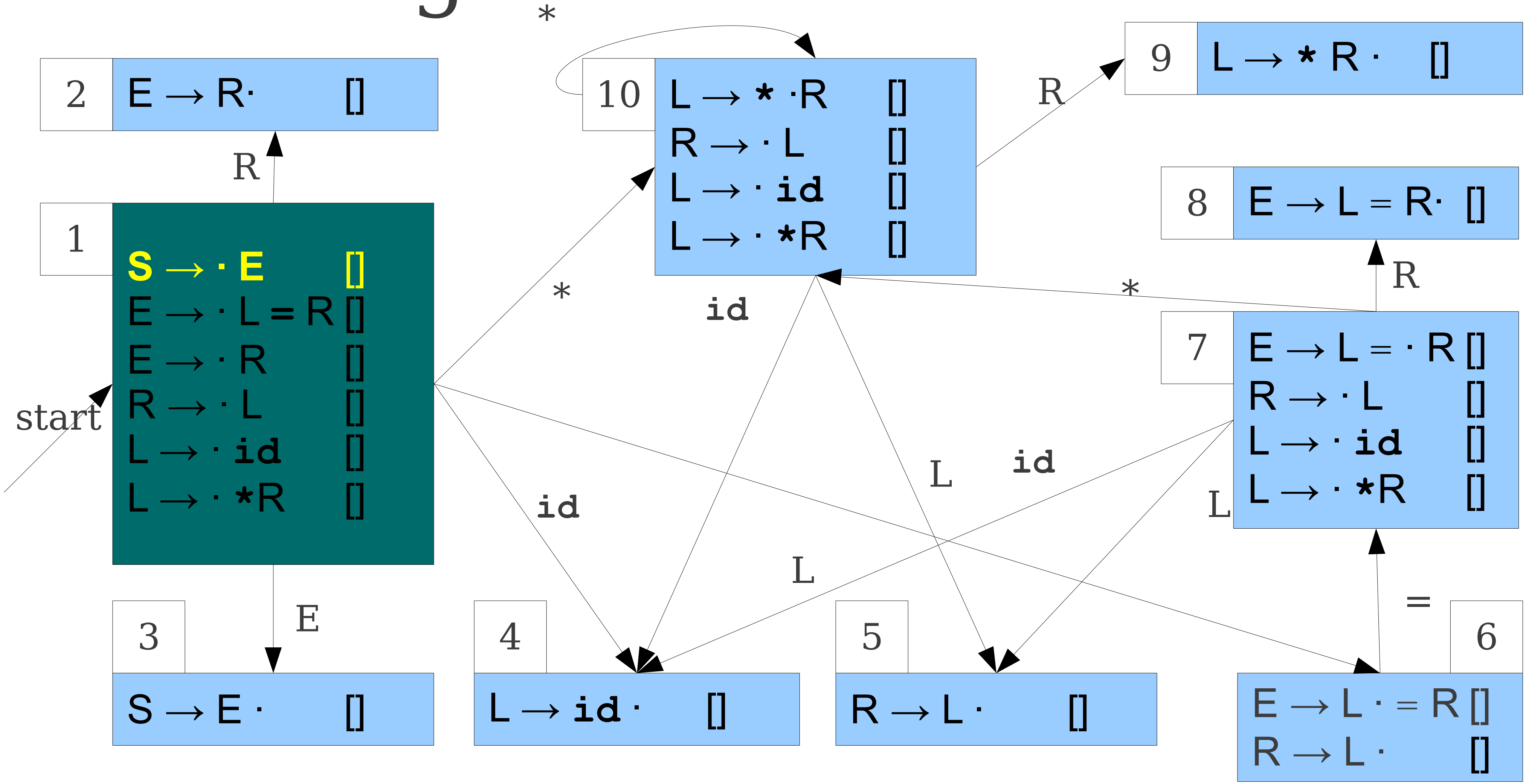


Using Our FOLLOW Sets



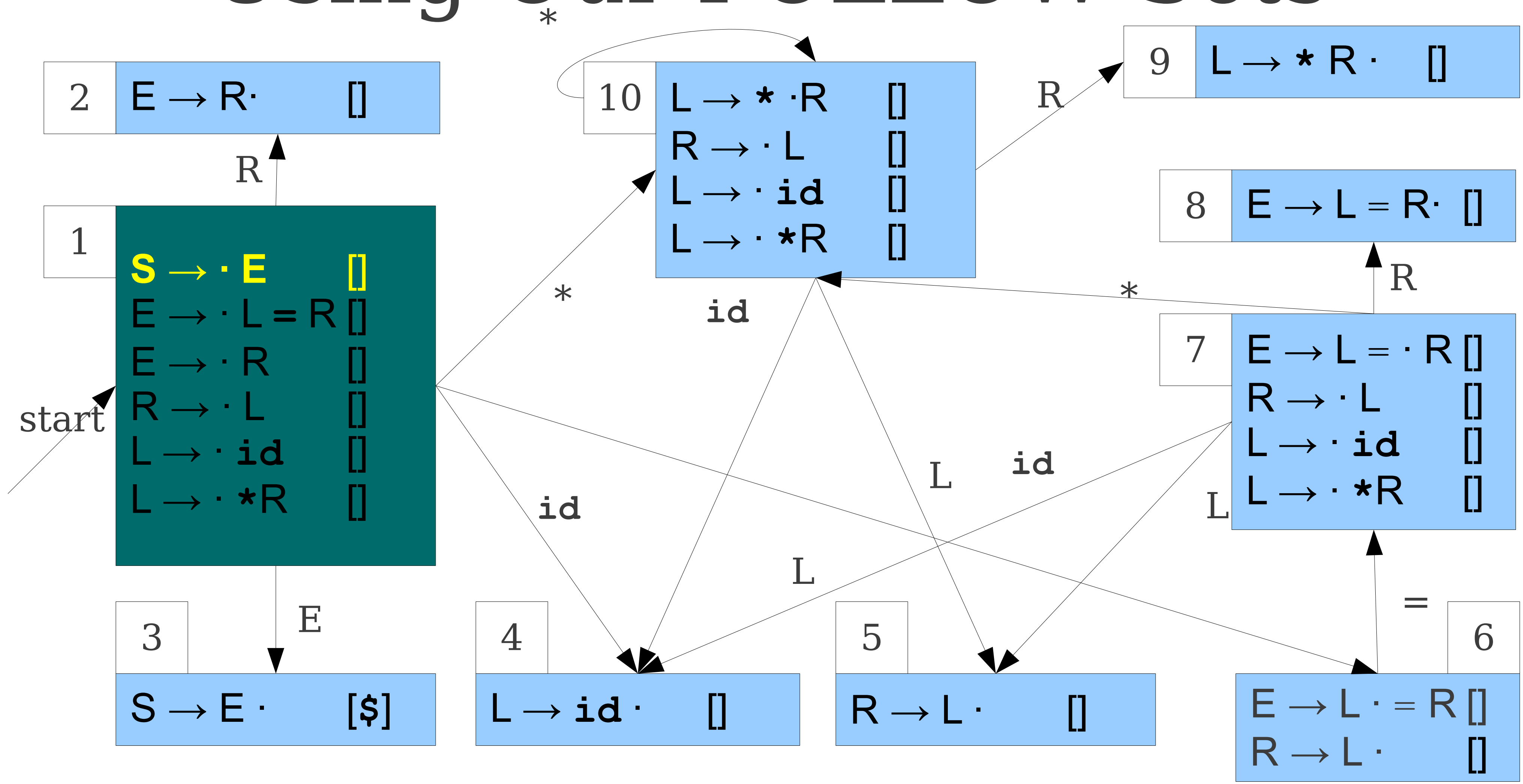
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



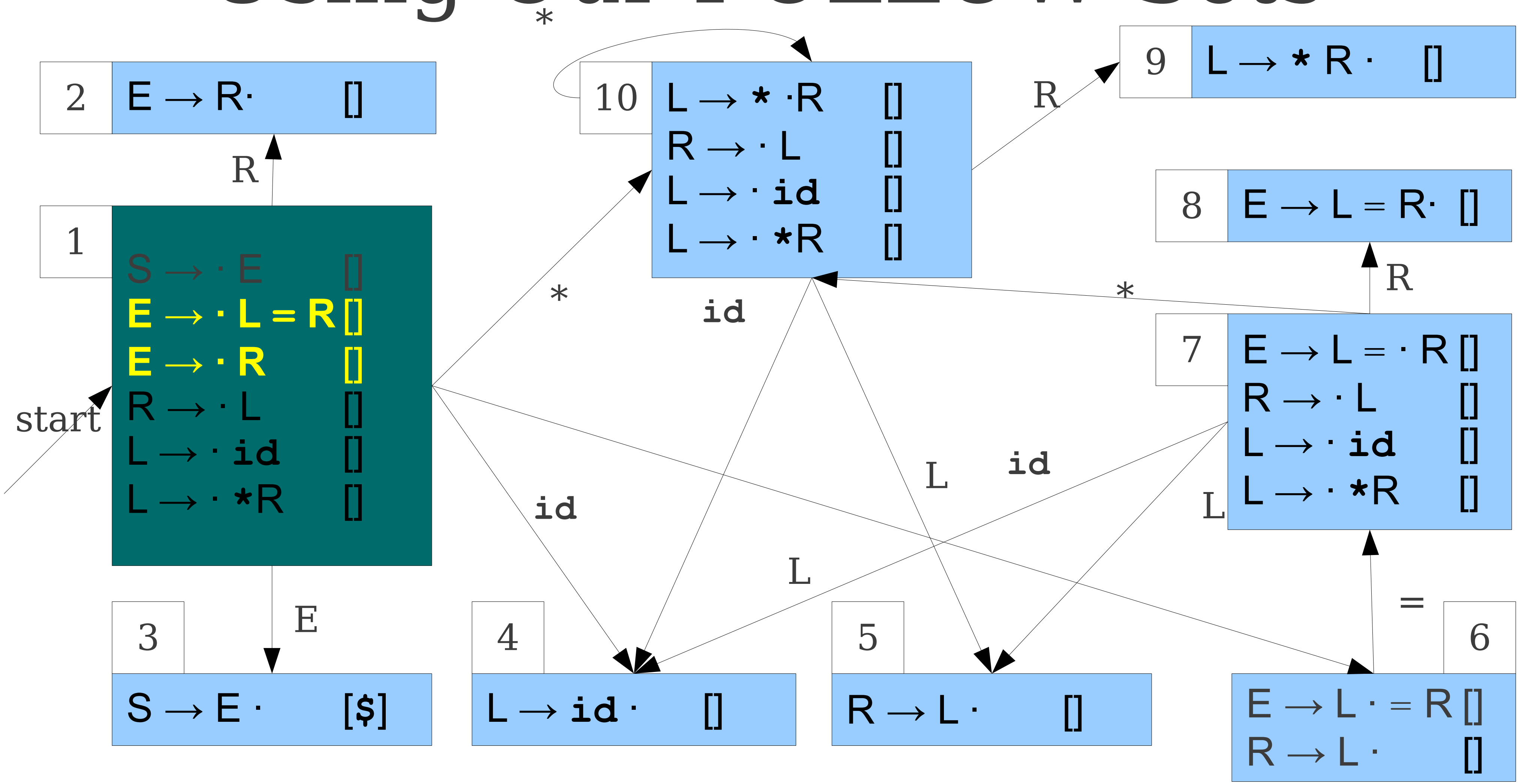
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



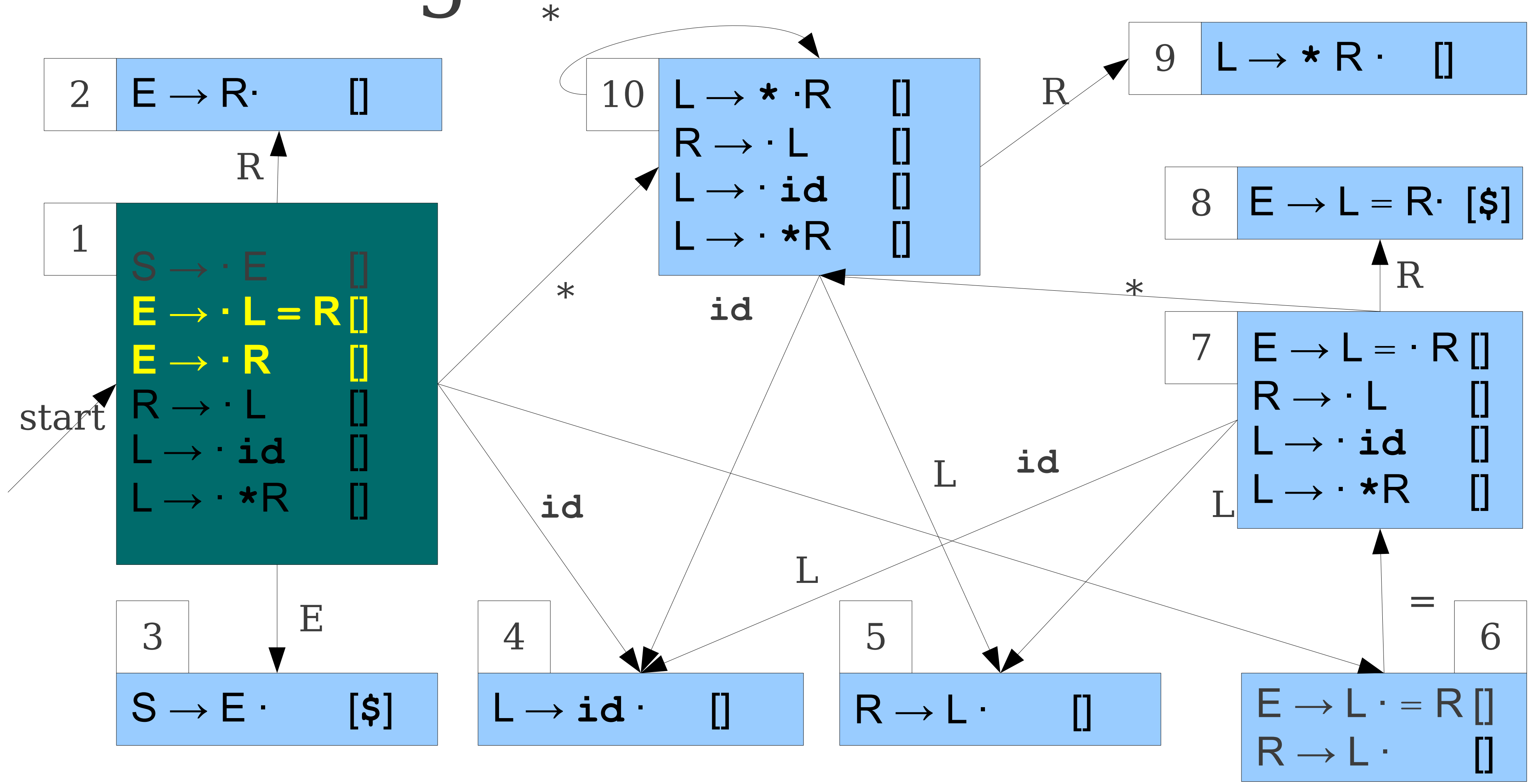
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



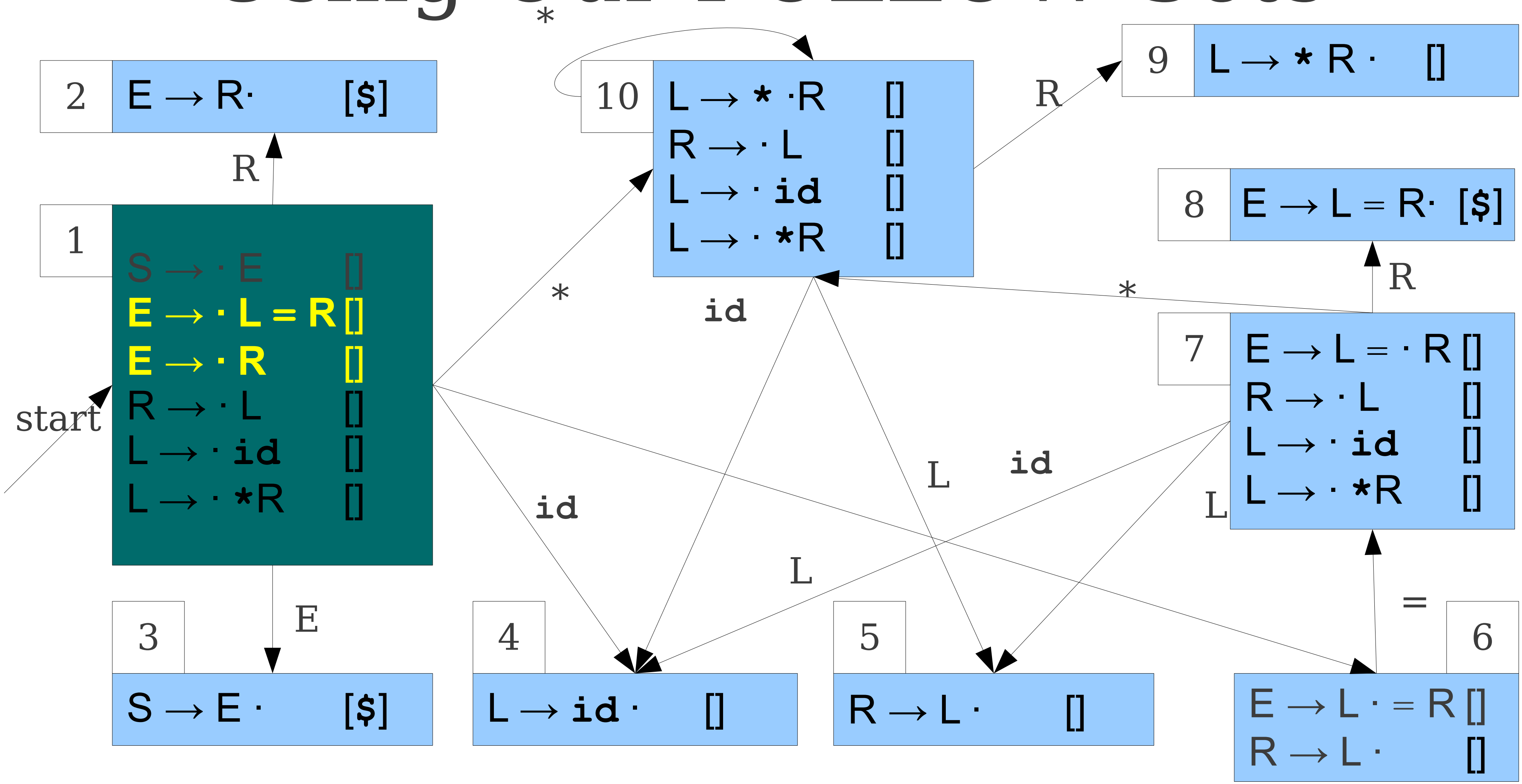
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



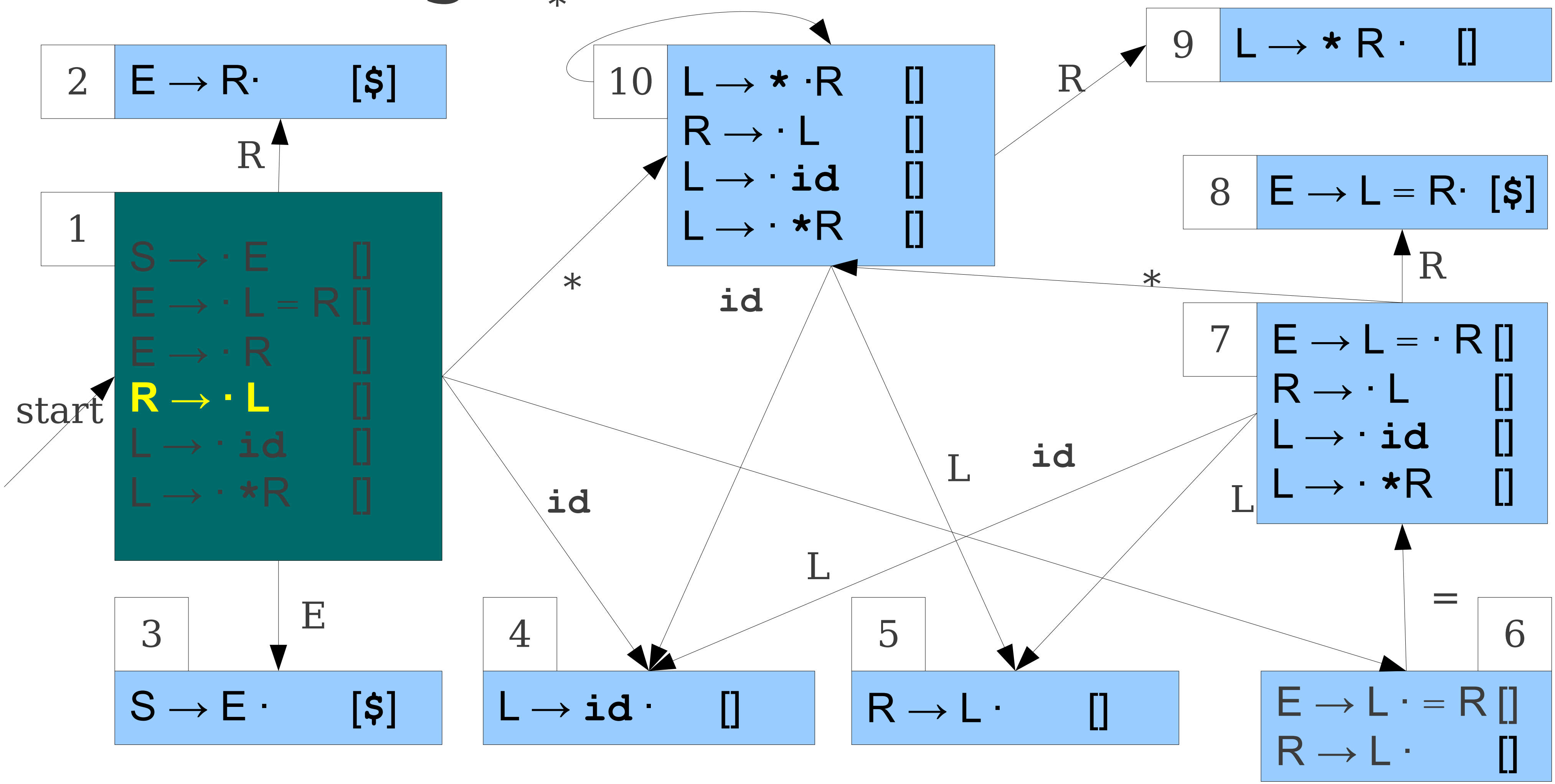
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



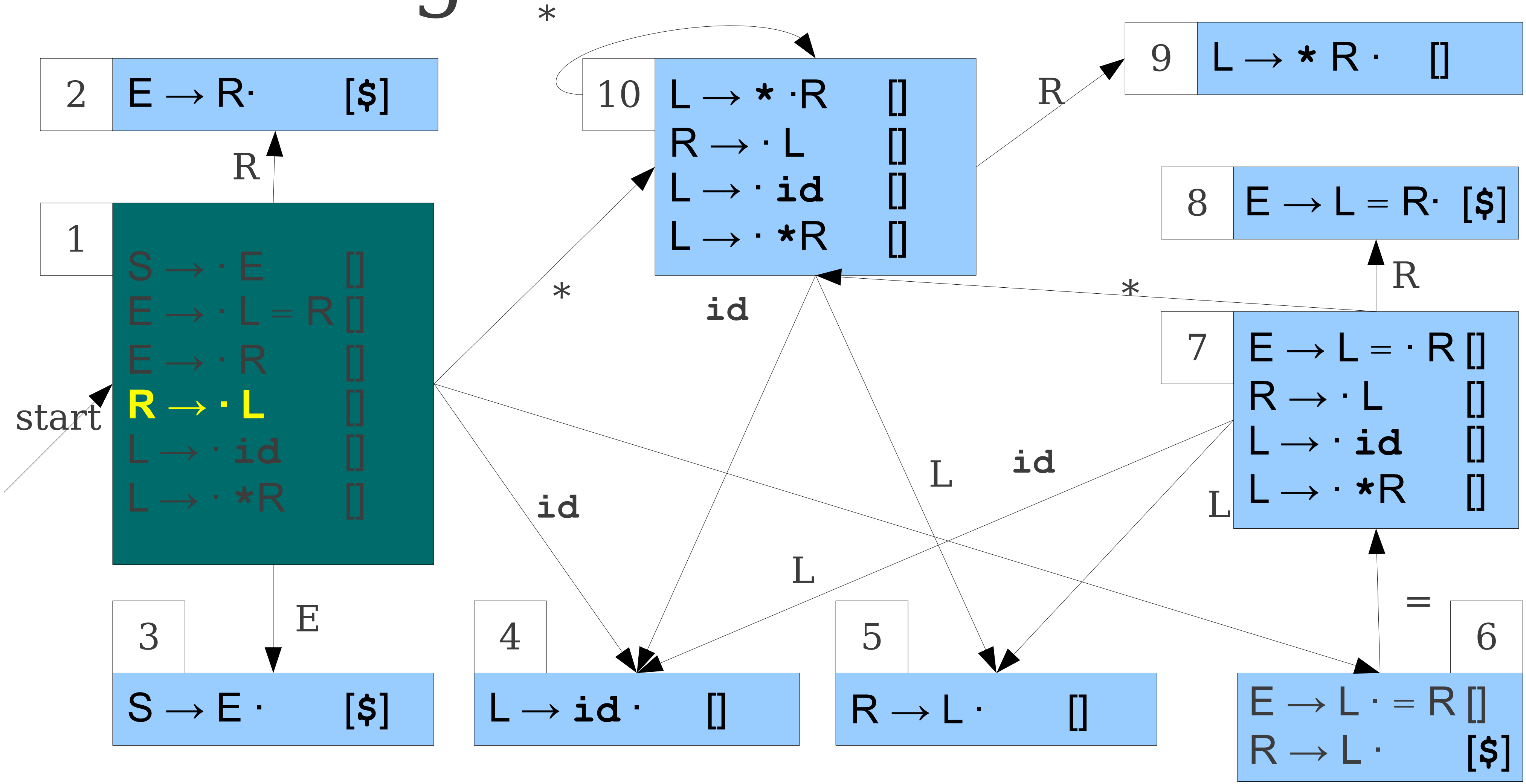
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



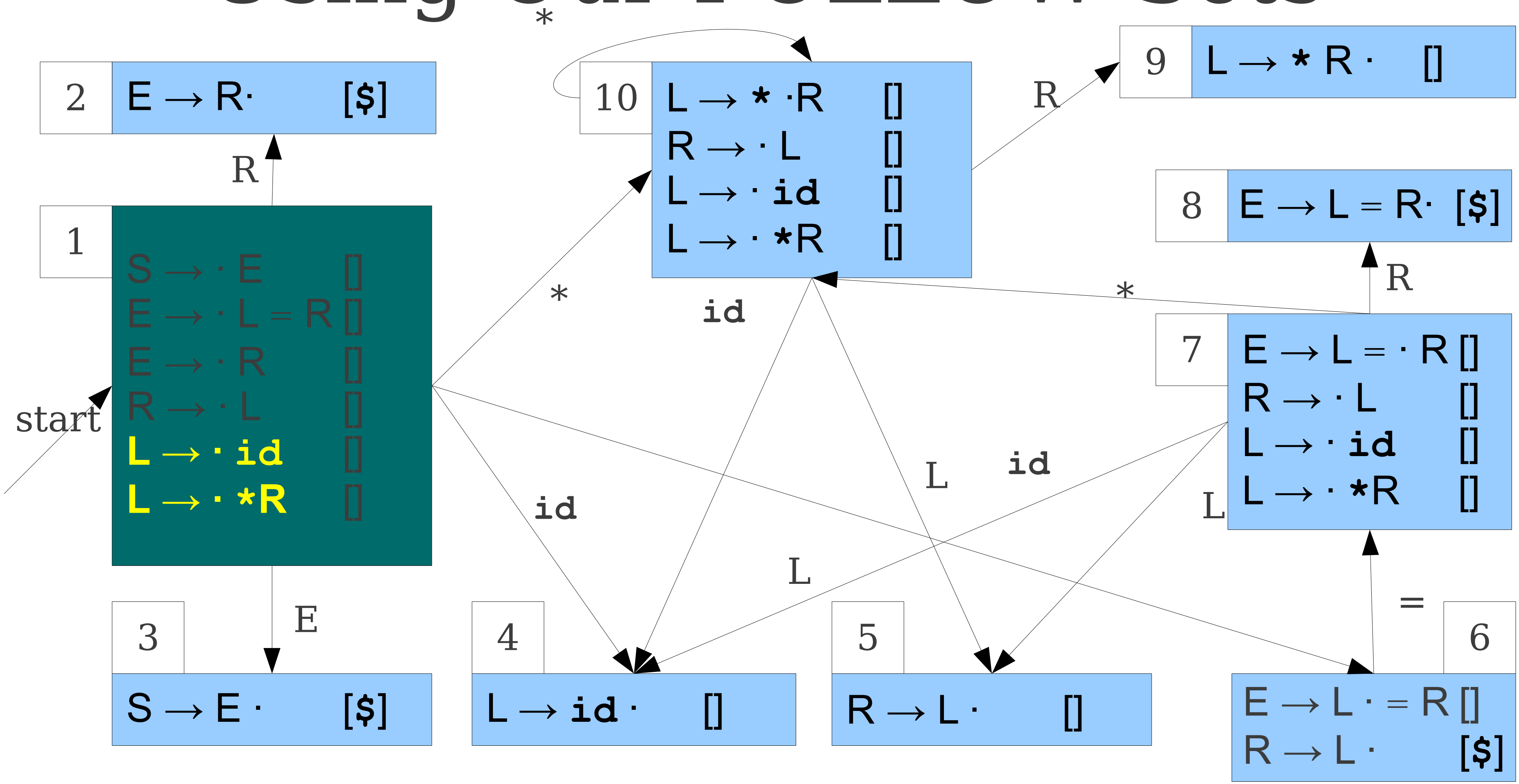
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



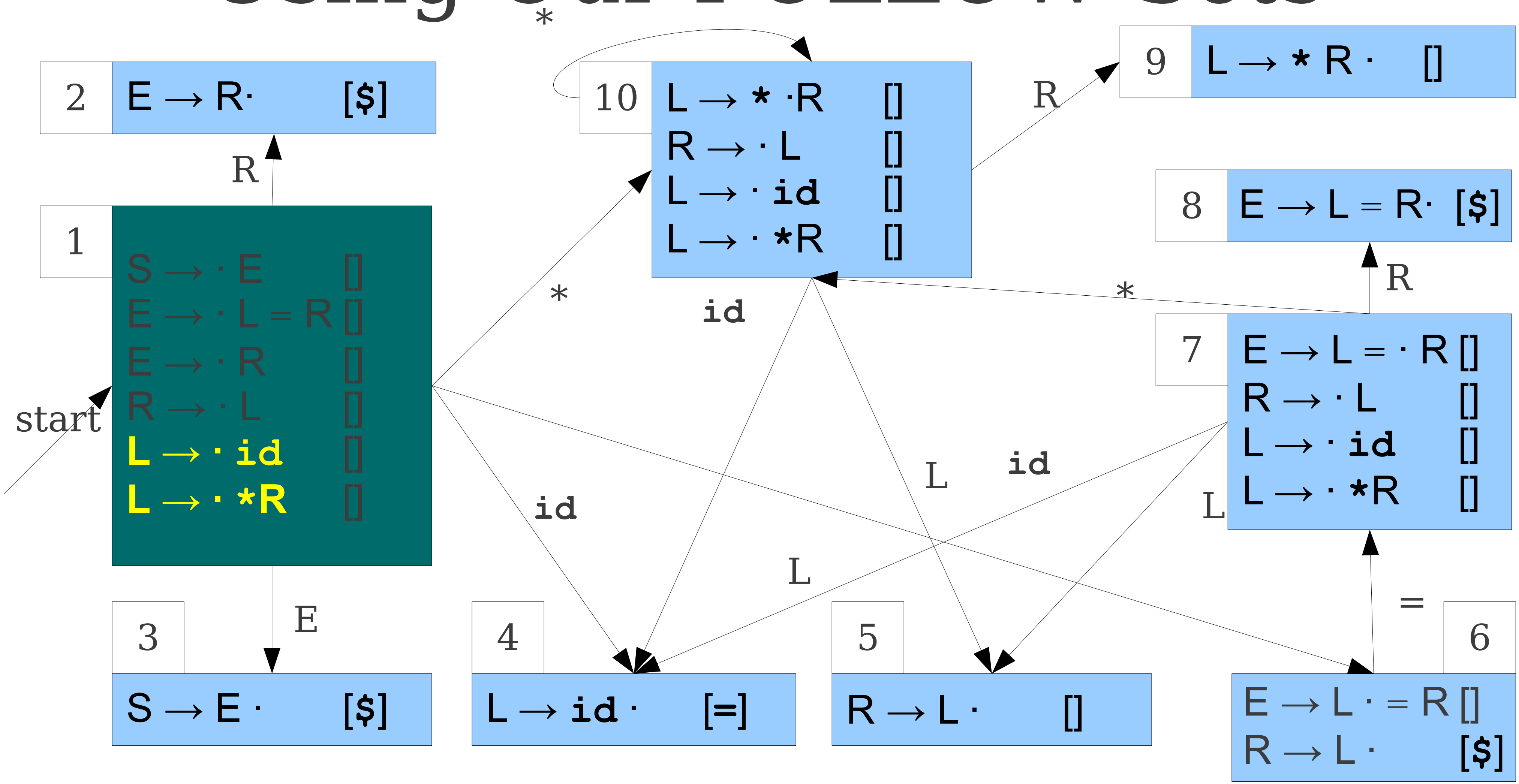
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



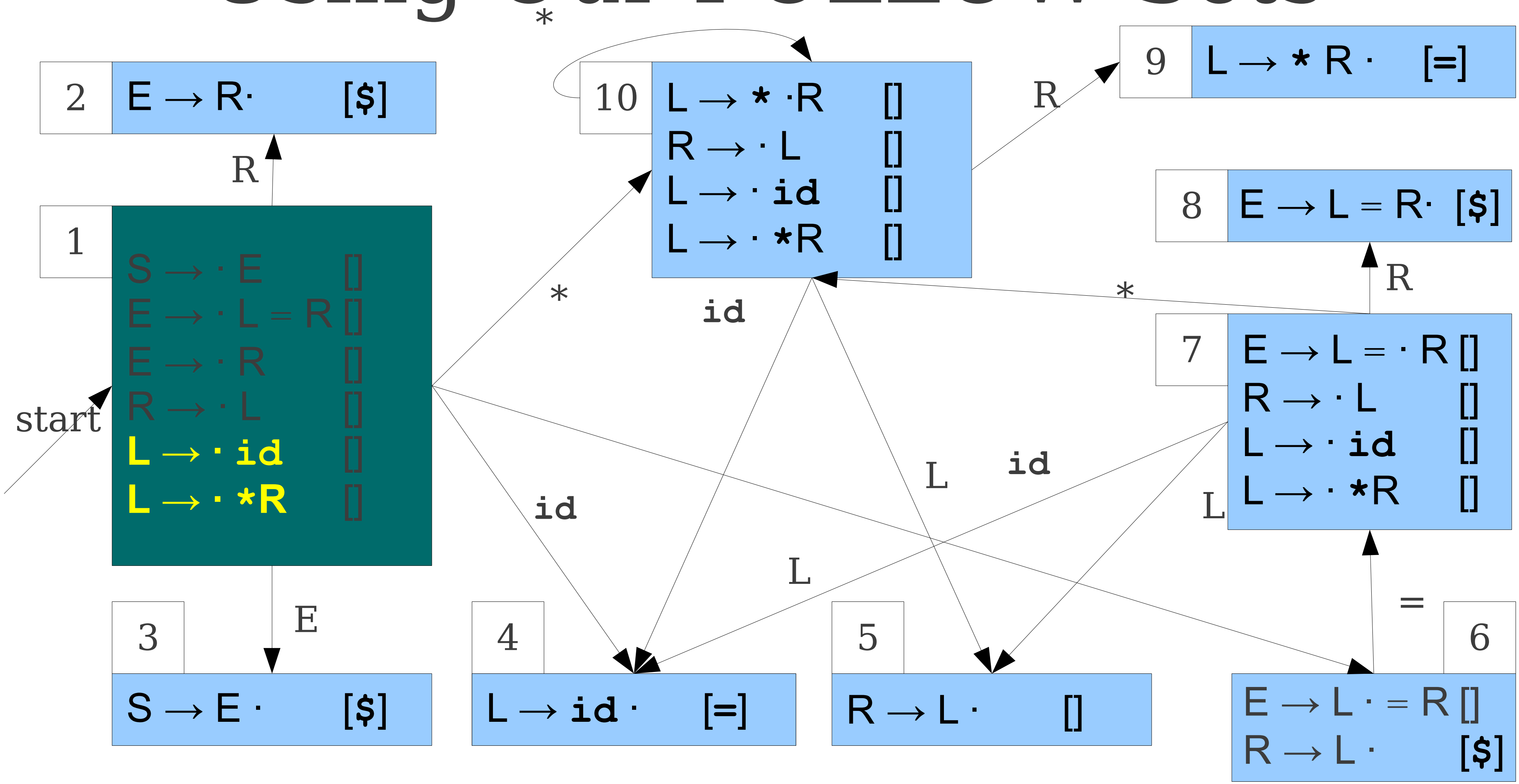
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



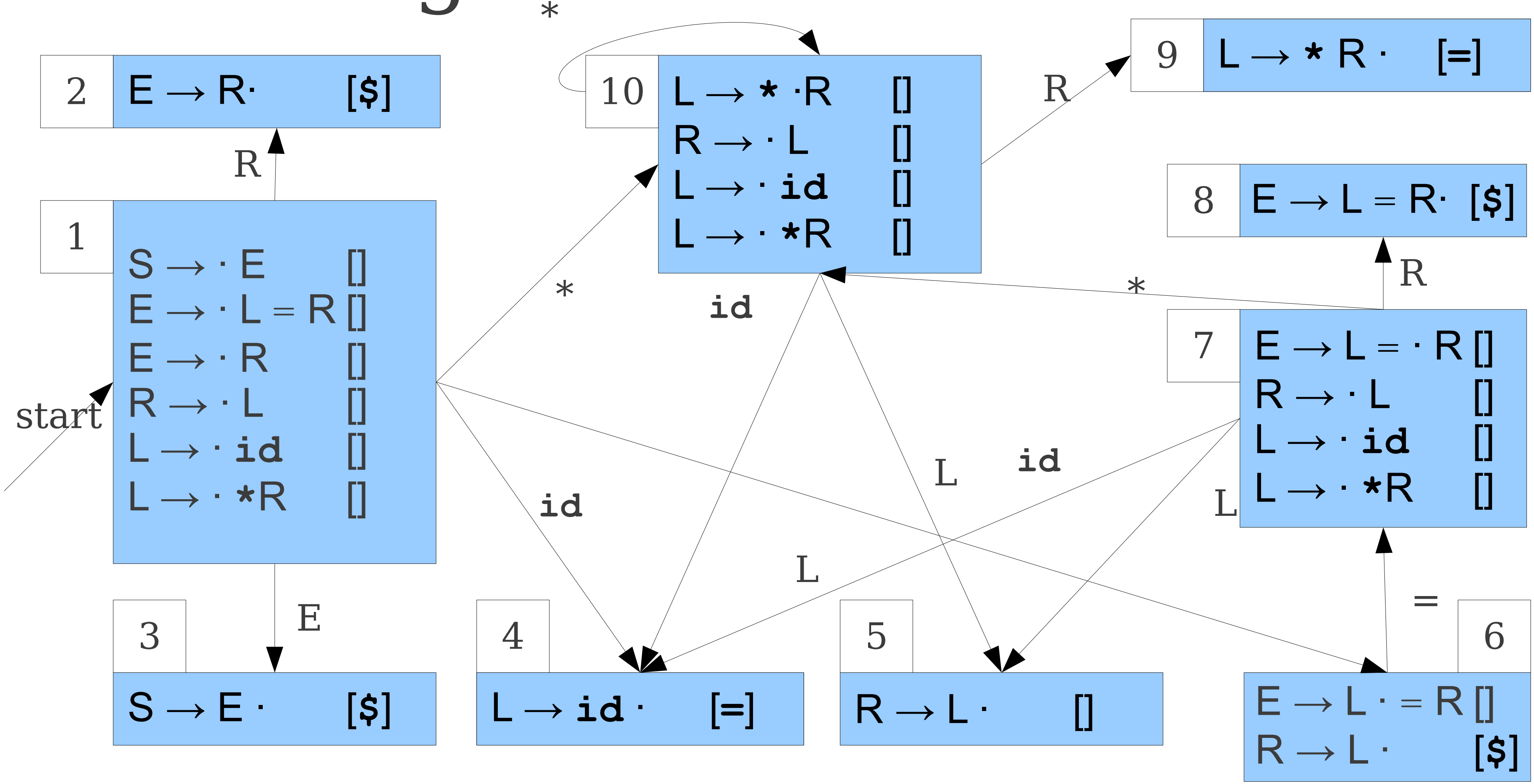
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



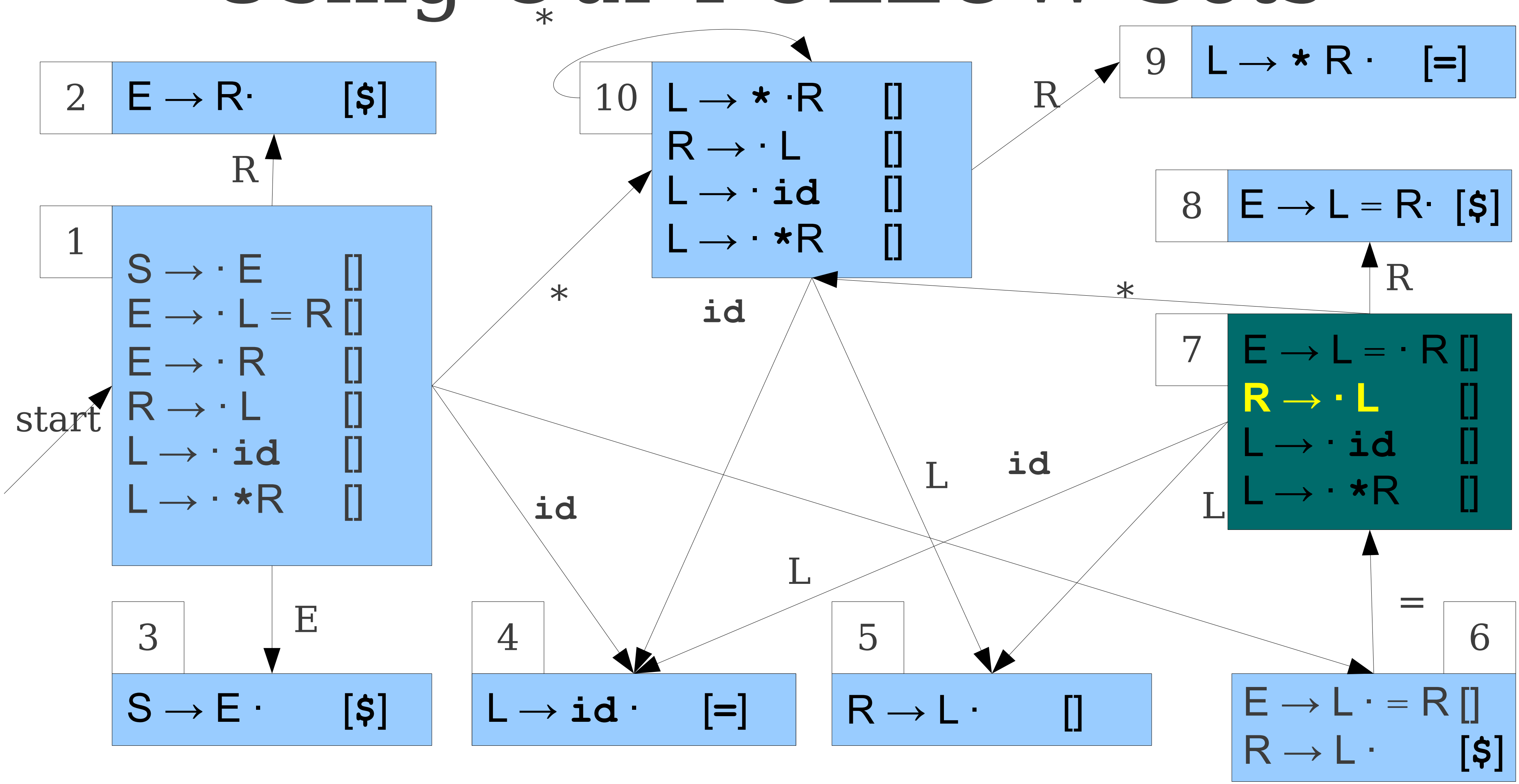
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



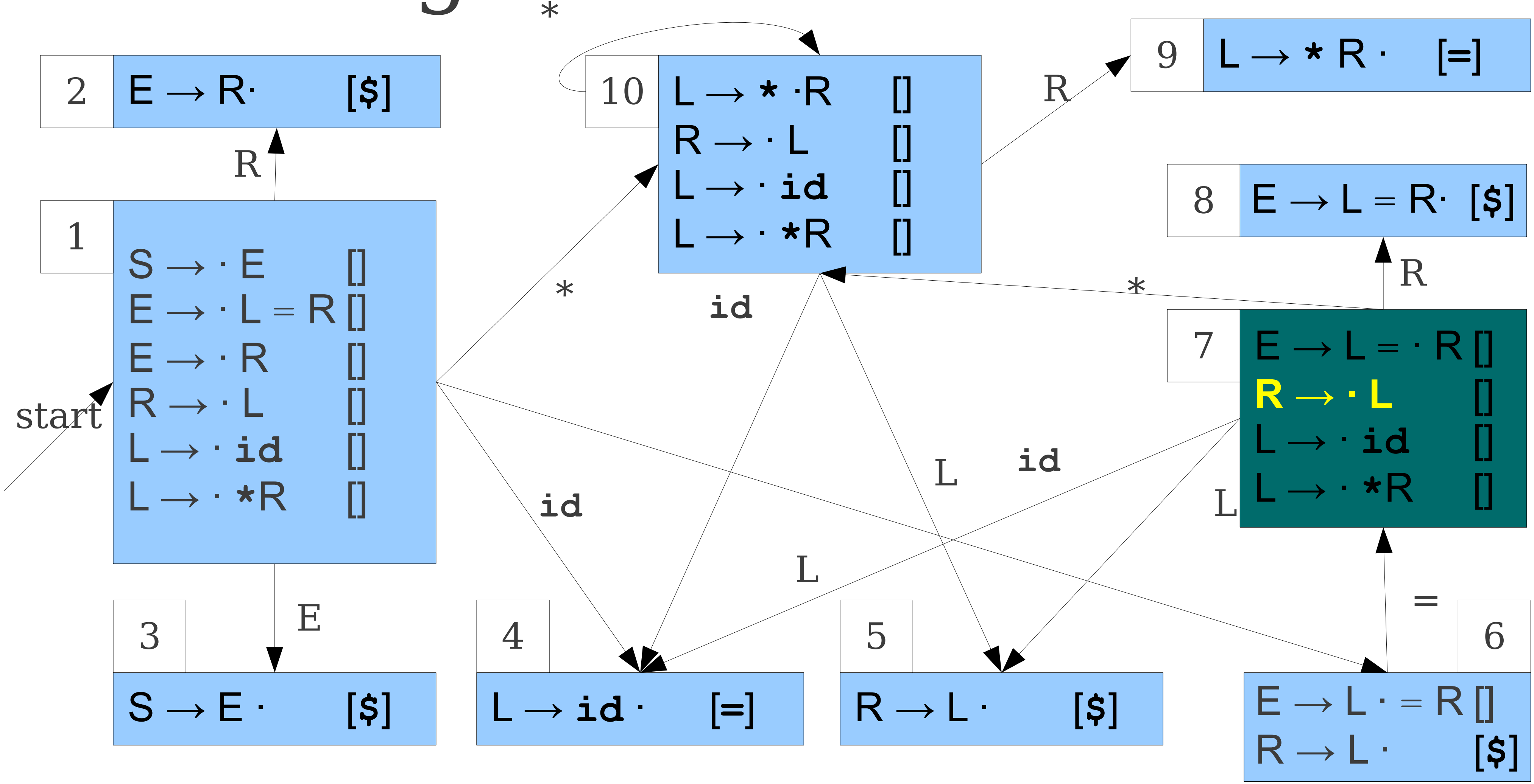
S_{1}	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



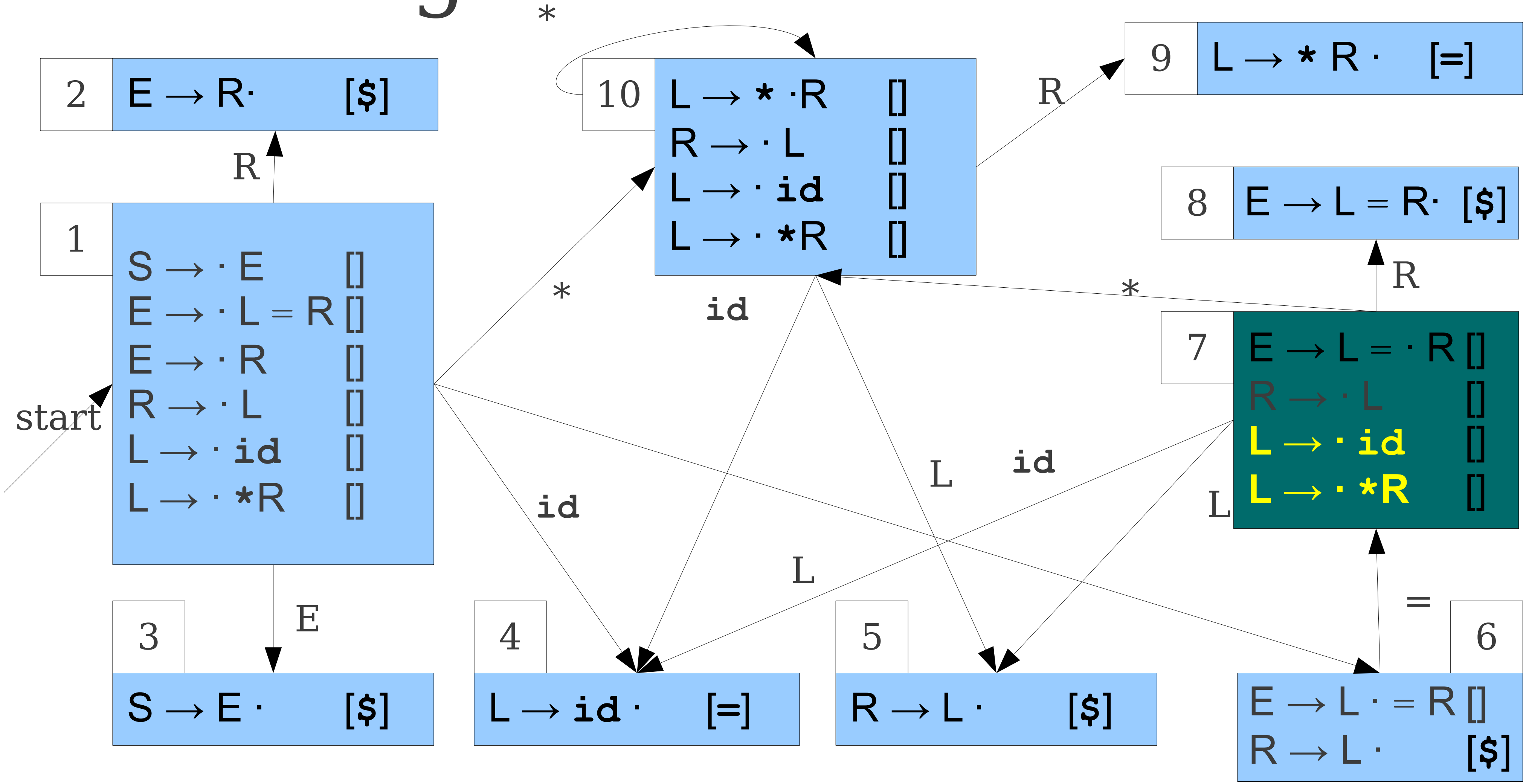
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



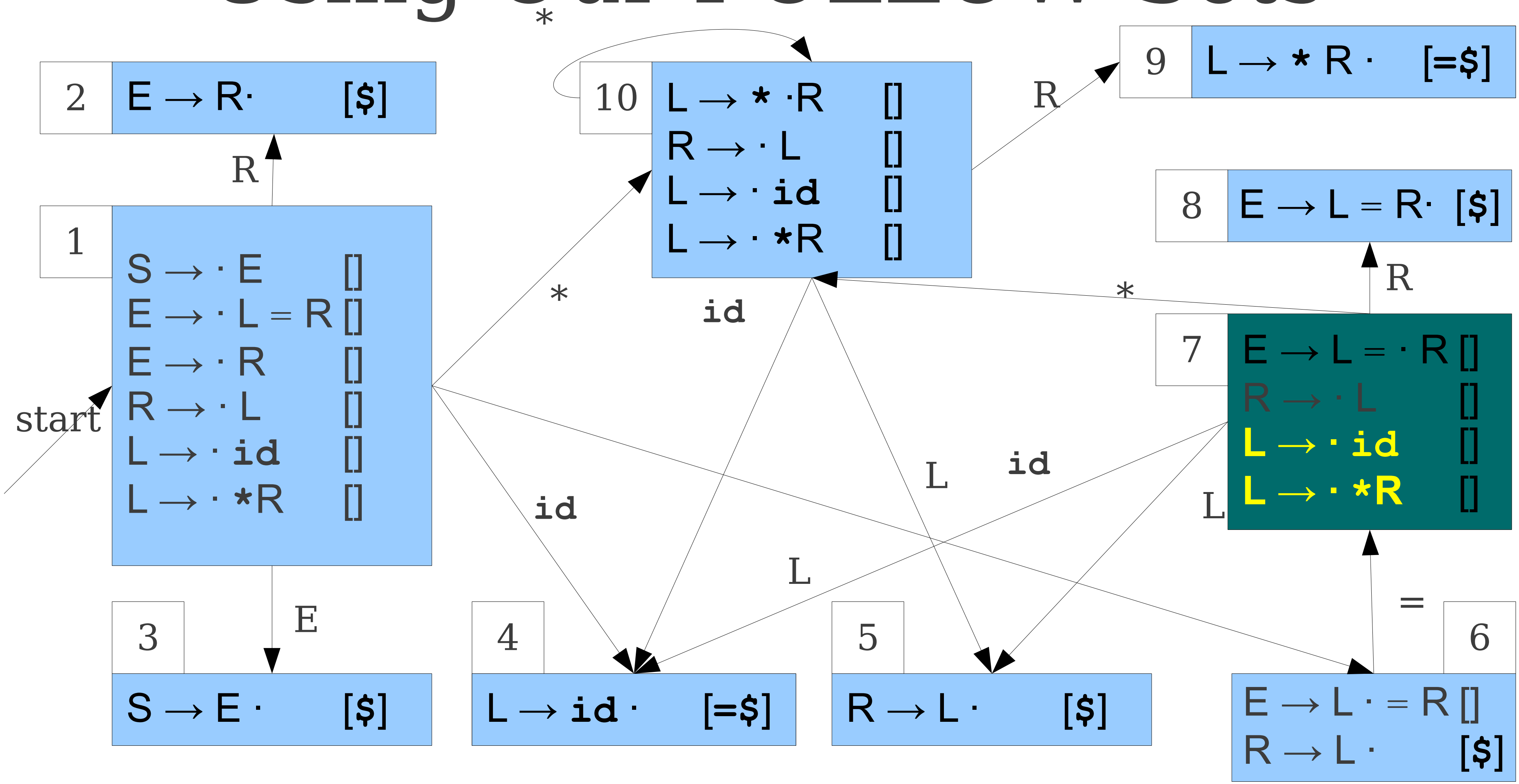
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



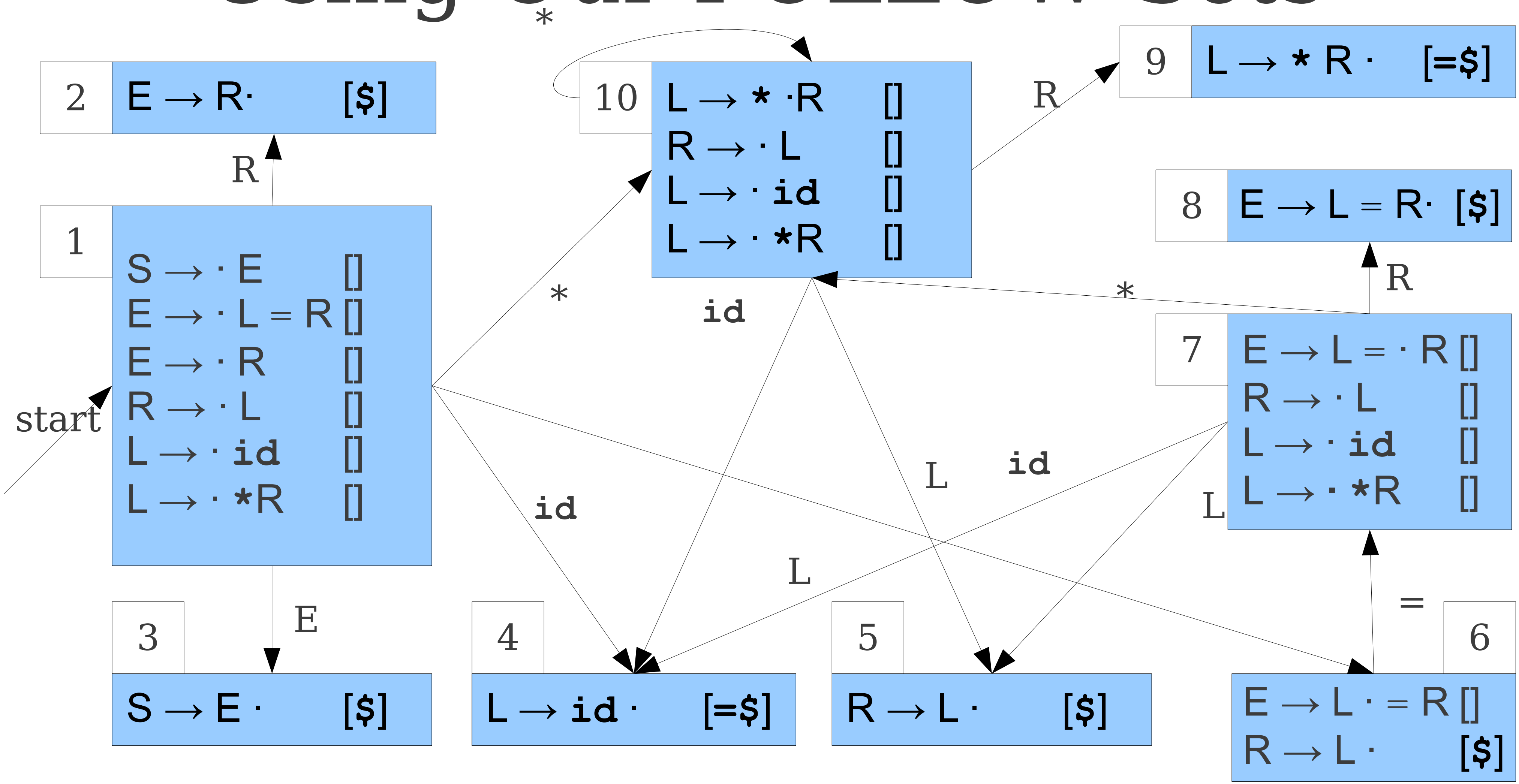
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



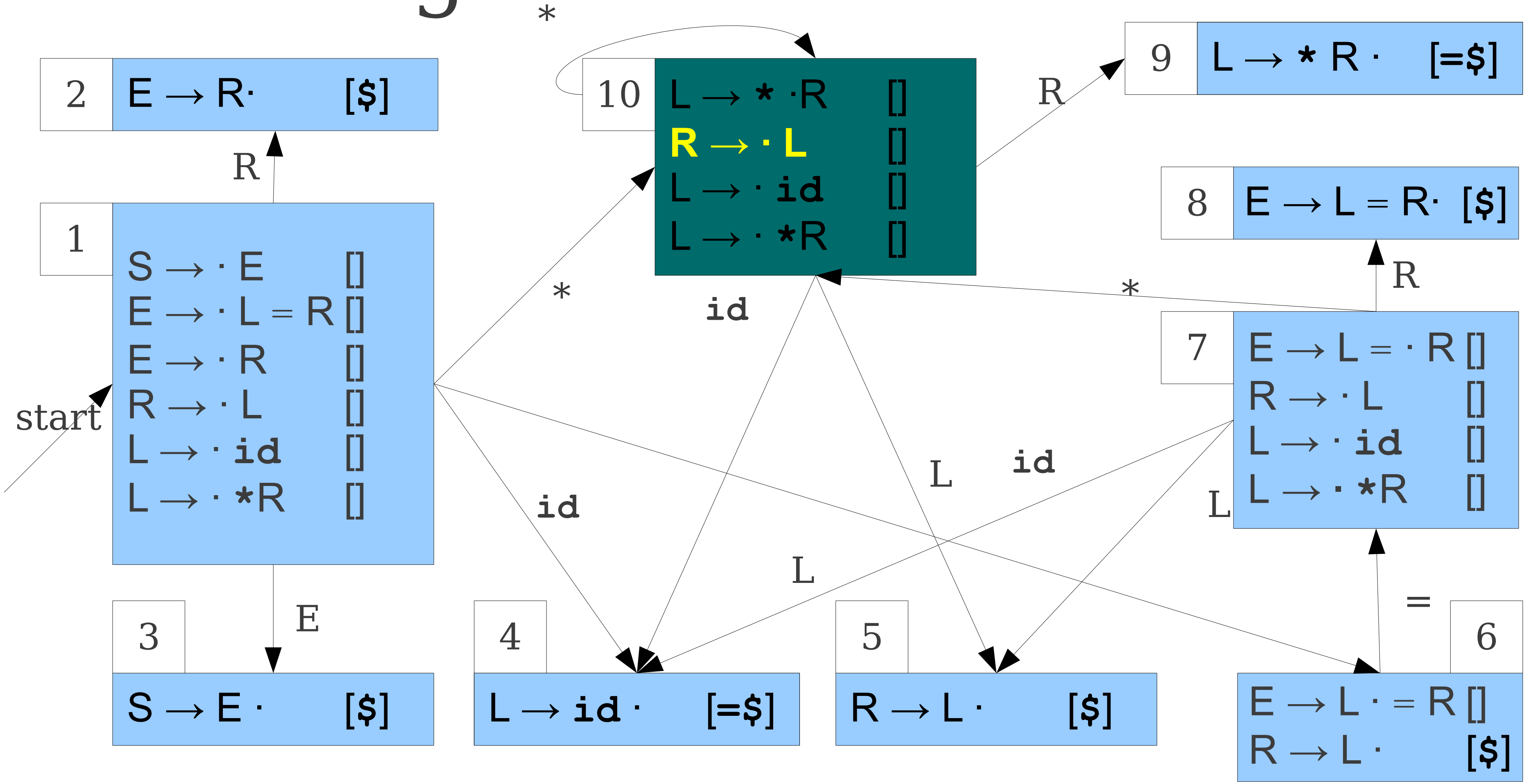
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



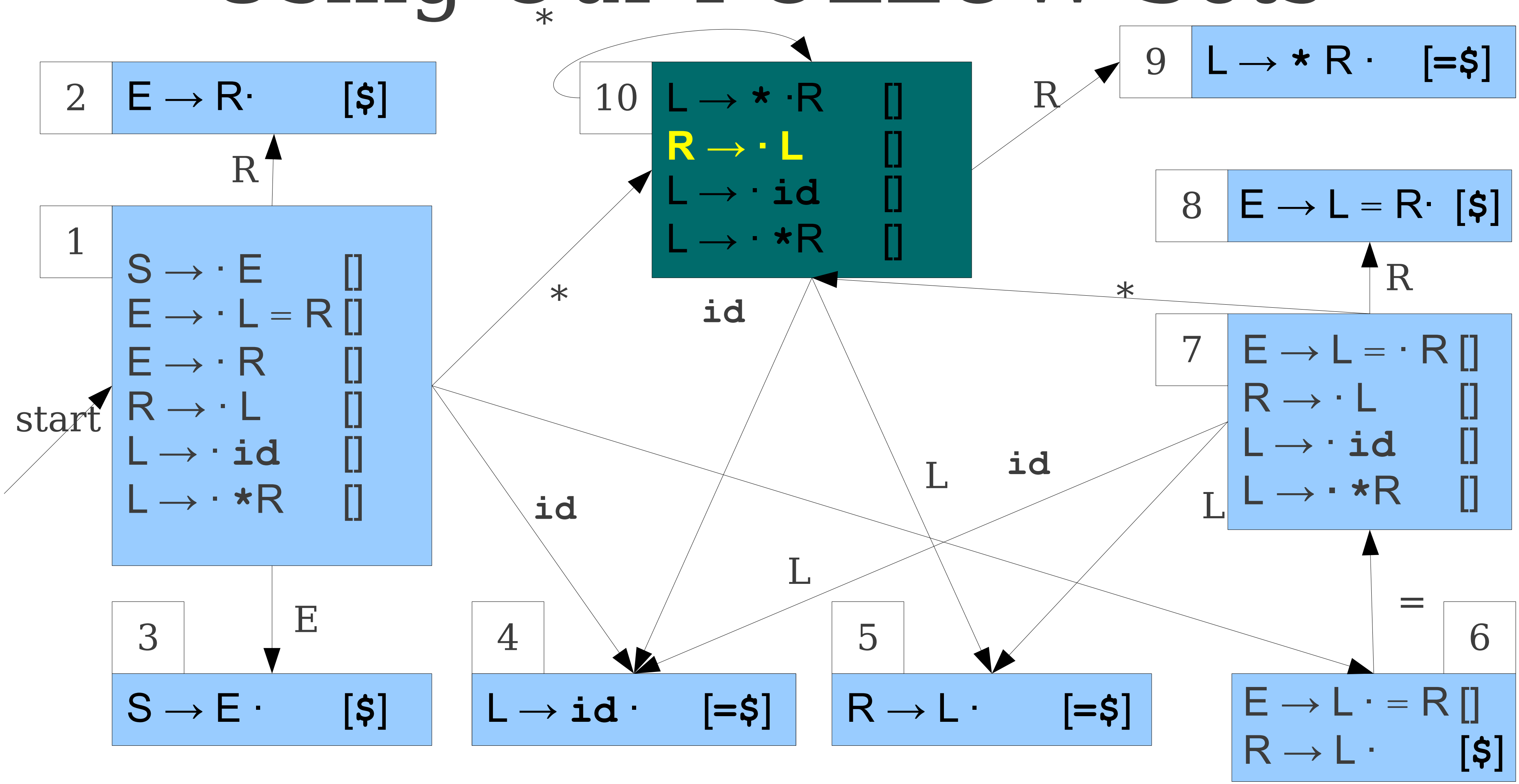
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



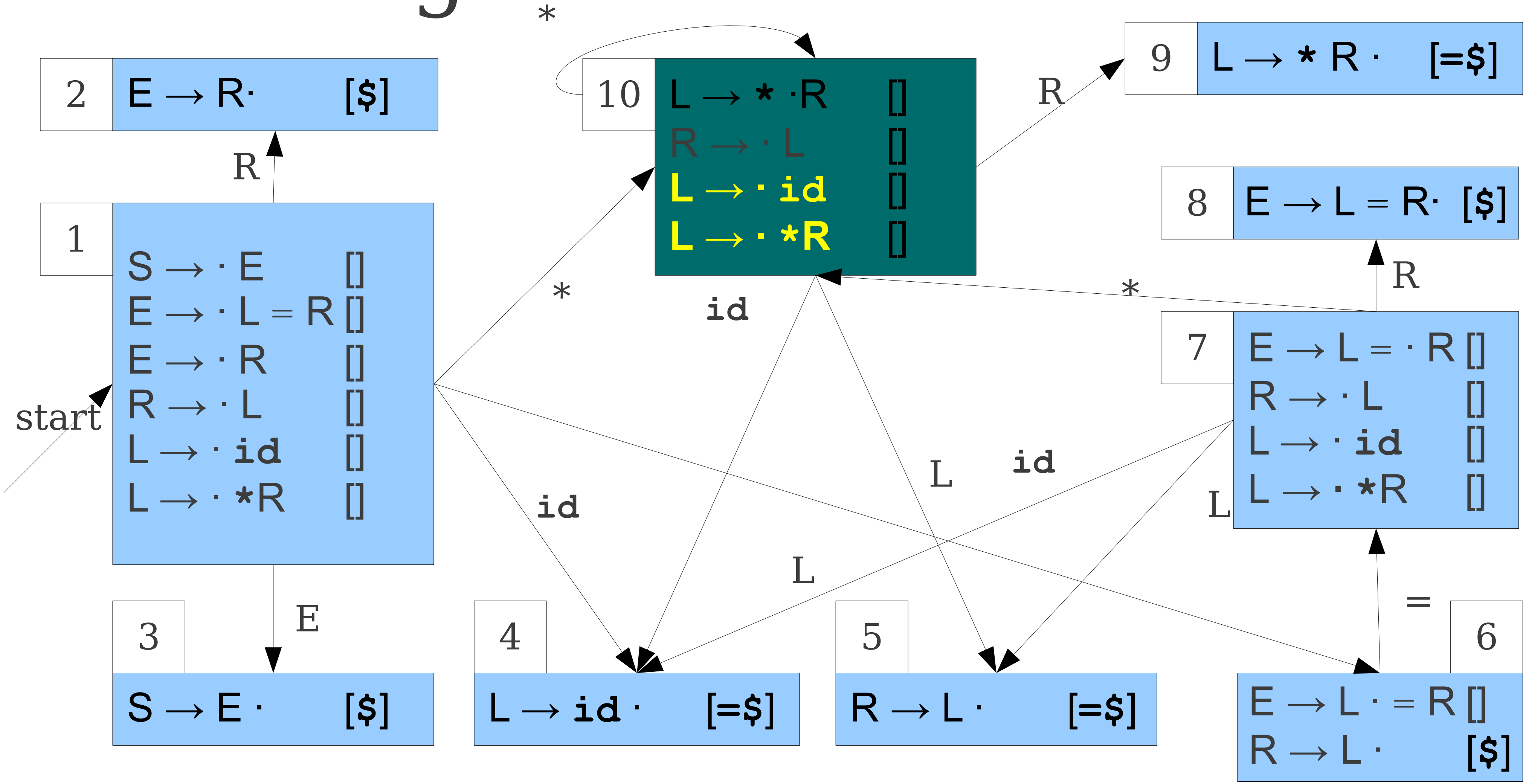
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



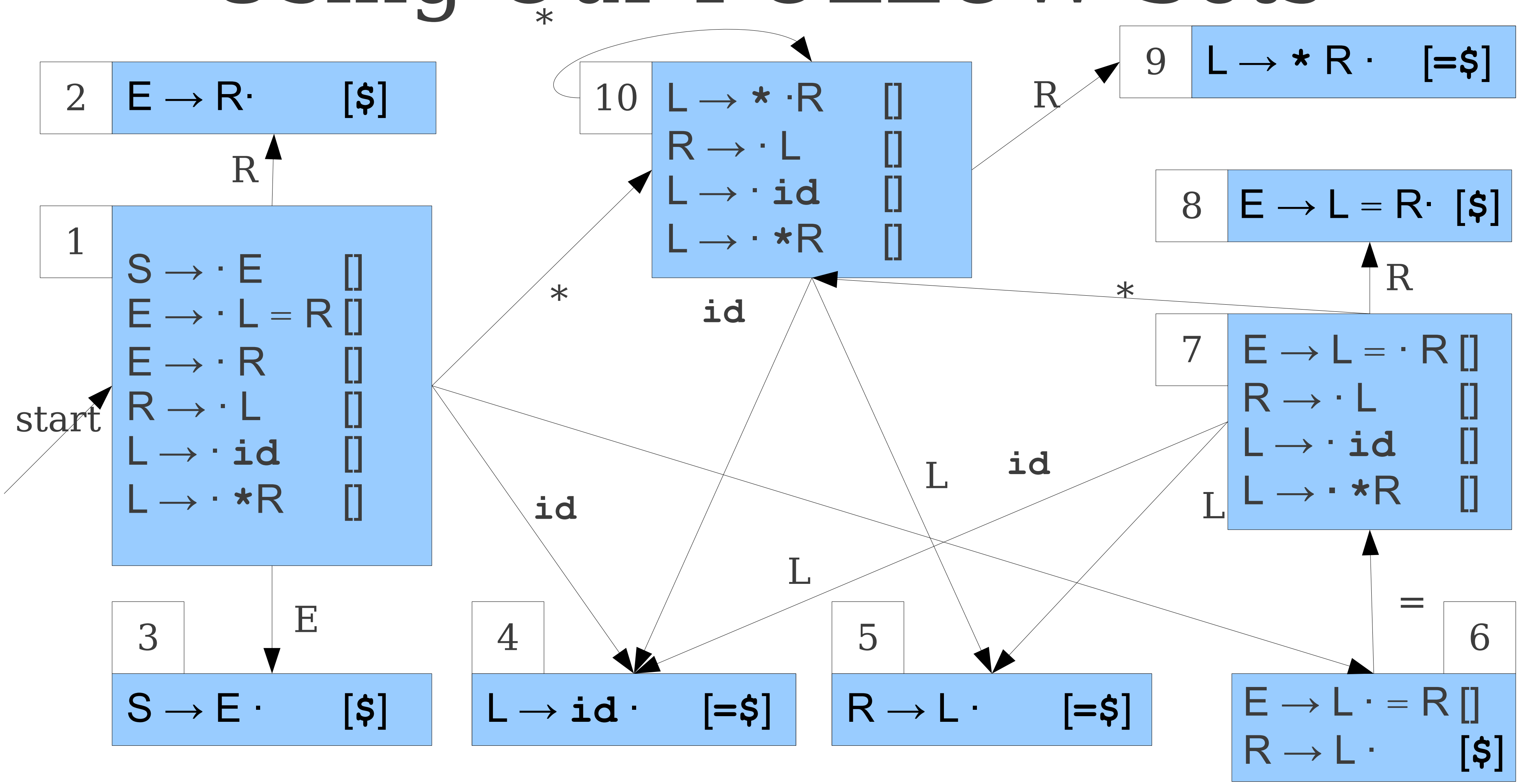
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



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

Using Our FOLLOW Sets



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

Propagating Changes

- For each item $\mathbf{A} \rightarrow \cdot\boldsymbol{\omega}$ in a state q:
  - Let $\mathbf{A}_{q \cdot r}$ be the nonterminal corresponding to $\mathbf{A}$ following the transition out of q into some state r.
  - Trace through the automaton along the path labeled by $\boldsymbol{\omega}$. This will lead to a state containing an item $\mathbf{A} \rightarrow \boldsymbol{\omega}\cdot$.
  - Add to the lookahead of $\mathbf{A} \rightarrow \boldsymbol{\omega} \cdot$ the contents of FOLLOW($\mathbf{A}_{q \cdot r}$)

LALR(1)-by-SLR(1)

- Fast and simple construction of LALR(1) lookaheads:
- 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.
- **Theorem** *(Bermudez and Logothetis)*: This correctly computes LALR(1) lookaheads.

Summary of LALR(1)

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

Practical Concerns

Where Theory Meets Practice

- We've just covered six powerful parsing algorithms:
  - Leftmost DFS
  - LL(1)
  - LR(0)
  - SLR(1)
  - LALR(1)
  - LR(1)
- How do we make them work in practice?

Two Practical Concerns

- **Ambiguity**
  - Real grammars are often ambiguous.
  - Programmers are *terrible* at eliminating it.
  - How do you build a parser to try to combat it?

作战，战斗

- **Error-handling**
  - How do you report errors intelligently?
  - How do you continue parsing after an error?

Ambiguity and Predictive Parsing

- The predictive parsers we have seen so far (LL(1), LR(0), SLR(1), LALR(1), LR(1)) only work on unambiguous grammars.
  - Intuitively: if grammar is ambiguous, cannot uniquely guess which production/reduction to use.
  - Formally proving this is somewhat involved.
- Most grammars for programming languages, unless cleverly written, are ambiguous.
- How can we handle this?

Parsing Ambiguous Grammars

- Consider this simple grammar for arithmetic expressions:

  S → **E**

  E → **E** `+` **E**

  E → **E** `*` **E**

  E → `int`

  E → `(`**E**`)`
- This grammar is ambiguous.
  - e.g. Two trees for `int + int * int`
- What happens if we try parsing it?

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

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

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

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

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

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10						
2							
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

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

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			
2							
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

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

	int	+	*	(	)	$	E
1	s10			s7			s2
2							
3							
4							
5							
6							
7							
8							
9							
10							

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

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

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

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2							
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3					
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

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

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4				
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

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

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. S → E
2. E → E + E
3. E → E * E
4. E → (E)
5. E → int

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

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

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

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10						
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

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

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

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

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

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

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10						
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1
S → · E E → · E + E E → · E * E E → · int E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

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

2	3	4	5
S → E · E → E ·+ E E → E ·* E	E → E + ·E E → · E + E E → · E * E E → · int E → · (E)	E → E * ·E E → · E + E E → · E * E E → · int E → · (E)	E → E + E · E → E · + E E → E · * E

6	7	8	9
E → E * E · E → E · + E E → E · * E	E → (·E) E → · E + E E → · E * E E → · int E → · (E)	E → (E ·) E → E ·+ E E → E ·* E	E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

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

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

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

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5							
6							
7							
8							
9							
10							

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

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

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3					
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3	s4				
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. $\mathbf{S} \rightarrow \mathbf{E}$
2. $\mathbf{E} \rightarrow \mathbf{E} + \mathbf{E}$
3. $\mathbf{E} \rightarrow \mathbf{E} * \mathbf{E}$
4. $\mathbf{E} \rightarrow (\mathbf{E})$
5. $\mathbf{E} \rightarrow$ `int`

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

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4				
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2				
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** __*__ **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, __*__, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2		
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

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

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S → E**
2. **E → E + E**
3. **E → E * E**
4. **E → (E)**
5. **E → int**

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → `int`

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

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3					
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. S → E
2. E → E + E
3. E → E * E
4. E → (E)
5. E → int

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

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3	s4				
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. S → E
2. E → E + E
3. E → E * E
4. E → (E)
5. E → int

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

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4				
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. S → E
2. E → E + E
3. E → E * E
4. E → (E)
5. E → int

FOLLOW(S) = { $ }

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

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3				
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S → E**
2. **E → E + E**
3. **E → E * E**
4. **E → (E)**
5. **E → int**

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

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

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3		
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

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

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

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

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

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

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

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10						
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** __*__ **E**
4. **E** → **(E)**
5. **E** → **int**

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

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

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

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. S → E
2. E → E + E
3. E → E * E
4. E → (E)
5. E → int

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

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8							
9							
10							

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

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

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3					
9							
10							

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4				
9							
10							

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. S → E
2. E → E + E
3. E → E * E
4. E → (E)
5. E → int

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

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

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

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

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4					
10							

SLR(1) Parsing with Ambiguity

1. **S → E**
2. **E → E + E**
3. **E → E * E**
4. **E → (E)**
5. **E → int**

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

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4				
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

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

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4		
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

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

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

1
S → · E E → · E + E E → · E * E E → · int E → · (E)

2	3	4	5
S → E · E → E ·+ E E → E ·* E	E → E + ·E E → · E + E E → · E * E E → · int E → · (E)	E → E * ·E E → · E + E E → · E * E E → · int E → · (E)	E → E + E · E → E · + E E → E · * E

6	7	8	9	10
E → E * E · E → E · + E E → E · * E	E → (·E) E → · E + E E → · E * E E → · int E → · (E)	E → (E ·) E → E ·+ E E → E ·* E	E → (E) ·	E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10							

SLR(1) Parsing with Ambiguity

1
S → · E E → · E + E E → · E * E E → · int E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

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

2	3	4	5
S → E · E → E ·+ E E → E ·* E	E → E + ·E E → · E + E E → · E * E E → · int E → · (E)	E → E * ·E E → · E + E E → · E * E E → · int E → · (E)	E → E + E · E → E · + E E → E · * E

6	7	8	9	10
E → E * E · E → E · + E E → E · * E	E → (·E) E → · E + E E → · E * E E → · int E → · (E)	E → (E ·) E → E ·+ E E → E ·* E	E → (E) ·	E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** `+` **E**
3. **E** → **E** `*` **E**
4. **E** → `(`**E**`)`
5. **E** → `int`

FOLLOW(**S**) = { `$` }
FOLLOW(**E**) = { `+`, `*`, `)`, `$` }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5					

SLR(1) Parsing with Ambiguity

1. S → E
2. E → E + E
3. E → E * E
4. E → (E)
5. E → int

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

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5				

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

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

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5		

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

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

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

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

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

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

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

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

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

Resolving Ambiguity

- Although the grammar is ambiguous, there is clearly one intended parse tree because of operator precedence.
- How can we use this precedence information to avoid LR conflicts?

Precedence Declarations

- Tell the parser generator about the *associativity* and *precedence* of certain rules.
- Productions can be left-associative, right-associative, or nonassociative.
- Productions can have their priorities ranked against one another.

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** `+` **E**
3. **E** → **E** `*` **E**
4. **E** → `(`**E**`)`
5. **E** → `int`

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Left-assoc, pri. 0)
3. **E** → **E * E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Left-assoc, pri. 0)
3. **E** → **E * E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Left-assoc, pri. 0)
3. **E** → **E * E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E + E** (Left-assoc, pri. 0)
3. **E** → **E * E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Left-assoc, pri. 0)
3. **E** → **E * E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Left-assoc, pri. 0)
3. **E** → **E * E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Left-assoc, pri. 0)
3. **E** → **E * E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Left-assoc, pri. 0)
3. **E** → **E * E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E** (Left-assoc, pri. 0)
3. **E** → **E** ***** **E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		r2	s4		r2	r2	
6		r3	r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S → E**
2. **E → E + E** (Left-assoc, pri. 0)
3. **E → E * E** (Left-assoc, pri. 1)
4. **E → (E)**
5. **E → int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		r2	s4		r2	r2	
6		r3	r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Rgt.-assoc, pri. 0)
3. **E** → **E * E** (Rgt.-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Rgt.-assoc, pri. 0)
3. **E** → **E * E** (Rgt.-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Rgt.-assoc, pri. 0)
3. **E** → **E * E** (Rgt.-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1
S → · E E → · E + E E → · E * E E → · int E → · (E)

2	3	4	5
S → E · E → E ·+ E E → E ·* E	E → E + ·E E → · E + E E → · E * E E → · int E → · (E)	E → E * ·E E → · E + E E → · E * E E → · int E → · (E)	E → E + E · E → E · + E E → E · * E

6	7	8	9
E → E * E · E → E · + E E → E · * E	E → (·E) E → · E + E E → · E * E E → · int E → · (E)	E → (E ·) E → E ·+ E E → E ·* E	E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3	s4		r2	r2	
6		r3	s4		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Rgt.-assoc, pri. 0)
3. **E** → **E * E** (Rgt.-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3	s4		r2	r2	
6		r3	s4		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

Resolving Conflicts with Precedence

- When choosing whether to reduce a rule containing `t` or shift the terminal `r`:
  - If `t` has higher priority, **reduce**.
  - If `r` has higher priority, **shift**.
  - If `t` and `r` have the same priority:
    - If `t` is left-associative, **reduce**.
    - If `t` is right-associative, **shift**.
    - If `t` is non-associative, **error**.

SLR(1) Parsing with Ambiguity

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E** (Left-assoc, pri. 0)
3. **E** → **E** ***** **E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		r2	s4		r2	r2	
6		r3	r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Left-assoc, pri. 0)
3. **E** → **E * E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		r2	s4		r2	r2	
6		r3	r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

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 Productions

- One idea: add productions to the grammar that identify common mistakes.
- For example:

$$\mathbf{E} \to \mathbf{E}\ \texttt{+}\ \mathbf{E}$$

$$\mathbf{E} \to \mathbf{E}\ \texttt{*}\ \mathbf{E}$$

$$\mathbf{E} \to \texttt{int}$$

$$\mathbf{E} \to \texttt{(}\mathbf{E}\texttt{)}$$

$$\mathbf{E} \to \mathbf{E}\ \mathbf{E} \quad \textit{(error production)}$$

$$\mathbf{E} \to \mathbf{E}\ \texttt{+} \quad \textit{(error production)}$$

$$\mathbf{E} \to \mathbf{E}\ \texttt{*} \quad \textit{(error production)}$$

Analysis of Error Productions

- Useful for diagnosing common programmer mistakes.
  - For example, using **`implements`** instead of **`extends`** in Java.
- Increases risk of parsing problems.
  - More likelihood for ambiguity.
  - More likelihood grammar won't be accepted by parser generator (i.e. not LALR(1))
- Forces parser generator to anticipate errors.

Panic Mode

- Idea: Augment grammar by adding rules for resuming parsing when Bad Things happen.
- Example:

  E → **E** `+` **E**

  E → **E** `*` **E**

  E → `(`**E**`)`

  E → `int`

  E → **error** `int`

  E → `(`**error**`)`
- Tokens after errors are called **synchronizing tokens**.
- Technique employed by `bison` and many other parser generators.

Using Panic Mode

- When parser encounters an error in a configurating set, search for a production containing an error term.
  - Repeatedly shift tokens onto the stack until the synchronizing token is found.
  - Reduce using the error rule.
- Resume parsing as normal.

Next Time

- **The Limits of Parsing:**
  - Parsing ambiguous grammars, take II.
  - Parsing arbitrary CFGs: The Earley Parser
  - Parsing with missing tokens: Intersection Parsing
  - A Most Excellent Parser: The Earley-on-Intersection Algorithm