بسم الله الرحمن الرحيم

Parsing:

Bottom-Up Parsing

LR(0), SLR(1) and LR(1)

Compilers - 002 - S011

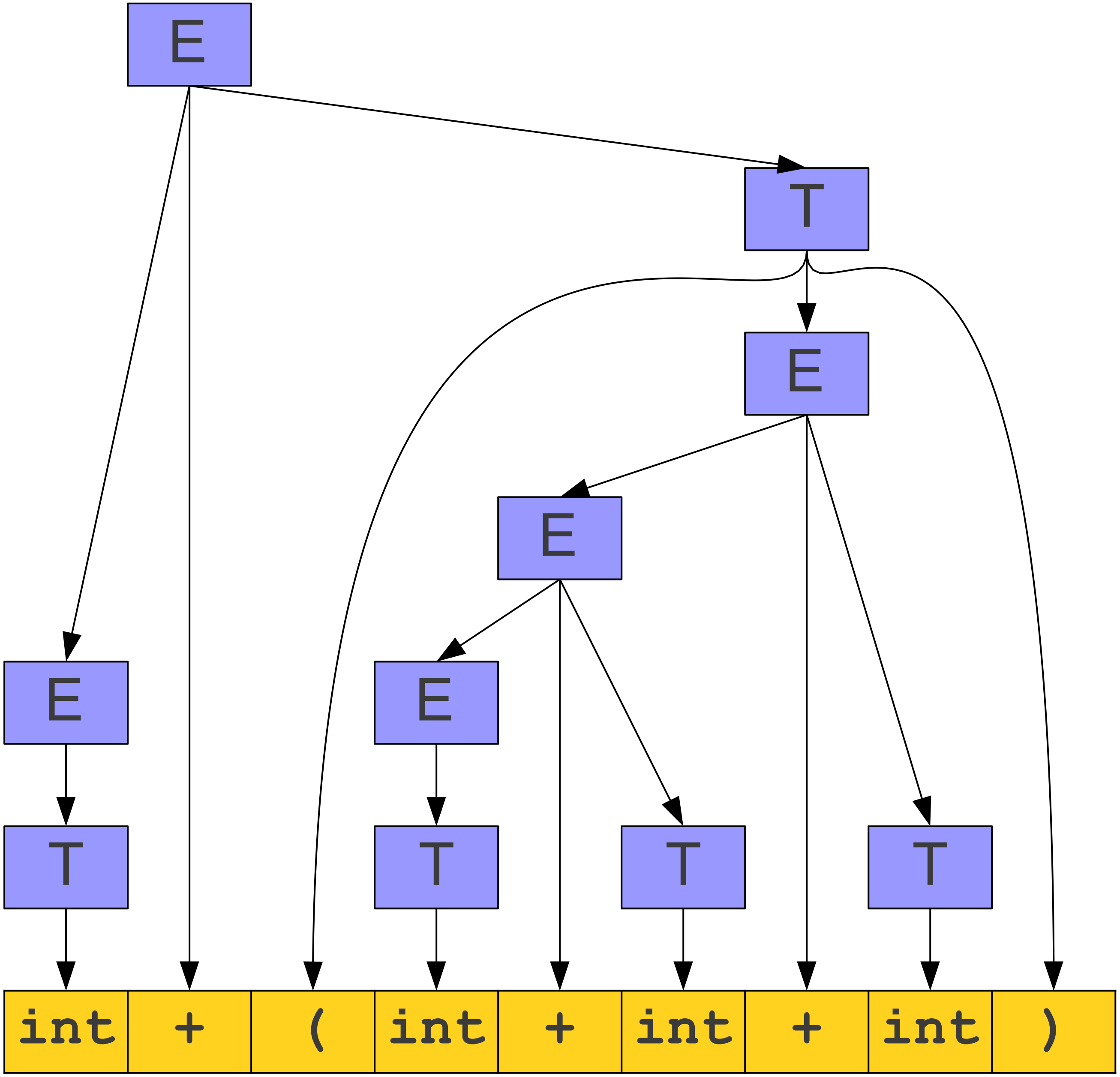
One View of a Bottom-Up Parse

E → **T**

E → **E + T**

T → **int**

T → **(E)**



A Second View of a Bottom-Up Parse

E → T

E → E + T

T → int

T → (E)

int + (int + int + int)

⇒ T + (int + int + int)

⇒ E + (int + int + int)

⇒ E + (T + int + int)

⇒ E + (E + int + int)

⇒ E + (E + T + int)

⇒ E + (E + int)

⇒ E + (E + T)

⇒ E + (E)

⇒ E + T

⇒ E

A Second View of a Bottom-Up Parse

E → T

E → E + T

T → int

T → (E)

```
  int + (int + int + int)
⇒ T + (int + int + int)
⇒ E + (int + int + int)
⇒ E + (T + int + int)
⇒ E + (E + int + int)
⇒ E + (E + T + int)
⇒ E + (E + int)
⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E
```

A left-to-right, bottom-up parse is a rightmost derivation traced in reverse.

Question One:

Where are handles?

Recognizing Left-Hand Sides

- Idea: At each point, track
  - Which production we are in, and
  - Where we are in that production.
- At each point, we can do one of two things:
  - **Match** the next symbol
  - (Just for now) **Guess** which production used.
  - (More precisely the production chooses non-deterministically)

Recognizing Left-Hand Sides

S → E

E → F

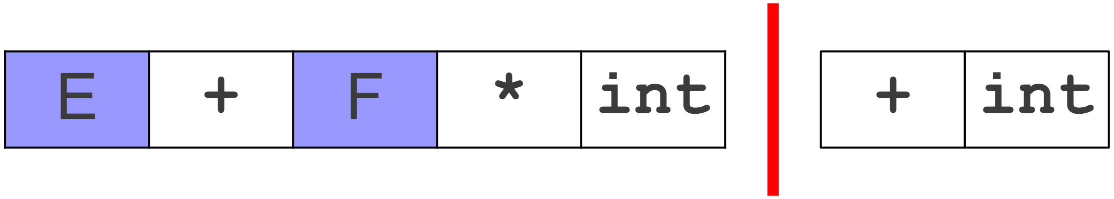
E → E + F

F → F * T

F → T

T → int

T → (E)

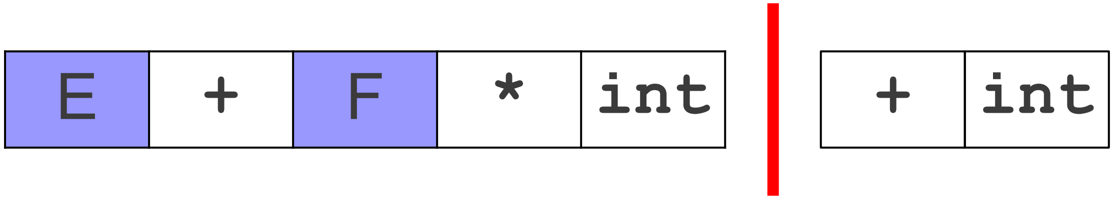


E + F * int | + int

Recognizing Left-Hand Sides

S → · E

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



Recognizing Left-Hand Sides

S → · E

S → E

E → F

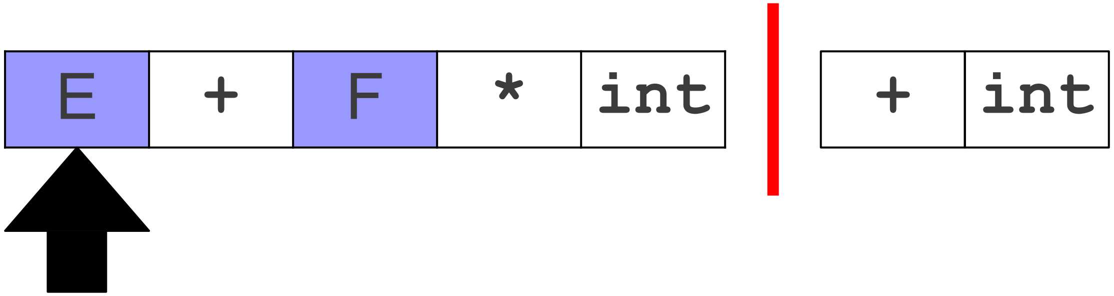
E → E + F

F → F * T

F → T

T → int

T → (E)



E + F * int | + int

Recognizing Left-Hand Sides

S → E

E → F

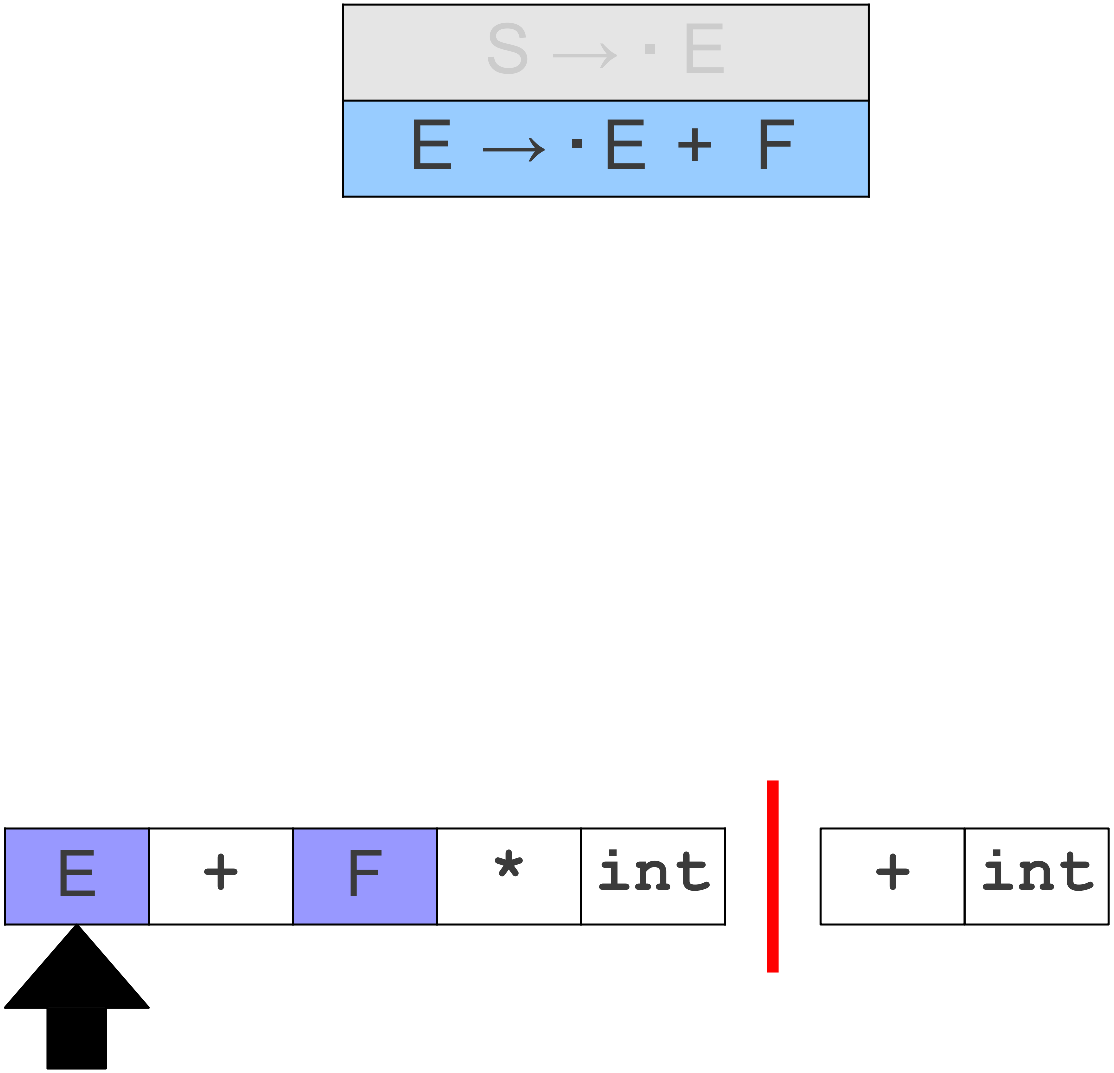
E → E + F

F → F * T

F → T

T → int

T → (E)



Recognizing Left-Hand Sides

S → E

E → F

E → E + F

F → F * T

F → T

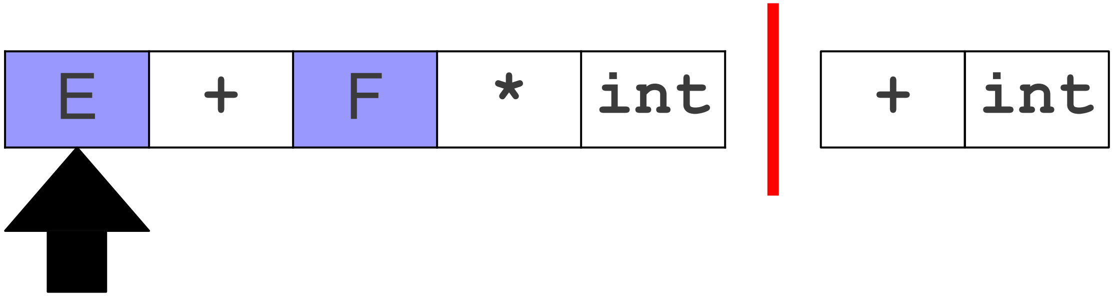
T → int

T → (E)

S → · E

E → · E + F

E → · E + F



E + F * int | + int

Recognizing Left-Hand Sides

S → E

E → F

E → E + F

F → F * T

F → T

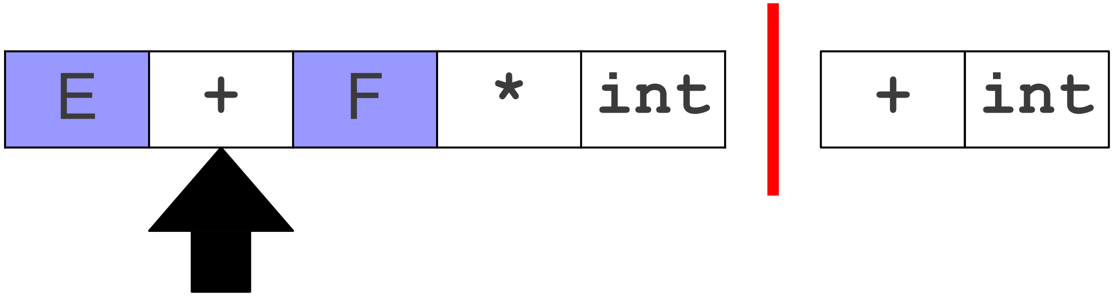
T → int

T → (E)

S → · E

E → · E + F

E → E · + F



E + F * int | + int

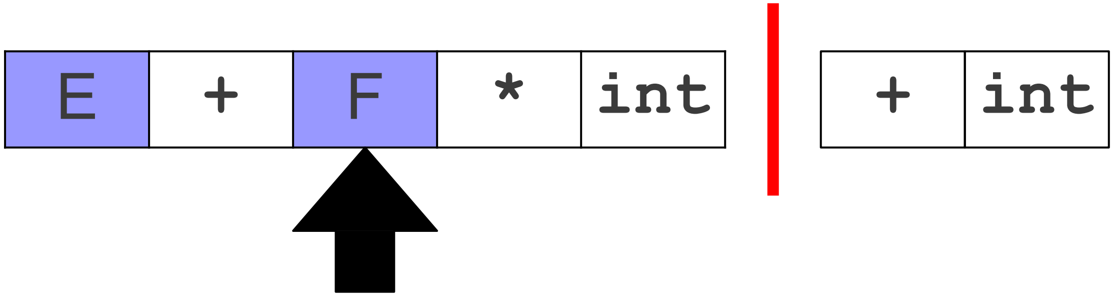
Recognizing Left-Hand Sides

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

S → · E

E → · E + F

E → E + · F



Recognizing Left-Hand Sides

S → E

E → F

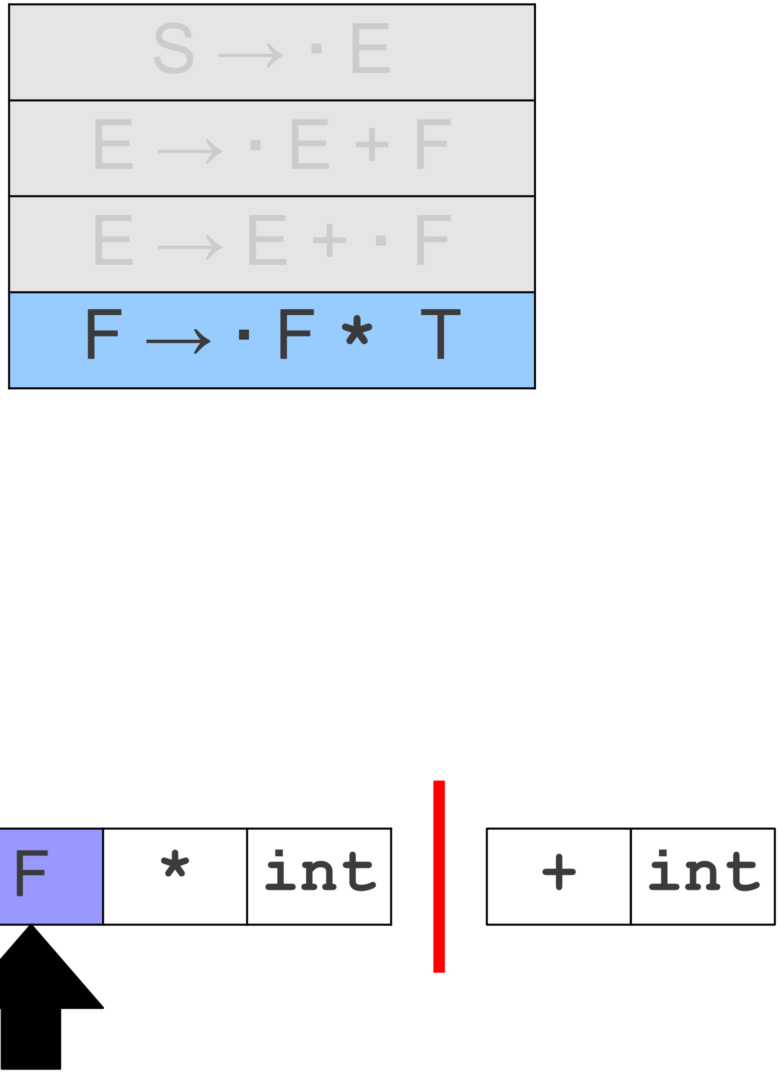
E → E + F

F → F * T

F → T

T → int

T → (E)



S → · E

E → · E + F

E → E + · F

F → · F * T

E	+	F	*	int		+	int

Recognizing Left-Hand Sides

S → E

E → F

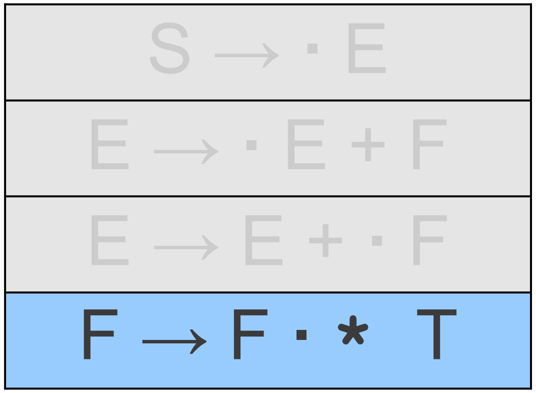
E → E + F

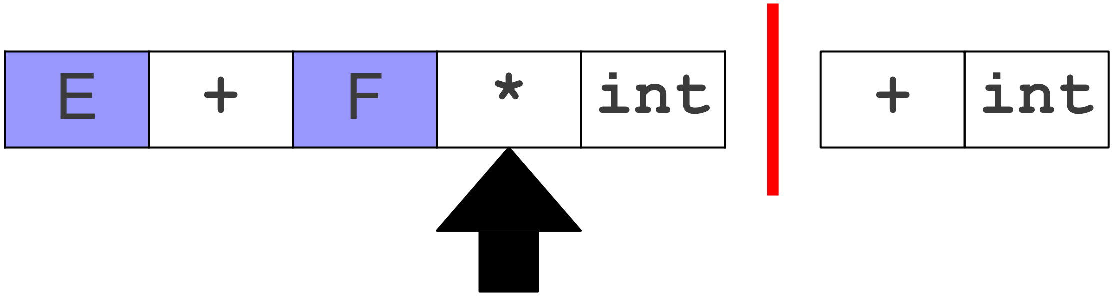
F → F * T

F → T

T → int

T → (E)





Recognizing Left-Hand Sides

S → E

E → F

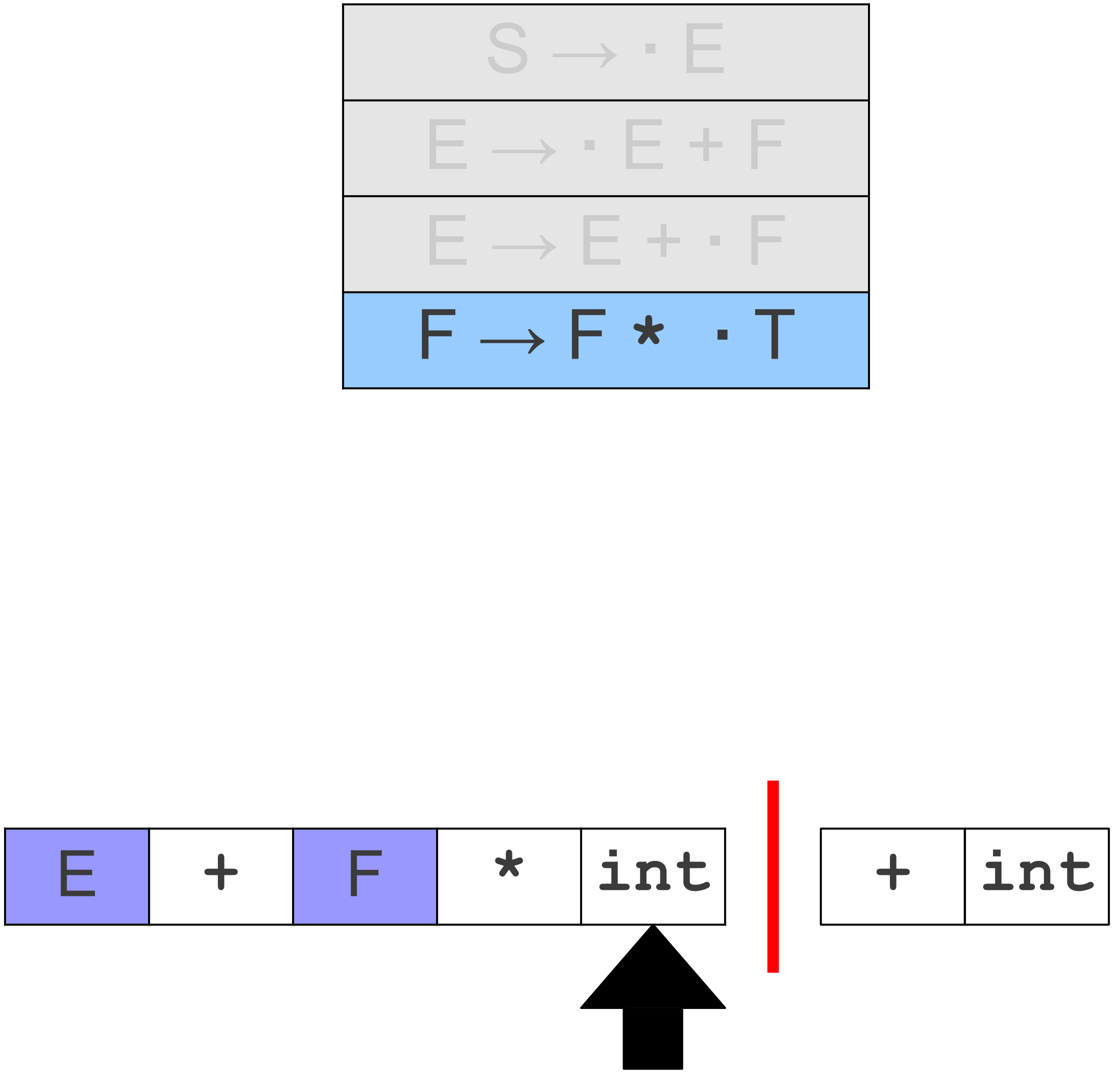
E → E + F

F → F * T

F → T

T → int

T → (E)



S → · E

E → · E + F

E → E + · F

F → F * · T

E + F * int | + int

Recognizing Left-Hand Sides

S → E

E → F

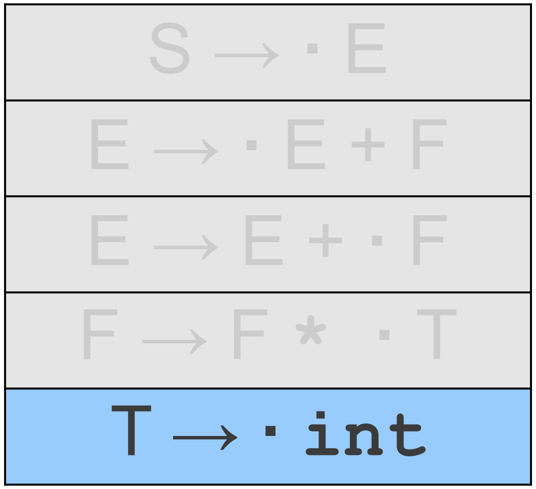
E → E + F

F → F * T

F → T

T → int

T → (E)



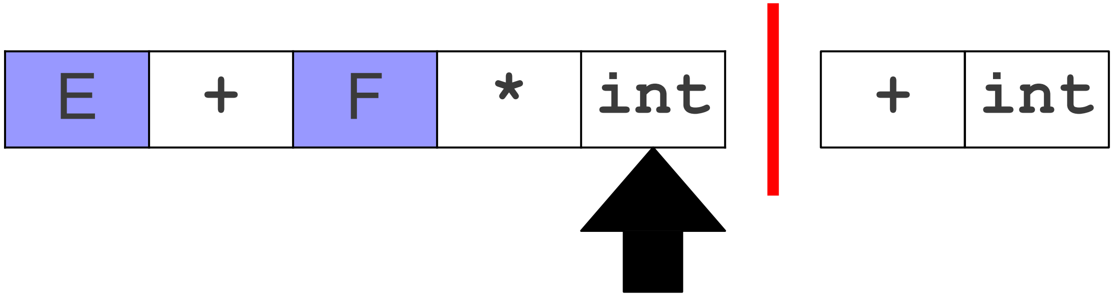
S → · E

E → · E + F

E → E + · F

F → F * · T

T → · int



Recognizing Left-Hand Sides

S → E

E → F

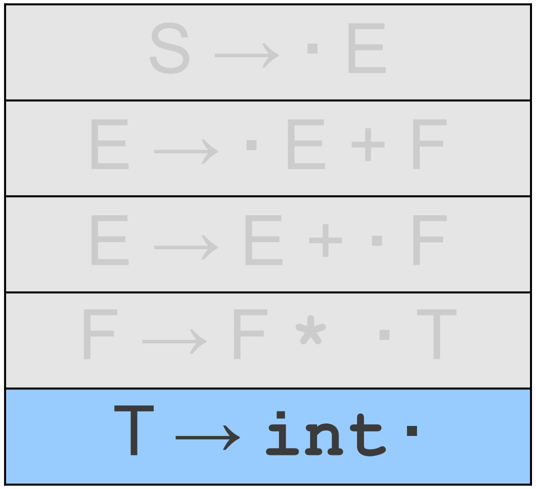
E → E + F

F → F * T

F → T

T → int

T → (E)



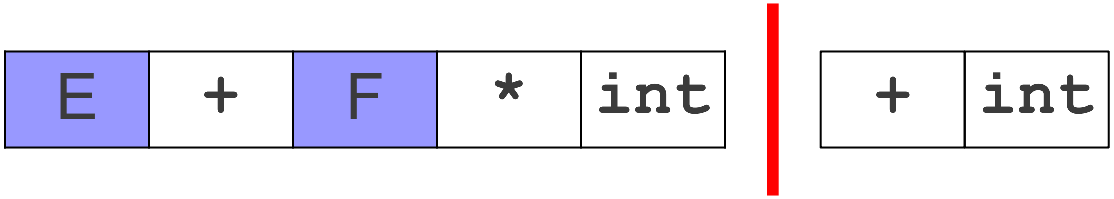
S → · E

E → · E + F

E → E + · F

F → F * · T

T → int ·



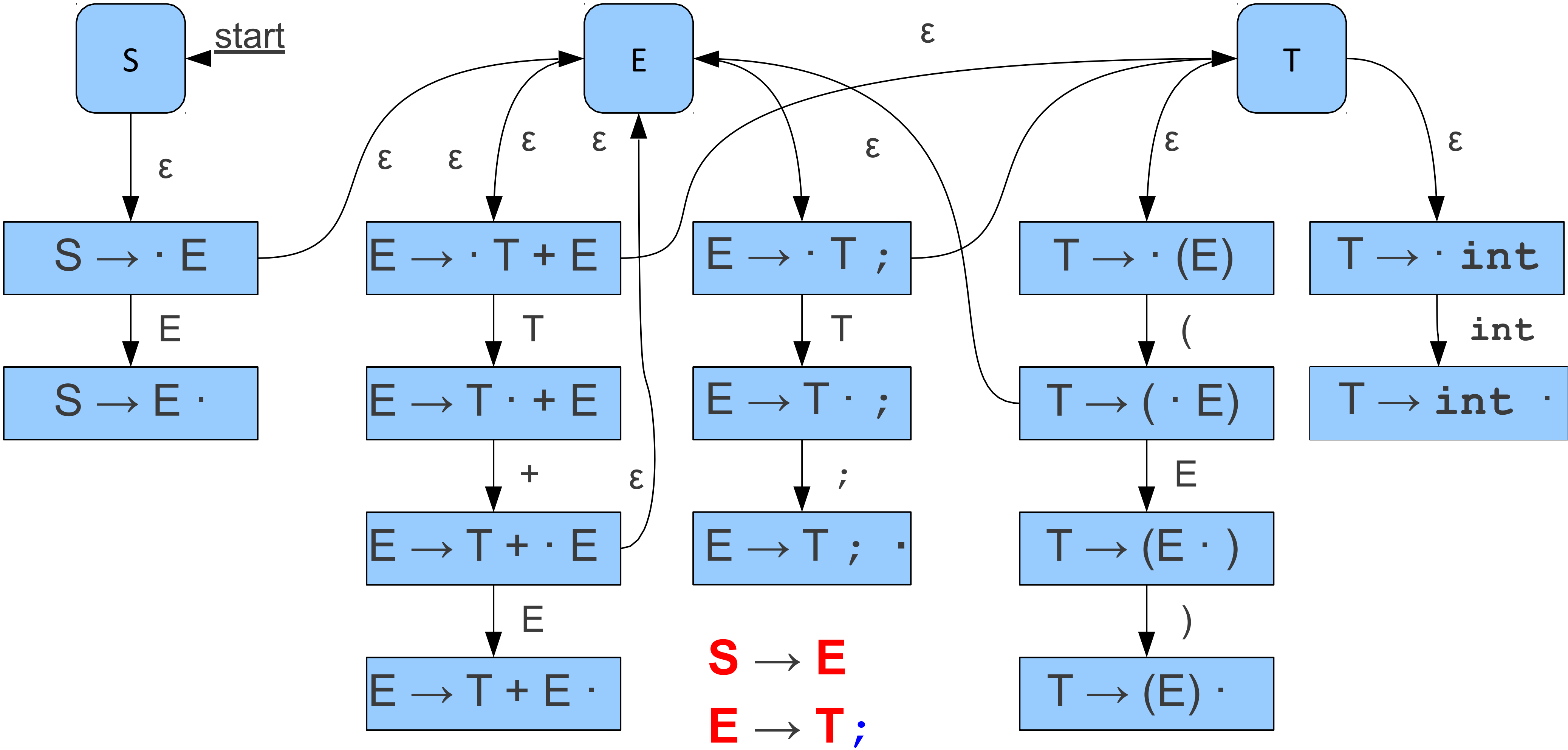
E + F * int | + int

An Important Result

- At any point in time, we only need to track where we are in **one production**.

- We can use a finite automaton as our recognizer.

An Automaton for Left Areas



S → E

E → T;

E → T + E

T → int

T → (E)

Constructing the Automaton

- Create a state for each nonterminal. For
- each production **A** → *γ*:
  - Construct states **A** → *α* · *ω* for each possible way of splitting *γ* into two substrings *α* and *ω*.
  - Add transitions on ***x*** between **A** → *α* · ***x****ω* and **A** → *α****x*** · *ω*.
- For each state **A** → *α* · **B***ω* for nonterminal **B**, add an ε-transition from **A** → *α* · **B***ω* to **B**.

Why This Matters

- Our initial goal was to find handles.
- When running this automaton, if we ever end up in a state with a rule of the form

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

- Then we might be looking at a handle.

Constructing the Automaton II

- Begin in a state containing **S** → · **A**, where **S** is the augmented start symbol.

Constructing the Automaton II

- Begin in a state containing **S** → · **A**, where **S** is the augmented start symbol.
- Compute the **closure** of the state:
  - If **A** → *α* · **B***ω* is in the state, add **B** → · *γ* to the state for each production **B** → *γ*.
  - Yet another fixed-point iteration!

Constructing the Automaton II

- Begin in a state containing **S** → · **A**, where **S** is the augmented start symbol.
- Compute the **closure** of the state:
  - If **A** → *α* · **B***ω* is in the state, add **B** → · *γ* to the state for each production **B** → *γ*.
  - Yet another fixed-point iteration!
- Repeat until no new states are added:
  - If a state contains a production **A** → *α* · ***x****ω* for symbol ***x***, add a transition on ***x*** from that state to the state containing the closure of **A** → *α**x*** · *ω*

Constructing the Automaton II

- Begin in a state containing **S** → · **A**, where **S** is the augmented start symbol.
- Compute the **closure** of the state:
  - If **A** → *α* · **B***ω* is in the state, add **B** → · *γ* to the state for each production **B** → *γ*.
  - Yet another fixed-point iteration!
- Repeat until no new states are added:
  - If a state contains a production **A** → *α* · ***x****ω* for symbol ***x***, add a transition on ***x*** from that state to the state containing the closure of **A** → *α****x*** · *ω*
- This is equivalent to a subset construction on the NFA.

A Deterministic Automaton

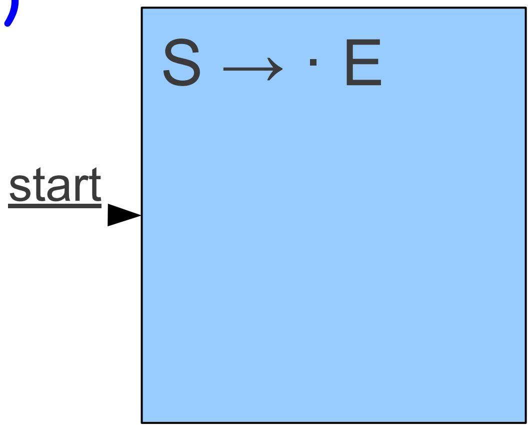
S → E

E → T;

E → T + E

T → int

T → (E)



start

S → · E

A Deterministic Automaton

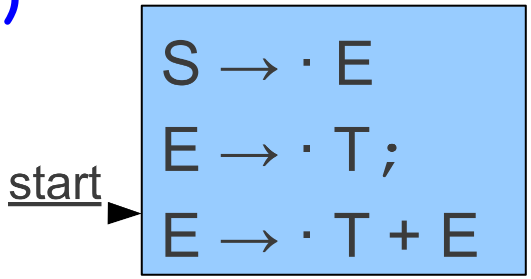
S → E

E → T;

E → T + E

T → int

T → (E)



A Deterministic Automaton

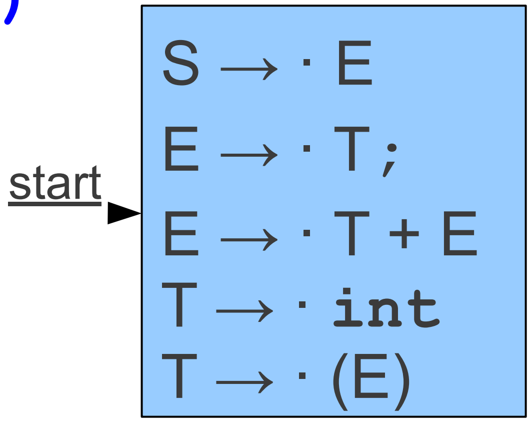
S → E

E → T;

E → T + E

T → **int**

T → (E)



start

S → · E

E → · T;

E → · T + E

T → · **int**

T → · (E)

A Deterministic Automaton

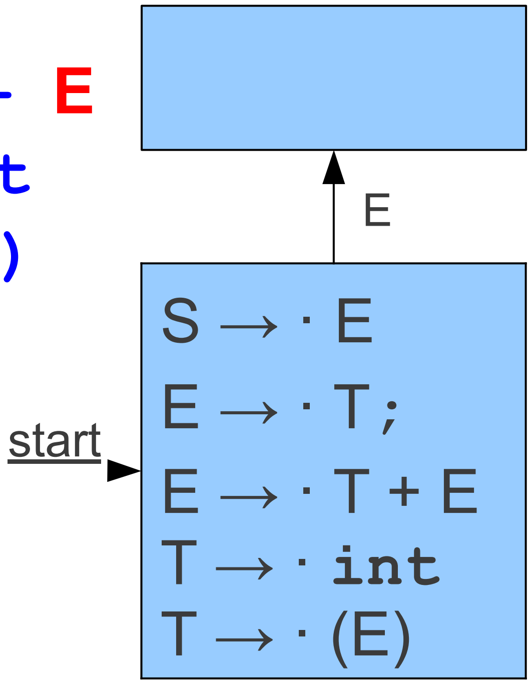
S → E

E → T;

E → T + E

T → int

T → (E)



S → · E

E → · T;

E → · T + E

T → · int

T → · (E)

A Deterministic Automaton

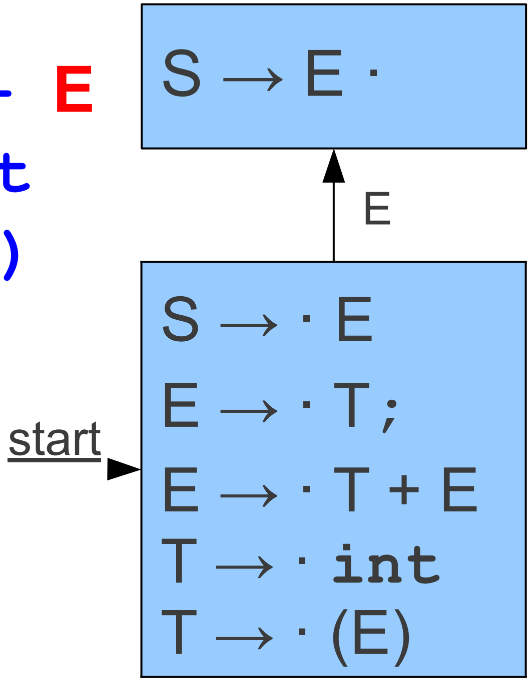
S → E

E → T`;`

E → T `+` E

T → `int`

T → `(`E`)`



S → E ·

E

start

S → · E

E → · T;

E → · T + E

T → · `int`

T → · (E)

A Deterministic Automaton

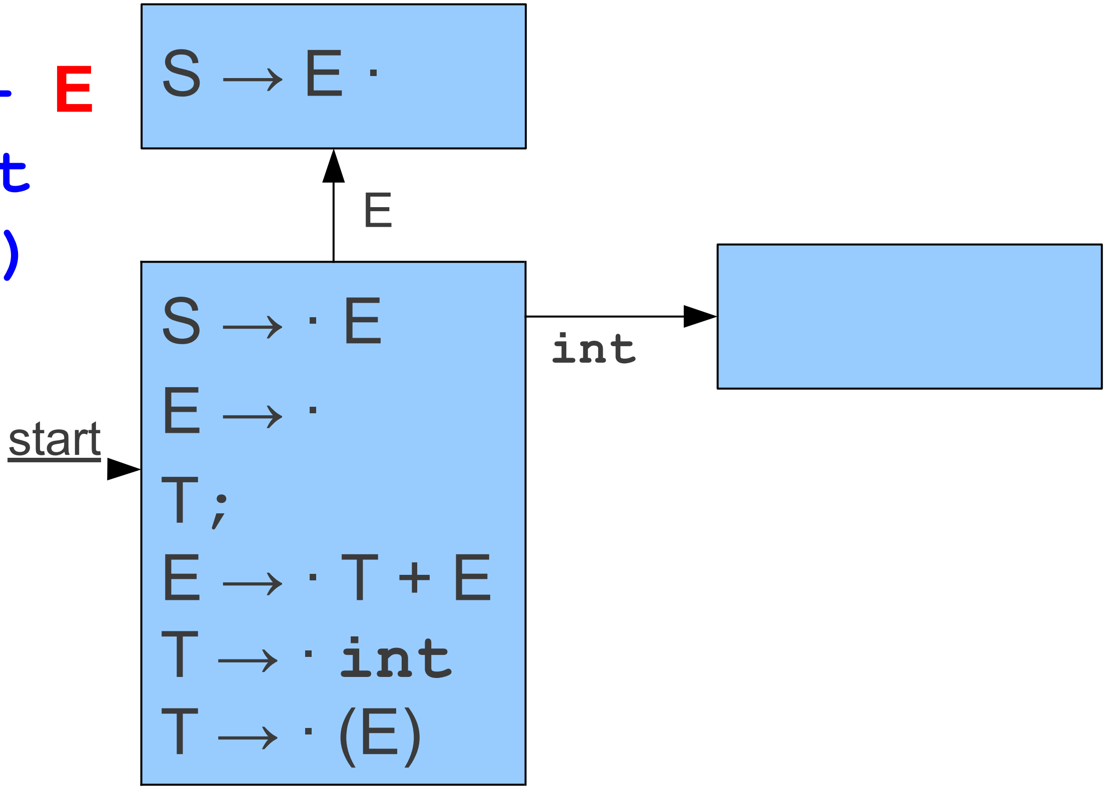
S → **E**

E → **T** `;`

E → **T** `+` **E**

T → `int`

T → `(`**E**`)`



A Deterministic Automaton

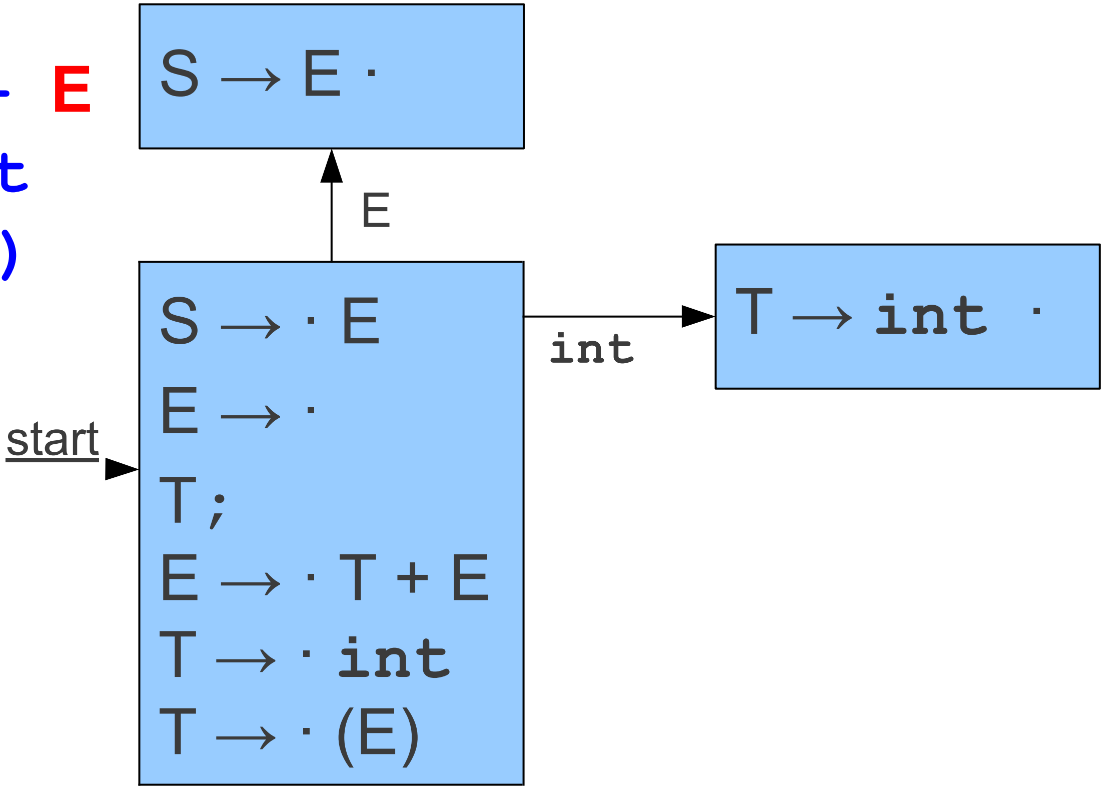
S → E

E → T;

E → T + E

T → int

T → (E)



A Deterministic Automaton

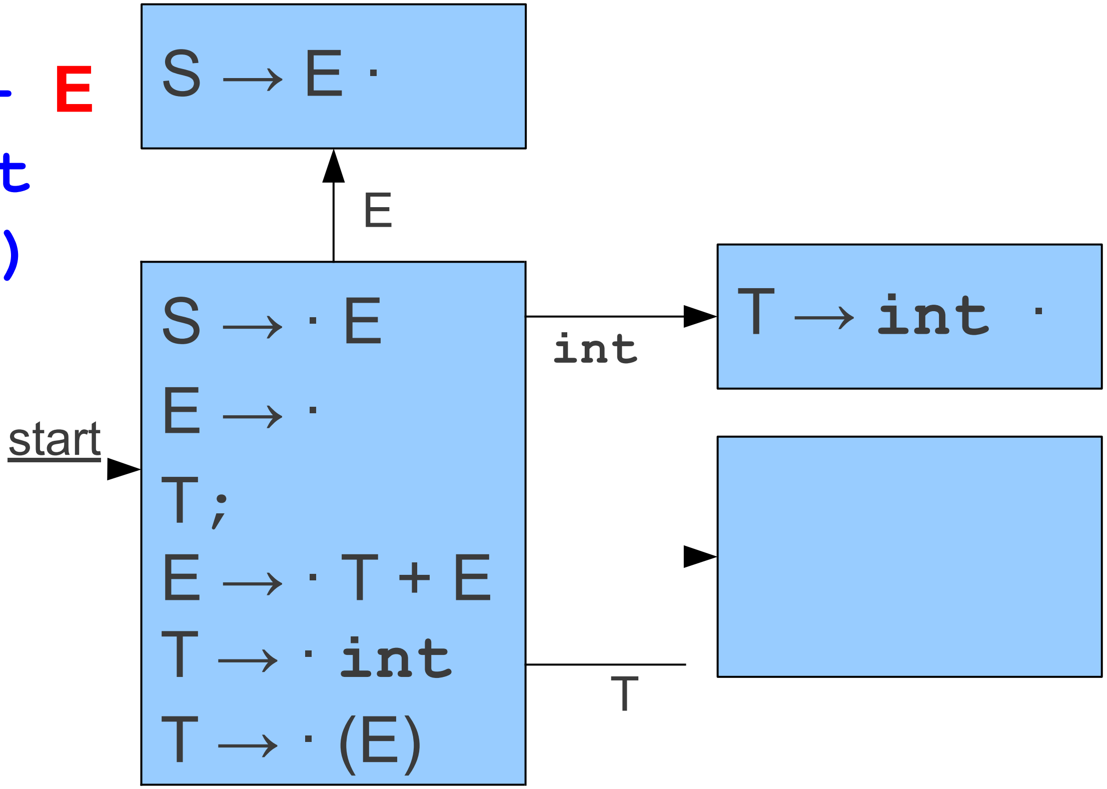
S → E

E → T;

E → T + E

T → int

T → (E)



A Deterministic Automaton

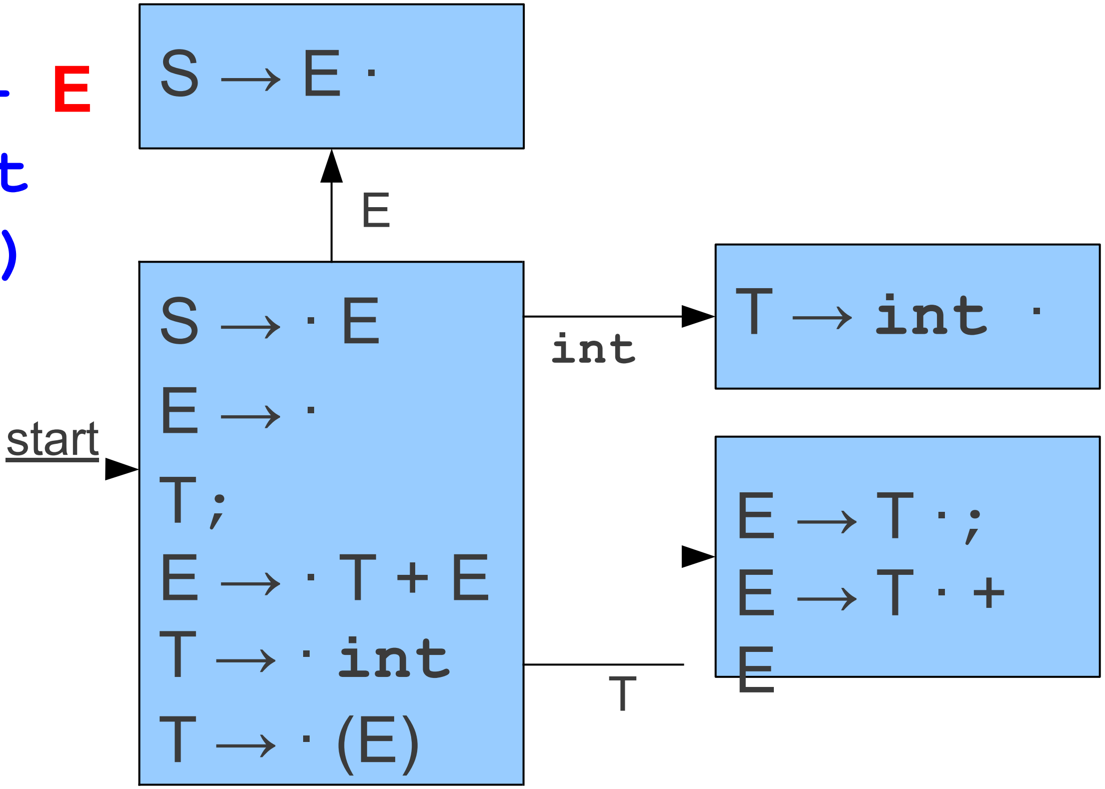
S → E

E → T;

E → T + E

T → int

T → (E)



S → E ·

E

start

S → · E

E → ·

T;

E → · T + E

T → · int

T → · (E)

int

T → int ·

T

E → T · ;

E → T · +

E

A Deterministic Automaton

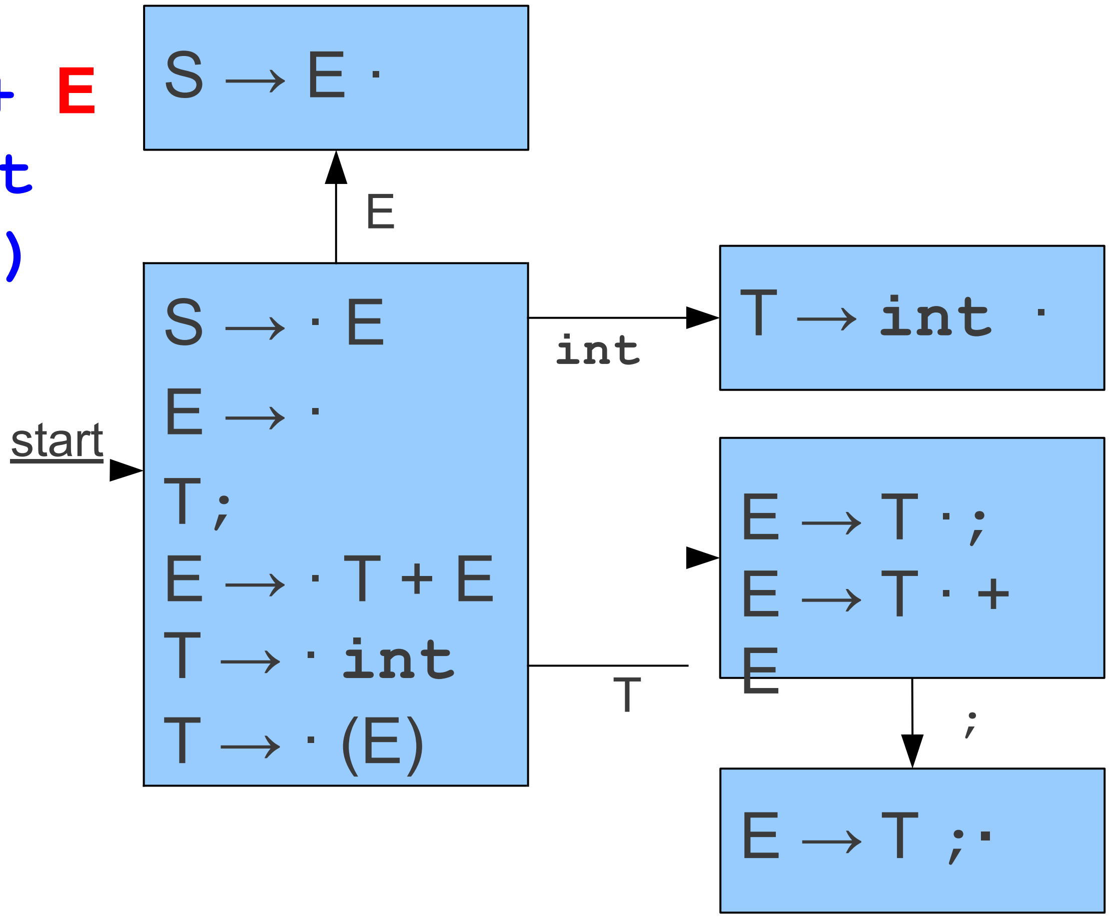
S → E

E → T ;

E → T + E

T → int

T → (E)



start

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

E

S → E ·

int

T → int ·

T

E → T · ;
E → T · +
E

;

E → T ; ·

A Deterministic Automaton

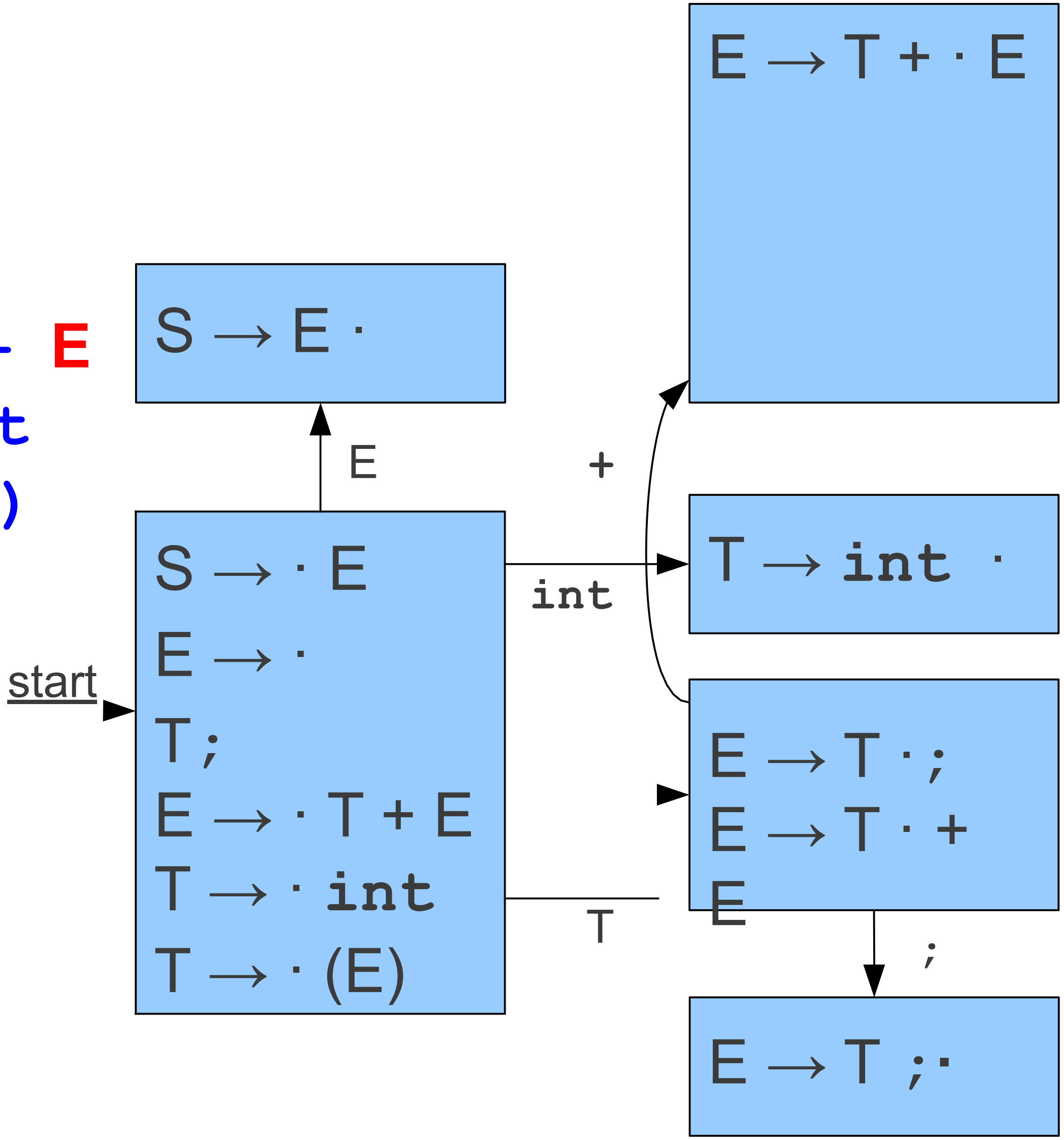
S → **E**

E → **T** ;

E → **T** + **E**

T → **int**

T → **(E)**



start

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

S → E ·

E → T + · E

T → **int** ·

E → T · ;
E → T · + E

E → T ; ·

A Deterministic Automaton

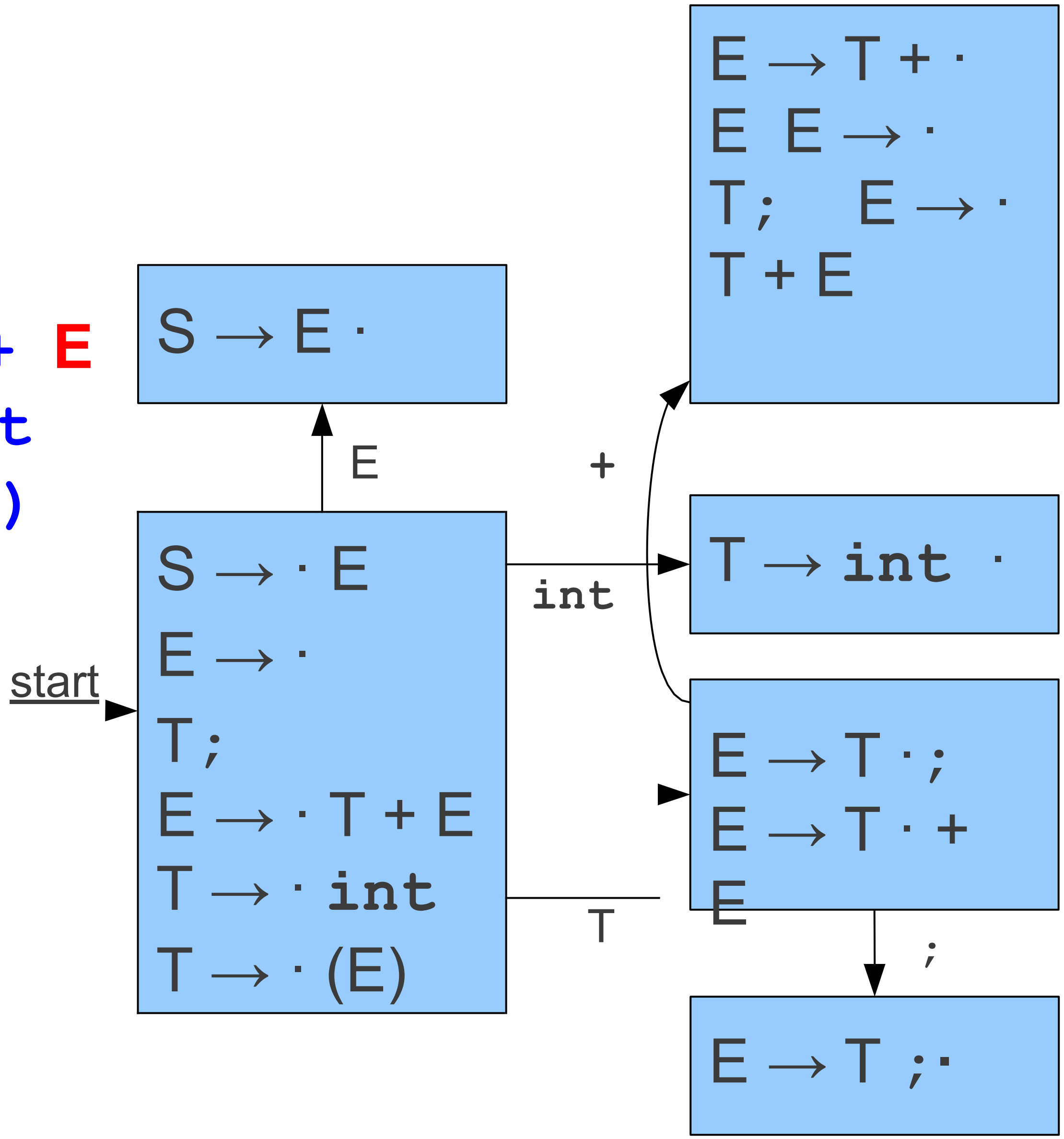
S → E

E → T;

E → T + E

T → int

T → (E)



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

S → E ·

E → T + ·
E E → ·
T; E → ·
T + E

T → int ·

E → T · ;
E → T · +
E

E → T ;·

A Deterministic Automaton

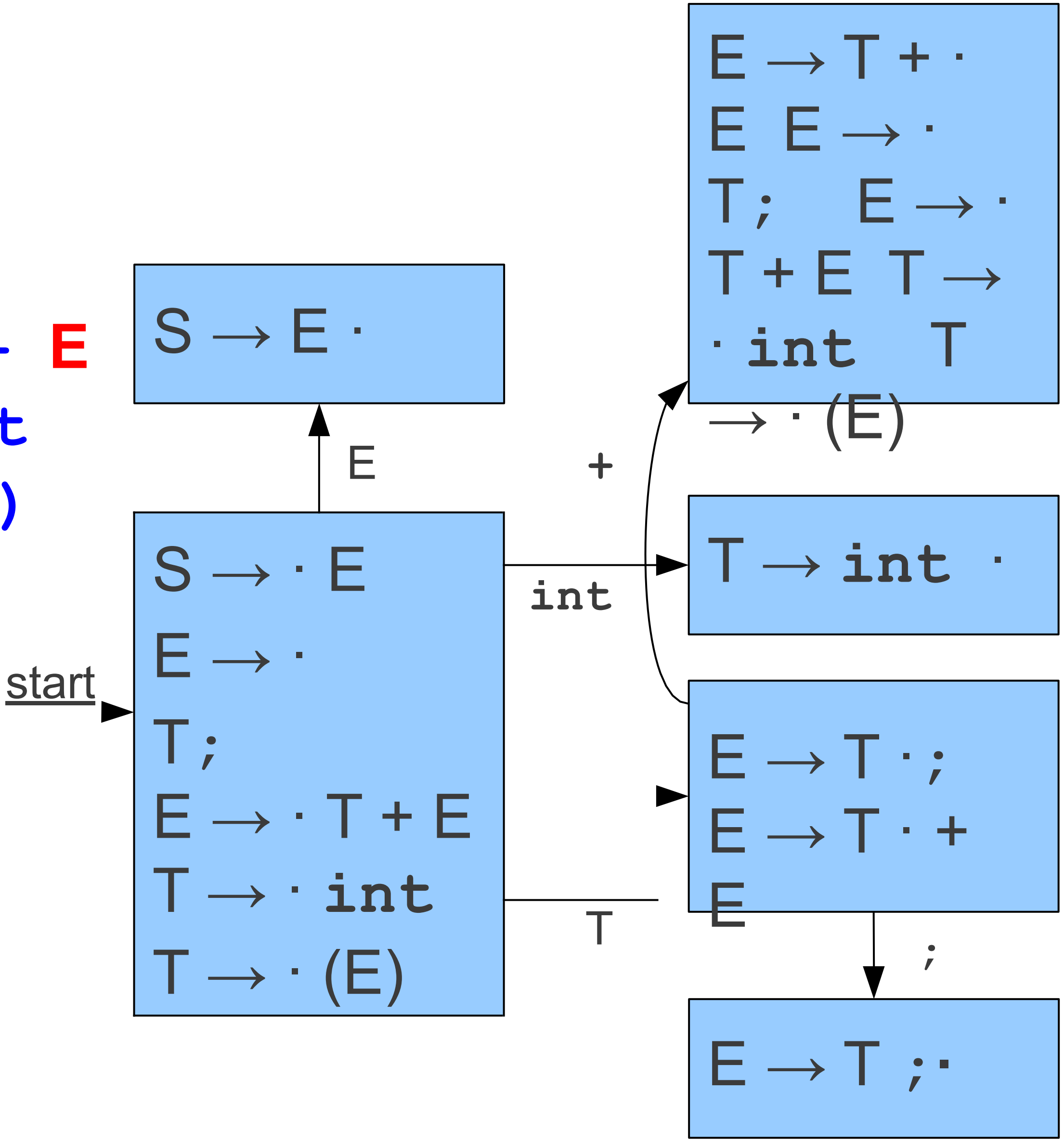
S → E

E → T;

E → T + E

T → int

T → (E)



S → E ·

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

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

T → int ·

E → T · ;
E → T · +
E

E → T ; ·

A Deterministic Automaton

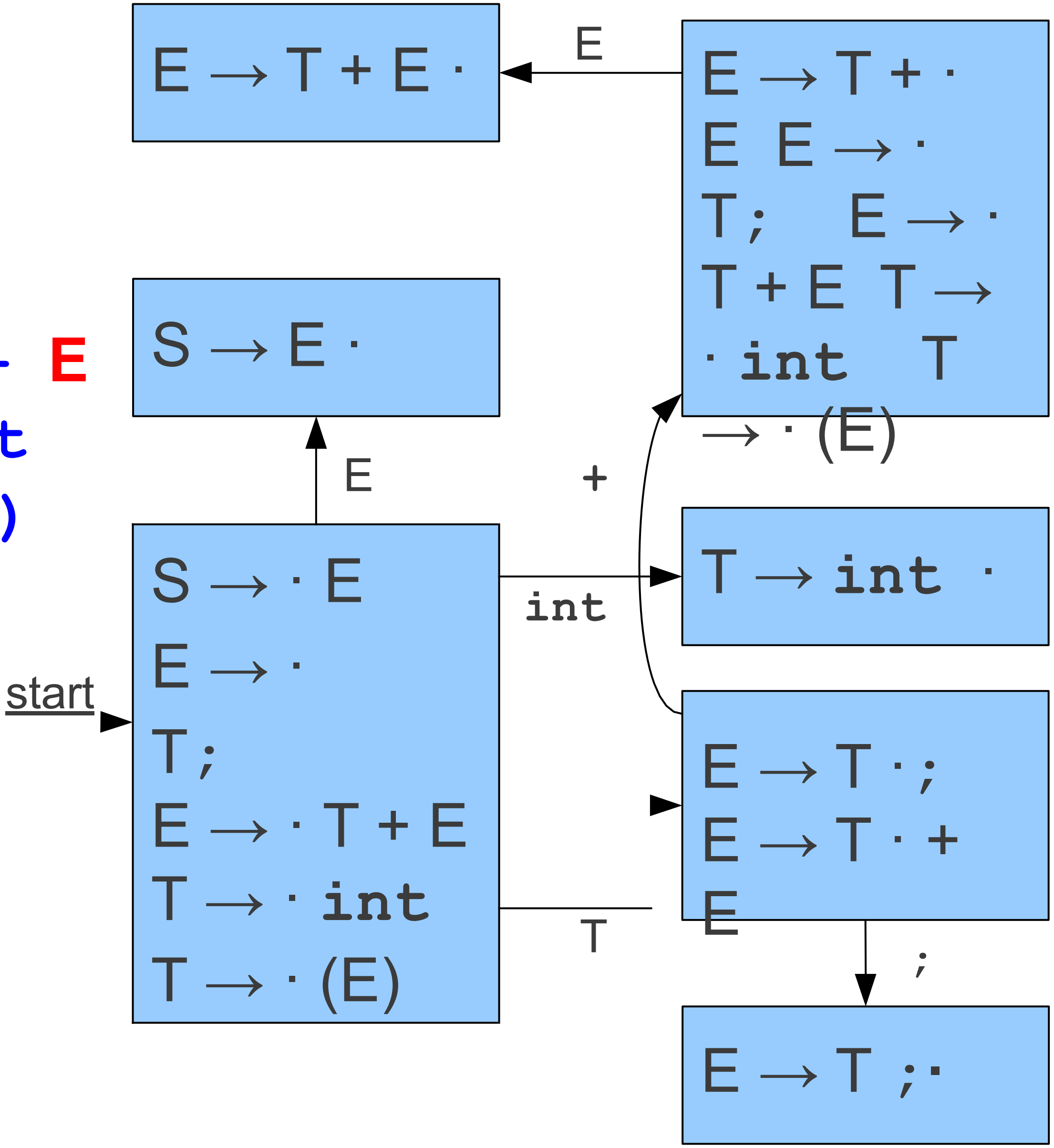
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

S → E ·

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

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

T → int ·

E → T · ;
E → T · +
E

E → T ; ·

A Deterministic Automaton

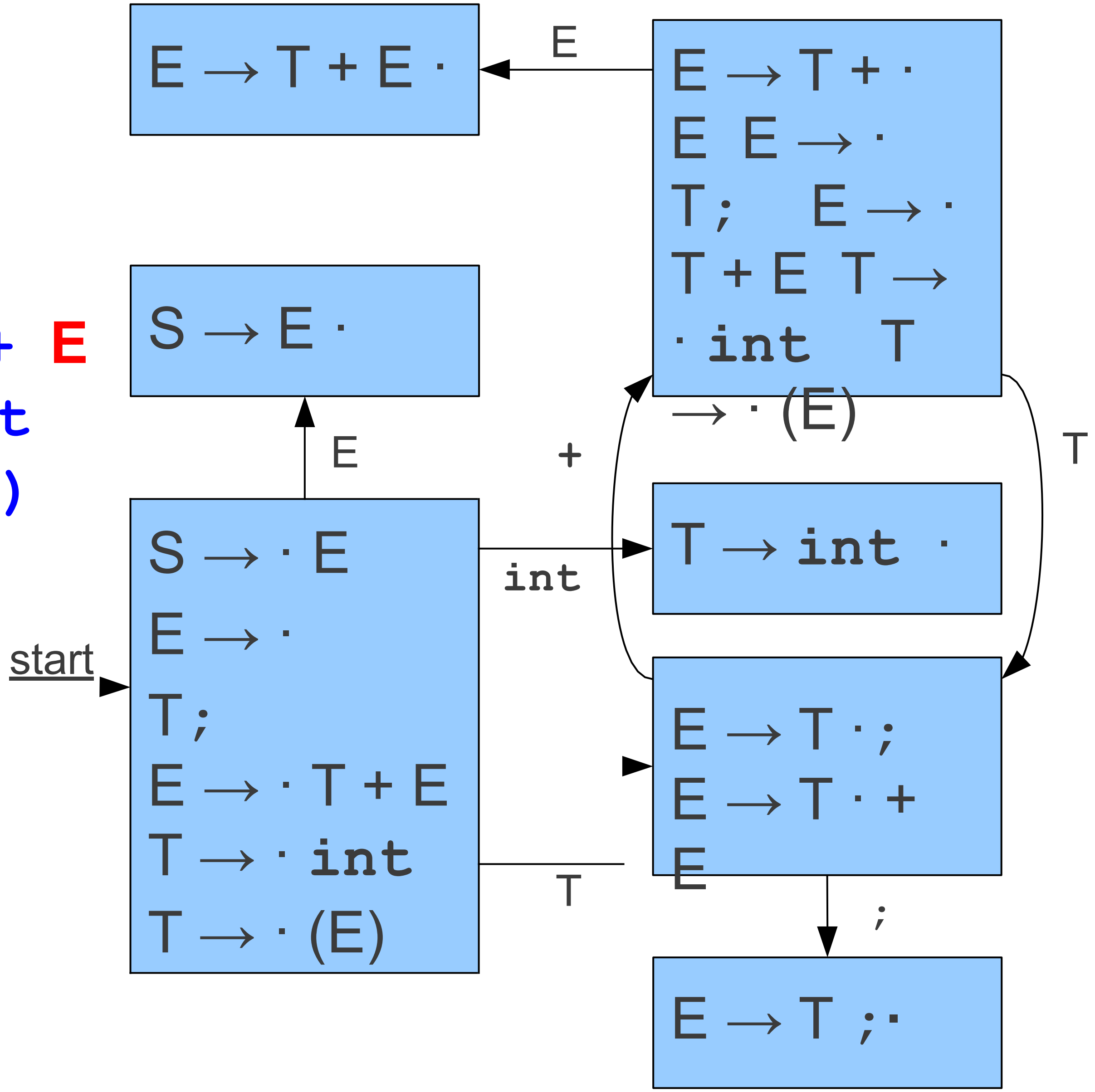
S → E

E → T;

E → T + E

T → int

T → (E)



A Deterministic Automaton

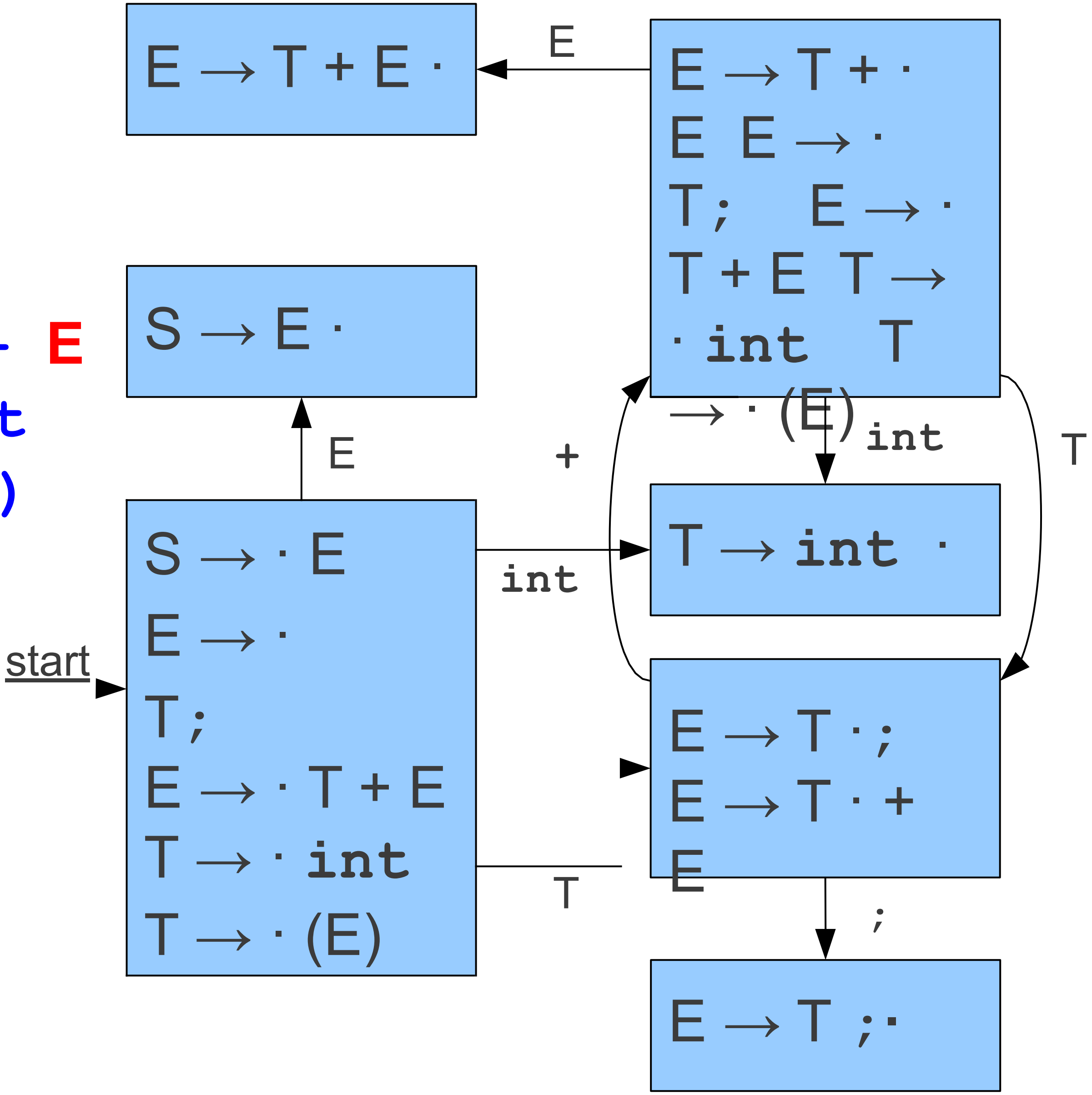
S → E

E → T;

E → T + E

T → int

T → (E)



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

S → E ·

E → T + E ·

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

T → int ·

E → T · ;
E → T · +
E

E → T ;·

A Deterministic Automaton

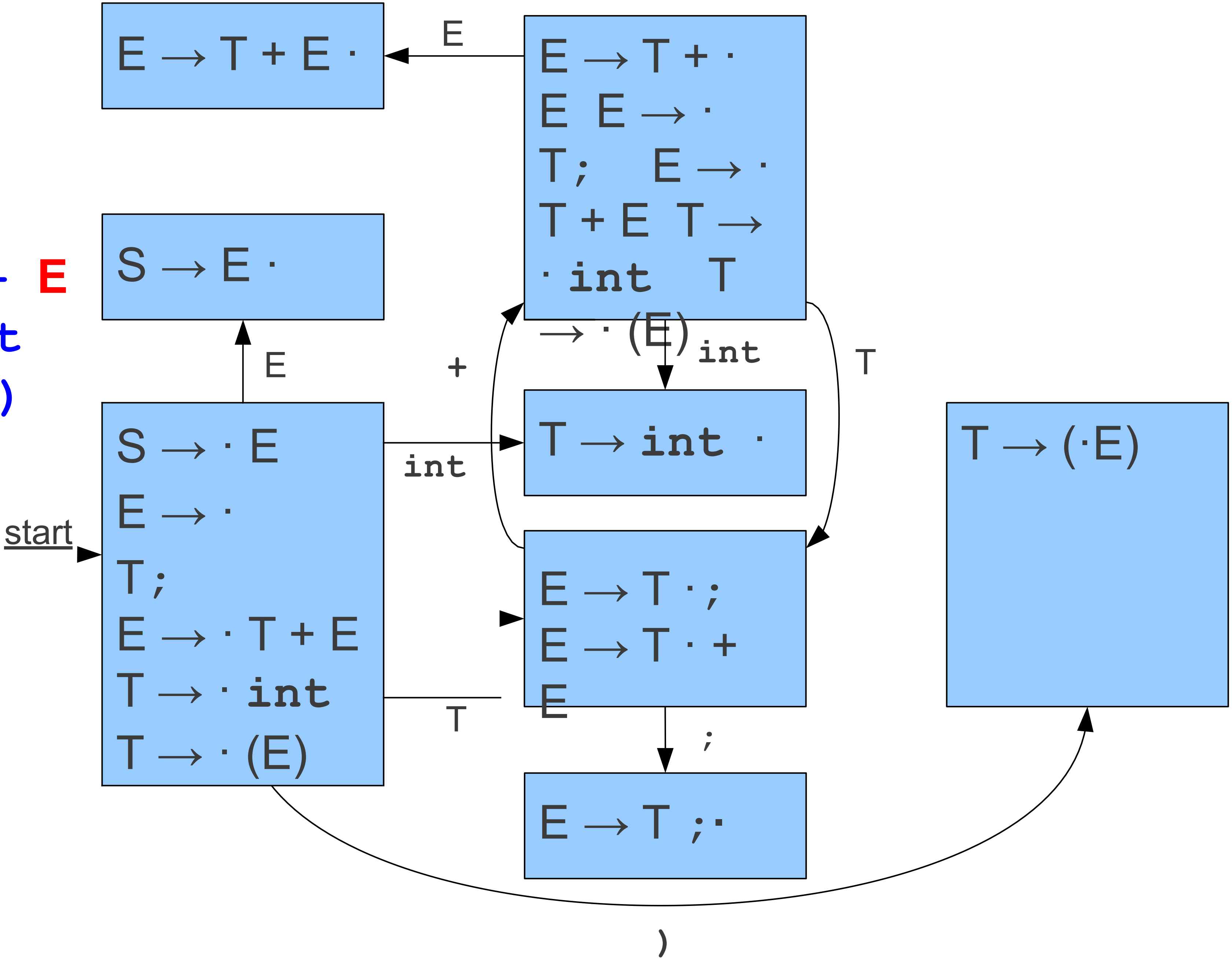
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

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

S → E ·

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

T → int ·

E → T · ;
E → T · +
E

E → T ;·

T → (·E)

A Deterministic Automaton

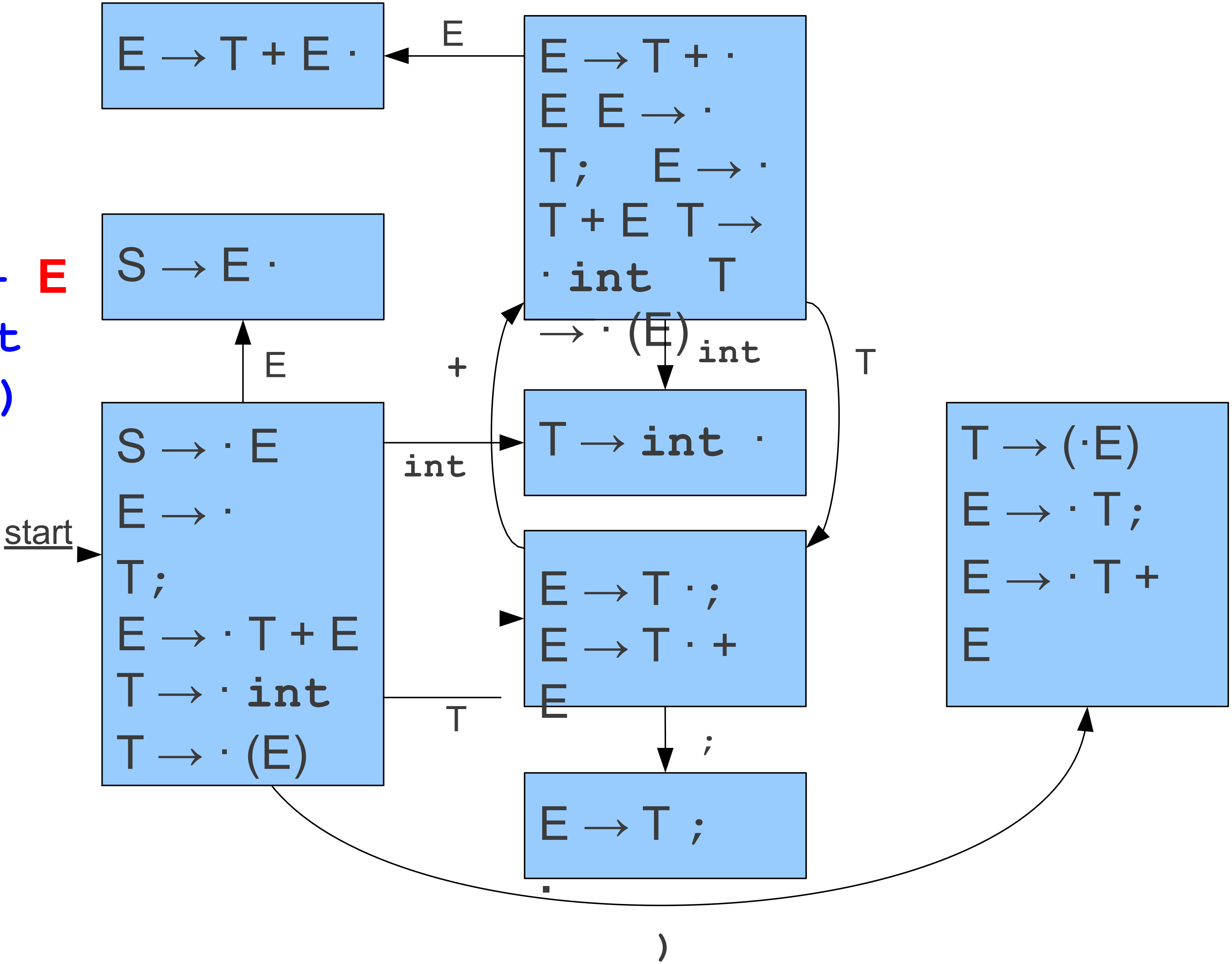
S → E

E → T;

E → T + E

T → int

T → (E)



A Deterministic Automaton

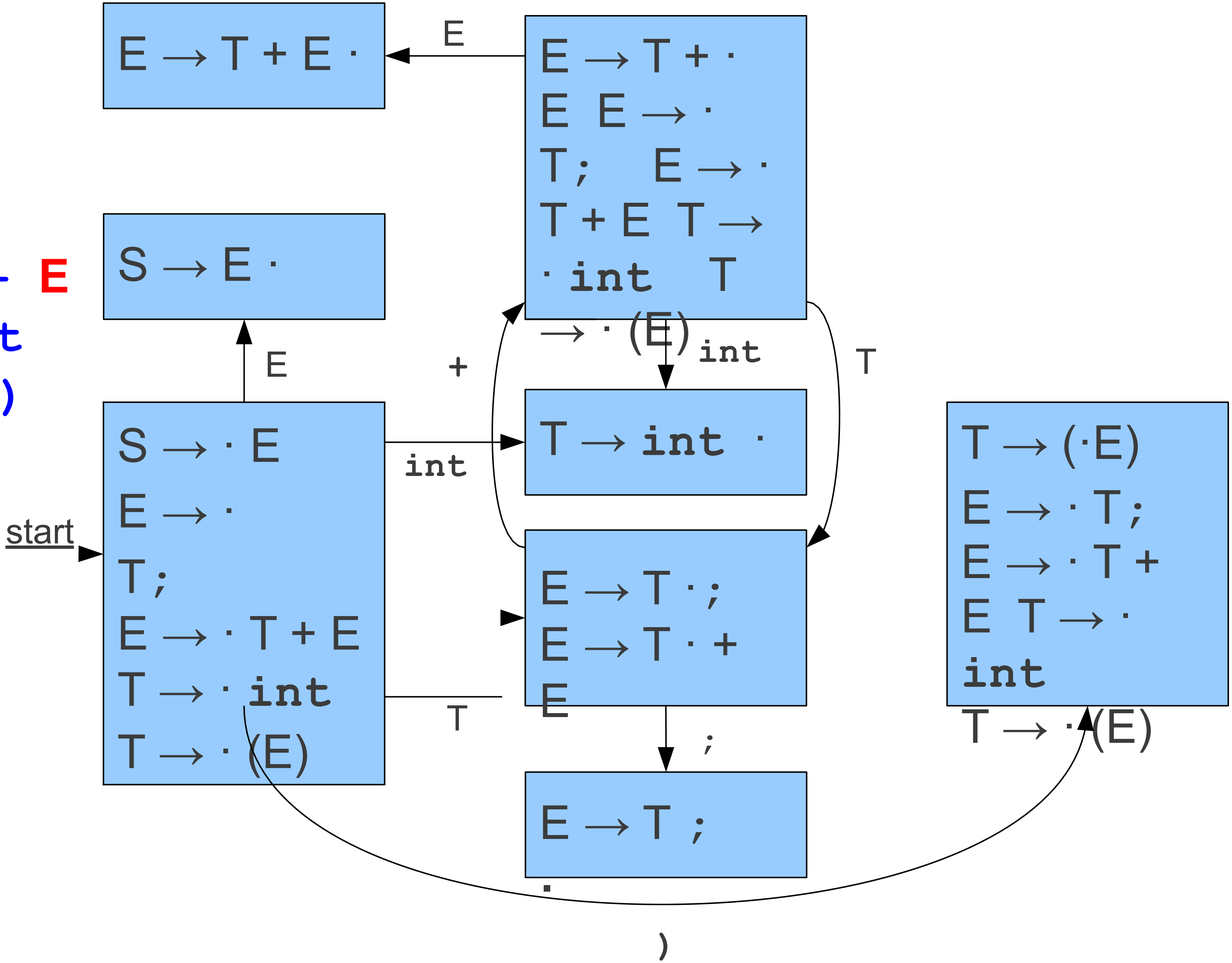
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

E → T + ·

E E → ·

T; E → ·

T + E T →

· int T

→ · (E)

S → E ·

S → · E

E → ·

T;

E → · T + E

T → · int

T → · (E)

T → int ·

E → T · ;

E → T · +

E

E → T ;

·

T → (·E)

E → · T;

E → · T +

E T → ·

int

T → · (E)

A Deterministic Automaton

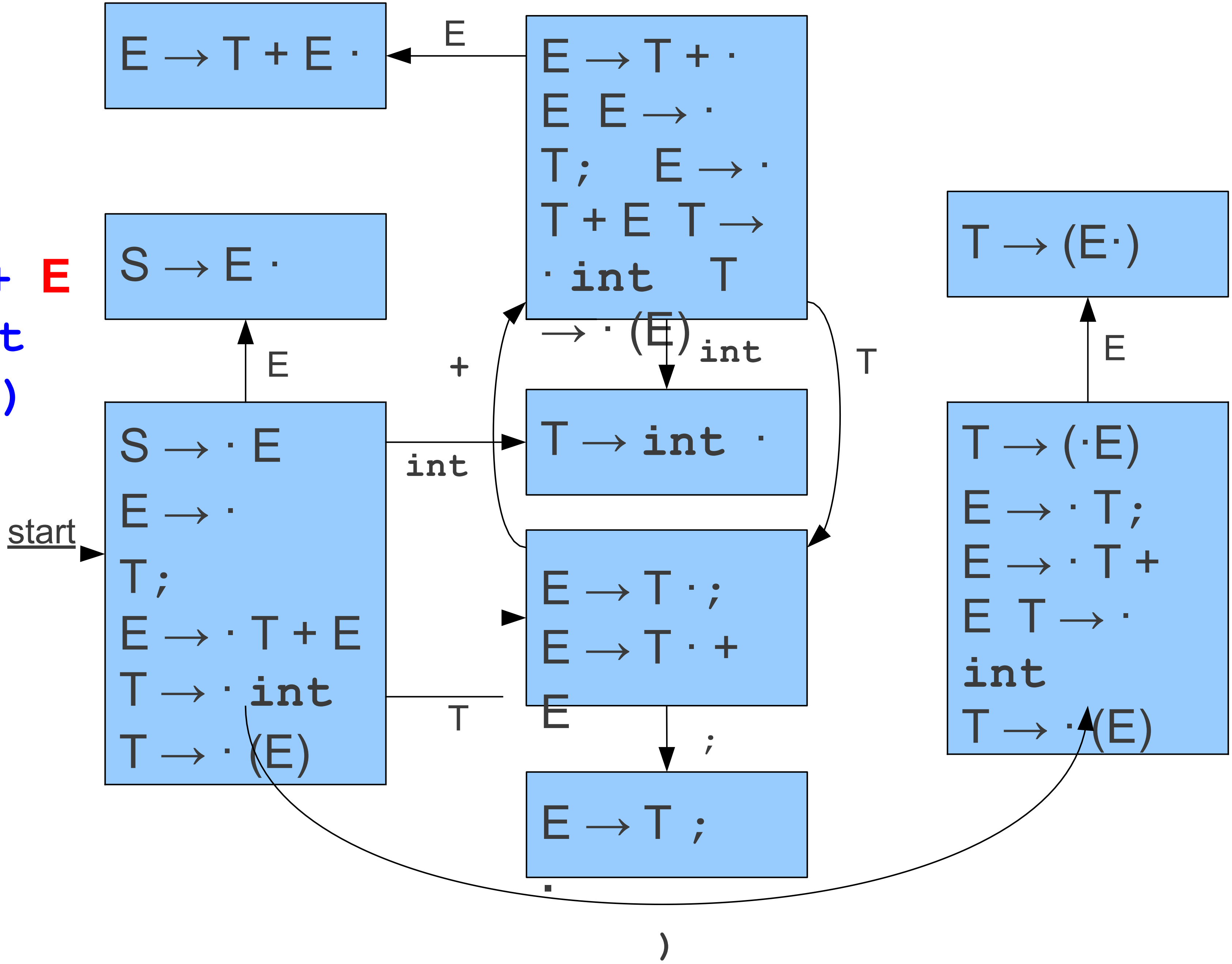
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

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

S → E ·

T → (E·)

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

T → int ·

E → T · ;
E → T · +
E

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

E → T ;
.

A Deterministic Automaton

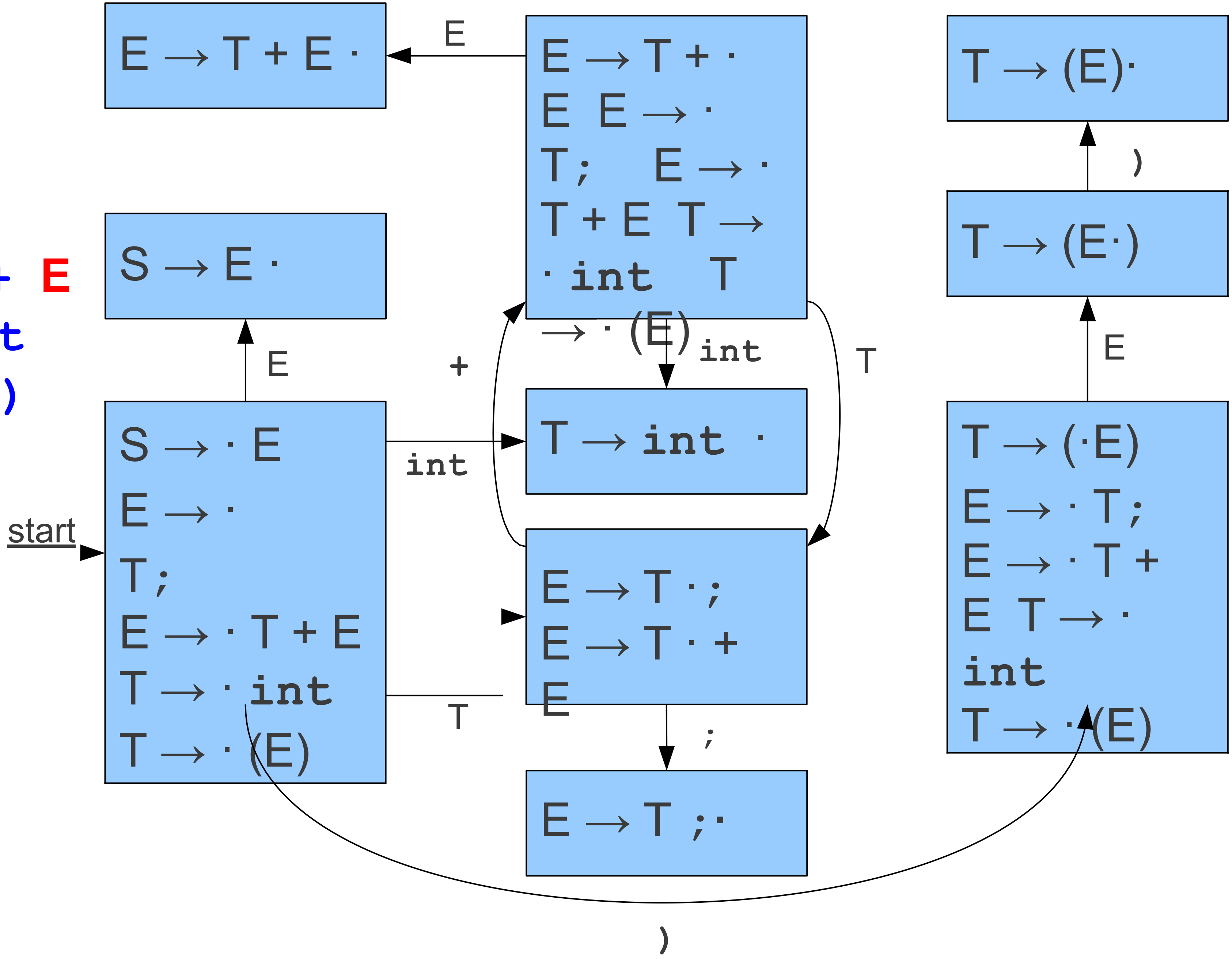
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

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

T → (E)·

S → E ·

T → (E·)

T → int ·

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

E → T ·;
E → T · +
E

E → T ;·

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

A Deterministic Automaton

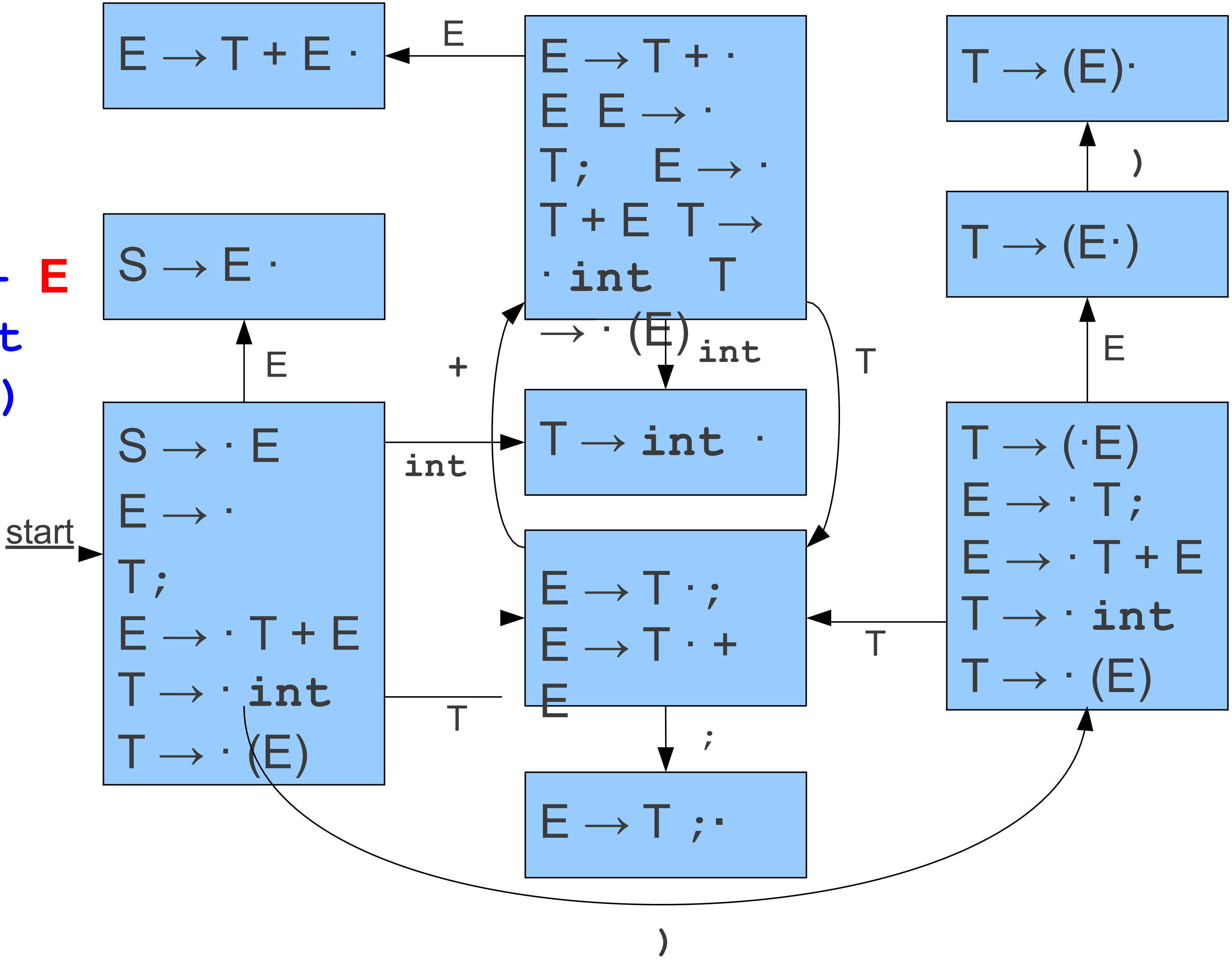
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

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

T → (E)·

S → E ·

T → (E·)

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

T → int ·

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

E → T ·;
E → T · +
E

E → T ;·

A Deterministic Automaton

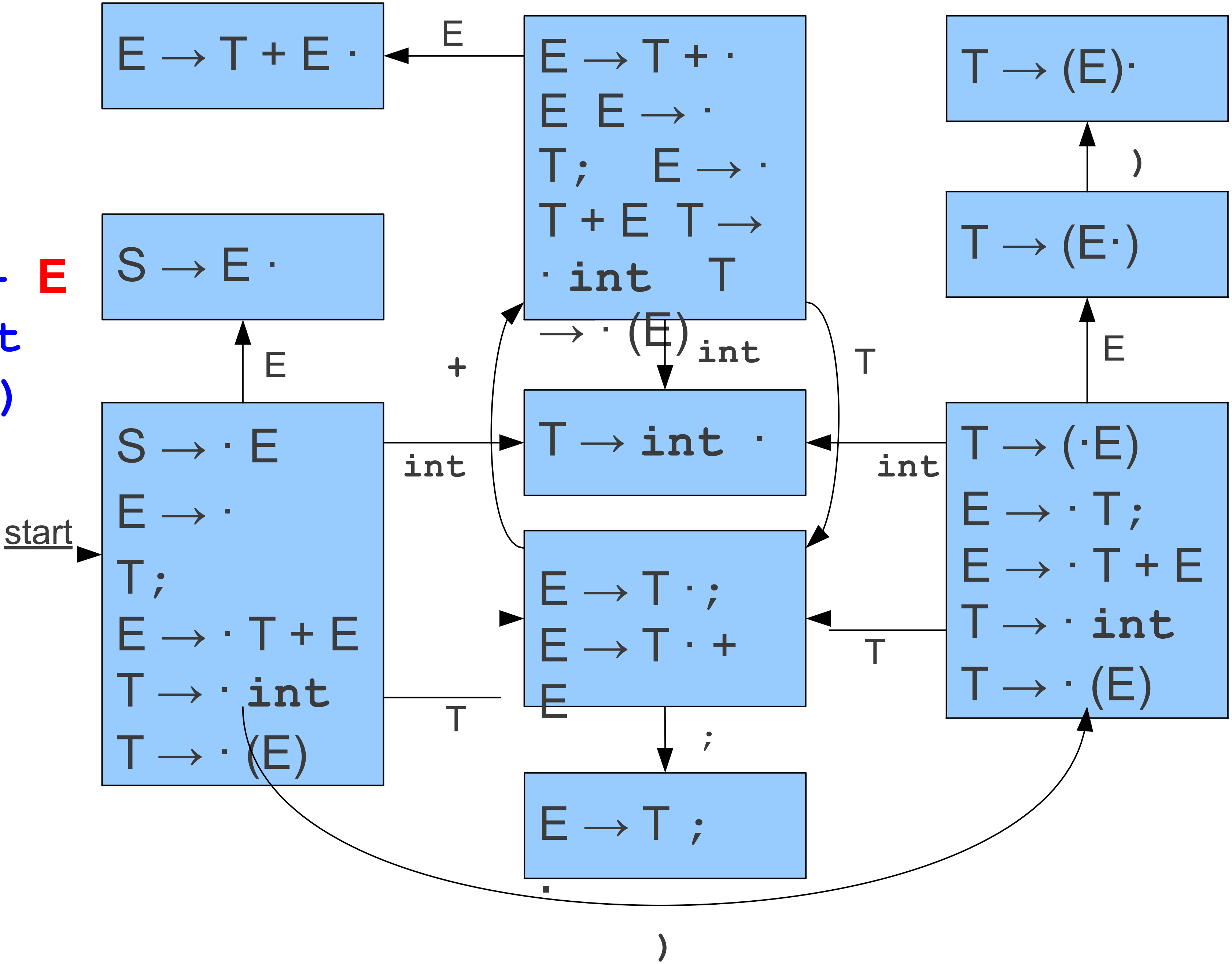
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

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

T → (E)·

T → (E·)

S → E ·

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

T → int ·

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

E → T ·;
E → T · +
E

E → T ;
.

A Deterministic Automaton

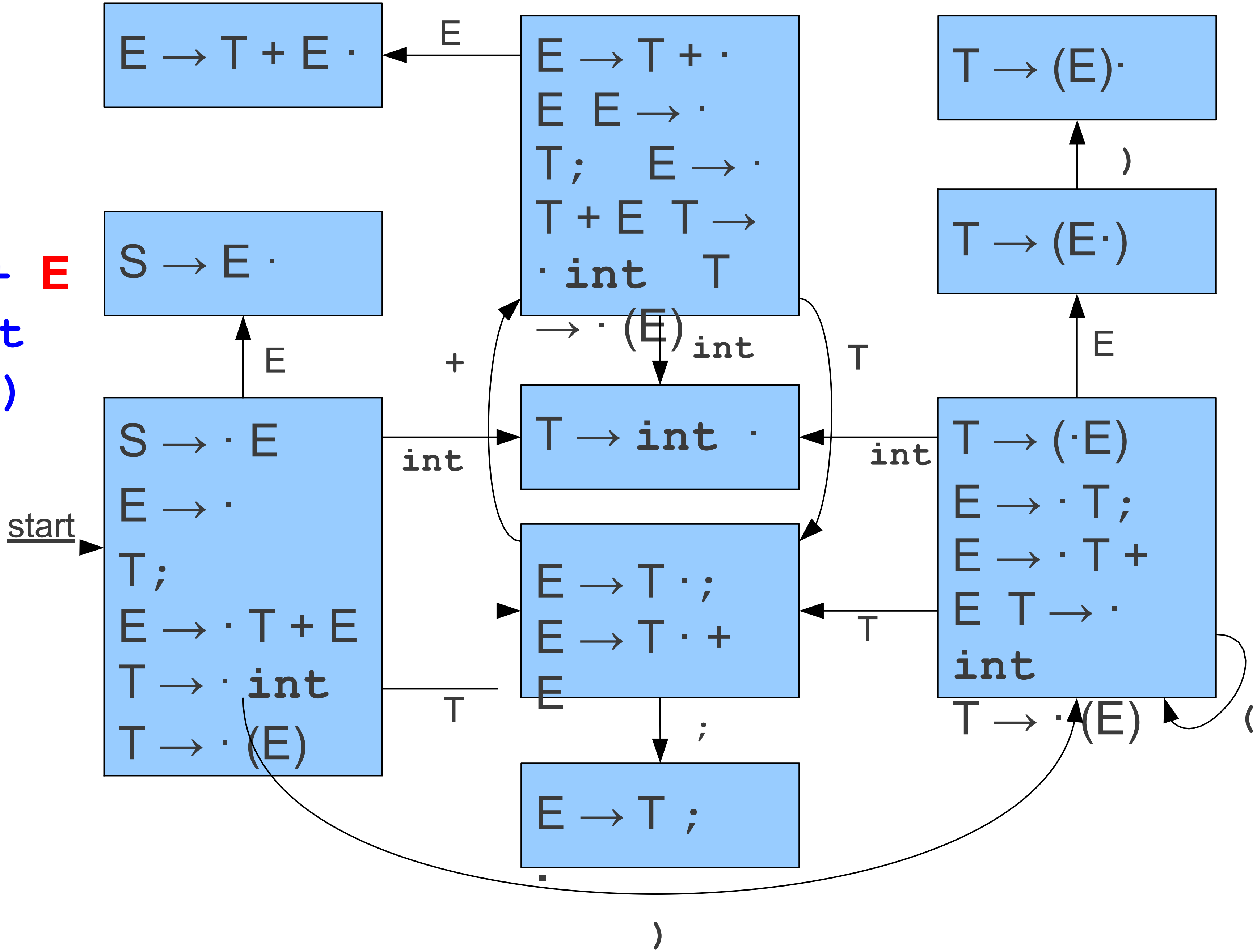
S → **E**

E → **T ;**

E → **T + E**

T → **int**

T → **(E)**



E → T + E ·

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

T → (E)·

S → E ·

T → (E·)

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

T → int ·

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

E → T ·;
E → T · +
E

E → T ;
·

A Deterministic Automaton

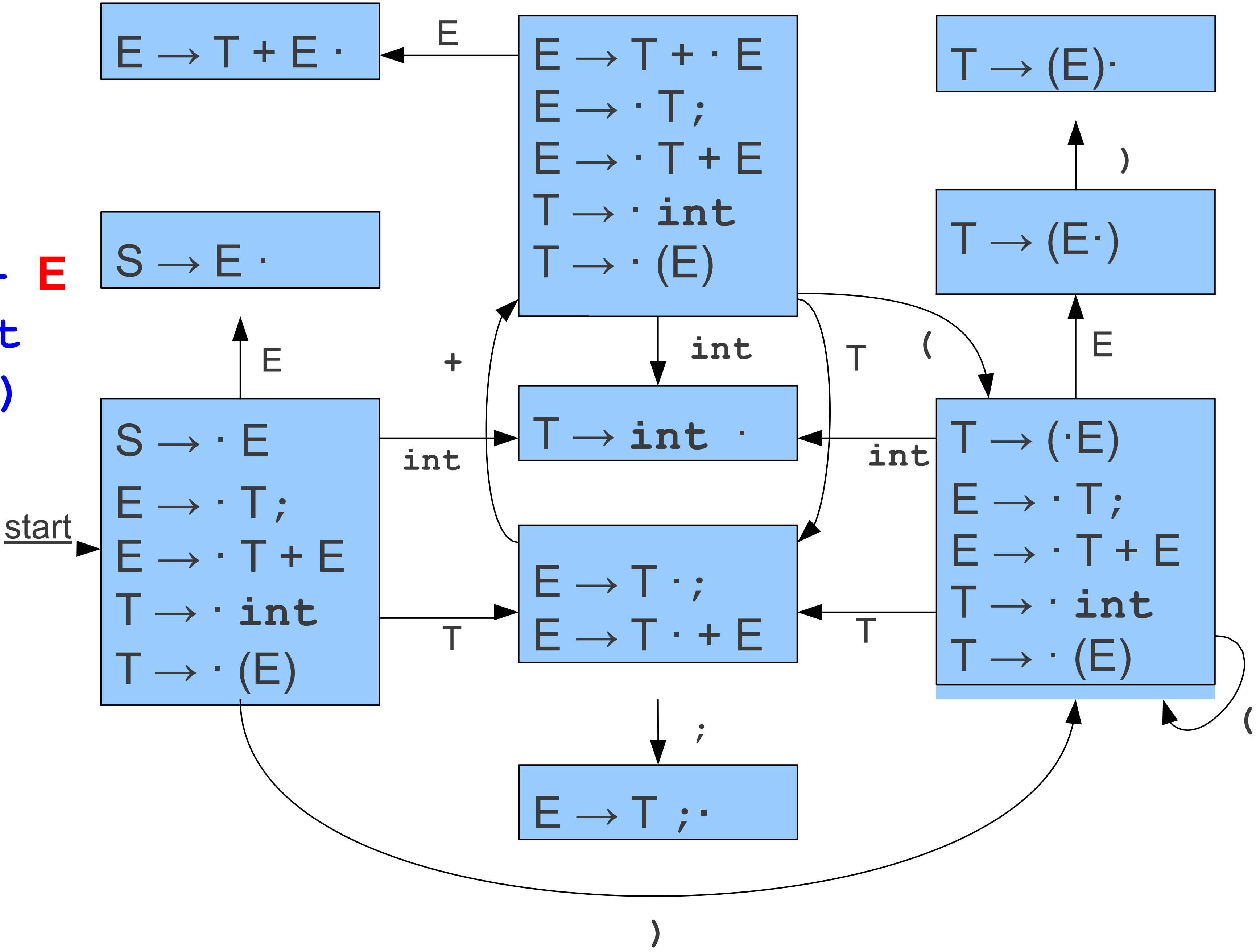
S → E

E → T;

E → T + E

T → int

T → (E)



Handle-Finding Automata

- Handling-finding automata can be very large.
- NFA has states proportional to the size of the grammar, so DFA can have size exponential in the size of the grammar.
    - There are grammars that can exhibit this worst-case.
- Automata are almost always generated by tools like **`bison`**.

Finding Handles

- Where do we look for handles?
  - **At the top of the stack.**
- How do we search for handles?
  - **Build a handle-finding automaton**.
- How do we recognize handles?
  - Once we've found a possible handle, how do we confirm that it's correct?

Question Three:

How do we recognize handles?

Handle Recognition

- Our automaton will tell us all places where a handle might be.
- However, if the automaton says that there might be a handle at a given point, we need a way to confirm this.

We'll thus use **predictive bottom-up parsing**:

Have a deterministic procedure for guessing where handles are.

There are many predictive algorithms, each of which recognize different grammars.

This decision is the difference of BU-Parssing Algorithms.

Our First Algorithm: **LR(0)**

- Bottom-up predictive parsing with:
  - **L**: **L**eft-to-right scan of the input.
  - **R**: **R**ightmost derivation.
  - (**0**): Zero tokens of **lookahead**.
- Use the handle-finding automaton, without any lookahead, to predict where handles are.

We talk about lookahead later.

LR(0) Parsing

S → E

E → T;

E → T + E

T → int

T → (E)

int	+	(	int	+	int	;	)	;

LR(0) Parsing

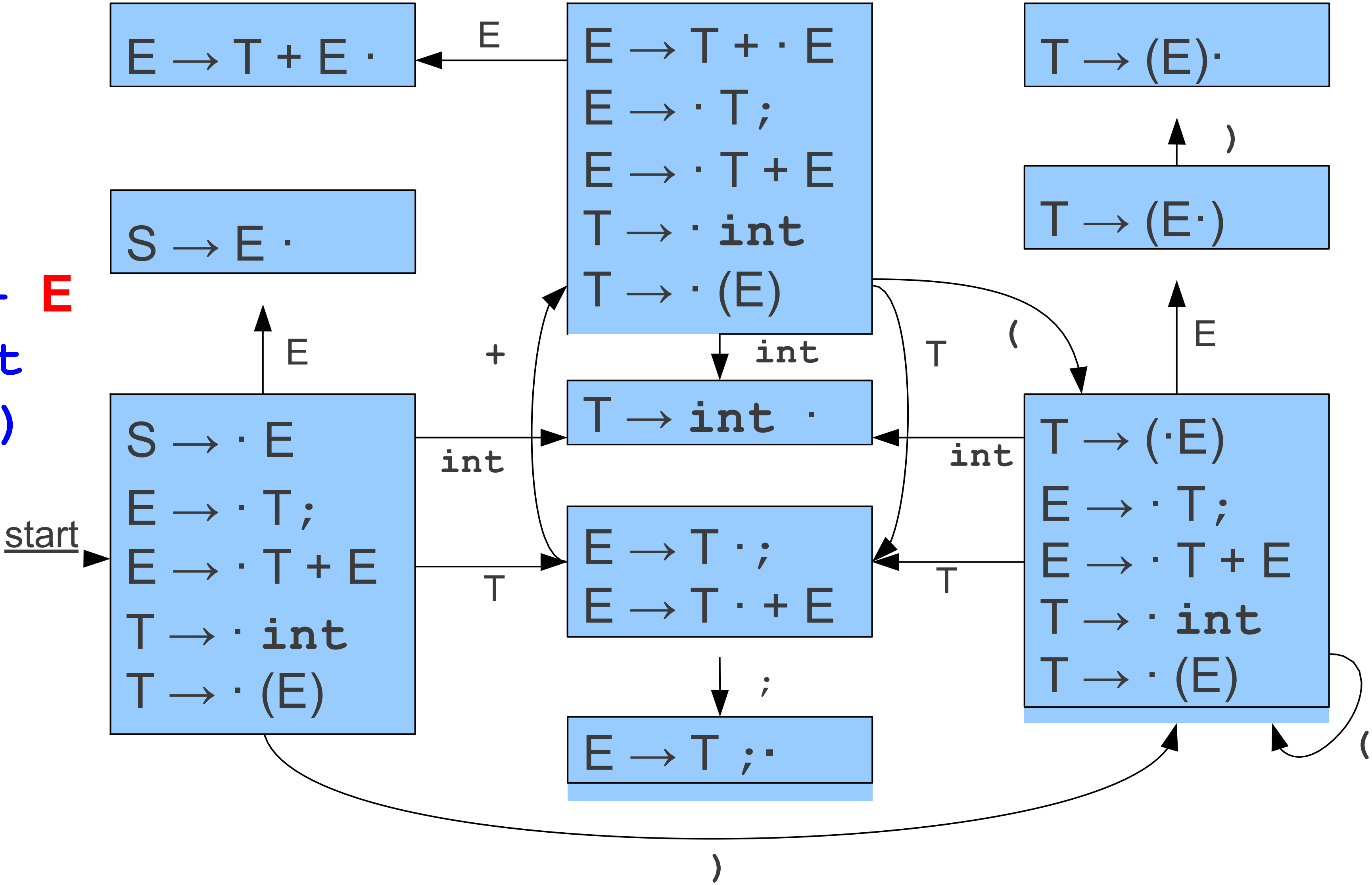
S → E

E → T;

E → T + E

T → int

T → (E)



int	+	(	int	+	int	;	)	;

LR(0) Parsing

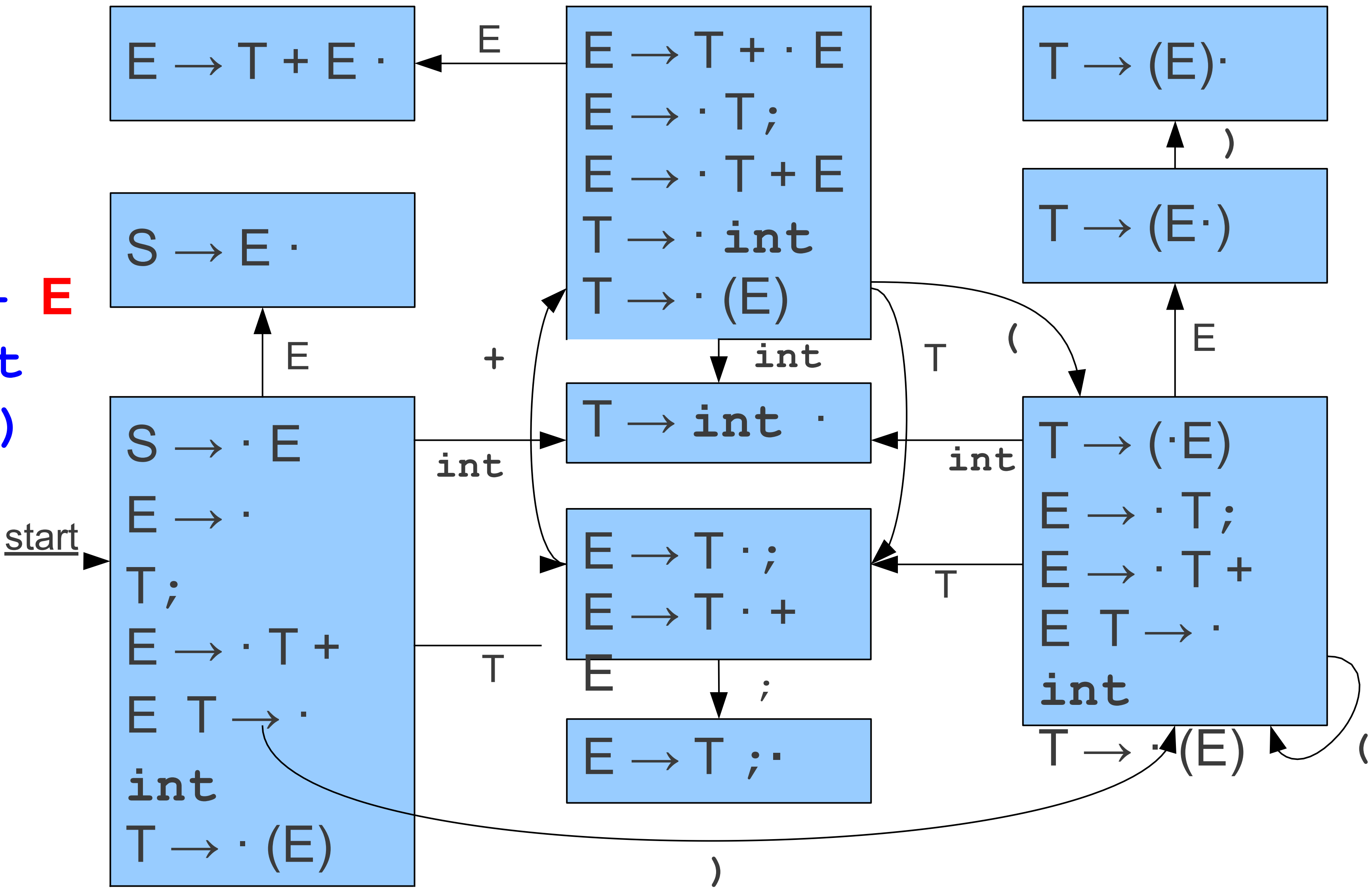
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

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

T → (E)·

S → E ·

T → (E·)

T → int ·

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

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

E → T · ;
E → T · +
E

E → T ;·

int	+	(	int	+	int	;	)	;

LR(0) Parsing

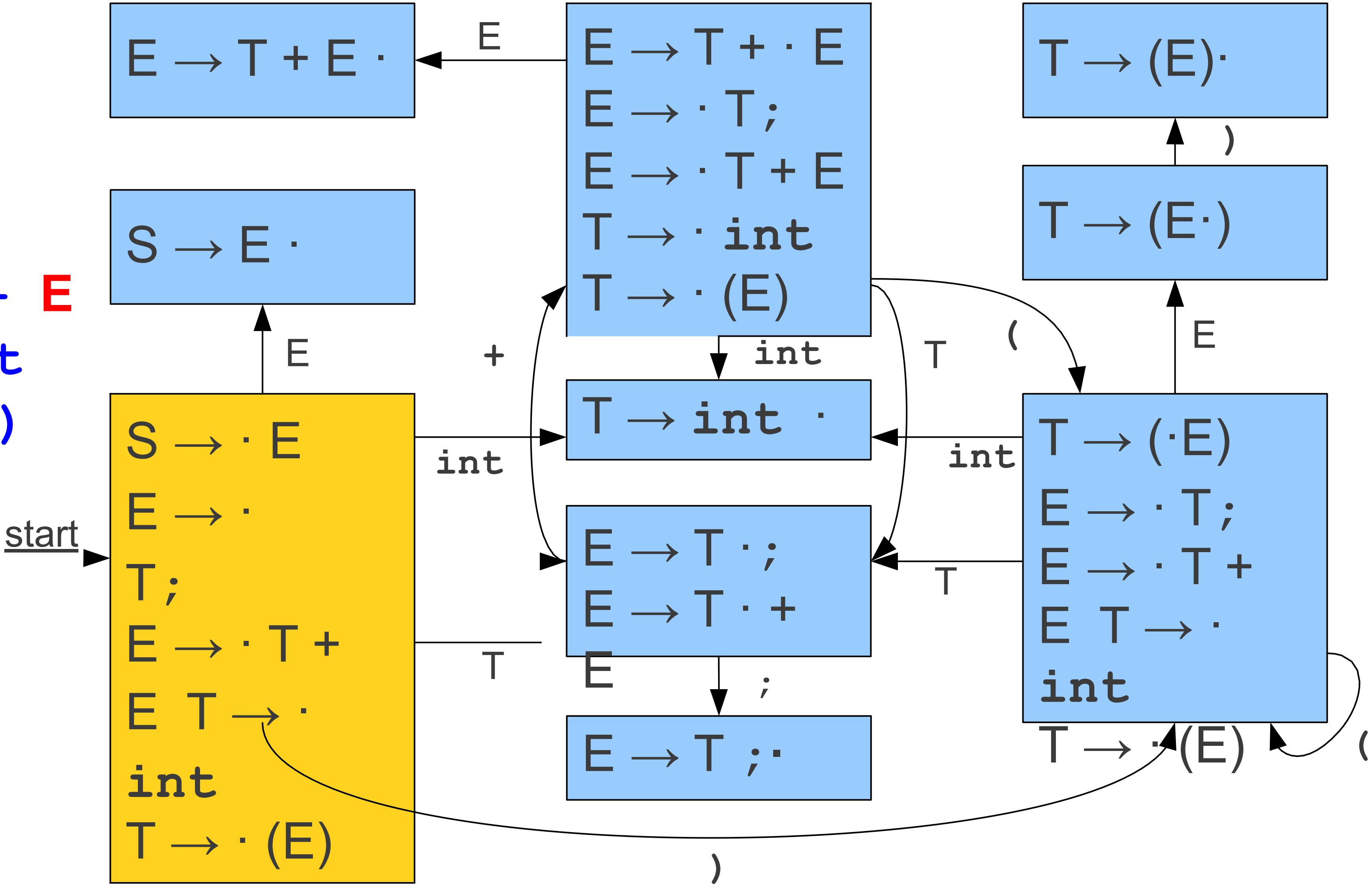
S → E

E → T;

E → T + E

T → int

T → (E)



start

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

S → E ·

E → T + E ·

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

T → int ·

E → T · ;
E → T · +
E

E → T ; ·

T → (E)·

T → (E·)

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

int	+	(	int	+	int	;	)	;

LR(0) Parsing

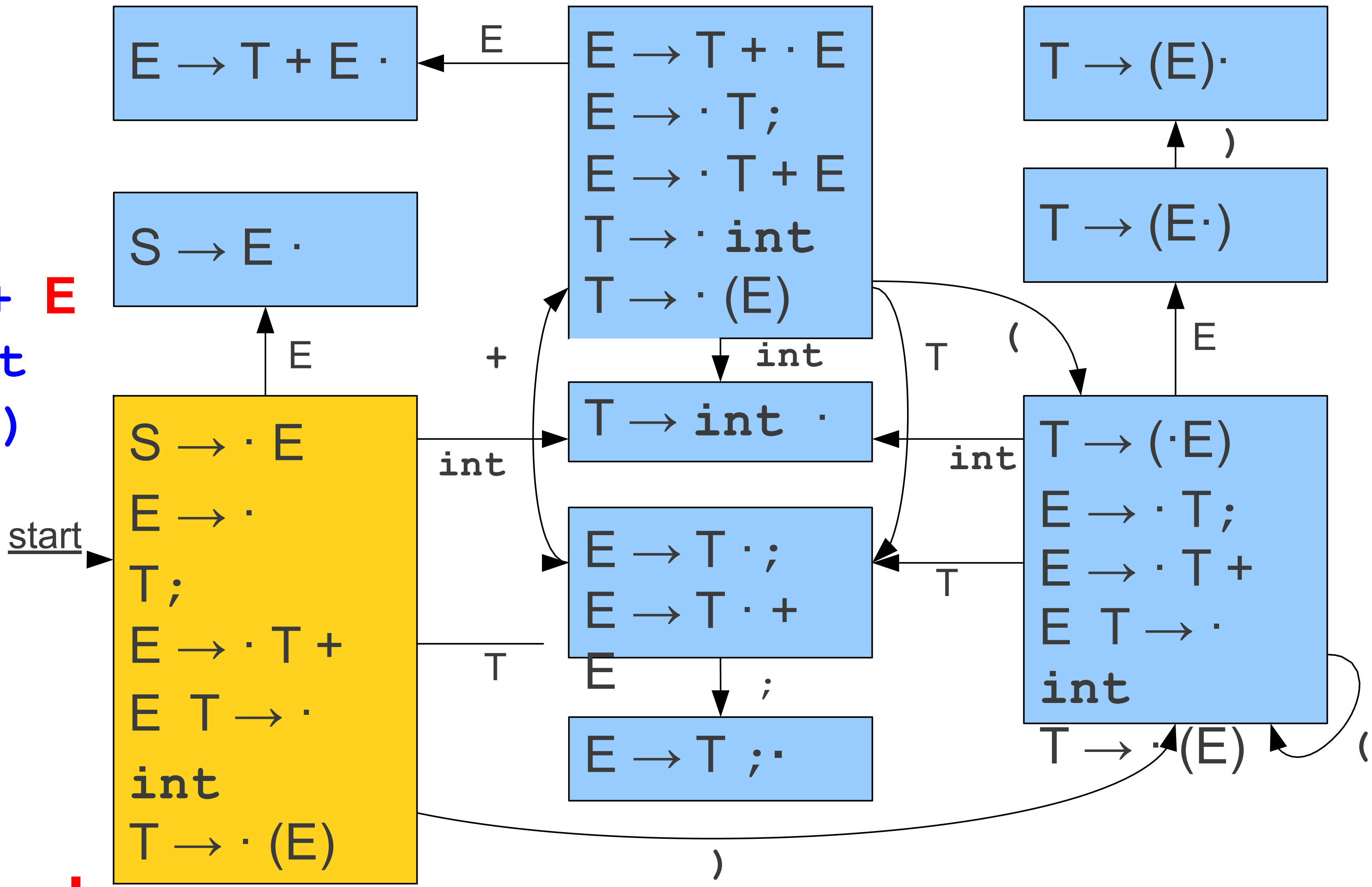
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

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

T → (E)·

S → E ·

T → (E·)

T → int ·

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

E → T · ;
E → T · +
E

E → T ;·

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

int

+	(	int	+	int	;	)	;

LR(0) Parsing

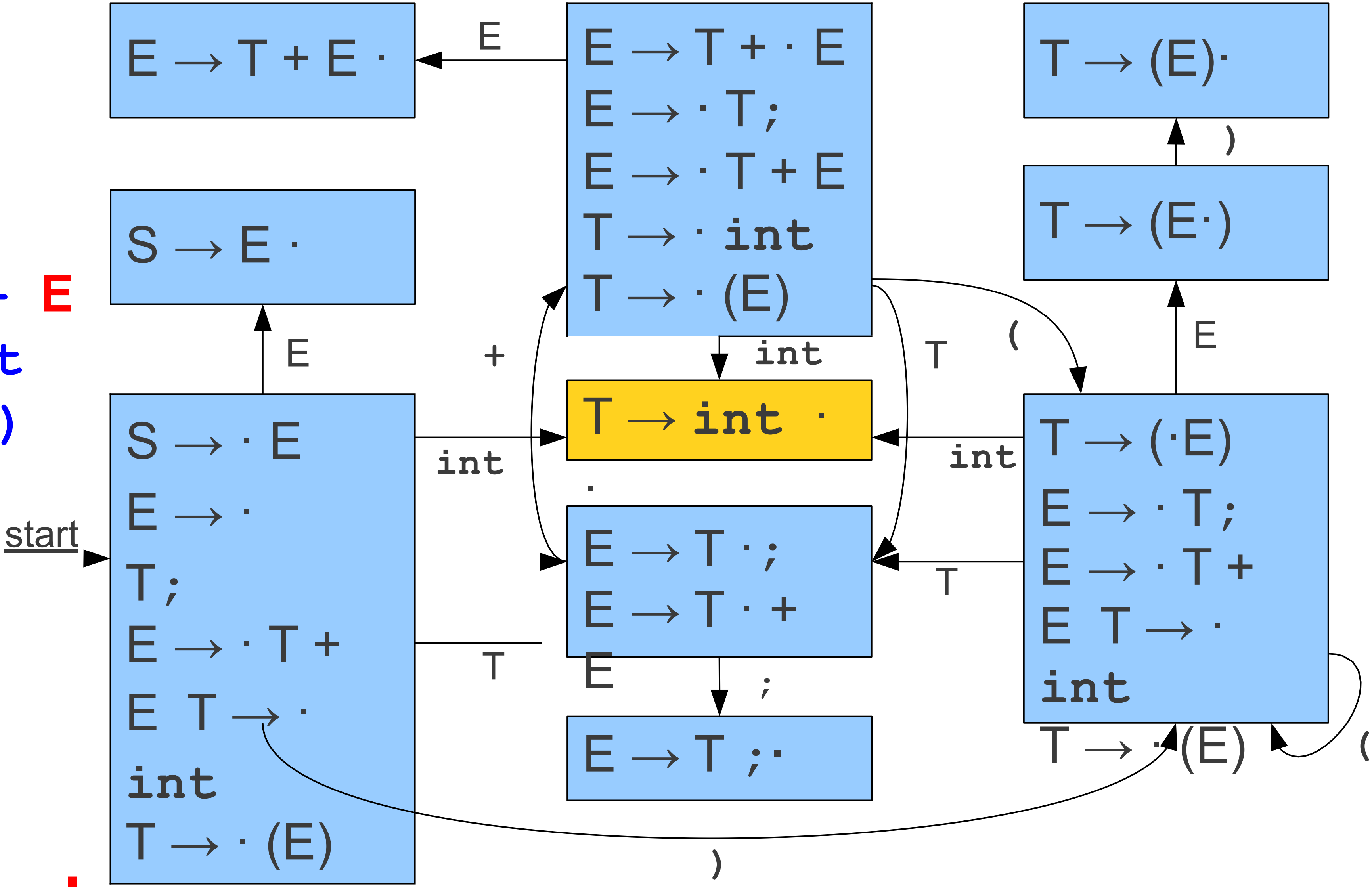
S → E

E → T;

E → T + E

T → int

T → (E)



int		+	(	int	+	int	;	)	;

LR(0) Parsing

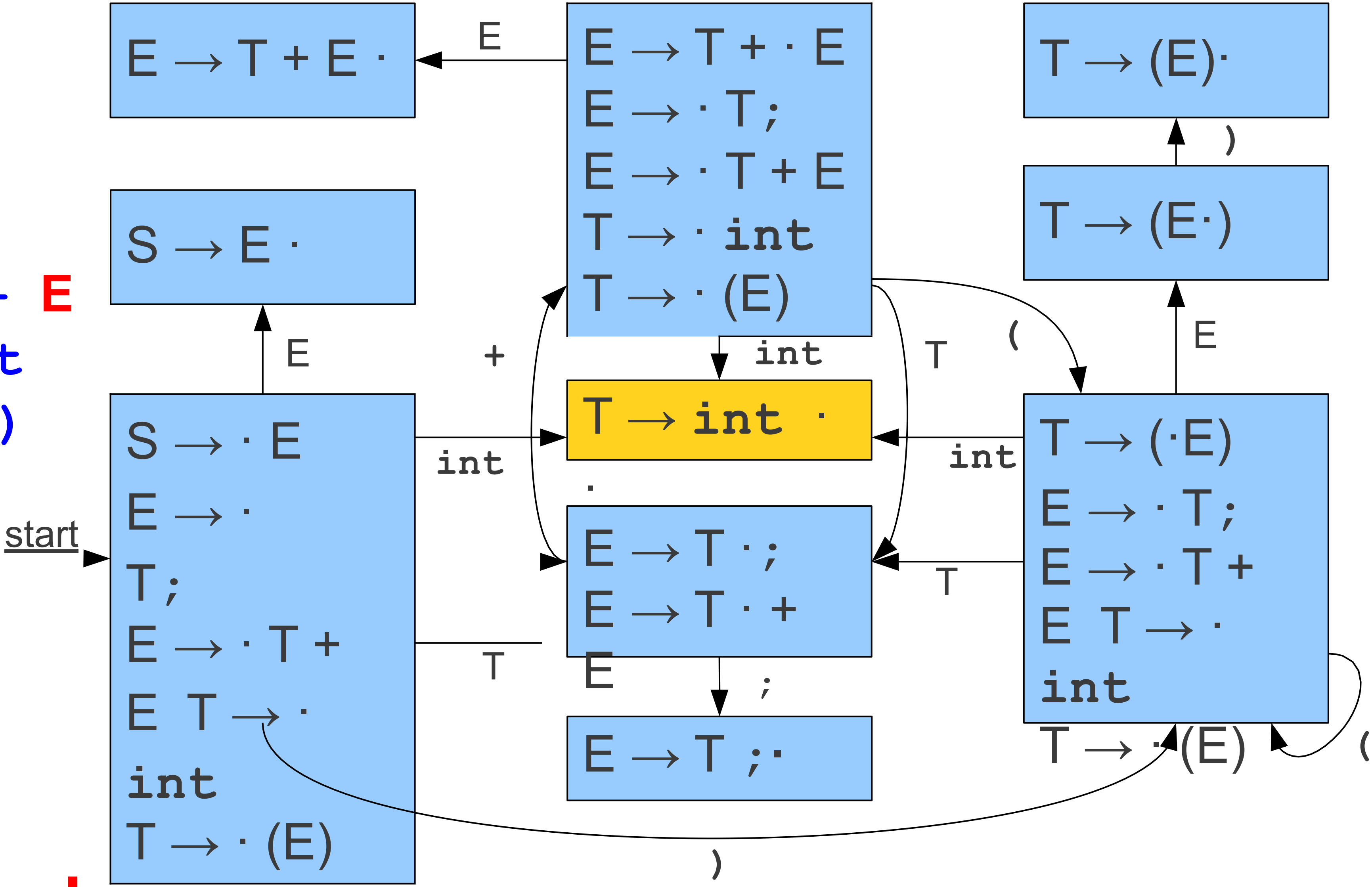
S → E

E → T;

E → T + E

T → int

T → (E)



+	(	int	+	int	;	)	;

LR(0) Parsing

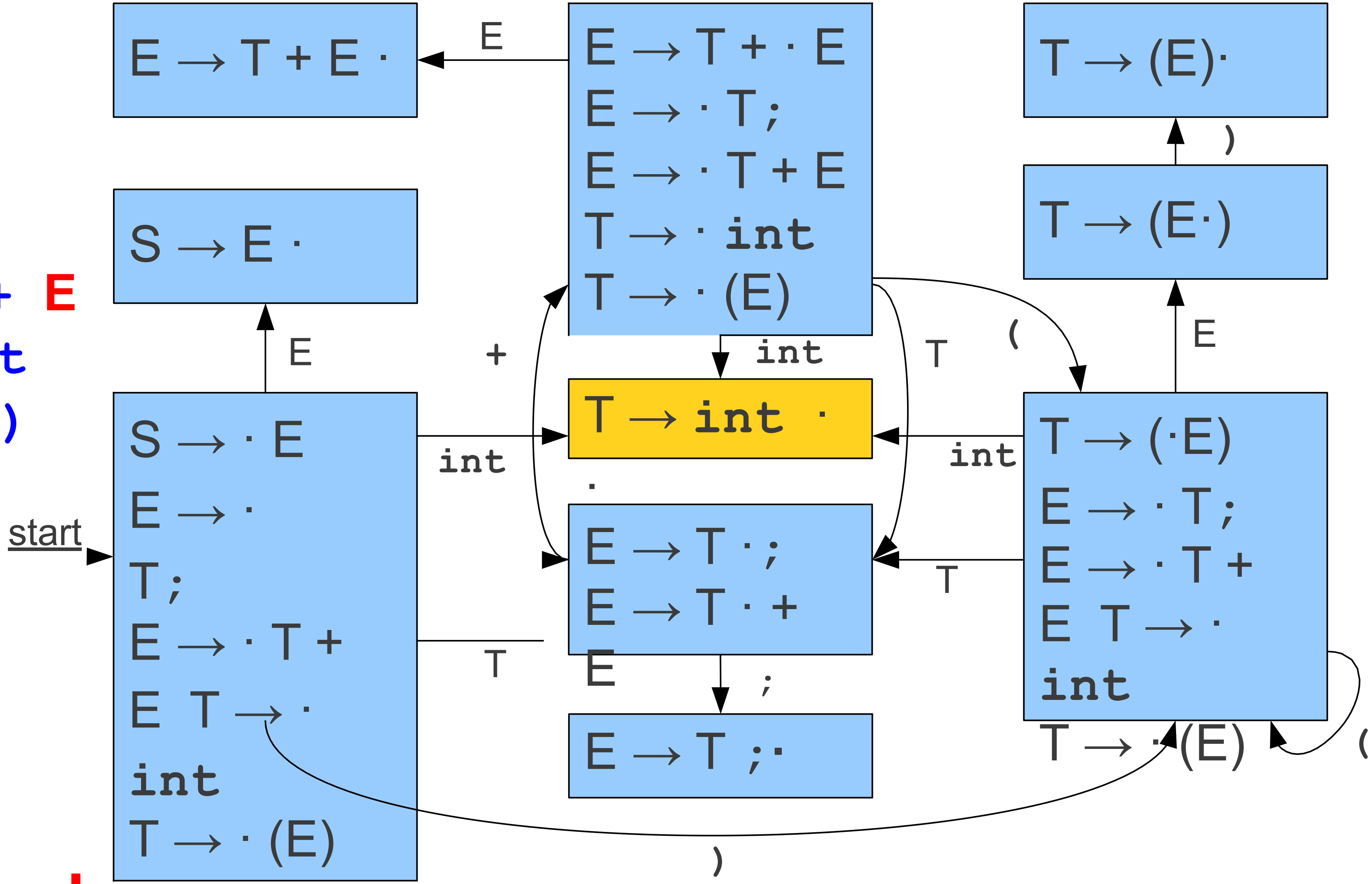
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

S → E ·

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

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

T → int ·

E → T · ;
E → T · +
E

E → T ;·

T → (E)·

T → (E·)

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

T

+	(	int	+	int	;	)	;

LR(0) Parsing

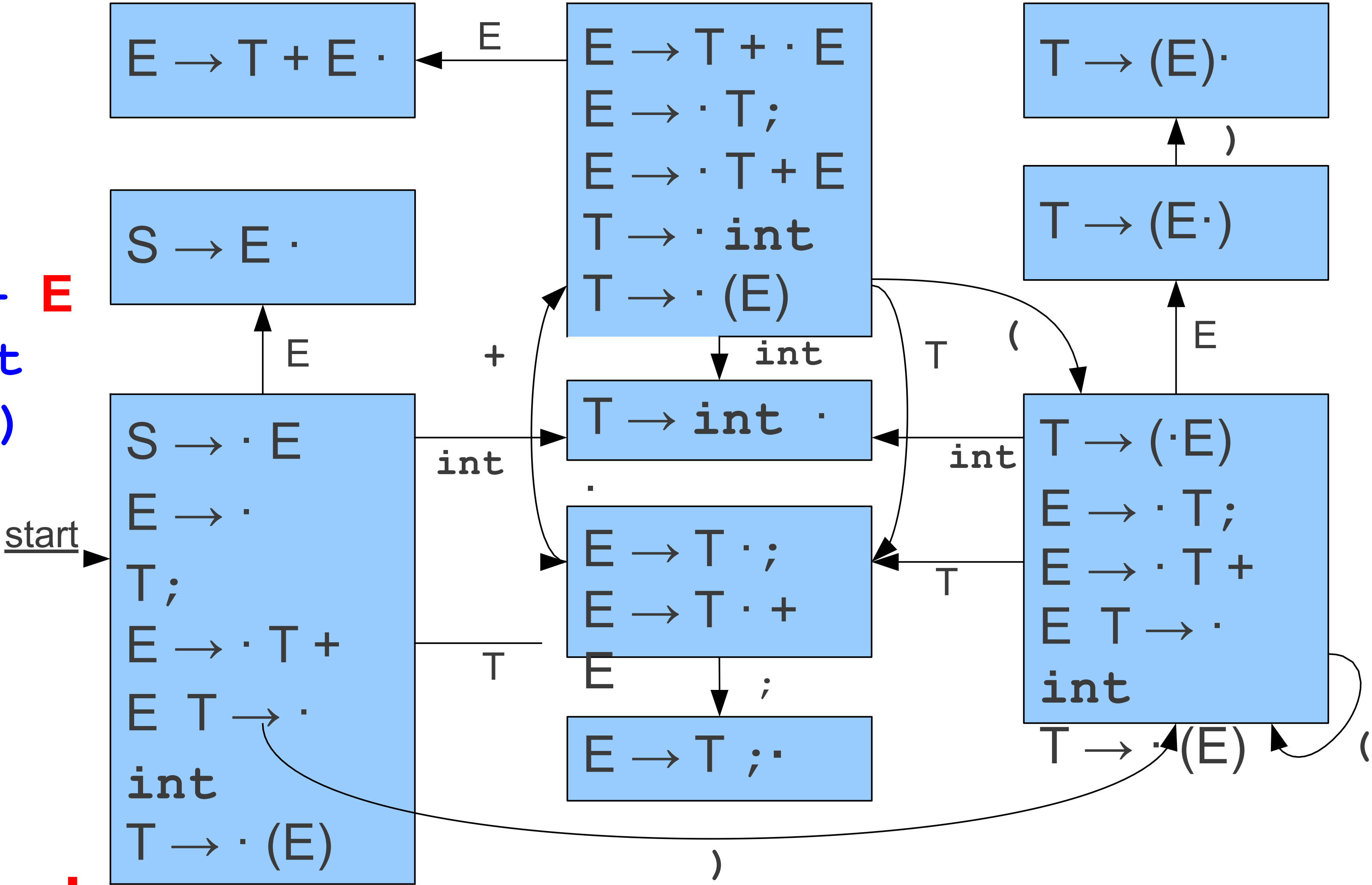
S → E

E → T;

E → T + E

T → int

T → (E)



T

+	(	int	+	int	;	)	;

LR(0) Parsing

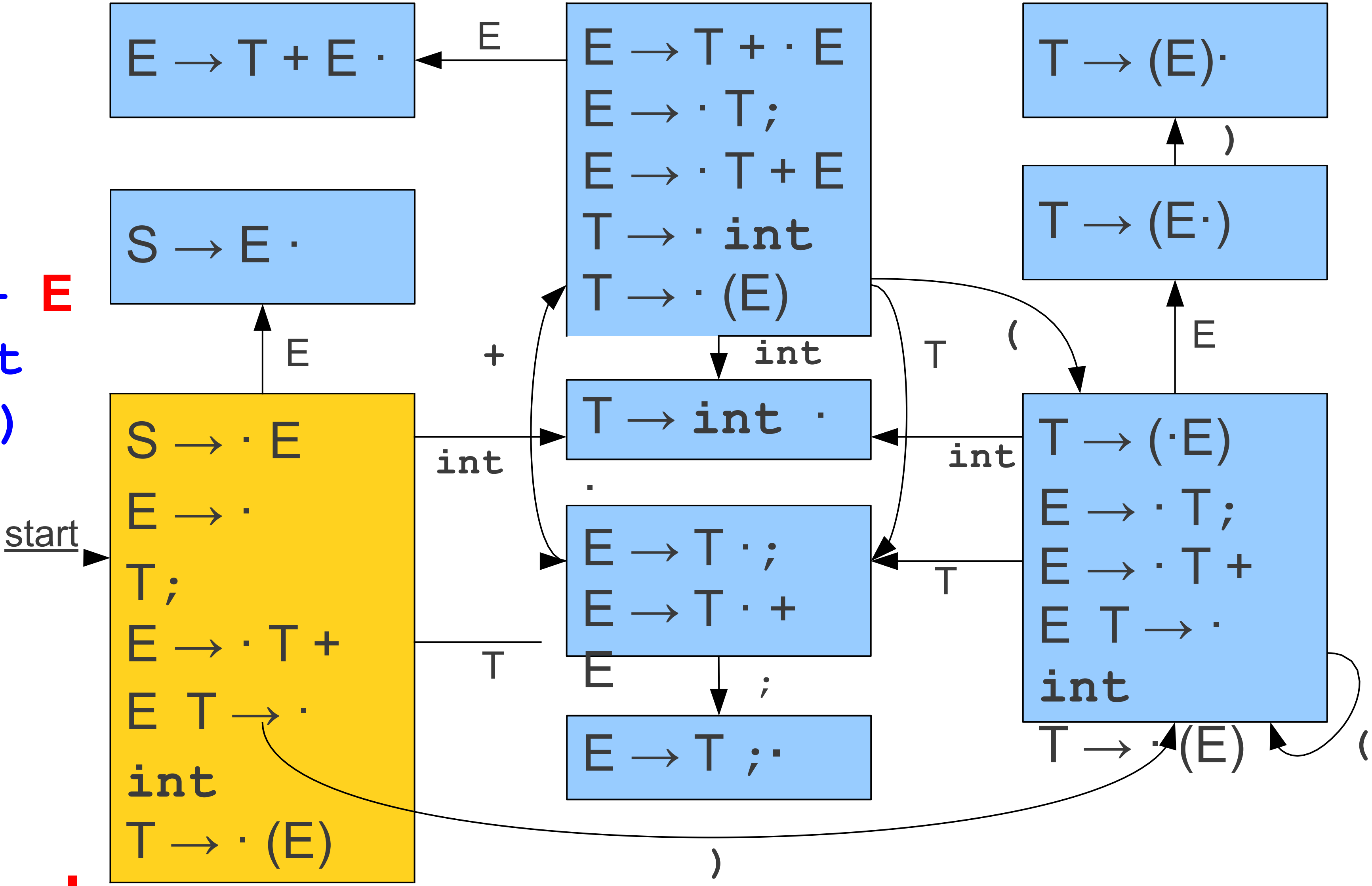
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(	int	+	int	;	)	;

LR(0) Parsing

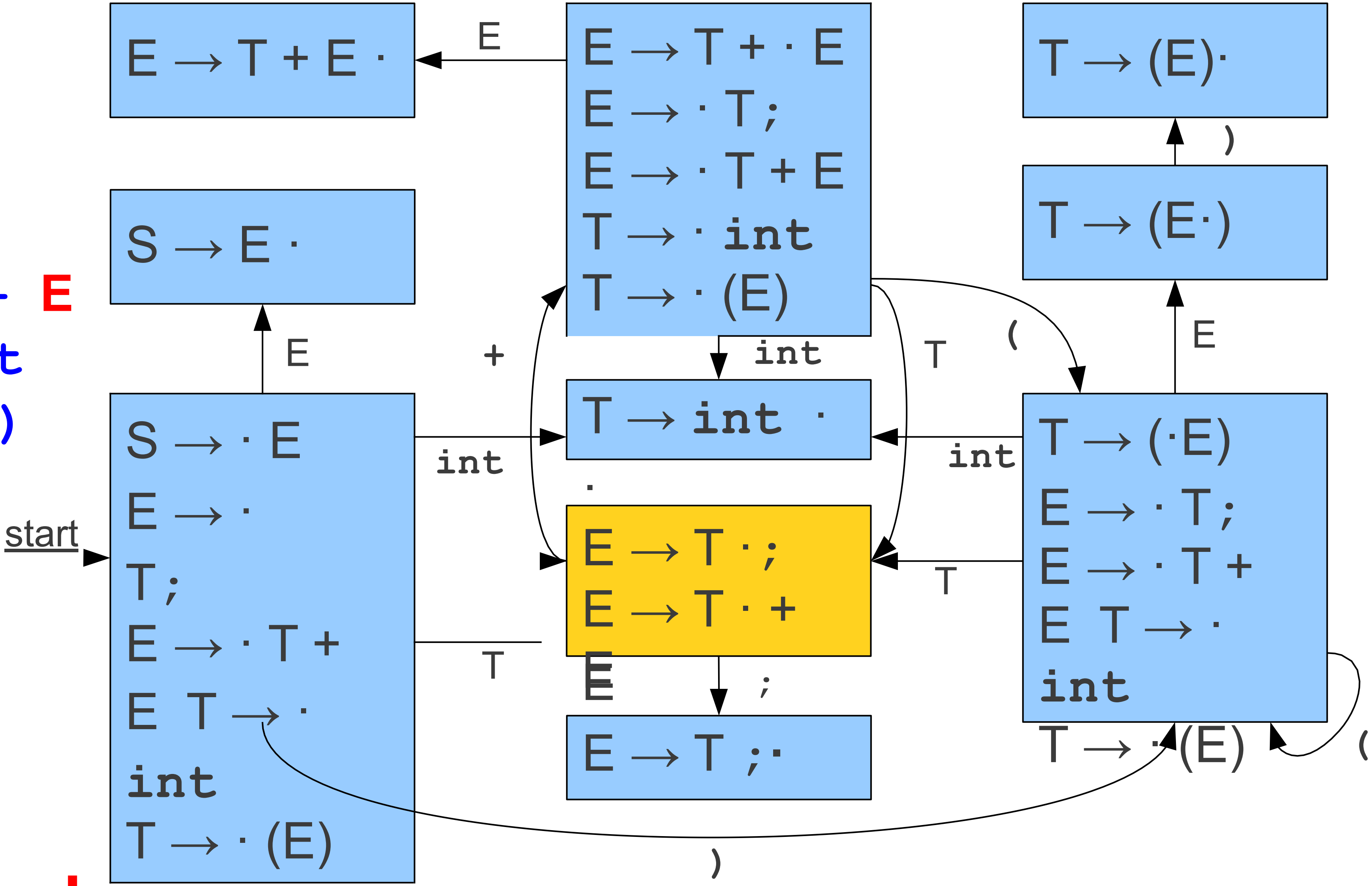
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(	int	+	int	;	)	;

LR(0) Parsing

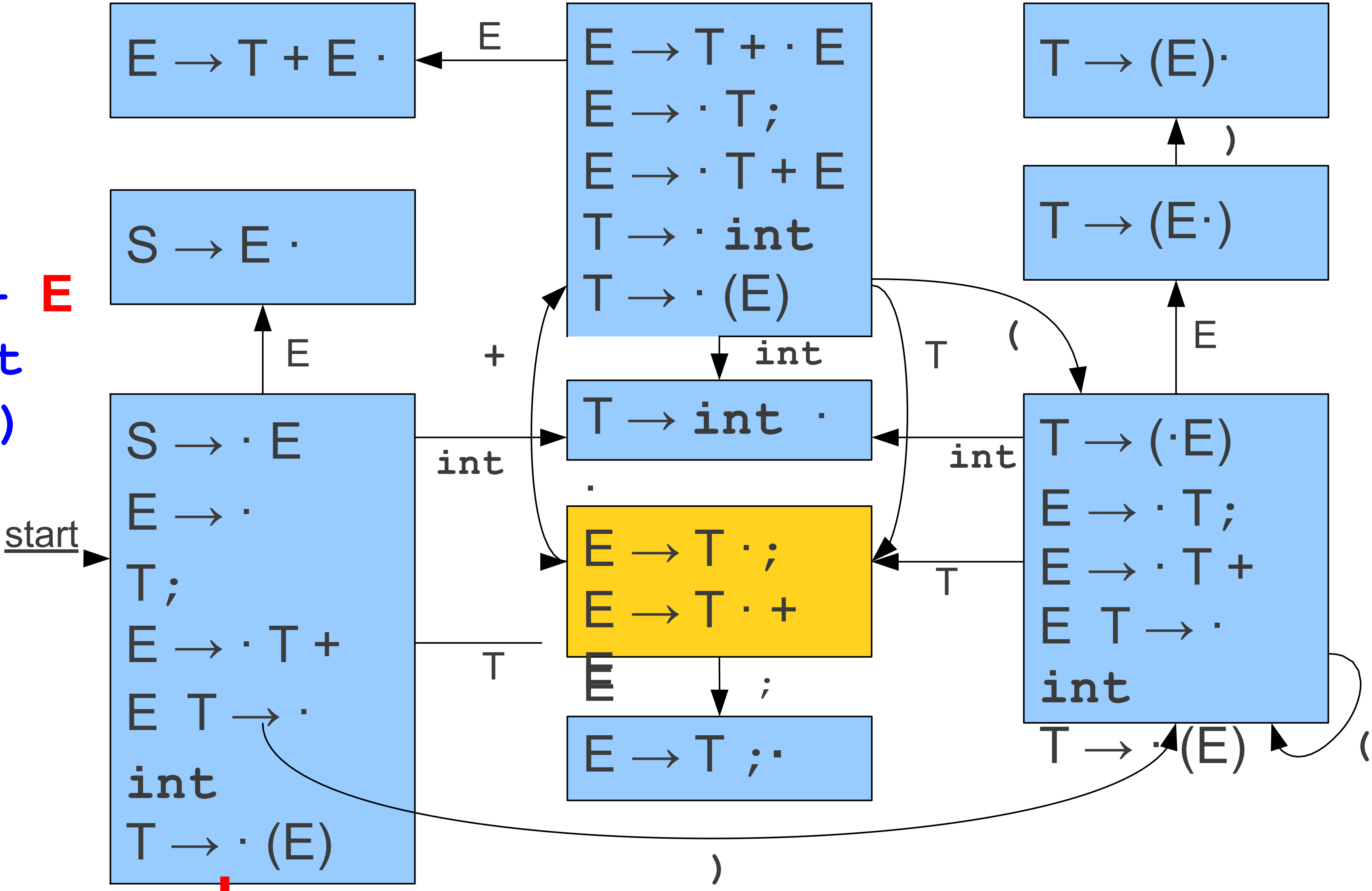
S → E

E → T;

E → T + E

T → int

T → (E)



T	+

(	int	+	int	;	)	;

LR(0) Parsing

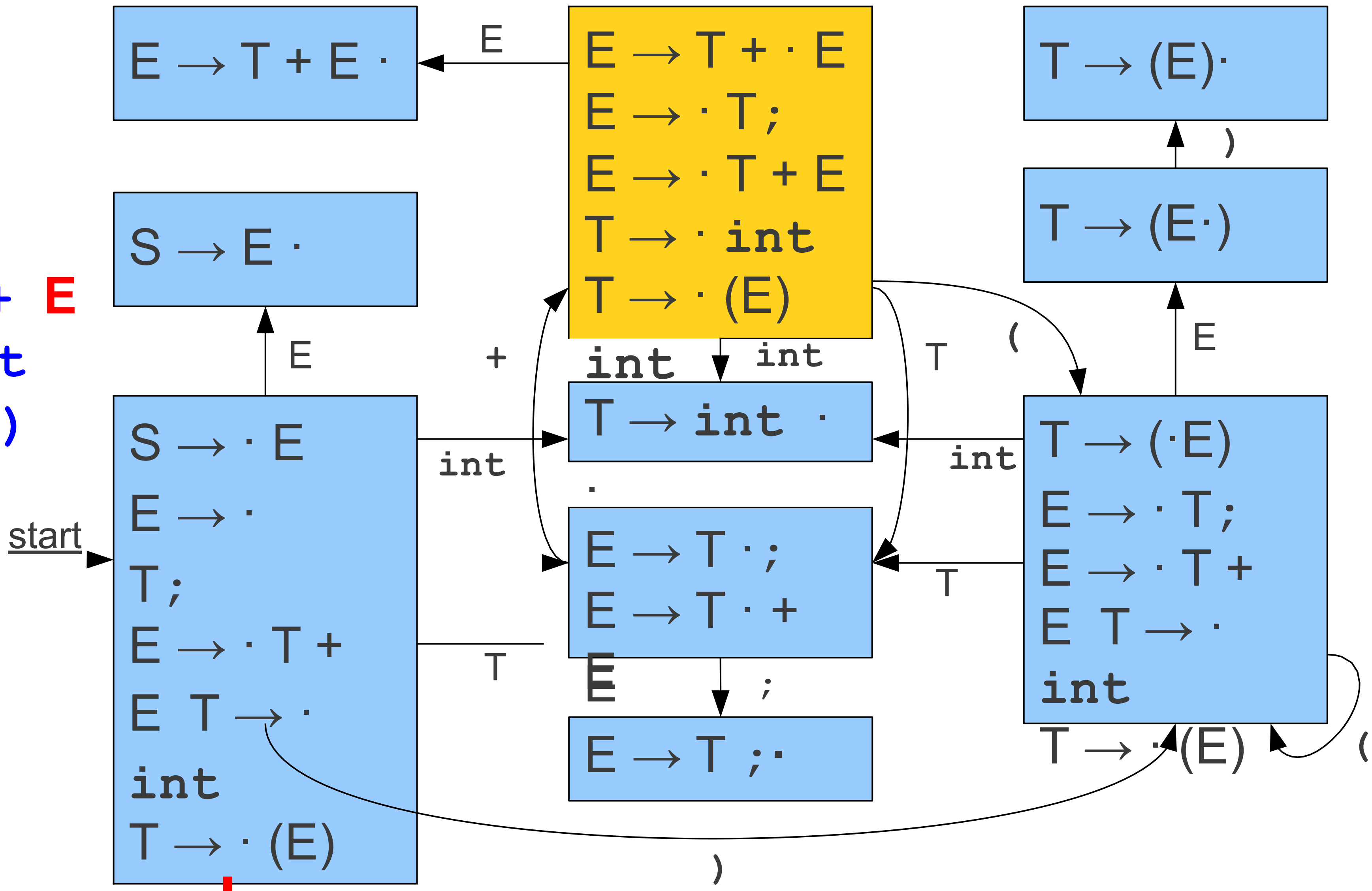
S → E

E → T;

E → T + E

T → int

T → (E)



T	+

(	int	+	int	;	)	;

LR(0) Parsing

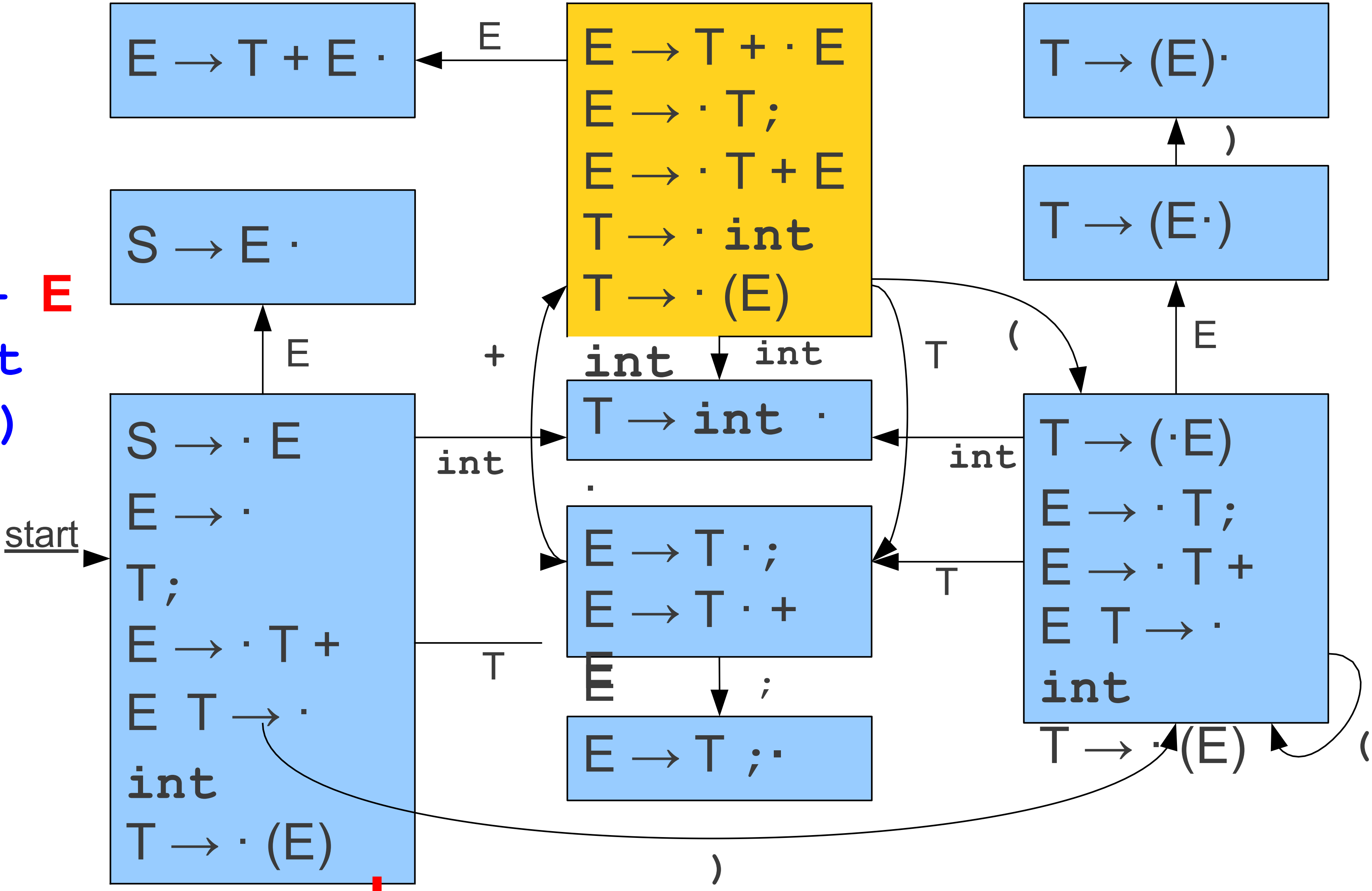
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(

int	+	int	;	)	;

LR(0) Parsing

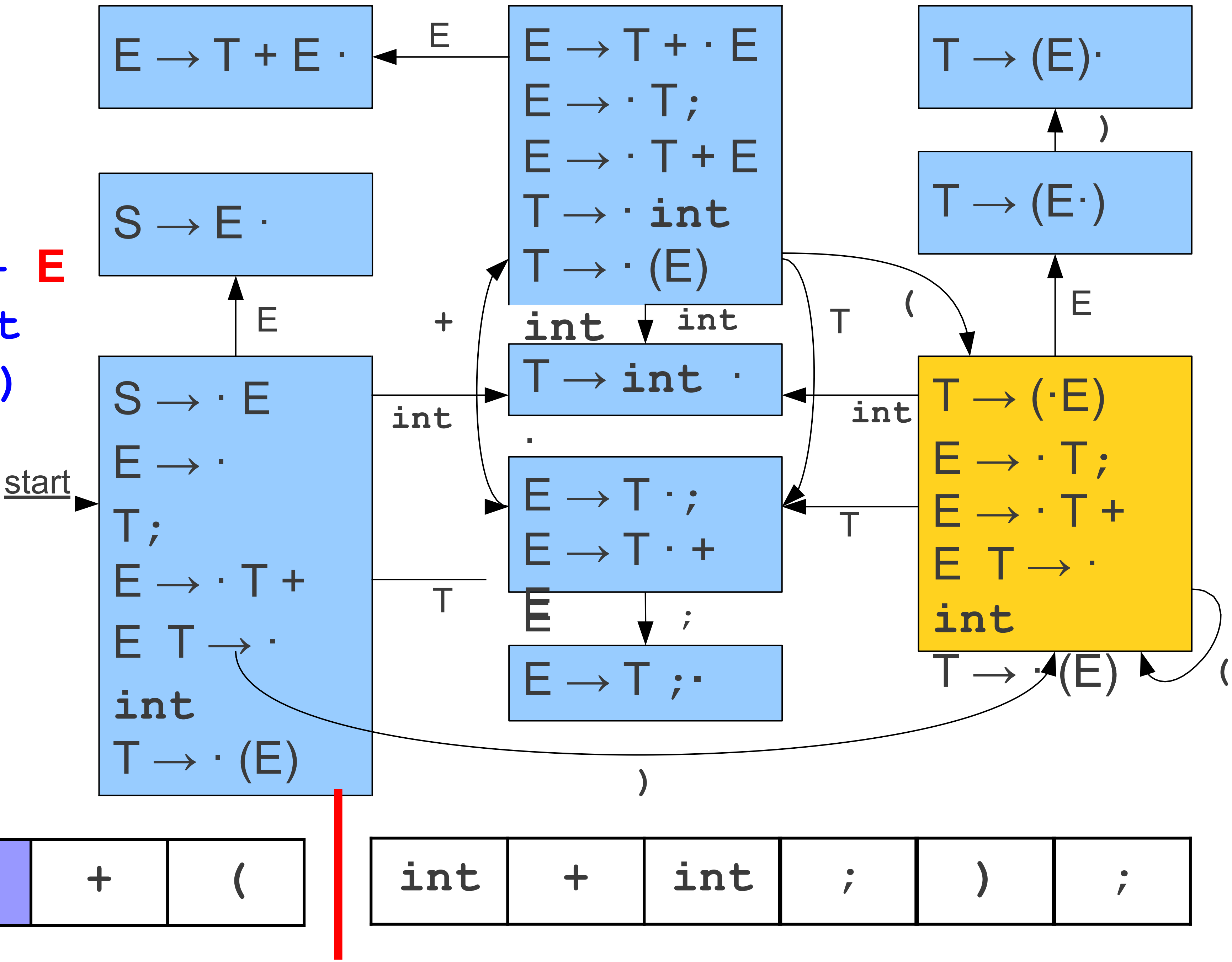
S → E

E → T;

E → T + E

T → int

T → (E)



start

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

S → E ·

E → T + E ·

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

T → int ·

E → T · ;
E → T · +
E

E → T ; ·

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

T → (E·)

T → (E)·

T + (

int + int ;) ;

LR(0) Parsing

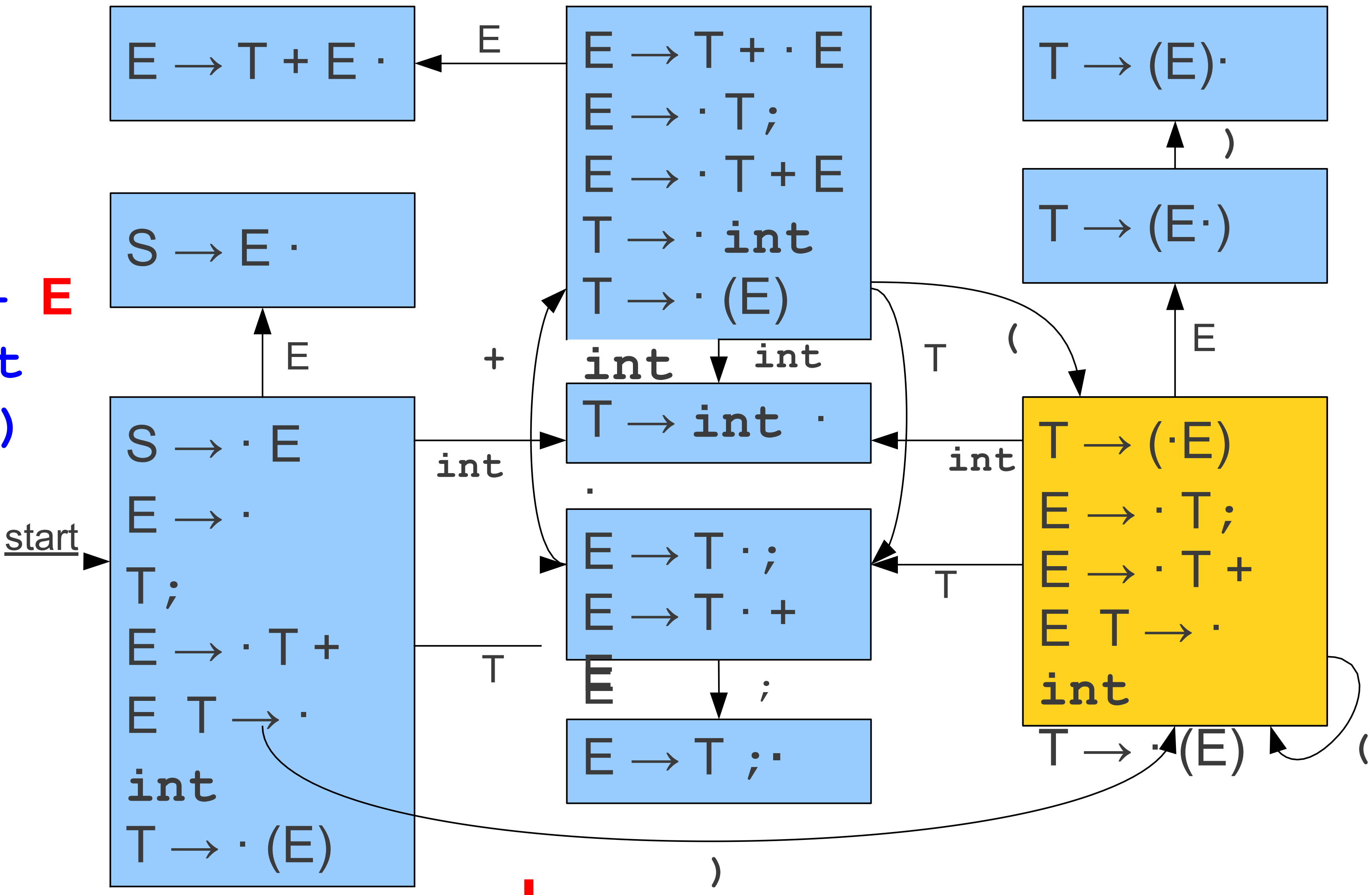
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(	int		+	int	;	)	;

LR(0) Parsing

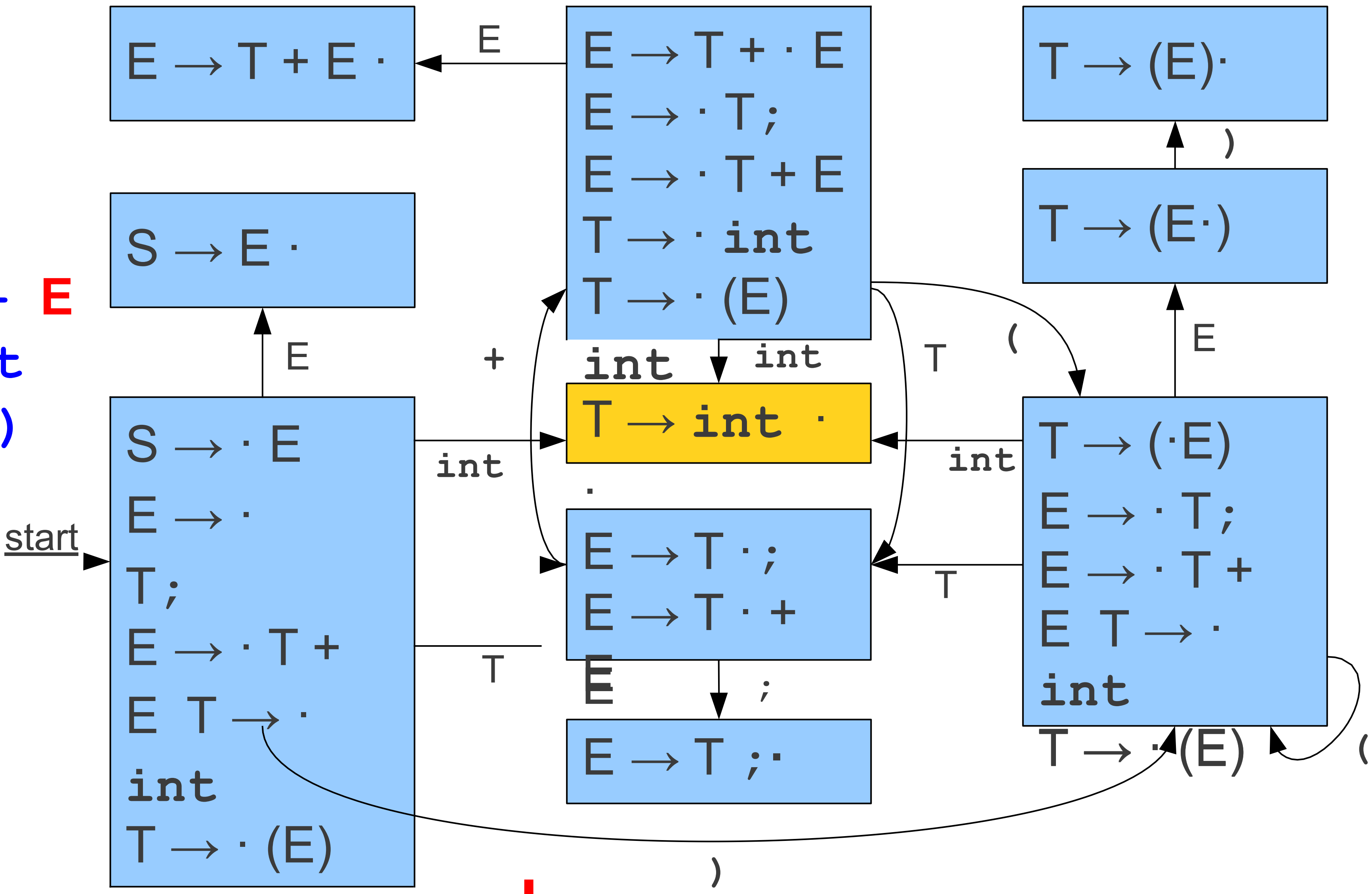
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(

+	int	;	)	;

LR(0) Parsing

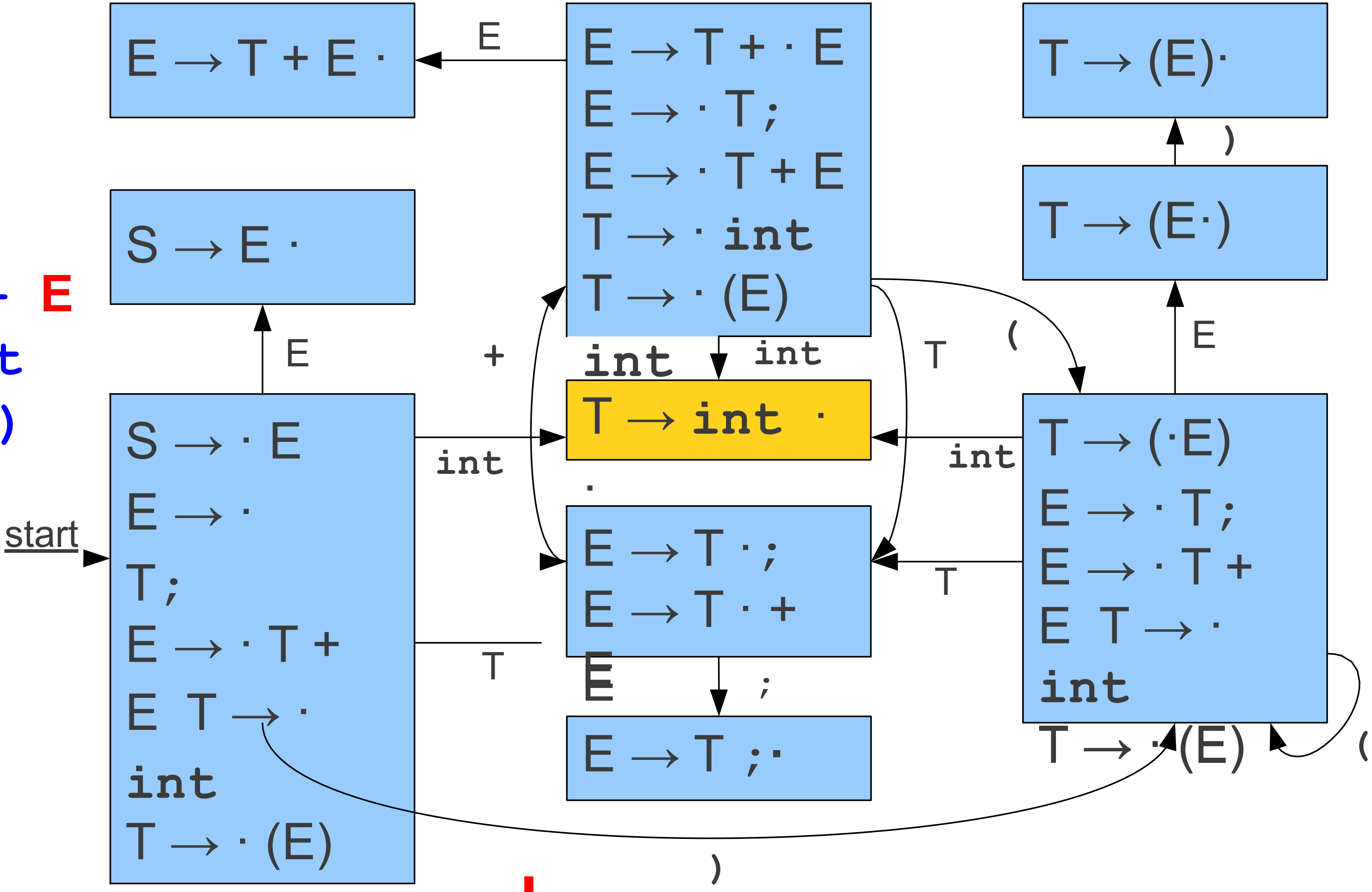
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

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

T → (E)·

S → E ·

T → (E·)

T → int ·

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

E → T · ;
E → T · +
E

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

E → T ; ·

T	+	(	T

+	int	;	)	;

LR(0) Parsing

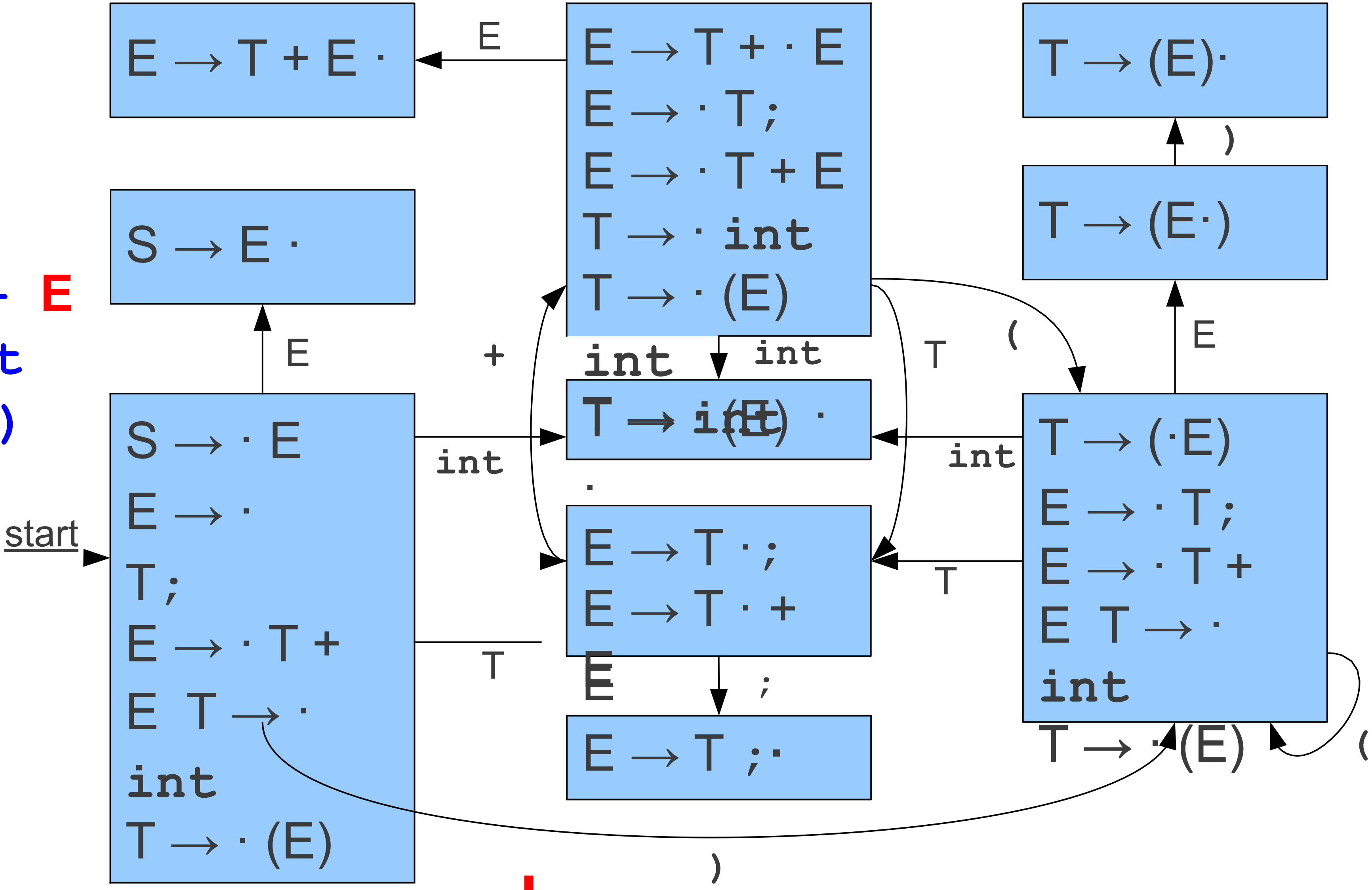
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(	T

+	int	;	)	;

LR(0) Parsing

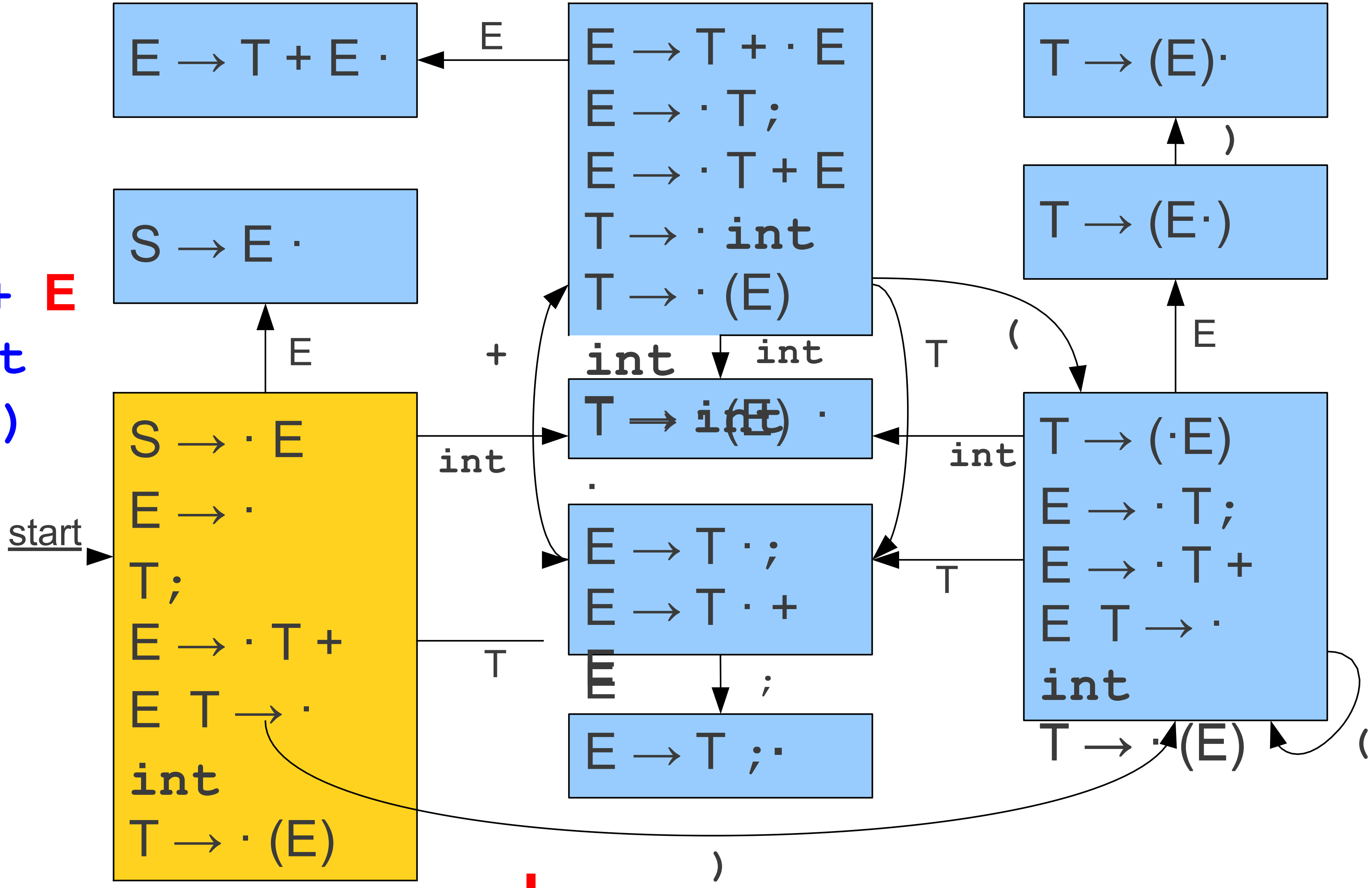
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(	T		+	int	;	)	;

LR(0) Parsing

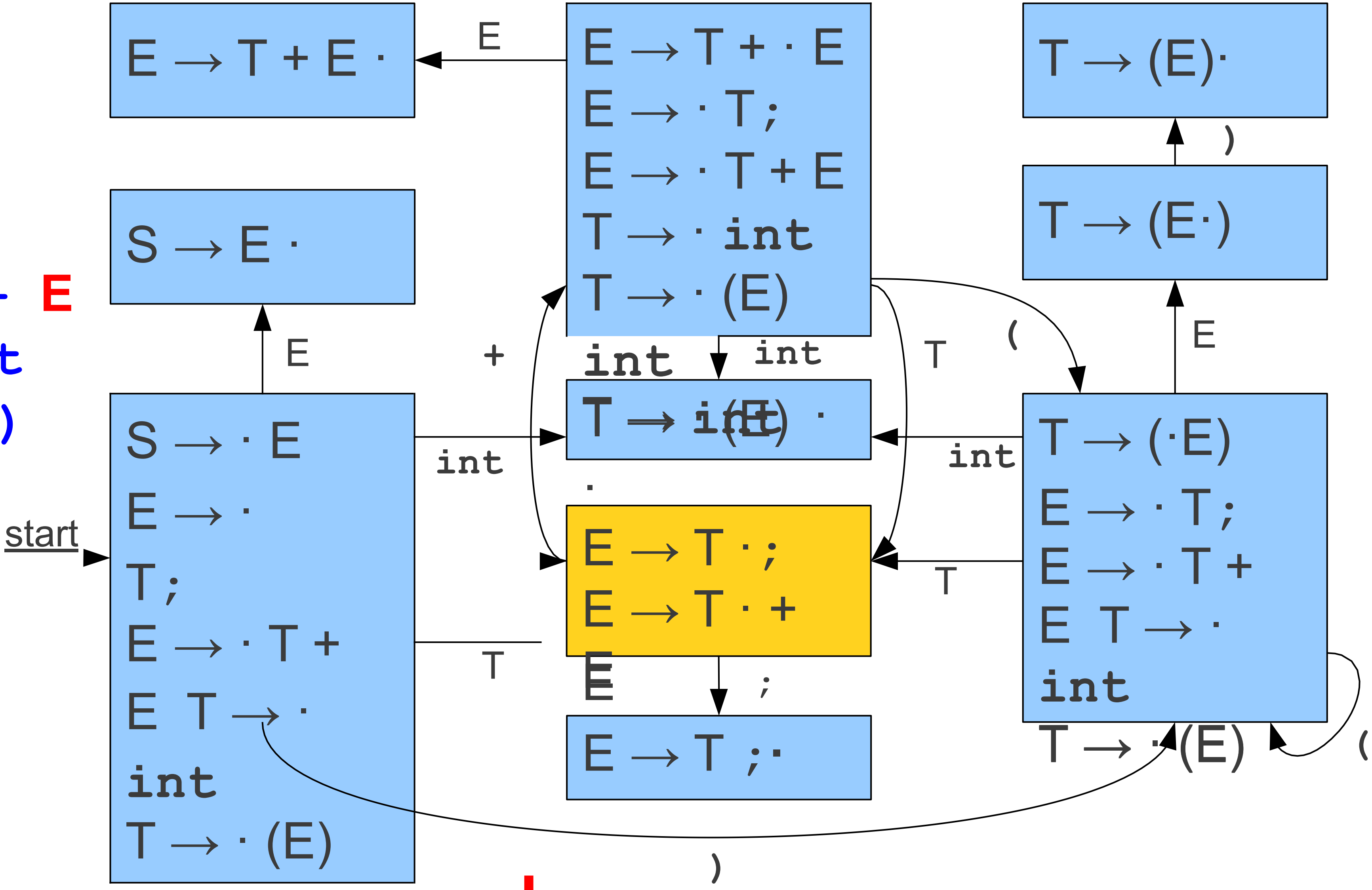
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

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

T → (E)·

S → E ·

T → (E·)

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

T → int ·

T → (E) ·

E → T · ;
E → T · + E

E → T ;·

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

T	+	(	T		+	int	;	)	;

LR(0) Parsing

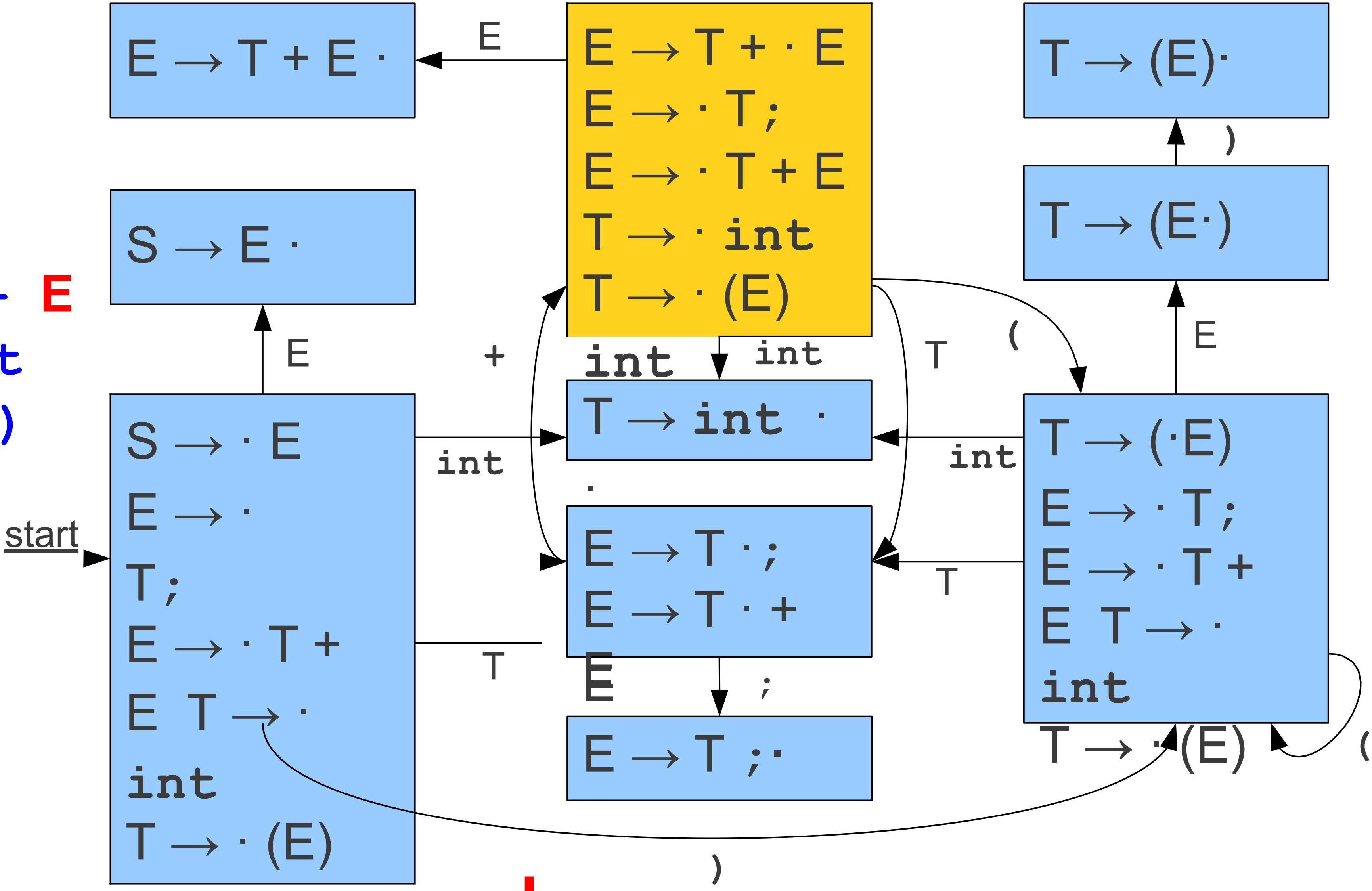
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(	T		+	int	;	)	;

LR(0) Parsing

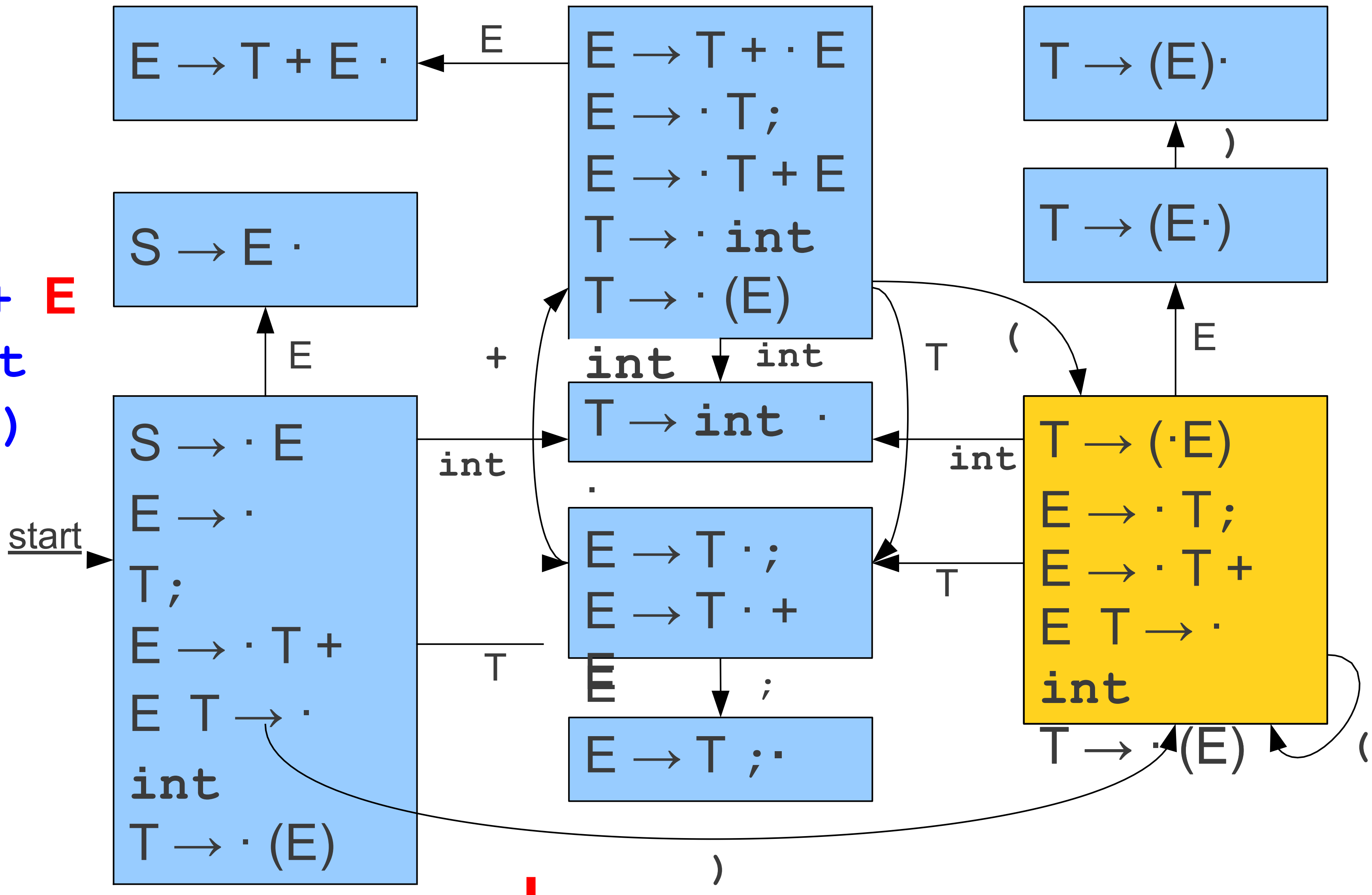
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

E

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

T → (E)·

)

T → (E·)

S → E ·

E

+

int

int

T

(

E

start

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

int

T → int ·

int

·

E → T · ;
E → T · +
E

T

T

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

(

;

E → T ;·

)

T	+	(	T

+	int	;	)	;

LR(0) Parsing

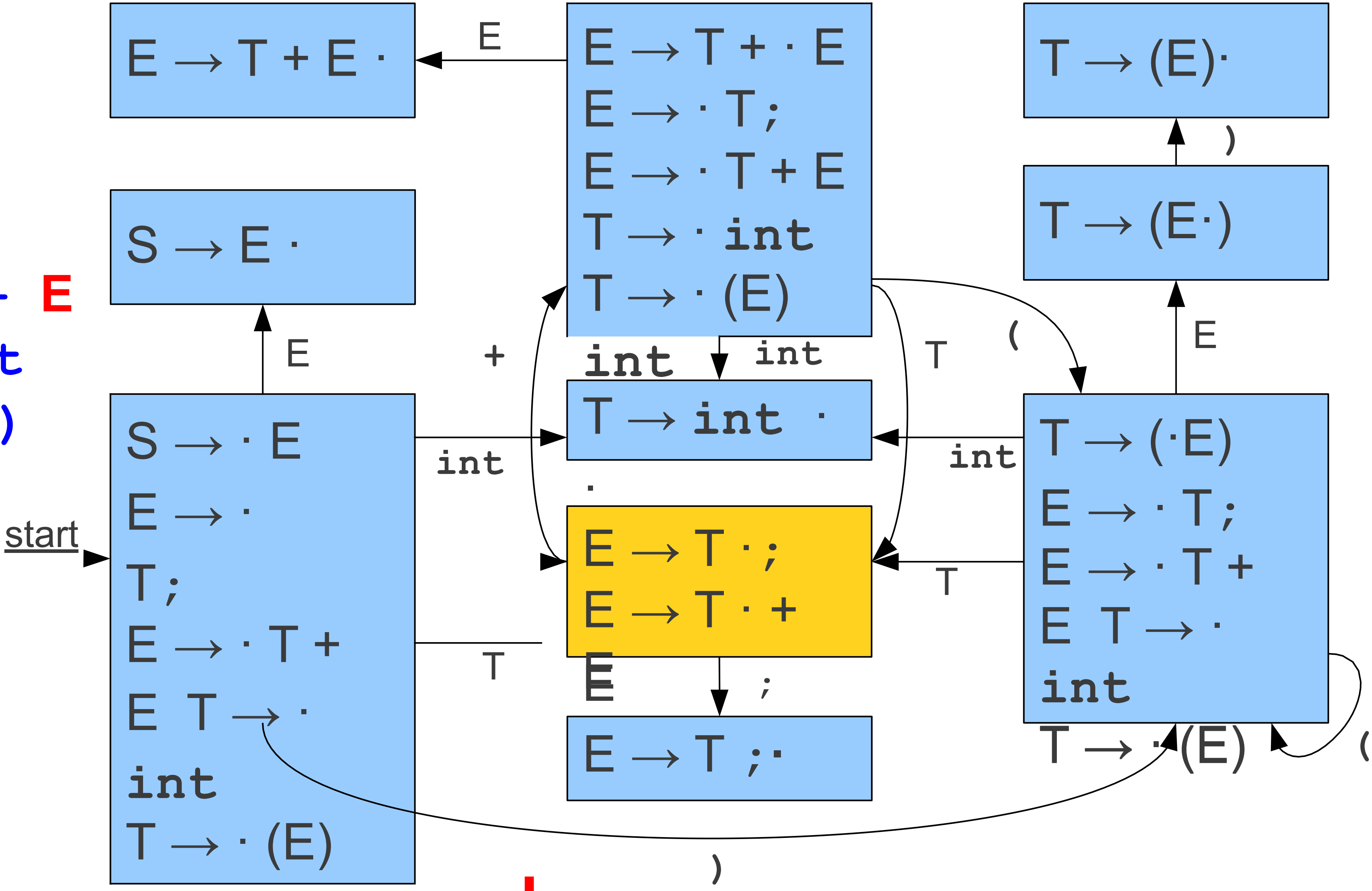
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

S → E ·

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

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

T → int ·

E → T · ;
E → T · +
E

E → T ; ·

T → (E)·

T → (E·)

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

T	+	(	T

+	int	;	)	;

LR(0) Parsing

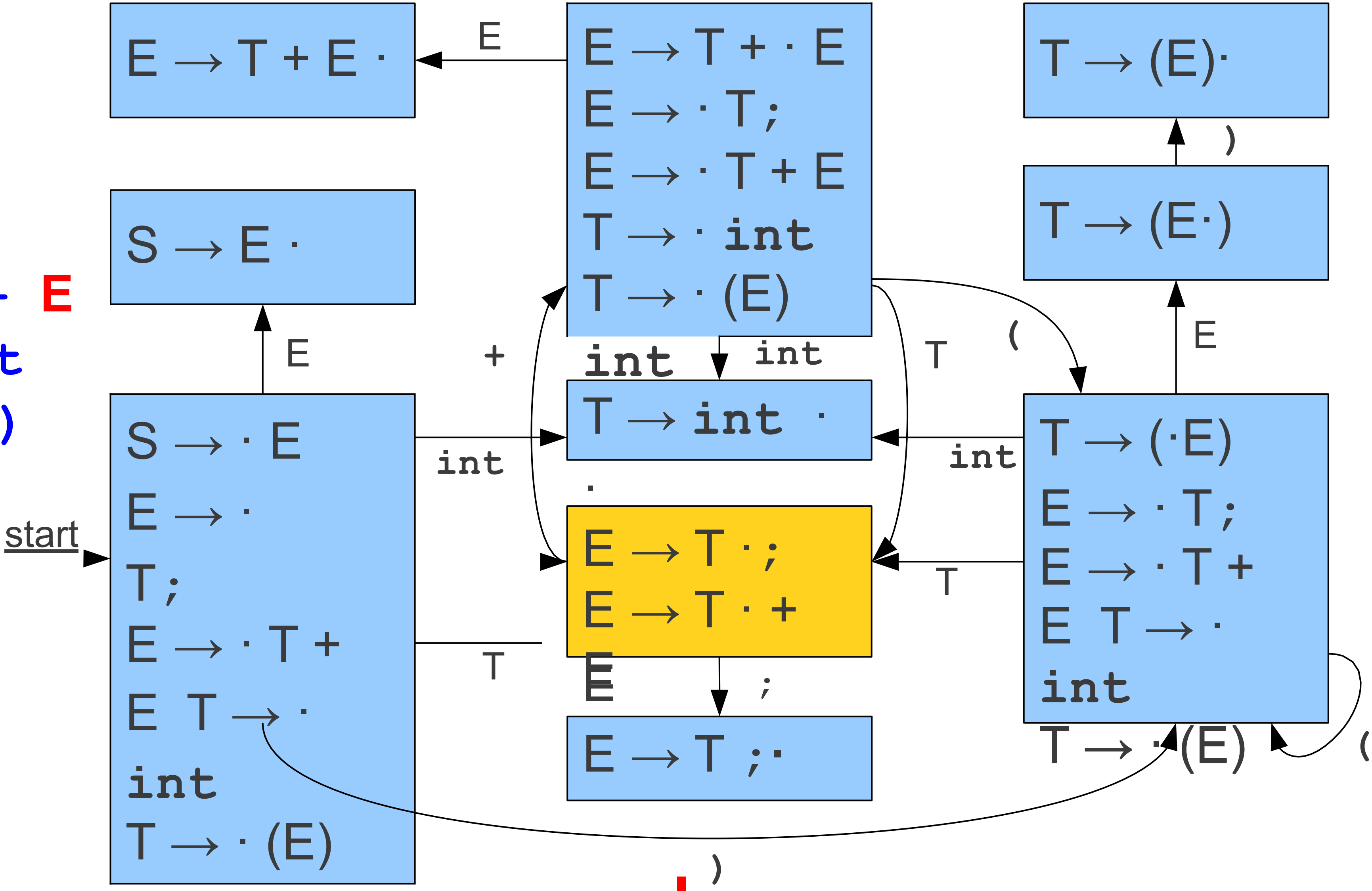
S → E

E → T;

E → T + E

T → int

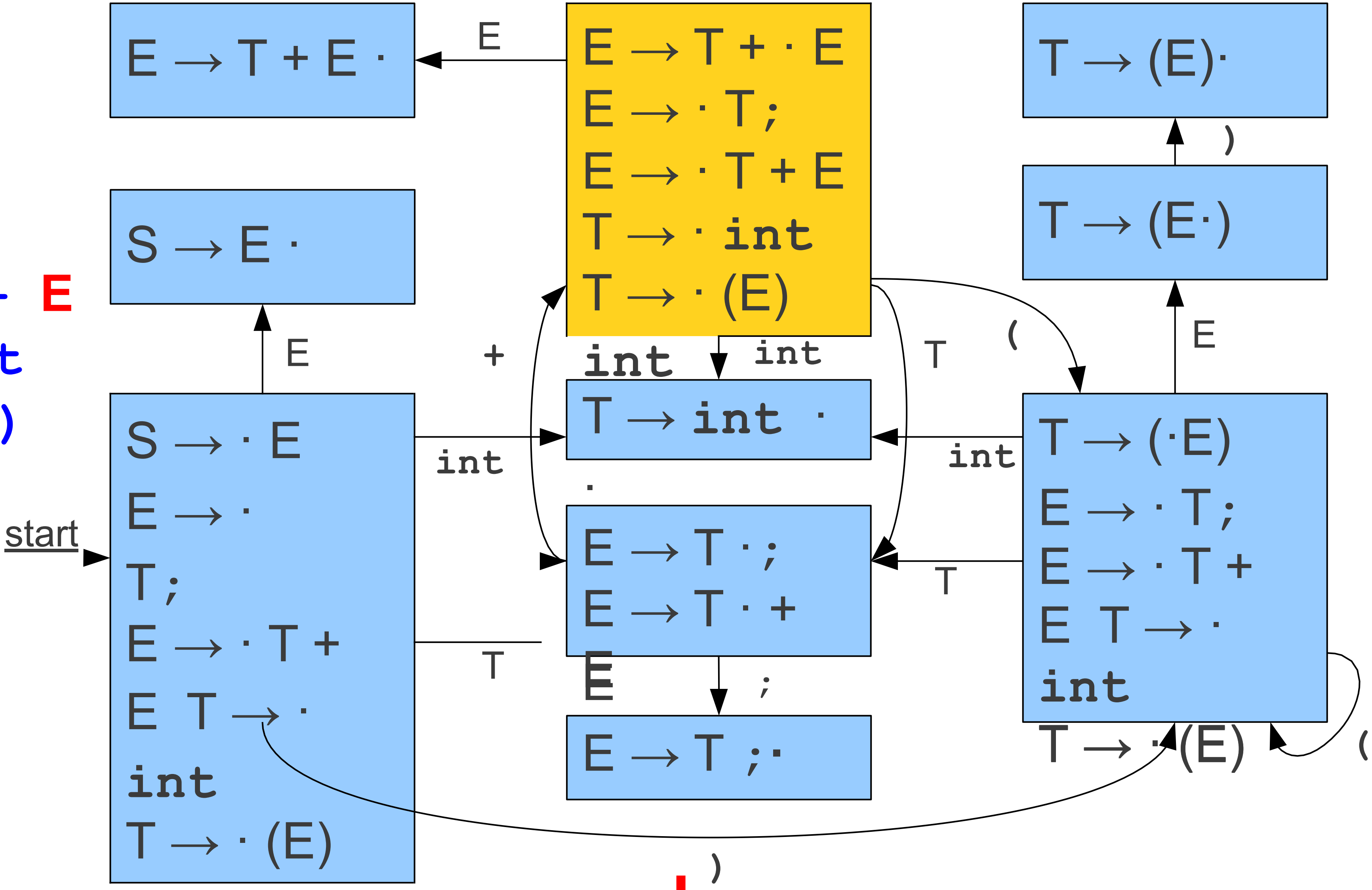
T → (E)



T	+	(	T	+	int	;	)	;

LR(0) Parsing

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



T	+	(	T	+

int	;	)	;

LR(0) Parsing

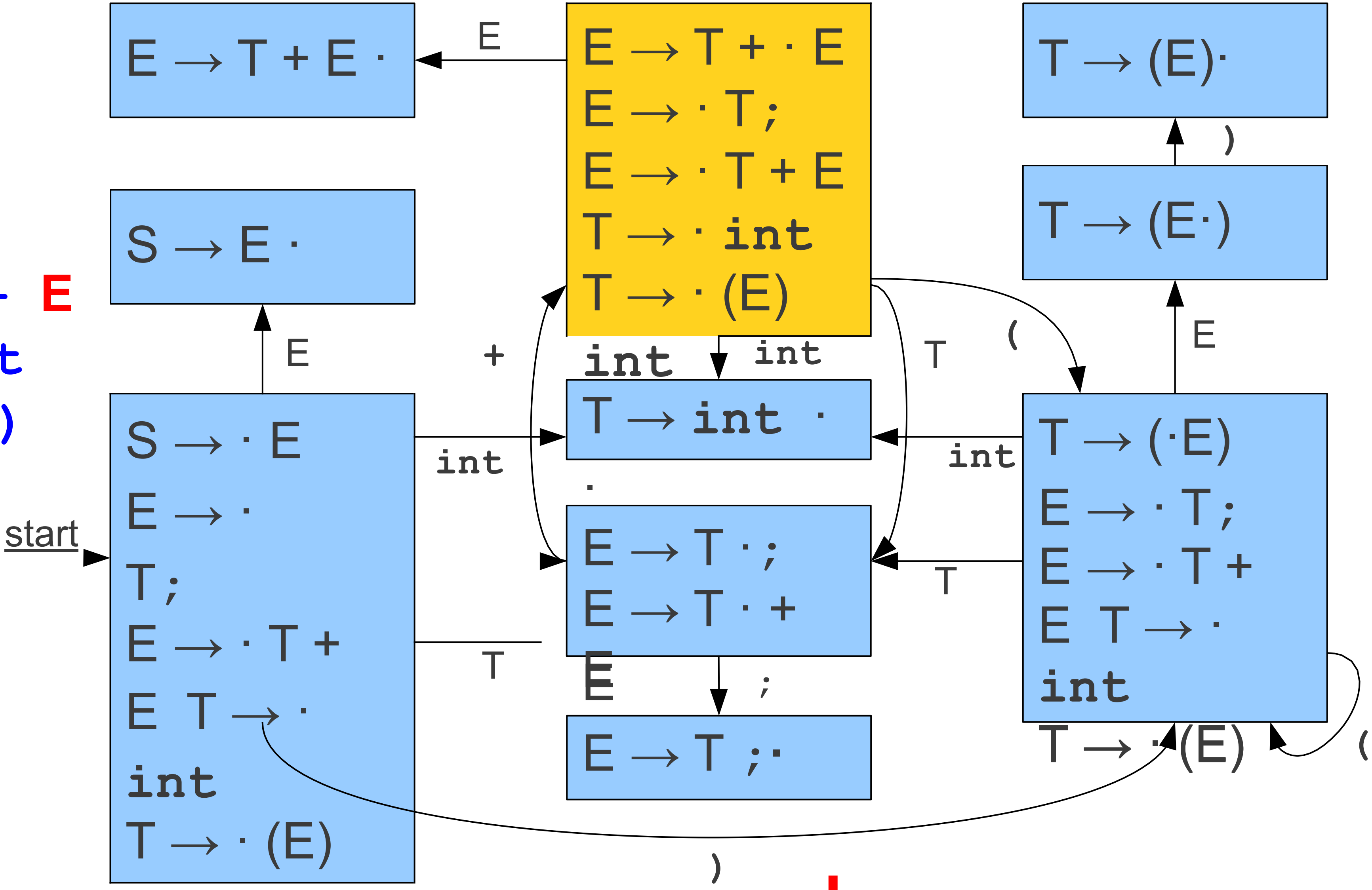
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

S → E ·

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

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

T → int ·

E → T · ;
E → T · +
E

E → T ; ·

T → (E)·

T → (E·)

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

T	+	(	T	+	int

;	)	;

LR(0) Parsing

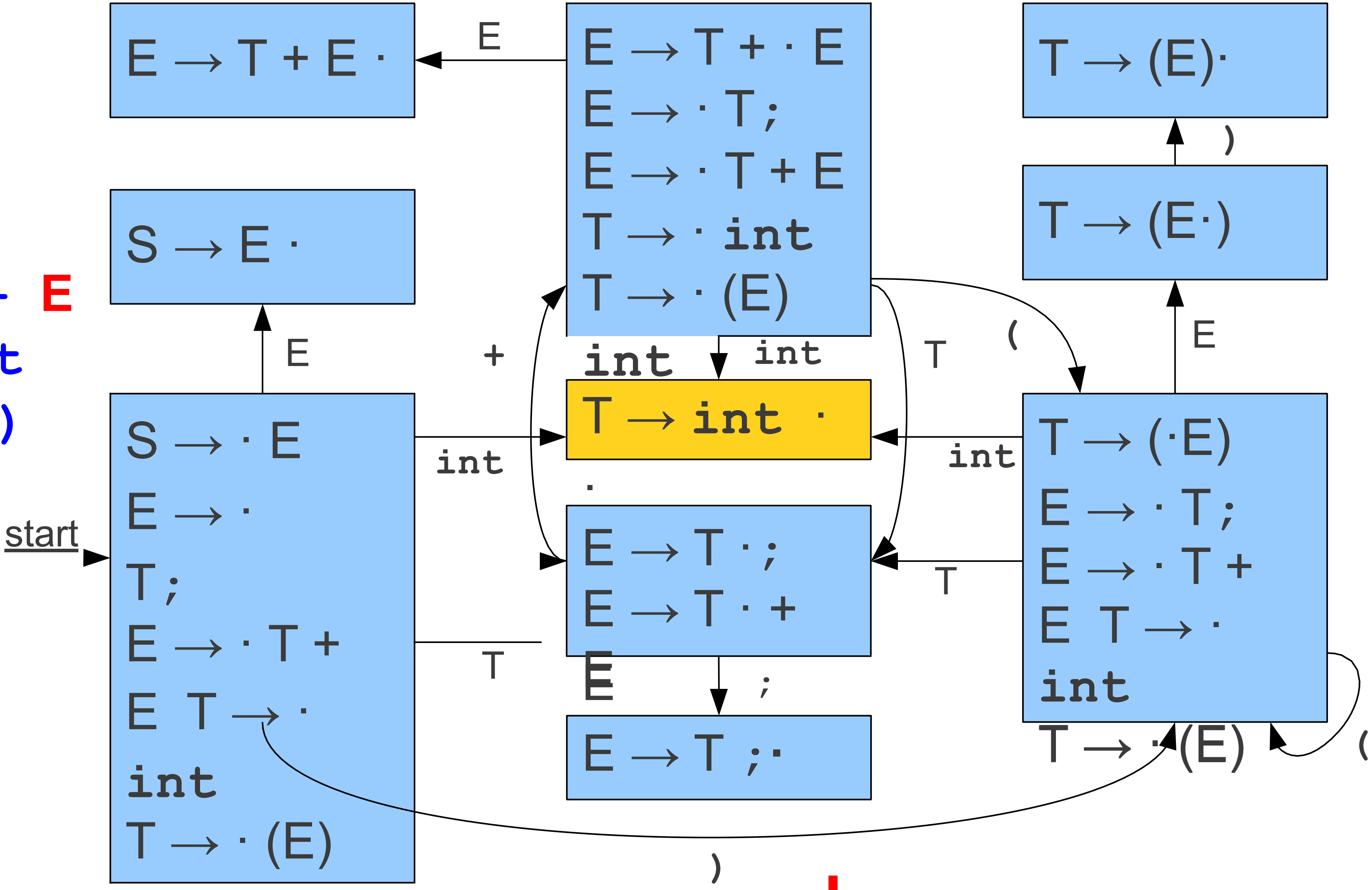
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

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

T → (E)·

S → E ·

T → (E·)

T → int ·

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

E → T · ;
E → T · +
E

E → T ;·

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

T	+	(	T	+	int	;	)	;

LR(0) Parsing

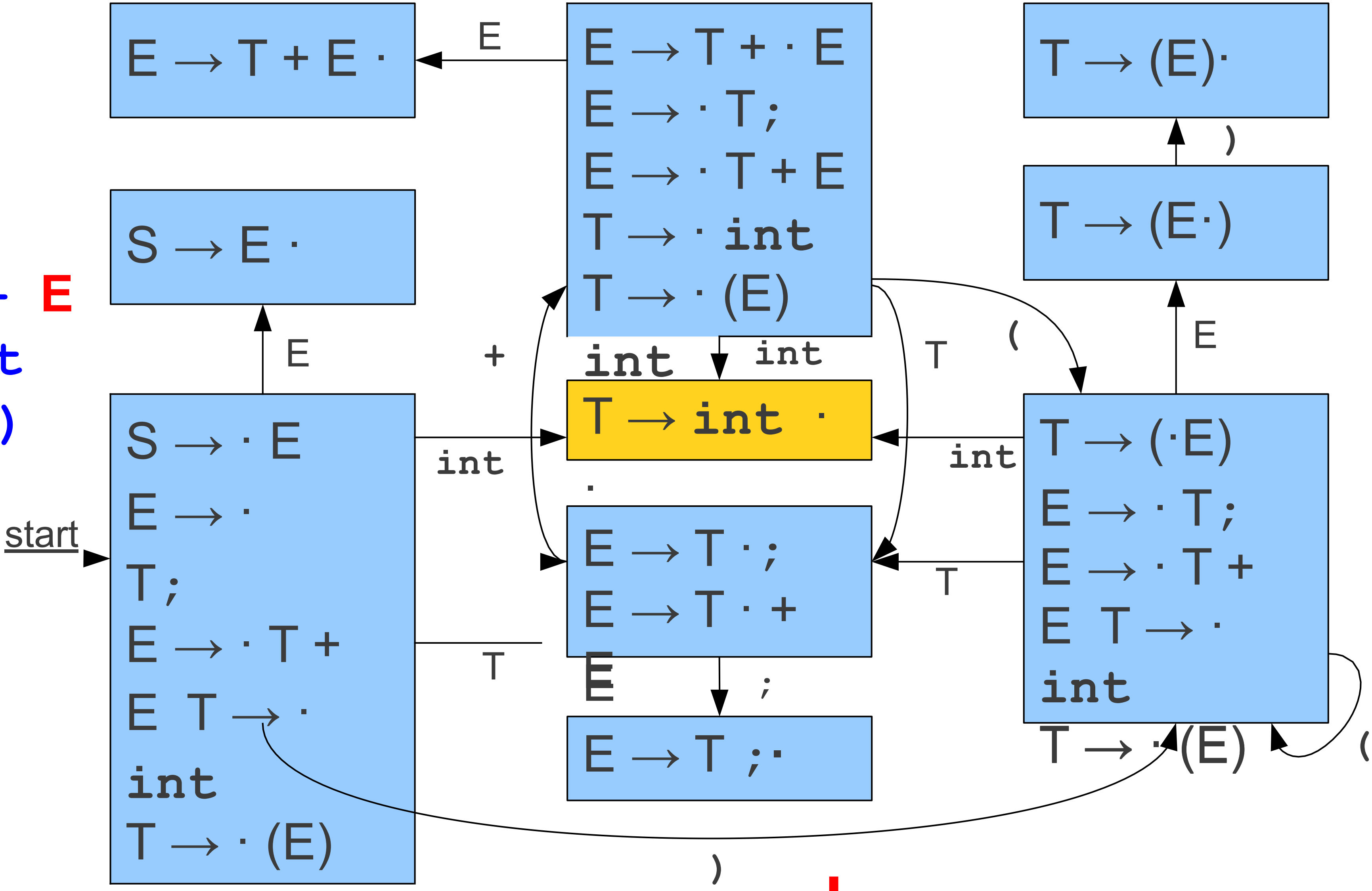
S → E

E → T;

E → T + E

T → int

T → (E)



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

E → T + E ·

S → E ·

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

T → int ·

E → T · ;
E → T · +
E

E → T ; ·

T → (E)·

T → (E·)

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

T	+	(	T	+	|	;	)	;

LR(0) Parsing

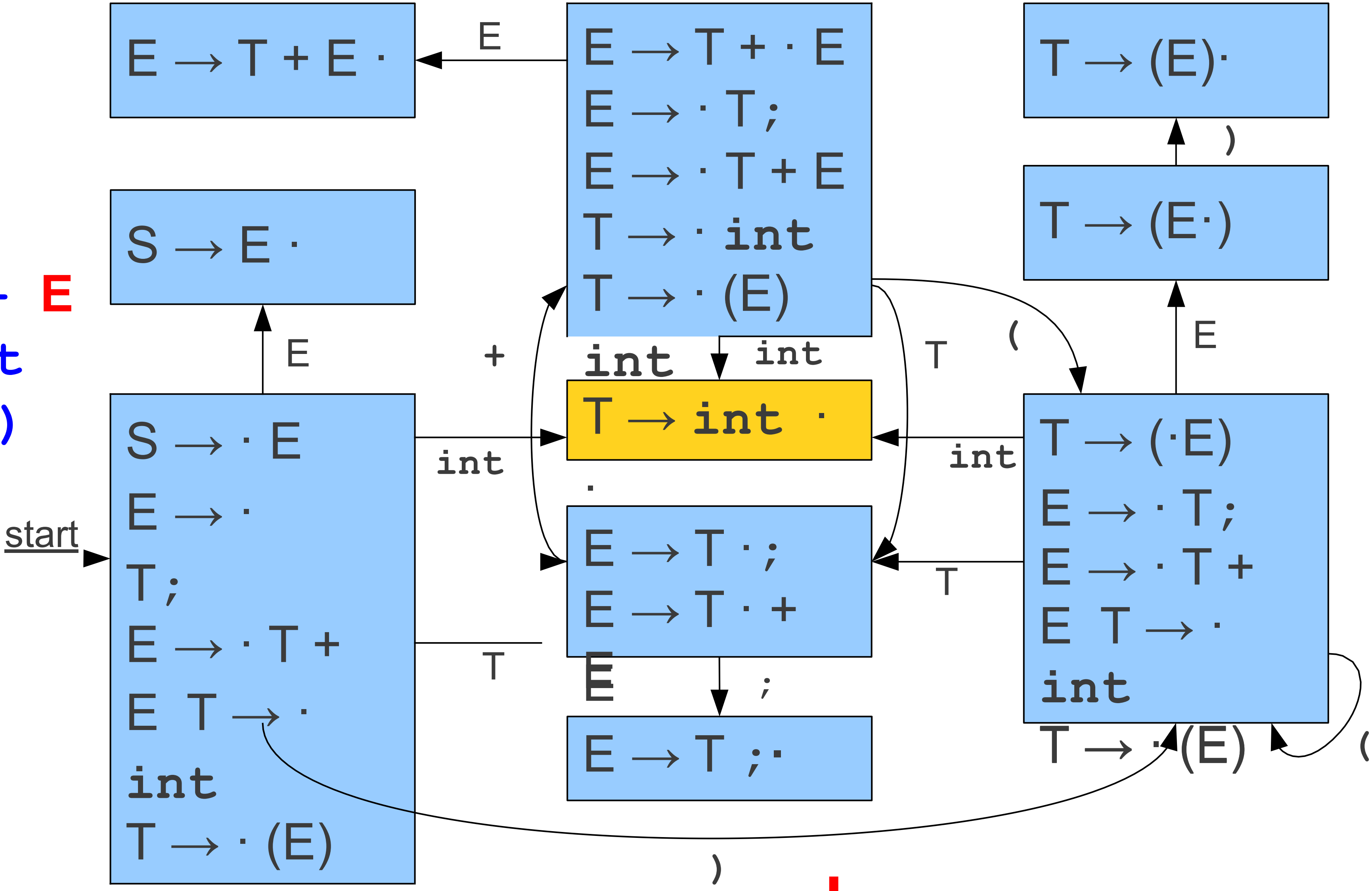
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(	T	+	T		;	)	;

LR(0) Parsing

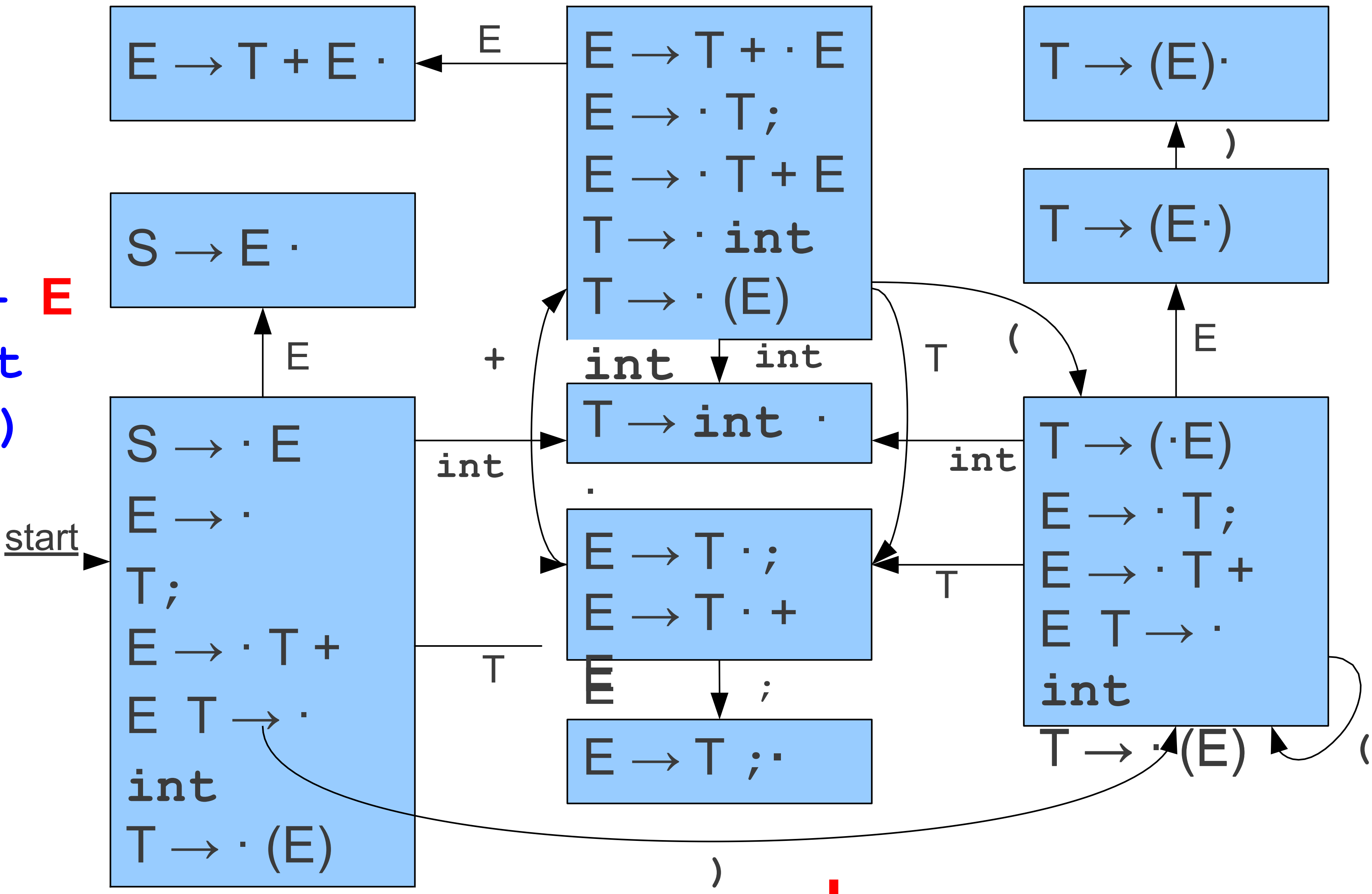
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(	T	+	T		;	)	;

An Optimization

- Rather than restart the automaton on each reduction, remember **what state we were in** for each symbol.
- When applying a reduction, restart the automaton from the last known good state.

LR(0) Parsing

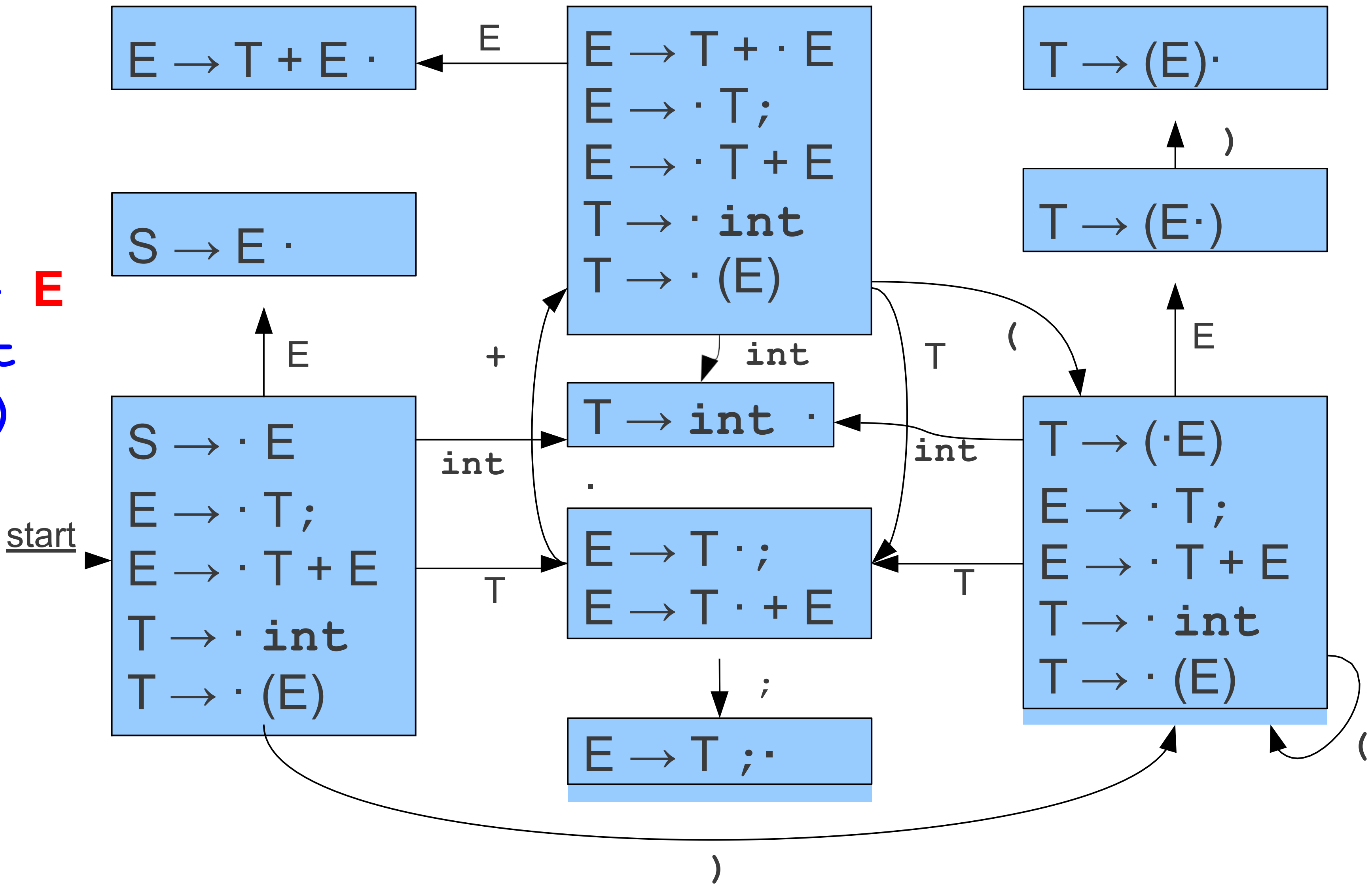
S → E

E → T;

E → T + E

T → int

T → (E)



int	+	(	int	+	int	;	)	;

LR(0) Parsing

S → E

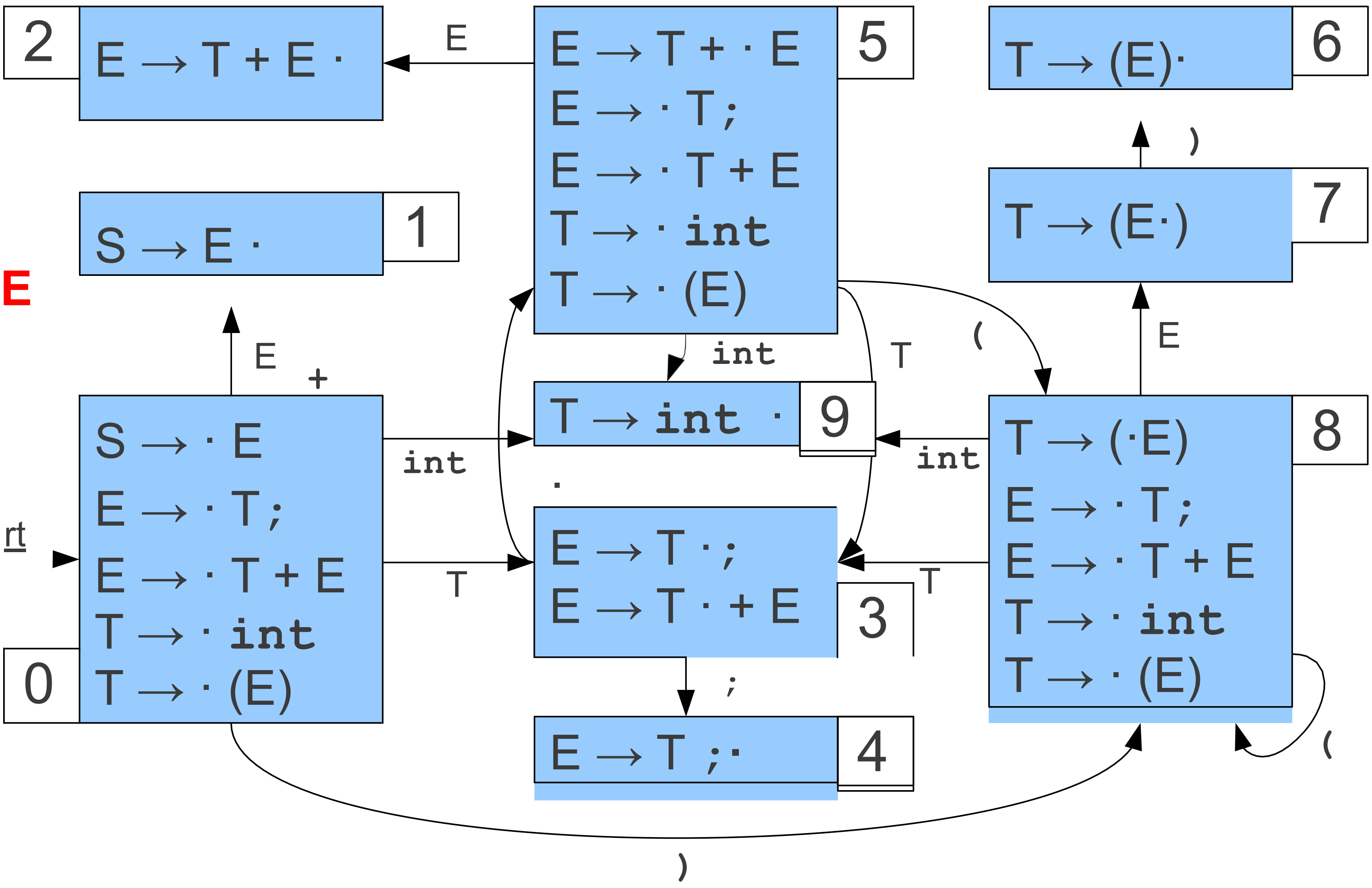
E → T;

E → T + E

T → int

T → (E)

sta



2: E → T + E ·

1: S → E ·

0:
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

5:
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

9: T → int ·

3:
E → T · ;
E → T · + E

4: E → T ;·

6: T → (E)·

7: T → (E·)

8:
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

int	+	(	int	+	int	;	)	;

LR(0) Parsing

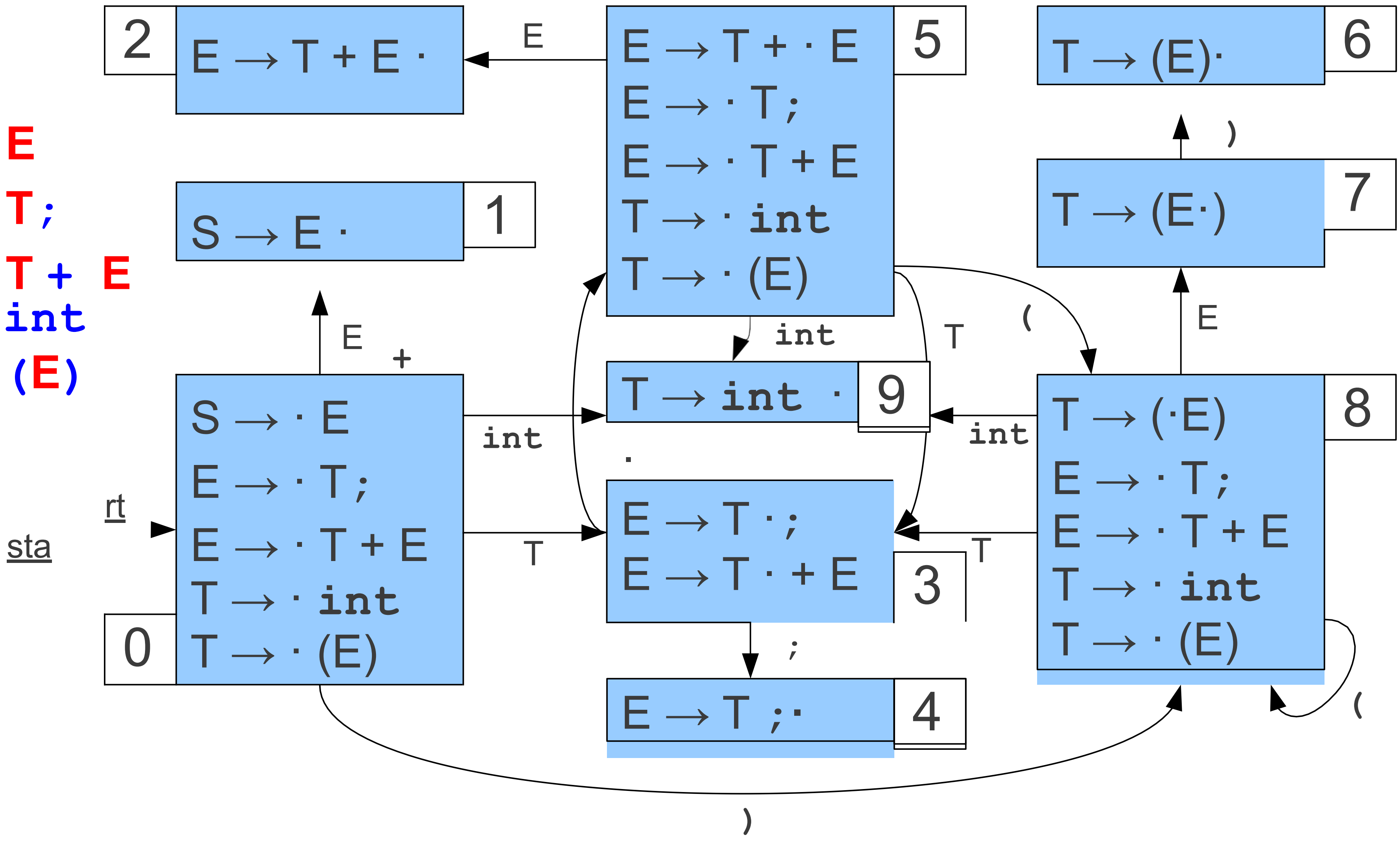
S → E

E → T ;

E → T + E

T → int

T → (E)



0: S → · E; E → · T ; ; E → · T + E; T → · int; T → · (E)

1: S → E ·

2: E → T + E ·

3: E → T · ; ; E → T · + E

4: E → T ; ·

5: E → T + · E; E → · T ; ; E → · T + E; T → · int; T → · (E)

6: T → (E)·

7: T → (E·)

8: T → (·E); E → · T ; ; E → · T + E; T → · int; T → · (E)

9: T → int ·

int + (int + int ;) ;

0

LR(0) Parsing

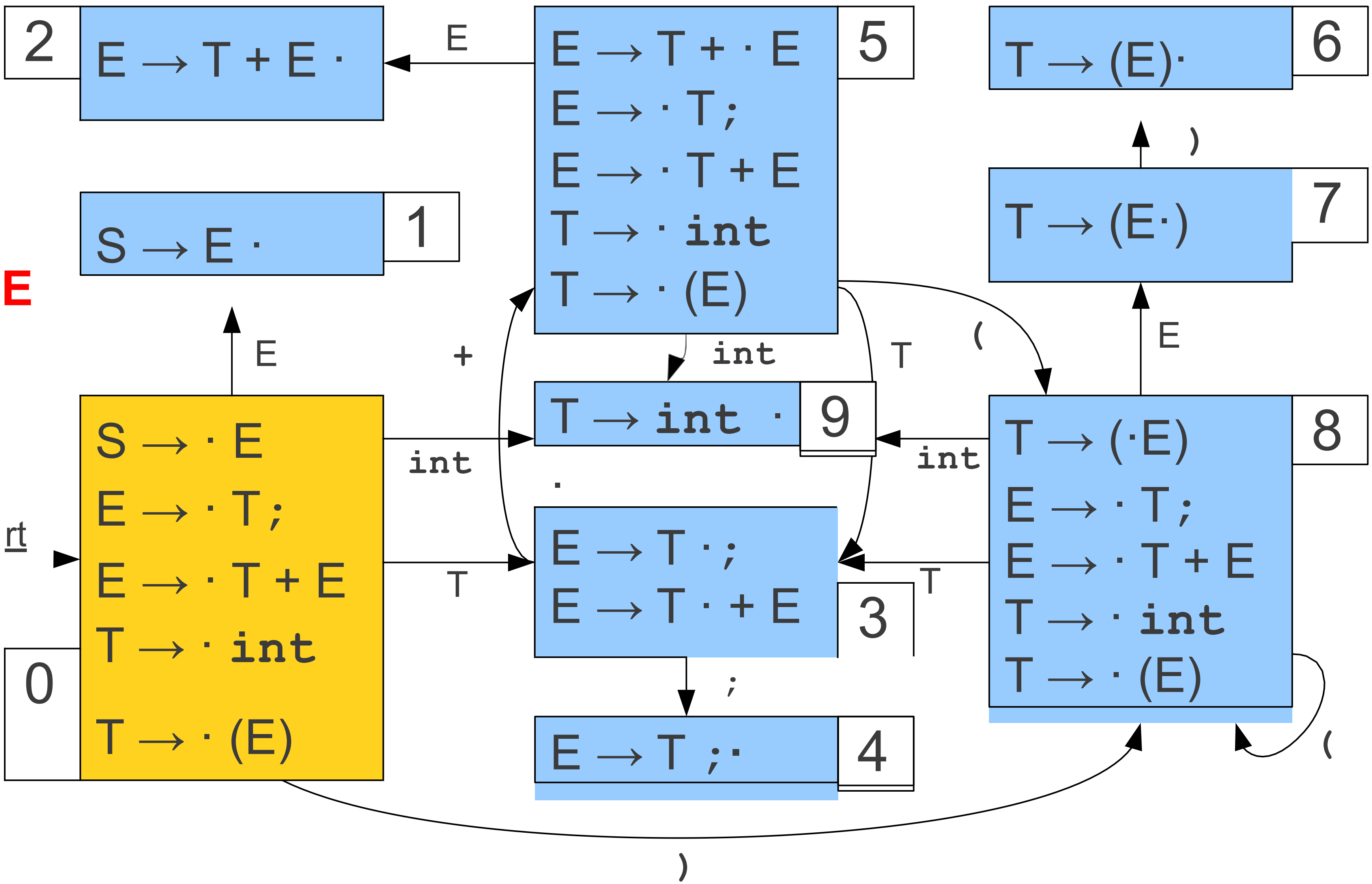
S → E

E → T;

E → T + E

T → int

T → (E)



int	+	(	int	+	int	;	)	;

0

LR(0) Parsing

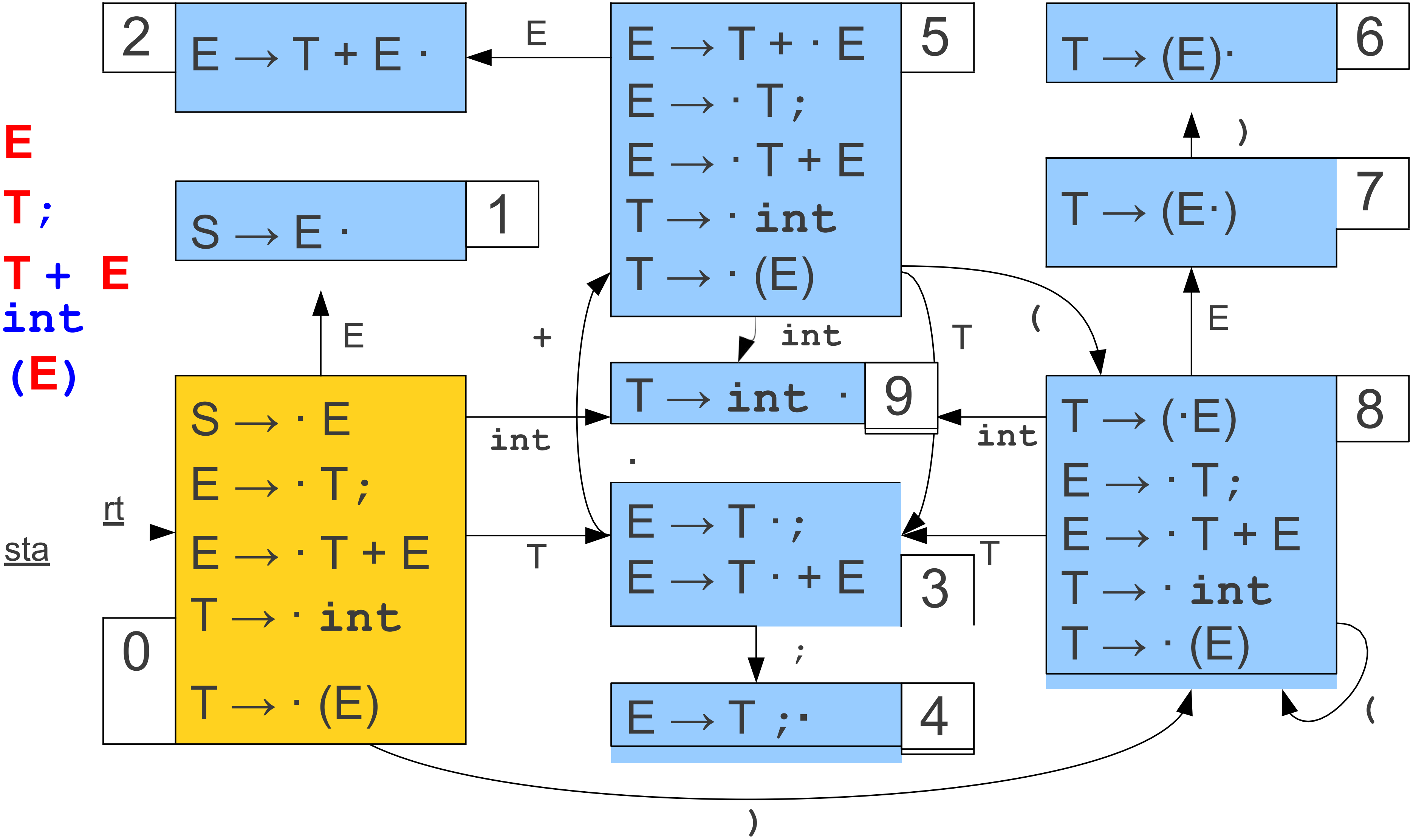
S → E

E → T;

E → T + E

T → int

T → (E)



int		+	(	int	+	int	;	)	;

0

LR(0) Parsing

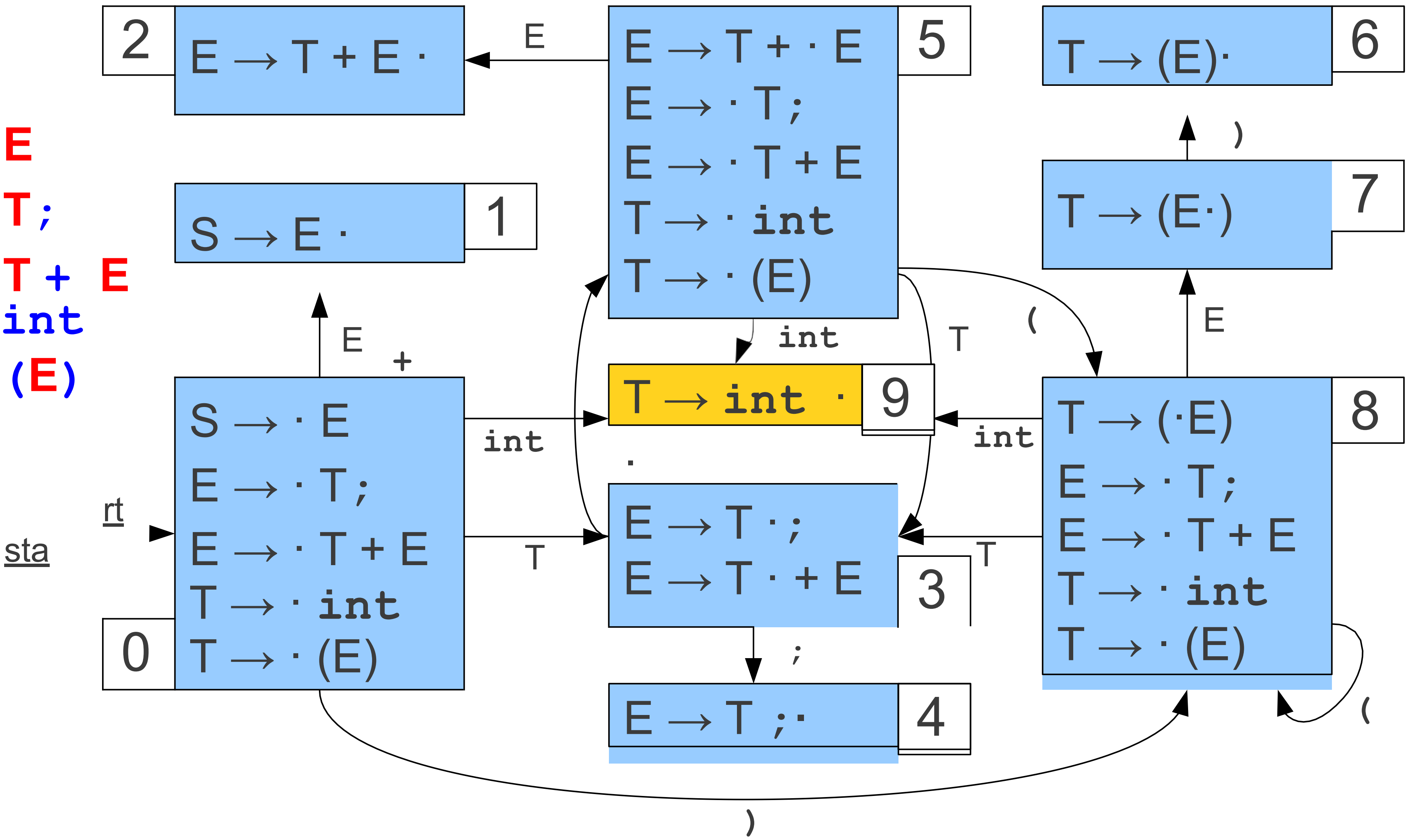
S → E

E → T;

E → T + E

T → int

T → (E)



2: E → T + E ·

1: S → E ·

0:
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

5:
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

9: T → int ·

3:
E → T · ;
E → T · + E

4: E → T ;·

6: T → (E)·

7: T → (E·)

8:
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

int	|	+	(	int	+	int	;	)	;

0

LR(0) Parsing

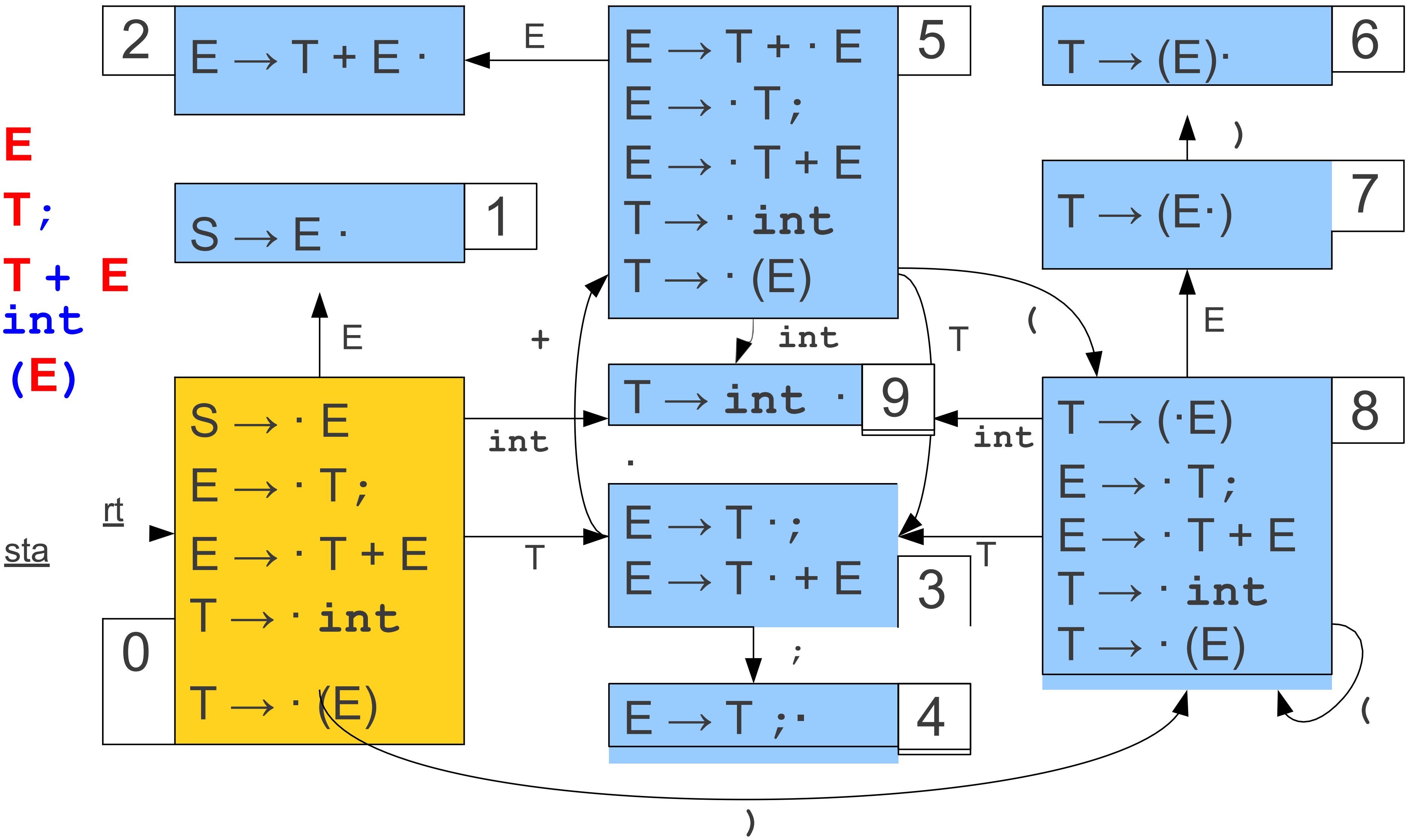
S → E

E → T;

E → T + E

T → int

T → (E)



+	(	int	+	int	;	)	;

0

LR(0) Parsing

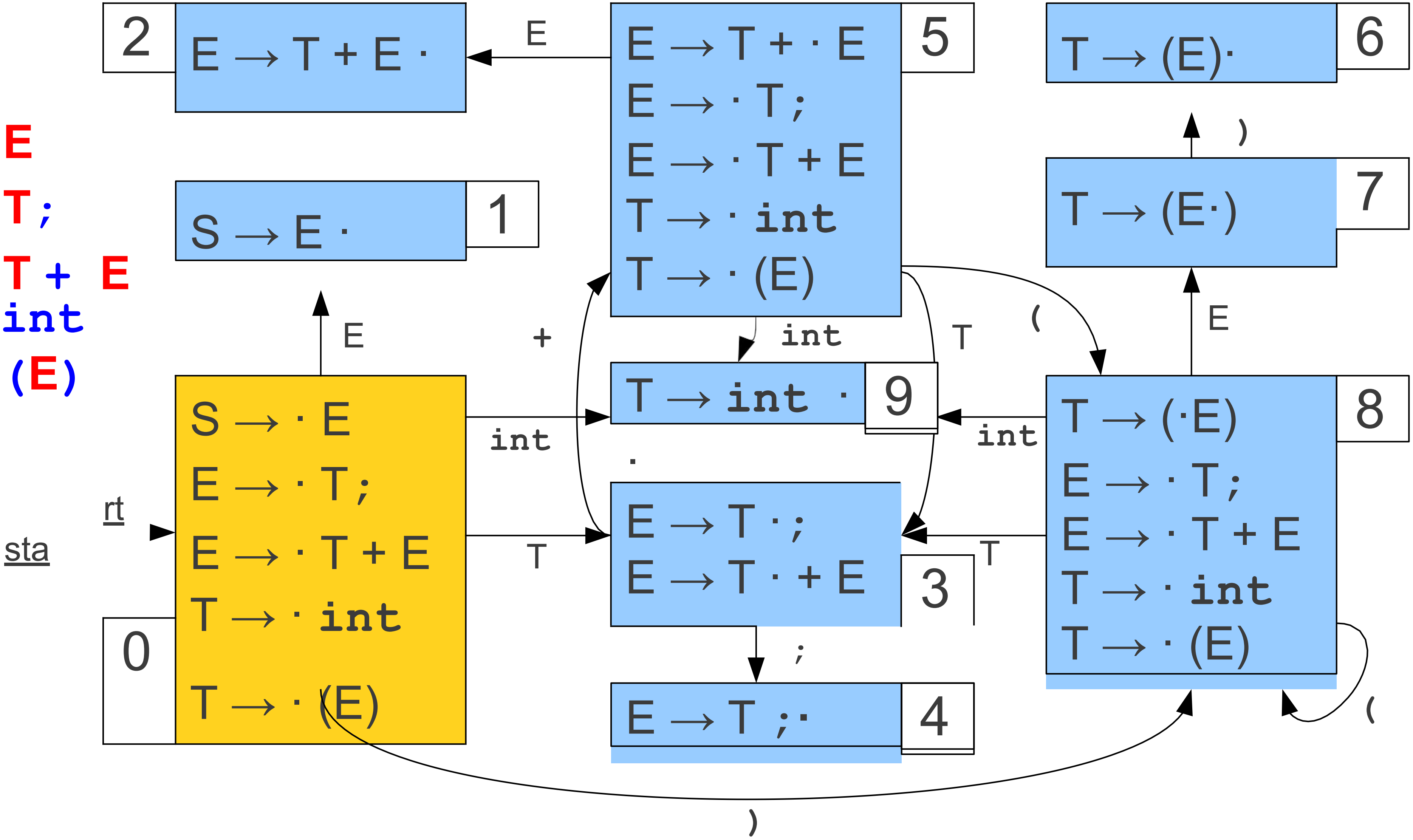
S → E

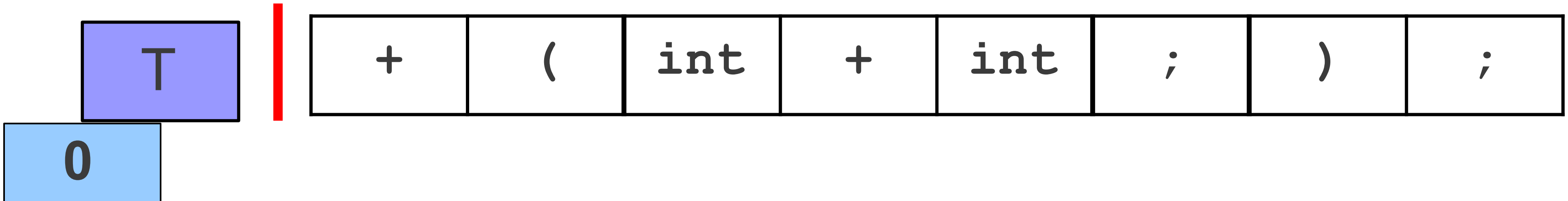
E → T;

E → T + E

T → int

T → (E)





LR(0) Parsing

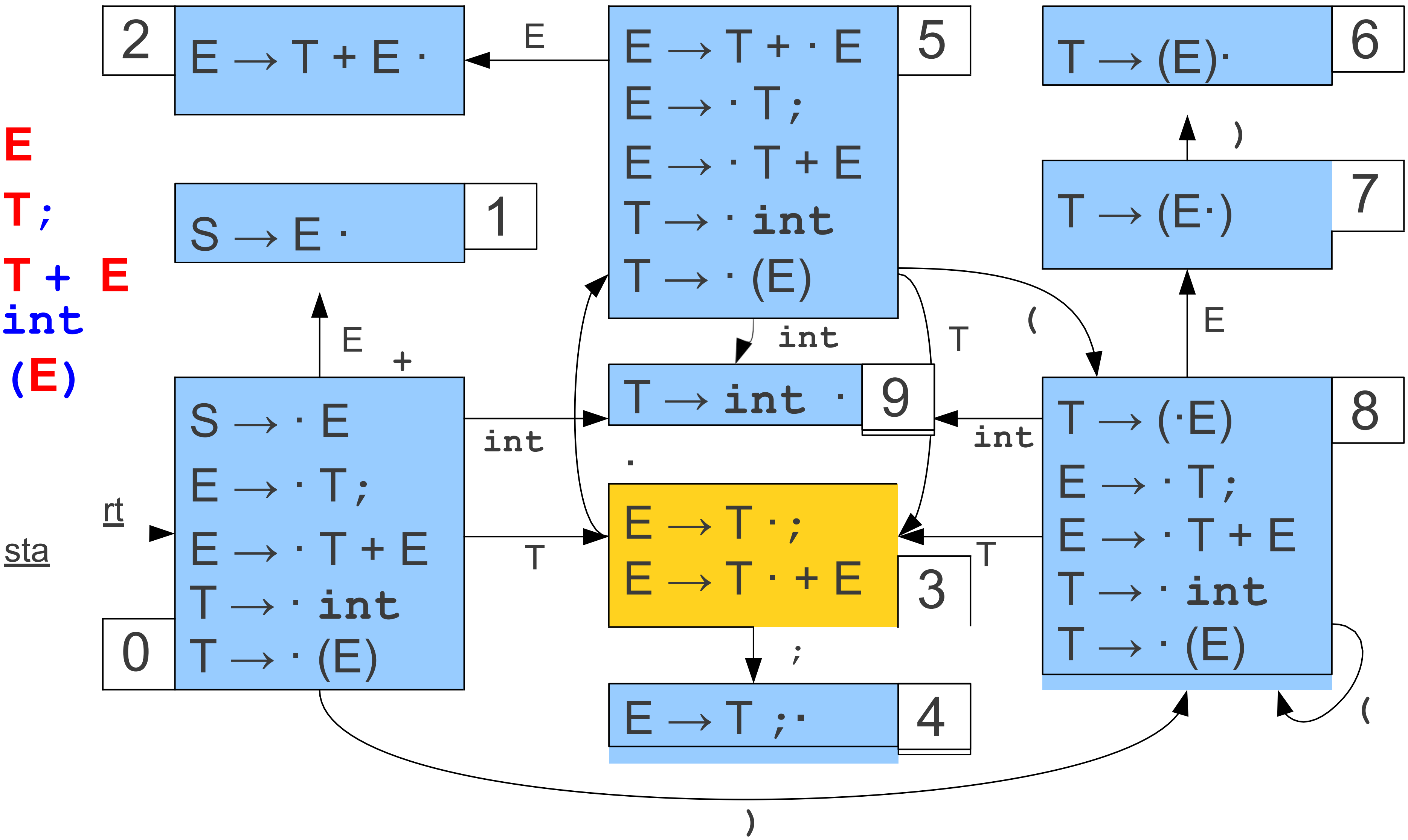
S → E

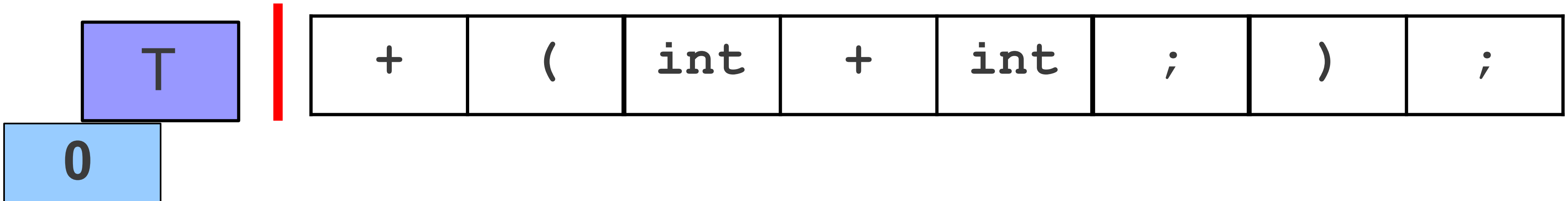
E → T;

E → T + E

T → int

T → (E)





LR(0) Parsing

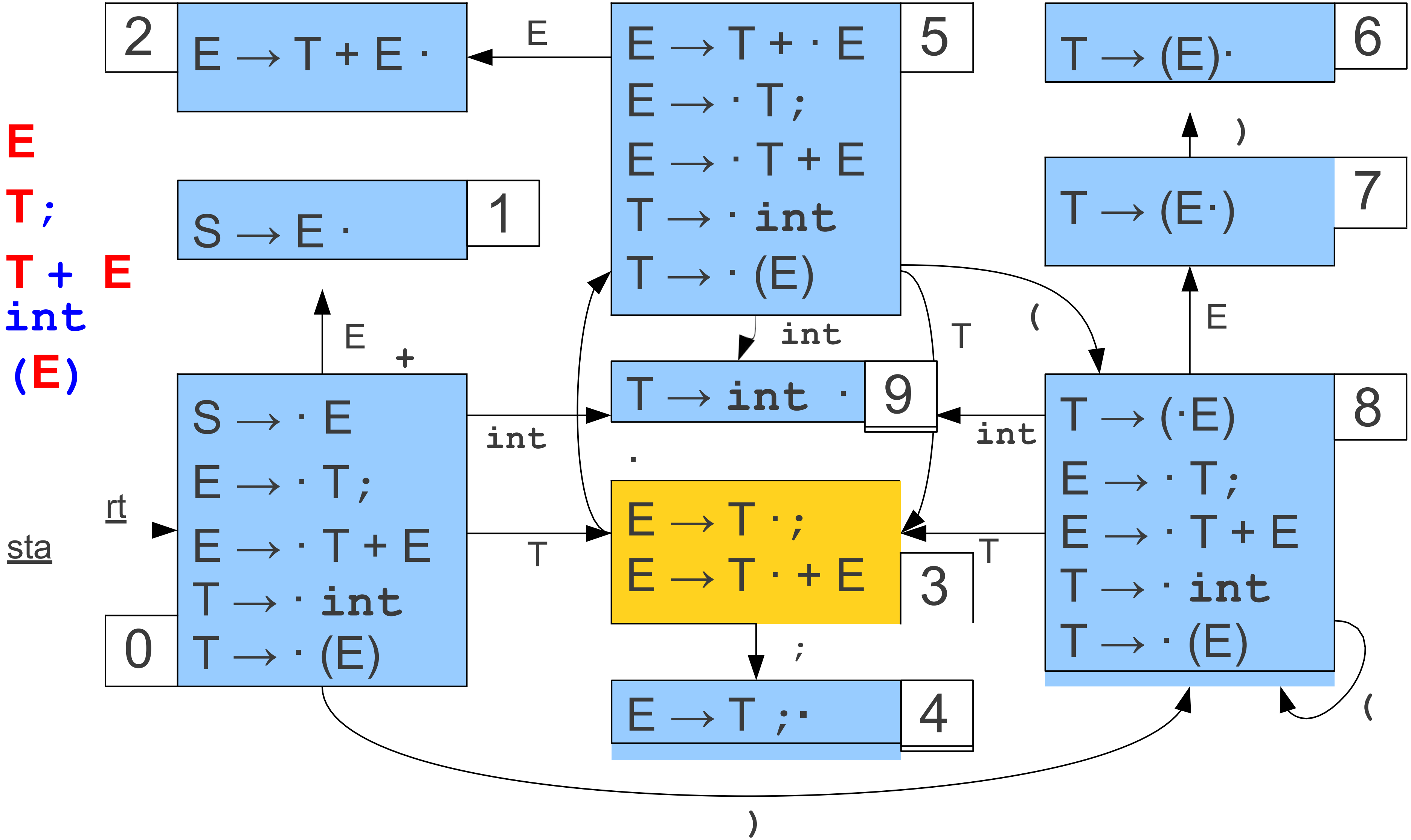
S → E

E → T;

E → T + E

T → int

T → (E)



2: E → T + E ·

5: E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6: T → (E)·

1: S → E ·

7: T → (E·)

9: T → int ·

0: S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

3: E → T · ;
E → T · + E

8: T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

4: E → T ;·

T		+	(	int	+	int	;	)	;
0	3								

LR(0) Parsing

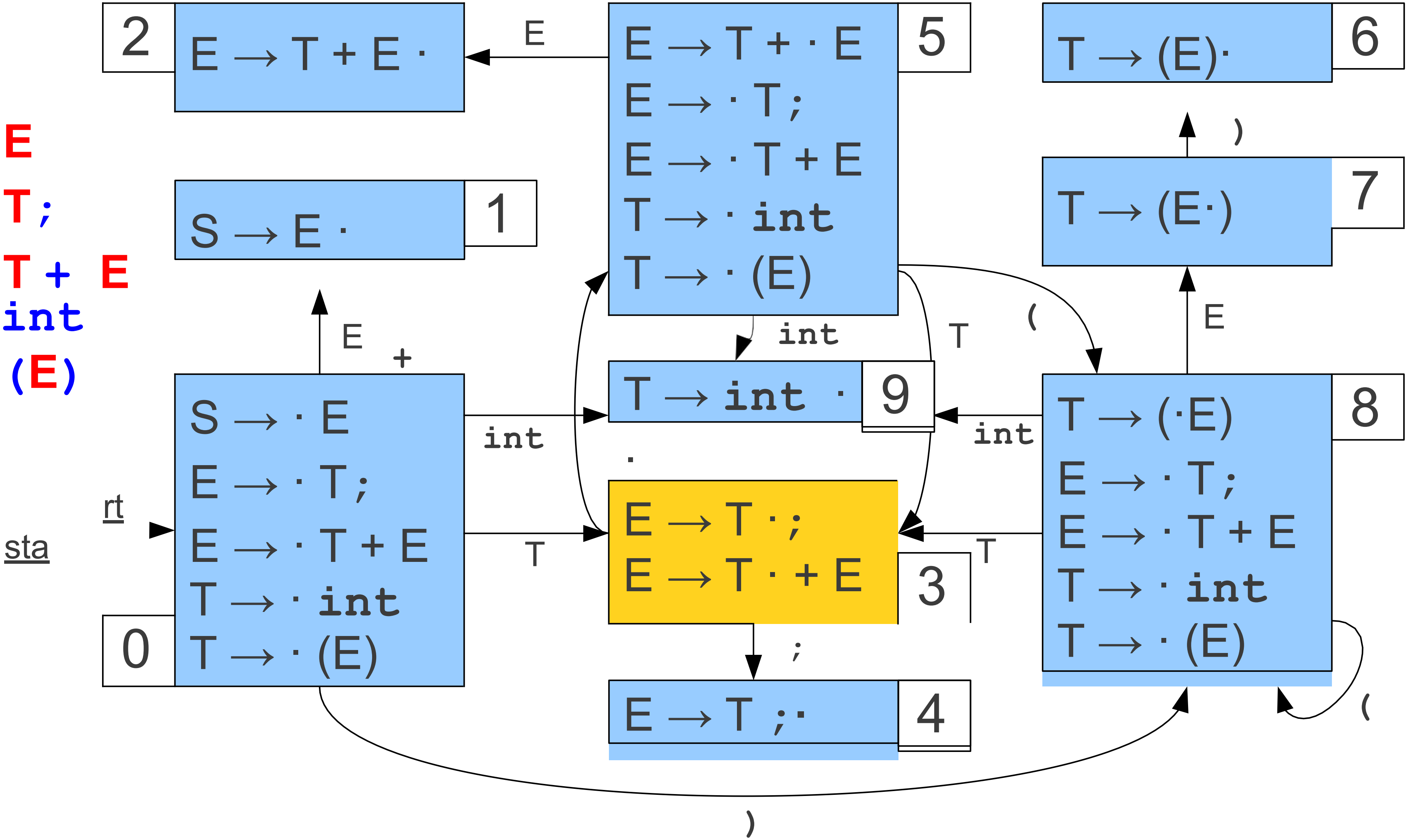
S → E

E → T;

E → T + E

T → int

T → (E)



2: E → T + E ·

5: E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6: T → (E)·

1: S → E ·

7: T → (E·)

9: T → int ·

0: S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

3: E → T · ;
E → T · + E

8: T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

4: E → T ;·

T	+
0	3

(	int	+	int	;	)	;

LR(0) Parsing

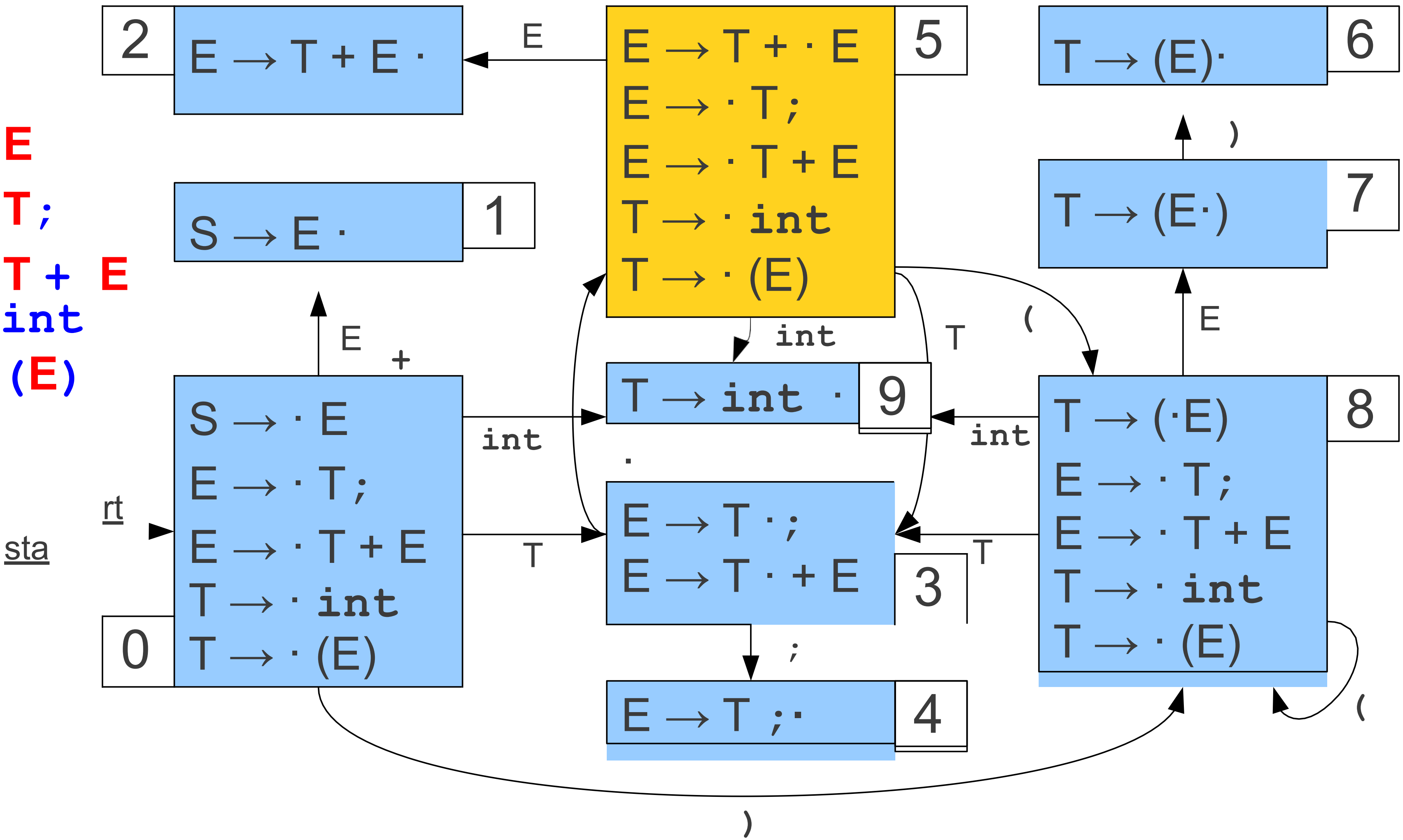
S → E

E → T;

E → T + E

T → int

T → (E)



2: E → T + E ·

1: S → E ·

5:
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6: T → (E)·

7: T → (E·)

0:
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

9: T → int ·

3:
E → T · ;
E → T · + E

4: E → T ;·

8:
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

T	+
0	3

(	int	+	int	;	)	;

LR(0) Parsing

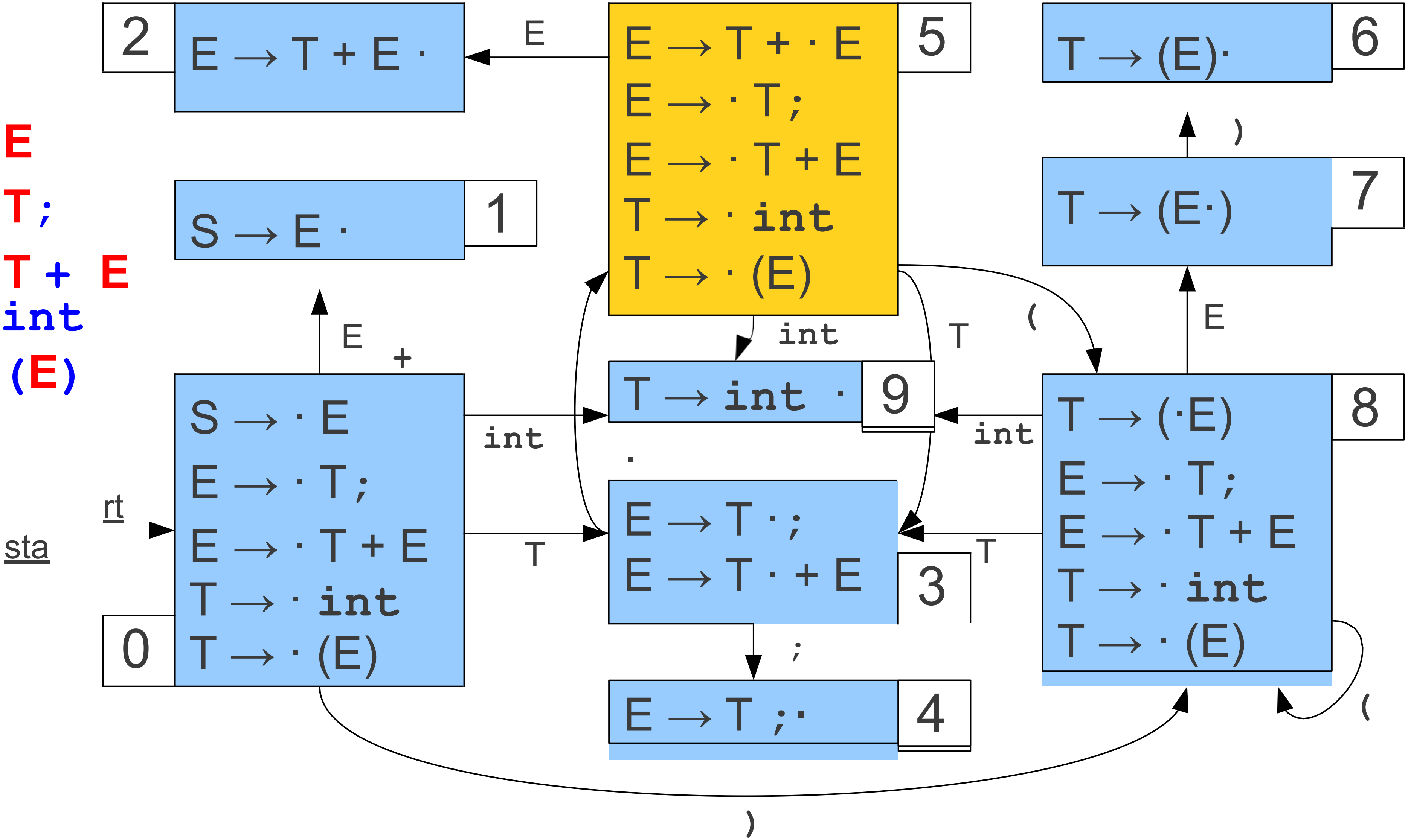
S → E

E → T;

E → T + E

T → int

T → (E)



2: E → T + E ·

1: S → E ·

0:
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

5:
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

9: T → int ·

3:
E → T · ;
E → T · + E

4: E → T ; ·

6: T → (E)·

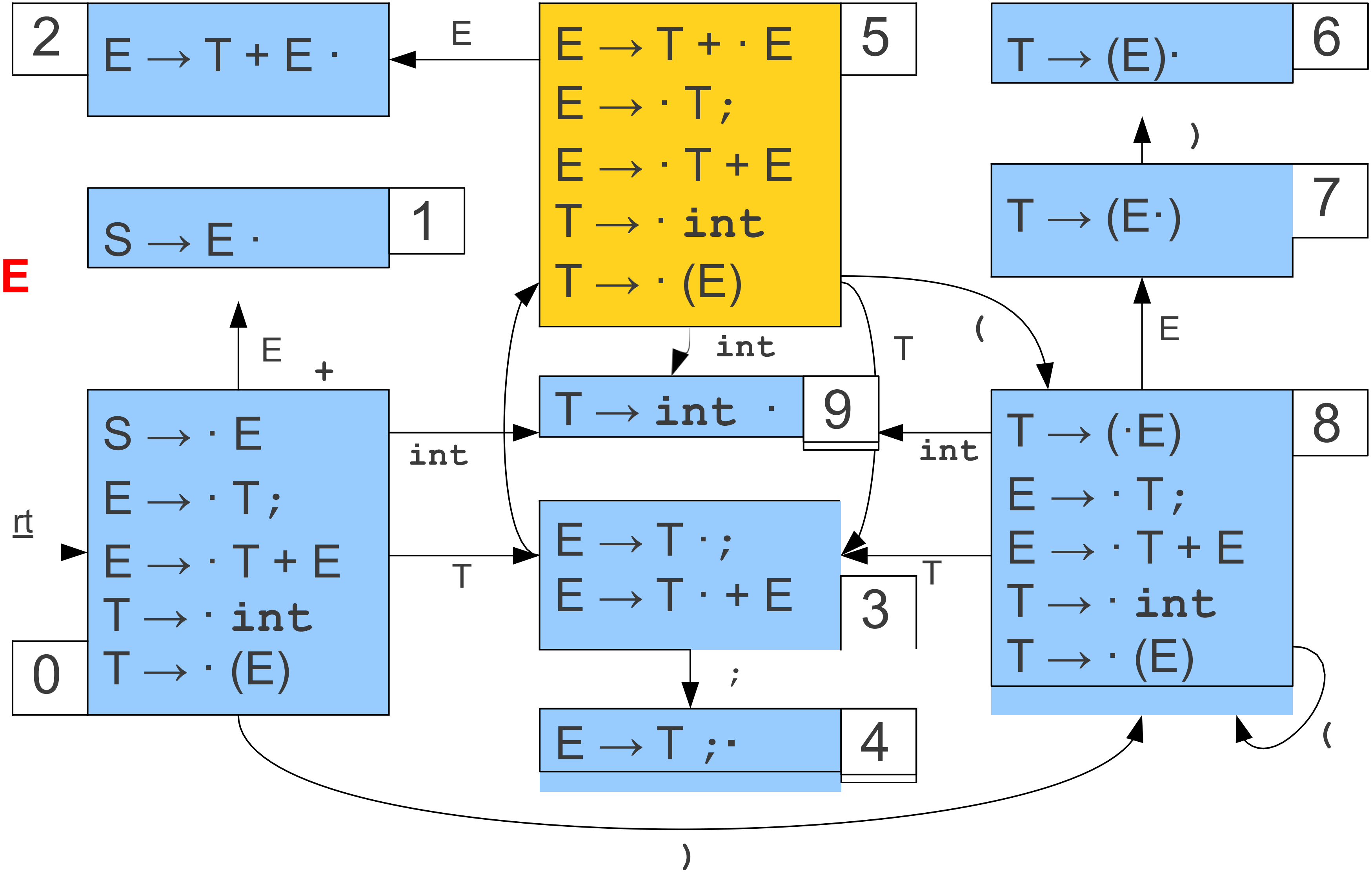
7: T → (E·)

8:
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

T	+		(	int	+	int	;	)	;
0	3	5							

LR(0) Parsing

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



sta rt

T	+	(
0	3	5

int	+	int	;	)	;

LR(0) Parsing

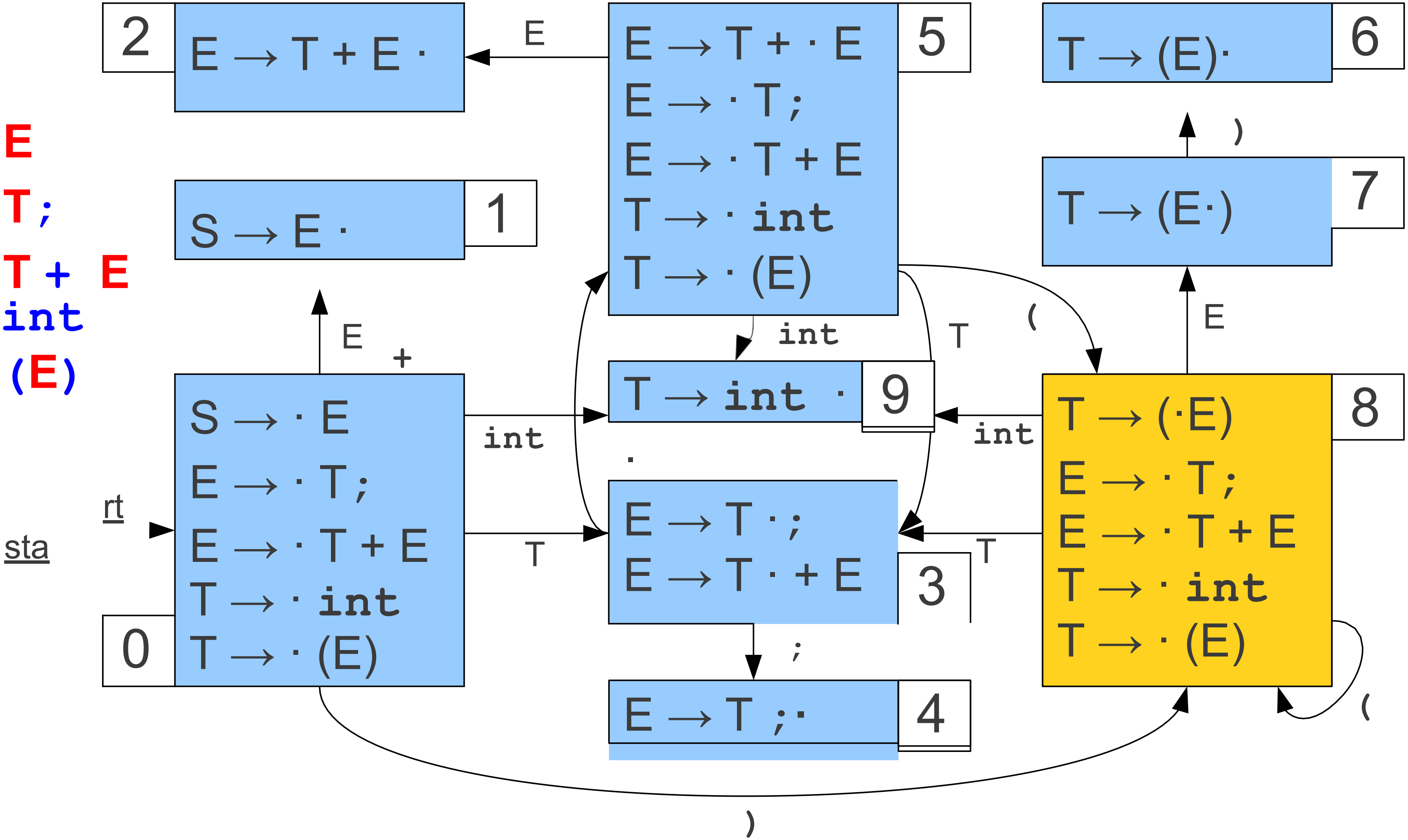
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(
0	3	5

int	+	int	;	)	;

LR(0) Parsing

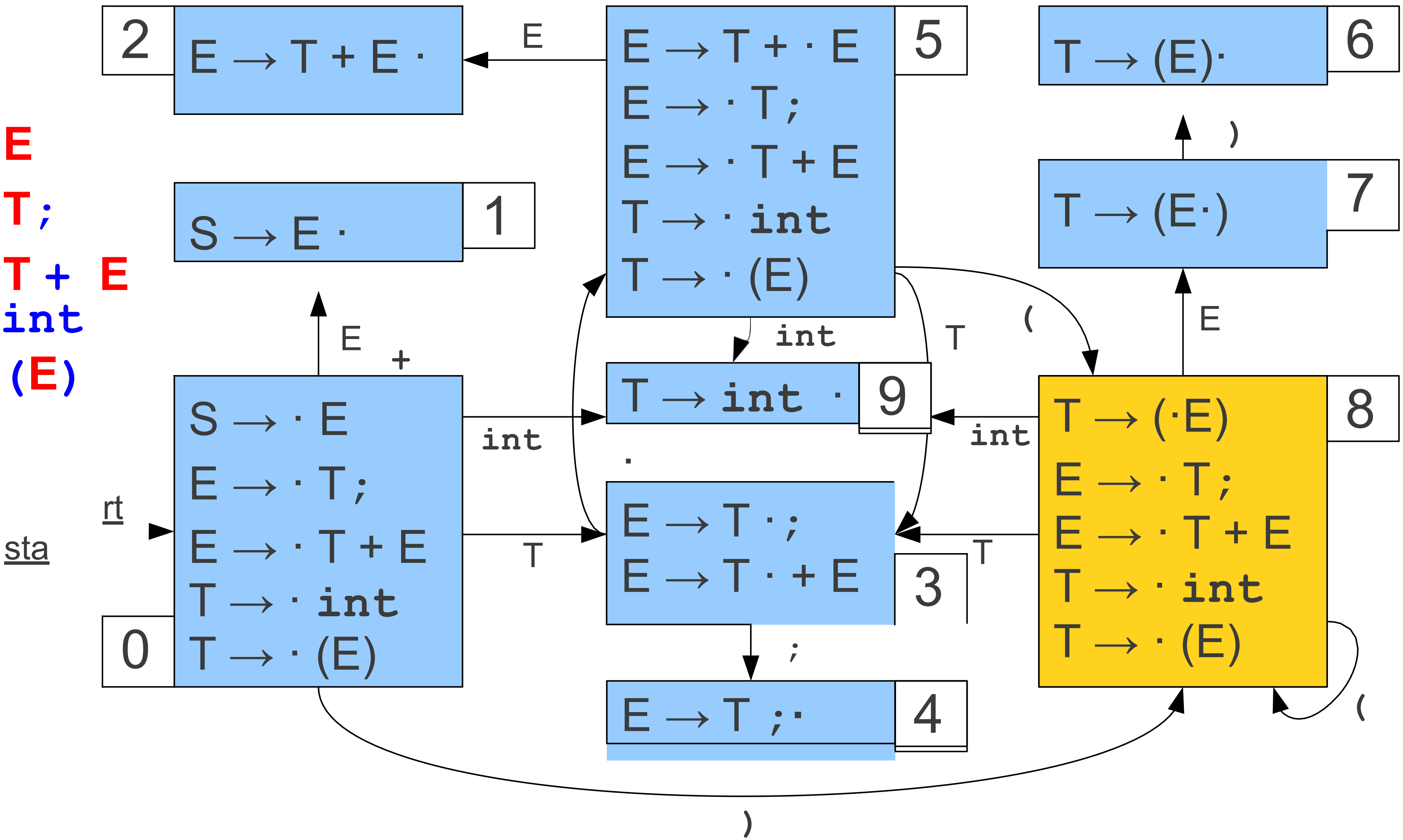
S → E

E → T;

E → T + E

T → int

T → (E)



2: E → T + E ·

1: S → E ·

0: S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

5: E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

9: T → int ·

3: E → T · ;
E → T · + E

4: E → T ; ·

6: T → (E)·

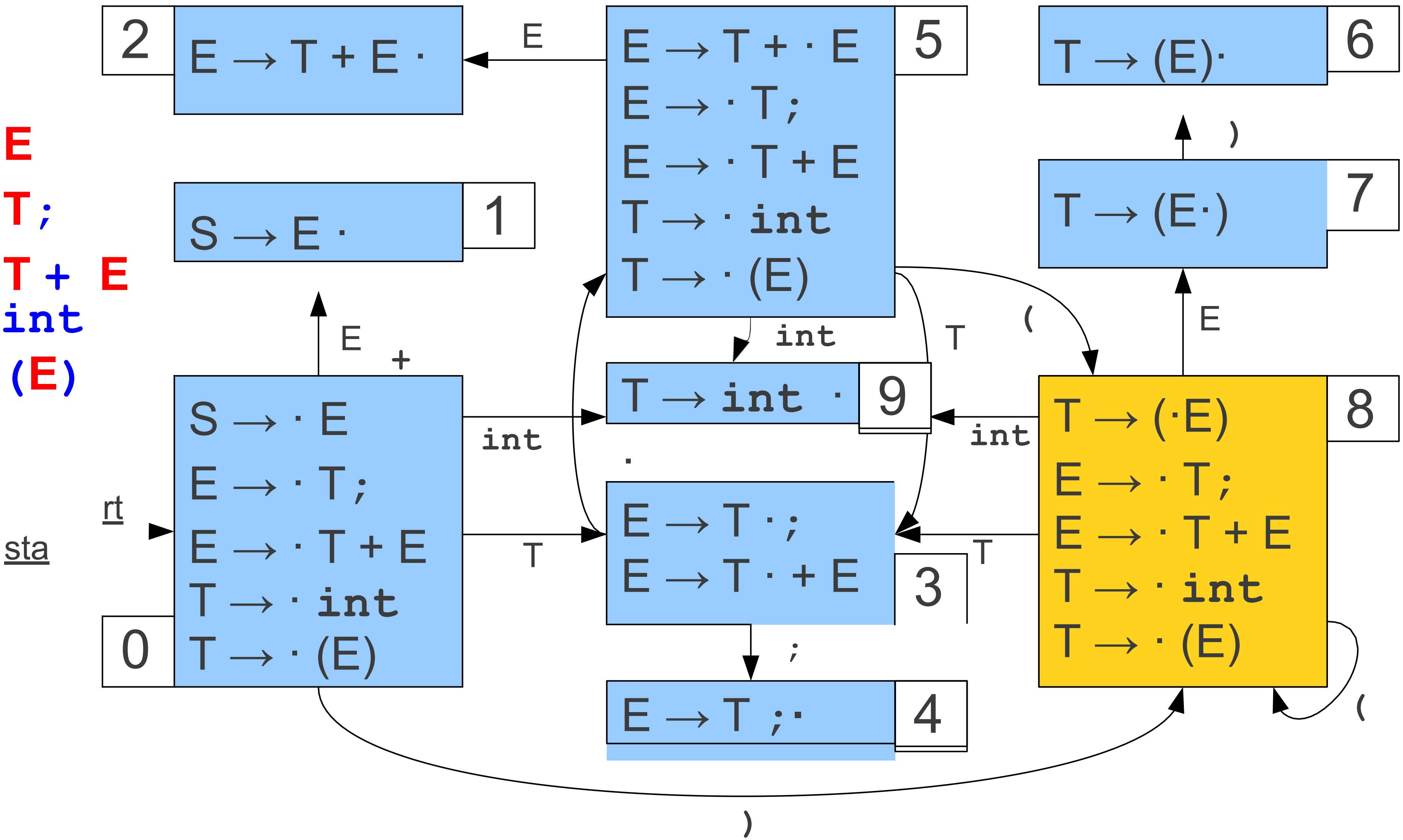
7: T → (E·)

8: T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

T	+	(	int	+	int	;	)	;
0	3	5	8					

LR(0) Parsing

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



T	+	(	int		+	int	;	)	;
0	3	5	8						

LR(0) Parsing

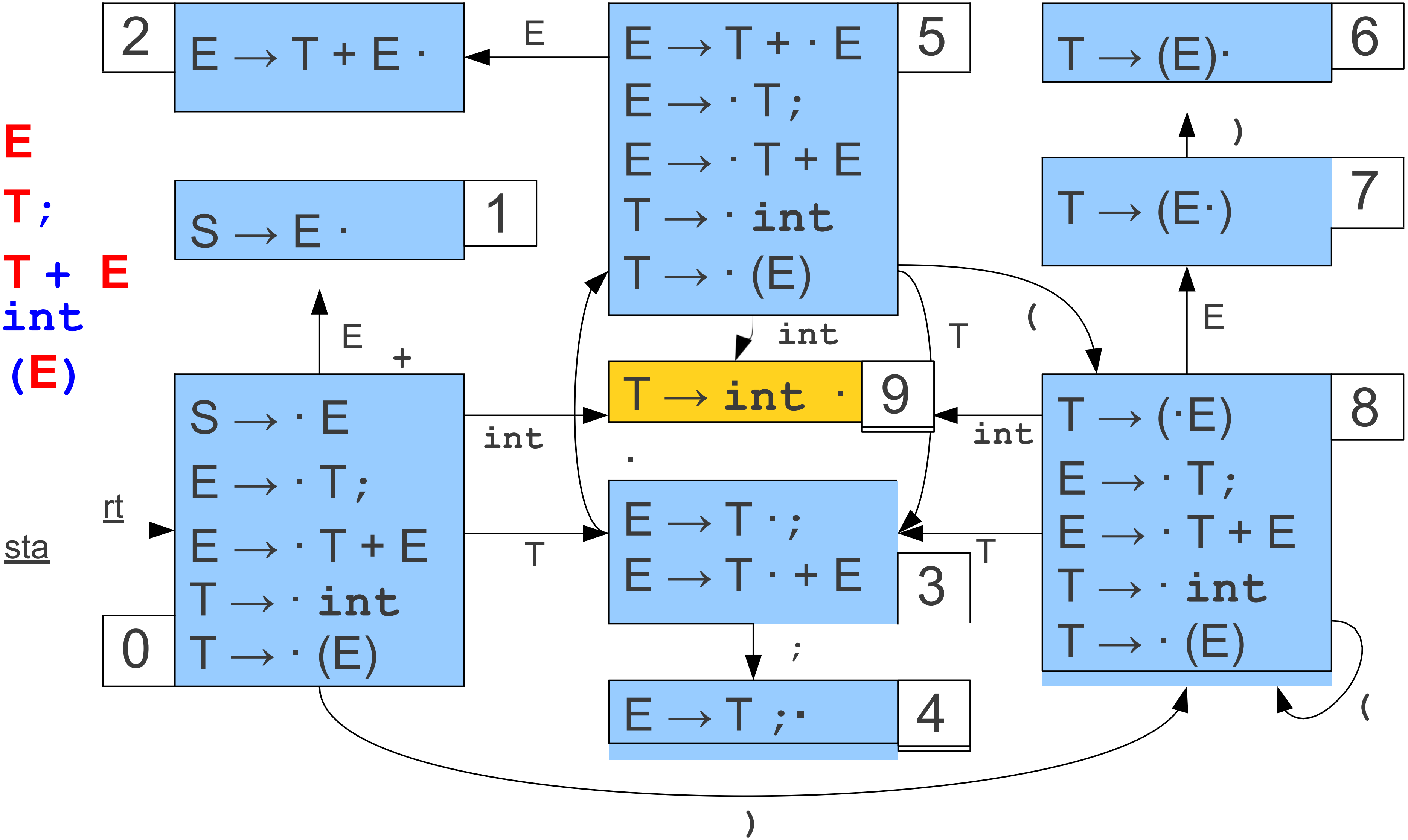
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(	int
0	3	5	8

+	int	;	)	;

LR(0) Parsing

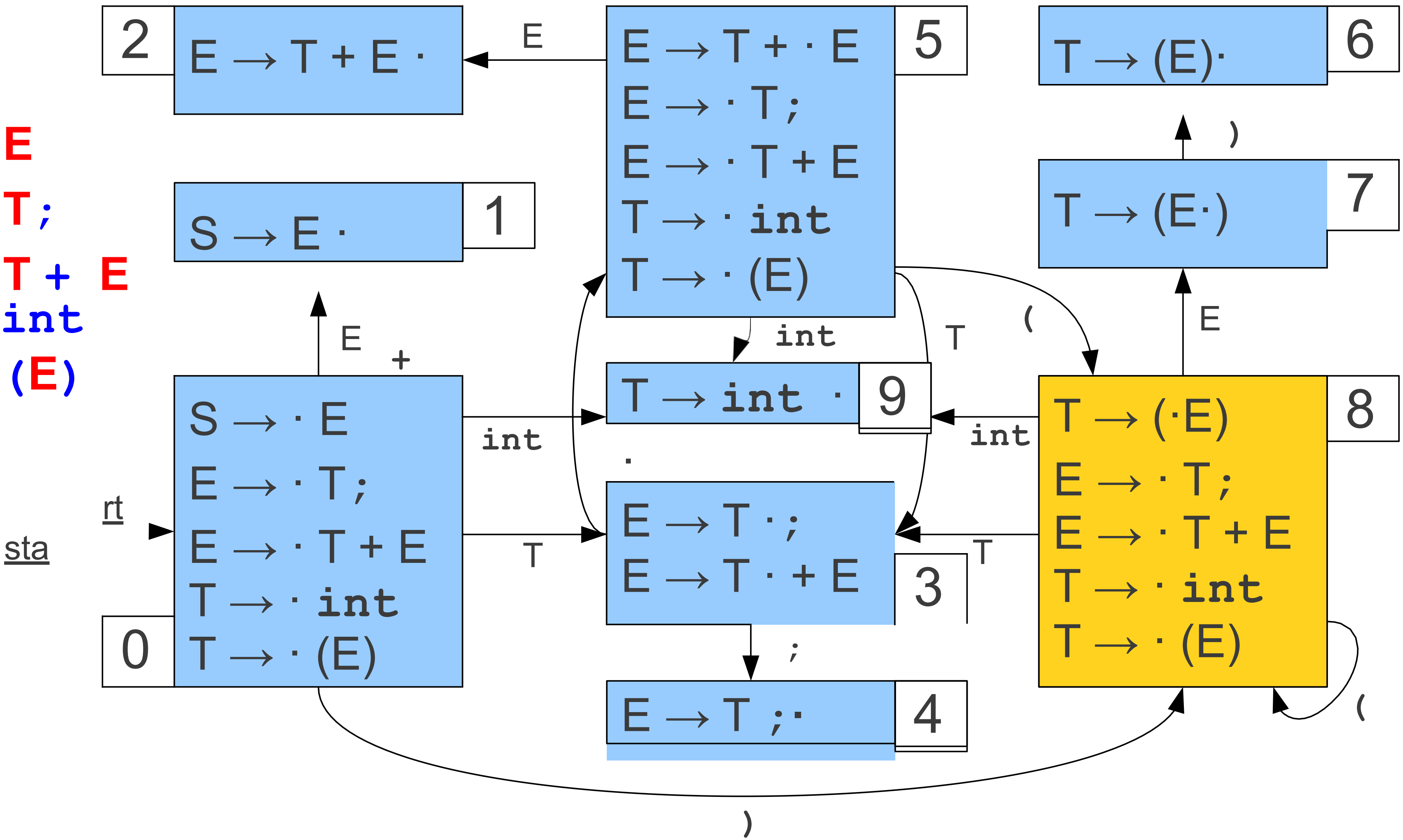
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(

0	3	5	8

+	int	;	)	;

LR(0) Parsing

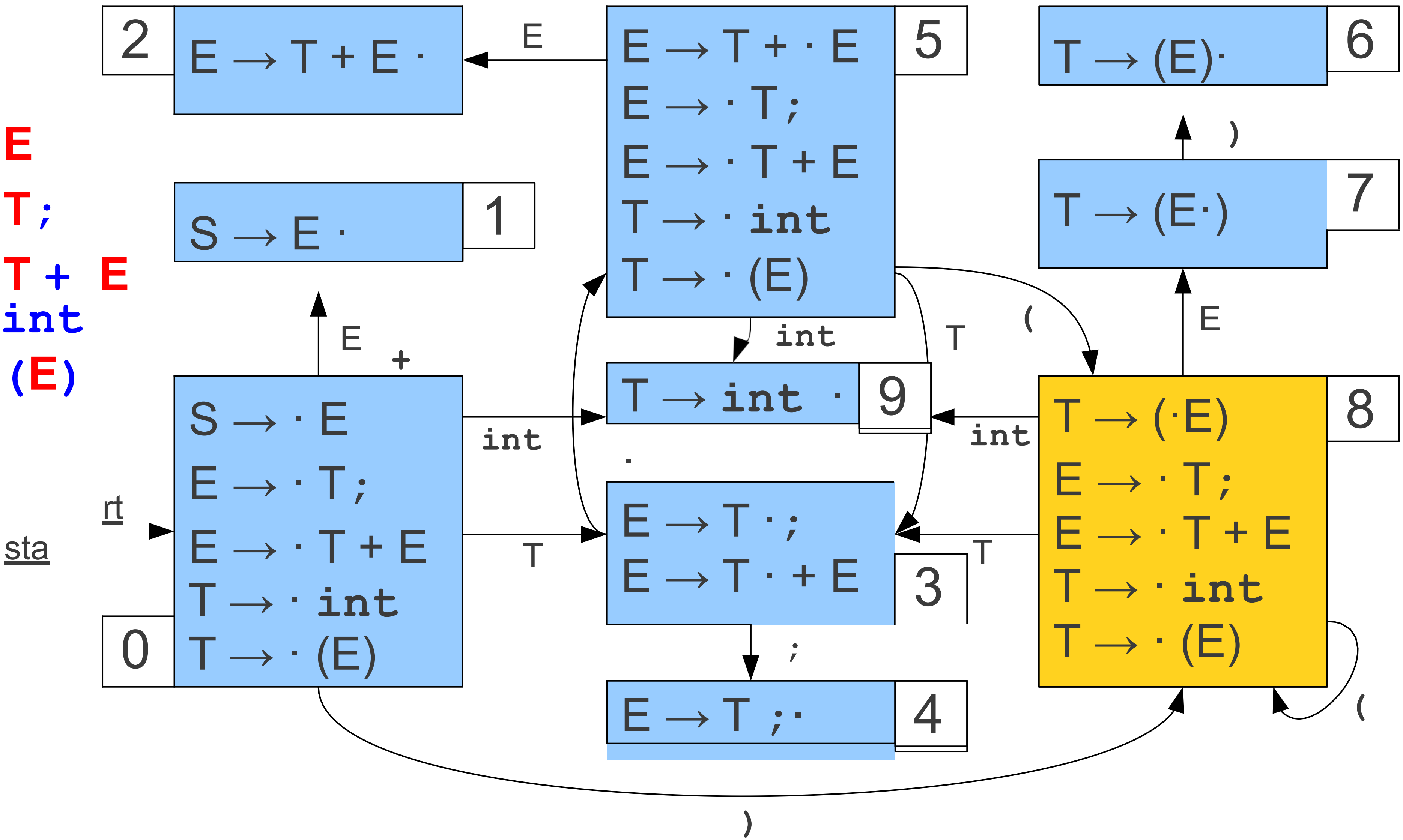
S → E

E → T ;

E → T + E

T → int

T → (E)



0
S → · E
E → · T ;
E → · T + E
T → · int
T → · (E)

1
S → E ·

2
E → T + E ·

3
E → T · ;
E → T · + E

4
E → T ; ·

5
E → T + · E
E → · T ;
E → · T + E
T → · int
T → · (E)

6
T → (E) ·

7
T → (E ·)

8
T → (· E)
E → · T ;
E → · T + E
T → · int
T → · (E)

9
T → int ·

	T	+	(	T		+	int	;	)	;
0	3	5	8						

LR(0) Parsing

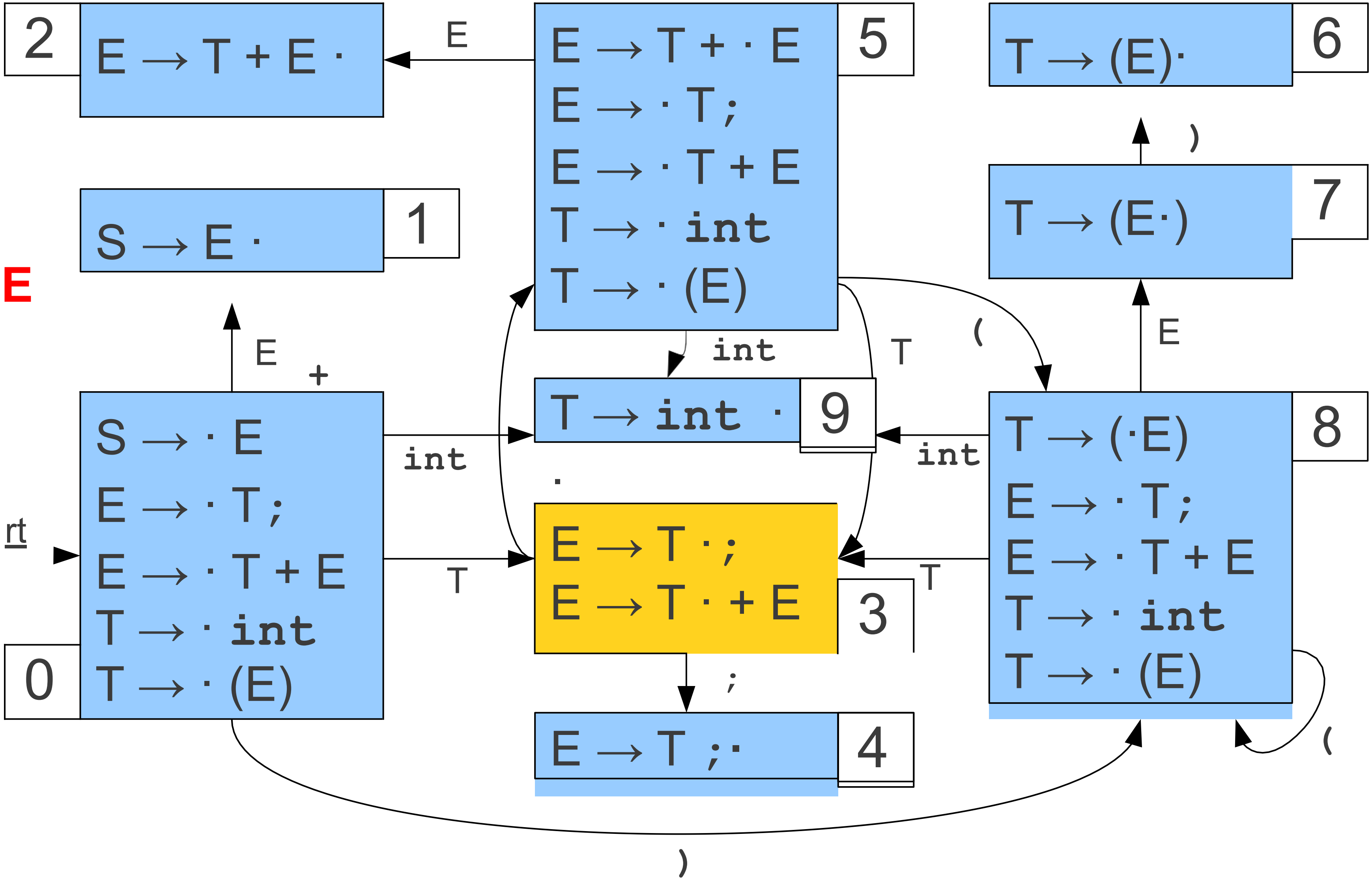
S → E

E → T;

E → T + E

T → int

T → (E)



2: E → T + E ·

5:
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6: T → (E)·

7: T → (E·)

1: S → E ·

0:
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

9: T → int ·

3:
E → T · ;
E → T · + E

8:
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

4: E → T ;·

T	+	(	T
0	3	5	8

+	int	;	)	;

LR(0) Parsing

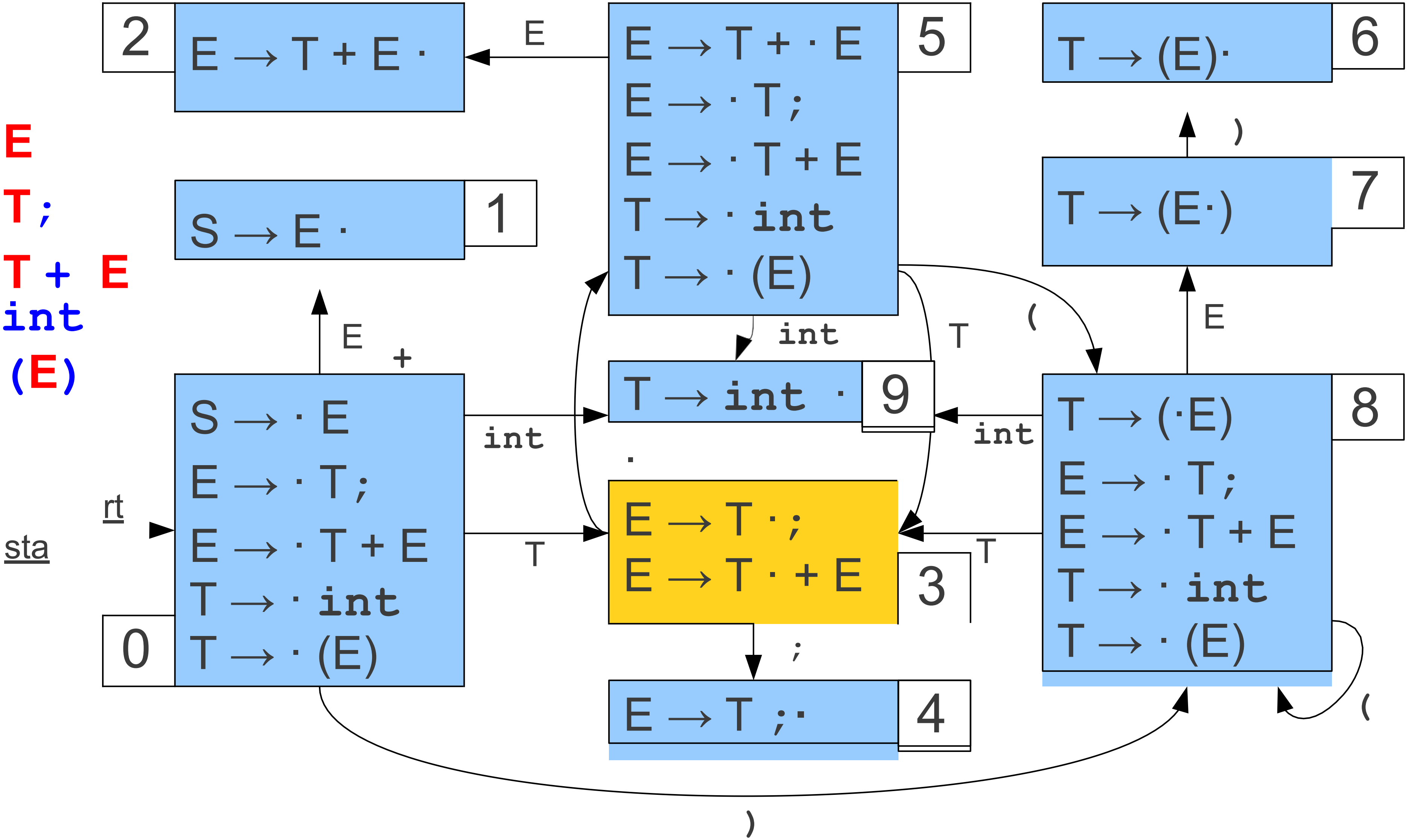
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(	T

+	int	;	)	;

0	3	5	8	3

LR(0) Parsing

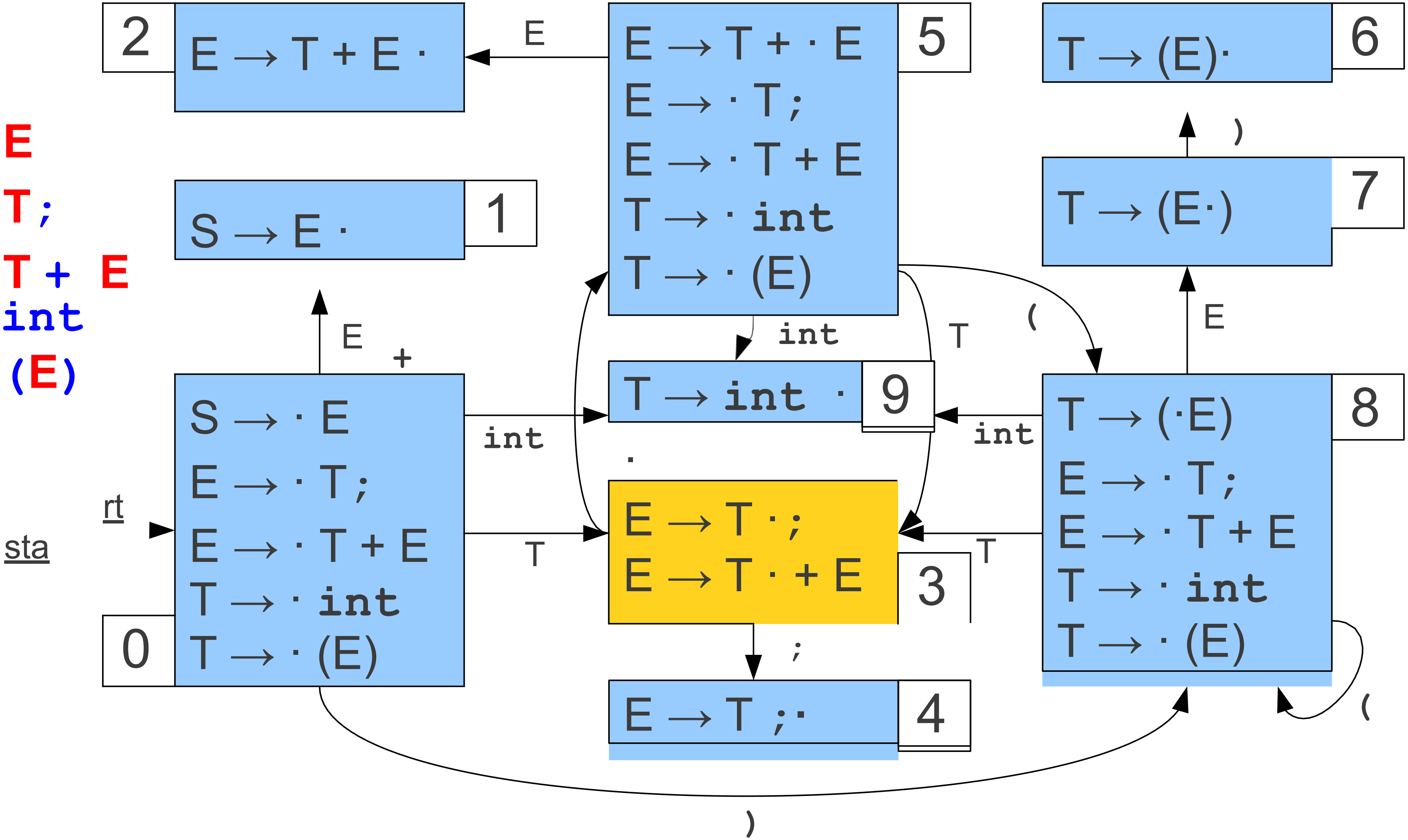
S → E

E → T;

E → T + E

T → int

T → (E)



2: E → T + E ·

5:
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6: T → (E)·

7: T → (E·)

1: S → E ·

0:
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

9: T → int ·

3:
E → T · ;
E → T · + E

4: E → T ;·

8:
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

T	+	(	T	+
0	3	5	8	3

int	;	)	;

LR(0) Parsing

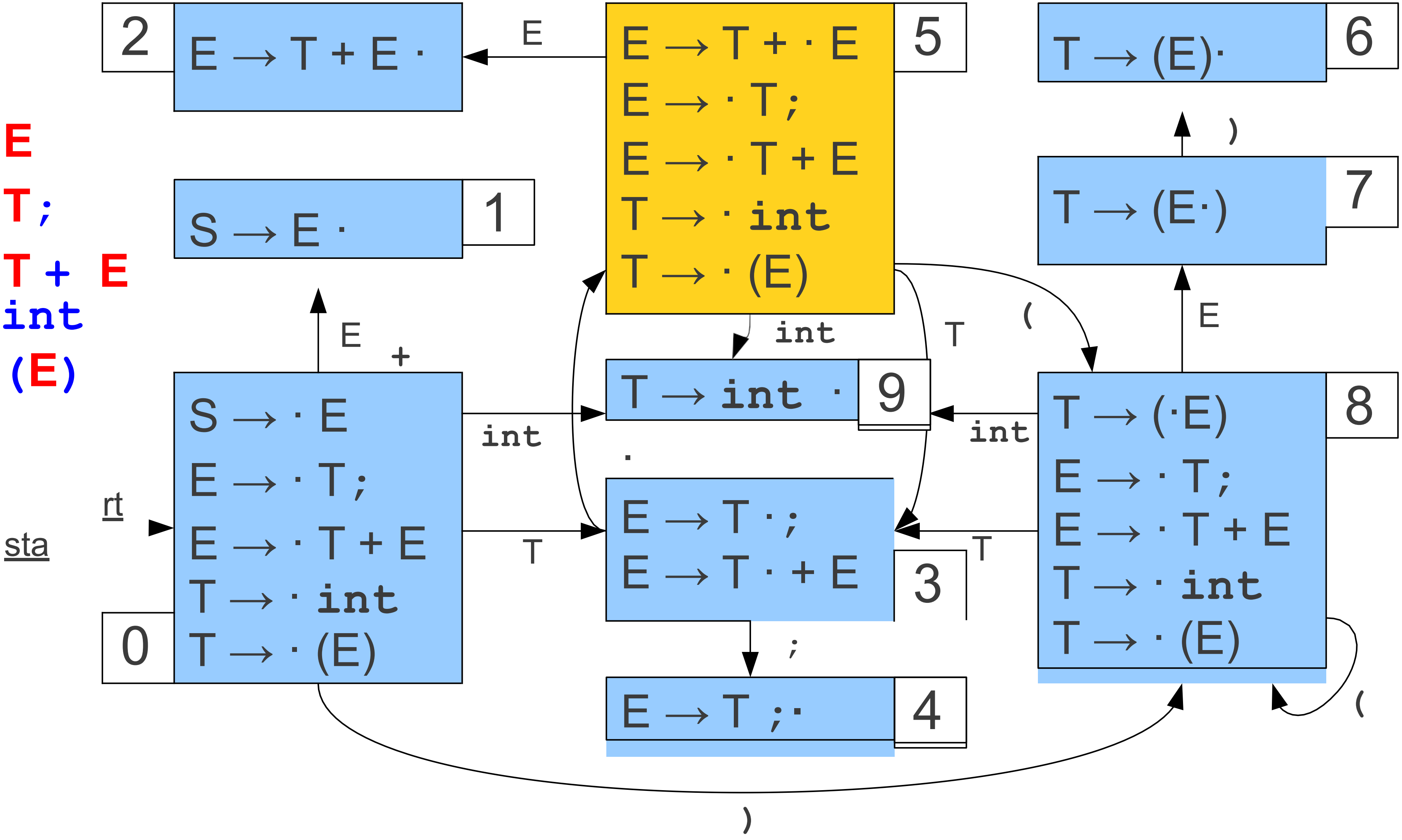
S → E

E → T;

E → T + E

T → int

T → (E)

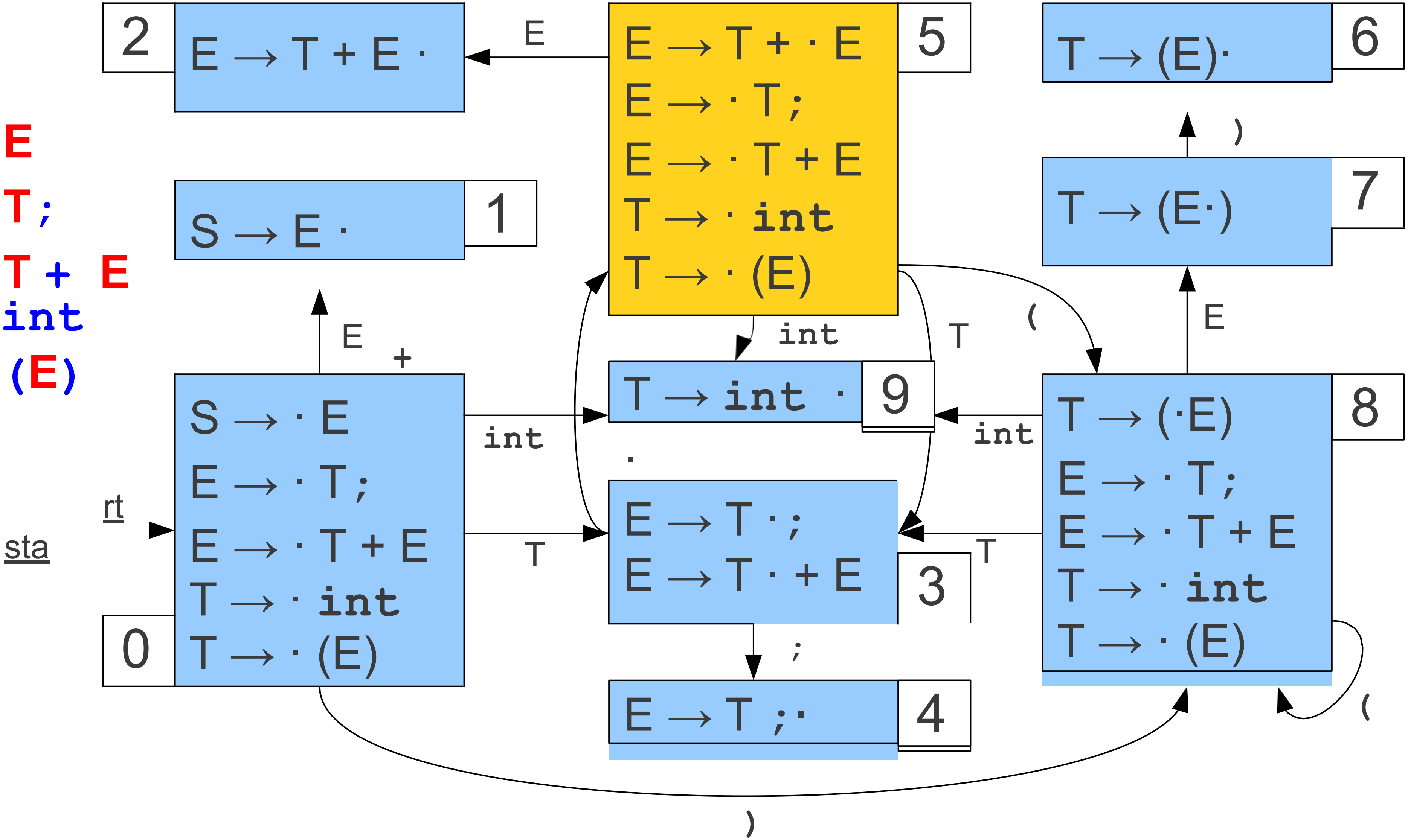


T	+	(	T	+
0	3	5	8	3

int	;	)	;

LR(0) Parsing

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



T	+	(	T	+		int	;	)	;
0	3	5	8	3	5				

LR(0) Parsing

S → E

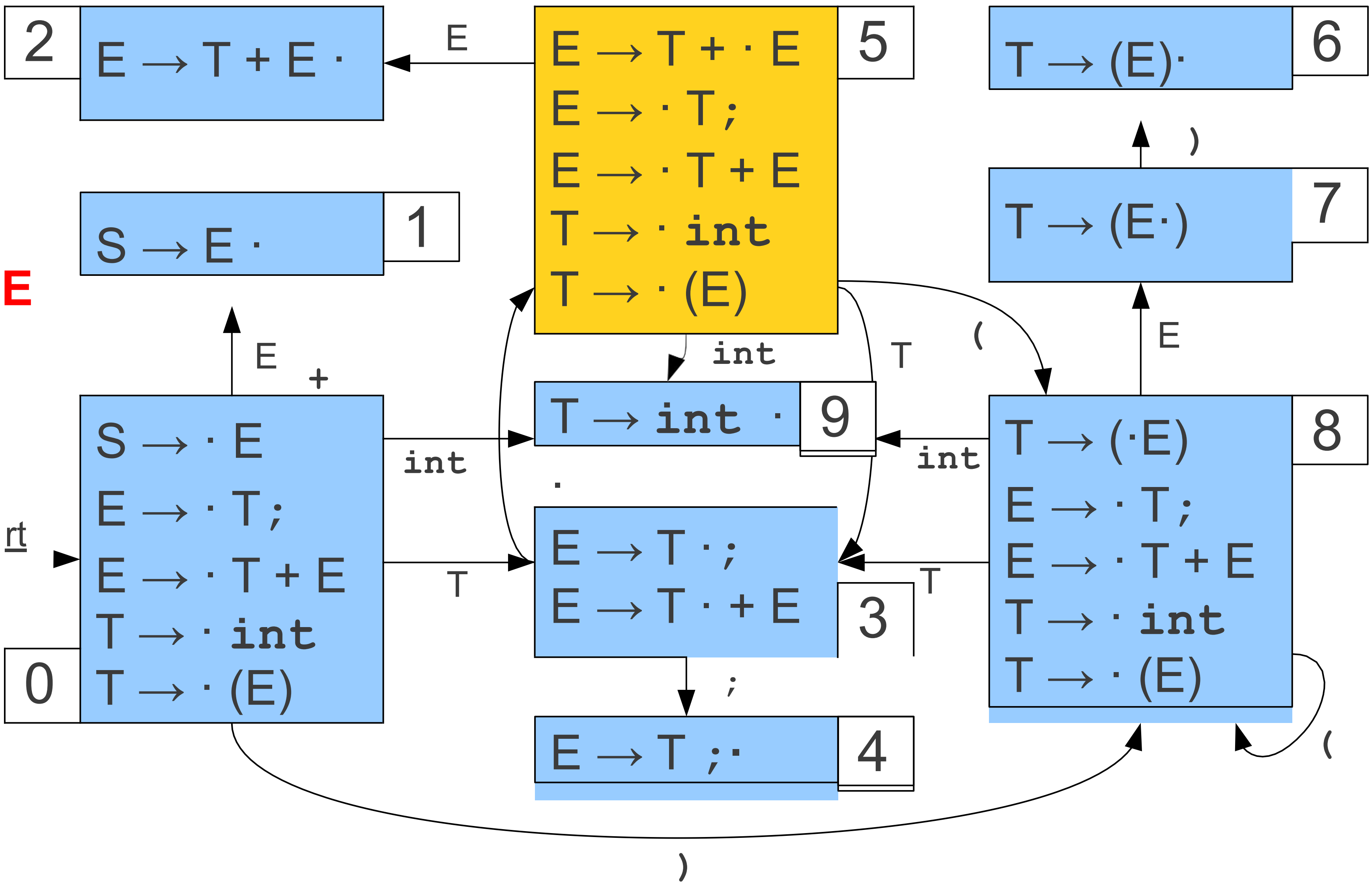
E → T;

E → T + E

T → int

T → (E)

sta rt



T	+	(	T	+	int

;	)	;

0	3	5	8	3	5

LR(0) Parsing

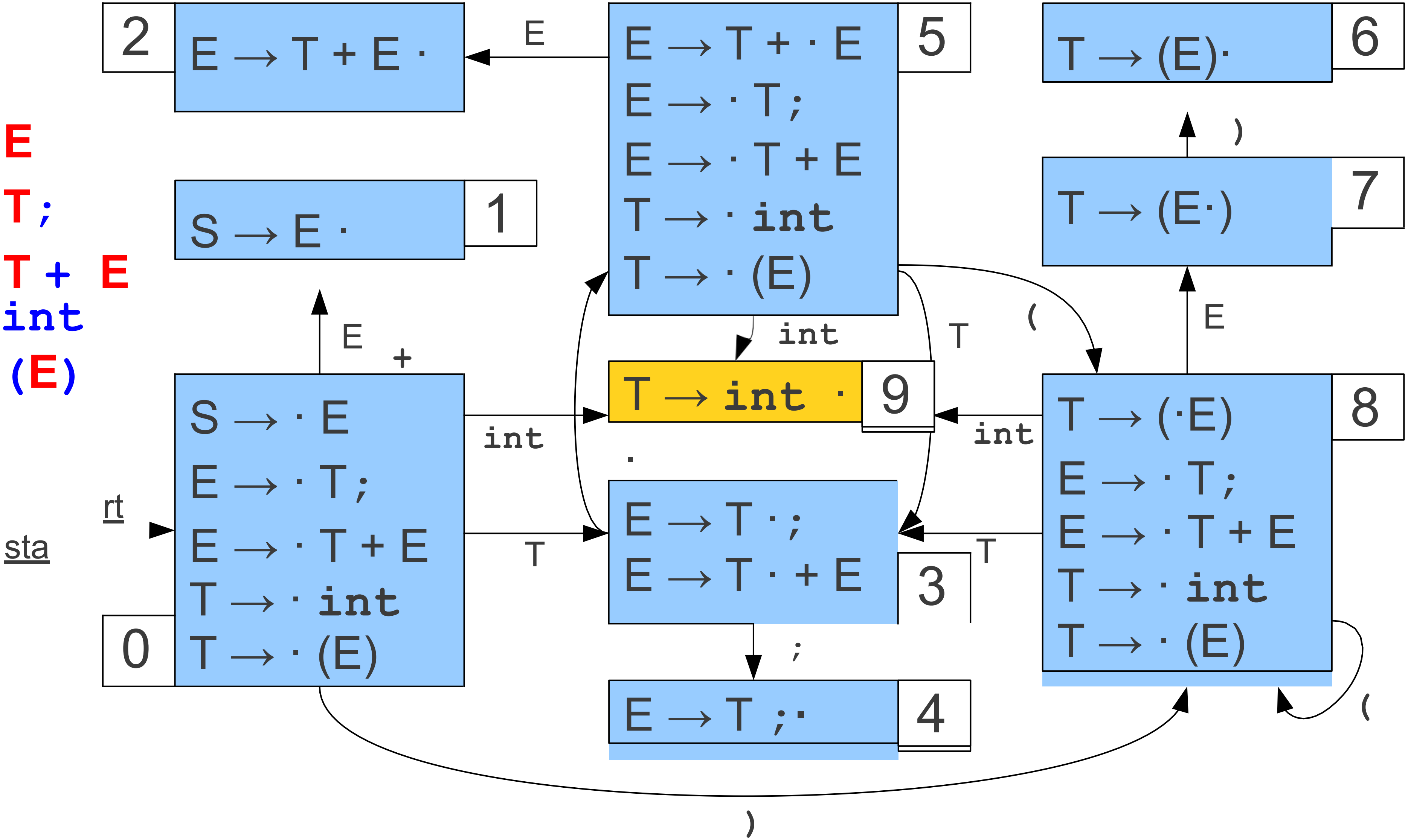
S → **E**

E → **T**`;`

E → **T** `+` **E**

T → `int`

T → `(`**E**`)`



T	+	(	T	+	int	;	)	;

0	3	5	8	3	5

LR(0) Parsing

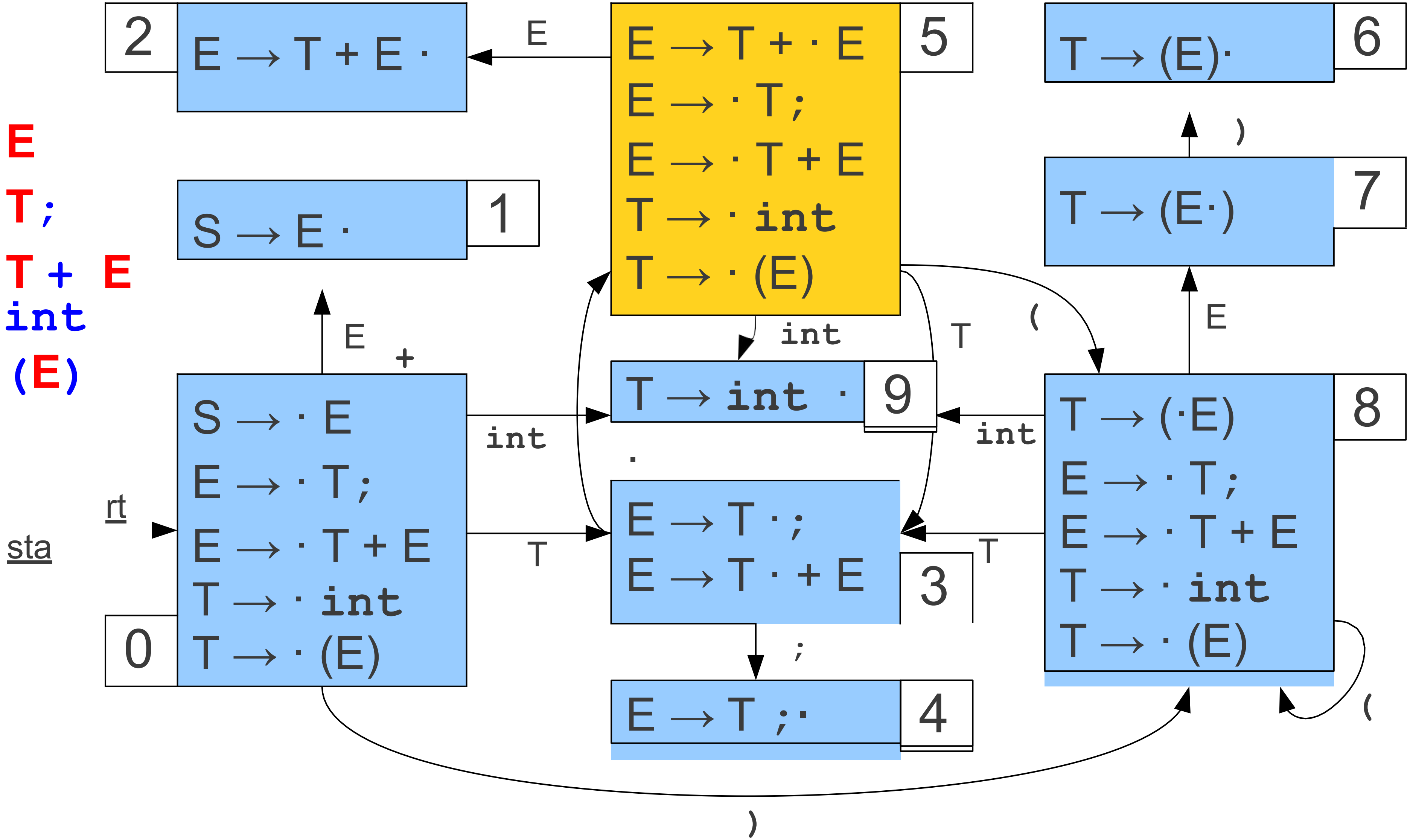
S → E

E → T;

E → T + E

T → int

T → (E)



2: E → T + E ·

5: E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6: T → (E)·

7: T → (E·)

1: S → E ·

0: S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

9: T → int ·

3: E → T · ;
E → T · + E

4: E → T ;·

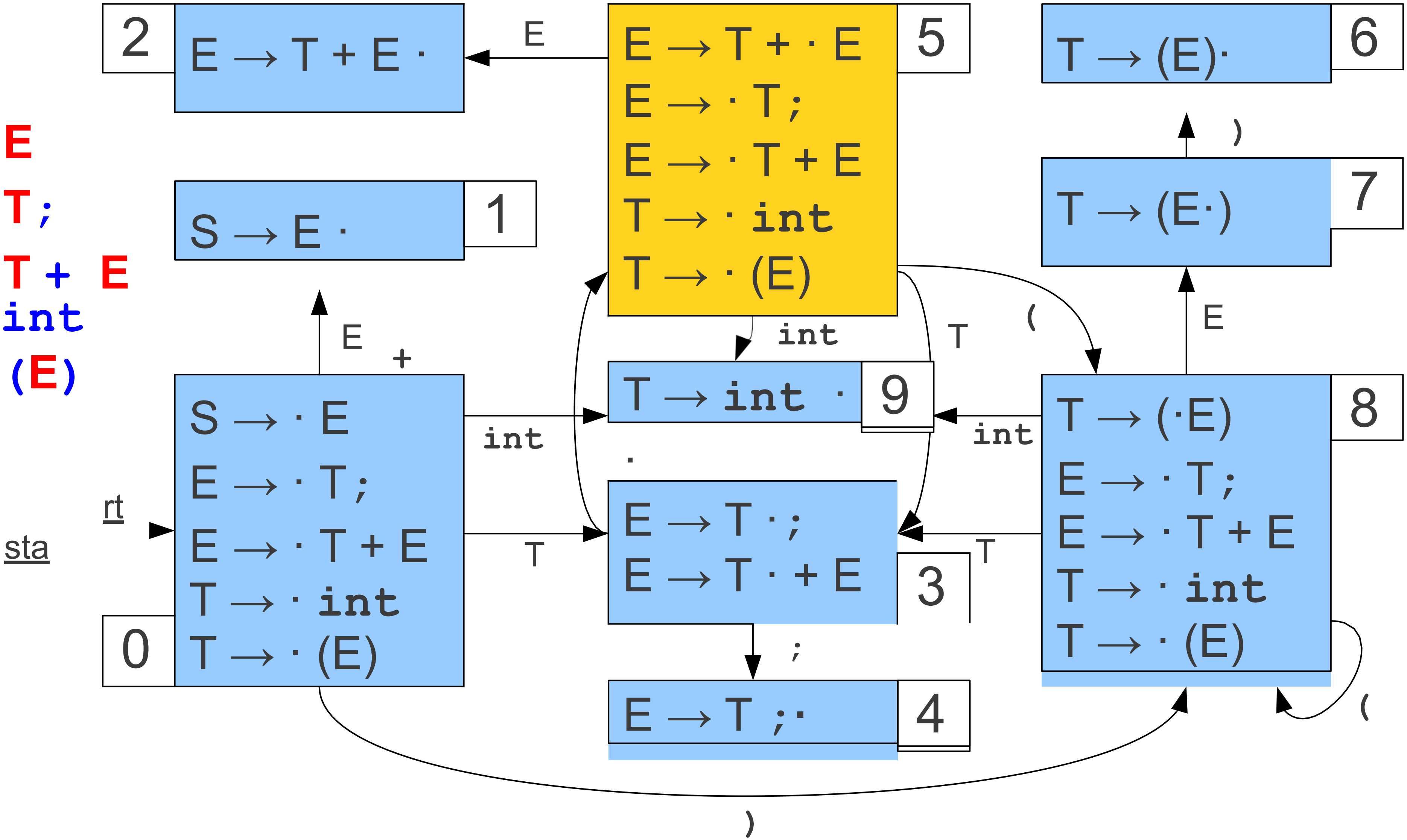
8: T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

	T	+	(	T	+
0	3	5	8	3	5

;	)	;

LR(0) Parsing

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



2: E → T + E ·

1: S → E ·

5:
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6: T → (E)·

7: T → (E·)

0:
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

9: T → int ·

3:
E → T · ;
E → T · + E

8:
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

4: E → T ;·

T	+	(	T	+	T
0	3	5	8	3	5

LR(0) Parsing

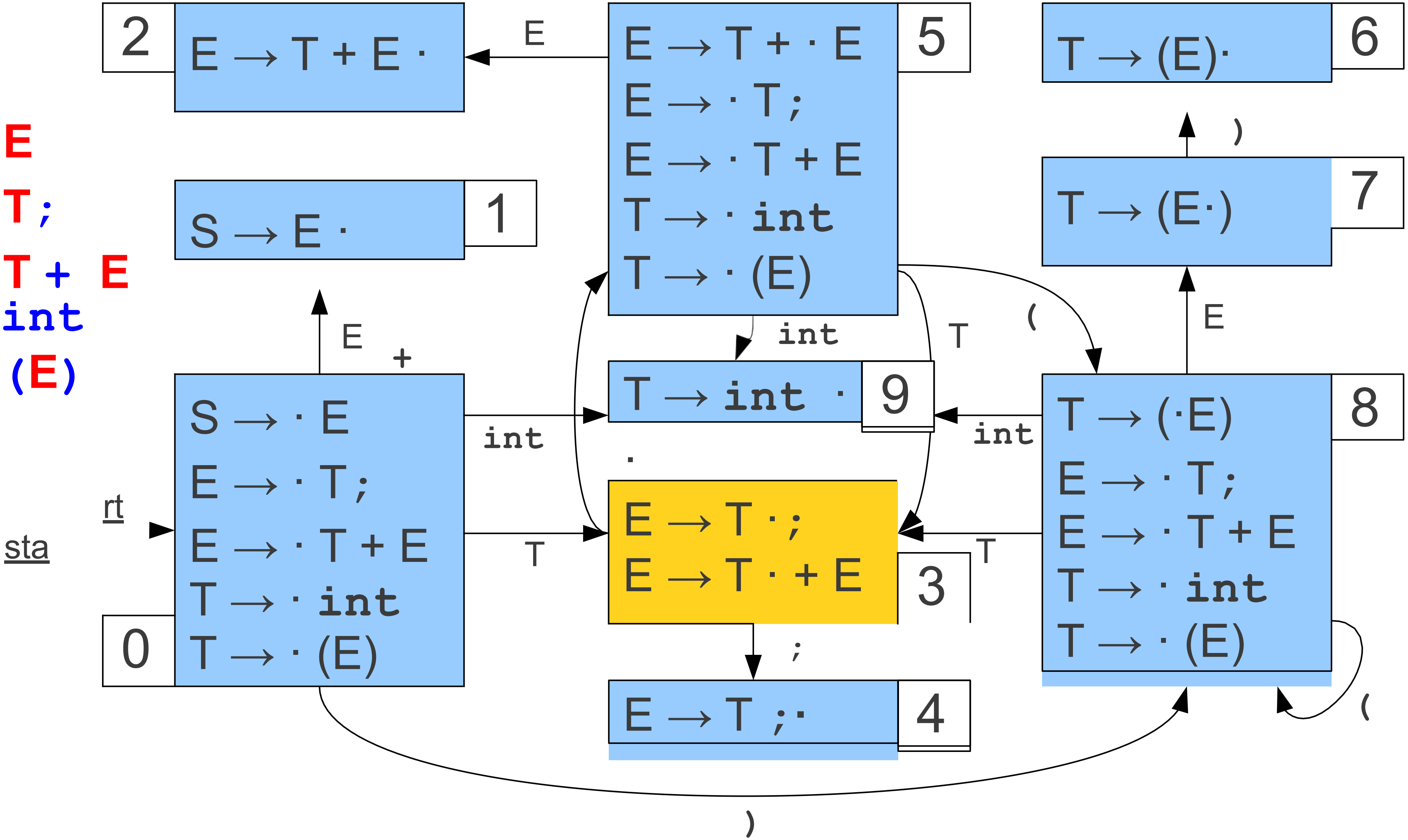
S → E

E → T;

E → T + E

T → int

T → (E)



0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

1
S → E ·

2
E → T + E ·

3
E → T · ;
E → T · + E

4
E → T ;·

5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6
T → (E)·

7
T → (E·)

8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

9
T → int ·

T	+	(	T	+	T
0	3	5	8	3	5

LR(0) Parsing

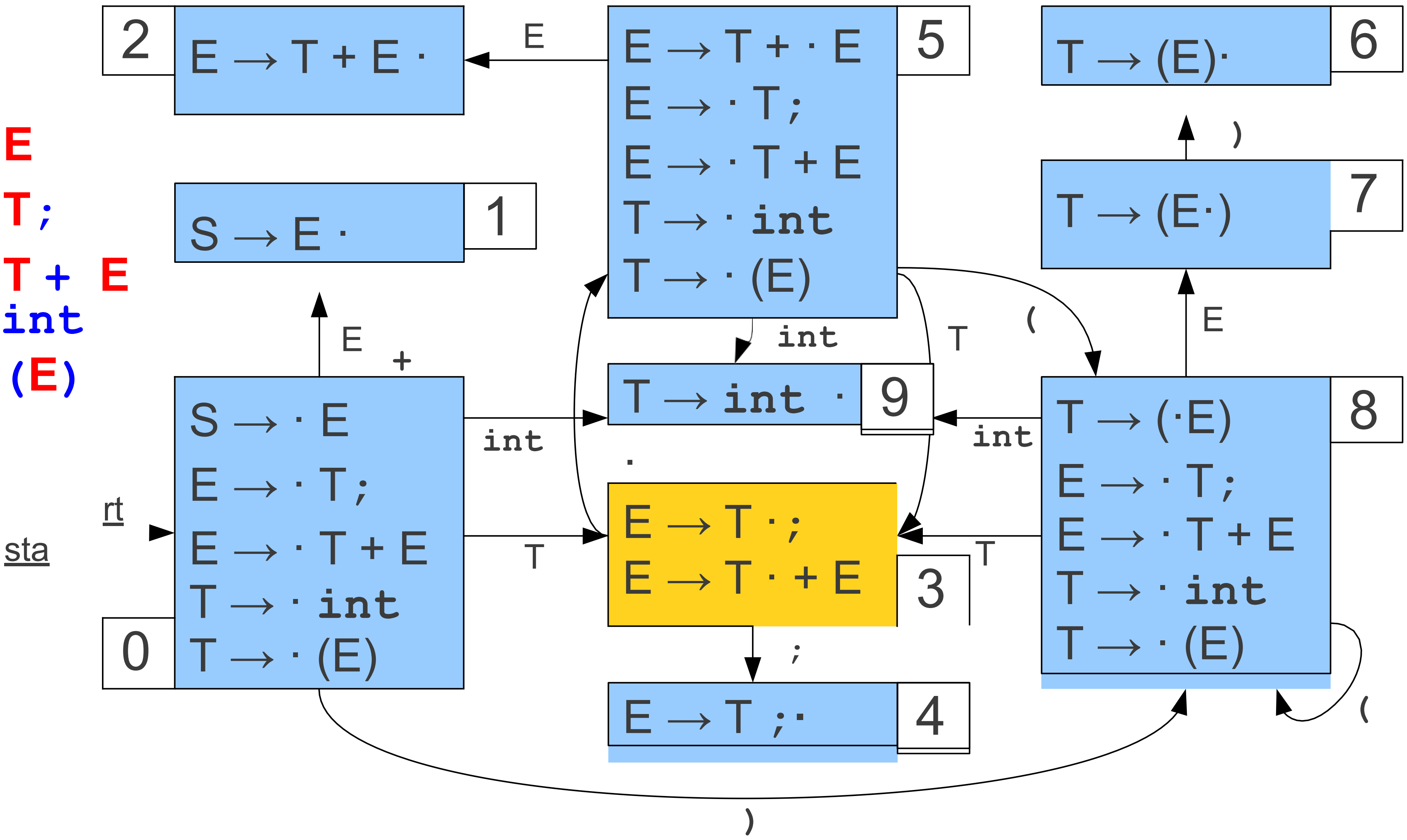
S → E

E → T;

E → T + E

T → int

T → (E)



T + (T + T ; |) ;

0 3 5 8 3 5 3

LR(0) Parsing

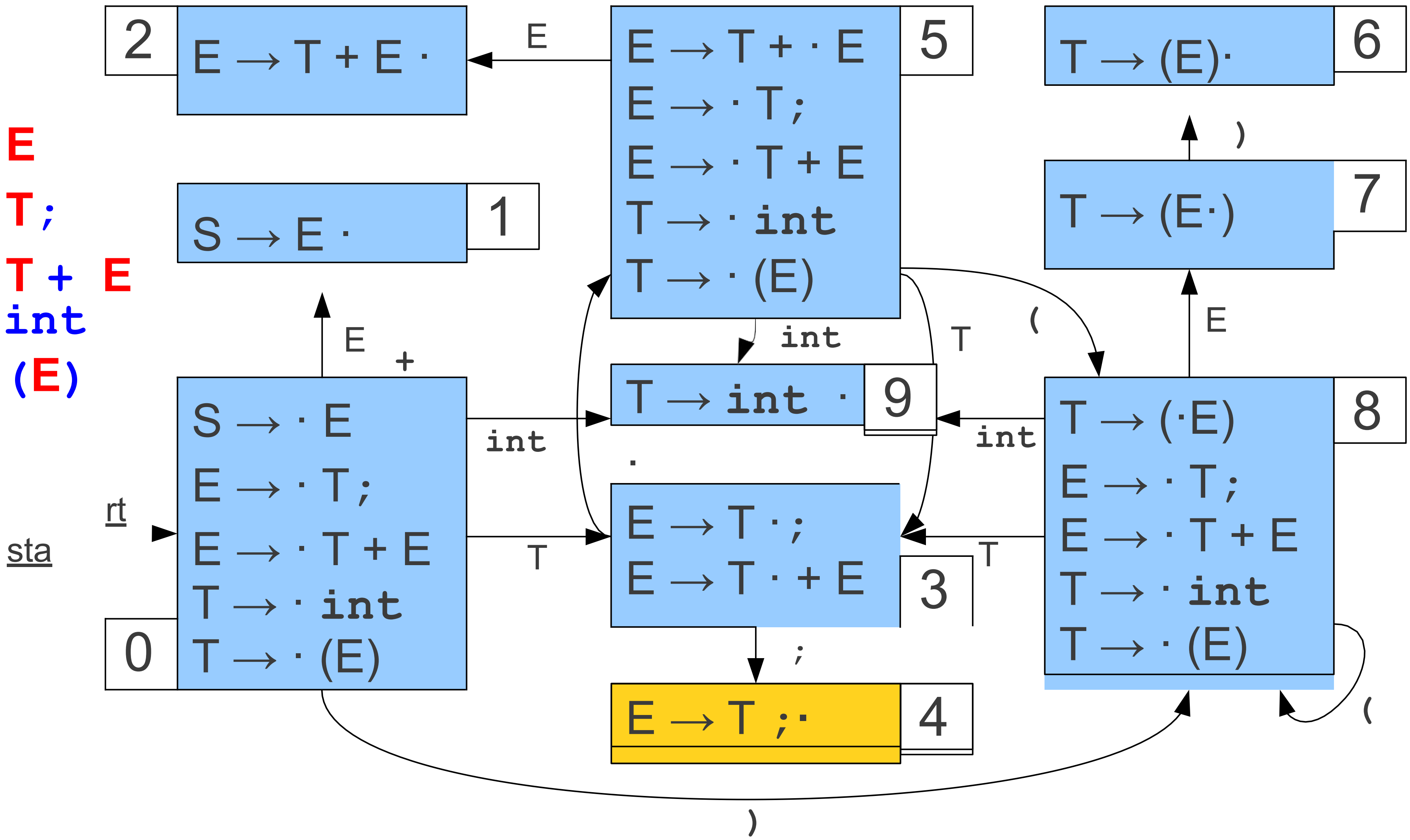
S → E

E → T;

E → T + E

T → int

T → (E)



2: E → T + E ·

1: S → E ·

0: S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

5: E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

9: T → int ·

3: E → T · ;
E → T · + E

4: E → T ;·

6: T → (E)·

7: T → (E·)

8: T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

T	+	(	T	+	T	;	|	)	;
0	3	5	8	3	5	3			

LR(0) Parsing

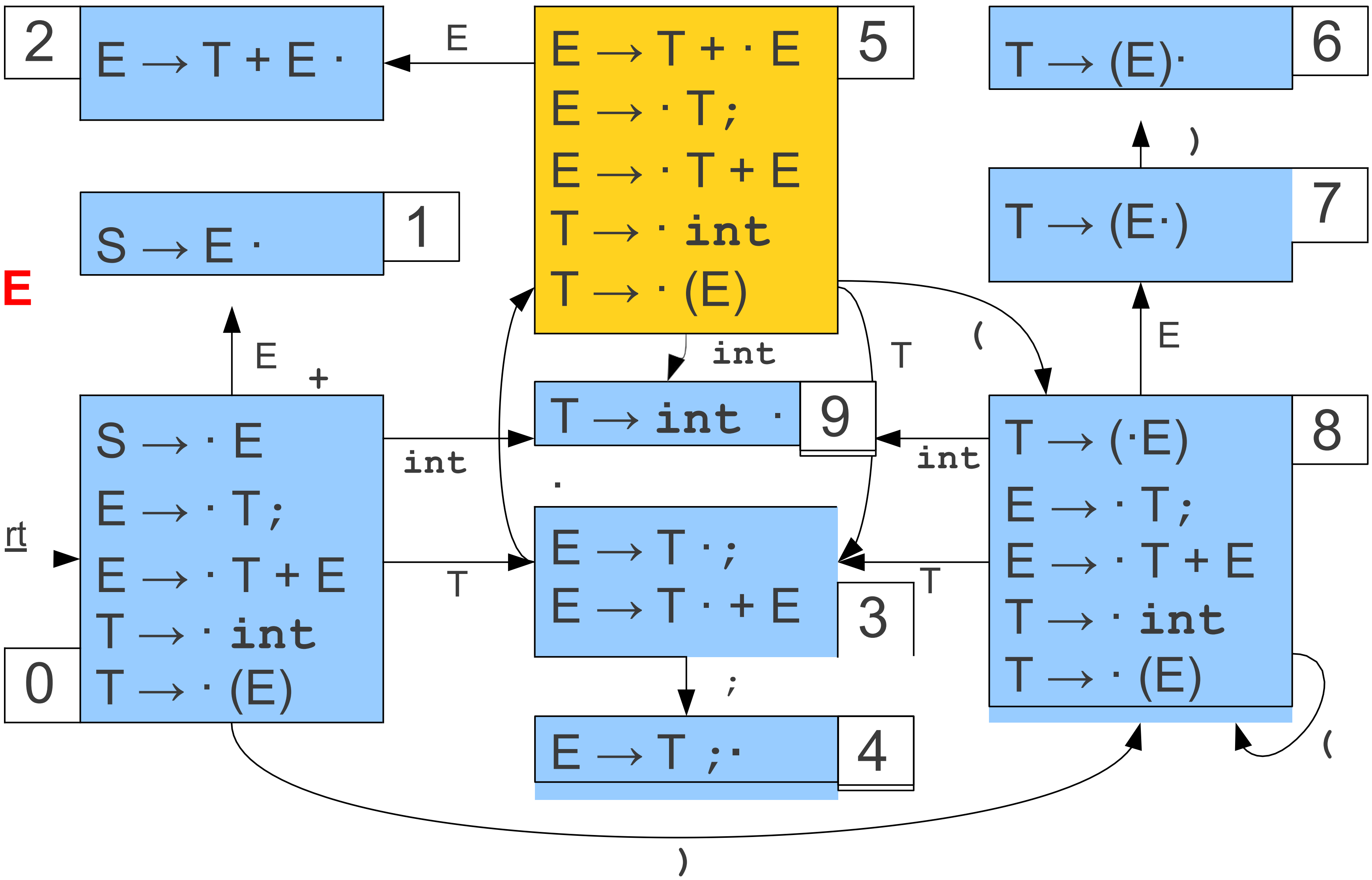
S → E

E → T;

E → T + E

T → int

T → (E)



2 E → T + E ·

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

6 T → (E)·

1 S → E ·

7 T → (E·)

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

9 T → int ·

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

3 E → T · ;
E → T · + E

4 E → T ;·

T	+	(	T	+	
0	3	5	8	3	5

)	;

LR(0) Parsing

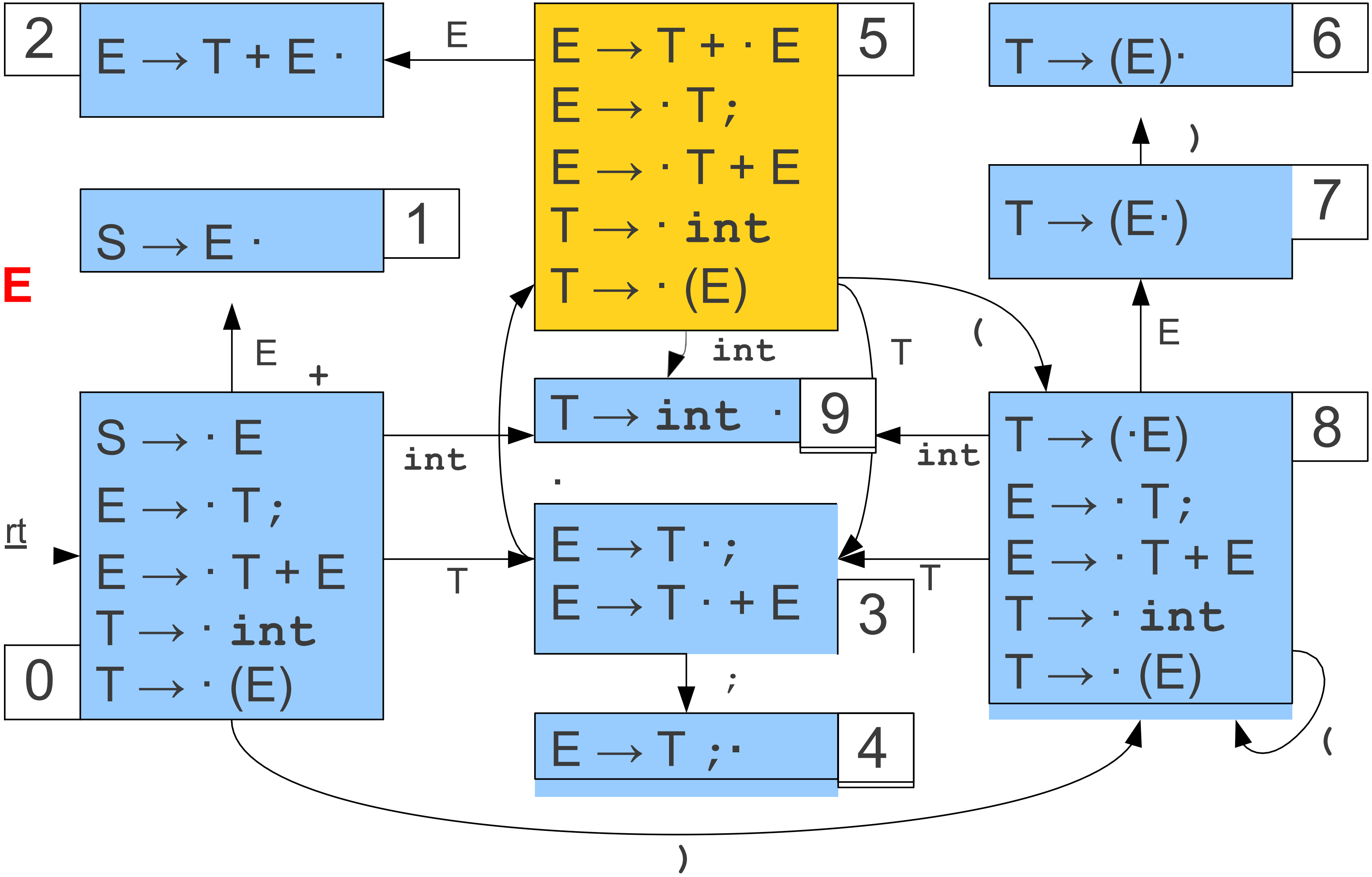
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(	T	+	E		)	;

0	3	5	8	3	5

LR(0) Parsing

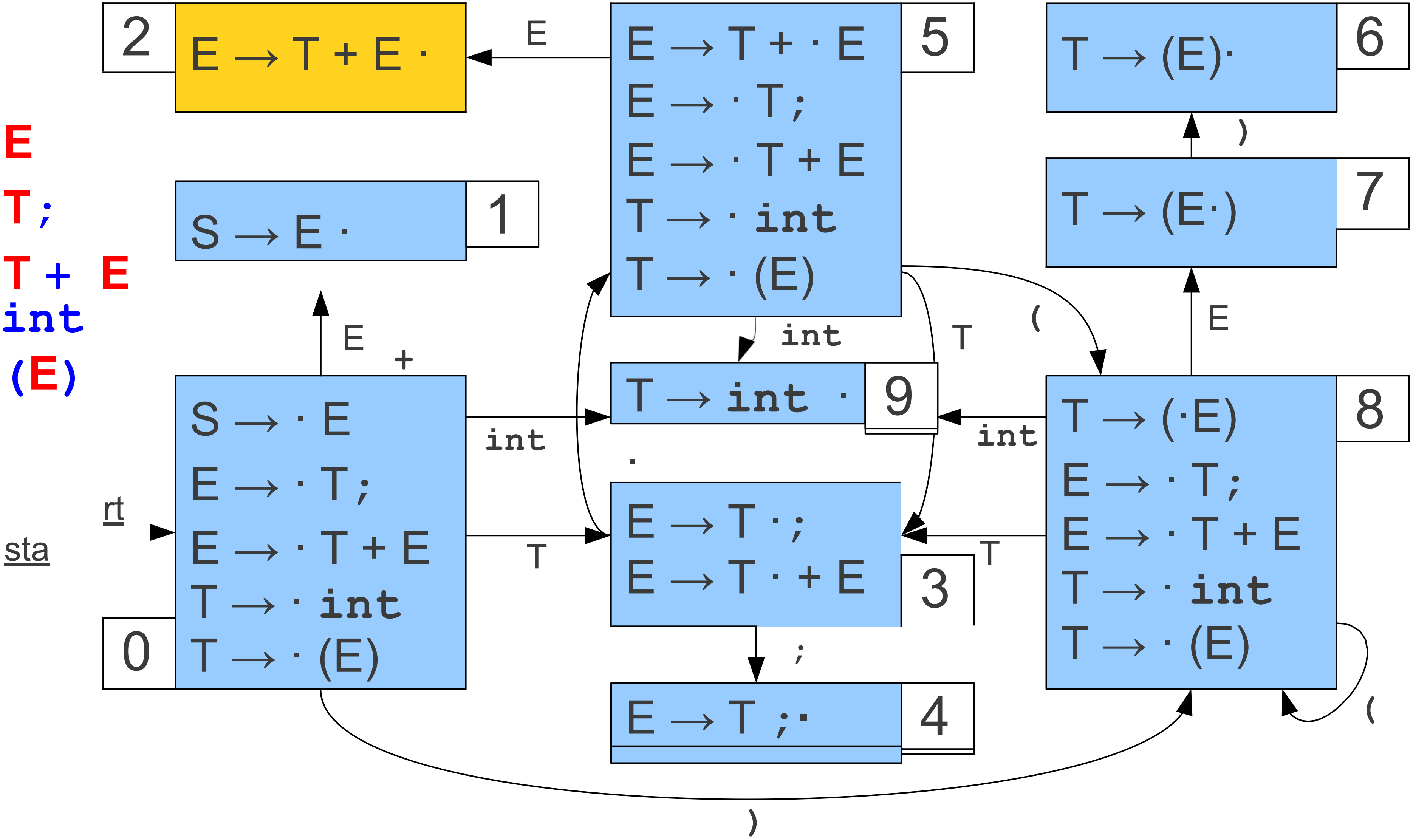
S → E

E → T;

E → T + E

T → int

T → (E)



0: S → · E / E → · T; / E → · T + E / T → · int / T → · (E)

1: S → E ·

2: E → T + E ·

3: E → T · ; / E → T · + E

4: E → T ;·

5: E → T + · E / E → · T; / E → · T + E / T → · int / T → · (E)

6: T → (E)·

7: T → (E·)

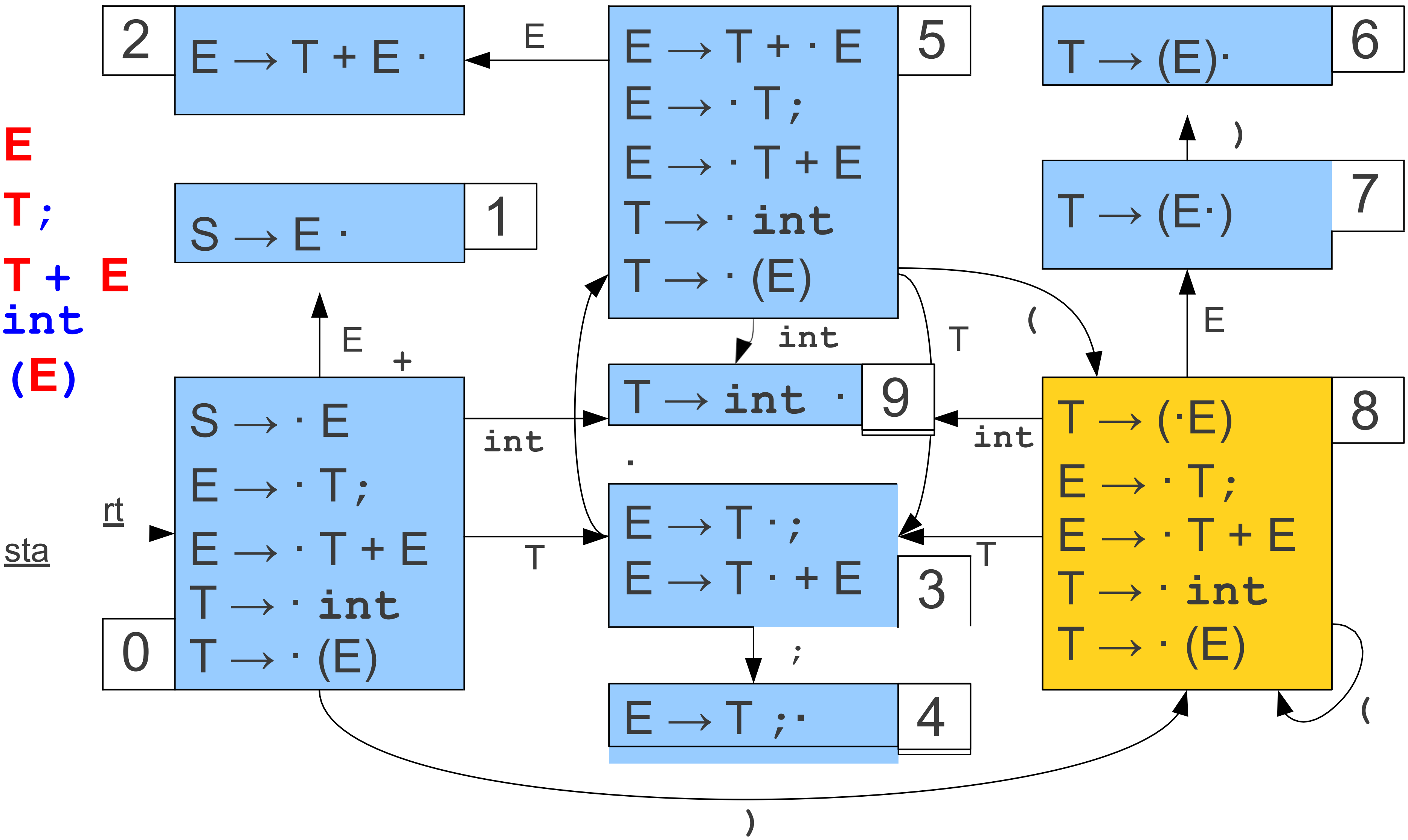
8: T → (·E) / E → · T; / E → · T + E / T → · int / T → · (E)

9: T → int ·

T	+	(	T	+	E	|	)	;
0	3	5	8	3	5			

LR(0) Parsing

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



0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

1
S → E ·

2
E → T + E ·

3
E → T · ;
E → T · + E

4
E → T ; ·

5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6
T → (E)·

7
T → (E·)

8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

9
T → int ·

T	+	(	
0	3	5	8

)	;

LR(0) Parsing

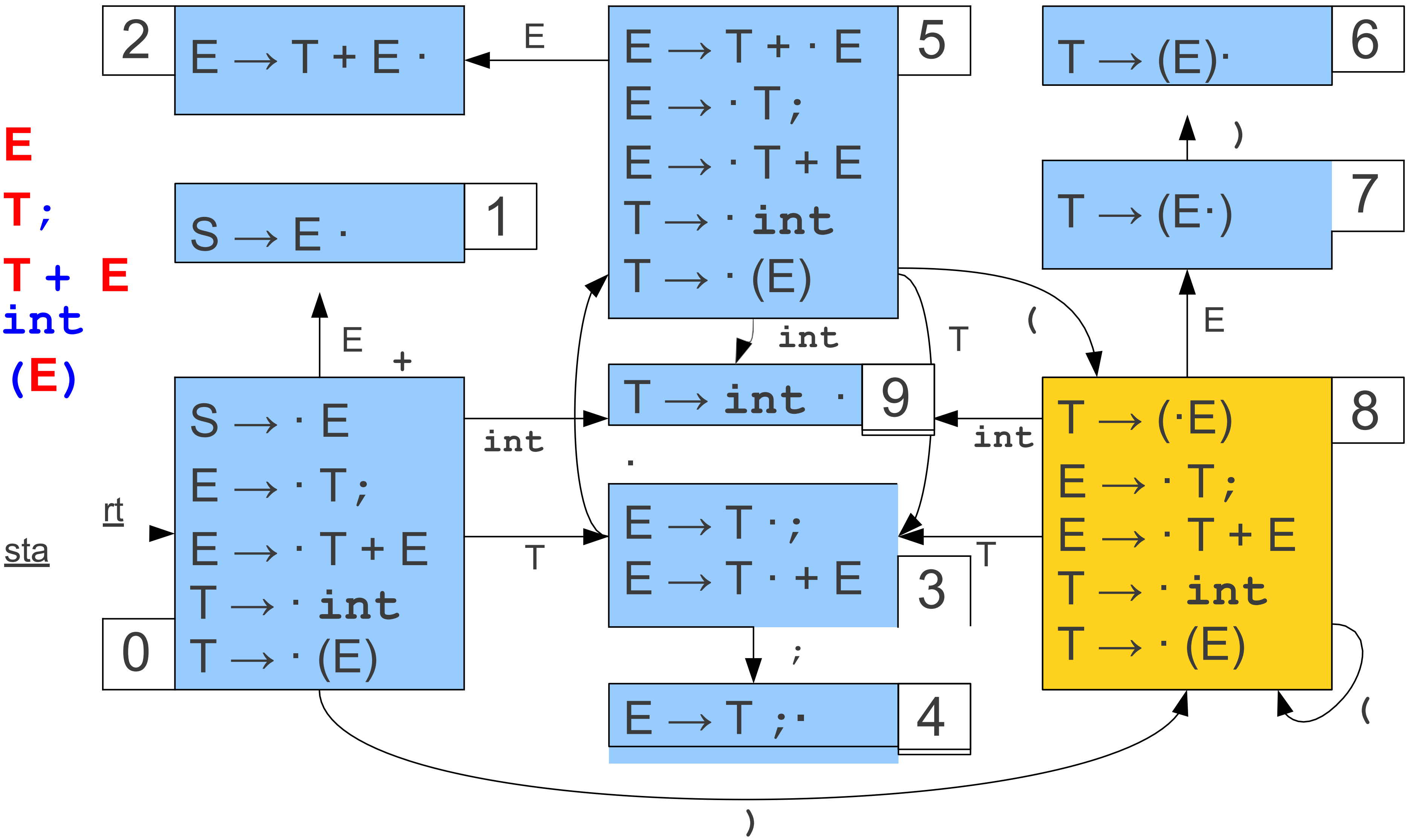
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(	E
0	3	5	8

)	;

LR(0) Parsing

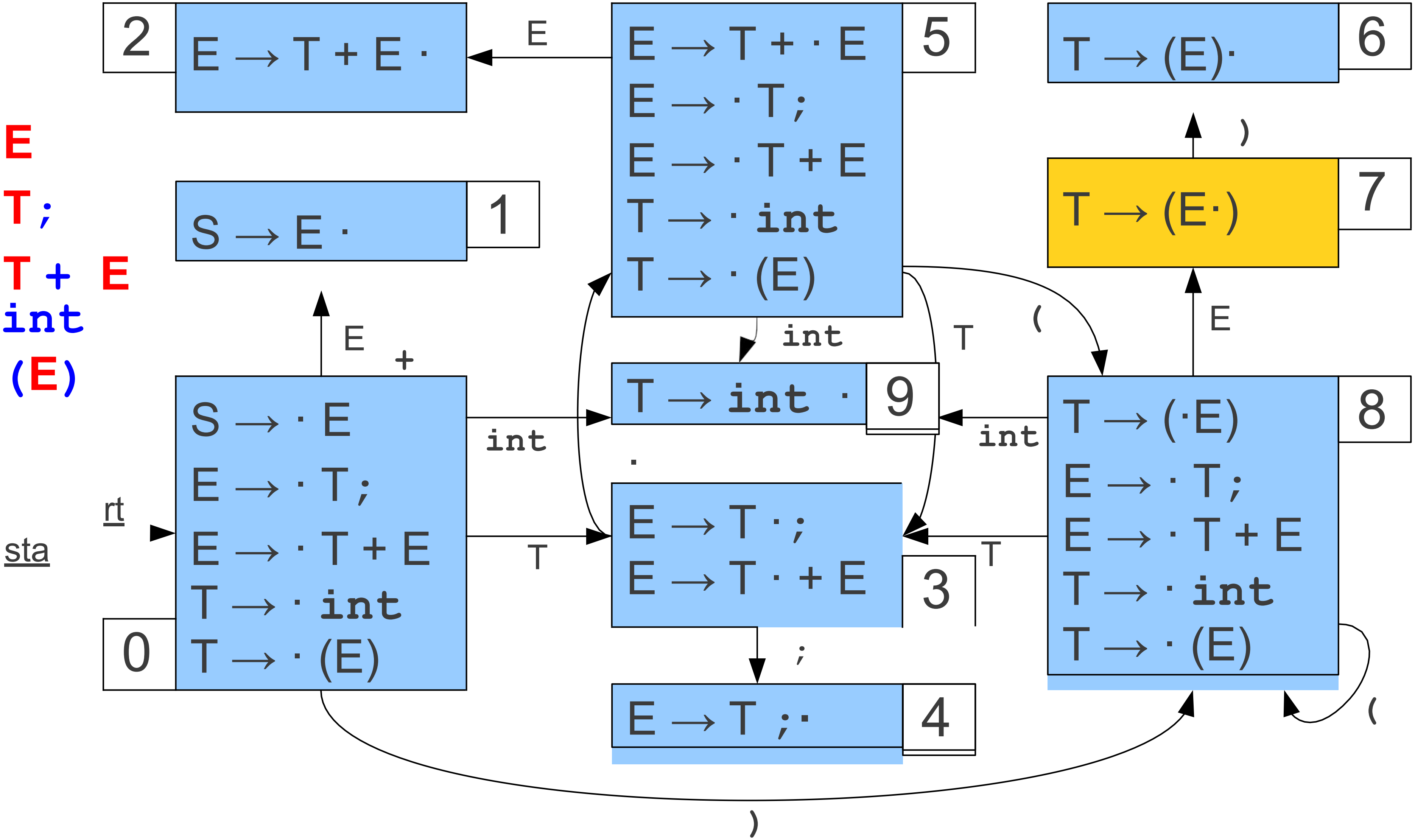
S → E

E → T;

E → T + E

T → int

T → (E)



T	+	(	E

)	;

0	3	5	8

LR(0) Parsing

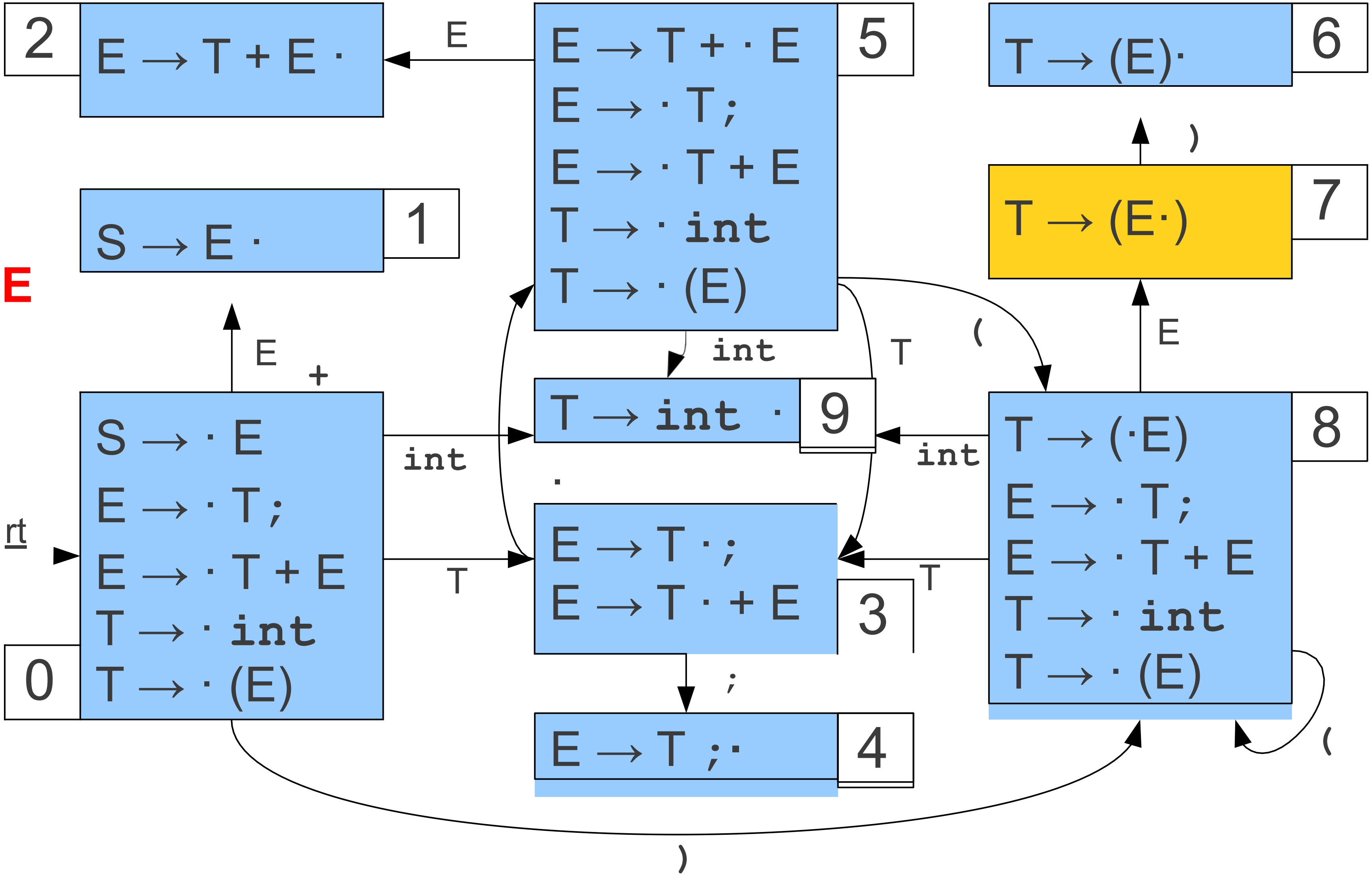
S → E

E → T;

E → T + E

T → int

T → (E)



0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

1
S → E ·

2
E → T + E ·

3
E → T · ;
E → T · + E

4
E → T ;·

5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6
T → (E)·

7
T → (E·)

8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

9
T → int ·

T + (E |) ;

0 3 5 8 7

LR(0) Parsing

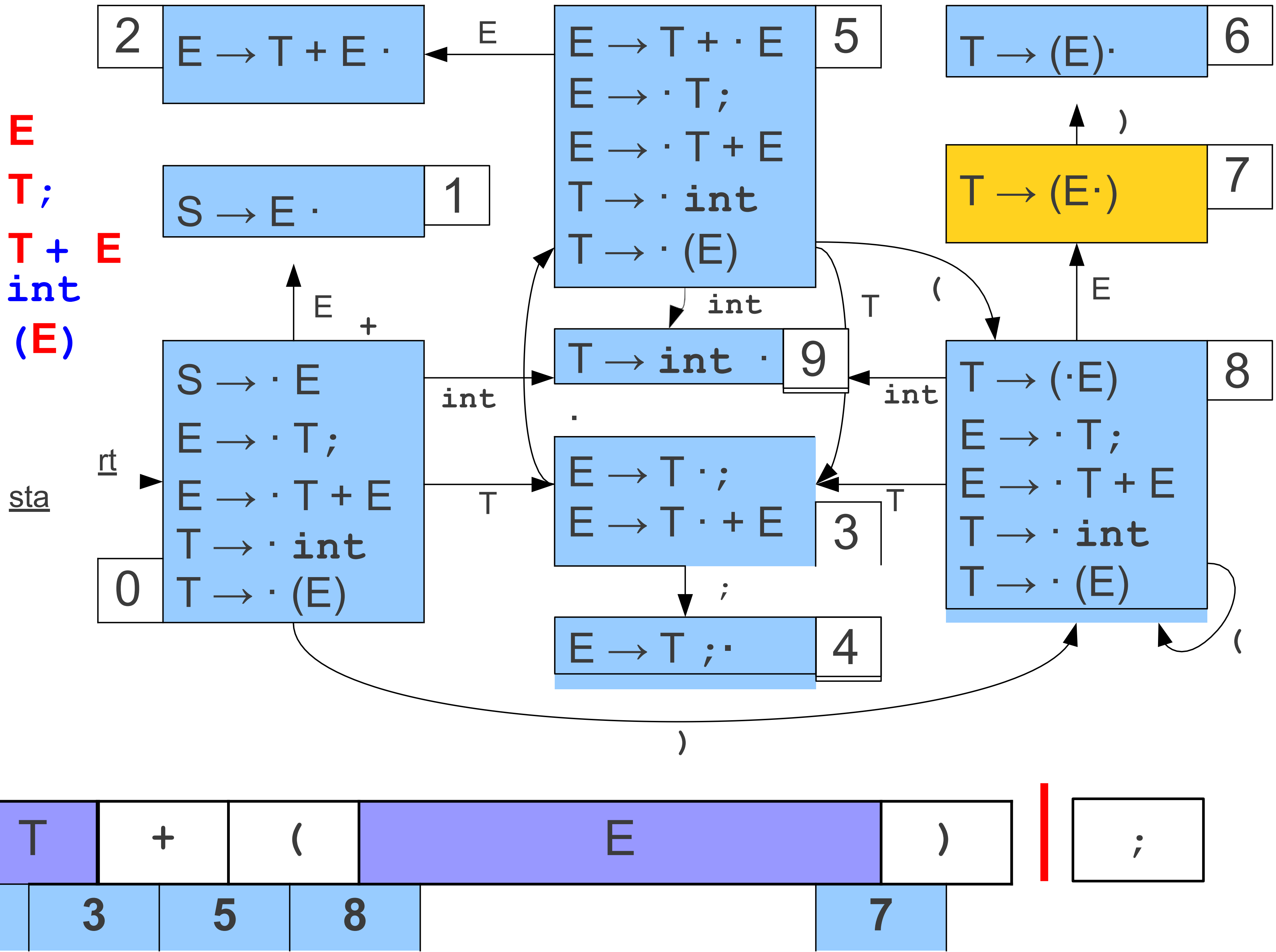
S → E

E → T;

E → T + E

T → int

T → (E)



LR(0) Parsing

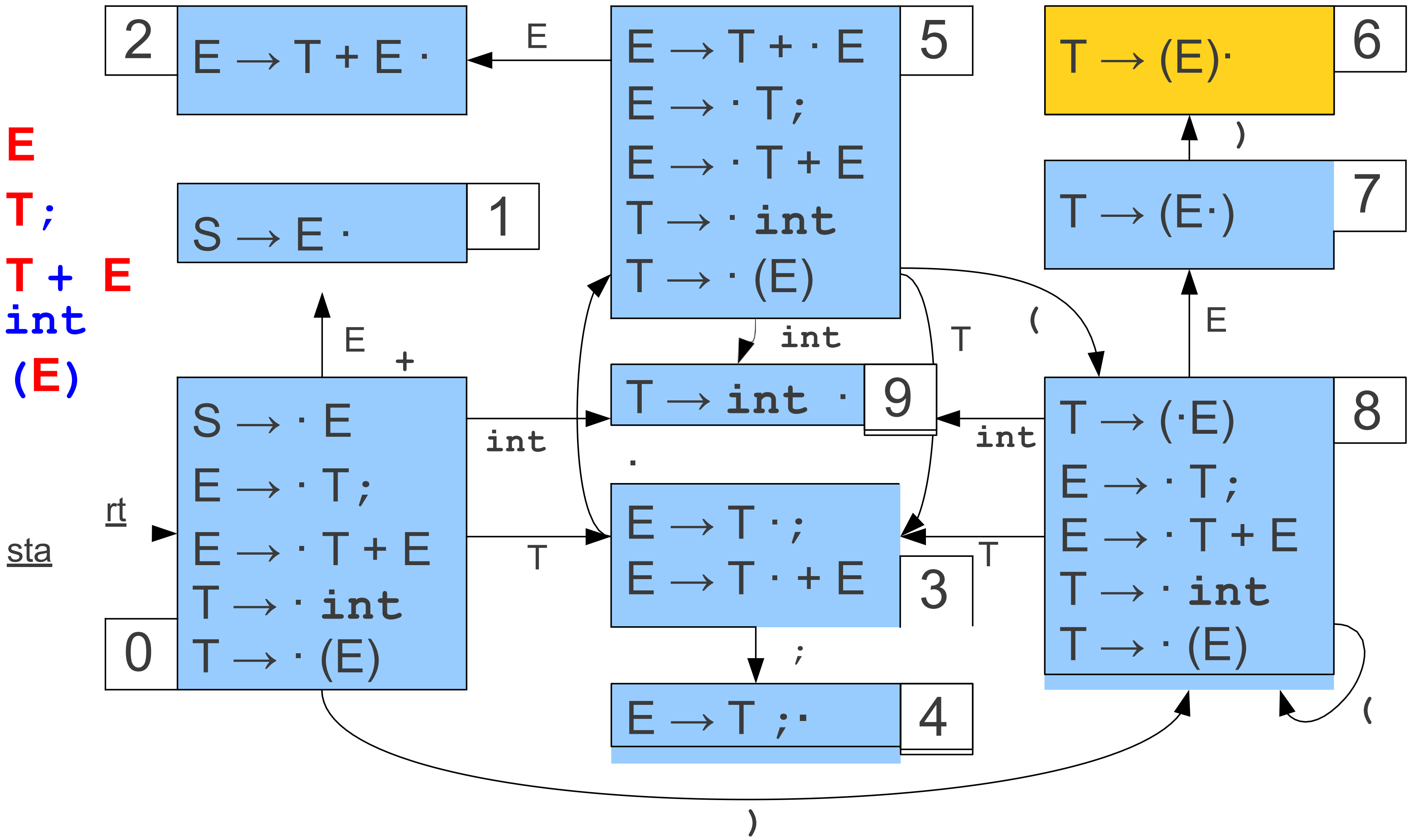
S → E

E → T;

E → T + E

T → int

T → (E)



T | + | (| E |) | ;

0 | 3 | 5 | 8 | 7

LR(0) Parsing

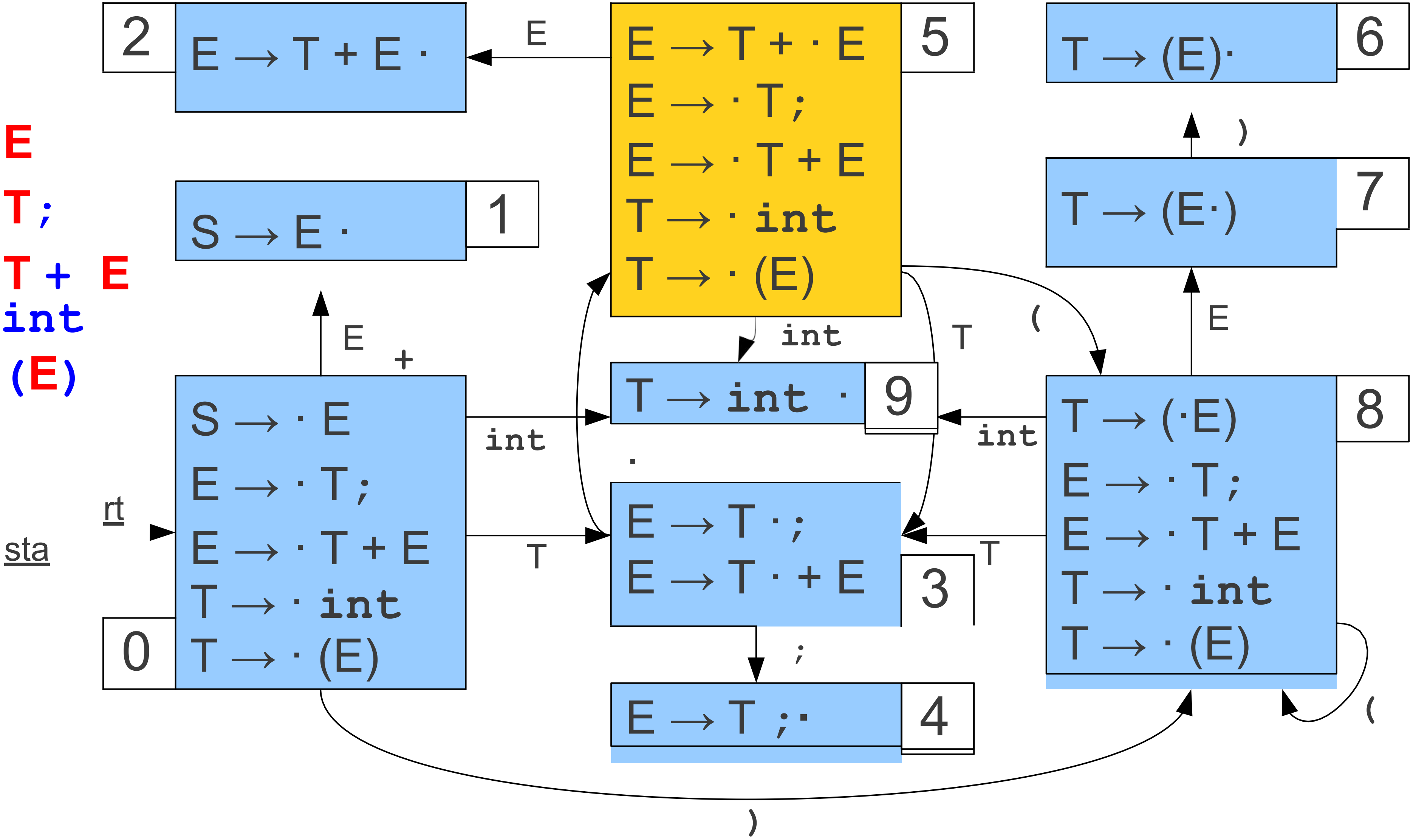
S → E

E → T;

E → T + E

T → int

T → (E)



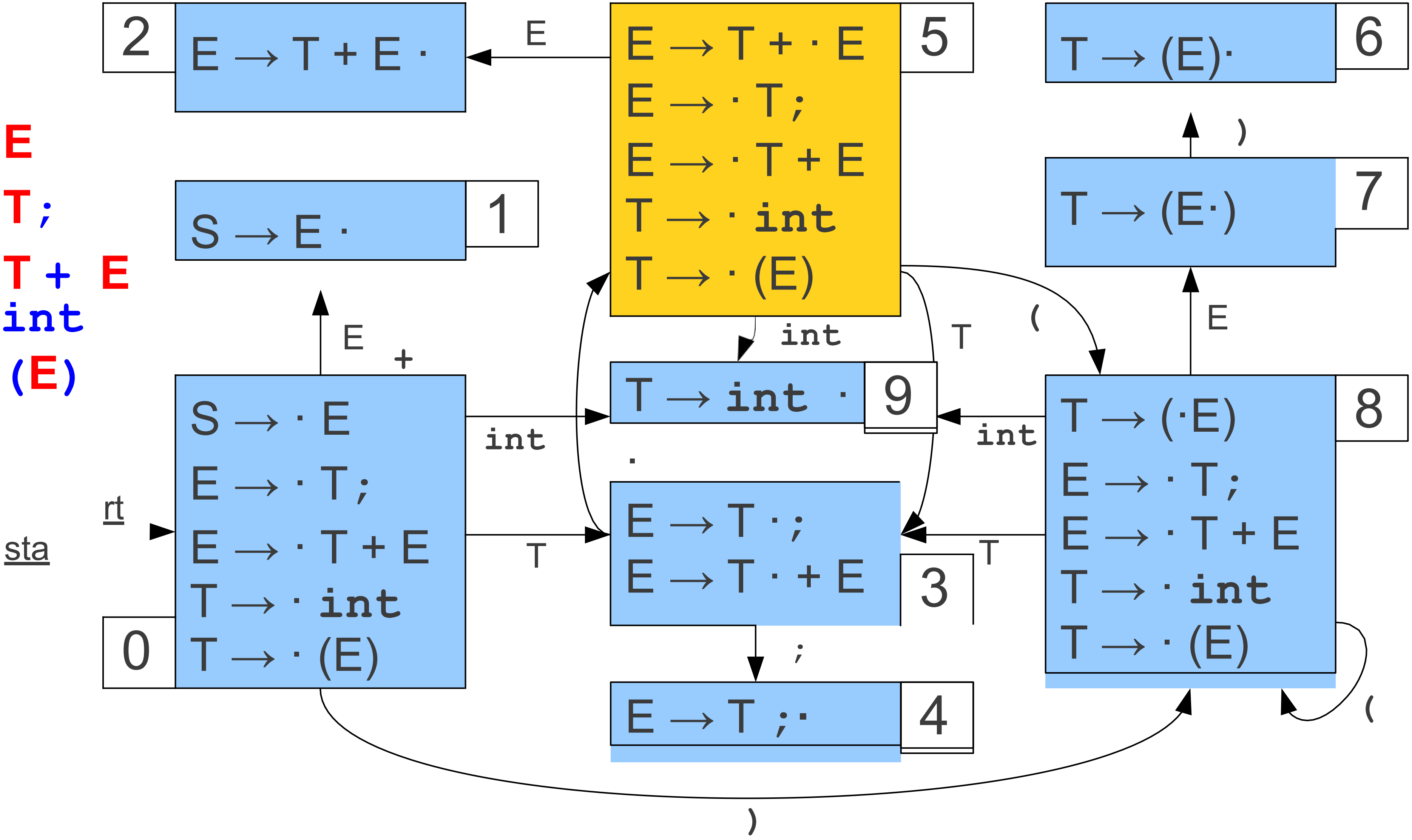
T +

0 3 5

;

LR(0) Parsing

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



2: E → T + E ·

5: E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6: T → (E)·

1: S → E ·

7: T → (E·)

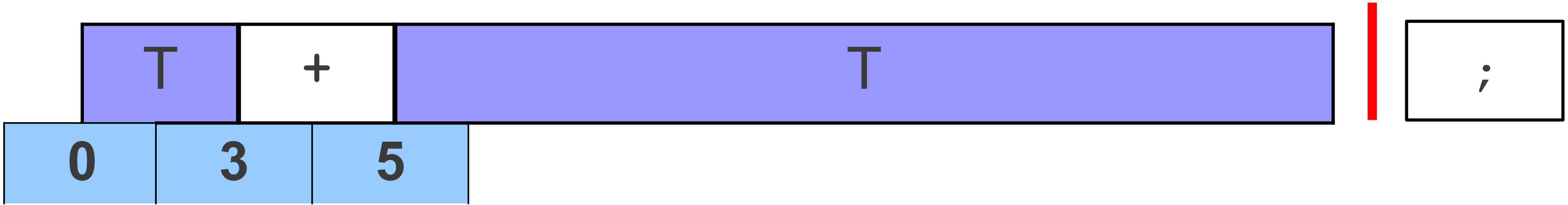
9: T → int ·

0: S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

3: E → T · ;
E → T · + E

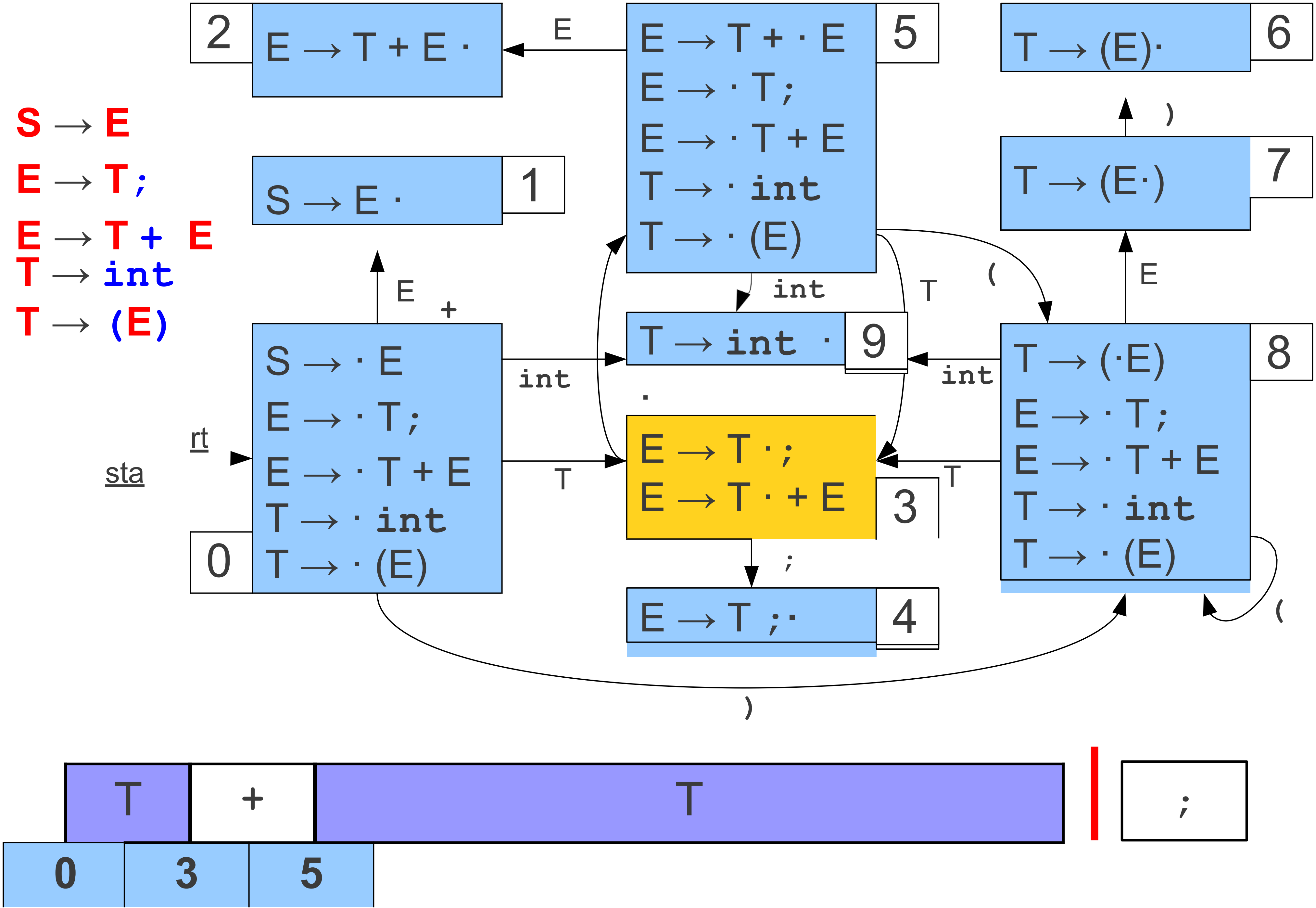
8: T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

4: E → T ;·



LR(0) Parsing

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



2
E → T + E ·

1
S → E ·

0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

9
T → int ·

3
E → T · ;
E → T · + E

4
E → T ;·

6
T → (E)·

7
T → (E·)

8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

LR(0) Parsing

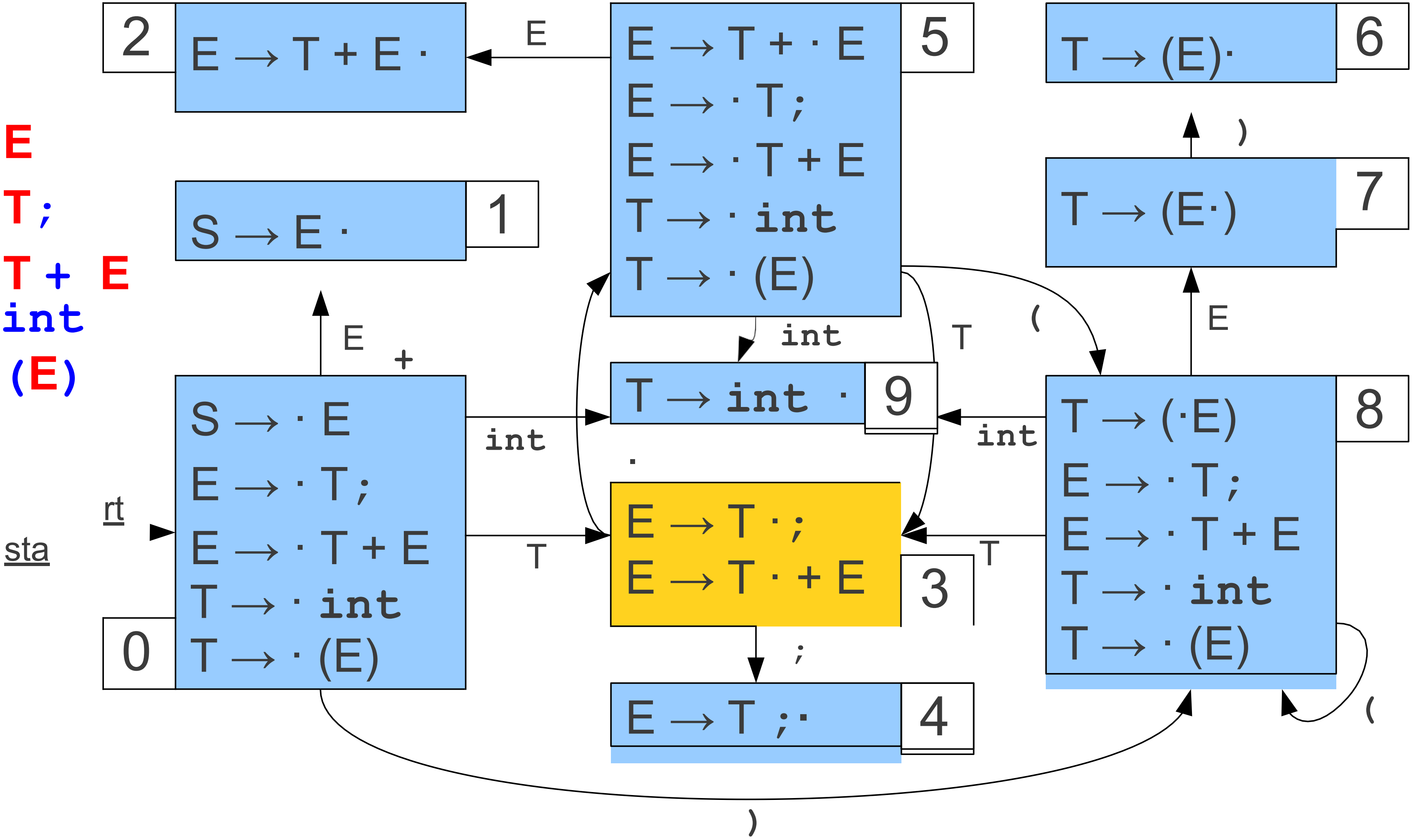
S → E

E → T;

E → T + E

T → int

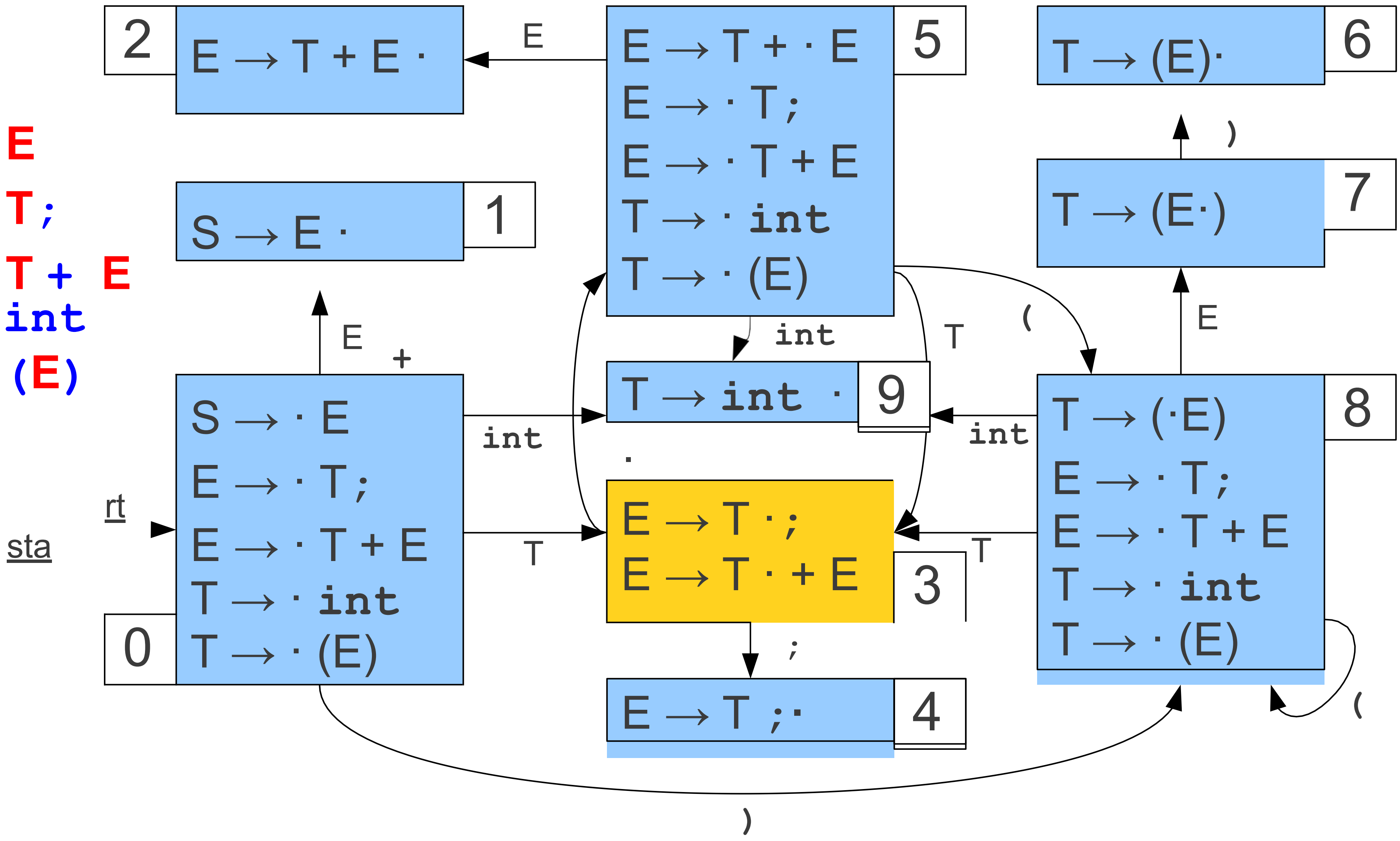
T → (E)



T	+	T		;
0	3	5	3	

LR(0) Parsing

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



0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

1
S → E ·

2
E → T + E ·

3
E → T · ;
E → T · + E

4
E → T ;·

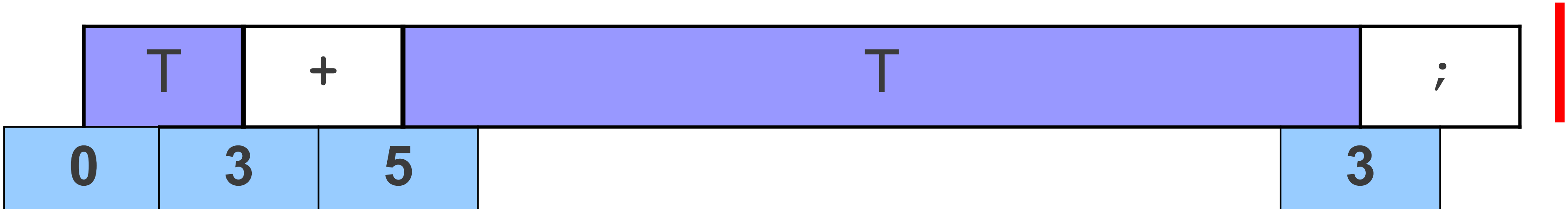
5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6
T → (E)·

7
T → (E·)

8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

9
T → int ·



LR(0) Parsing

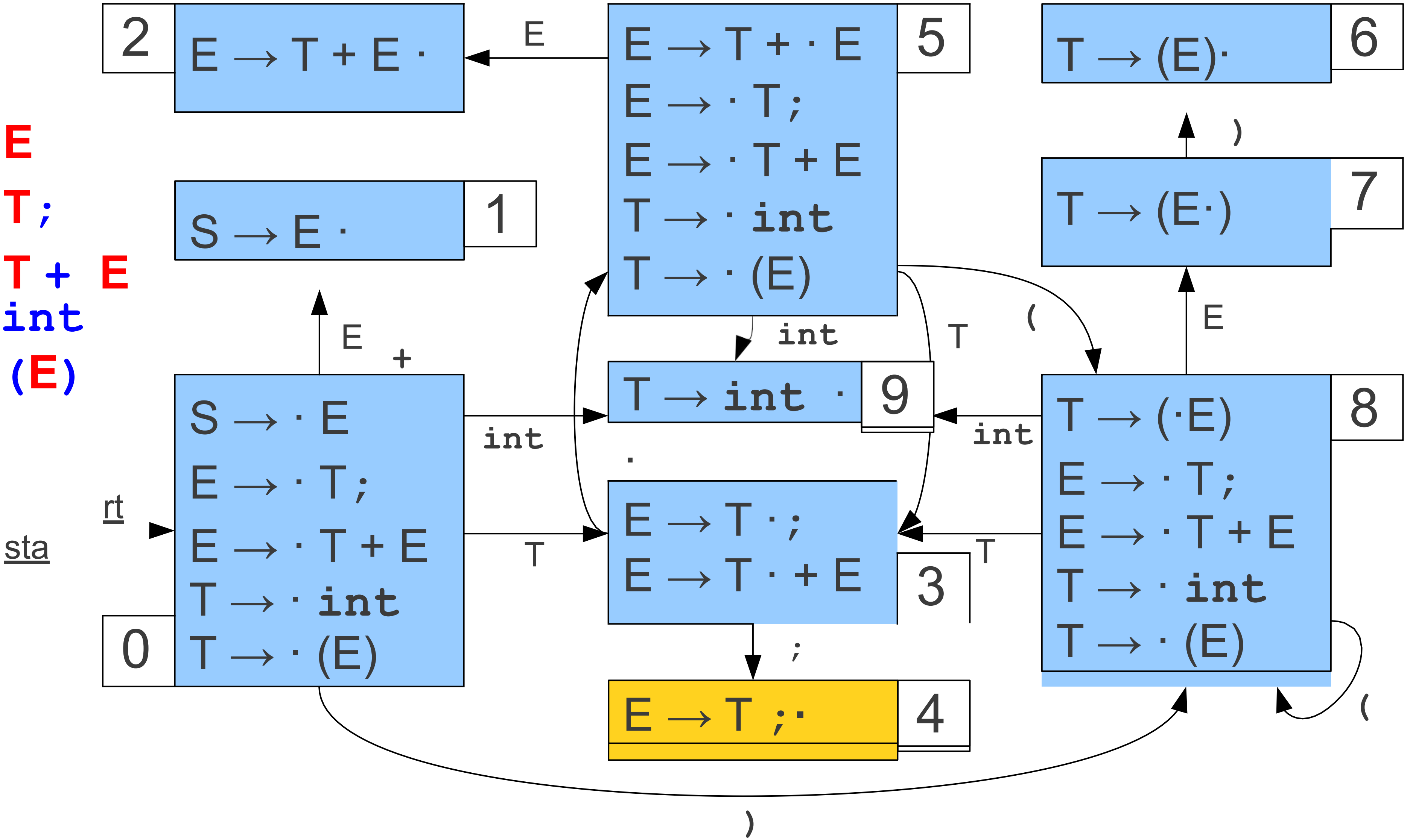
S → E

E → T;

E → T + E

T → int

T → (E)



2: E → T + E ·

1: S → E ·

0:
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

5:
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

9: T → int ·

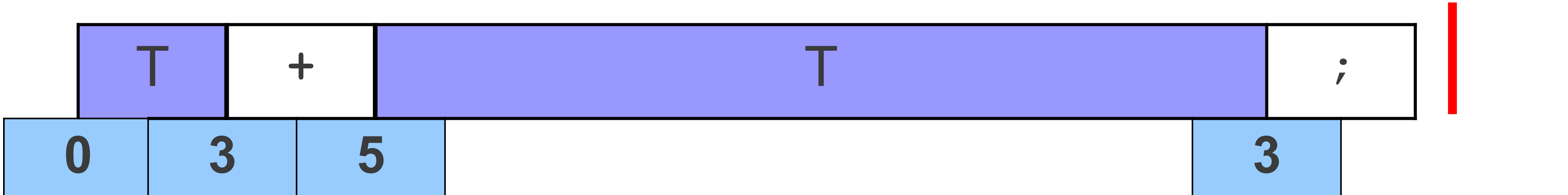
3:
E → T · ;
E → T · + E

4: E → T ; ·

6: T → (E)·

7: T → (E·)

8:
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)



LR(0) Parsing

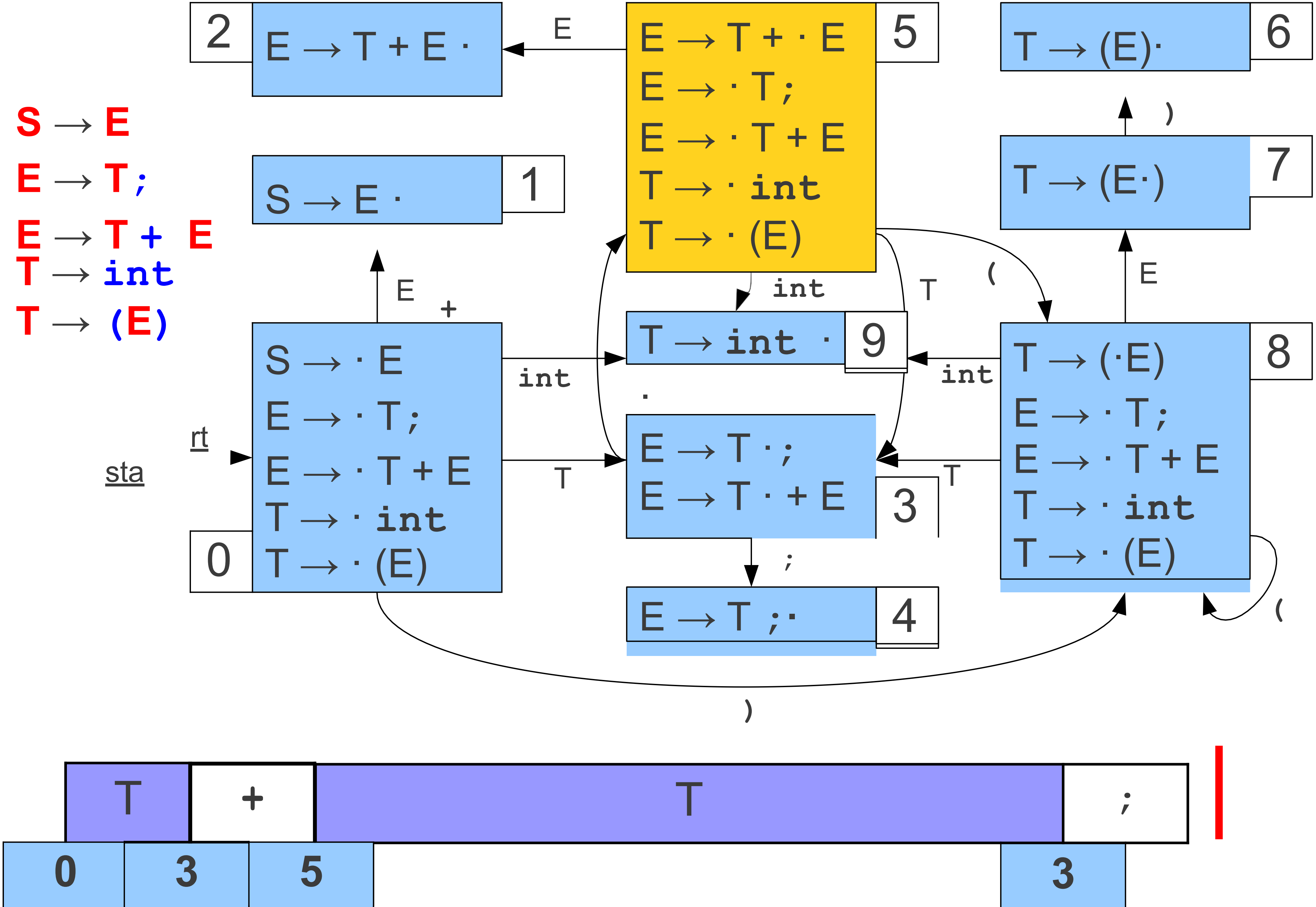
S → E

E → T;

E → T + E

T → int

T → (E)



2: E → T + E ·

5: E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6: T → (E)·

1: S → E ·

7: T → (E·)

0: S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

9: T → int ·

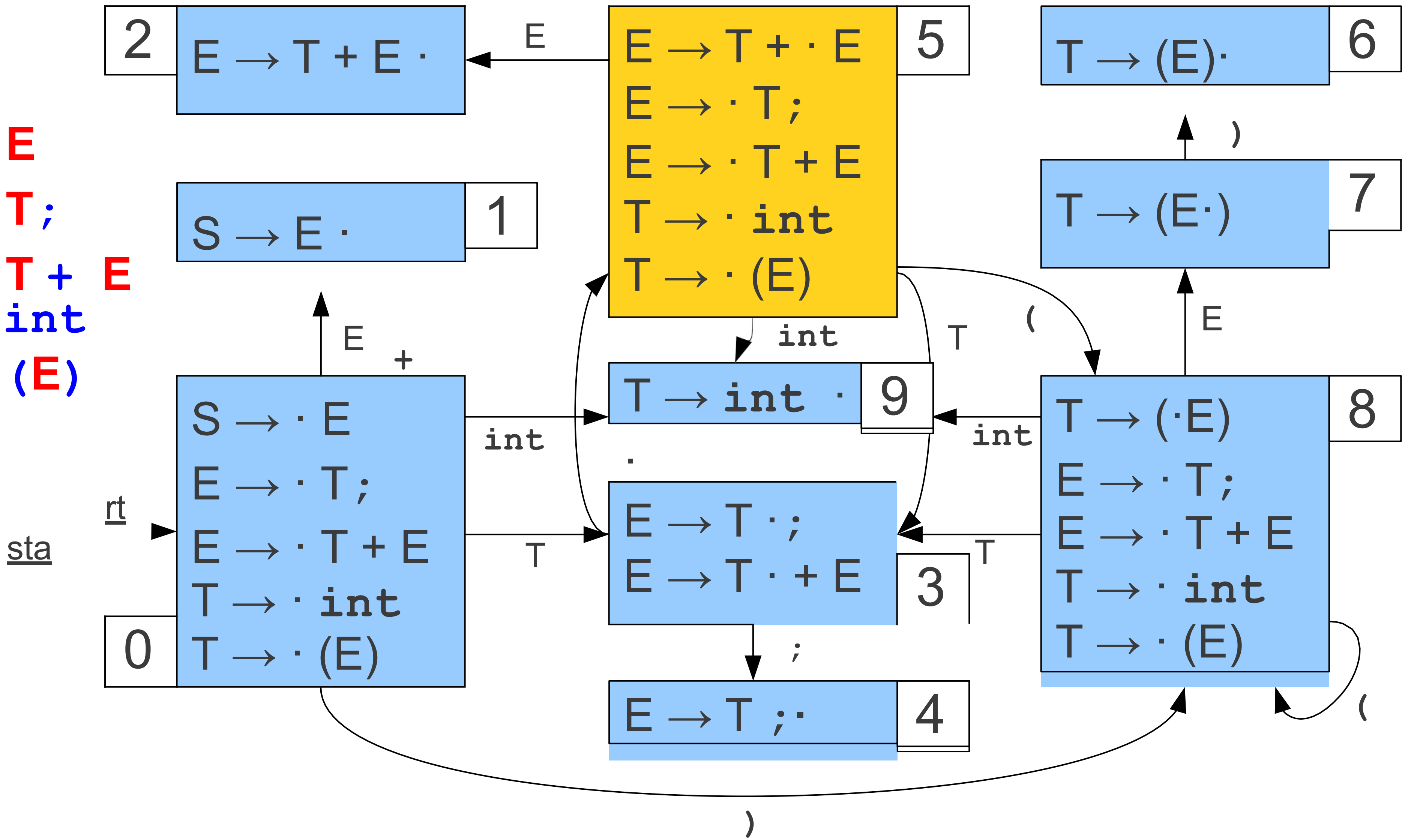
3: E → T · ;
E → T · + E

8: T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

4: E → T ;·

LR(0) Parsing

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



0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

1
S → E ·

2
E → T + E ·

3
E → T · ;
E → T · + E

4
E → T ;·

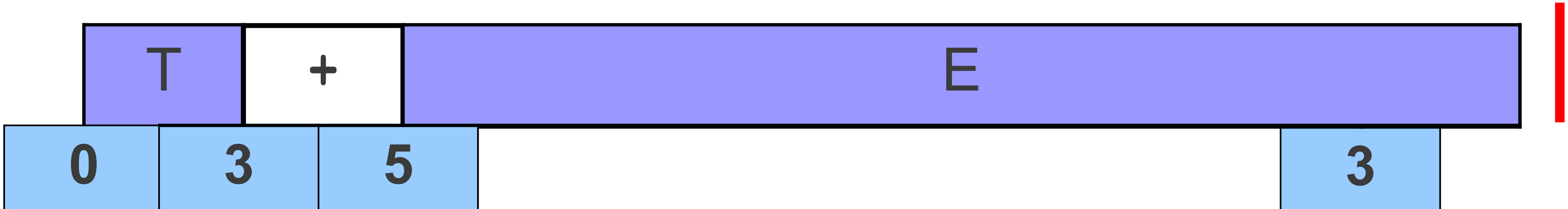
5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6
T → (E)·

7
T → (E·)

8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

9
T → int ·



LR(0) Parsing

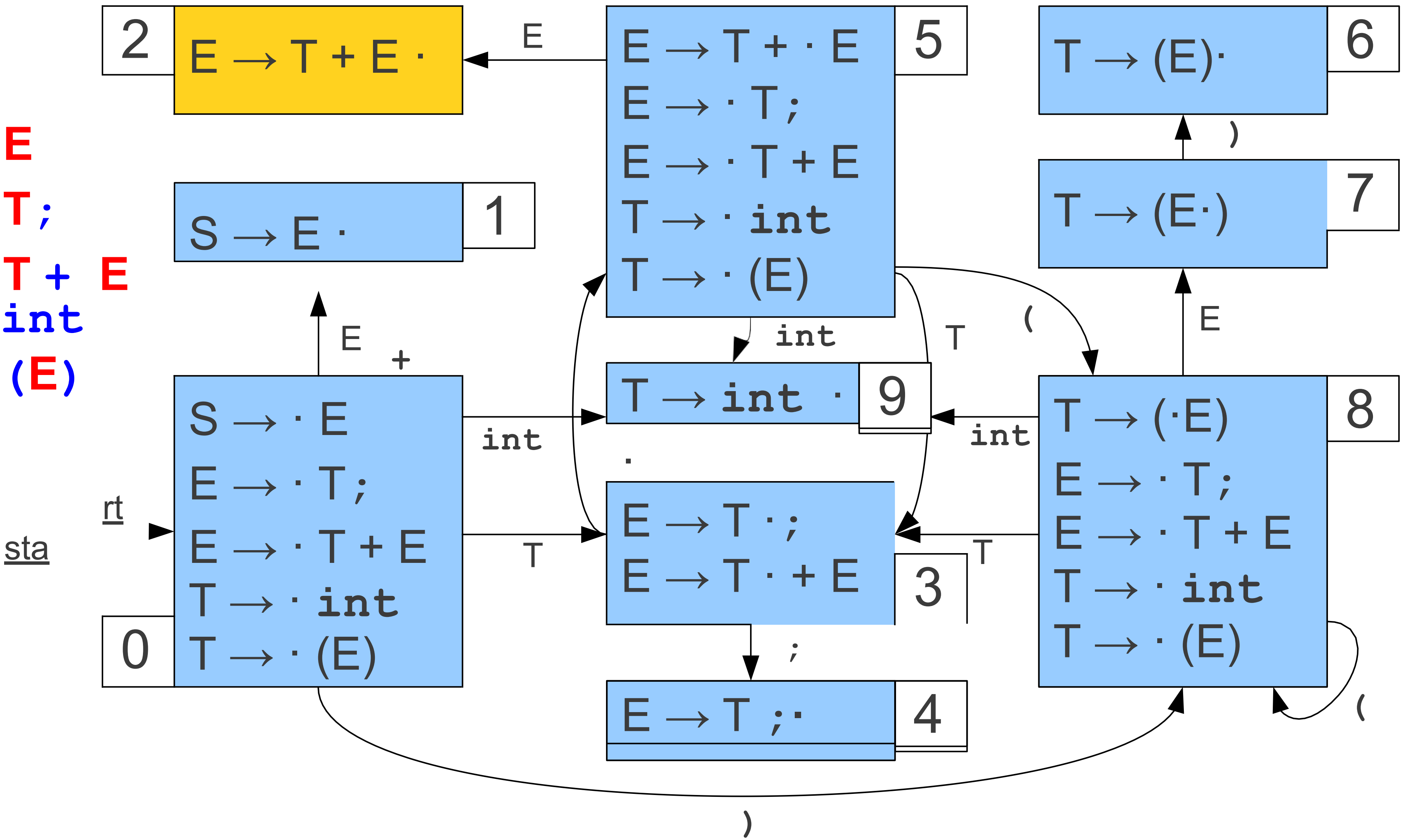
S → E

E → T;

E → T + E

T → int

T → (E)



2: E → T + E ·

5:
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6: T → (E)·

7: T → (E·)

1: S → E ·

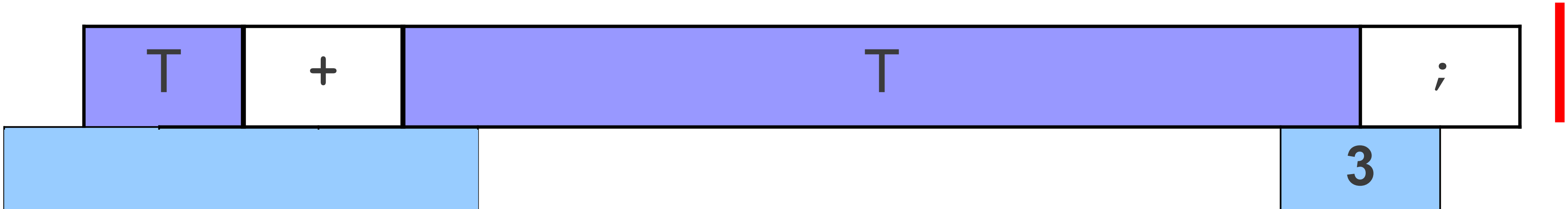
0:
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

9: T → int ·

3:
E → T · ;
E → T · + E

8:
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

4: E → T ;·



T	+	T	;

3

LR(0) Parsing

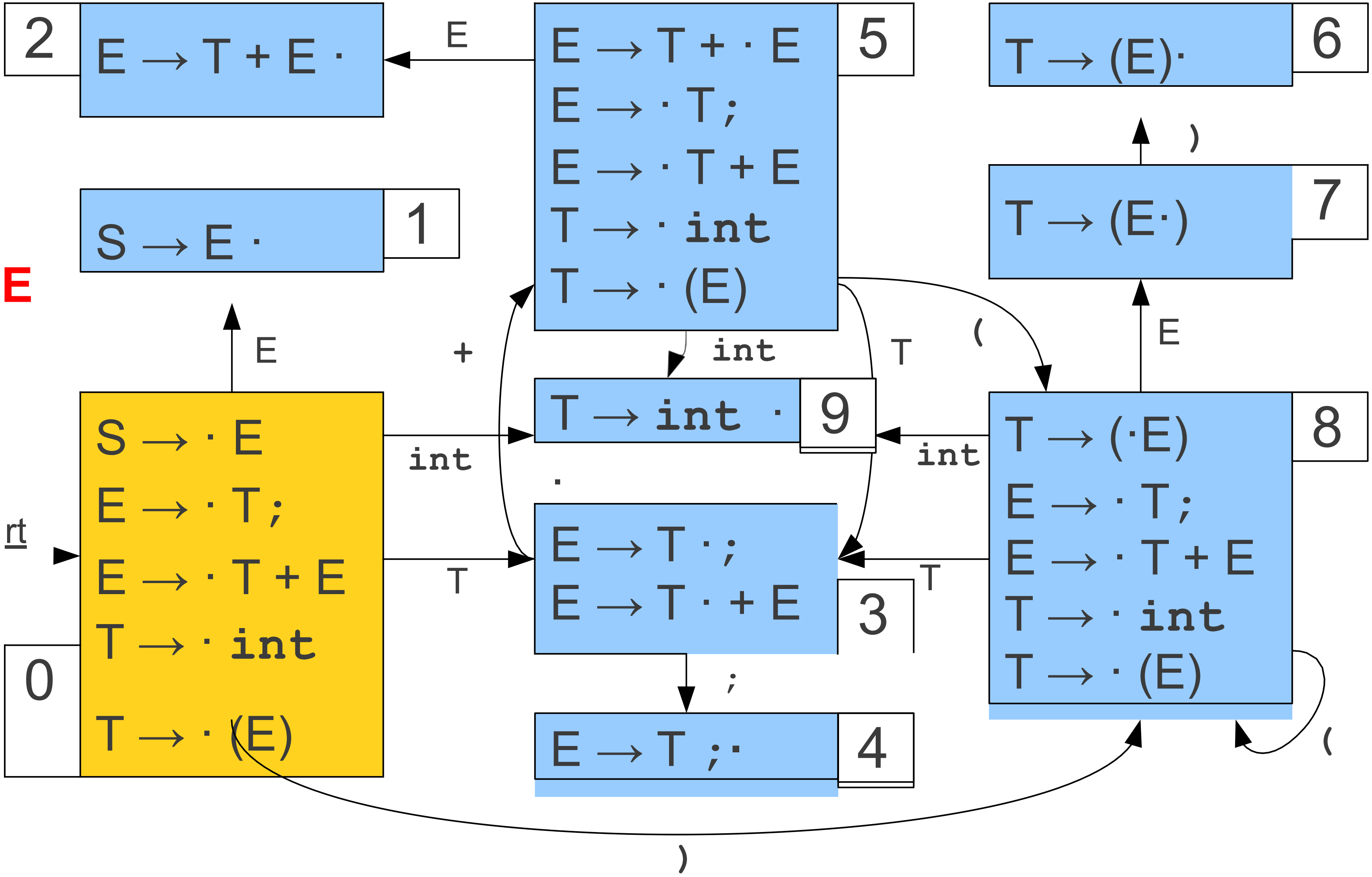
S → E

E → T;

E → T + E

T → int

T → (E)



0

LR(0) Parsing

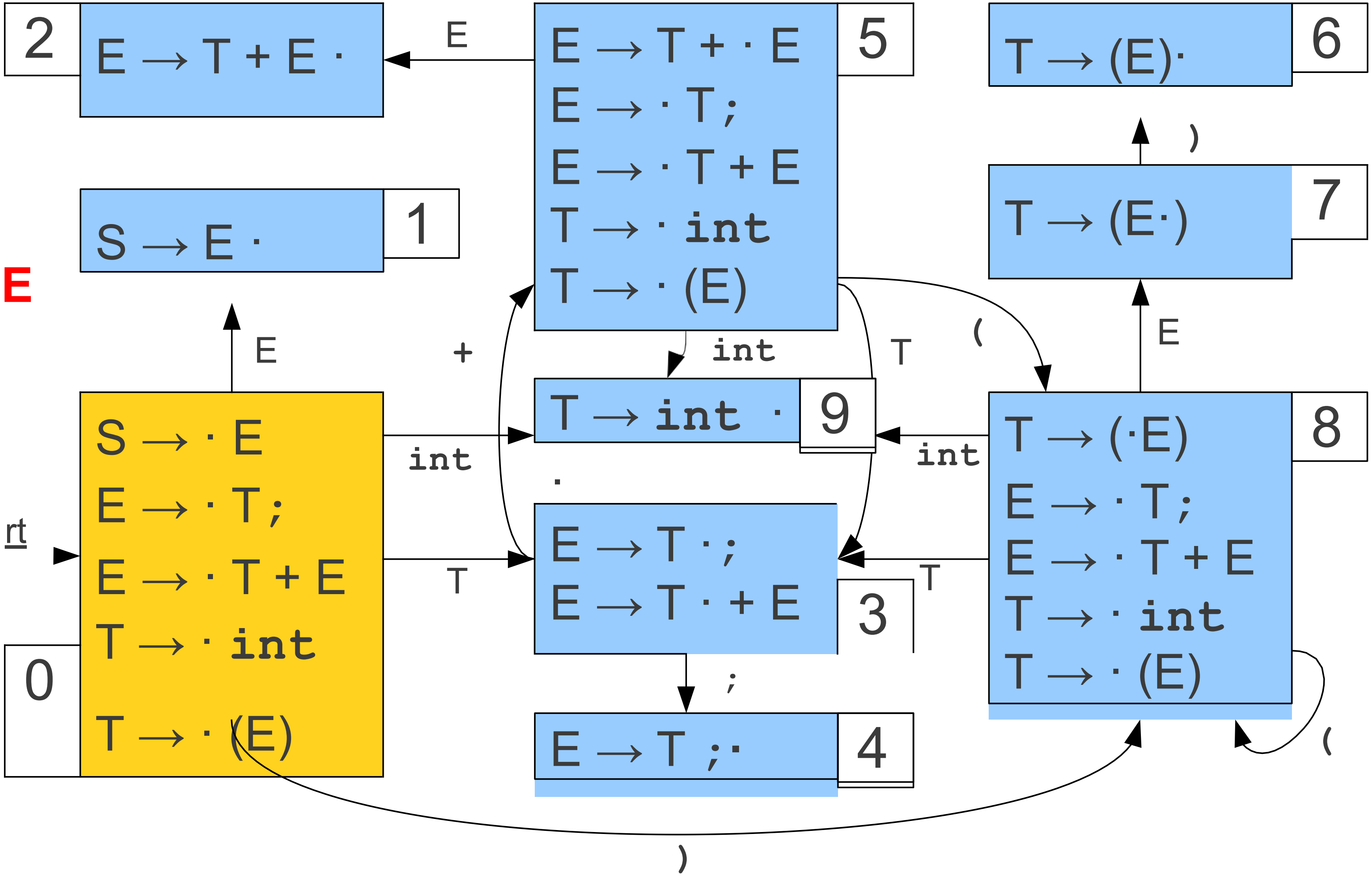
S → E

E → T;

E → T + E

T → int

T → (E)



0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

1
S → E ·

2
E → T + E ·

3
E → T · ;
E → T · + E

4
E → T ; ·

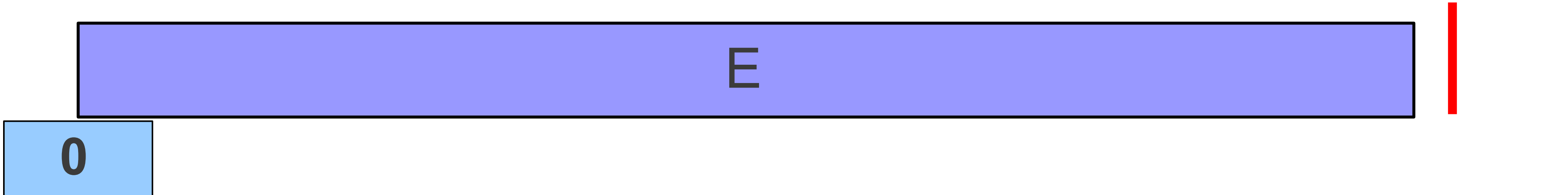
5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6
T → (E)·

7
T → (E·)

8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

9
T → int ·



LR(0) Parsing

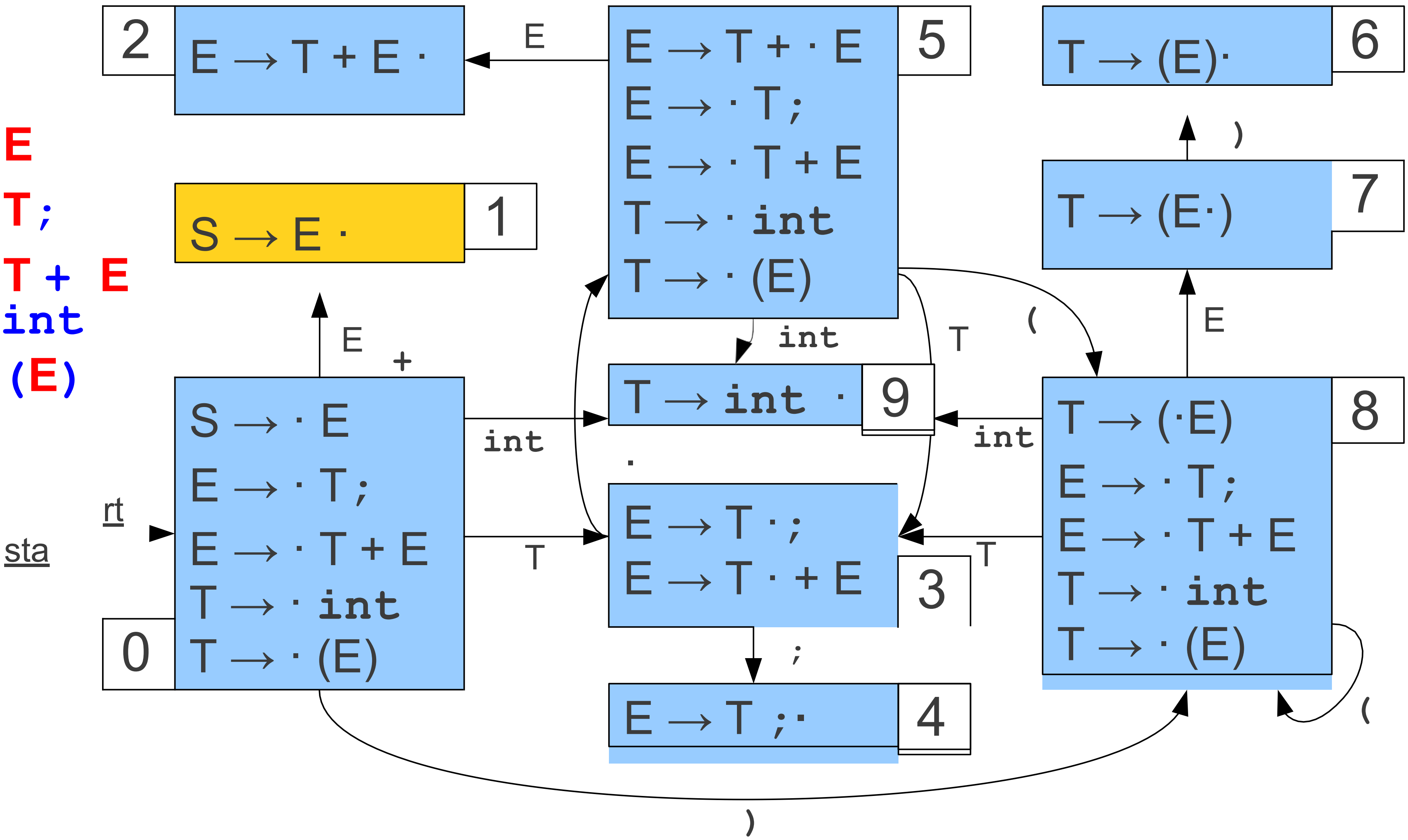
S → E

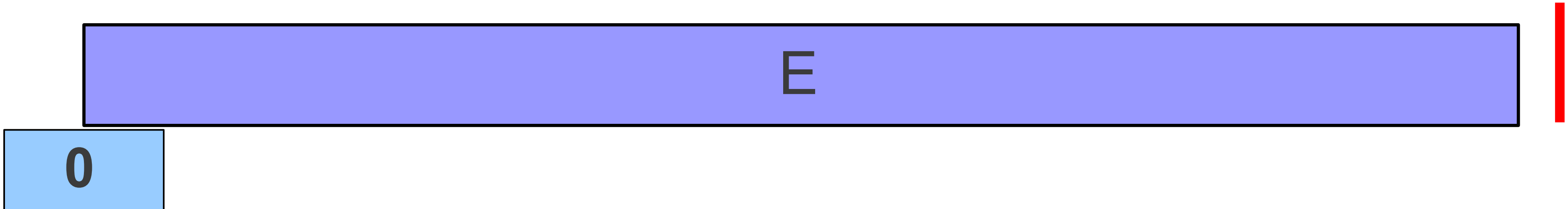
E → T;

E → T + E

T → int

T → (E)





Representing the Automaton

- LR(0) parsers are usually represented via two tables: an **action** table and a **goto** table.
- The **action** table maps each state to an action:
  - **shift**, which shifts the next terminal, and
  - **reduce** **A** → ***ω***, which performs reduction **A** → ***ω***.
  - Any state of the form **A** → ***ω*** · does that reduction; everything else shifts.
- The **goto** table maps state/symbol pairs to a next state.
  - This is just the transition table for the automaton.

Building LR(0) Tables

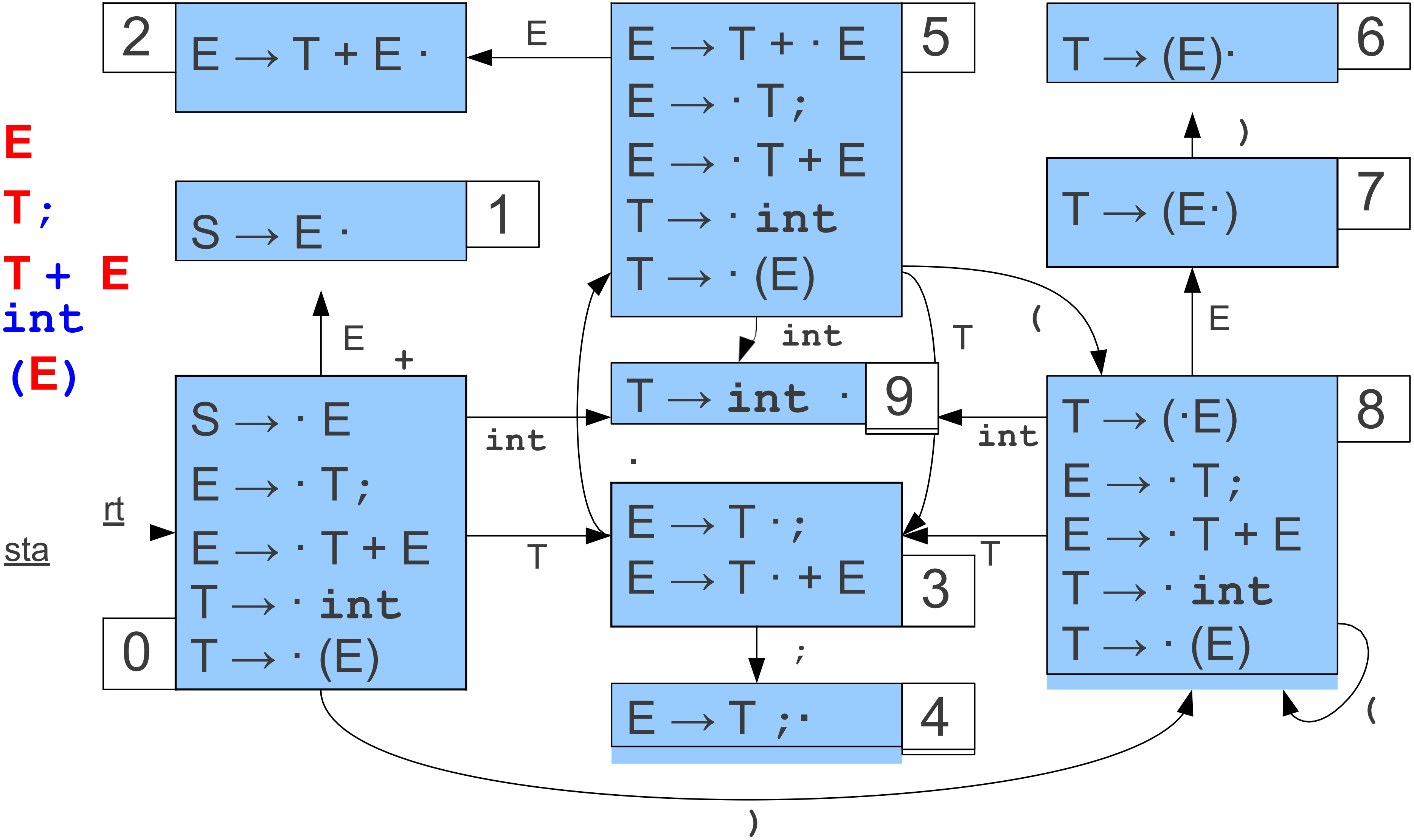
S → E

E → T;

E → T + E

T → int

T → (E)



0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

1
S → E ·

2
E → T + E ·

3
E → T ·;
E → T · + E

4
E → T ;·

5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6
T → (E)·

7
T → (E·)

8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

9
T → int ·

LR(0) Tables

	int	+	;	(	)	E	T	Action
0	9			8		1	3	Shift
1								Accept
2								Reduce **E** → **T + E**
3		5	4					Shift
4								Reduce **E** → **T ;**
5	9			8		2	3	Shift
6								Reduce **T** → **(E)**
7					6			Shift
8	9			8		7	3	Shift
9								Reduce **T** → **int**

LR(0) Tables

	int	+	;	(	)	E	T
0	S9			S8		G1	G3
1	AC	AC	AC	AC	AC	AC	AC
2	R3	R3	R3	R3	R3	R3	R3
3		S5	S4				
4	R2	R2	R2	R2	R2	R2	R2
5	S9			S8		G2	G3
6	R5	R5	R5	R5	R5	R5	R5
7					S6		
8	S9			S8		G7	G3
9	R4	R4	R4	R4	R4	R4	R4

The LR(0) Algorithm

- Maintain a stack of (symbol, state) pairs, which is initially (?, 1) for some dummy symbol ?.
- While the stack is not empty:
  - Let **state** be the top state. If
  - **action[state]** is **shift**:
    - Let t be the next symbol in the input. Push
    - (t, **goto[state, t]**) atop the stack.
  - If **action[state]** is **reduce A** → ***ω***:
    - Remove |***ω***| symbols from the top of the stack. Let
    - **top-state** be the state on top of the stack. Push (**A**,
    - **goto[top-state, A]**) atop the stack.
  - Otherwise, report an error.

The Limits of LR(0)

A Non-LR(0) Grammar

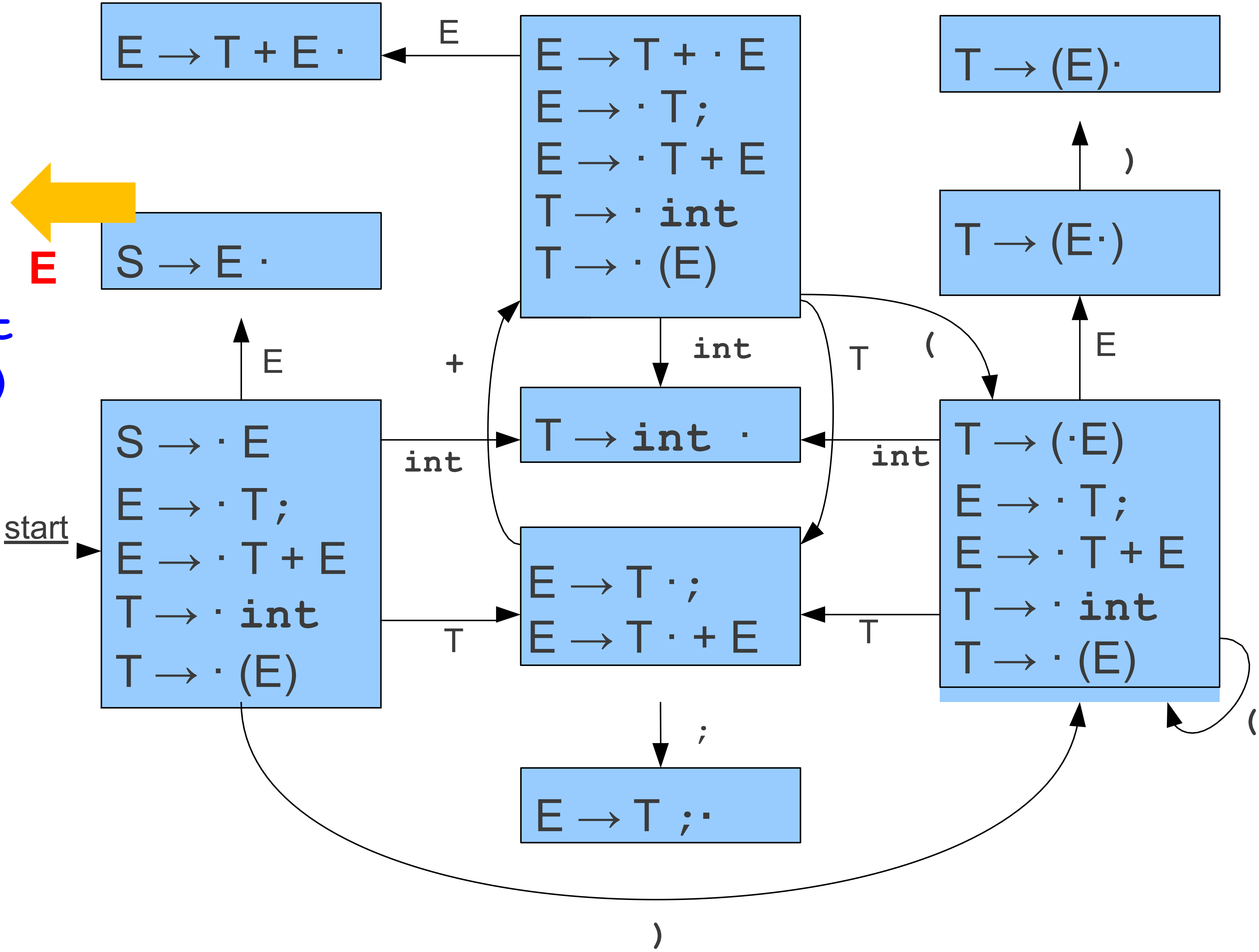
S → E

E → T;

E → T + E

T → int

T → (E)



E → T + E ·

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

T → (E)·

S → E ·

T → (E·)

T → int ·

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

E → T ·;
E → T · + E

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

E → T ;·

A Non-LR(0) Grammar

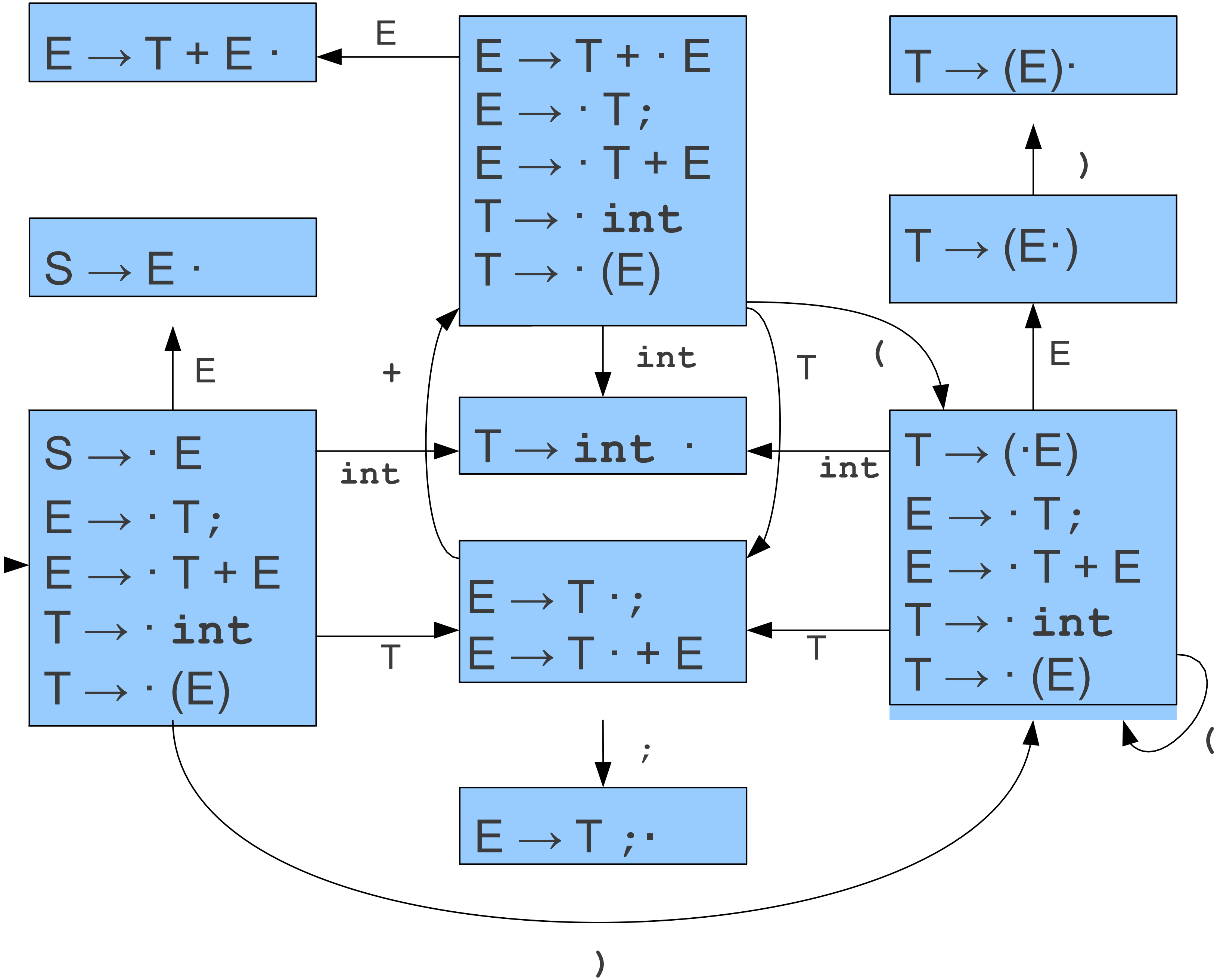
S → E

E → T

E → T + E

T → **int**

T → (E)



E → T + E ·

S → E ·

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

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

T → **int** ·

E → T · ;
E → T · + E

E → T ; ·

T → (E)·

T → (E·)

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

A Non-LR(0) Grammar

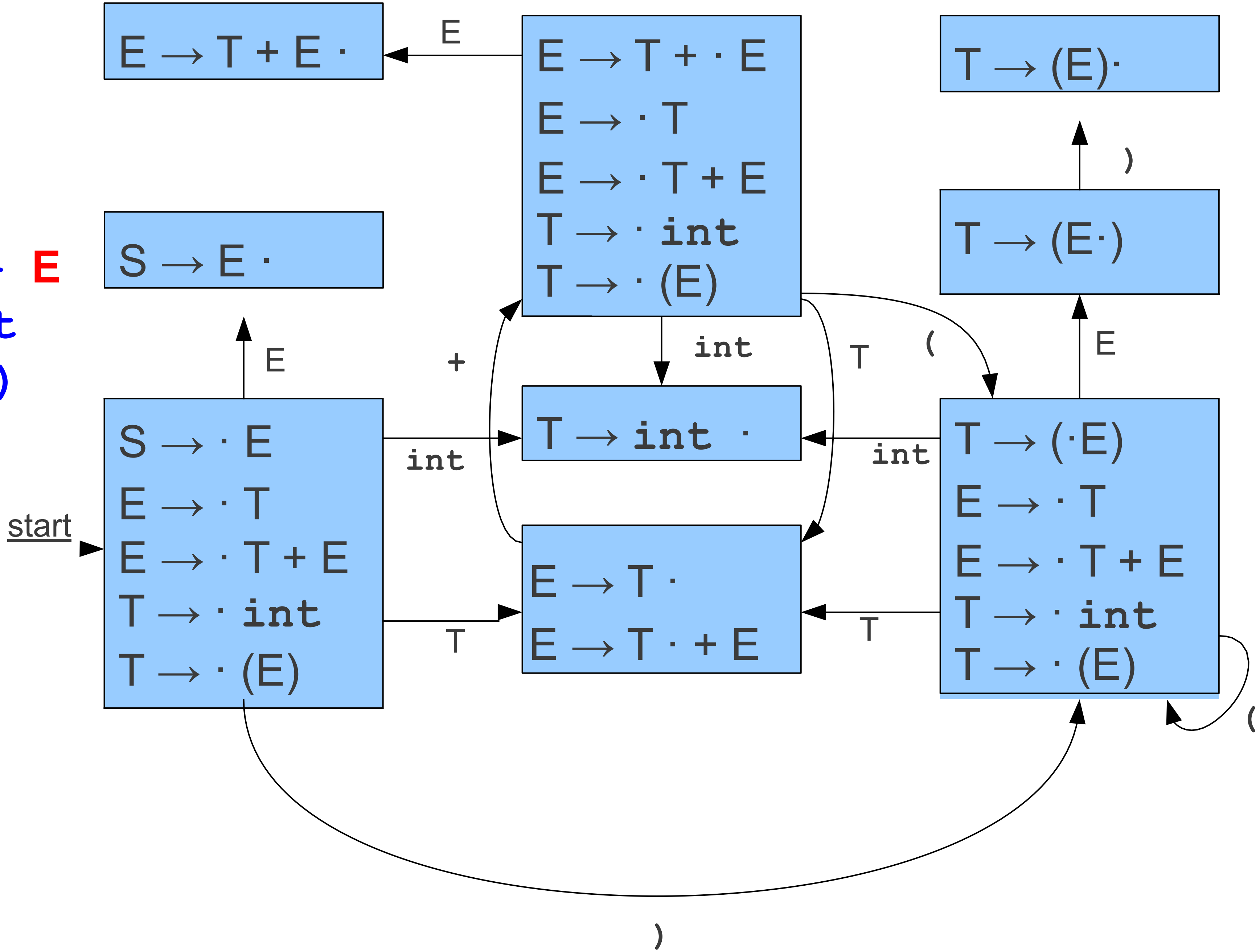
S → E

E → T

E → T + E

T → int

T → (E)



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

S → E ·

E → T + E ·

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

T → int ·

E → T ·
E → T · + E

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

T → (E·)

T → (E)·

A Non-LR(0) Grammar

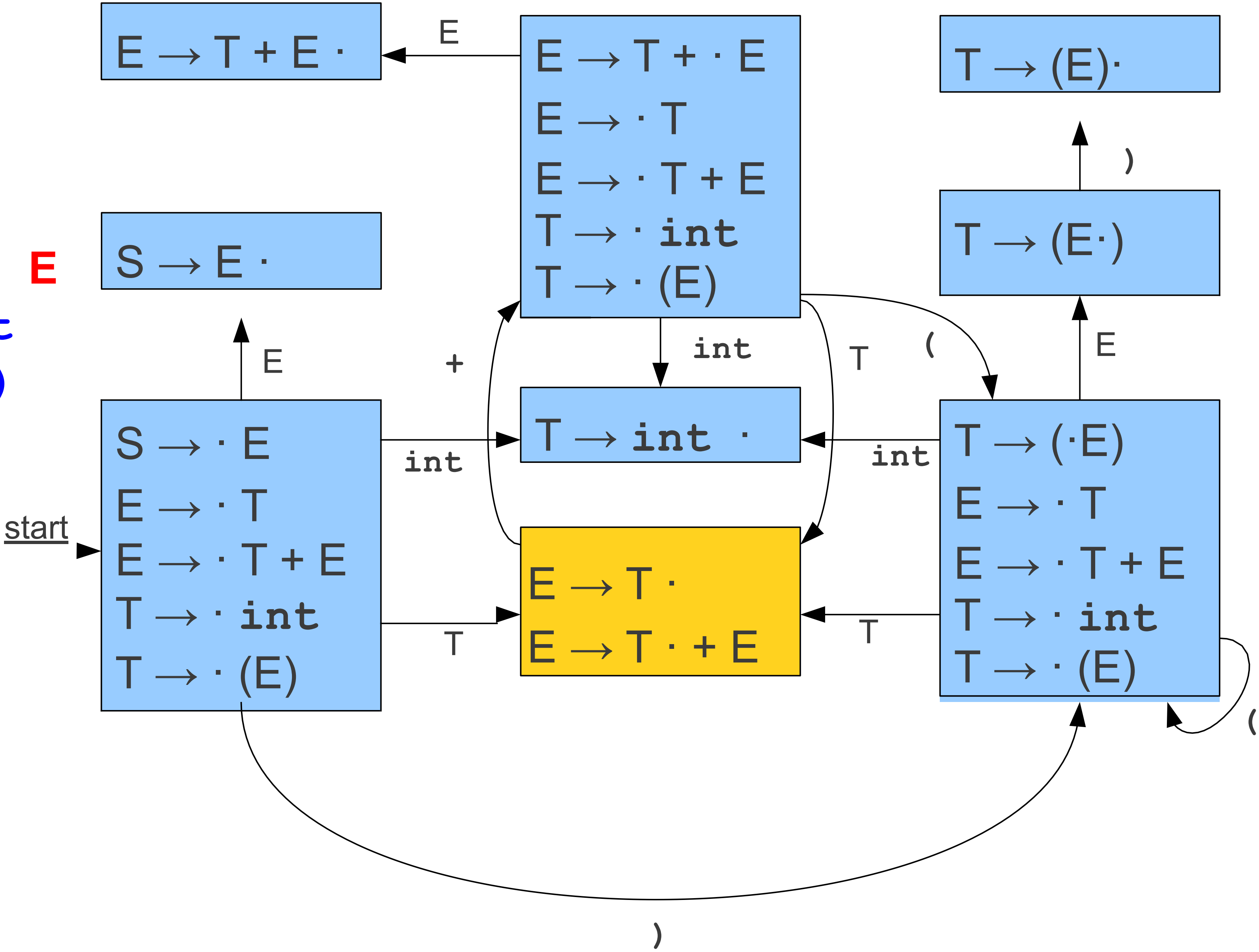
S → E

E → T

E → T + E

T → int

T → (E)



E → T + E ·

S → E ·

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

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

T → int ·

E → T ·
E → T · + E

T → (E)·

T → (E·)

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

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.

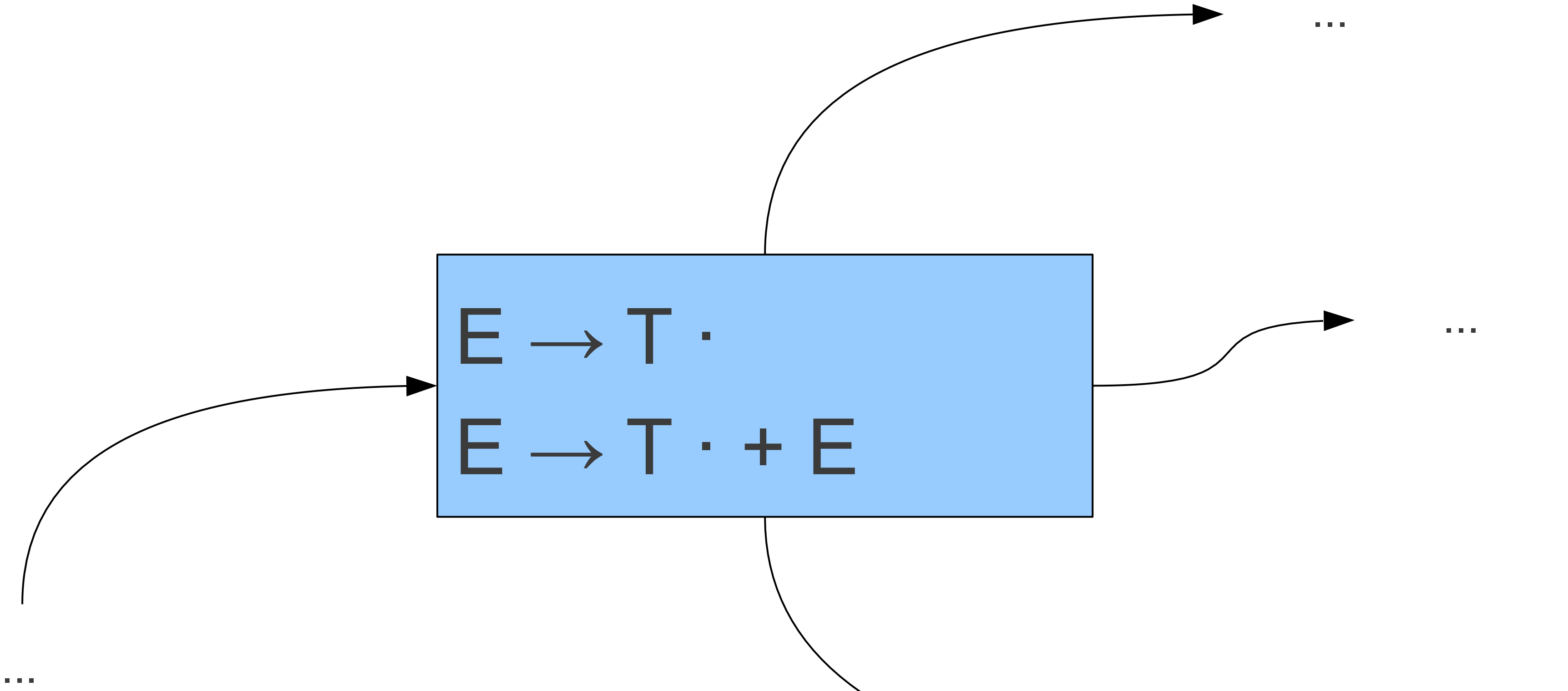
LR Conflicts

- 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.

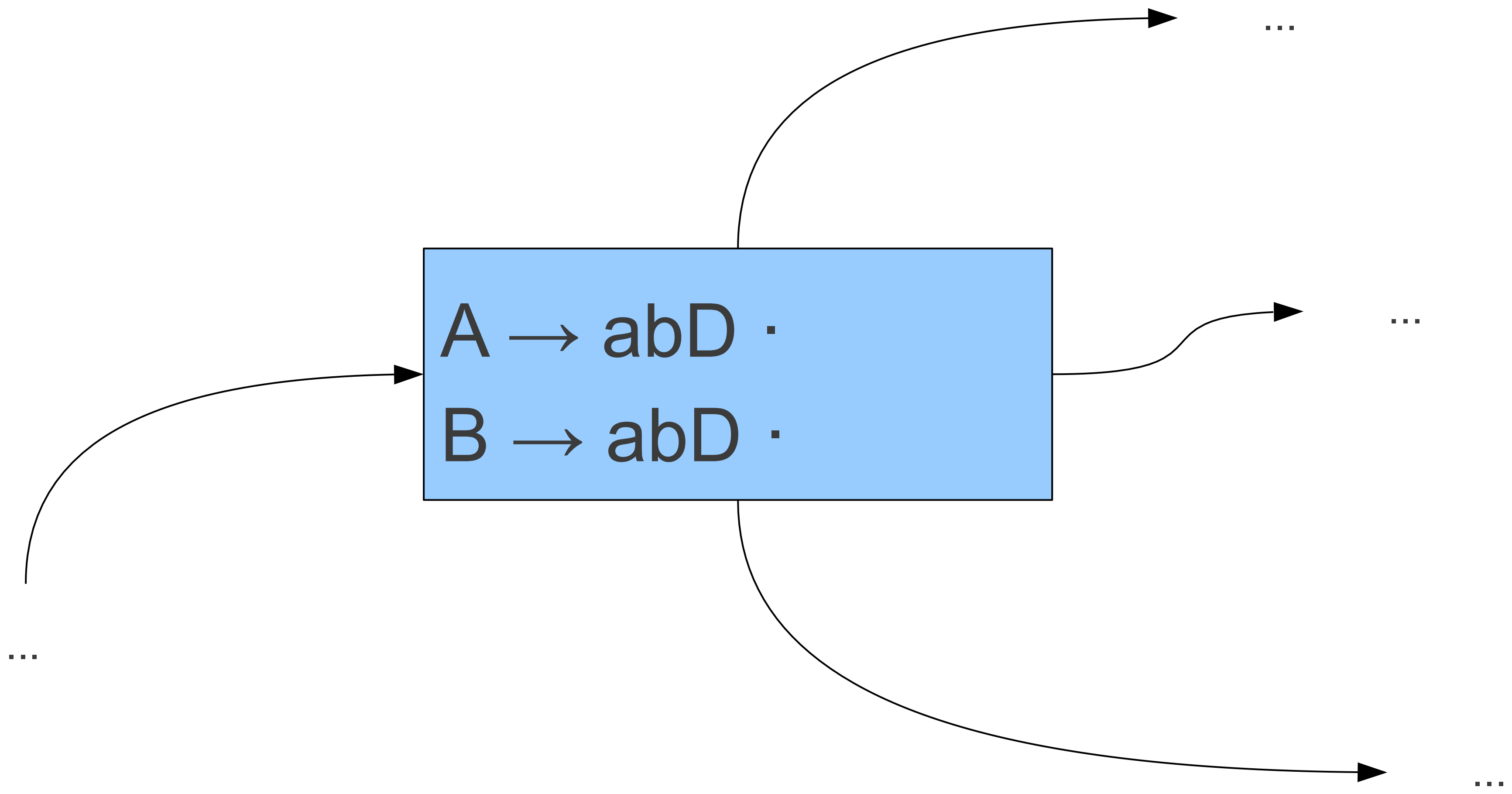
LR Conflicts

- 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).
- Can you have a shift/shift conflict?

What error is this?



What about this?



What do these conflicts mean?

- Recall: our automaton was constructed by looking for viable prefixes.

What do these conflicts mean?

- Recall: our automaton was constructed by looking for viable prefixes.
- Each accepting state represents a point where the handle might occur.

What do these 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.

What do these 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**.
- i.e. we don’t use the right side terminals.
- Our shift/reduce parser only looks to the left of the handle, not to the right.

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

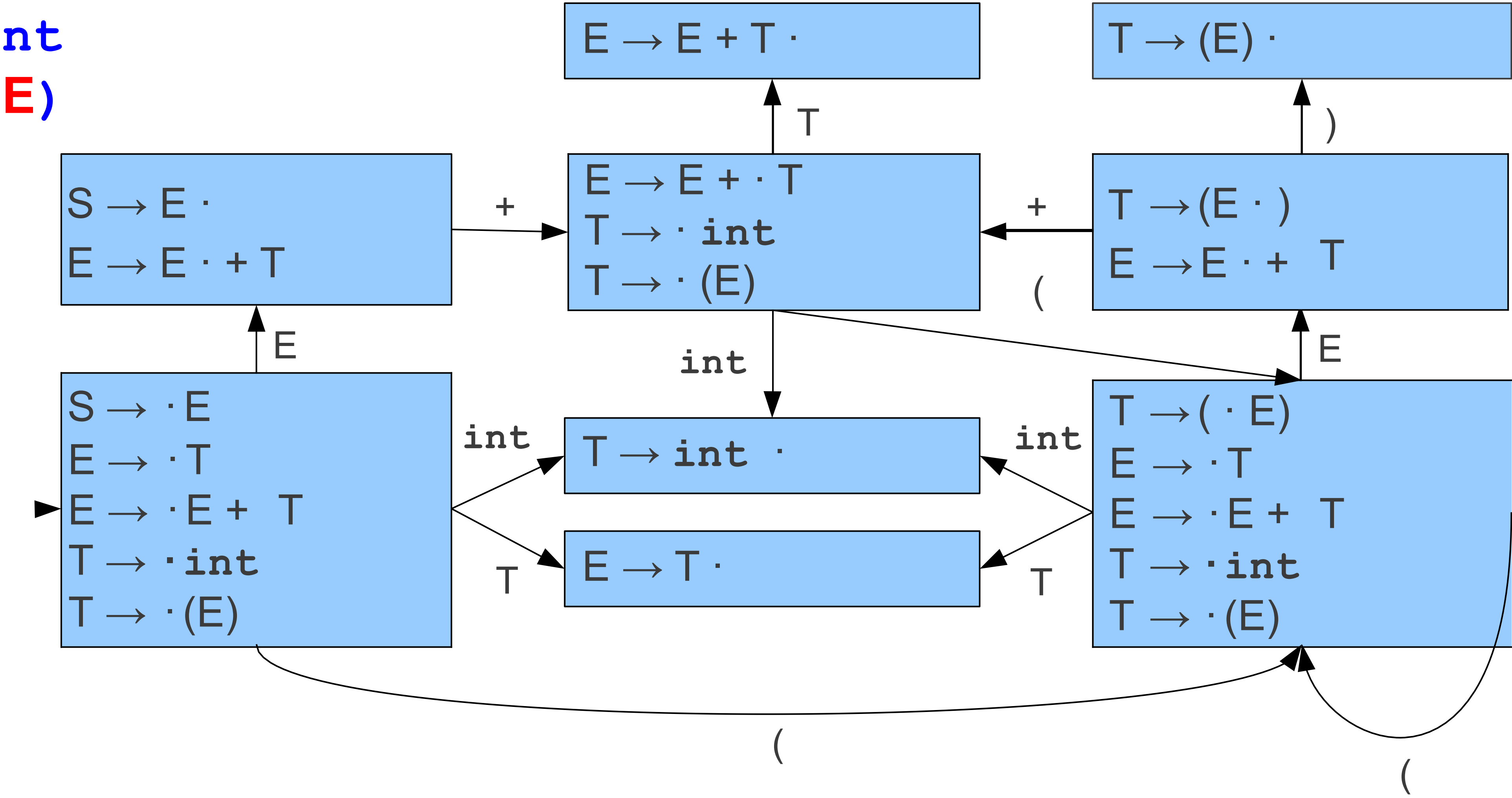
E → T

E → E + T

T → int

T → (E)

SLR(1)Parsing



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

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

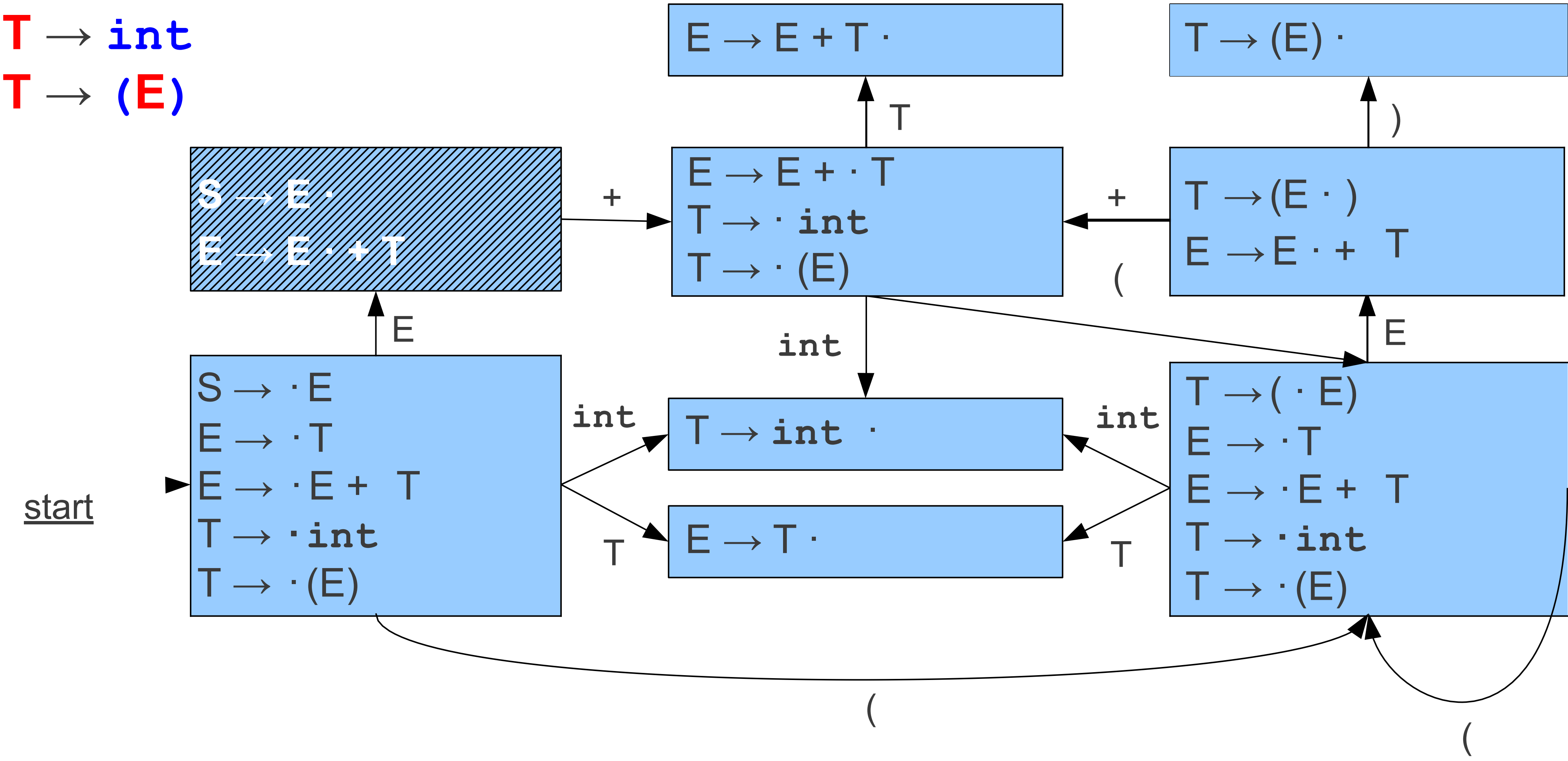
E → T

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 ·

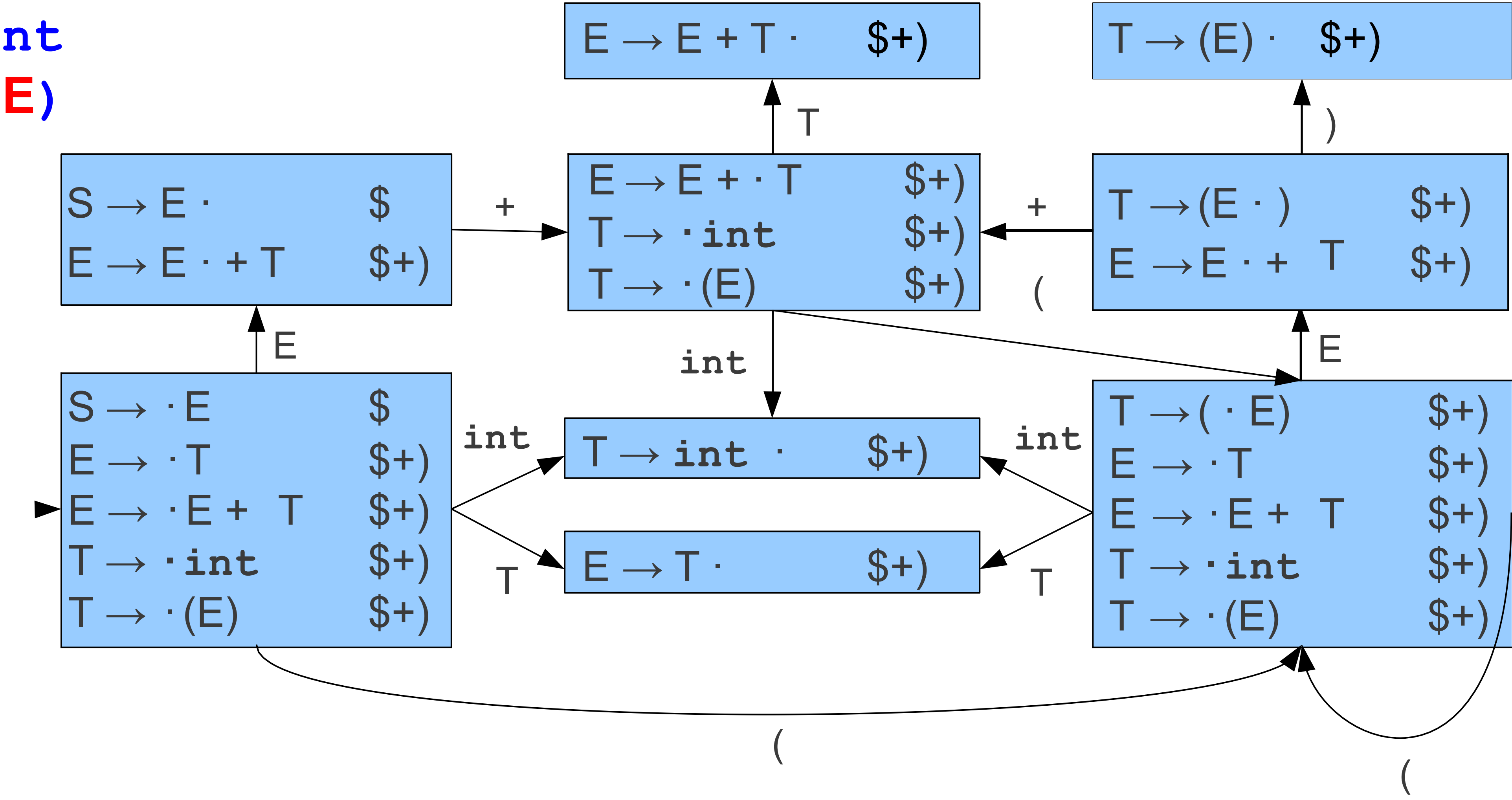
E → T ·

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



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 · $+)

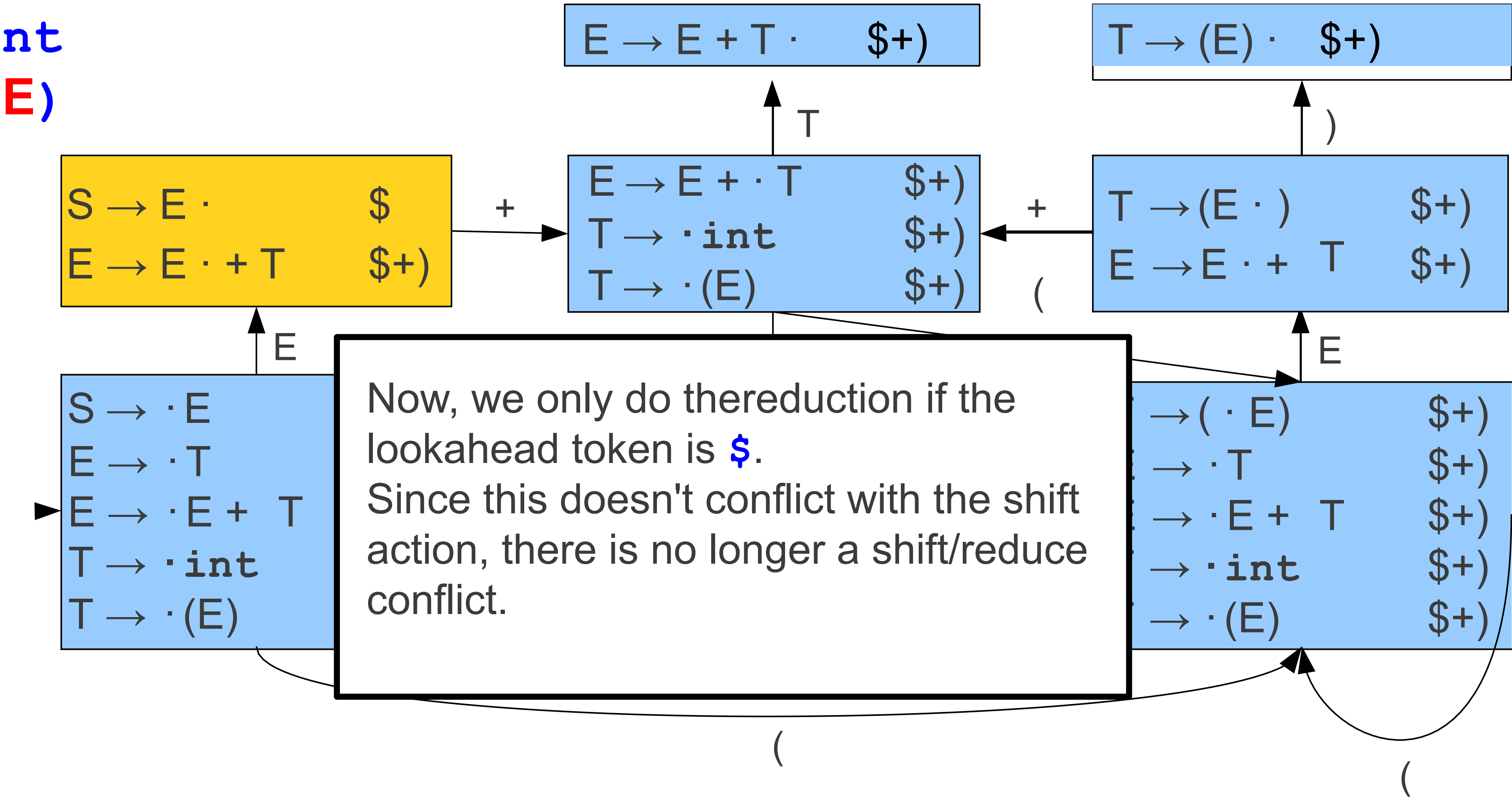
E → T · $+)

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 thereduction if the lookahead token is $.
Since this doesn't conflict with the shift action, there is no longer a shift/reduce conflict.

S → E

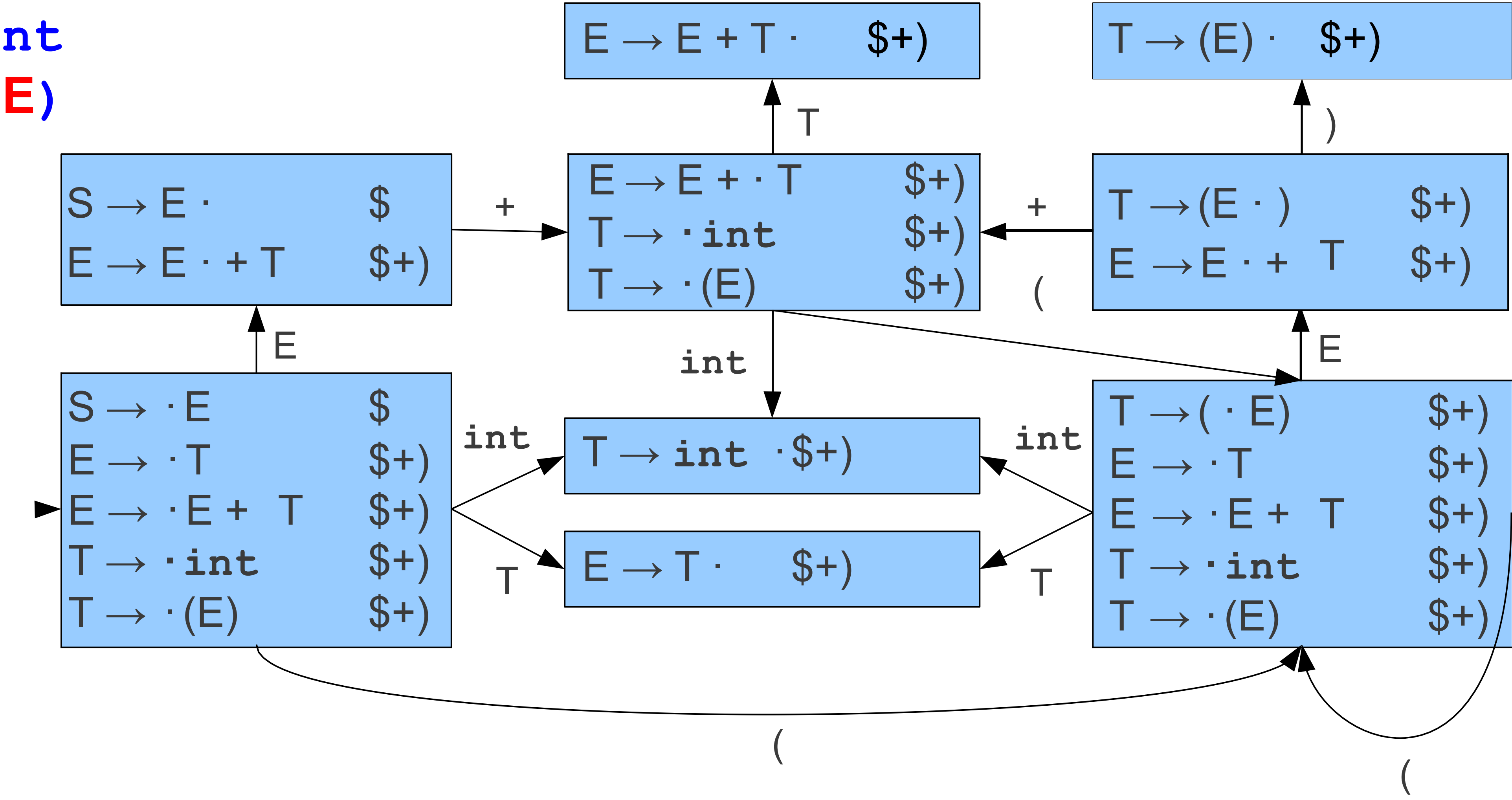
E → T

E → E + T

T → int

T → (E)

SLR(1)Parsing



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

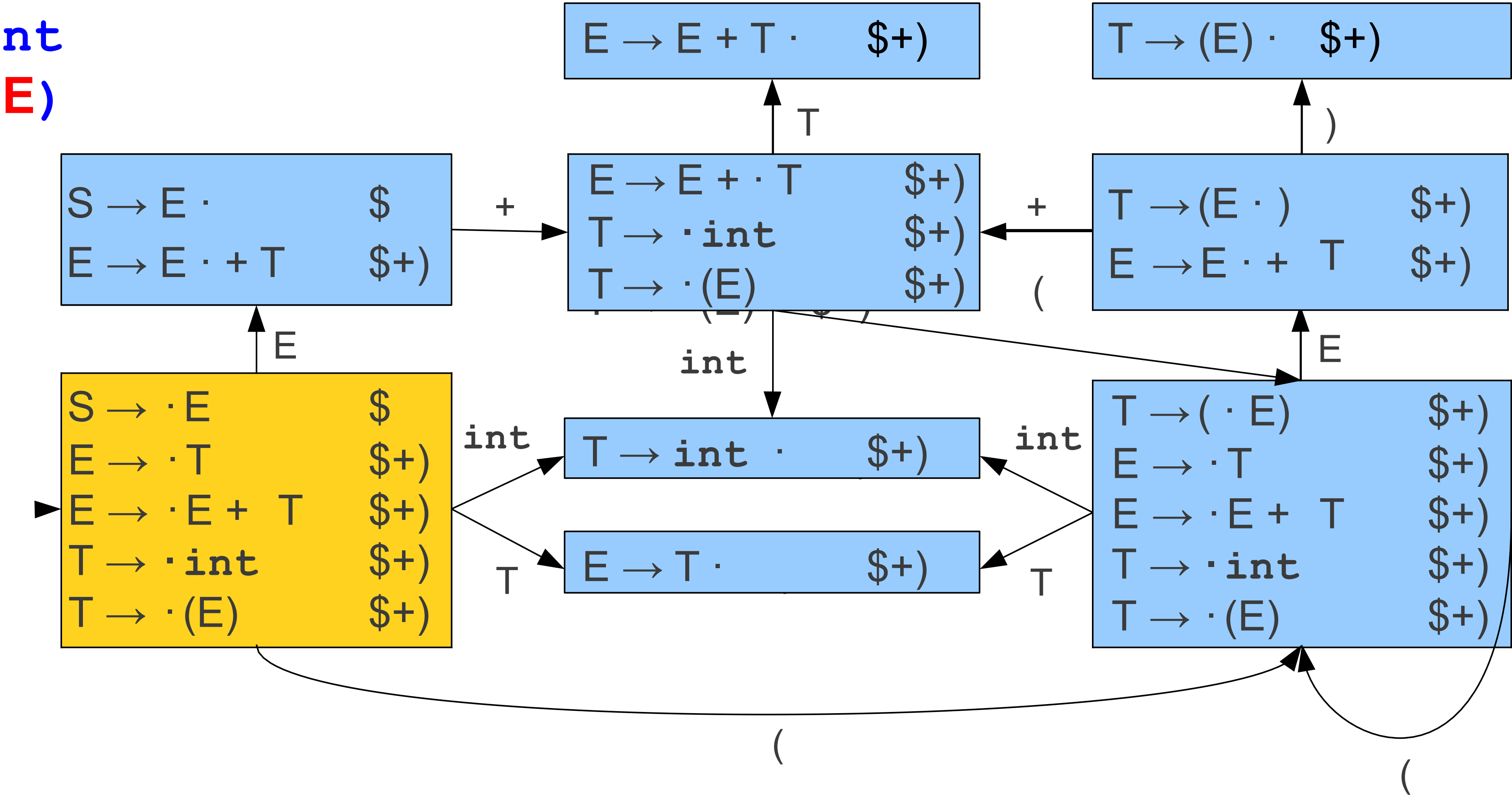
E → T

E → E + T

T → int

T → (E)

SLR(1)Parsing



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	)	$

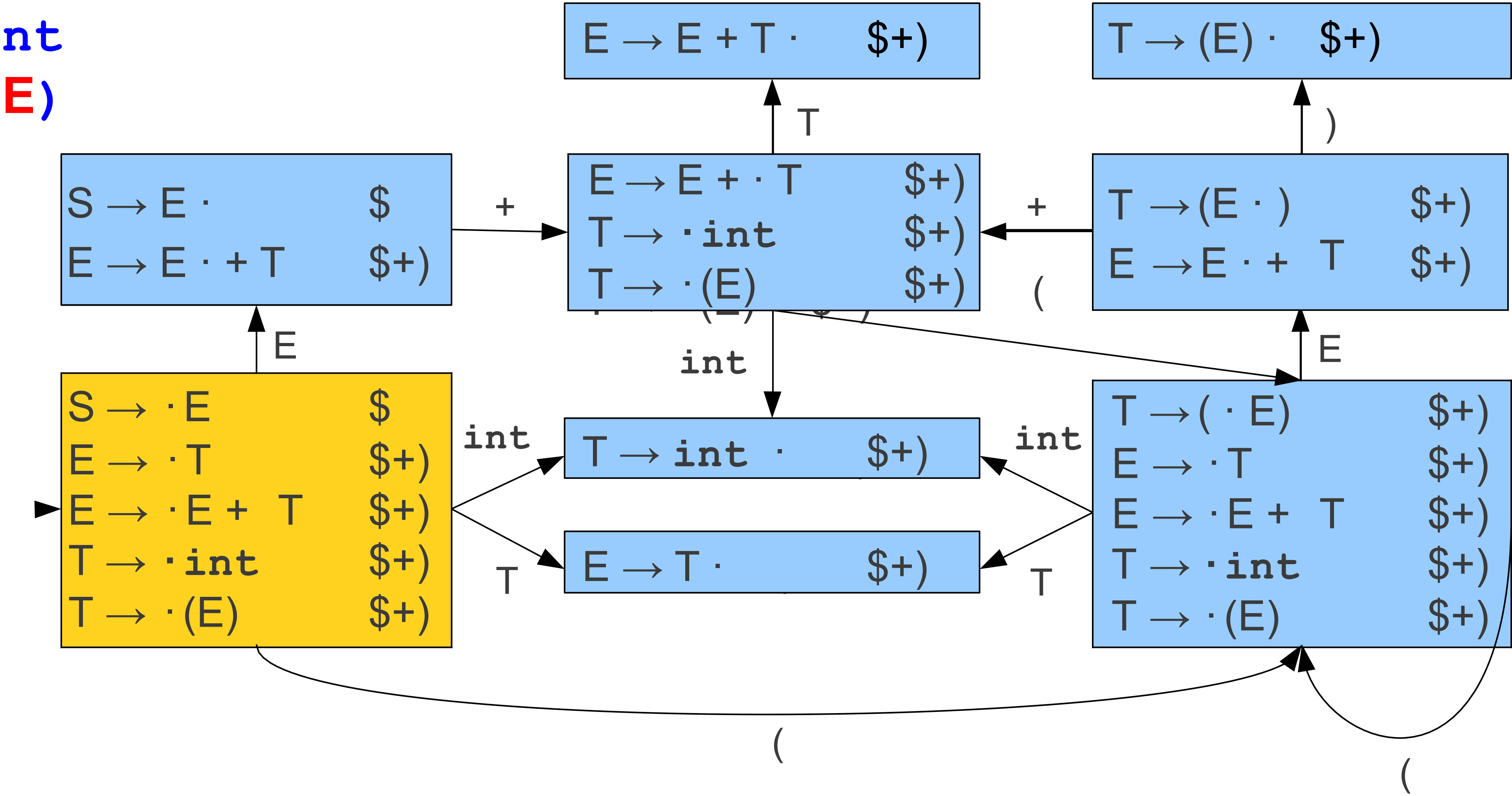
S → E

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

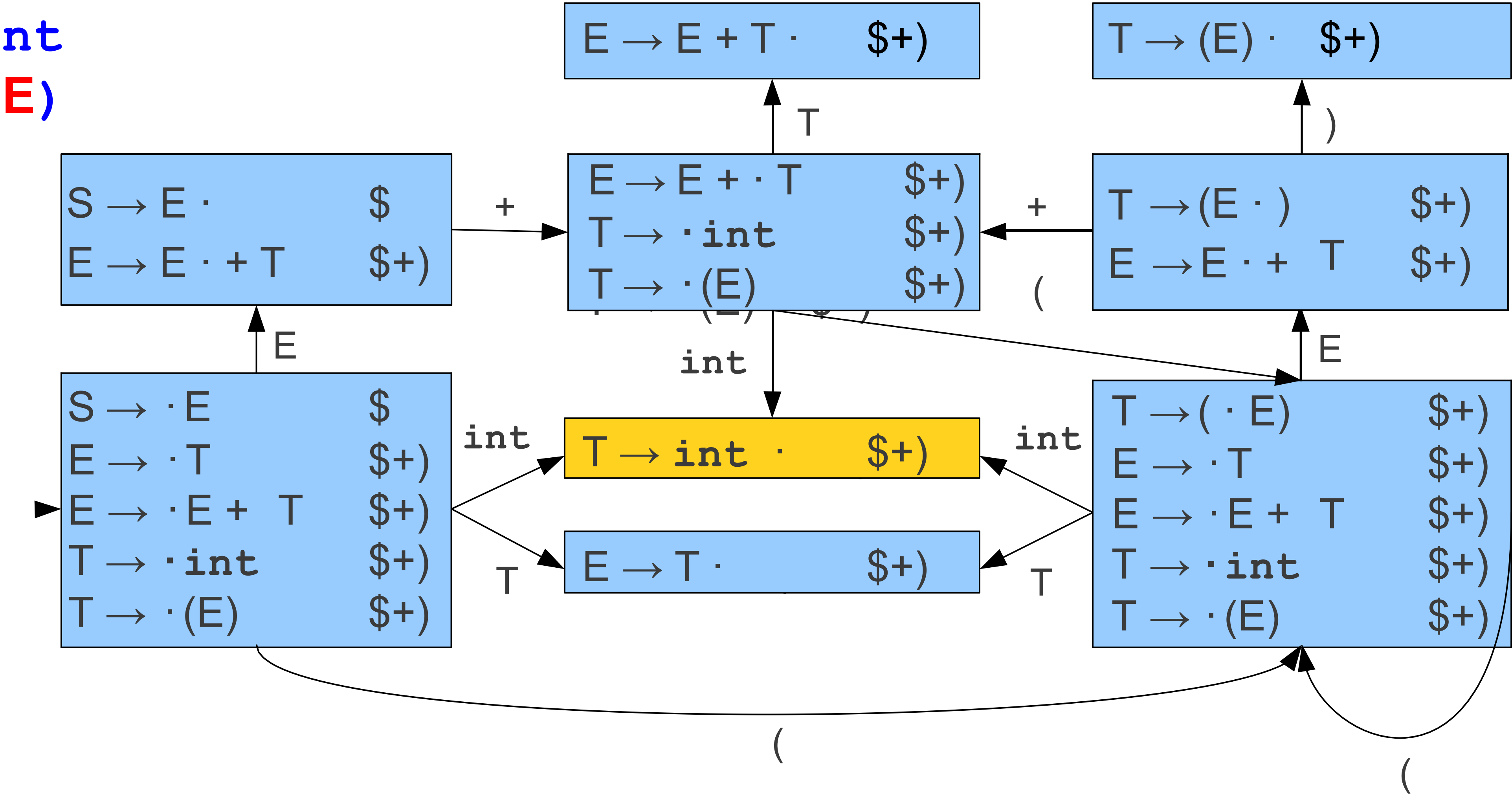
E → T

E → E + T

T → int

T → (E)

SLR(1)Parsing



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

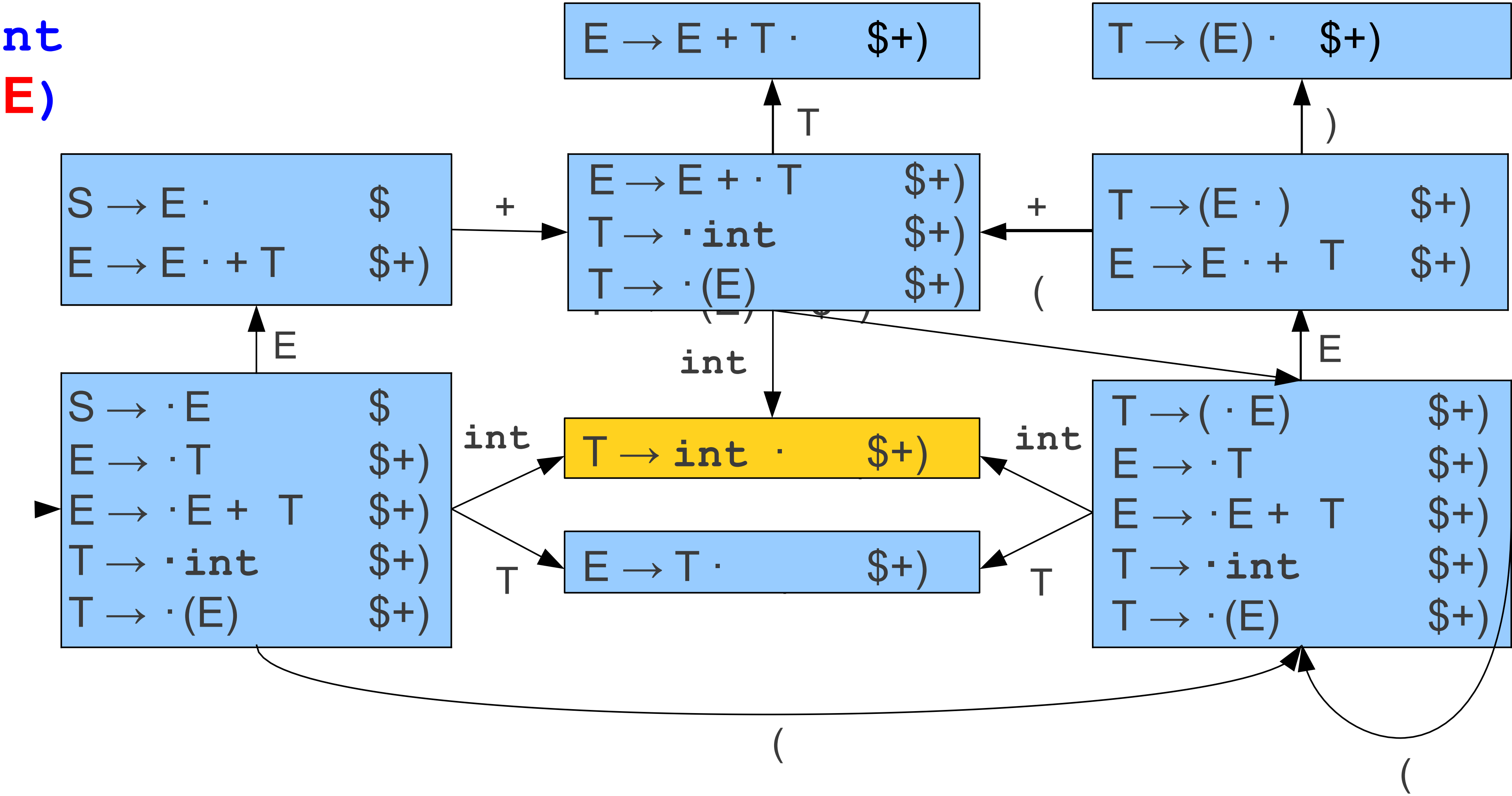
E → T

E → E + T

T → int

T → (E)

SLR(1)Parsing



int | + (int + int + int) $

S → E

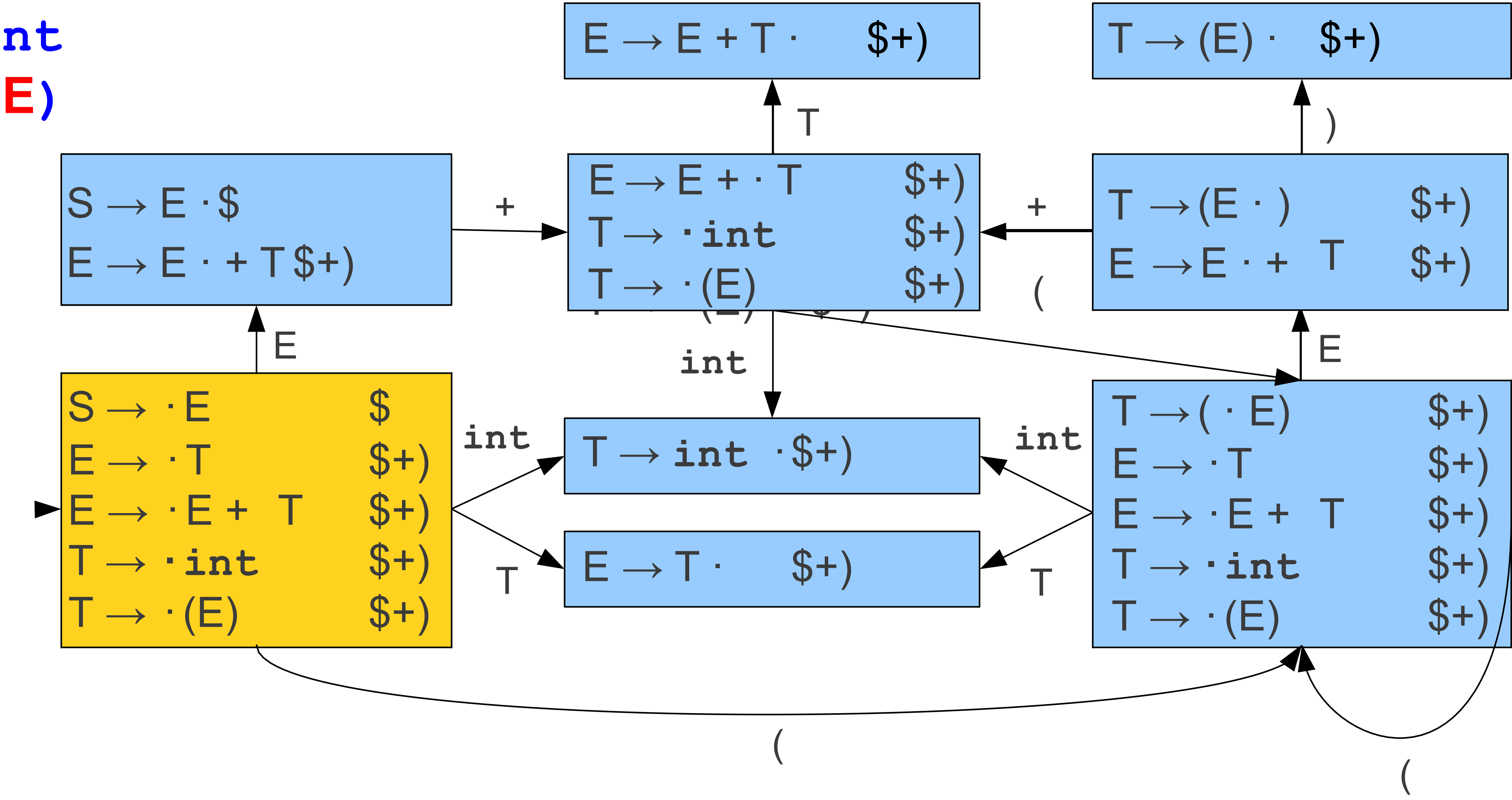
E → T

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) $+)

T		+	(	int	+	int	+	int	)	$

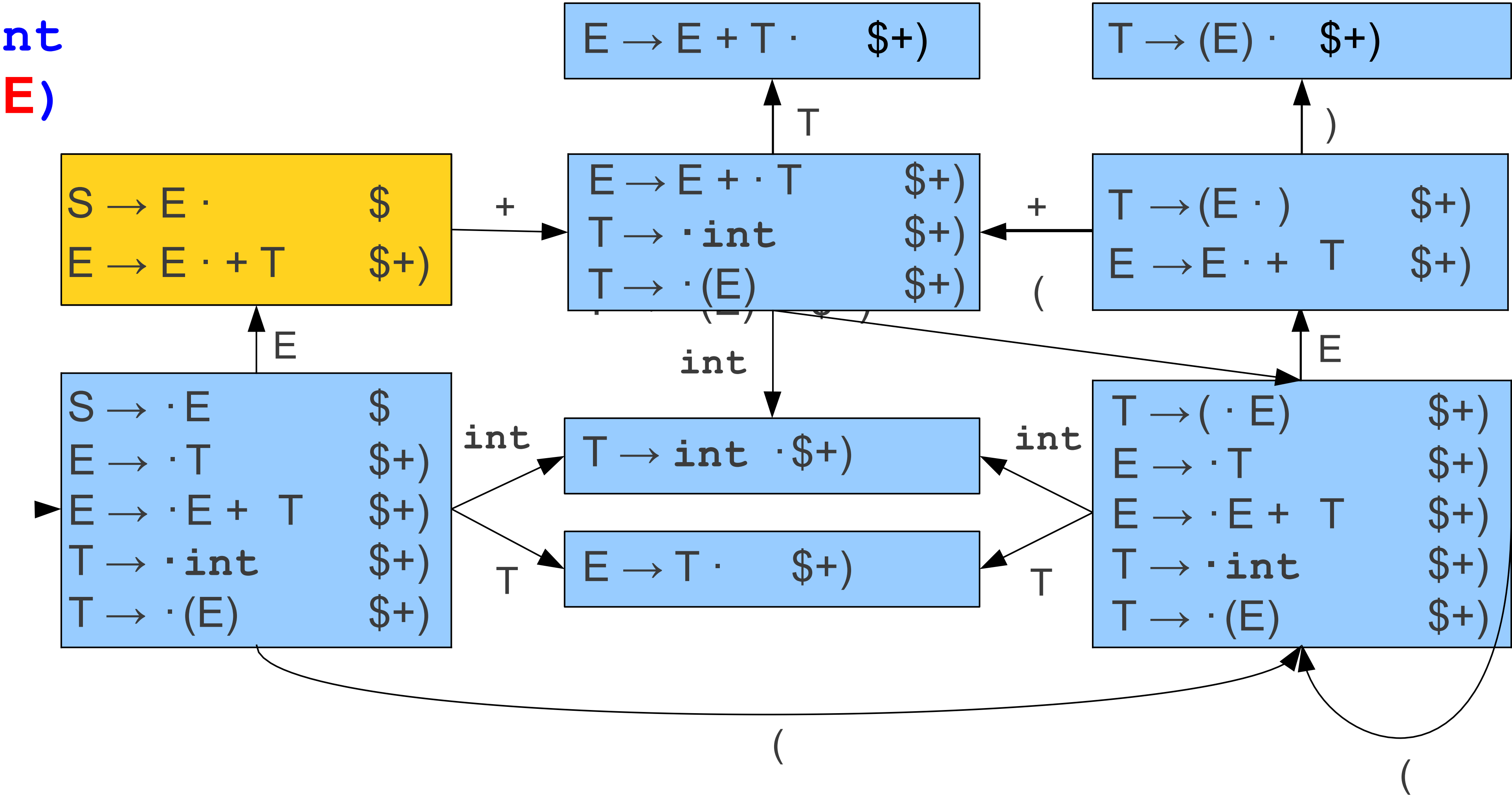
S → E

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 ·) $+)
E → E · + T $+)

T → (E) · $+)

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

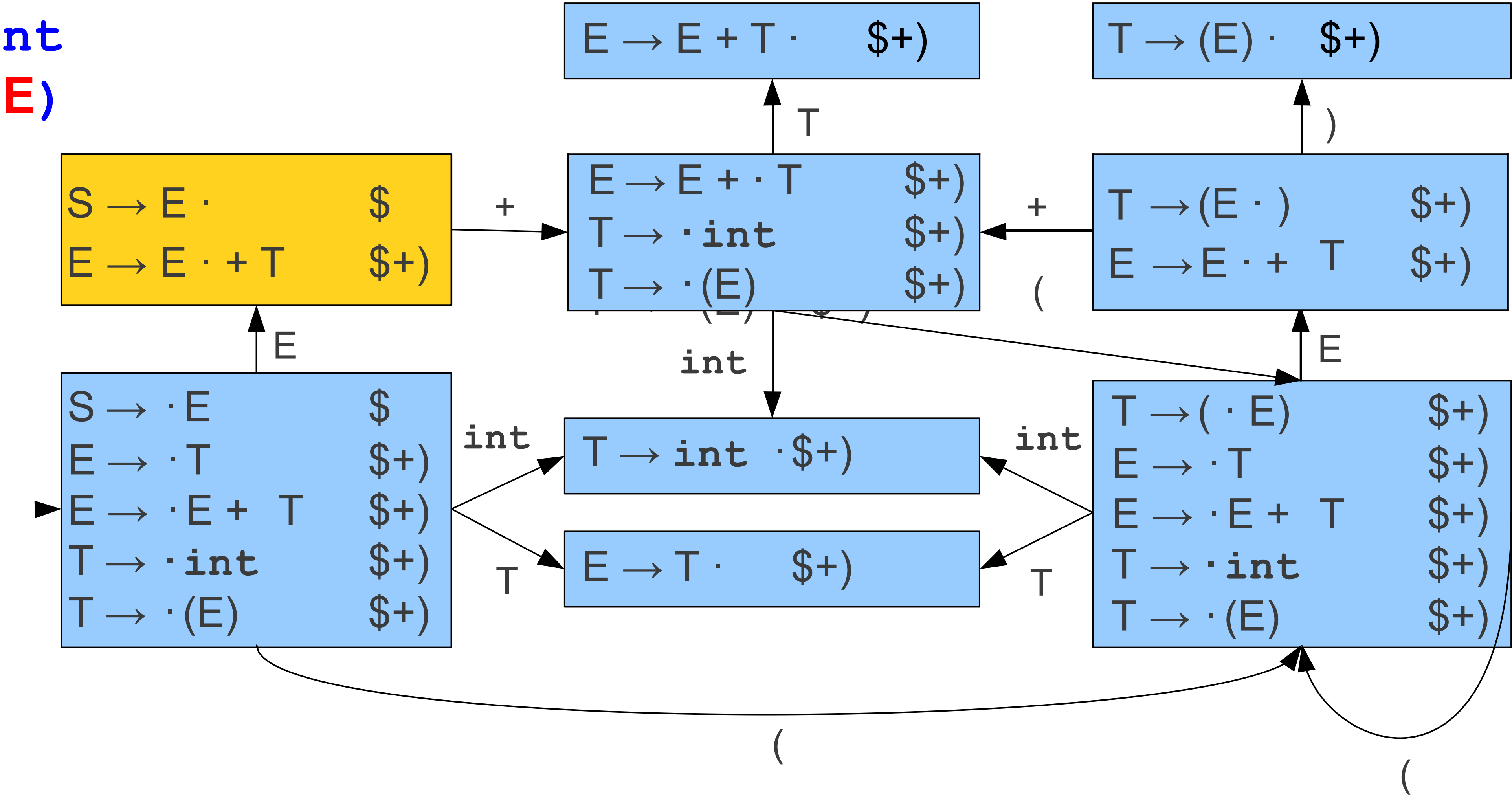
E → T

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

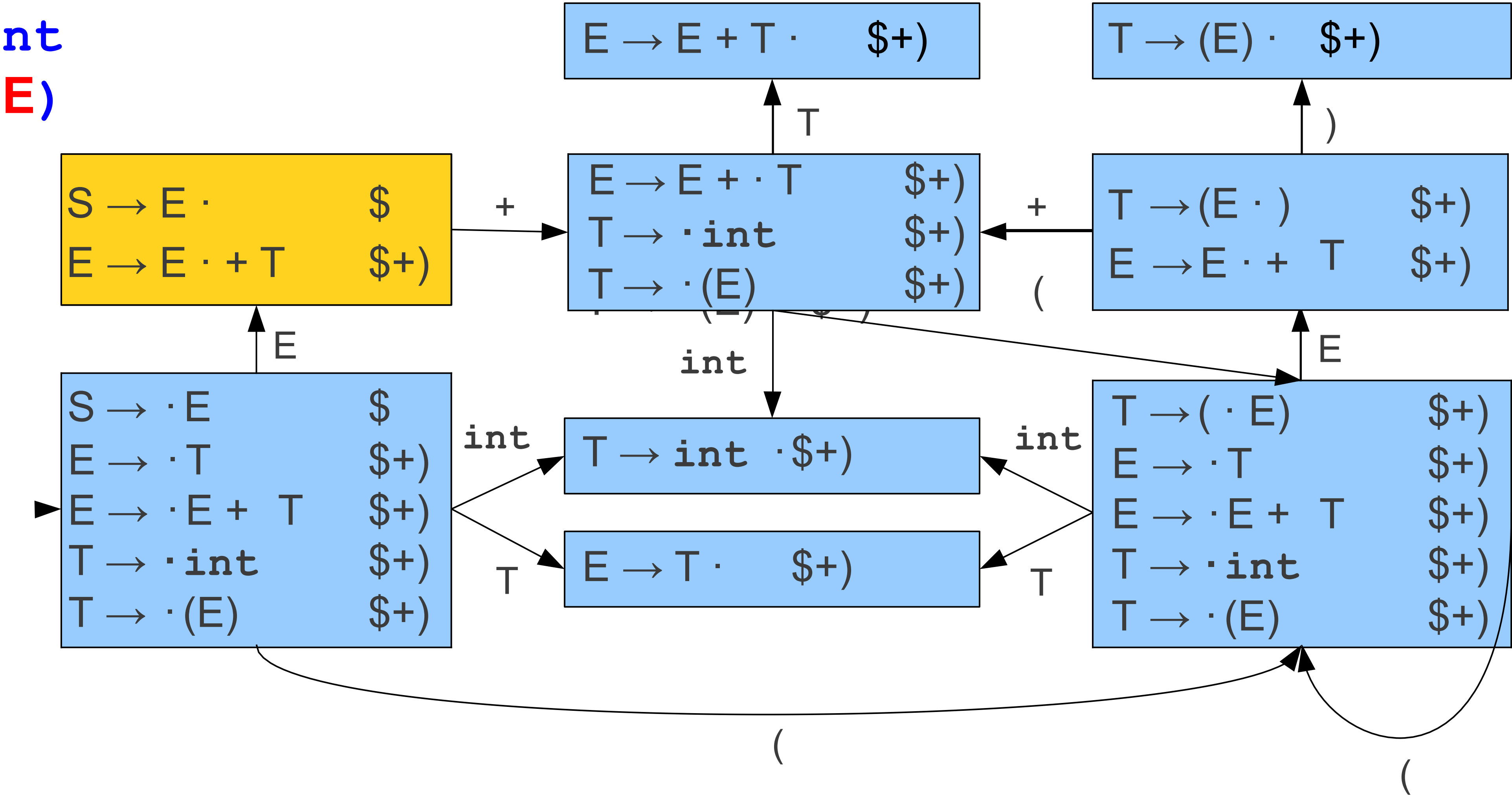
E → T

E → E + T

T → int

T → (E)

SLR(1)Parsing



E → 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

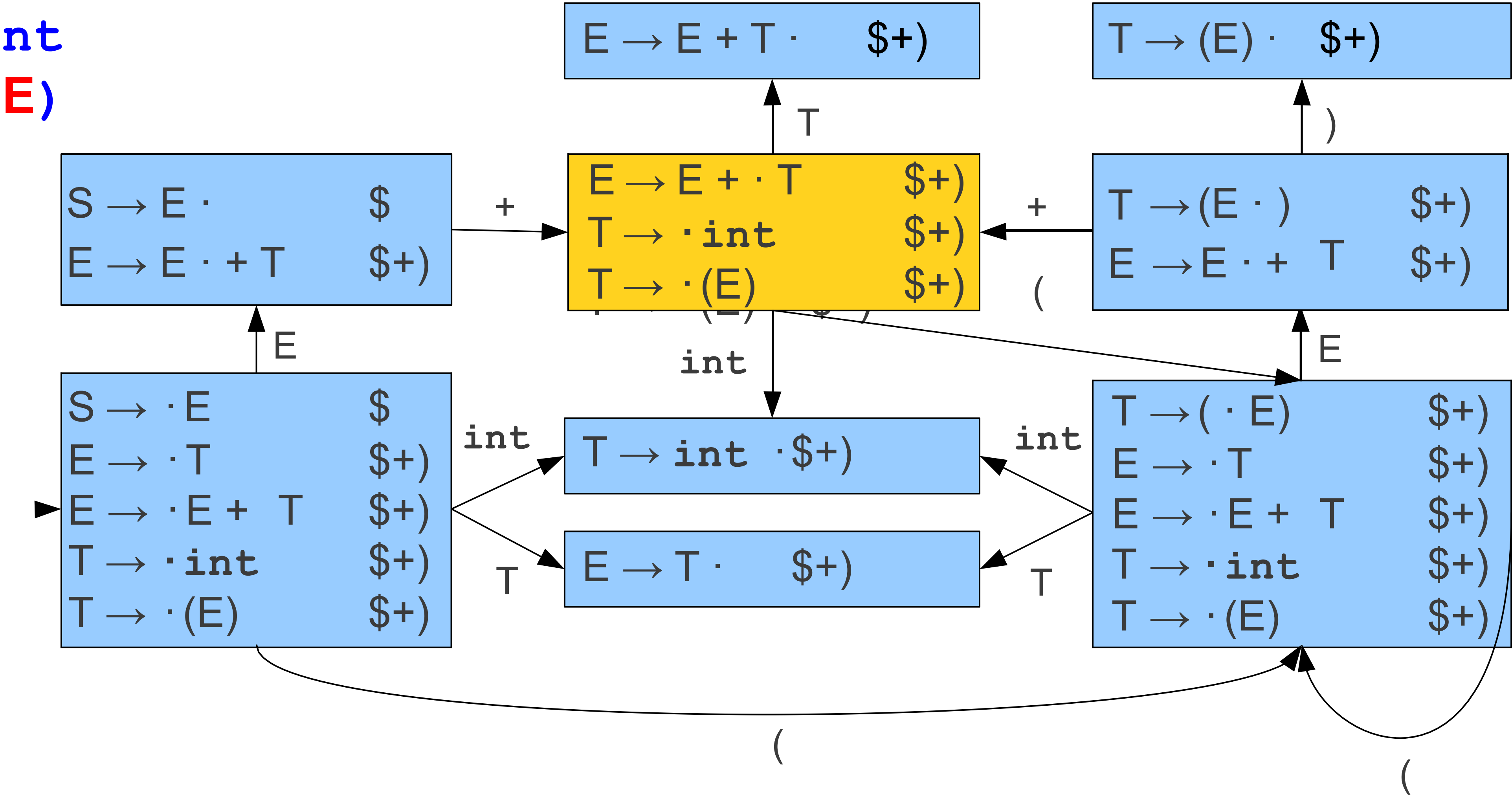
E → T

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) $+)

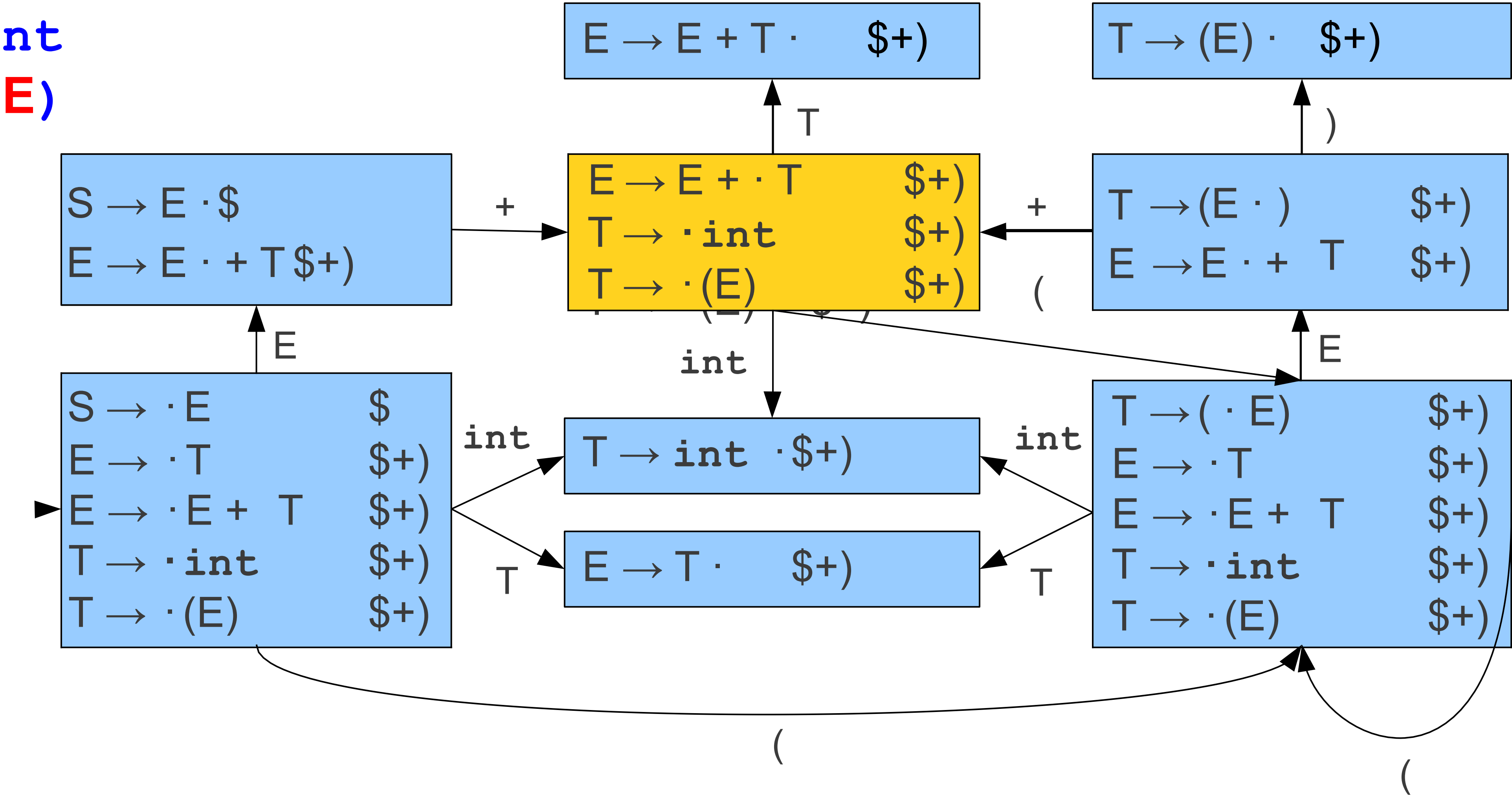
E	+

(	int	+	int	+	int	)	$

S → E

E → T
E → E + T
T → int
T → (E)

SLR(1)Parsing



E → 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

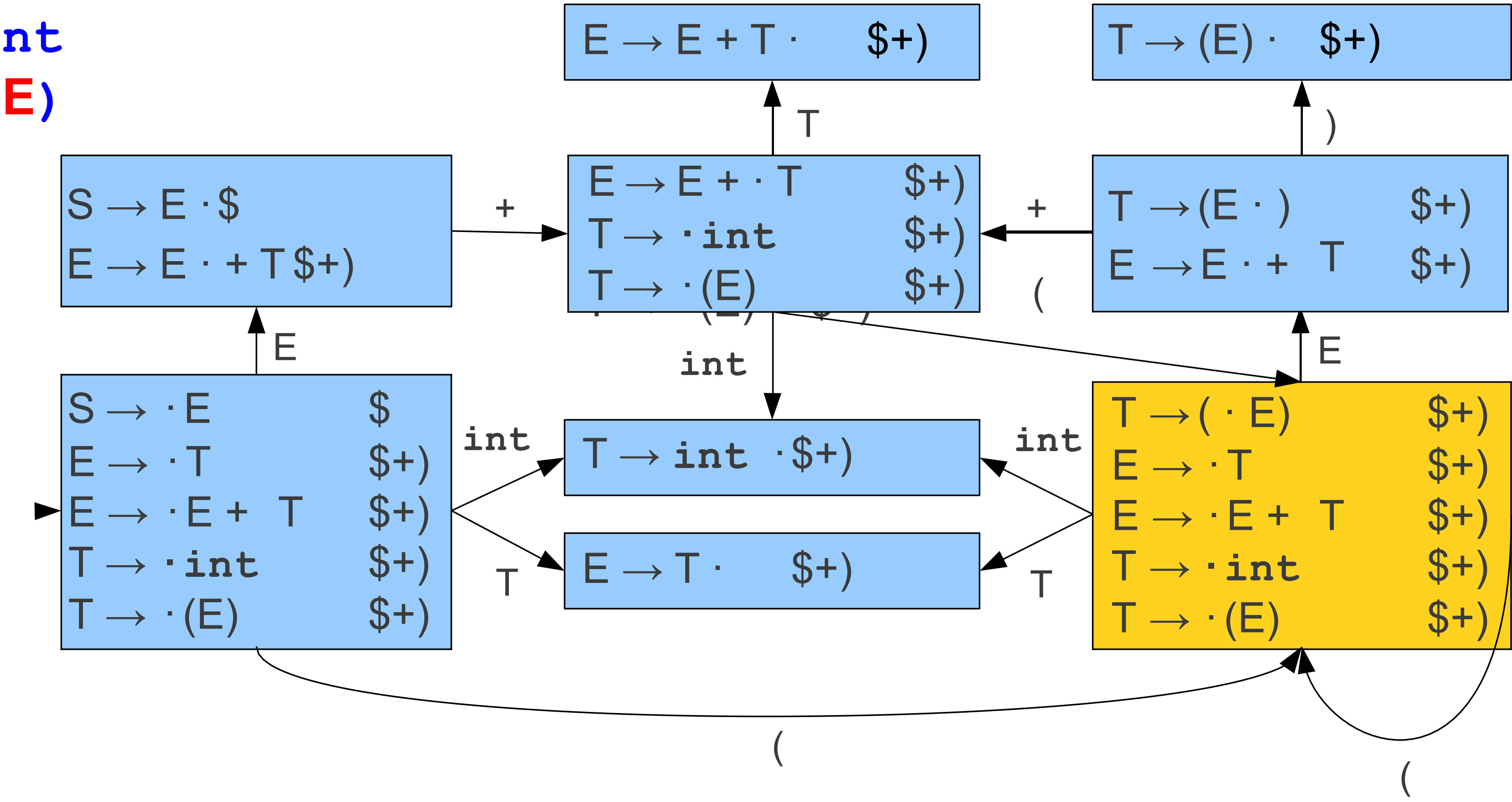
E → T

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 ·) $+)
E → E · + T $+)

T → (E) · $+)

start

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	)	$

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)

$S \rightarrow E$

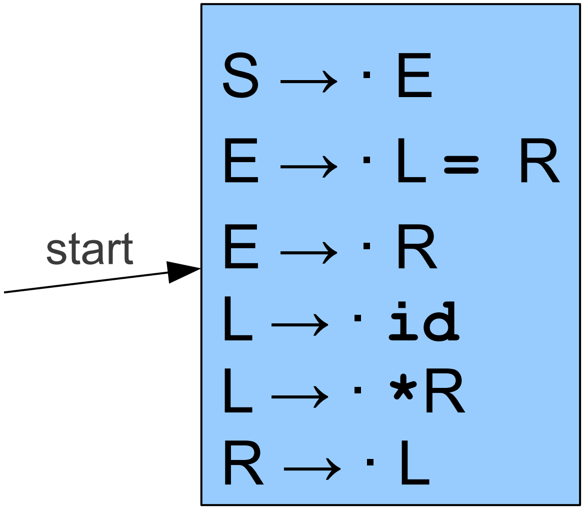
$E \rightarrow L \,\mathtt{=}\, R$

$E \rightarrow R$

$L \rightarrow \mathtt{id}$

$L \rightarrow \mathtt{*}R$

$R \rightarrow L$



$S \rightarrow \cdot E$

$E \rightarrow \cdot L \,\mathtt{=}\, R$

$E \rightarrow \cdot R$

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

$L \rightarrow \cdot \mathtt{*}R$

$R \rightarrow \cdot L$

The Limits of SLR(1)

$S \to E$

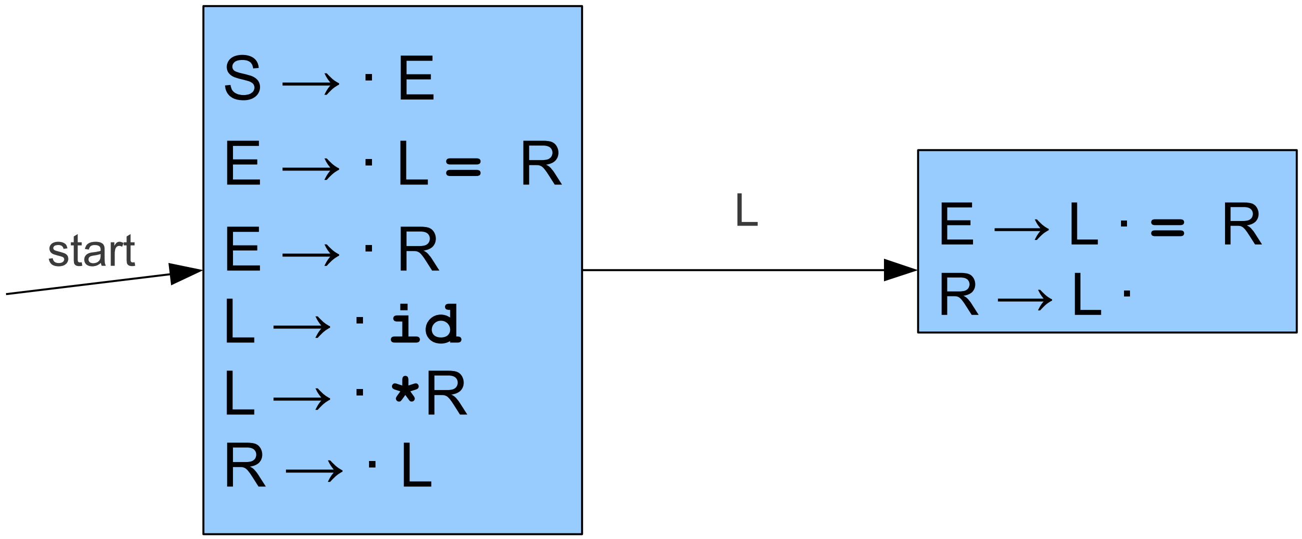
$E \to L\ \texttt{=}\ R$

$E \to R$

$L \to \texttt{id}$

$L \to \texttt{*}R$

$R \to L$



The Limits of SLR(1)

S → E

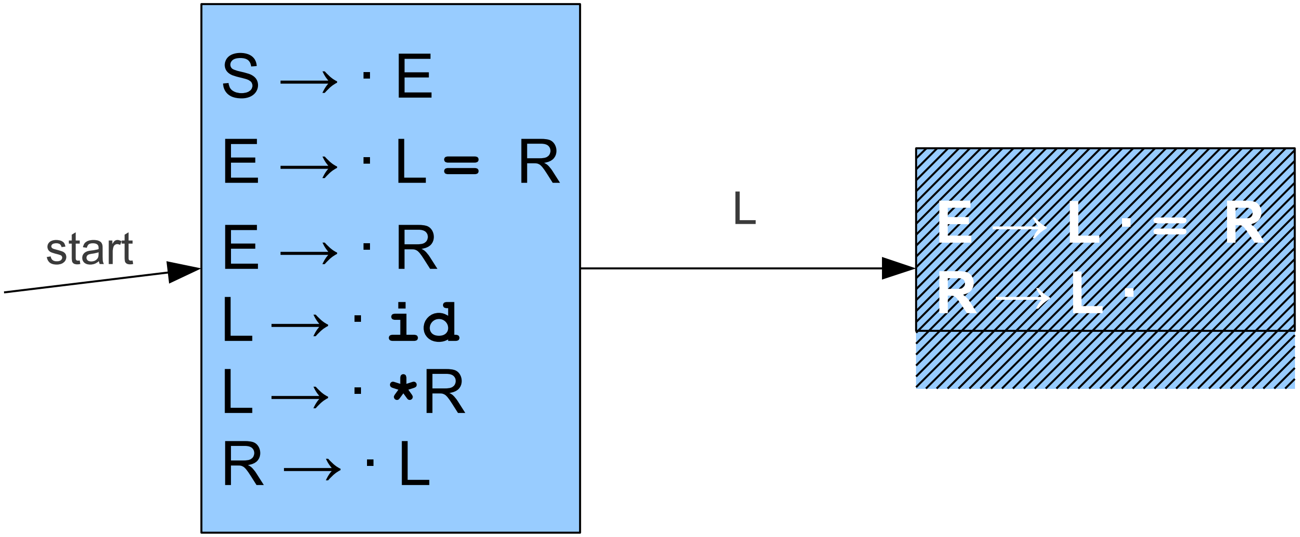
E → L = R

E → R

L → id

L → *R

R → L



The Limits of SLR(1)

$S \rightarrow E$

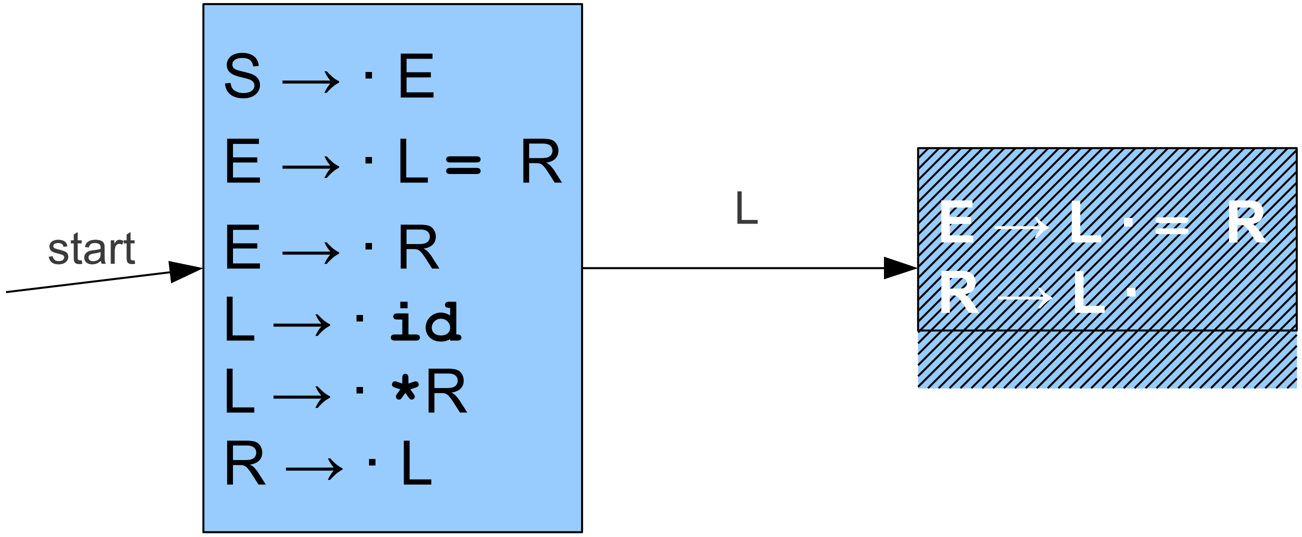
$E \rightarrow L \texttt{=} R$

$E \rightarrow R$

$L \rightarrow \texttt{id}$

$L \rightarrow \texttt{*}R$

$R \rightarrow L$



$S \rightarrow \cdot E$

$E \rightarrow \cdot L \texttt{=} R$

$E \rightarrow \cdot R$

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

$L \rightarrow \cdot \texttt{*}R$

$R \rightarrow \cdot L$

$E \rightarrow L \cdot \texttt{=} R$

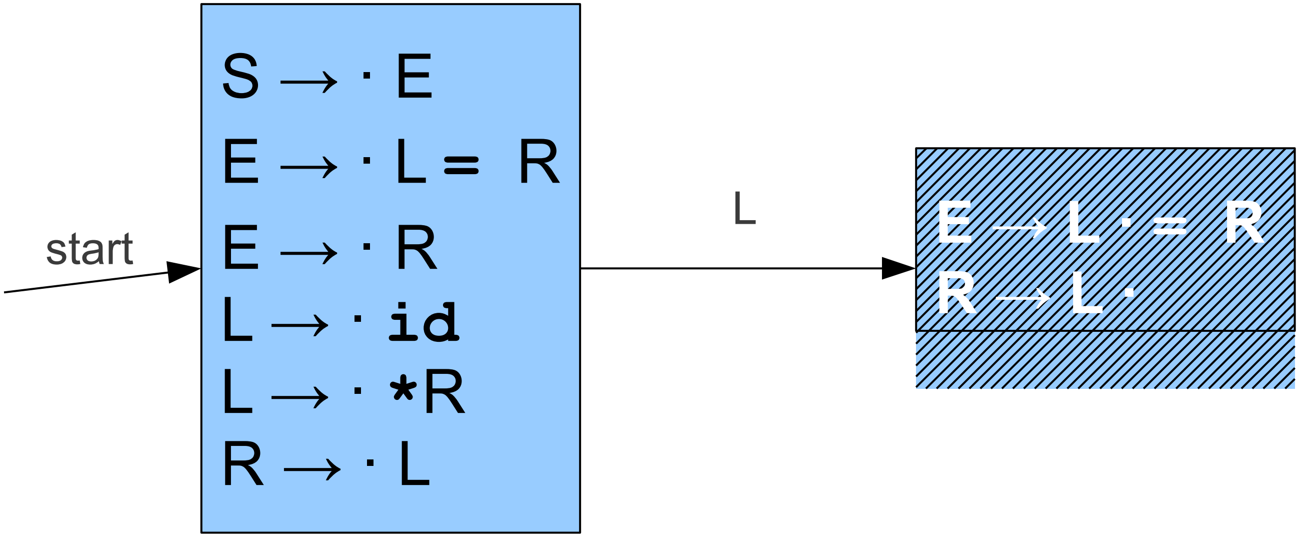
$R \rightarrow L \cdot$

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

$R \rightarrow L \cdot$ 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)

$S \rightarrow E$

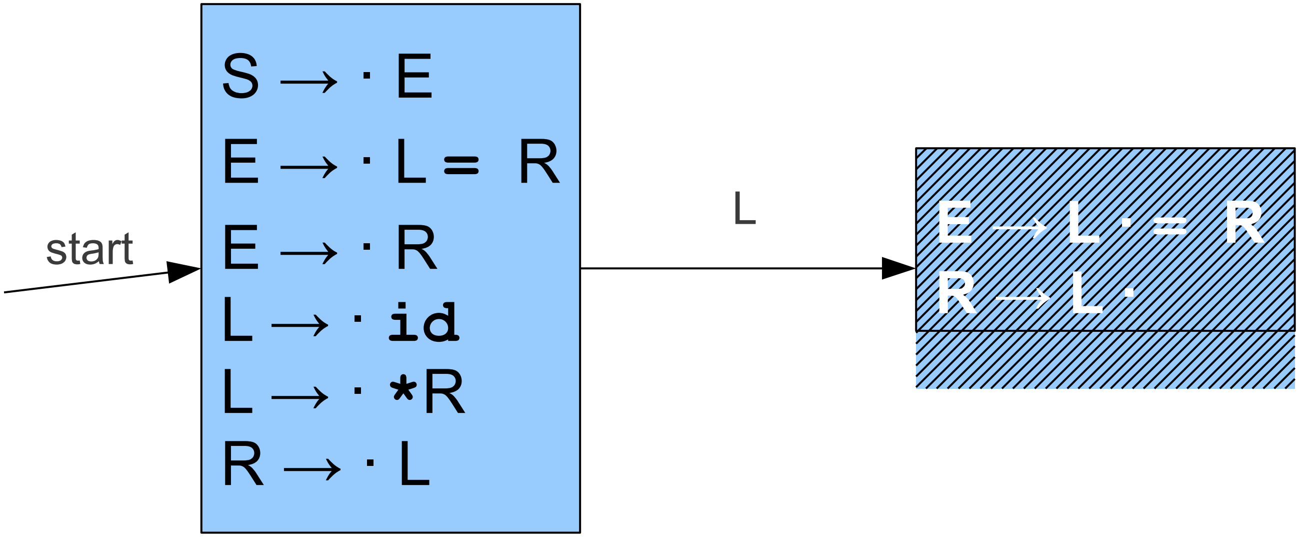
$E \rightarrow L = R$

$E \rightarrow R$

$L \rightarrow \mathtt{id}$

$L \rightarrow *R$

$R \rightarrow L$



$E \rightarrow L \cdot = R$ 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

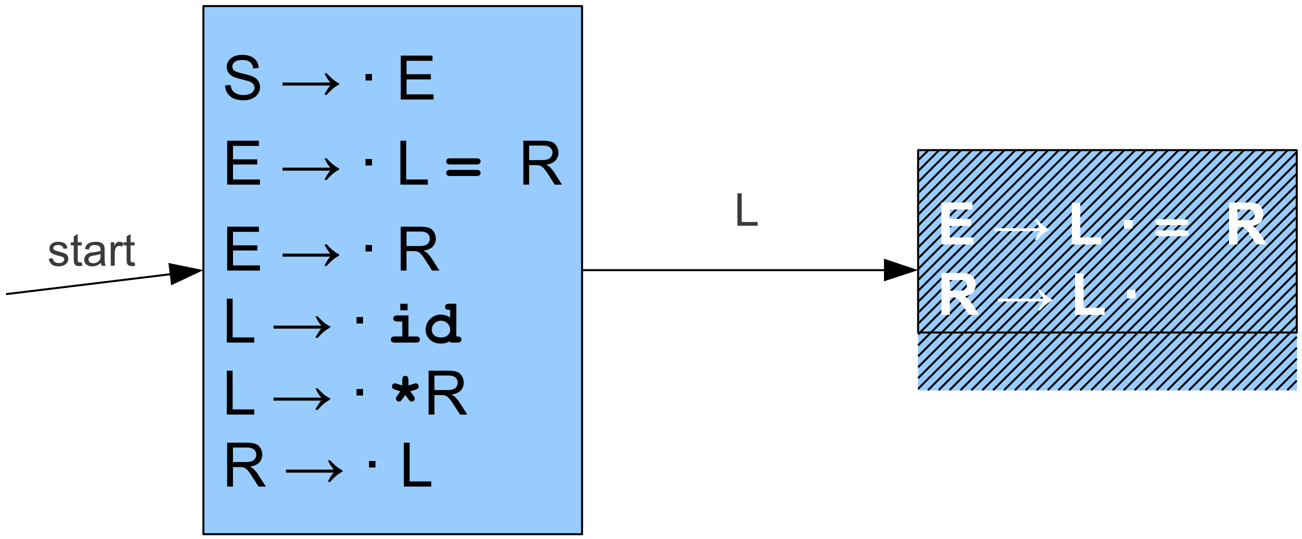
E → L = R

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.
  - 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

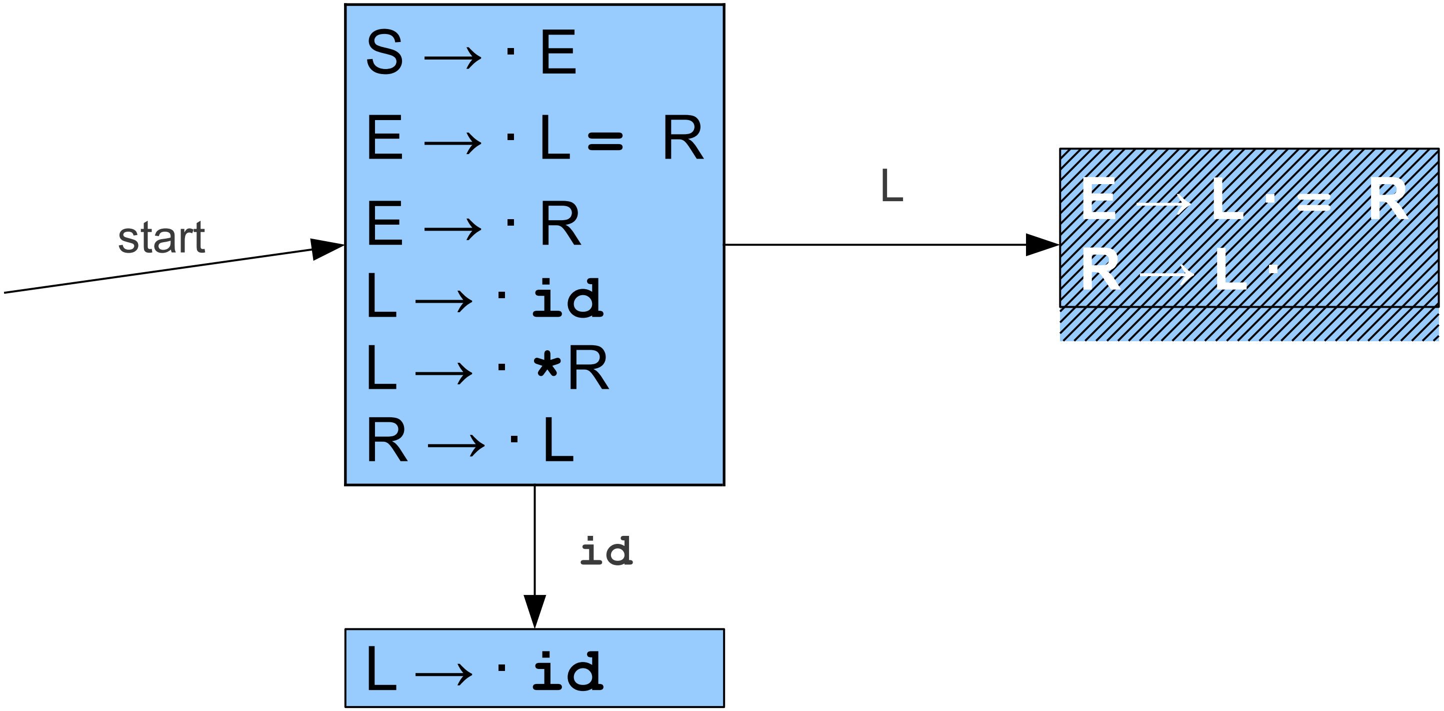
E → L = R

E → R

L → id

L → *R

R → L



start

S → · E

E → · L = R

E → · R

L → · id

L → · *R

R → · L

L

E → L · = R

R → L ·

id

L → · id

A Lack of Context

S → E

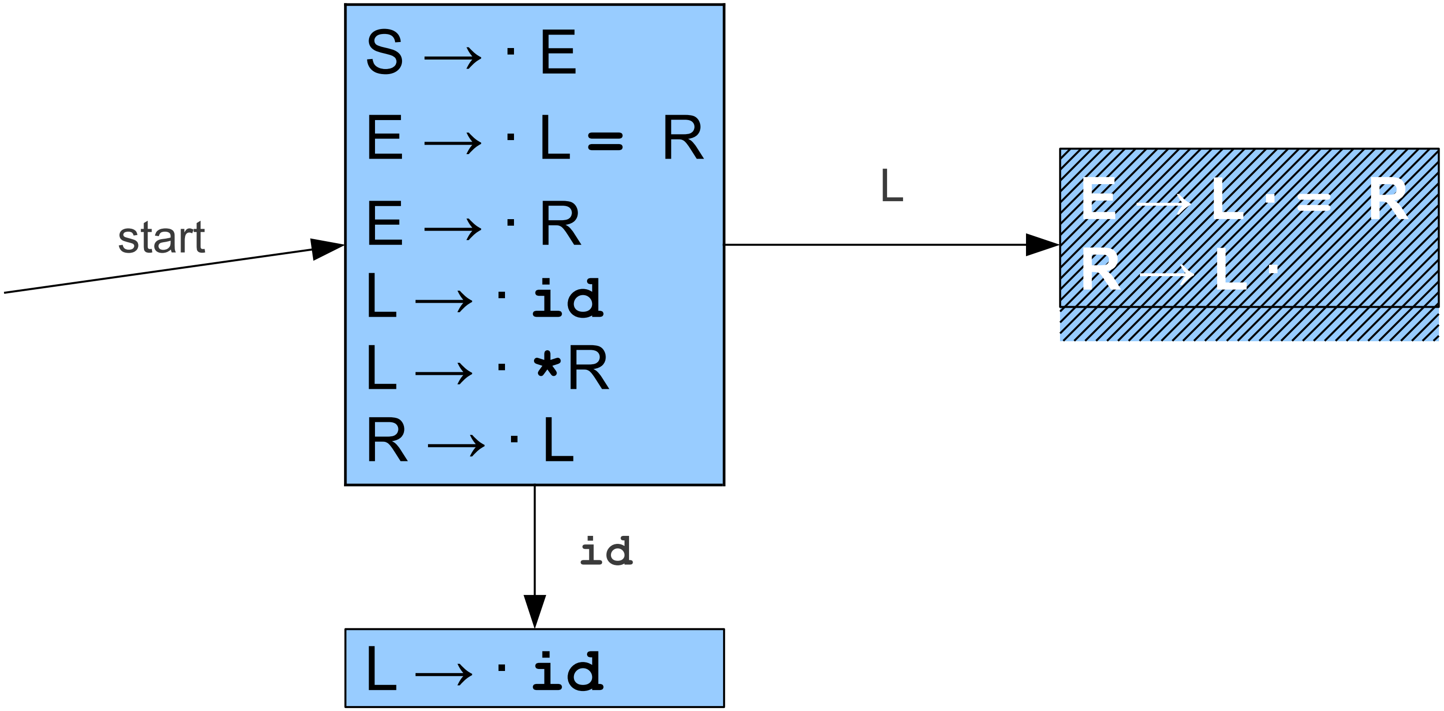
E → L = R

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

id	=	*	id

A Lack of Context

S → E

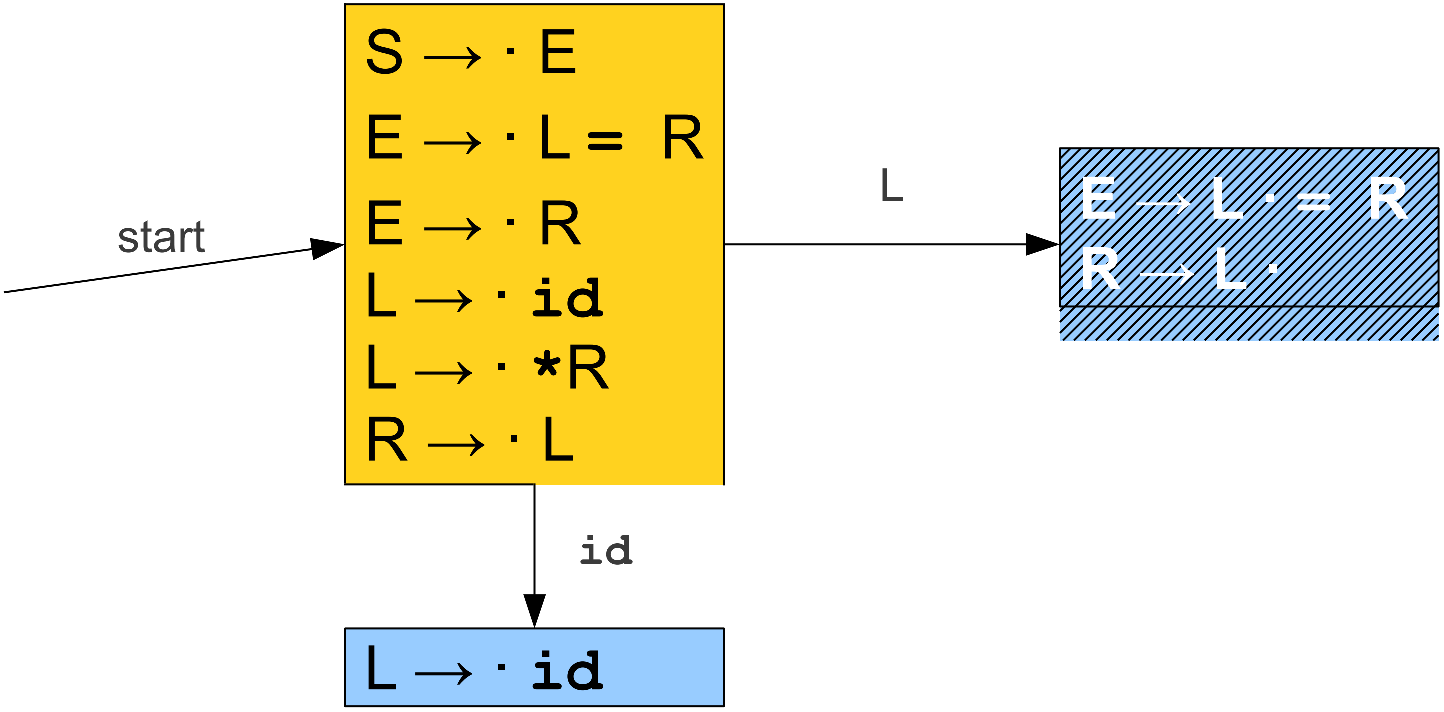
E → L = R

E → R

L → id

L → *R

R → L



start

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

E → L · = R
R → L ·

L → · id

id	=	*	id

A Lack of Context

S → E

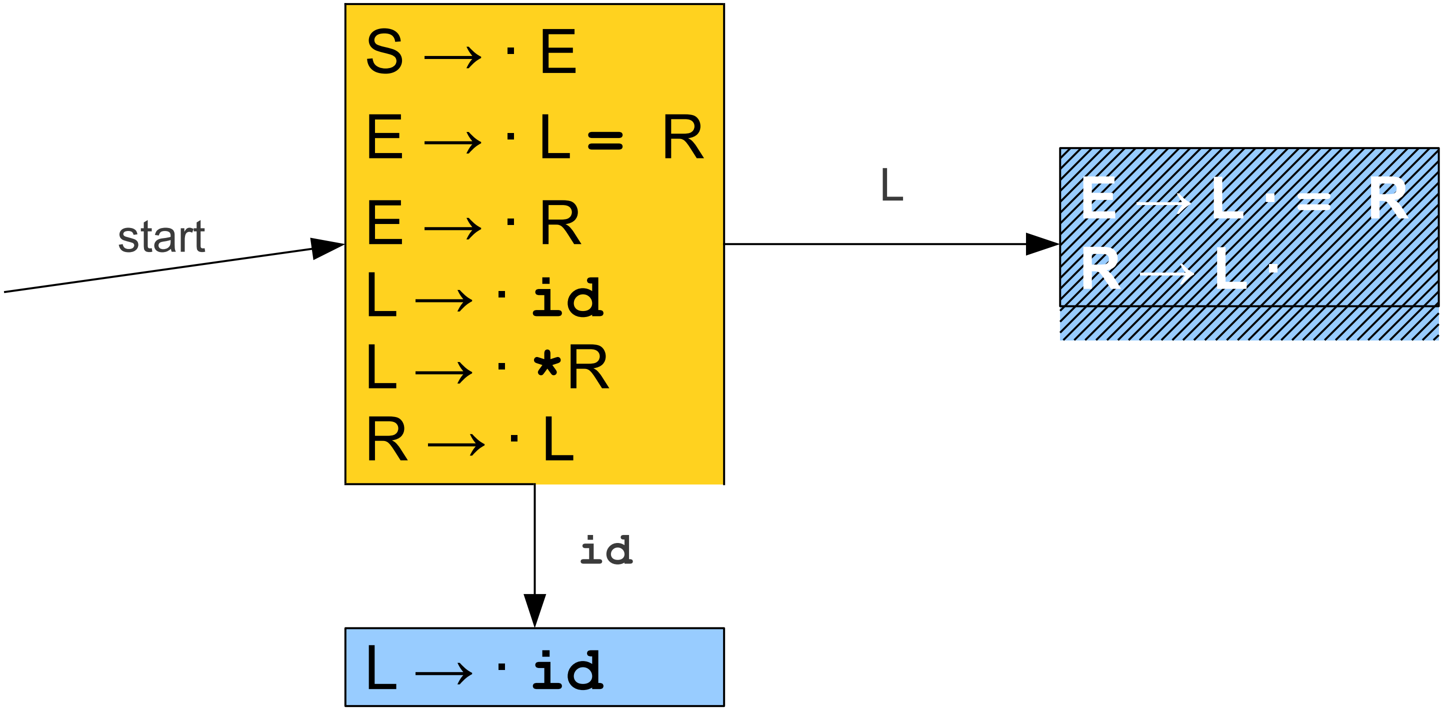
E → L = R

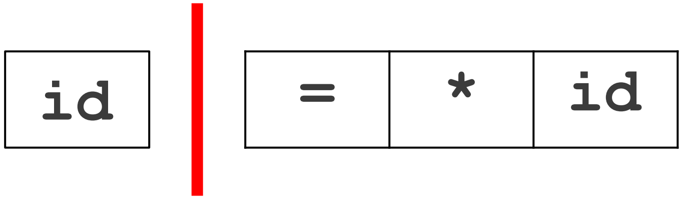
E → R

L → id

L → *R

R → L





A Lack of Context

S → E

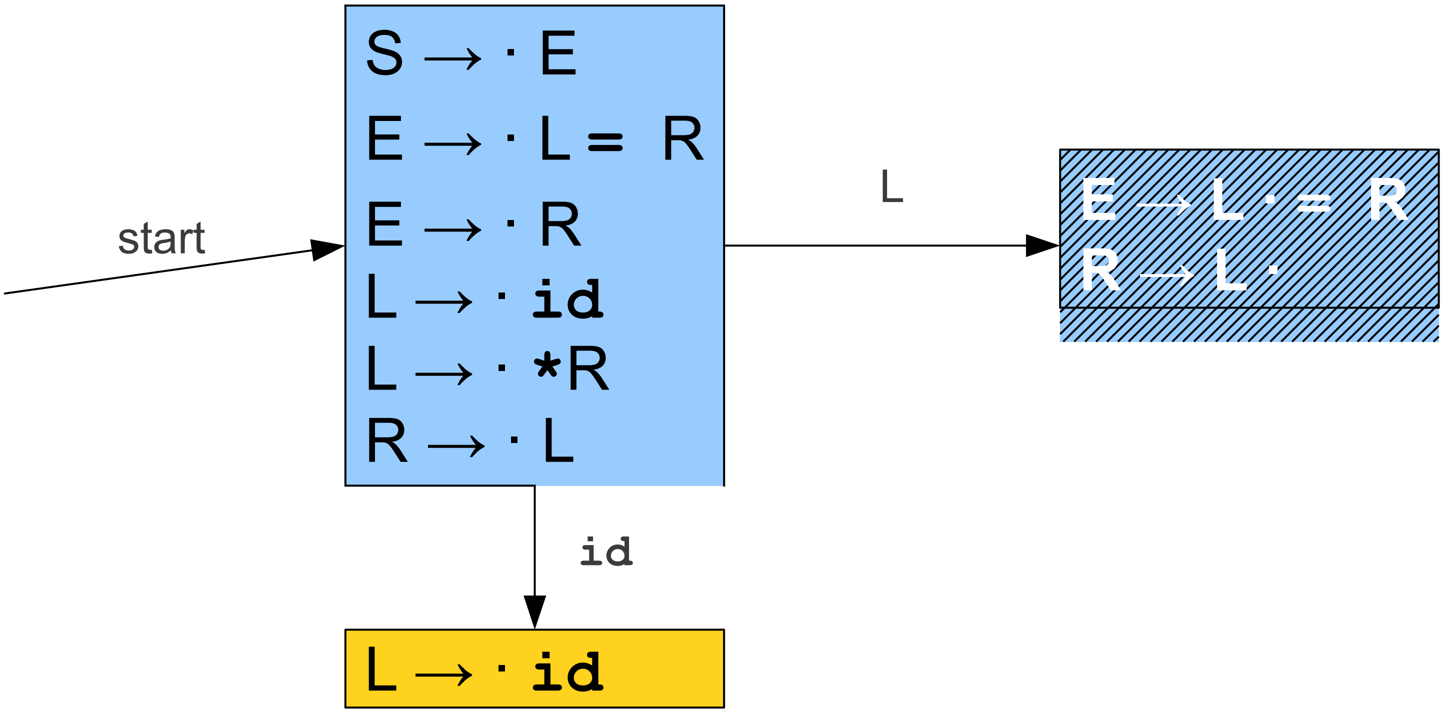
E → L = R

E → R

L → id

L → *R

R → L



start

S → · E

E → · L = R

E → · R

L → · id

L → · *R

R → · L

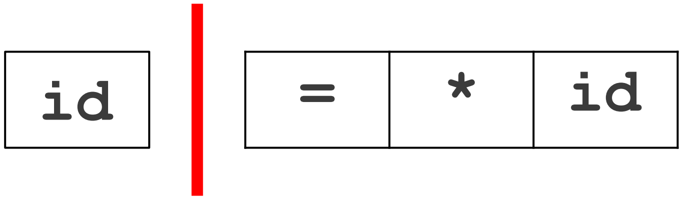
L

E → L · = R

R → L ·

id

L → · id



id

= * id

A Lack of Context

$S \to E$

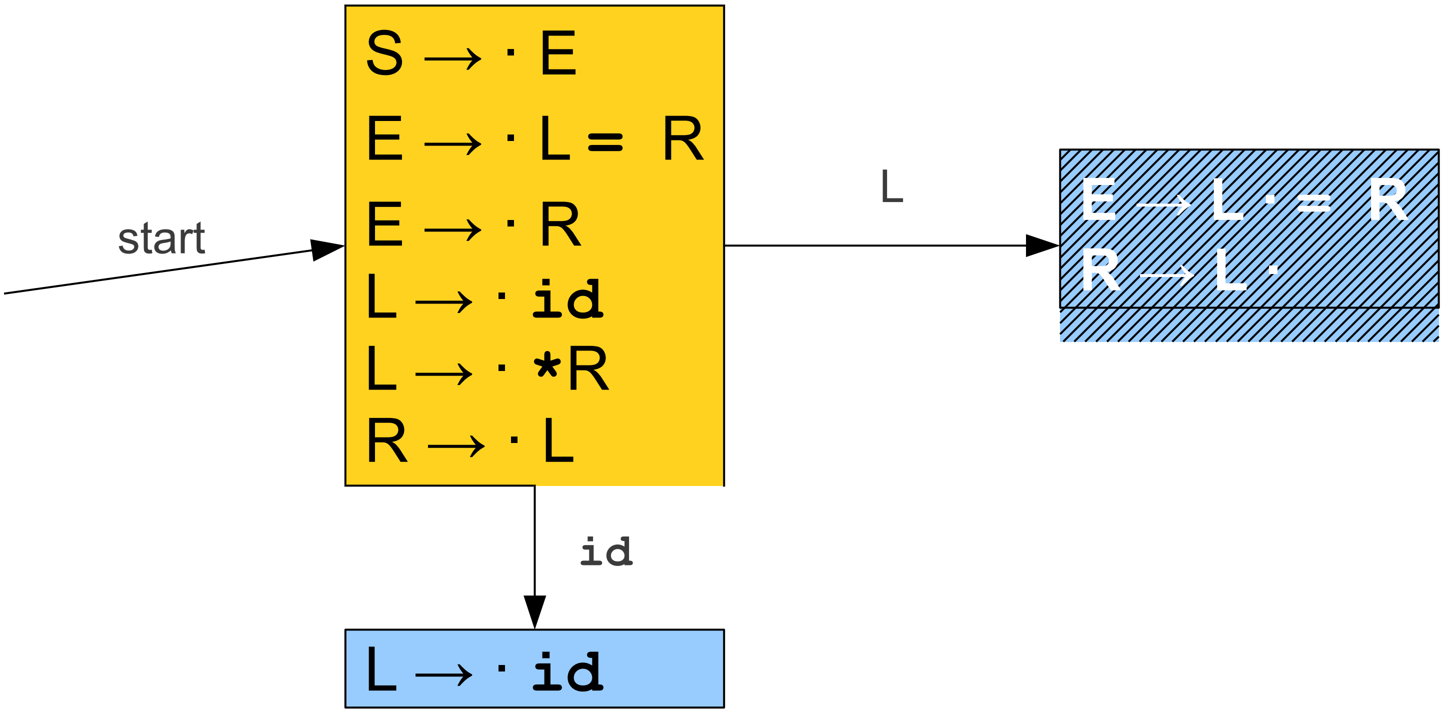
$E \to L$ **=** R

$E \to R$

$L \to$ **id**

$L \to$ *****R

$R \to L$



start

$S \to \cdot E$

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

$E \to \cdot R$

$L \to \cdot$ **id**

$L \to \cdot$ *****R

$R \to \cdot L$

L

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

$R \to L \cdot$

id

$L \to \cdot$ **id**

=	*	id

A Lack of Context

$S \rightarrow E$

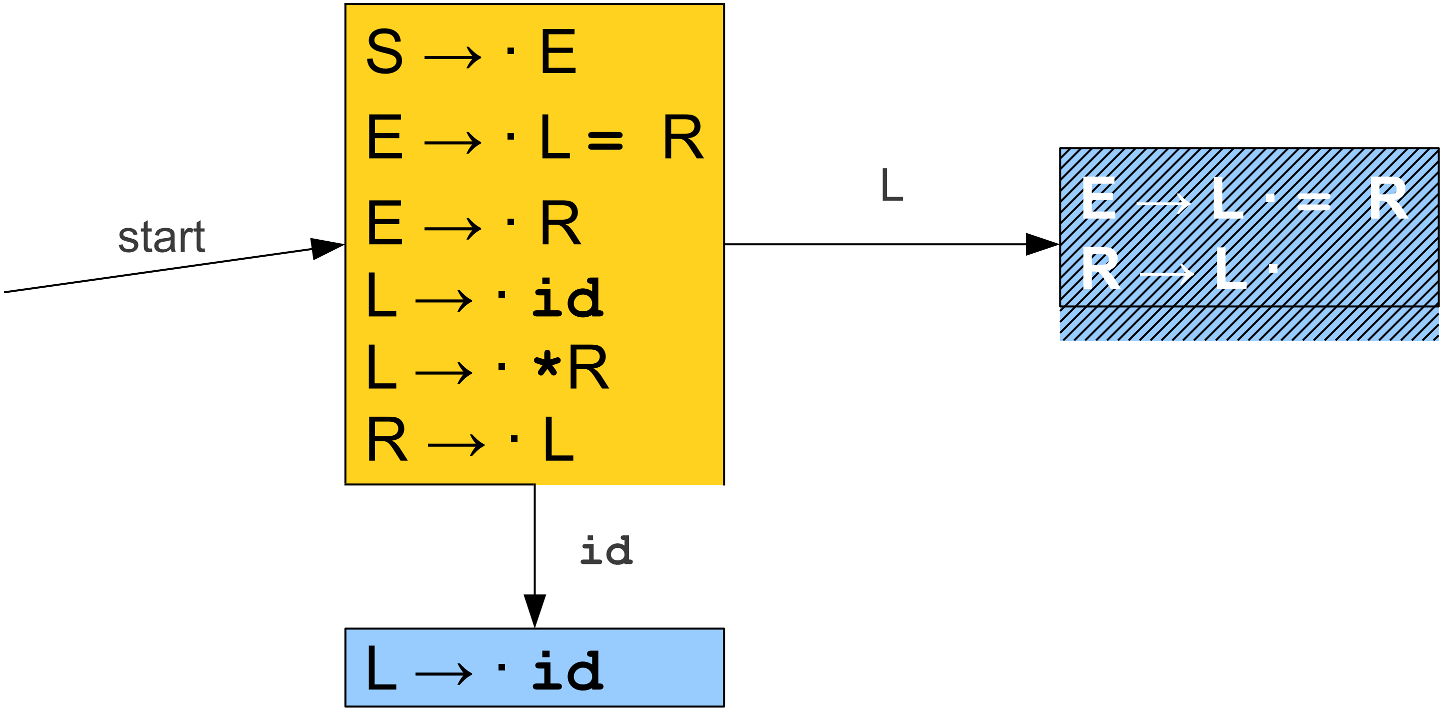
$E \rightarrow L = R$

$E \rightarrow R$

$L \rightarrow \texttt{id}$

$L \rightarrow * R$

$R \rightarrow L$



start

$S \rightarrow \cdot E$

$E \rightarrow \cdot L = R$

$E \rightarrow \cdot R$

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

$L \rightarrow \cdot * R$

$R \rightarrow \cdot L$

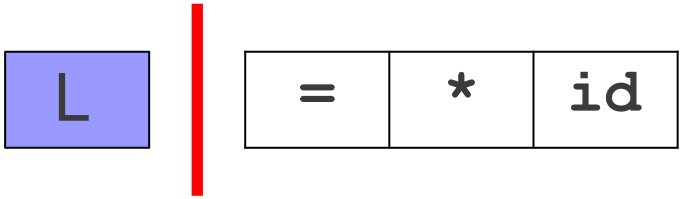
L

$E \rightarrow L \cdot = R$

$R \rightarrow L \cdot$

id

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



A Lack of Context

$S \rightarrow E$

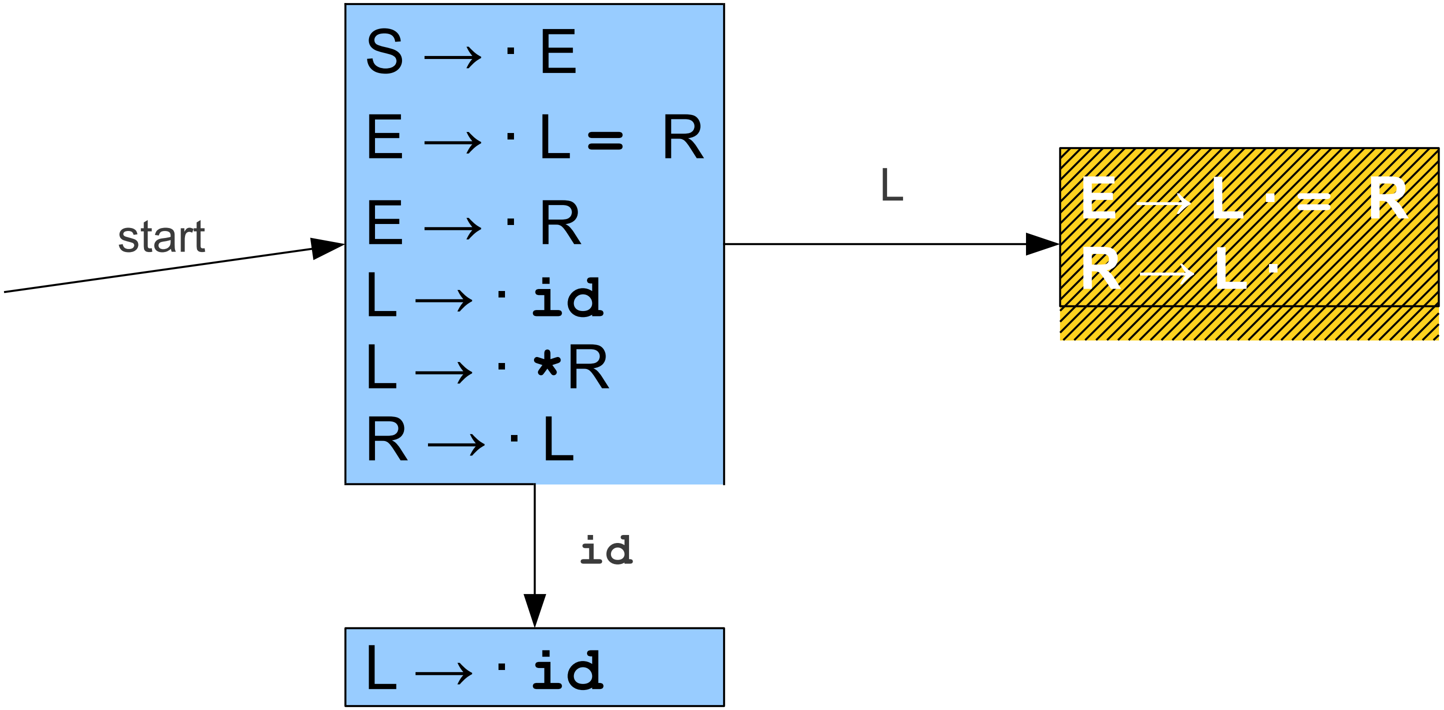
$E \rightarrow L = R$

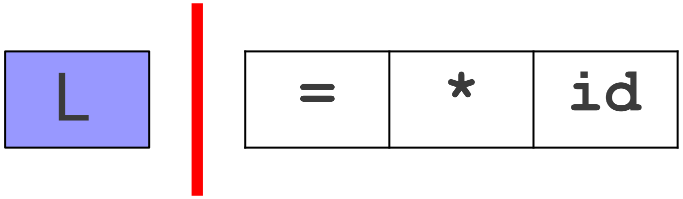
$E \rightarrow R$

$L \rightarrow \texttt{id}$

$L \rightarrow \texttt{*}R$

$R \rightarrow L$





A Lack of Context

S → E

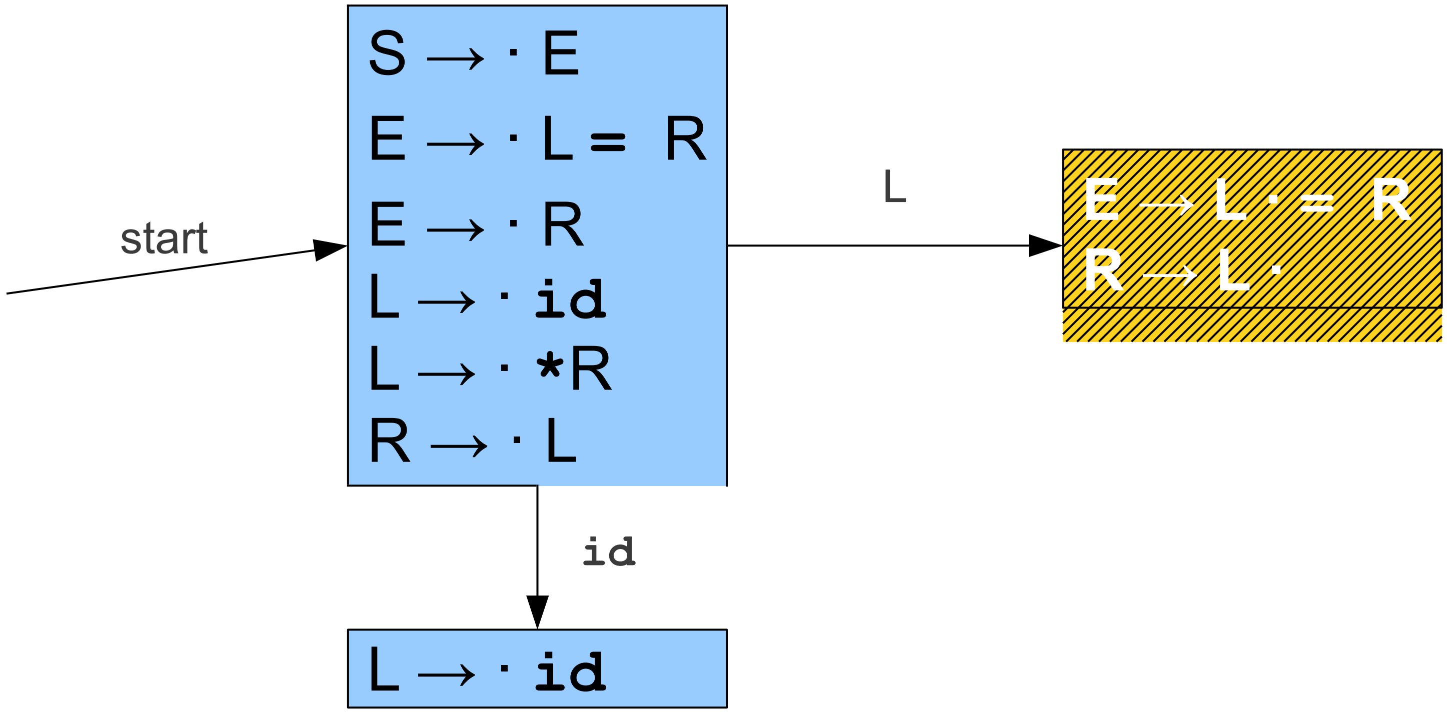
E → L = R

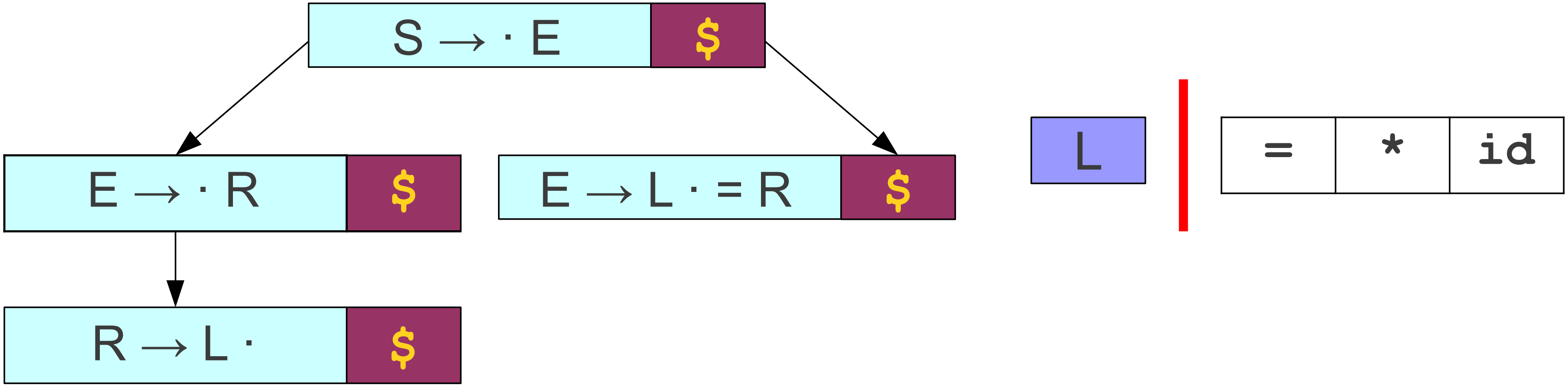
E → R

L → id

L → *R

R → L





A Lack of Context

S → E

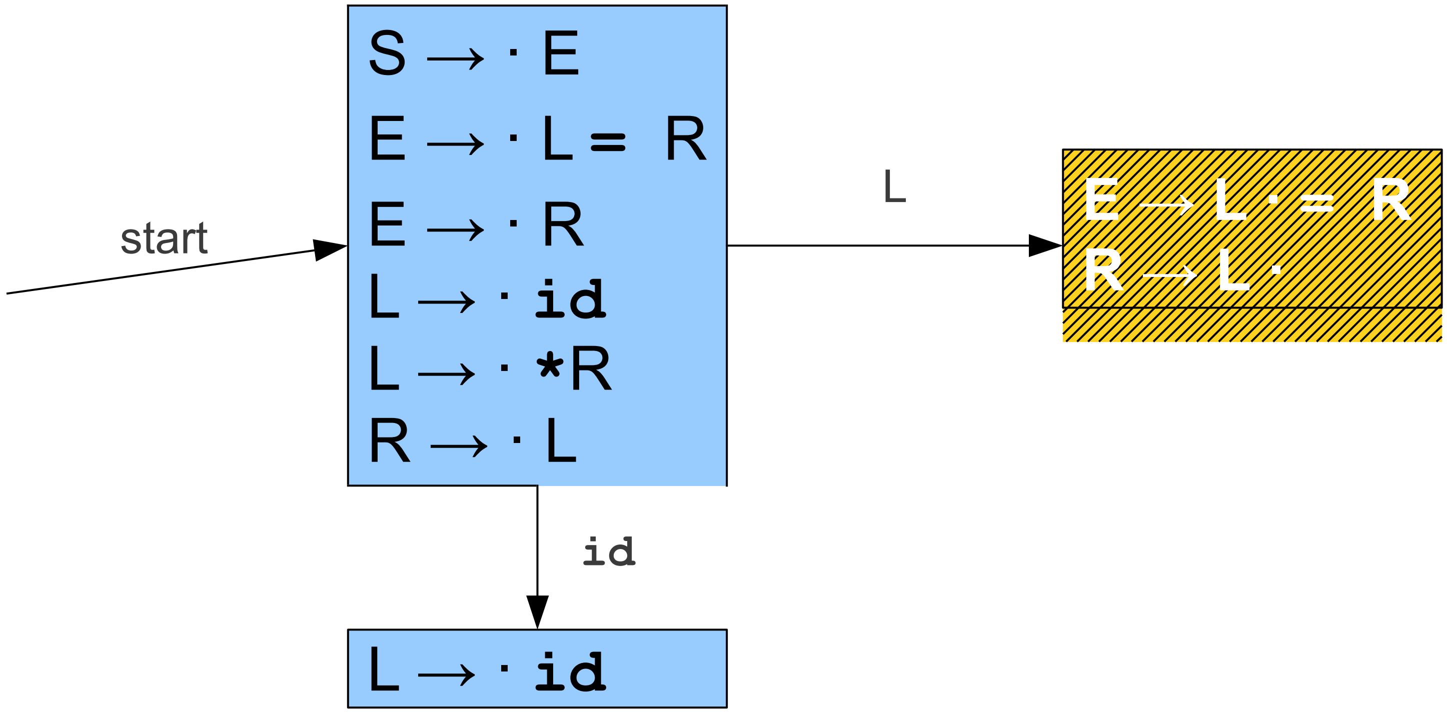
E → L = R

E → R

L → id

L → *R

R → L



start

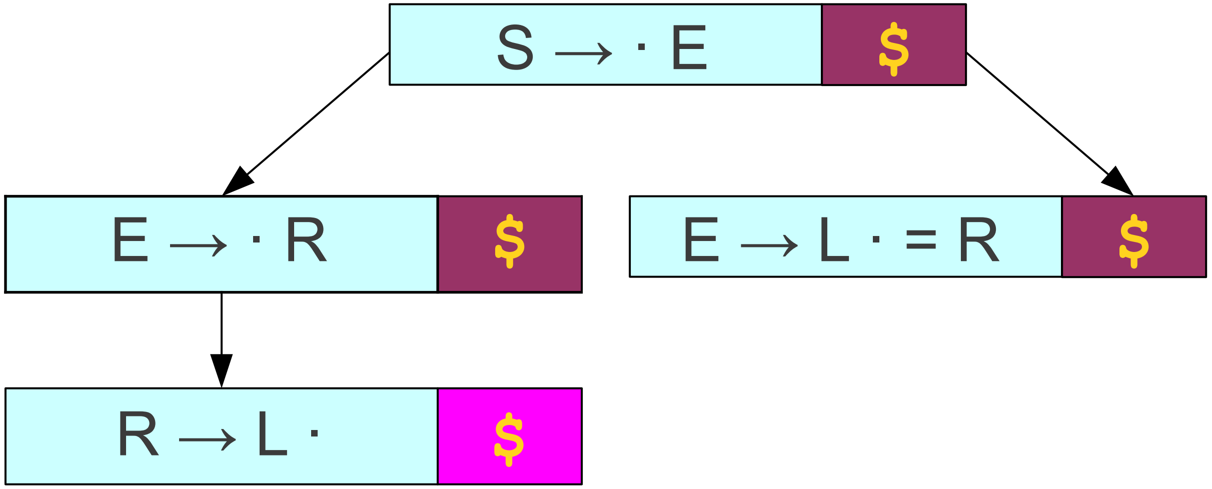
S → · E
E → · L = R
E → · R
L → · id
L → · *R
R → · L

L

E → L · = R
R → L ·

id

L → · id

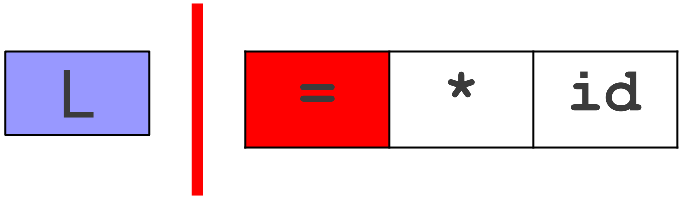


S → · E $

E → · R $

E → L · = R $

R → L · $



L		=	*	id