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.

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).

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

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.
  - 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.

Revisiting Shift/Reduce Conflicts

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

  $\mathbf{A} \to \omega \cdot$

  $\mathbf{B} \to \alpha \cdot \beta$

- 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)

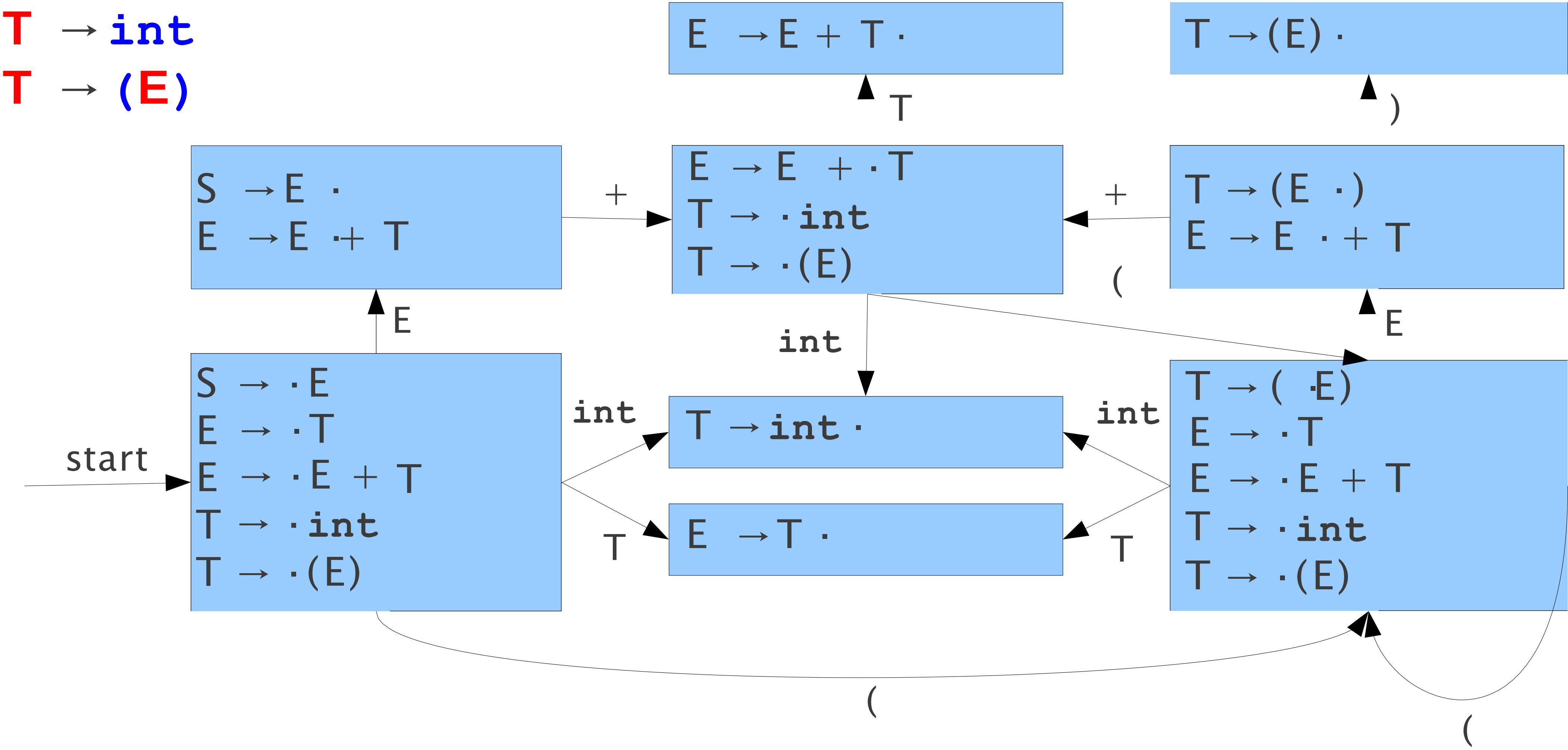
- **Simple LR(1)**
- Minor modification to LR(0) automaton that uses lookahead to avoid shift/reduce conflicts.
- Idea: Only reduce **A** → ω if the next token **t** is in FOLLOW(**A**).
- Automaton identical to LR(0) automaton; only change is when we choose to reduce.

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

SLR(1) Parsing

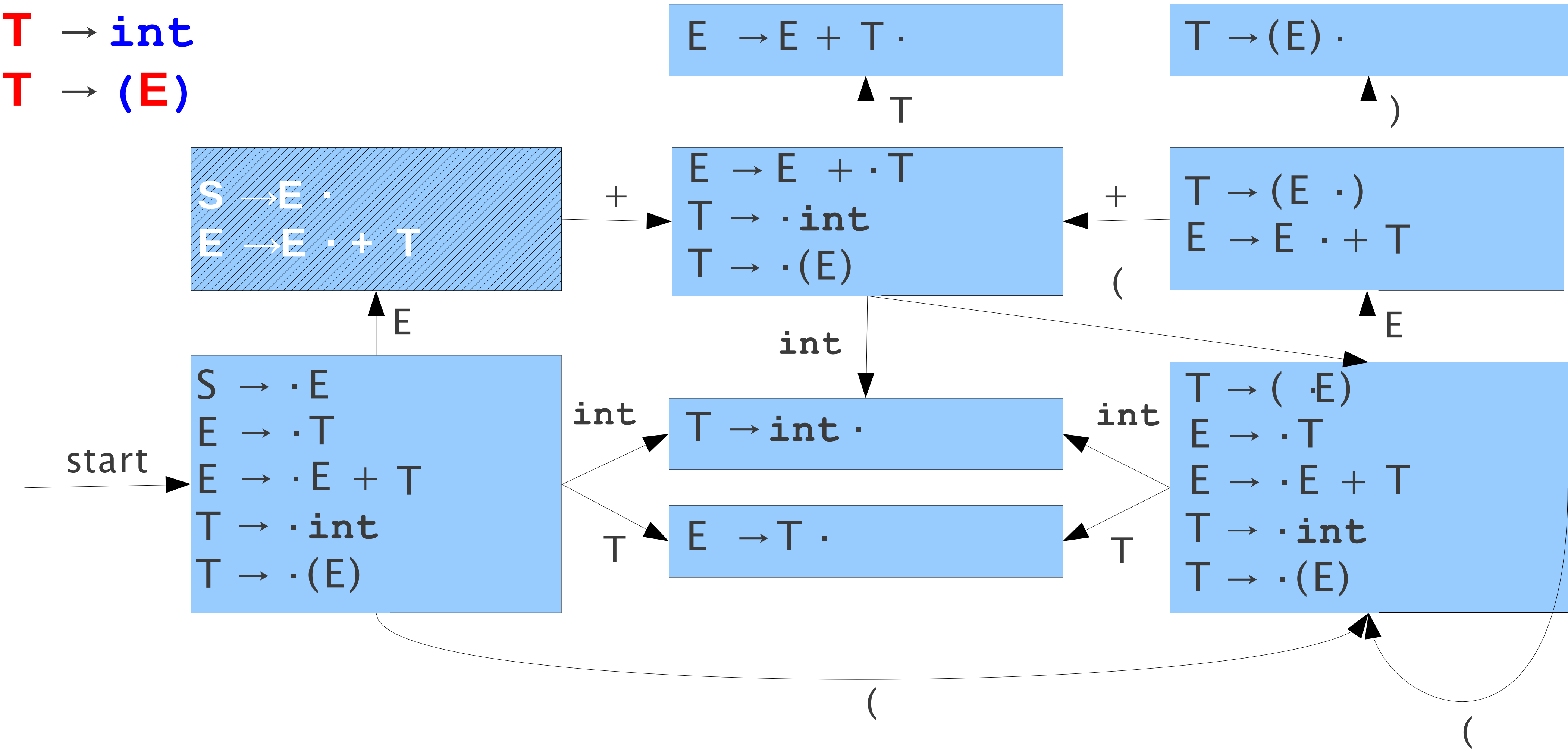
S → E
E → T
E → E + T
T → `int`
T → (E)

SLR(1) Parsing



S → E
E → T
E → E + T
T → int
T → (E)

SLR(1) Parsing



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

S → E ·
E → E · + T

E → E + ·T
T → ·int
T → ·(E)

E → E + T ·

T → int ·

E → T ·

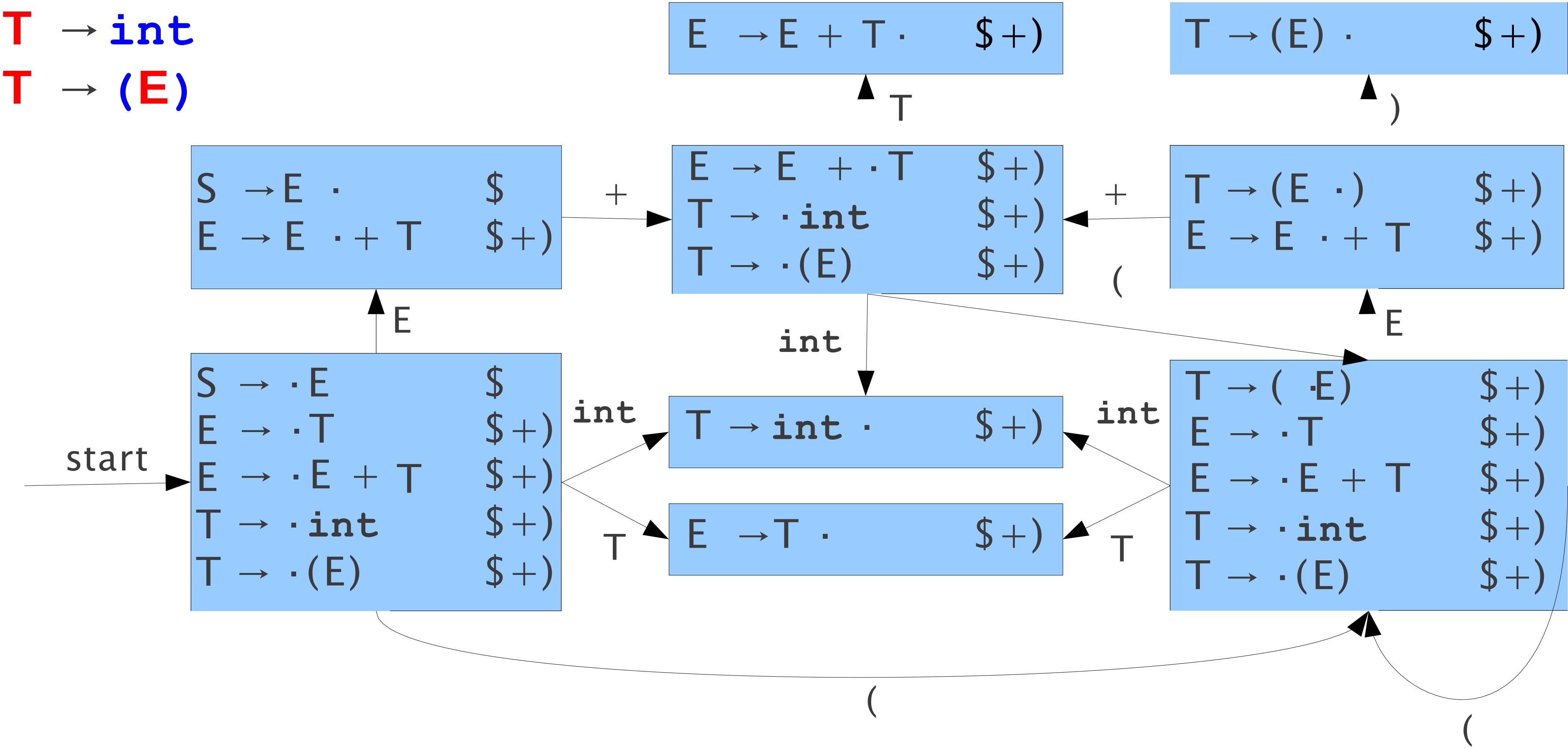
T → (·E)
E → ·T
E → ·E + T
T → ·int
T → ·(E)

T → (E ·)
E → E · + T

T → (E) ·

S → E
E → T
E → E + T
T → int
T → (E)

SLR(1) Parsing



E → E + T · $+)

T → (E) · $+)

S → E · $
E → E · + T $+)

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

T → (E ·) $+)
E → E · + T $+)

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

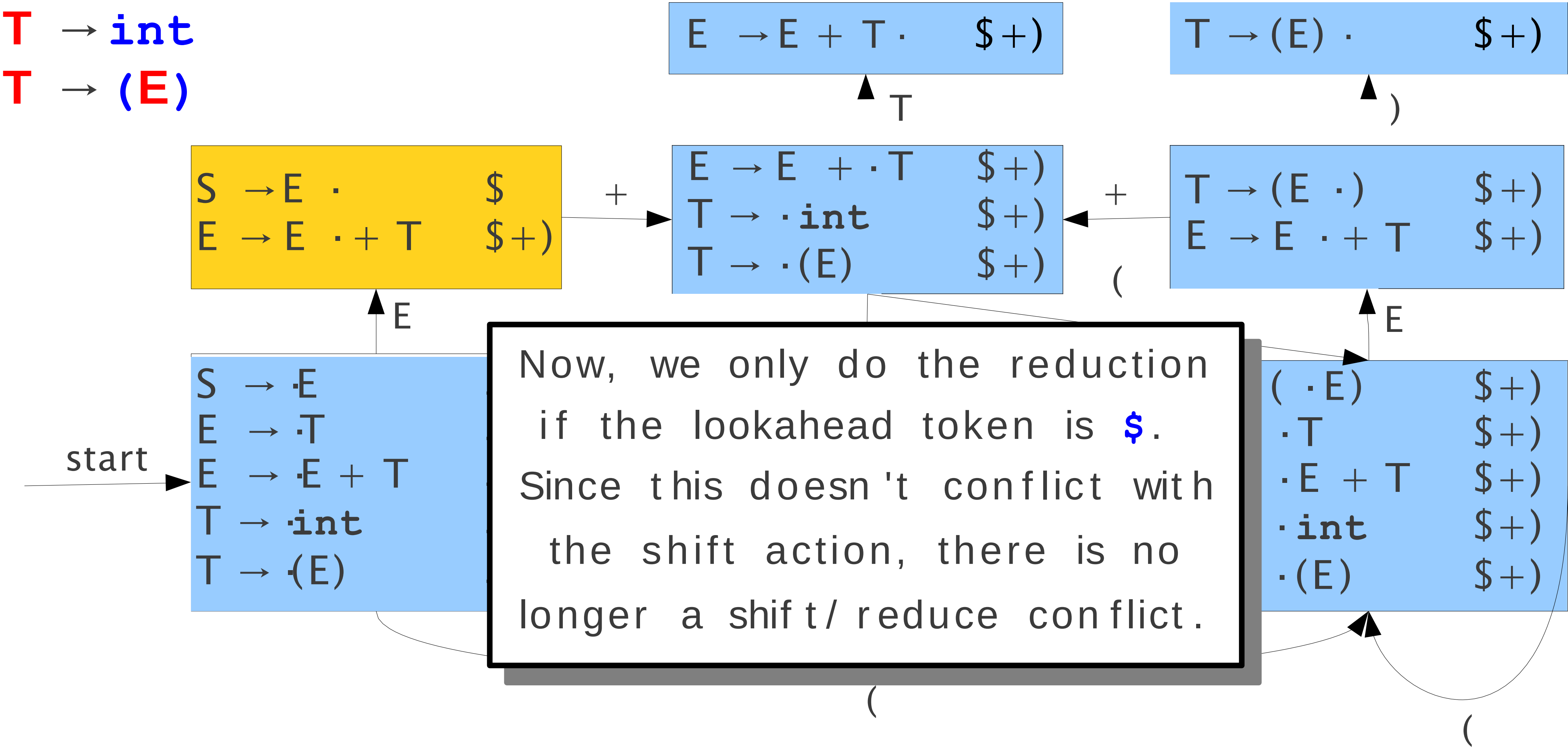
T → **int** · $+)

E → T · $+)

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

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

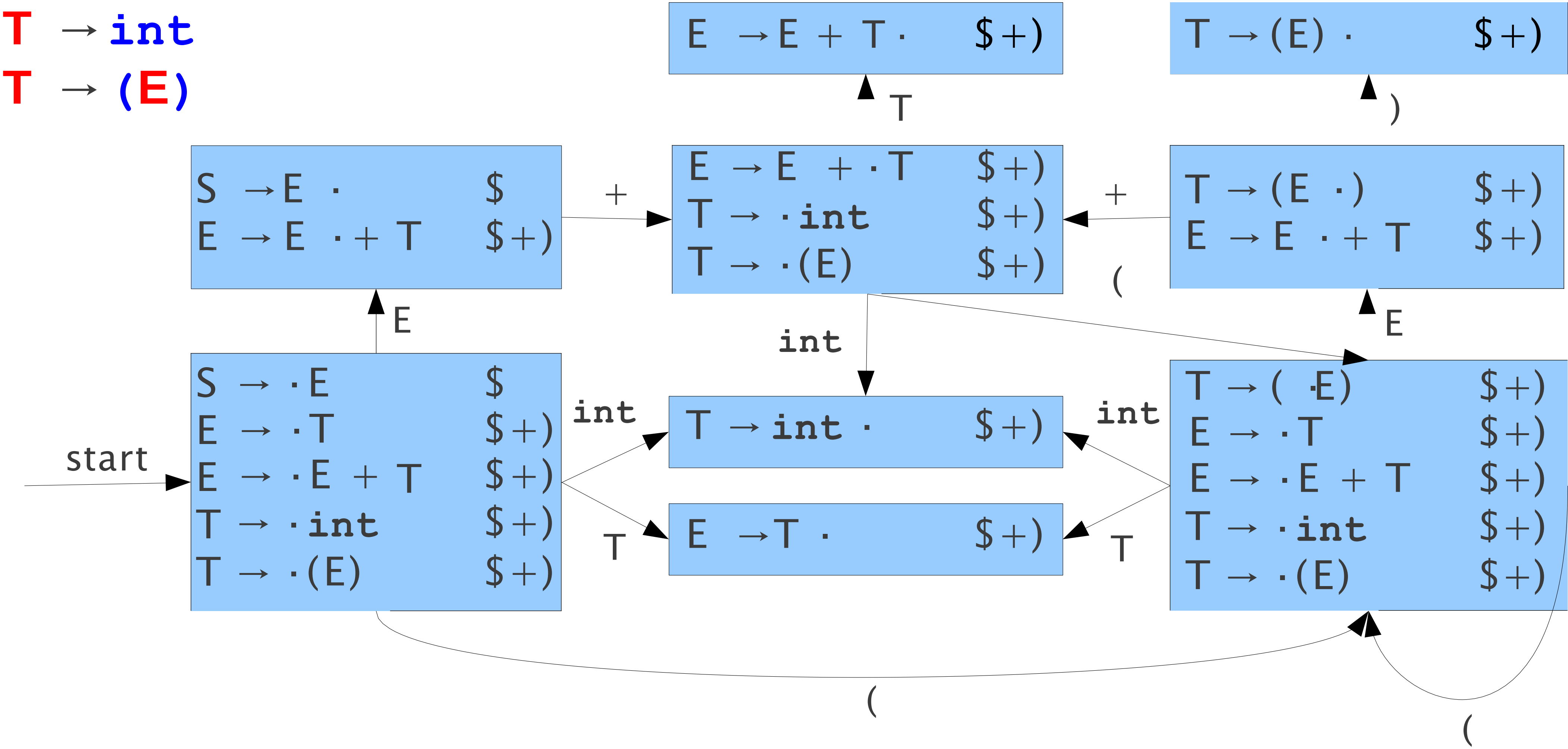
SLR(1) Parsing



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


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

S → E · $
E → E · + T $+)

E → E + · T $+)
T → ·int $+)
T → ·(E) $+)

E → E + T · $+)

T → int · $+)

E → T · $+)

T → (·E) $+)
E → ·T $+)
E → ·E + T $+)
T → ·int $+)
T → ·(E) $+)

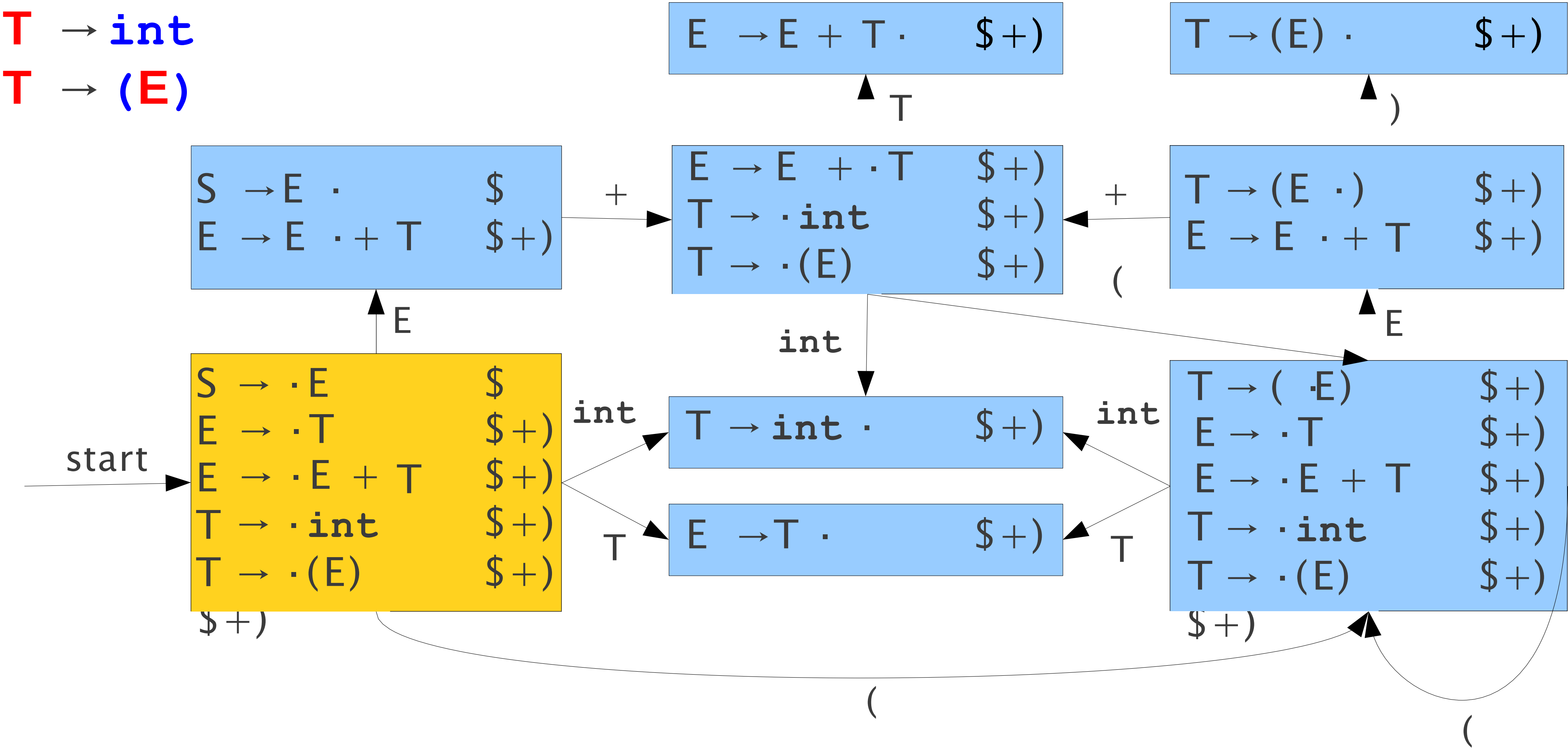
T → (E ·) $+)
E → E · + T $+)

T → (E) · $+)

int	+	(	int	+	int	+	int	)	$

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

SLR(1) Parsing



E → E + T· $+)

T → (E) · $+)

S → E · $
E → E ·+ T $+)

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

T → (E ·) $+)
E → E ·+ T $+)

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

T → **int** · $+)

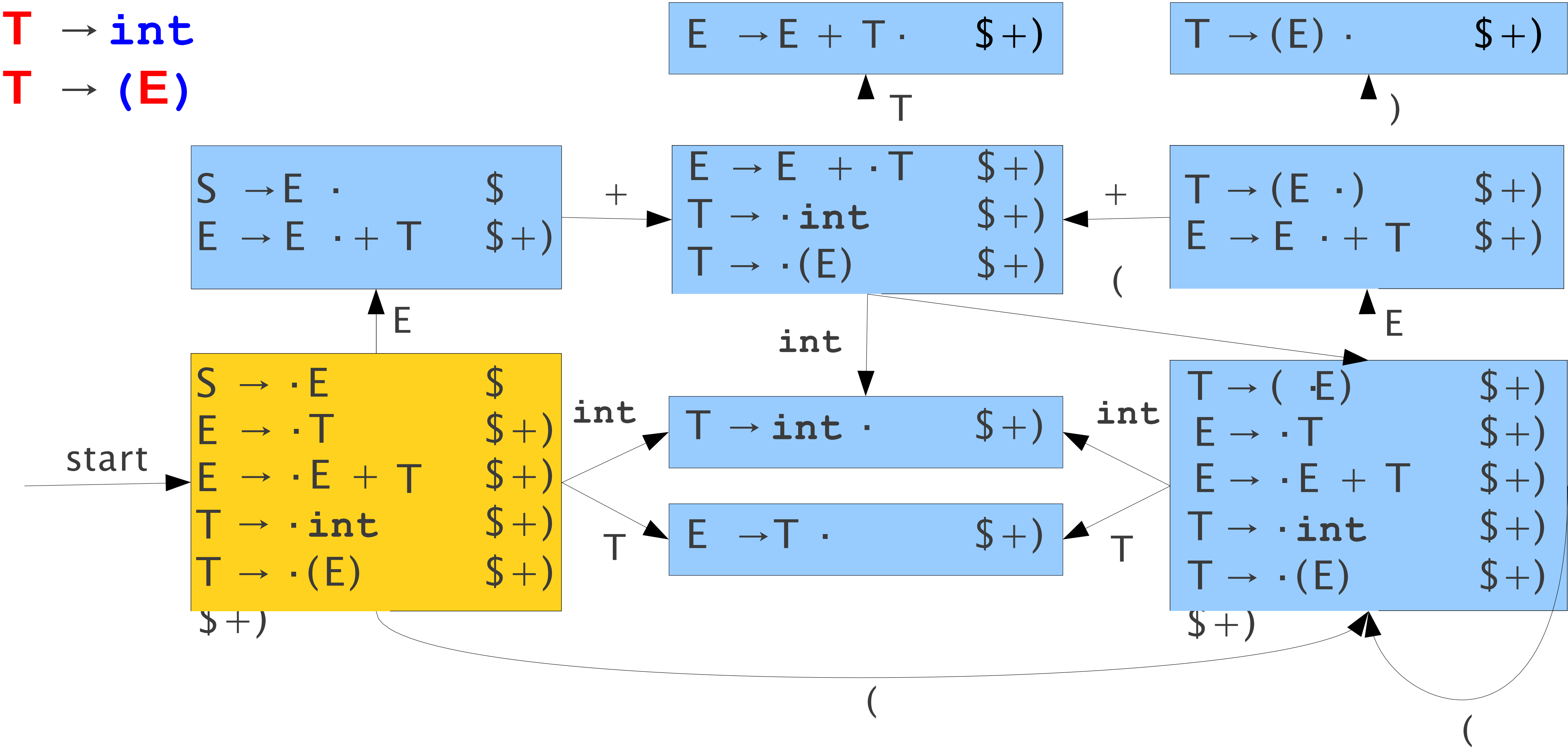
E → T · $+)

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

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

S → E
E → T
E → E + T
T → int
T → (E)

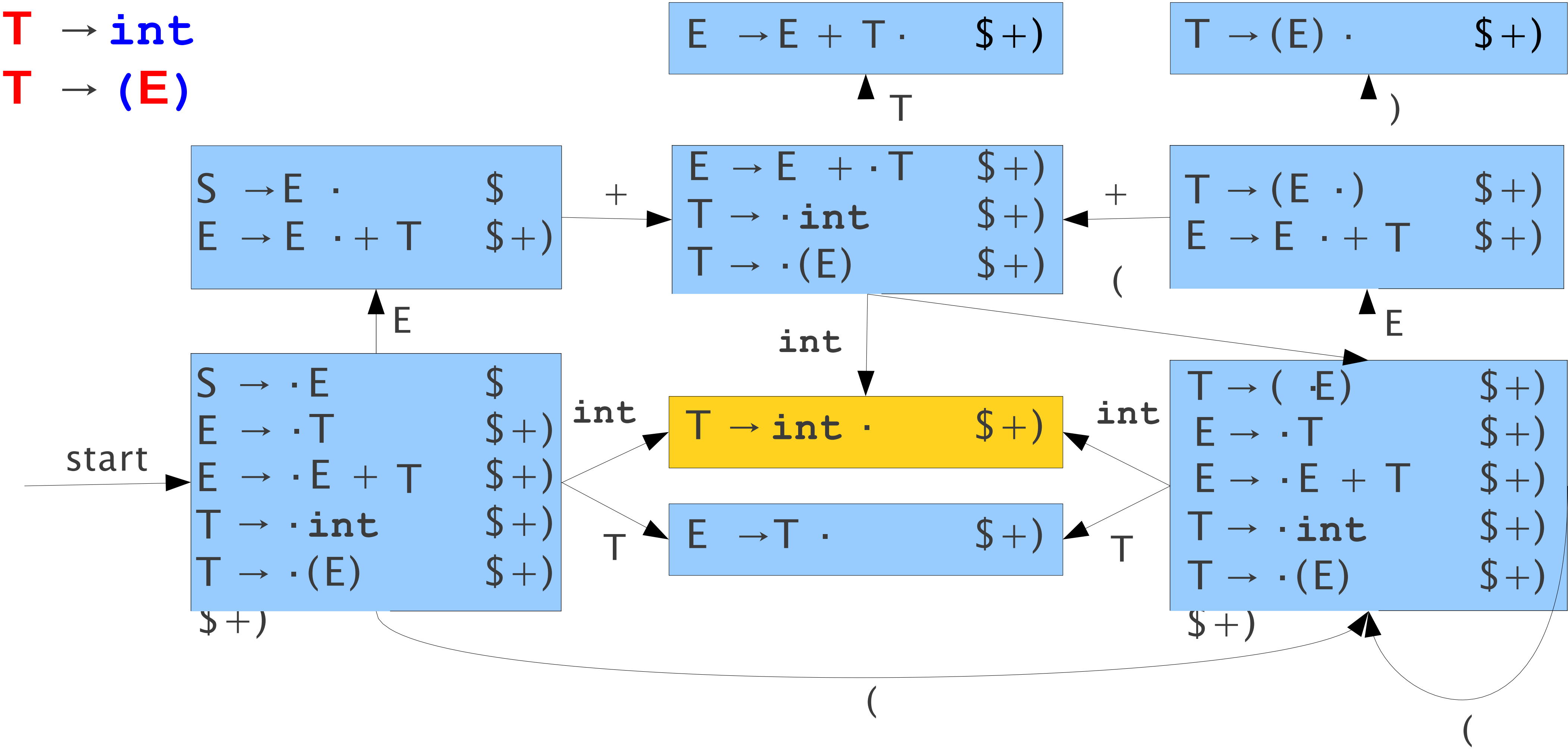
SLR(1) Parsing



int		+	(	int	+	int	+	int	)	$

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

SLR(1) Parsing



E → E + T · $+)

T → (E) · $+)

S → E · $
E → E · + T $+)

E → E + · T $+)
T → · int $+)
T → · (E) $+)

T → (E ·) $+)
E → E · + T $+)

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

T → int · $+)

E → T · $+)

T → (·E) $+)
E → · T $+)
E → · E + T $+)
T → · int $+)
T → · (E) $+)

$+)

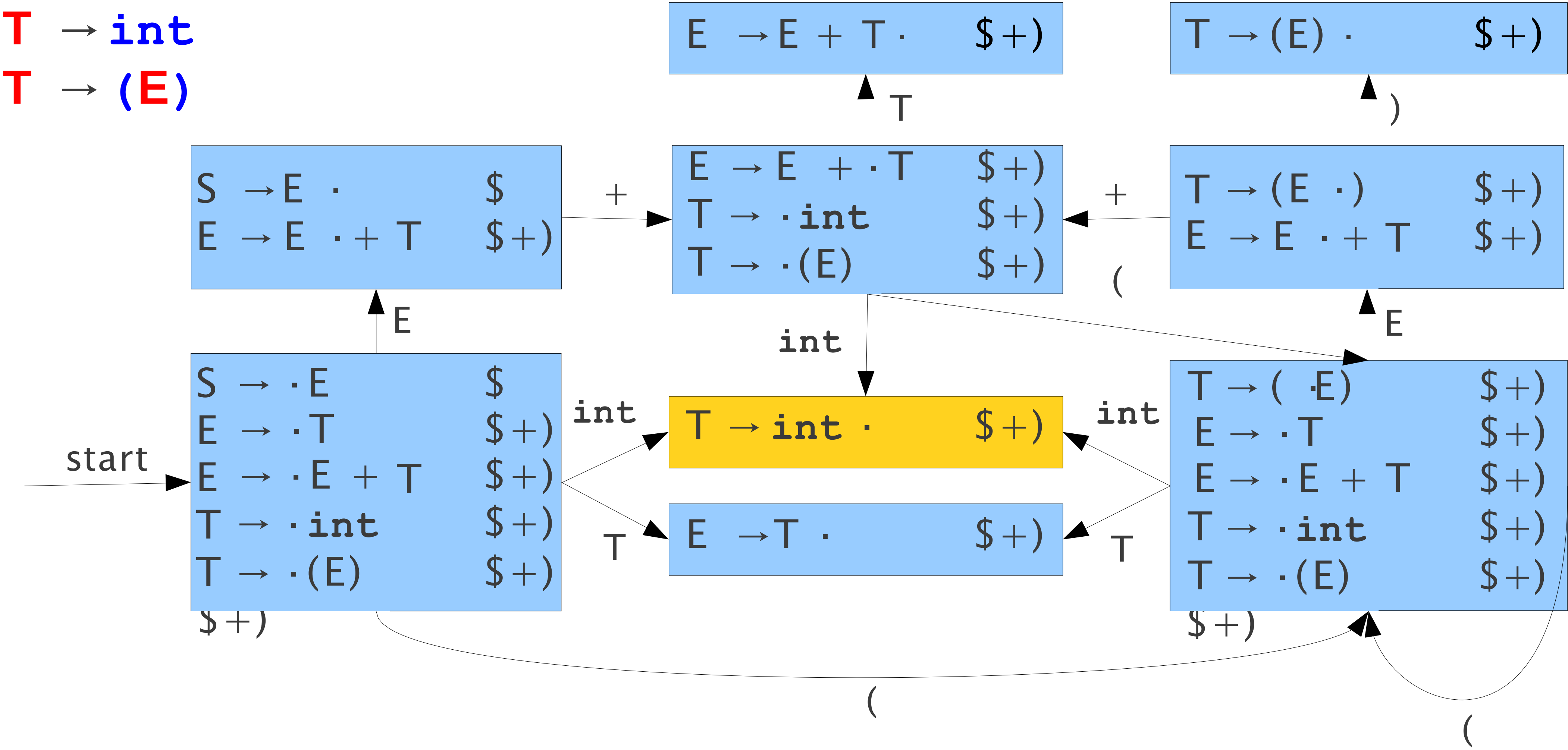
$+)

int

+	(	int	+	int	+	int	)	$

S → E
E → T
E → E + T
T → int
T → (E)

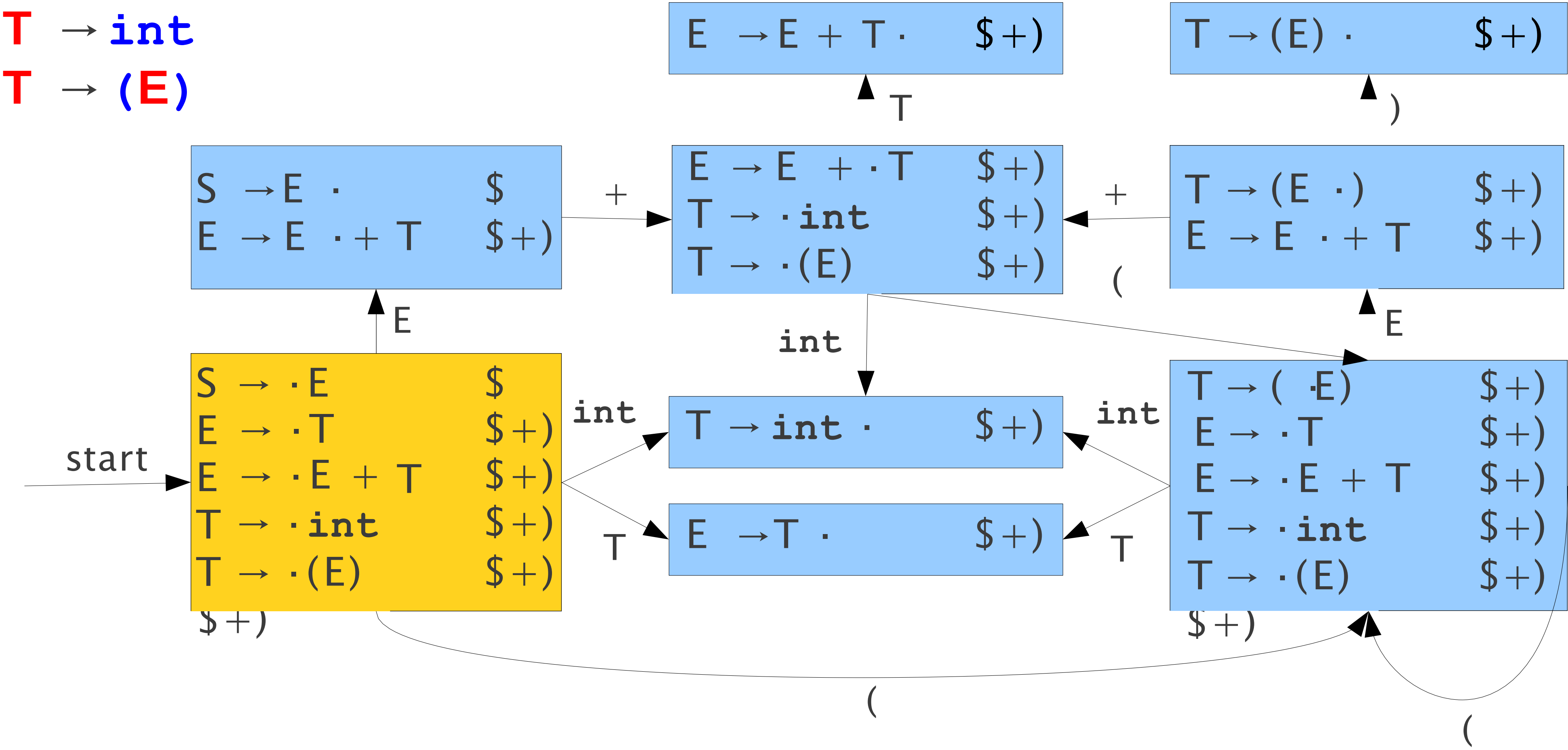
SLR(1) Parsing


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

int	+	(	int	+	int	+	int	)	$

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

SLR(1) Parsing



E → E + T · $+)

T → (E) · $+)

S → E · $
E → E · + T $+)

E → E + · T $+)
T → · int $+)
T → · (E) $+)

T → (E ·) $+)
E → E · + T $+)

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

T → int · $+)

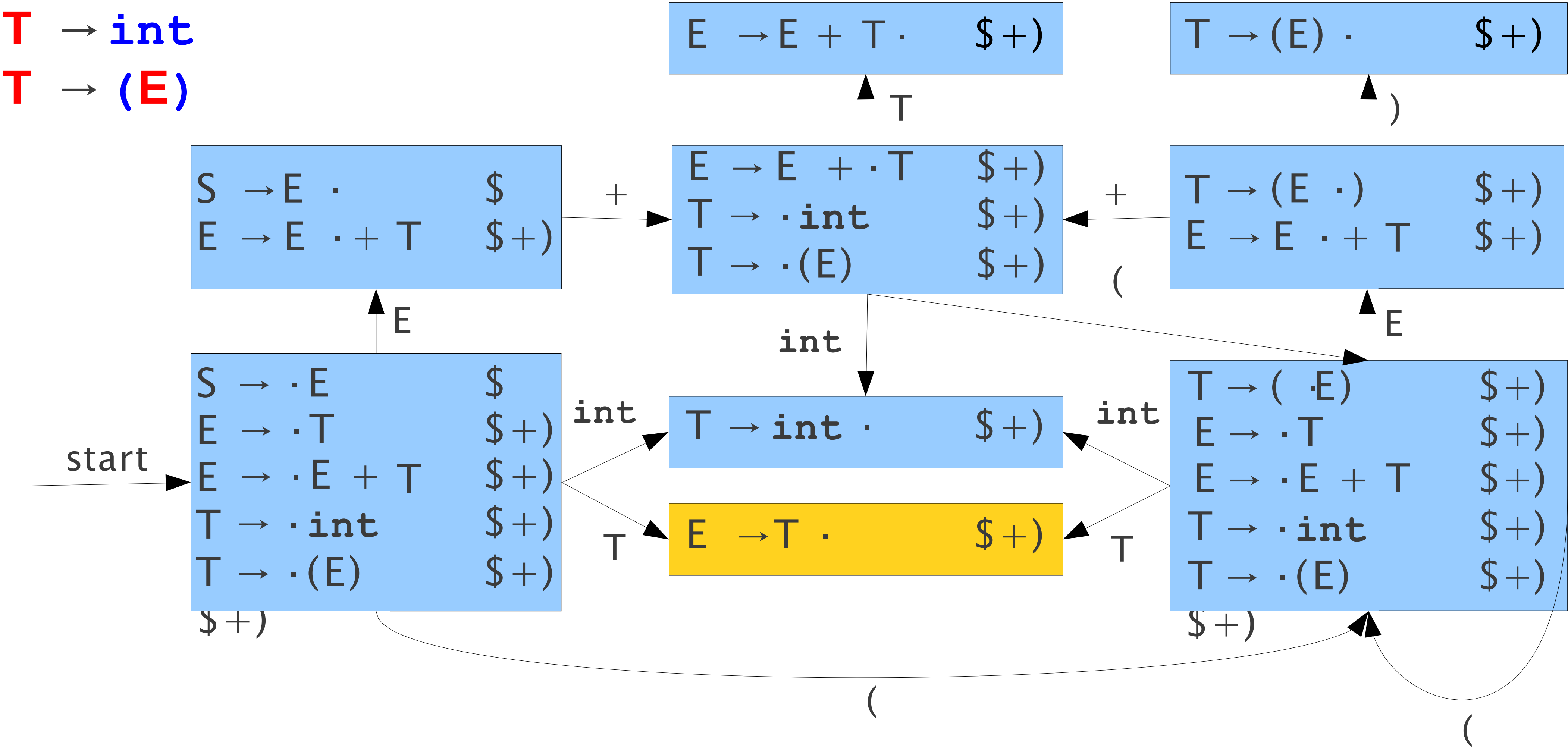
E → T · $+)

T → (·E) $+)
E → · T $+)
E → · E + T $+)
T → · int $+)
T → · (E) $+)
$+)

T		+	(	int	+	int	+	int	)	$

S → E
E → T
E → E + T
T → int
T → (E)

SLR(1) Parsing


E → E + T· $+)

T → (E) · $+)

S → E · $
E → E ·+ T $+)

E → E + ·T $+)
T → ·int $+)
T → ·(E) $+)

T → (E ·) $+)
E → E · + T $+)

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

T → int · $+)

E → T · $+)

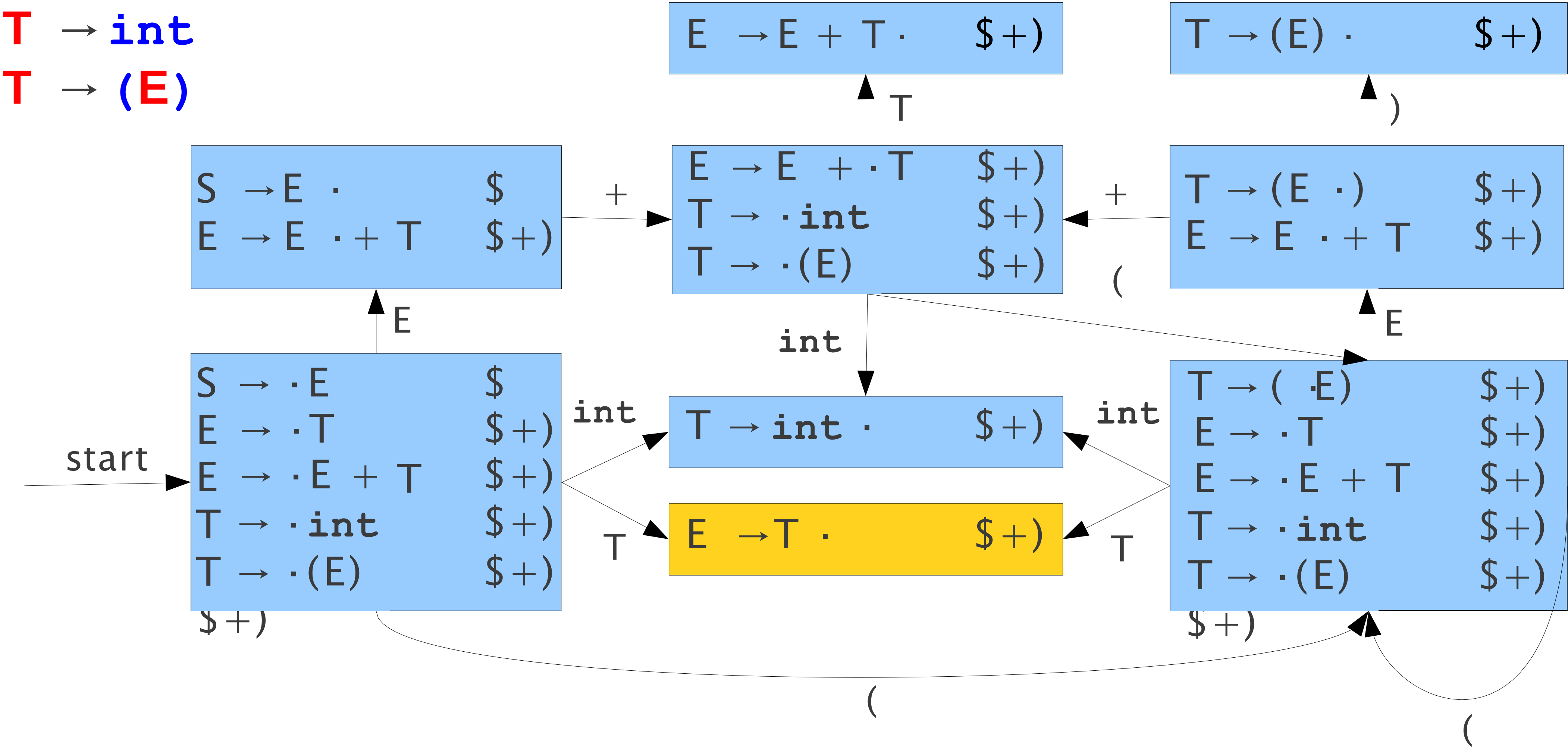
T → (·E) $+)
E → ·T $+)
E → ·E + T $+)
T → ·int $+)
T → ·(E) $+)
$+)

T

+	(	int	+	int	+	int	)	$

S → E
E → T
E → E + T
T → int
T → (E)

SLR(1) Parsing


E → E + T· $+)

T → (E) · $+)

S → E · $
E → E ·+ T $+)

E → E + ·T $+)
T → ·int $+)
T → ·(E) $+)

T → (E ·) $+)
E → E · + T $+)

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

T → int · $+)

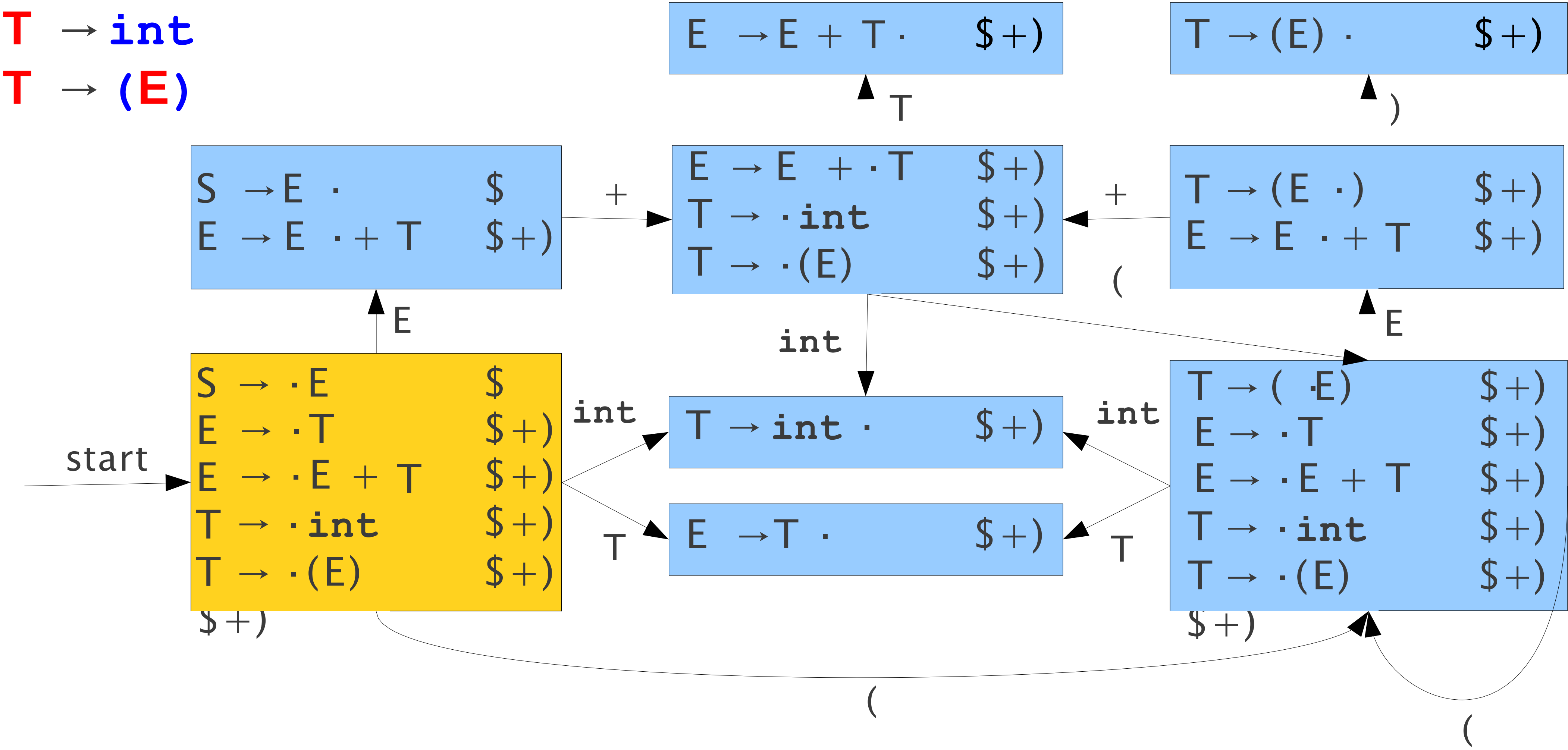
E → T · $+)

T → (·E) $+)
E → ·T $+)
E → ·E + T $+)
T → ·int $+)
T → ·(E) $+)
$+)

T		+	(	int	+	int	+	int	)	$

S → E
E → T
E → E + T
T → int
T → (E)

SLR(1) Parsing



E → E + T · $+)

T → (E) · $+)

S → E · $
E → E · + T $+)

E → E + · T $+)
T → · int $+)
T → · (E) $+)

T → (E ·) $+)
E → E · + T $+)

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

T → int · $+)

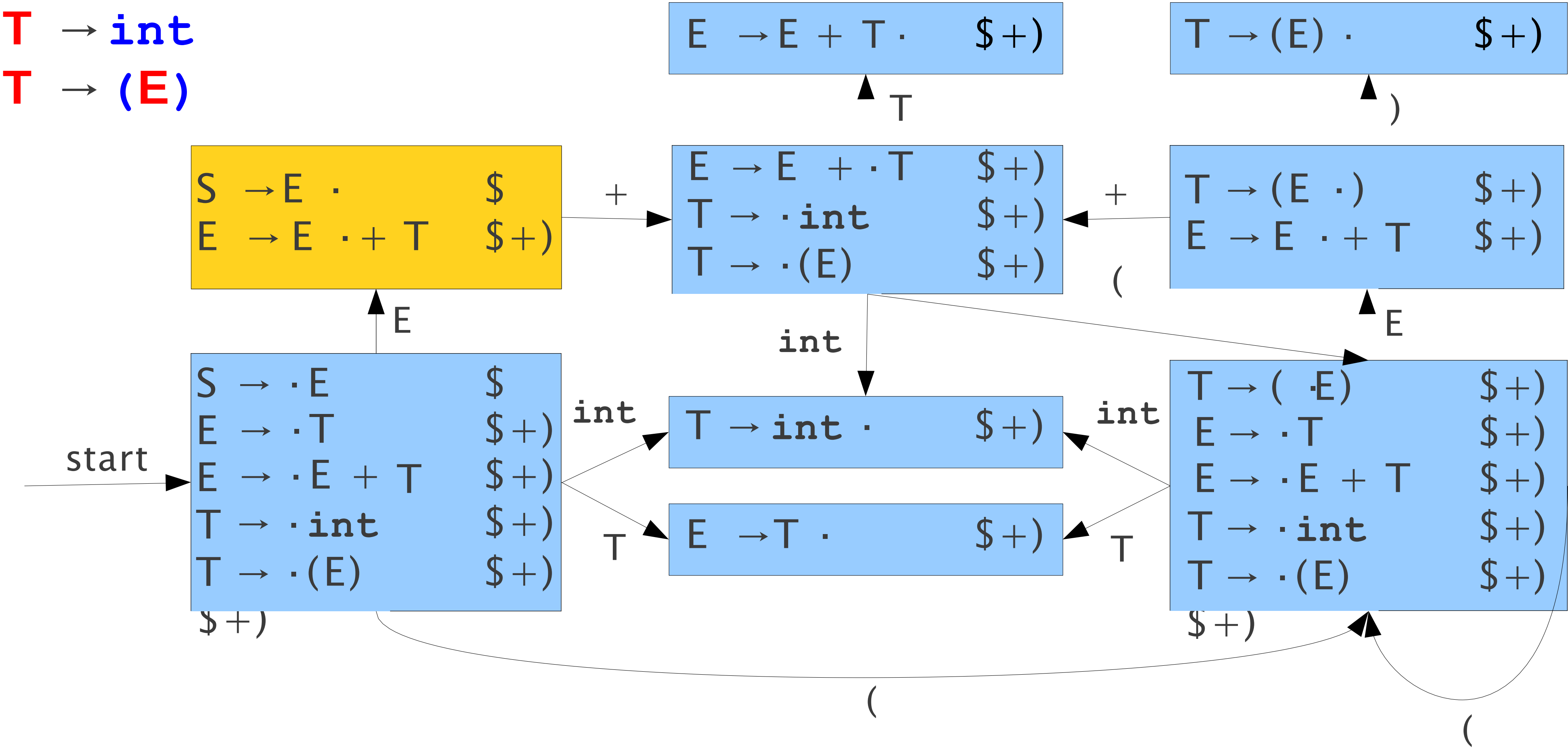
E → T · $+)

T → (· E) $+)
E → · T $+)
E → · E + T $+)
T → · int $+)
T → · (E) $+)

E | + (int + int + int) $

S → E
E → T
E → E + T
T → int
T → (E)

SLR(1) Parsing



E → E + T · $+)

T → (E) · $+)

S → E · $
E → E · + T $+)

E → E + · T $+)
T → · int $+)
T → · (E) $+)

T → (E ·) $+)
E → E · + T $+)

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

T → int · $+)

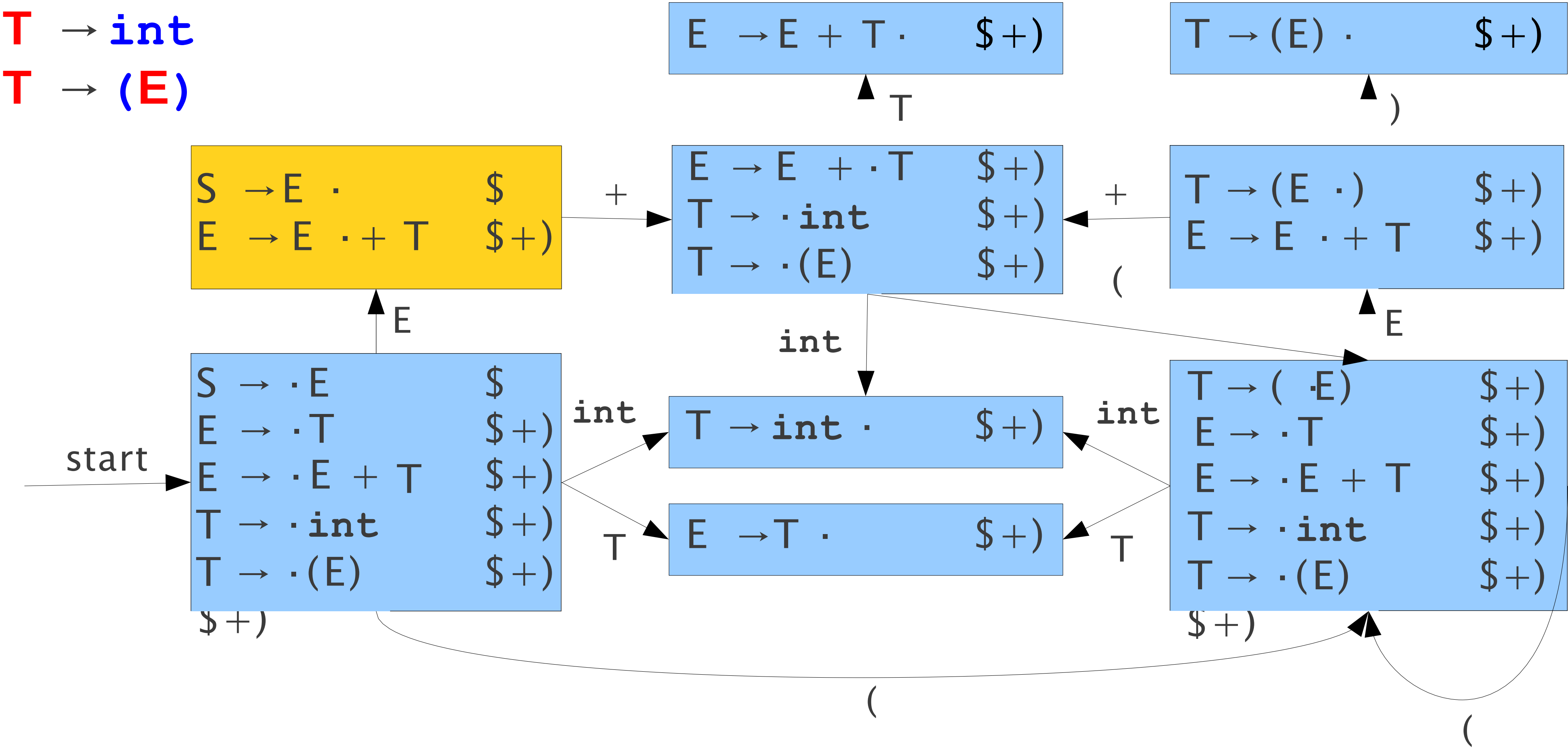
E → T · $+)

T → (·E) $+)
E → · T $+)
E → · E + T $+)
T → · int $+)
T → · (E) $+)

E	|	+	(	int	+	int	+	int	)	$

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

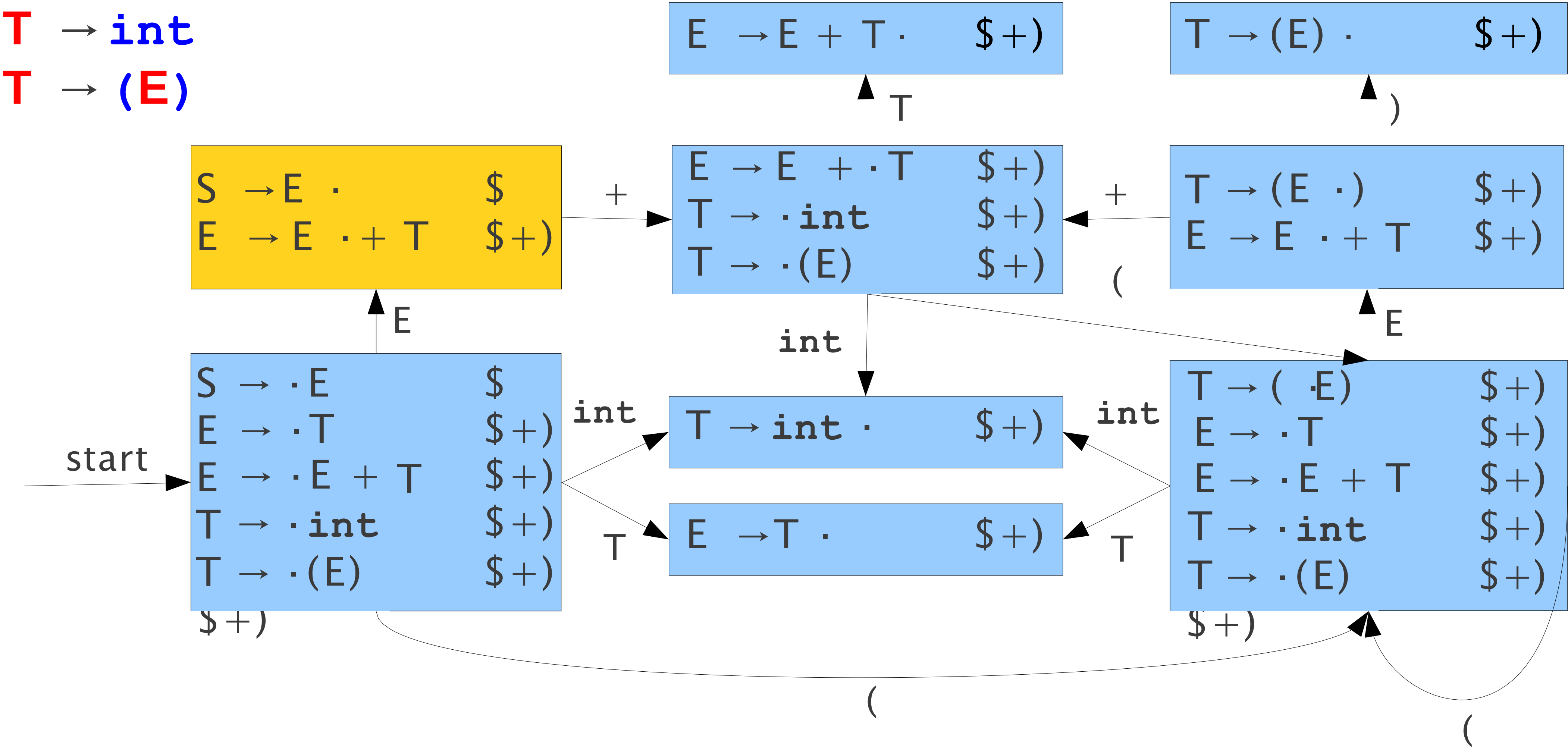
SLR(1) Parsing



E	|	+	(	int	+	int	+	int	)	$

S → E
E → T
E → E + T
T → int
T → (E)

SLR(1) Parsing

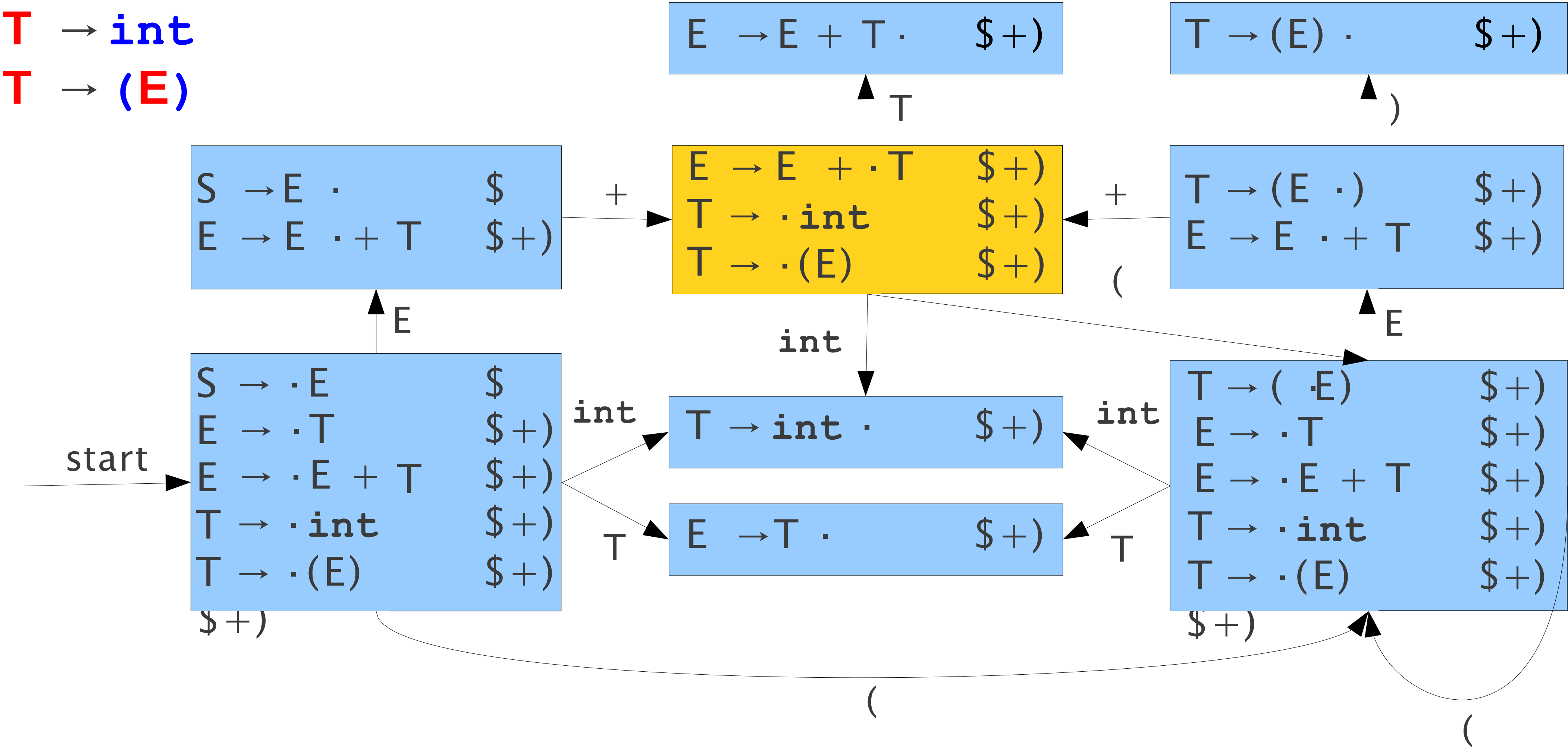


E	+

(	int	+	int	+	int	)	$

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

SLR(1) Parsing



E → E + T · $+)

T → (E) · $+)

S → E · $
E → E · + T $+)

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

T → (E ·) $+)
E → E · + T $+)

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

T → **int** · $+)

E → T · $+)

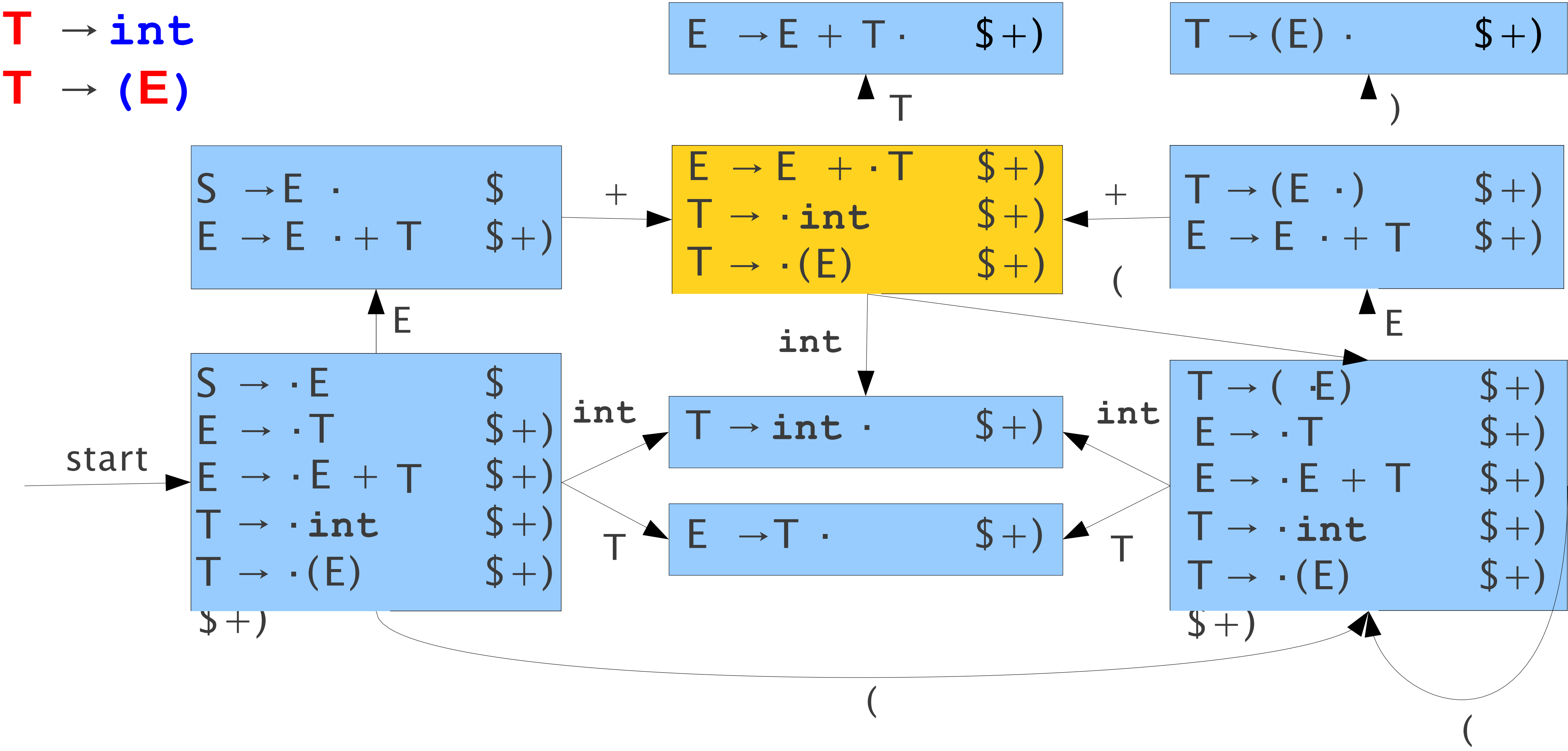
T → (·E) $+)
E → · T $+)
E → · E + T $+)
T → · **int** $+)
T → · (E) $+)

E	+

(	int	+	int	+	int	)	$

S → E
E → T
E → E + T
T → int
T → (E)

SLR(1) Parsing



S → E · $
E → E · + T $+)

E → E + · T $+)
T → · int $+)
T → · (E) $+)

E → E + T · $+)

T → (E) · $+)

T → (E ·) $+)
E → E · + T $+)

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

T → int · $+)

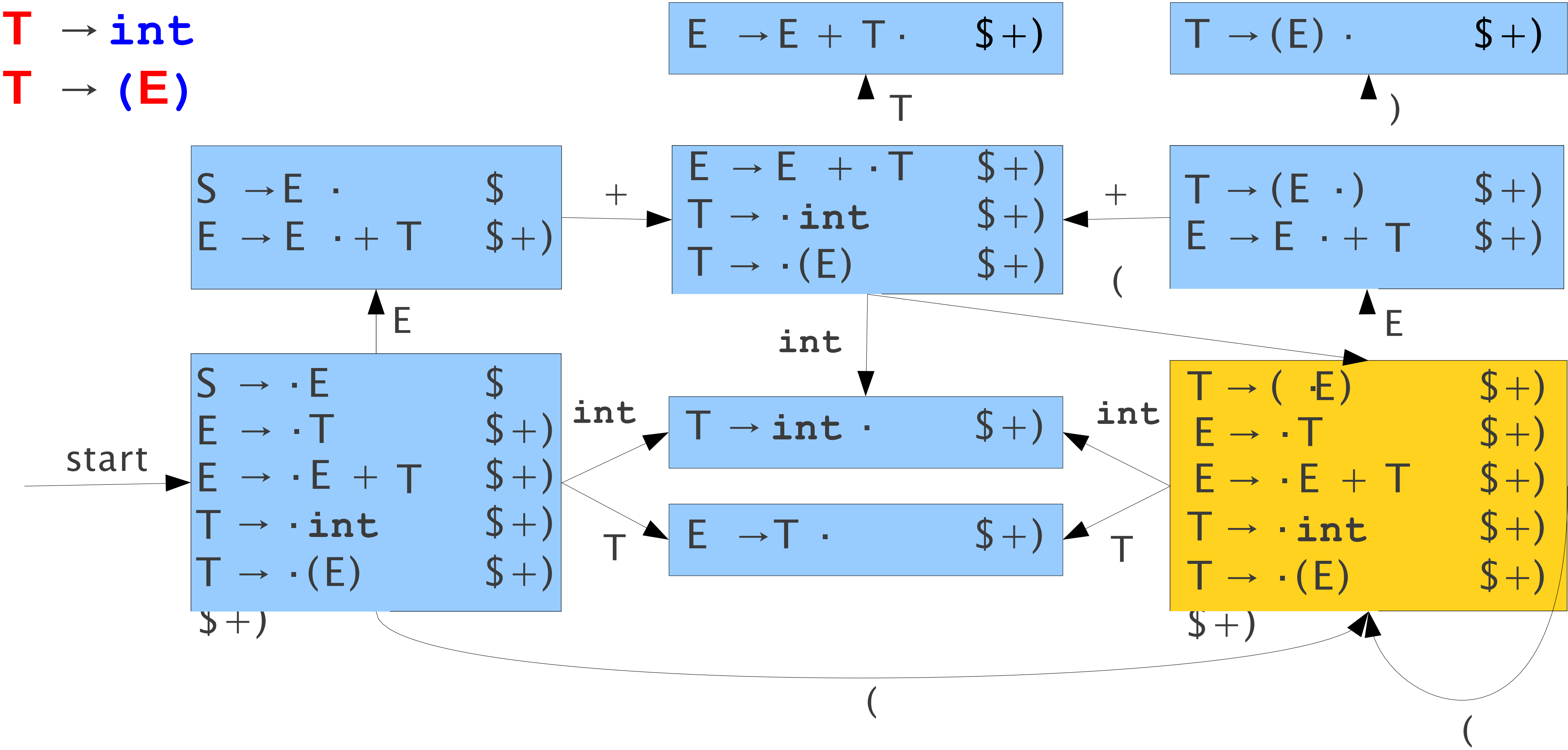
E → T · $+)

T → (·E) $+)
E → · T $+)
E → · E + T $+)
T → · int $+)
T → · (E) $+)
$+)

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)

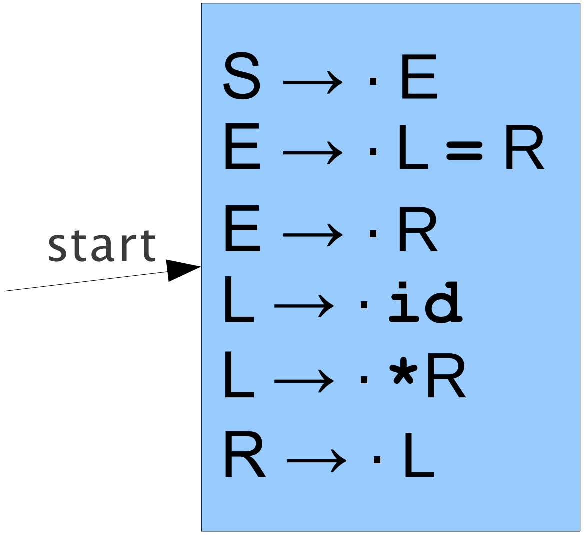
- Exploits lookahead in a small space.
  - Small automaton – same number of states as in as LR(0).
  - Works on many more grammars than LR(0)
- Too weak for most grammars: lose context from not having extra states.

The Limits of SLR(1)

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

The Limits of SLR(1)

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



The Limits of SLR(1)

$S \rightarrow E$

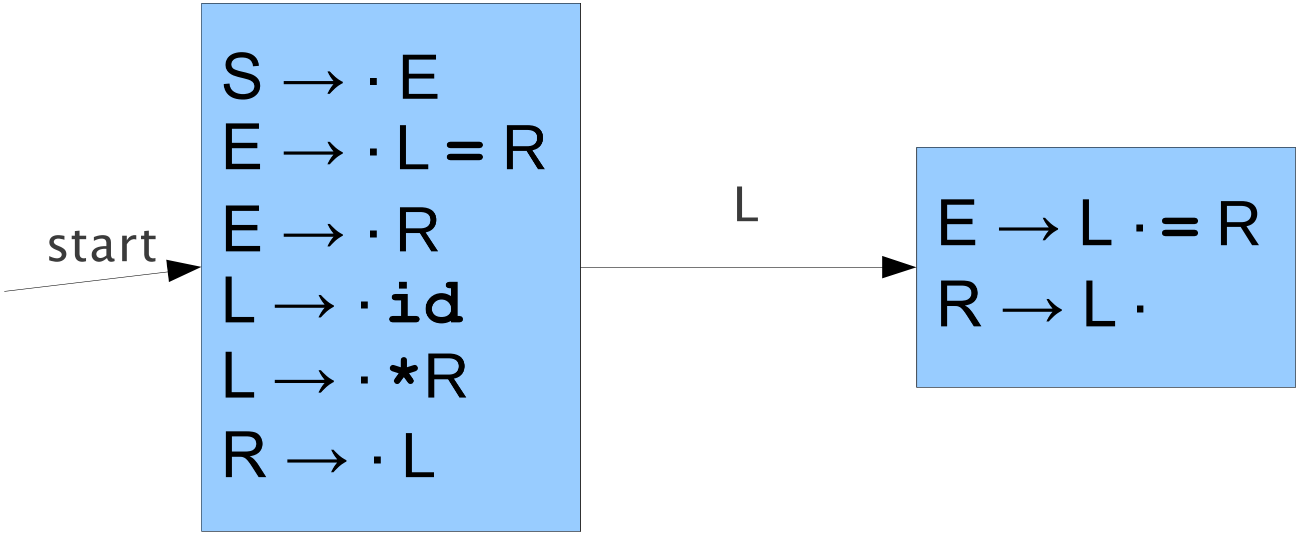
$E \rightarrow L$ **=** R

$E \rightarrow R$

$L \rightarrow$ **id**

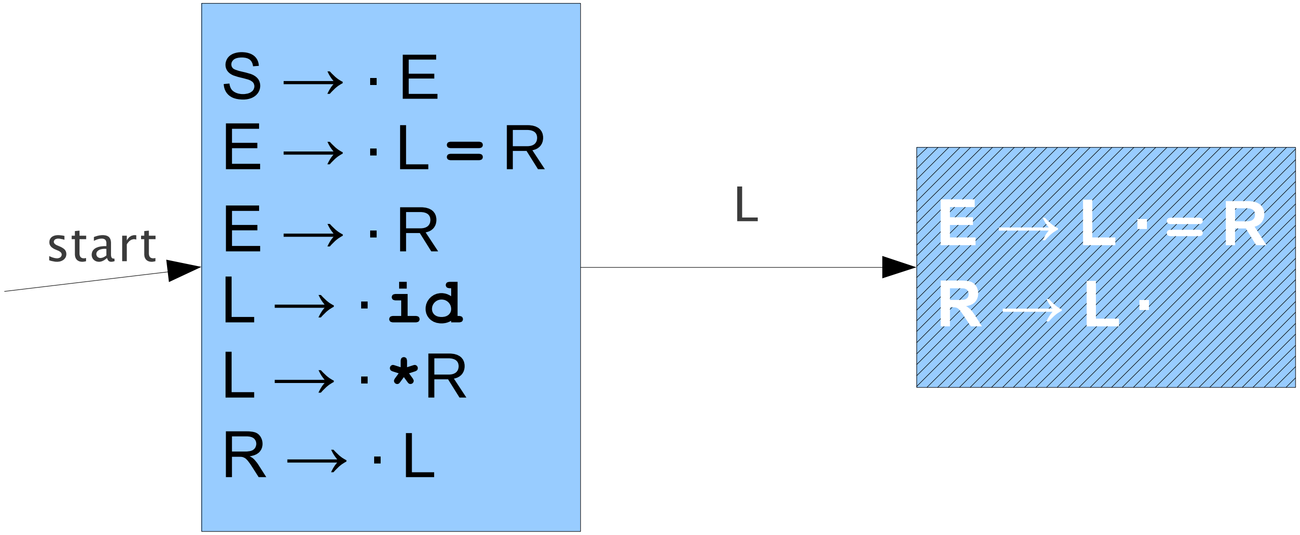
$L \rightarrow$ *****R

$R \rightarrow L$



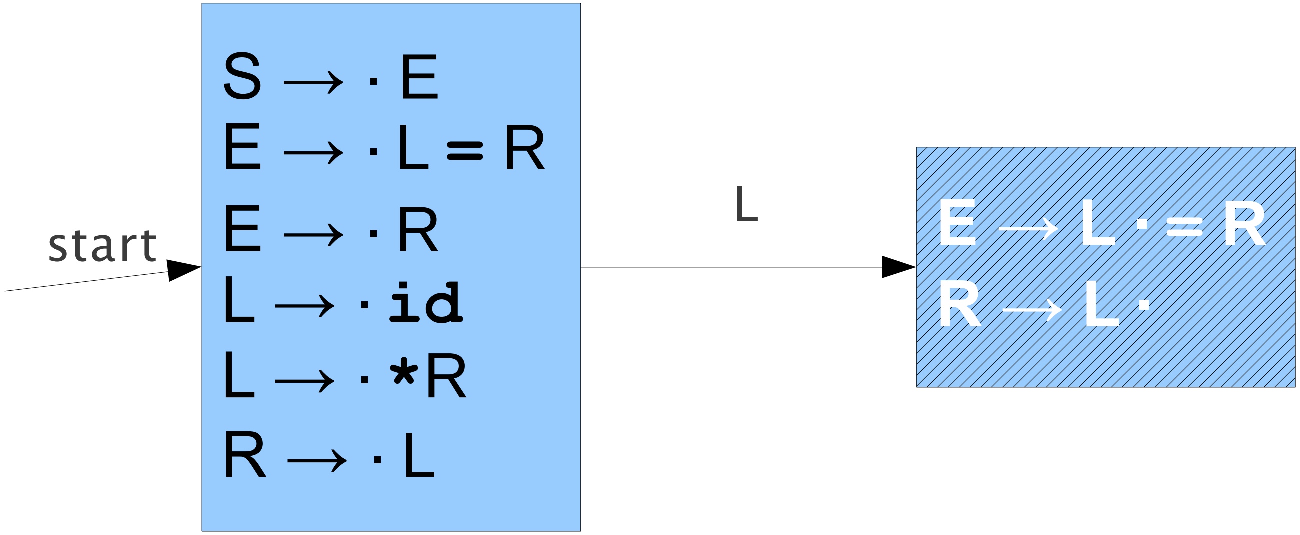
The Limits of SLR(1)

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



The Limits of SLR(1)

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



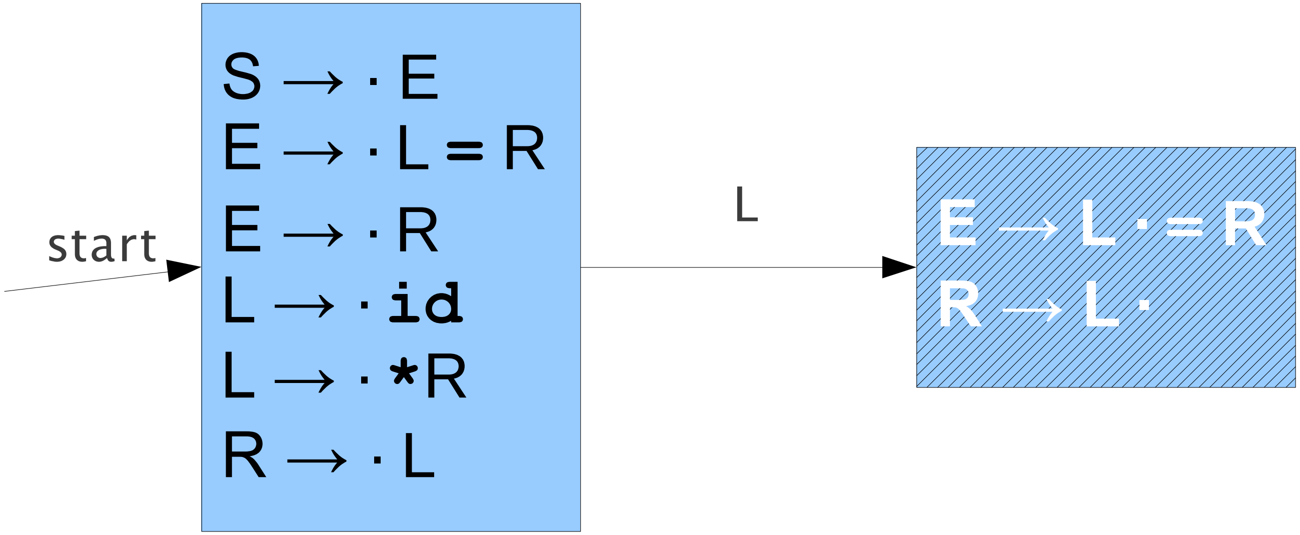
$S \to \cdot E$
$E \to \cdot L = R$
$E \to \cdot R$
$L \to \cdot \texttt{id}$
$L \to \cdot *R$
$R \to \cdot L$

$E \to L \cdot = R$
$R \to L \cdot$

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

The Limits of SLR(1)

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



start

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

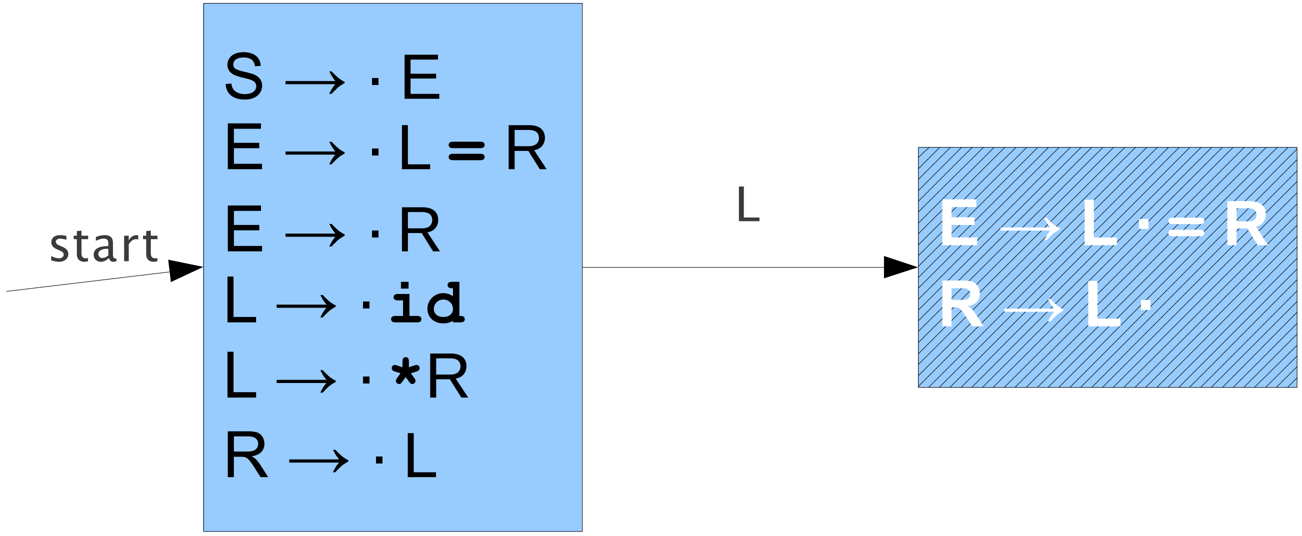
L

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

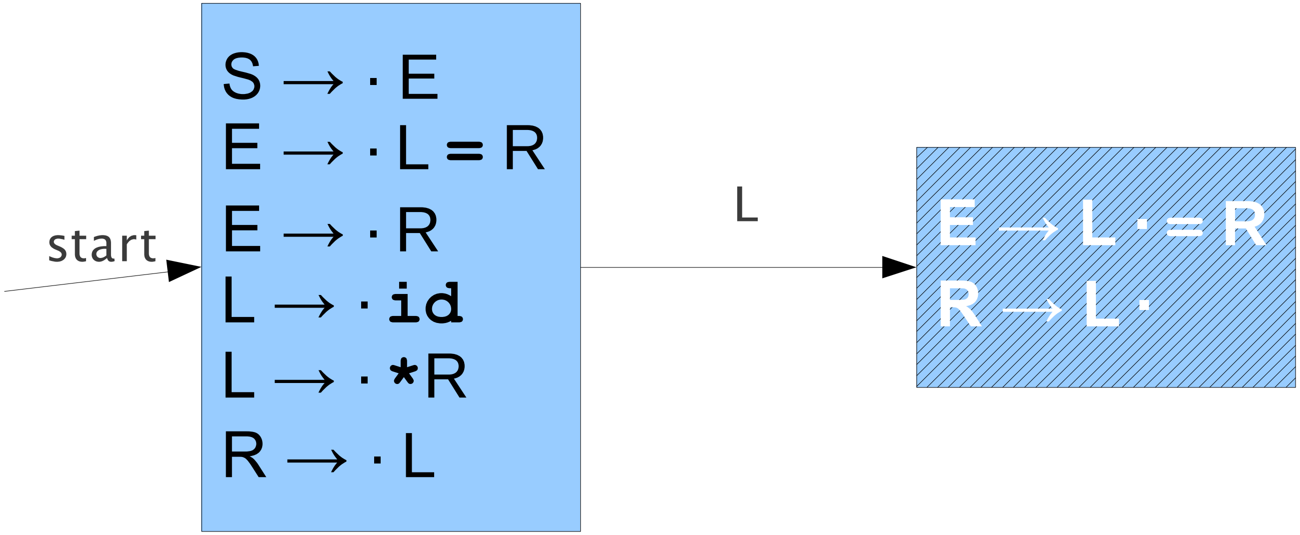


E → L · = R tells us to shift on seeing **=**
R → L · tells us to reduce on FOLLOW(**R**).

= ∈ FOLLOW(**R**).

The Limits of SLR(1)

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


start

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

L

E → L · = R
R → L ·

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

= ∈ FOLLOW(**R**).

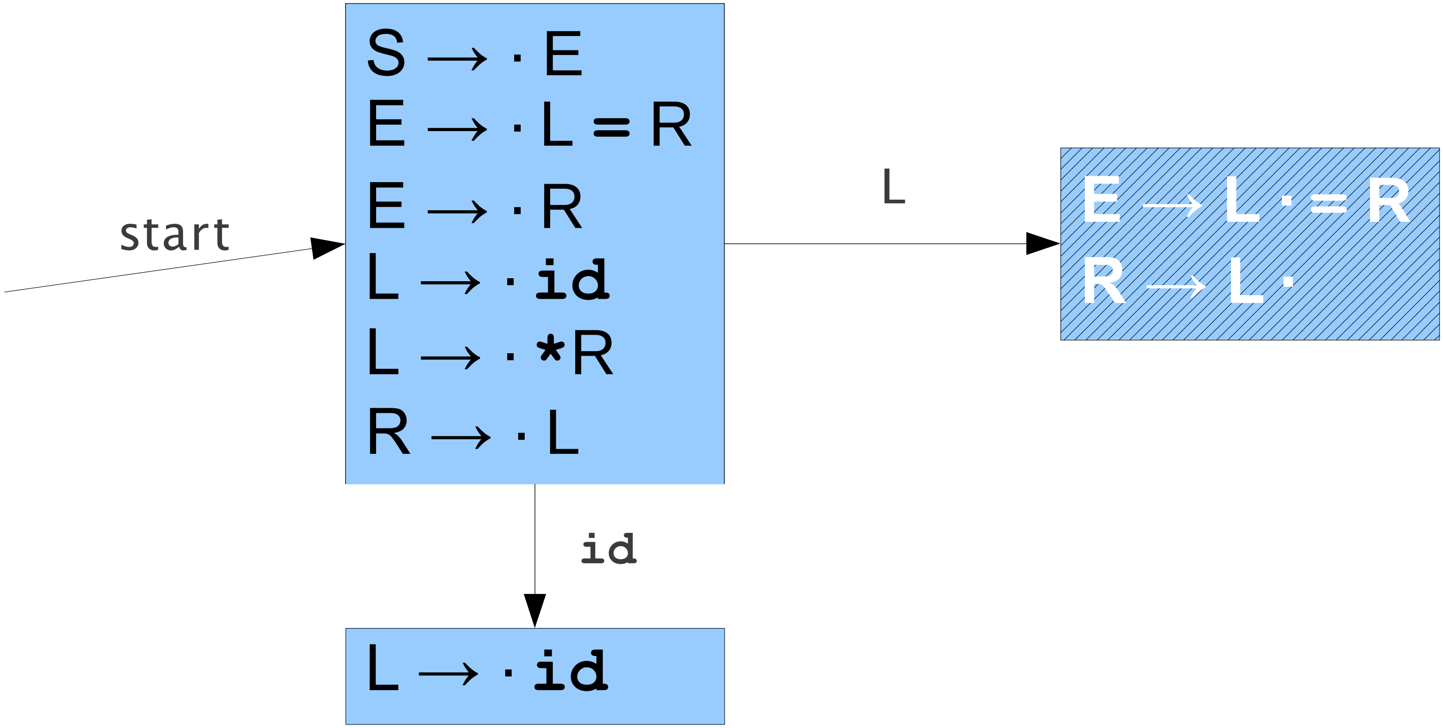
We have a conflict!

Why is SLR(1) Weak?

- With LR(1), incredible contextual information.
  - 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.

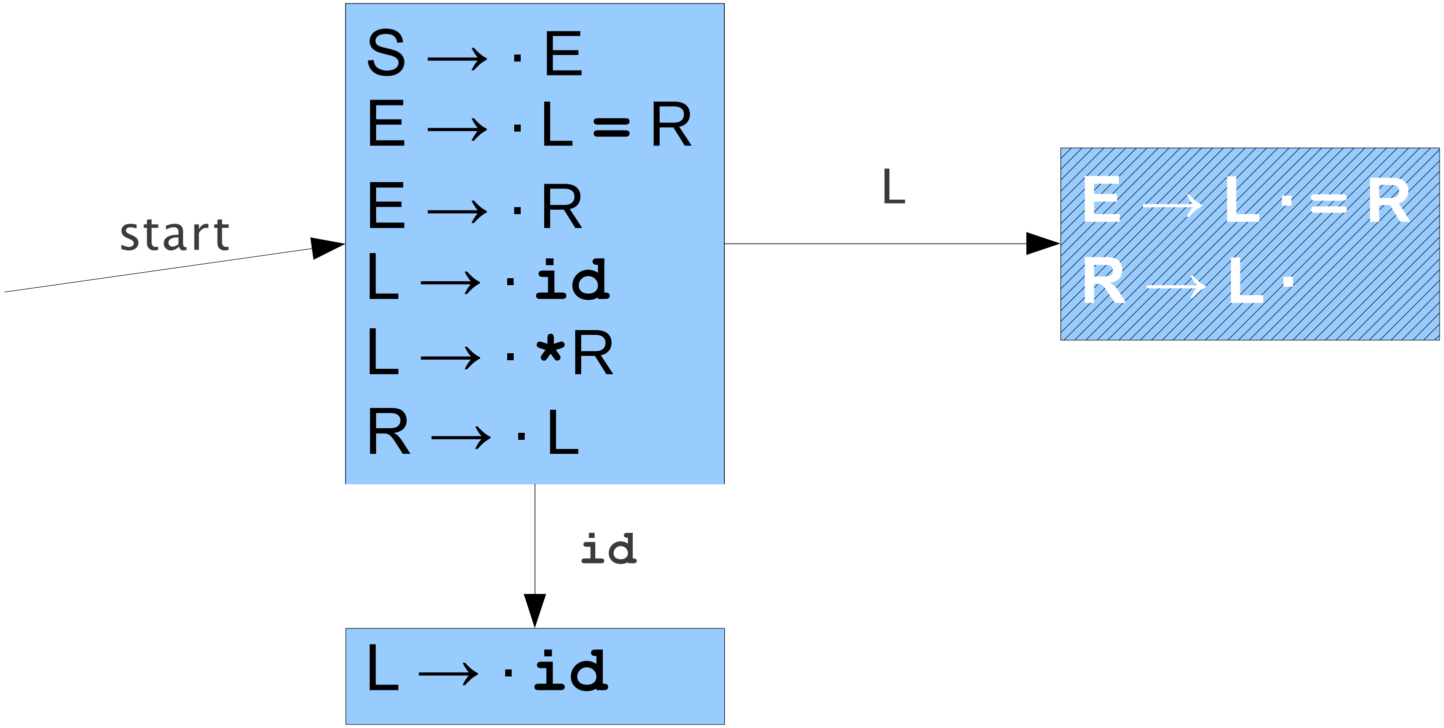
A Lack of Context

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



A Lack of Context

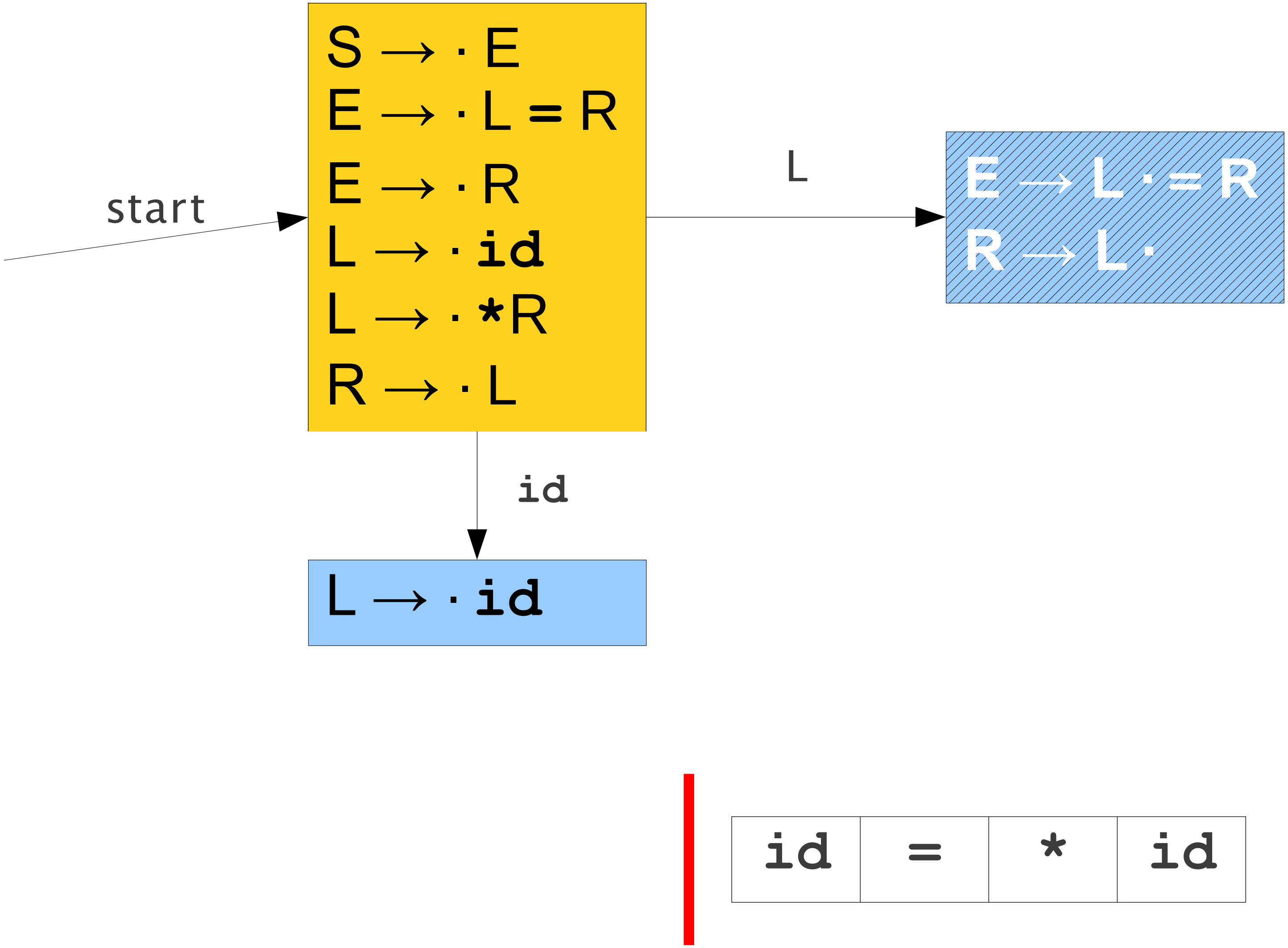
S → E
E → L **=** R
E → R
L → **id**
L → *****R
R → L



id	**=**	*****	**id**

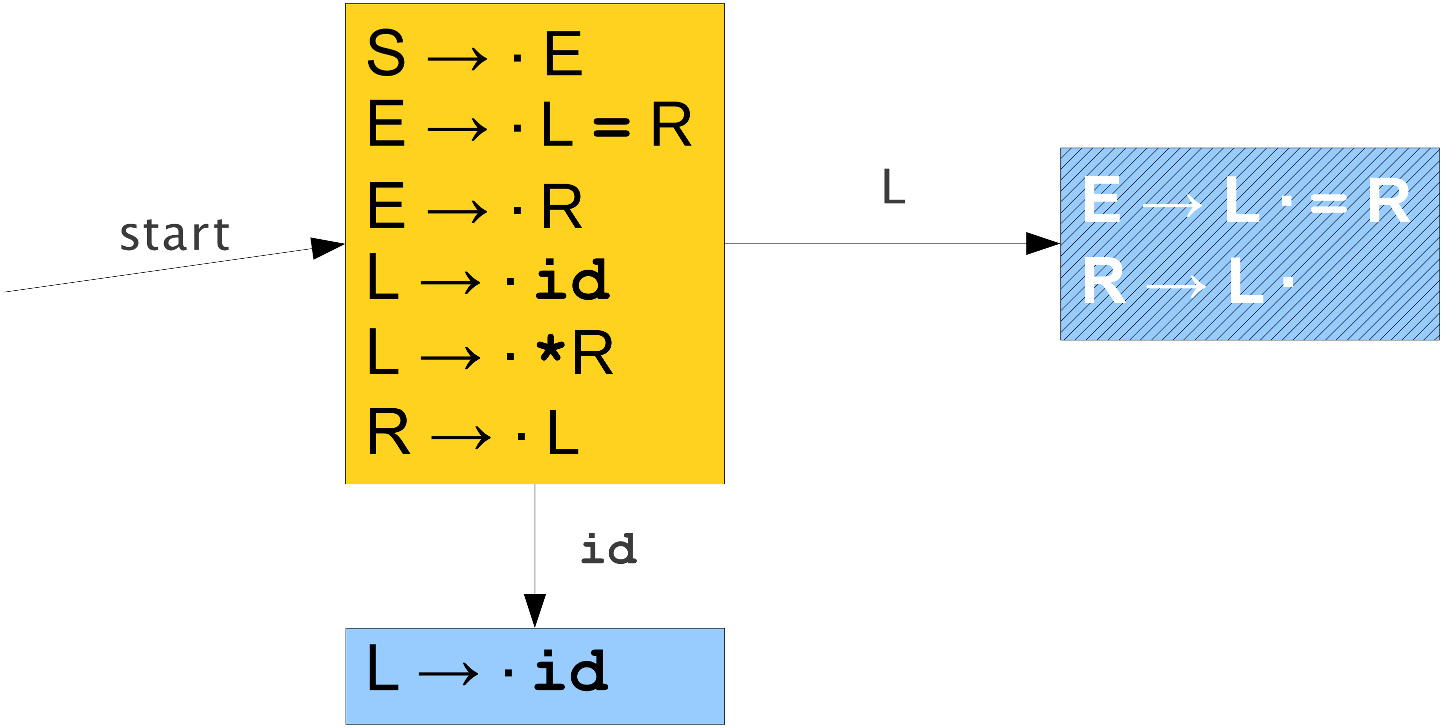
A Lack of Context

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



A Lack of Context

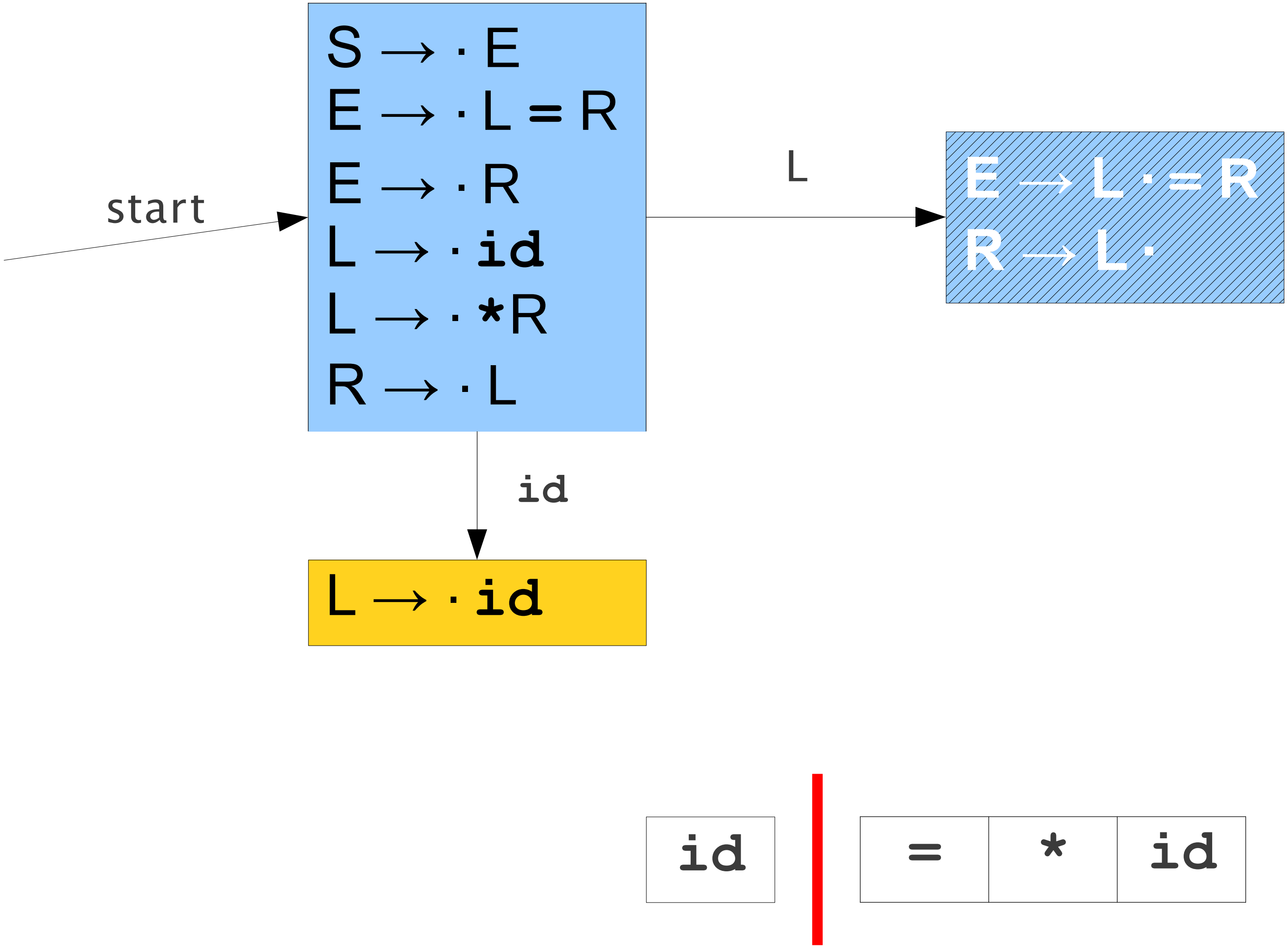
S → E
E → L = R
E → R
L → id
L → *R
R → L



id		=	*	id

A Lack of Context

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



start

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

L

$E \rightarrow L \cdot = R$
$R \rightarrow L \cdot$

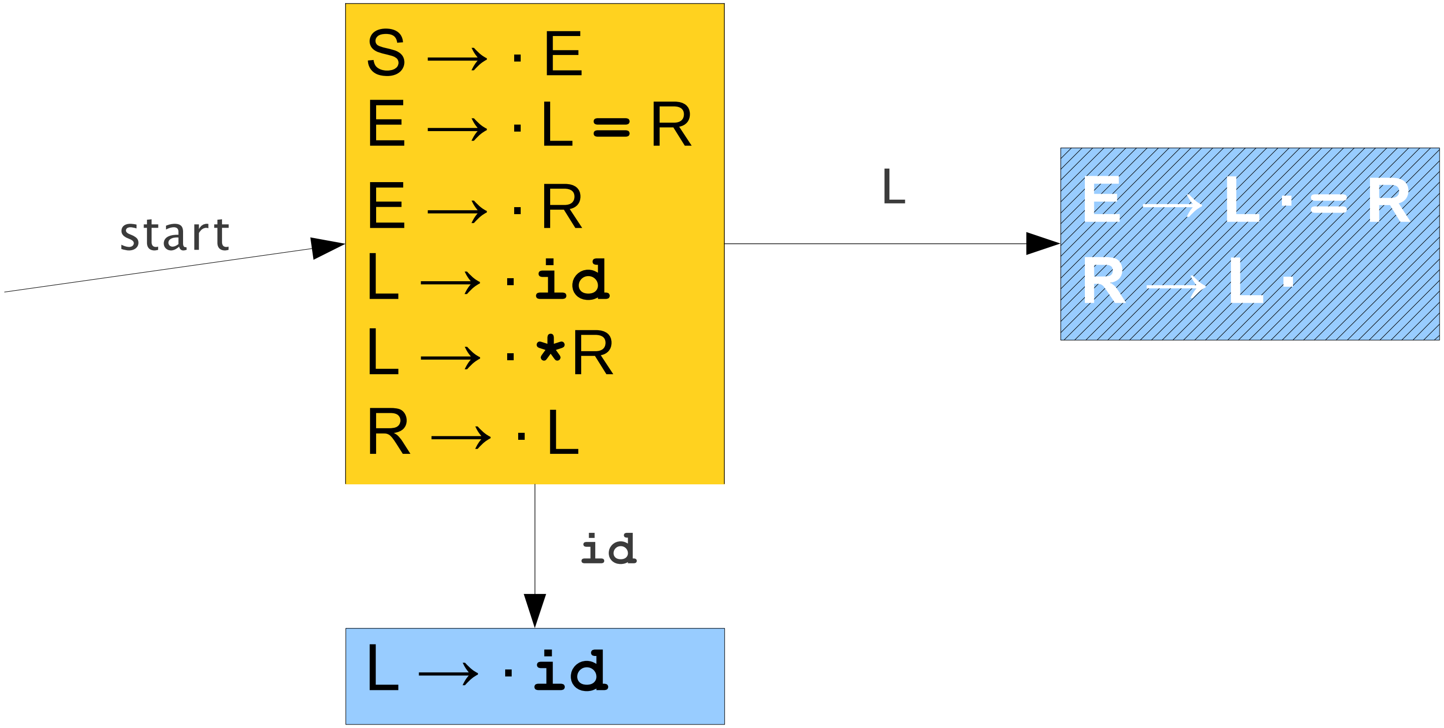
id

$L \rightarrow \cdot \texttt{id}$

id		=	*	id

A Lack of Context

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



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

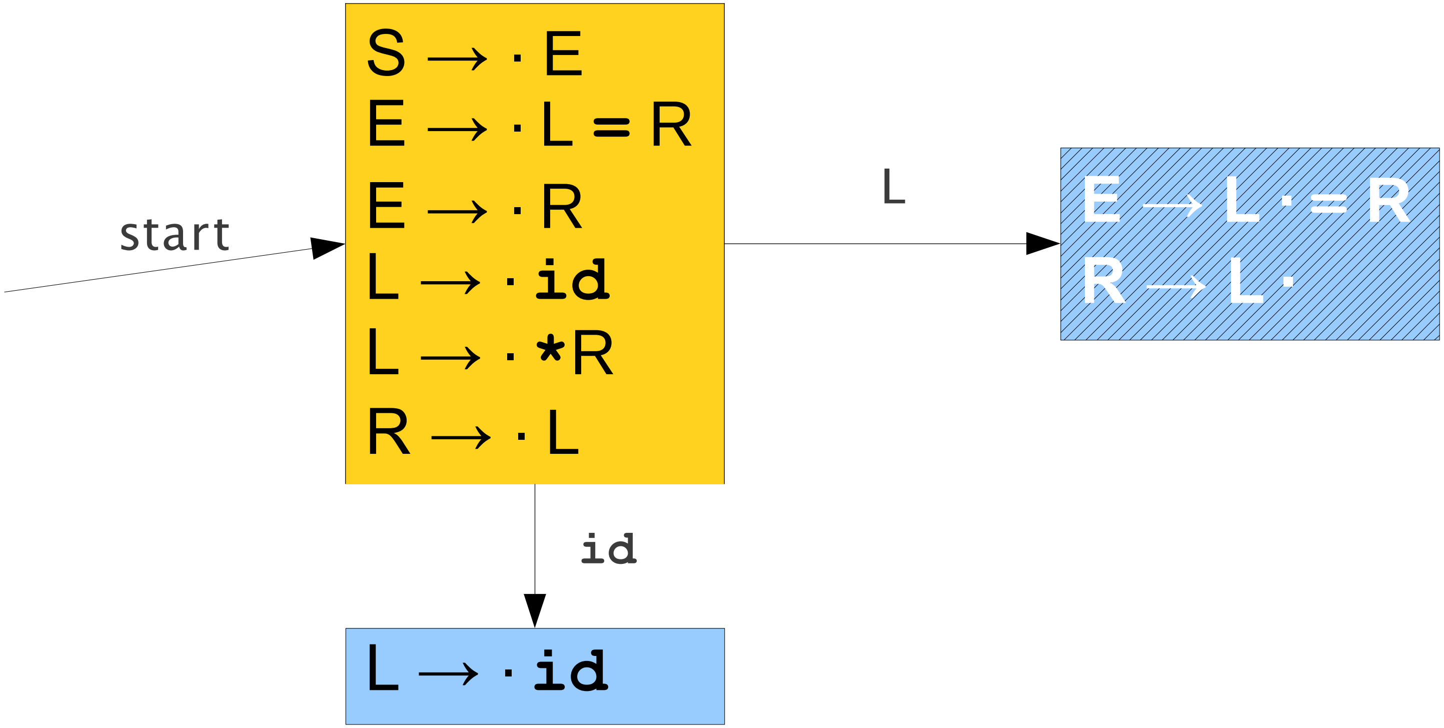
$E \rightarrow L \cdot = R$
$R \rightarrow L \cdot$

$L \rightarrow \cdot \texttt{id}$

=	*	id

A Lack of Context

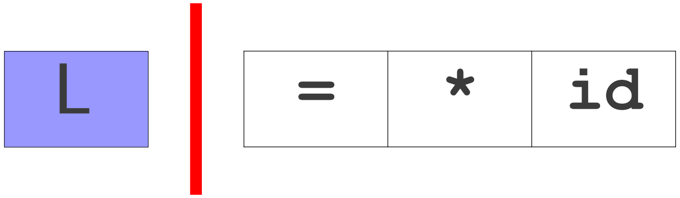
S → E
E → L = R
E → R
L → id
L → *R
R → L



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

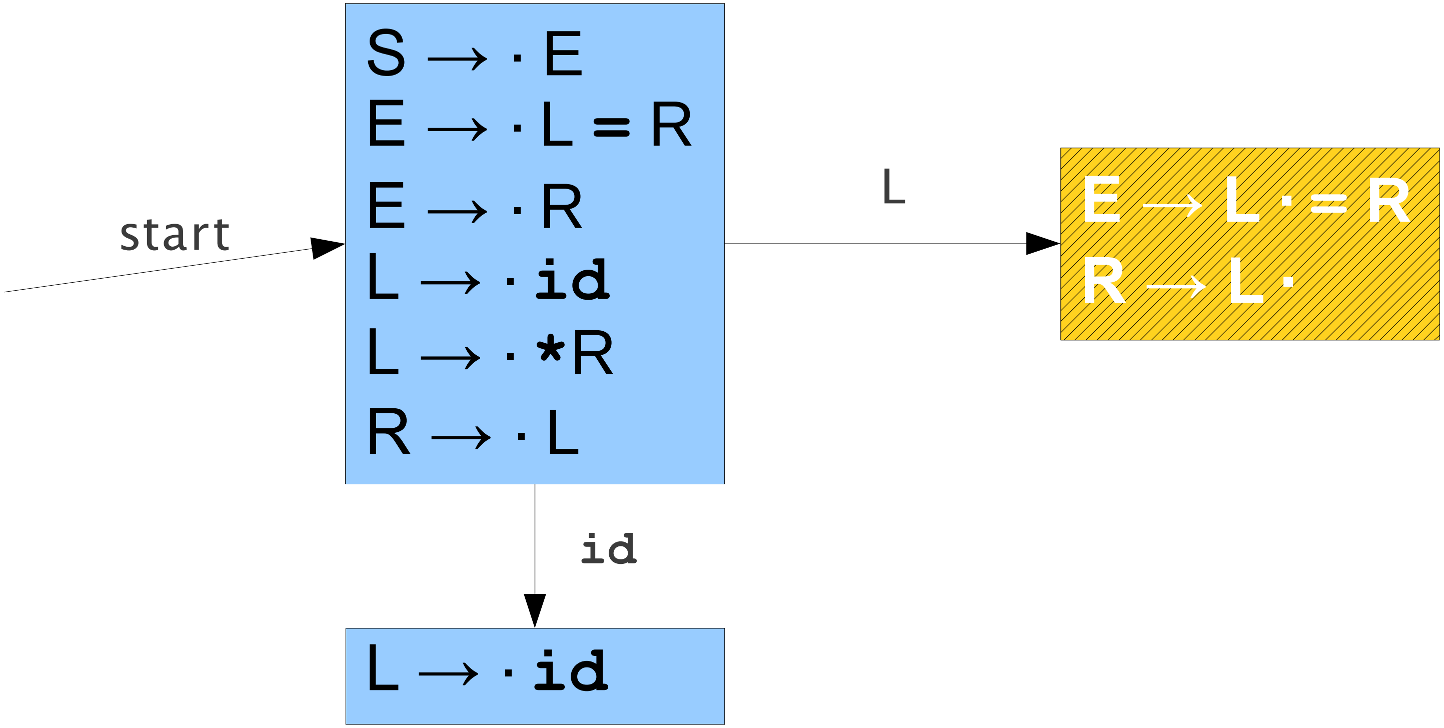
E → L · = R
R → L ·

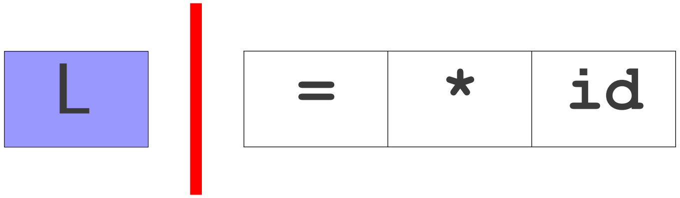
L → · id



A Lack of Context

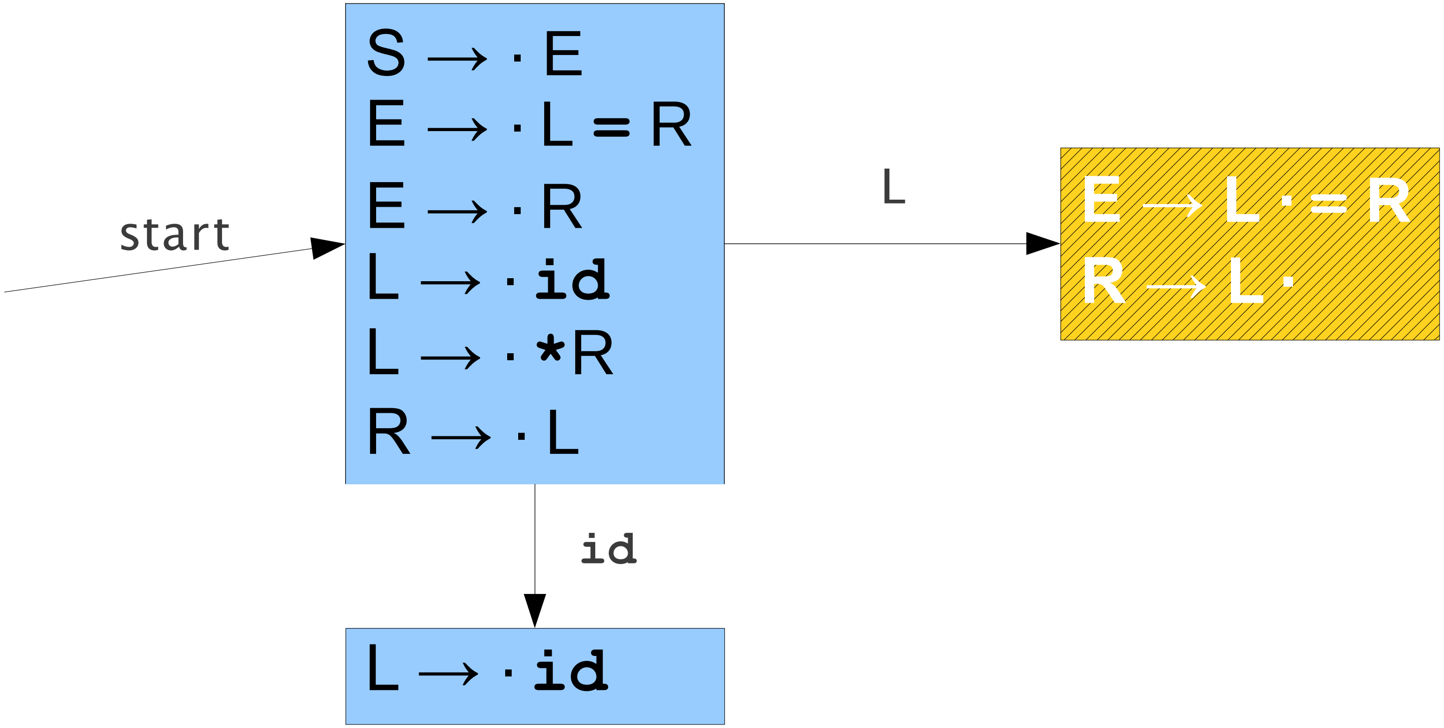
S → E
E → L = R
E → R
L → id
L → *R
R → L





A Lack of Context

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



start

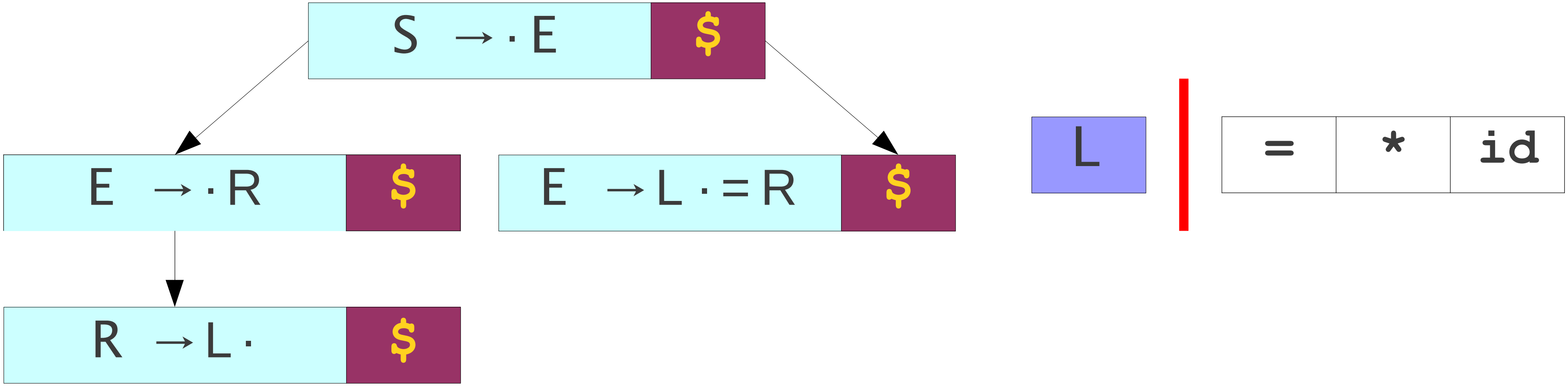
S → · E
E → · L = R
E → · R
L → · **id**
L → · *****R
R → · L

L

E → L · = R
R → L ·

id

L → · **id**



S → · E | $

E → · R | $

E → L · = R | $

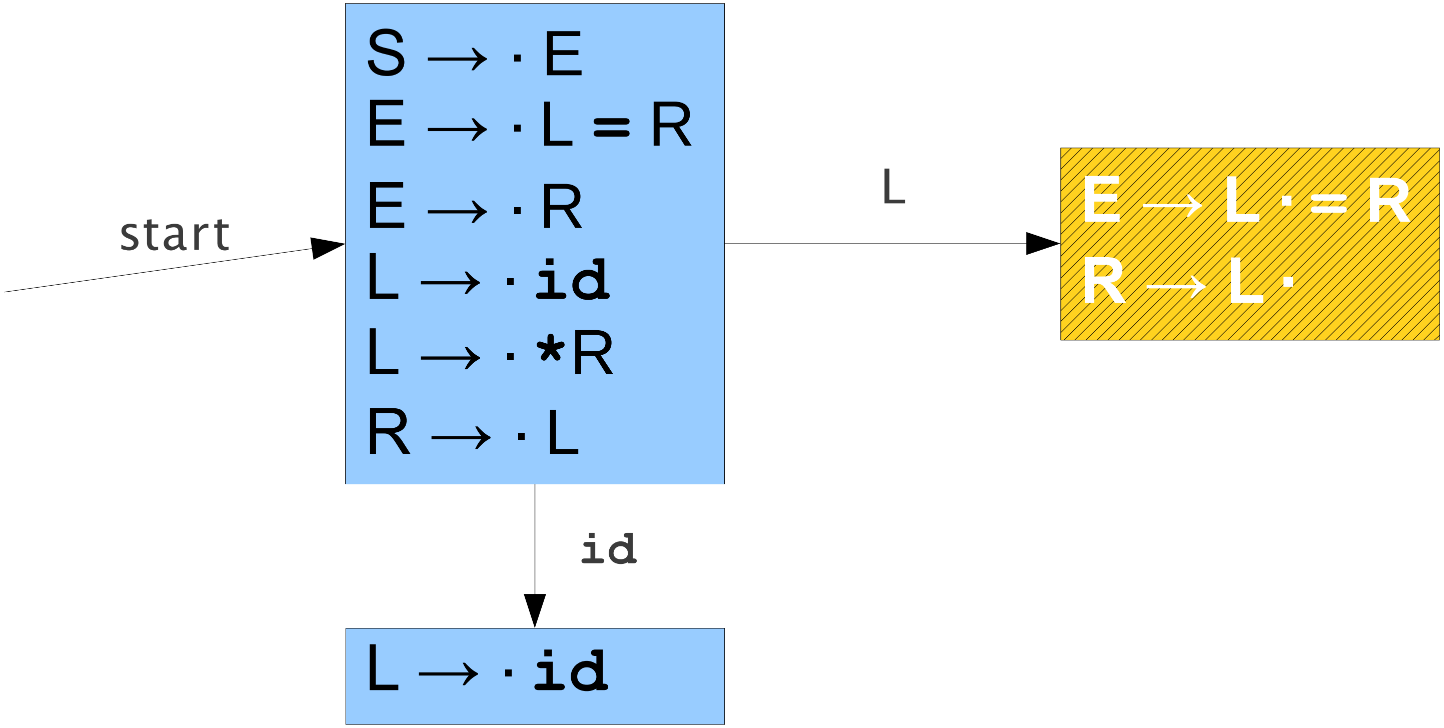
R → L · | $

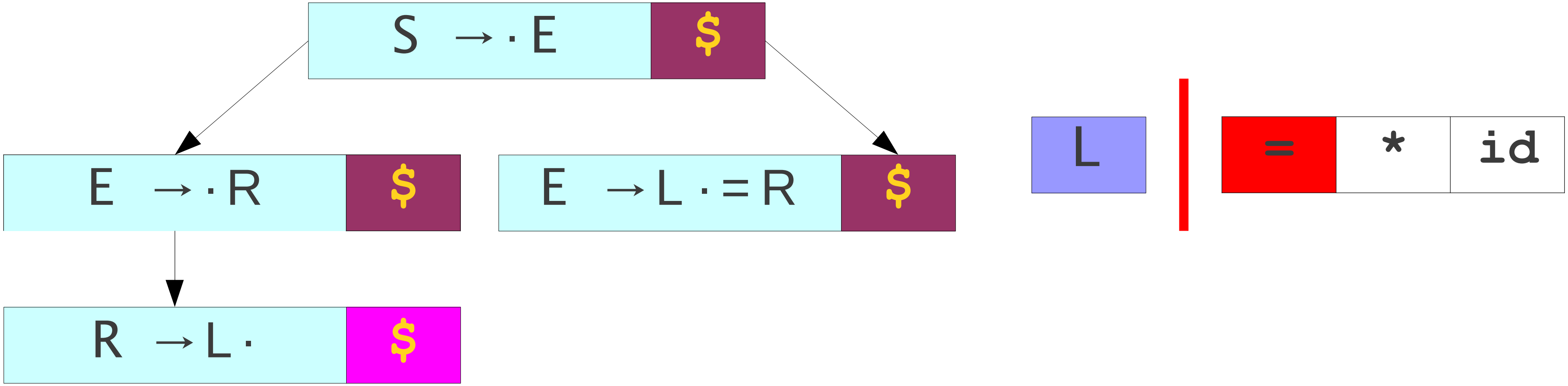
L

= | * | id

A Lack of Context

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



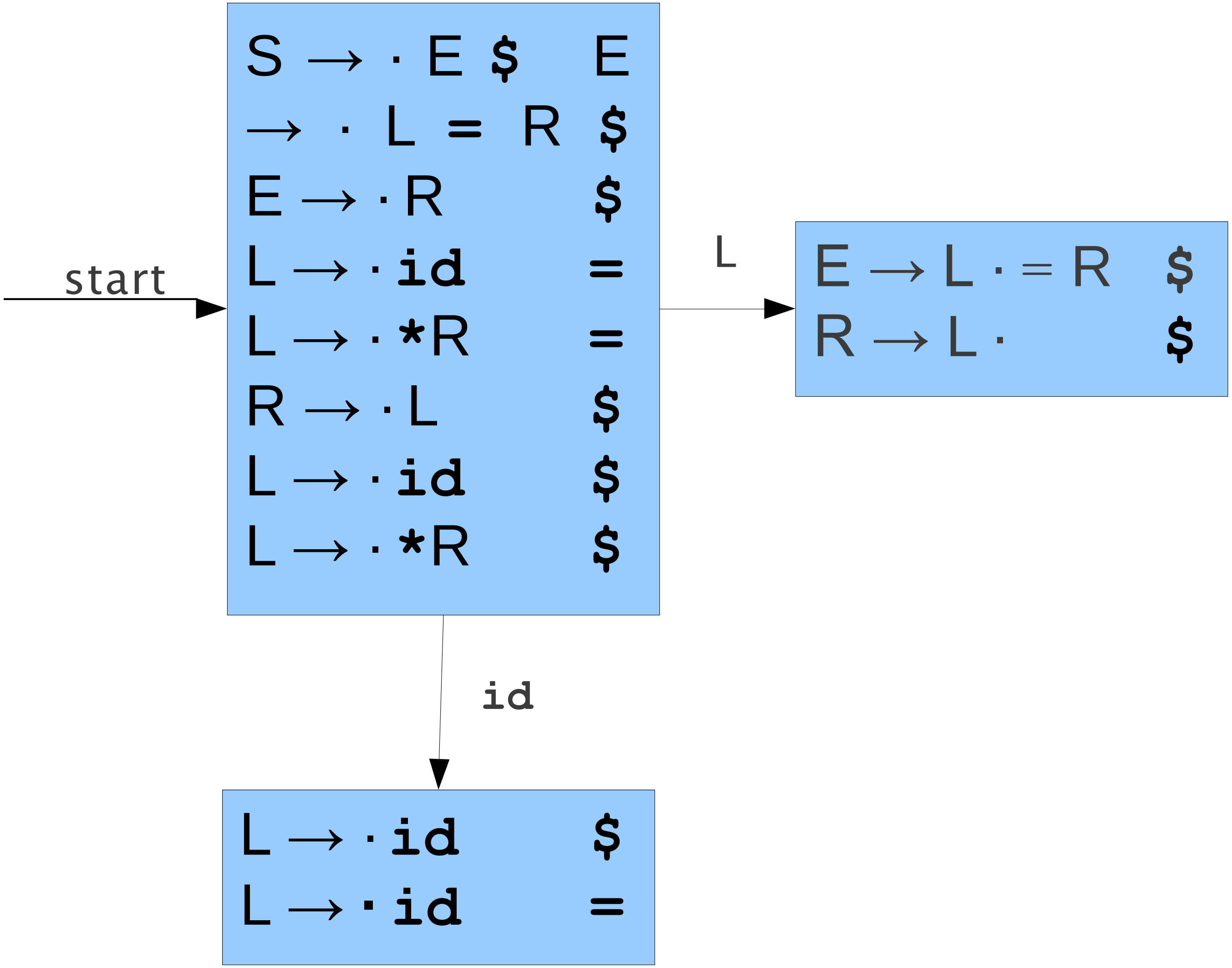


For Reference: LR(1) States

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

For Reference: LR(1) States

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



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

E → L · = R   **$**
R → L ·   **$**

L → · **id**   **$**
L → · **id**   **=**

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 → (·E)	$
E → ·E + T	)
E → ·T	)
T → ·`int`	)
T → ·(E)	)

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

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.

SLR uses FOLLOW sets

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

  FOLLOW(**A**) = { `t` | **S** $\Rightarrow^* \alpha$ **A**`t`ω }

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

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

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

LA and FOLLOW

- The **lookahead set** LA(q, **A** $\rightarrow \gamma$) is defined as

  LA(q, **A** $\rightarrow \gamma$) $= \{$ `t` $\mid$ **S** $\Rightarrow^* \alpha$**A**`t`ω and $\alpha\gamma$ reaches q $\}$

- The **follow set** FOLLOW(**A**) is defined as

  FOLLOW(**A**) $= \{$ `t` $\mid$ **S** $\Rightarrow^* \alpha$**A**`t`ω $\}$

- Note that LA(q, **A** $\rightarrow \gamma$) $\subseteq$ FOLLOW(**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.

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.

Bottom Up Parsing

17 89

GEORGETOWN UNIVERSITY

Bottom-up Parsing

Construct Parse tree from the leaves; bottom up

Produces rightmost derivation

LR Parsers (most common Bottom-Up)

Most common type of shift reduce parser (bottom up)
Left-to-right scan of input
Constructs Rightmost derivation (in reverse)

Characteristics of LR

Can parse essentially all programming language constructs
Most general (non-backtracking) shift reduce parser
Grammars recognized by LR parsers using k lookaheads can recognize all grammars recognized by a LL parser using k lookaheads
LR parsers can detect errors as fast as possible given a left to right scan

Bottom – Up Parsing

Crux: must select correct *handle*

At each reduction step, a RHS matching the substring of the input is replace by the LHS of the production (merging rather than expanding)

Finding the *correct symbol in a right-sentential* form to reduce will result in a right-most derivation in reverse.

β is the **handle** of a right sentential form $\alpha\beta w$ iff $S \Rightarrow^* \alpha A w \Rightarrow \alpha\beta w$

EX
S => aAAe
A => Ac | b

abbce
aAbce
aAAce
aAAe
S

Handle Pruning

The crux of bottom up parsing is finding the correct handle at each step. Parsing is essentially handle pruning – eliminating potential handles that are not the correct handle.

1. Must locate correct substring to reduce
2. Choose handle, production whose LHS we will use to replace.
   There may be more than one matching RHS, which do we choose?

Shift Reduce Parsers

Shift-Reduce Algorithms

Table Driven

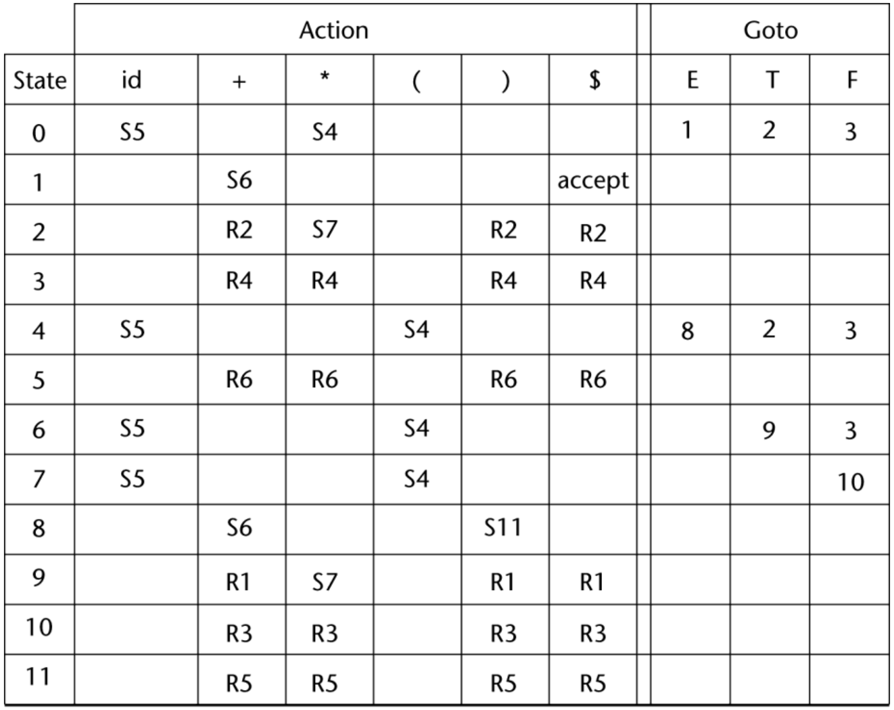
- Tables “difficult” to construct
- Often created using a tool
  - Eg, yacc

4 main operations:

- **Reduce** is the action of replacing the handle on the top of the parse stack with its corresponding LHS
  - For a Reduce, remove the handle from the stack, along with its state symbols. Push the LHS of the rule. Push the state symbol from the GOTO table, using the state symbol just below the new LHS in the stack and the LHS of the new rule as the row and column into the GOTO table
- **Shift** is the action of moving the next token to the top of the parse stack
  - For a Shift, the next symbol of input is pushed onto the stack, along with the state symbol that is part of the Shift specification in the Action table
- **Accept**
- **Reject**

LR Parsing Algorithm

```
// Assume we have an LR parse table
push startState = 0;
LET s = peek(); a = input[ pos ];
do{
                if action[ s, a ] == shift s'{
                push a; push s'; pos++;
                }
        else if action[ s, a ] == reduce A => α{
                pop 2*| α|; // pop α and corresponding state
                s' = peek();
                push A; push goto[ s', A ];
                // chosen handle A => α
                }
        else if action[ s, a ] == accept
                return accept;
        else
                return error;
} while true
```



	Action						Goto		
State	id	+	*	(	)	$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

LR Parsing Example

```
•        // Assume we have an LR parse table
push startState = 0;
do{
        s = peek(); a = input[ pos ];
        if action[ s, a ] == shift s'{
                push a; push s'; pos++;
                }
        else if action[ s, a ] == reduce A => α{
                pop 2*| α|;
                s' = peek();
                push A; push goto[ s', A ];
                // chosen handle A => α
                }
        else if action[ s, a ] == accept
                return accept;
        else
                return error;
} while true
```

Rules

1. E => E + T
2. E => T
3. T => T * F
4. T => F
5. F => (E)
6. F => id

Stack	Input	Action
0	id * id$	Shift 5
0 id 5	* id$	Reduce 6
0 F 3	* id$	Reduce 4
0 T 2	* id$	Shift 7
0 T 2 * 7	id$	Shift 5
0 T 2 * 7 id 5	$	Reduce 6
0 T 2 * 7 F 10	$	Reduce 3
0 T 2	$	Reduce 2
0 E 1	$	Accept

	Action						Goto		
State	id	+	*	(	)	$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

Limitations of LR

LR(1) parsers cannot parse

- Ambiguous grammars
- Grammars where the stack content and the next token are insufficient to determine handle

These concerns can be “fixed” in many instances through modification of grammar, a more detailed tokenizer, or the use of static semantics (attribute grammars).

Constructing LR tables

There are a variety of LR parsing algorithms. The differences are based on different ways to produce the LR parsing table.

These parse tables will generally have hundreds or even thousands of states

- sLR (simpleLR) tables
- LALR tables
- LR tables

Steps:

- Produce augmented grammar: add a start state with just 1 RHS
- Must produce states
  - Via CLOSURE operation
  - Via goto function
- Must produce action table
- Must produce goto table (from goto function)

SLR tables

LR(0) **items** – represent all productions at different stages of parsing (using ‘.’ as pointer). For all productions

A => ABC

<u>LR(0) Items</u> for this rule:

A =>.ABC
A =>A.BC
A =>AB.C
A =>ABC.

Determine the states using CLOSURE and GOTO

CLOSURE(I)

This characterizes a “state” of parsing. Group all similar LR(0) items into the <u>same state.</u>

1. ***Items*** in I are added to I
2. If A=> α.Bβ is in closure(I) and B => γ is a production, then add the item B => .γ to I. Repeat 2 until no more additions can be made.

Canonical Collection C created from closure of kernel items

Kernel items: initial item S’=>.S and all items with ‘.’ not on left

Non-kernel items: the rest

GOTO(I,X) where I is a set of items and X is a grammar symbol

this characterizes the “transition conditions” between states , goto (I,X) is the closure of the set of all items A => αX.β such that A => α.Xβ is in I

Construction of SLR table

Algorithm. Construct SLR table.

1. Construct canonical collection of states $I=\{I_1,\ldots,I_n\}$
2. Create Initial State from closure of E’ => E
3. State i corresponds to I_i
   1. If A => α . a β is in I_i And goto(I_i,a) = I_j then set action[i,a] to shift j
   2. If A => α . is in I_i then set action[i,a] to reduce A => a for all a in FOLLOW(A). A cannot be start.
   3. If E’ => E is in I_i , then action[i,$] = accept
4. For non terminals A, If goto(I_i,A) = I_j , then set goto[i,A]. //*Set goto table.*
5. All empty spots in tables are considered syntax errors

Example

Grammar

E = >E + T | T

T => (E) | id

Augmented Grammar (enumerating the rules allows us to use shorthand later)

1) E’ = > E
2) E => E + T
3) E => T
4) T => (E)
5) T => id

LR(0) items

E’ = > .E
E’ = > E.

E => .E + T
E => E. + T
E => E +. T
E => E + T.

E => .T
E => T.

T => .(E)
T => (.E)
T => .(E.)
T => (E).

T => .id
T => id.

Algorithm. Construct SLR table.

1. Construct canonical collection of states $I=\{I_1,\ldots,I_n\}$
2. Create Initial State from closure of E’ => .E
3. State i corresponds to I_i
    1. If A => α . a β is in I_i And goto(I_i,a) = I_j then set action[i,a] to shift j
    2. If A => α . is in I_i then set action[i,a] to reduce A => a for all a in FOLLOW(A). A cannot be start.
    3. If E’ => E is in I_i , then action[i,$] = accept
4. For non terminals A, If goto(I_i,A) = I_j , then set goto[i,A]. //*Set goto table.*
5. All empty spots in tables are considered syntax errors

Example SLR: Build GOTO()

Algorithm. Construct SLR table.

1. Construct canonical collection of states $I=\{I_1,\ldots,I_n\}$
2. Create Initial State from closure of E’ => .E
3. State i corresponds to I_i
    1. If A => α . a β is in I_i and goto(I_i,a) = I_j then set action[i,a] to shift j
    2. If A => α . is in I_i then set action[i,a] to reduce A => a for all a in FOLLOW(A). A cannot be start.
    3. If E’ => E is in I_i , then action[i,$] = accept
4. For non terminals A, If goto(I_i,A) = I_j , then set goto[i,A]. *//Set goto table.*
5. All empty spots in tables are considered syntax errors

LR(0) items

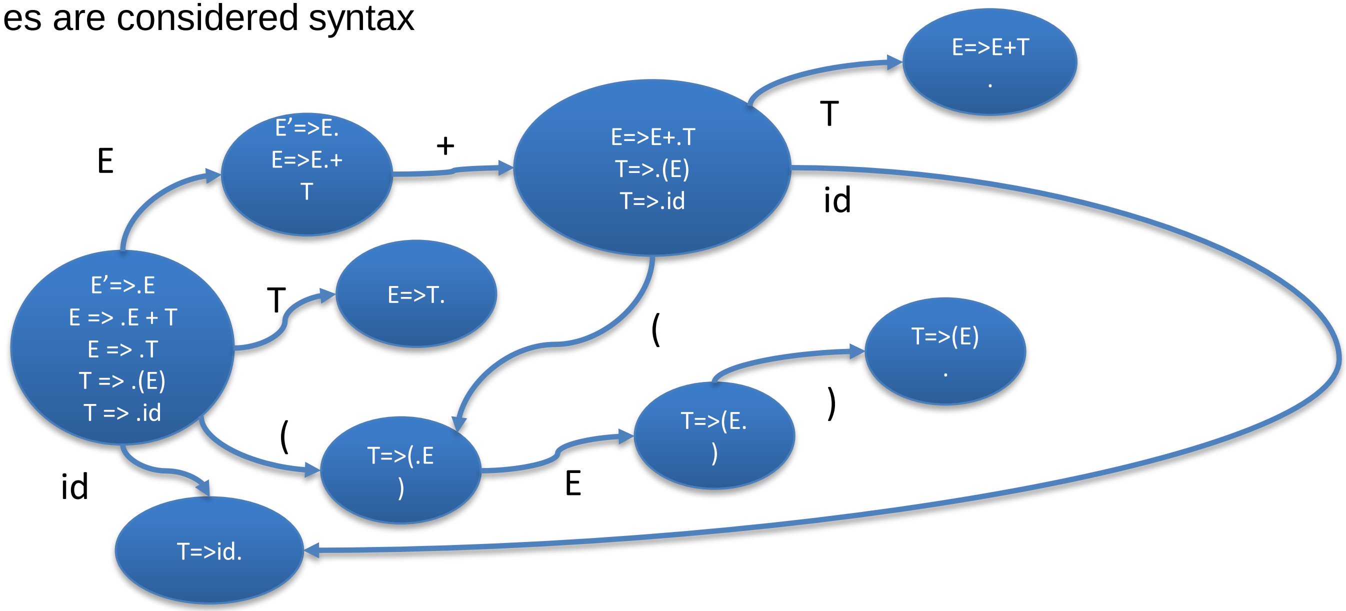
E’ = > .E
E’ = > E.

E => .E + T
E => E. + T
E => E +. T
E => E + T.

E => .T
E => T.

T => .(E)
T => (.E)
T => (E.)
T => (E).

T => .id
T => id.

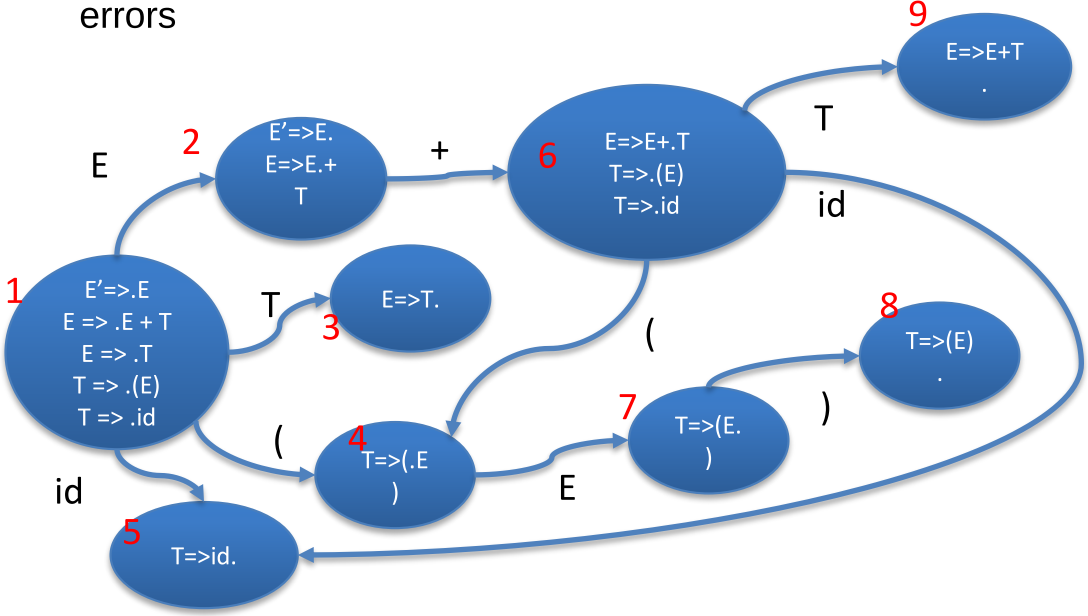


Build GOTO and ACTION Tables

Algorithm. Construct SLR table.

1. Construct canonical collection of states $I=\{I_1,\ldots,I_n\}$
2. Create Initial State from closure of E' => .E
3. State i corresponds to I_i
    1. If A => α . a β is in I_i and goto(I_i,a) = I_j then set action[i,a] to shift j
    2. If A => α . is in I_i then set action[i,a] to reduce A => α for all a in FOLLOW(A). A cannot be start.
    3. If E' => E . is in I_i , then action[i,$] = accept
4. For non terminals A, If goto(I_i,A) = I_j , then set goto[i,A]. *//Set goto table.*
5. All empty spots in tables are considered syntax errors

Augmented Grammar
1. E' = > E
2. E => E + T
3. E => T
4. T => (E)
5. T => id



Stat e	id	+	(	)	$	E	T
			Action			Goto	
1	s5		s4			2	3
2		s6			acc		
3					r3		
4						7	
5					r5		
6	s5		s4				9
7				s8			
8					r4		
9		r2			r2		

LL(k) vs. LR(k)

LR(k) must be able to recognize the occurrence of RHS production having seen all that has been derived up to that point +k lookahead(s)

LL(k) must be able to select the correct production having seen up to *k* symbols

A main difference is the use of the stack and creation of the states

LR(k) is less constraining that LL(k) and thus LR(k) supersets LL(k)

The weakness of LL(k) parsing techniques is that the parser must be able to predict which rhs to use, *having seen only the k next symbols of the input*. For LL(1), this means just one symbol must tell all. In contrast, for an *LR(k) grammar is able to postpone the decision until it has seen tokens corresponding to the entire righthand side* (plus k more tokens of lookahead).

Grammars

Notes about programming Languages.

More Expressive Languages belong to more complex grammar classes

Python and LISP

Have LR(1) implementations

C++ is technically an ambiguous** language

`x * y;`

**must disambiguate

Practical Concerns

Example: Ambiguity

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?
- **Error-handling**

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
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

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 ·

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

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

	int	+	*	(	)	$	E
1	s10			s7			
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
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							
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							
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
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				
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							
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	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
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							
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						
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			
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		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**
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							
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					
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	s10			s7			s6
5		s3	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

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 ·

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

	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

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 ·

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

	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
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	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				
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				
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		
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
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		s3					
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		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
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				
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		
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							

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							

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					

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**
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		

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	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
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	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**
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		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**

	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	

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** (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		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

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	

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**
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	