CS251: Introduction to Language Processing

Bottom-Up Parsing

Vishwesh Jatala

Department of CSE

Indian Institute of Technology Bhilai

vishwesh@iitbhilai.ac.in

2023-24 M

Acknowledgement

- Today's slides are modified from that of *Stanford University:*
  - *https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/*

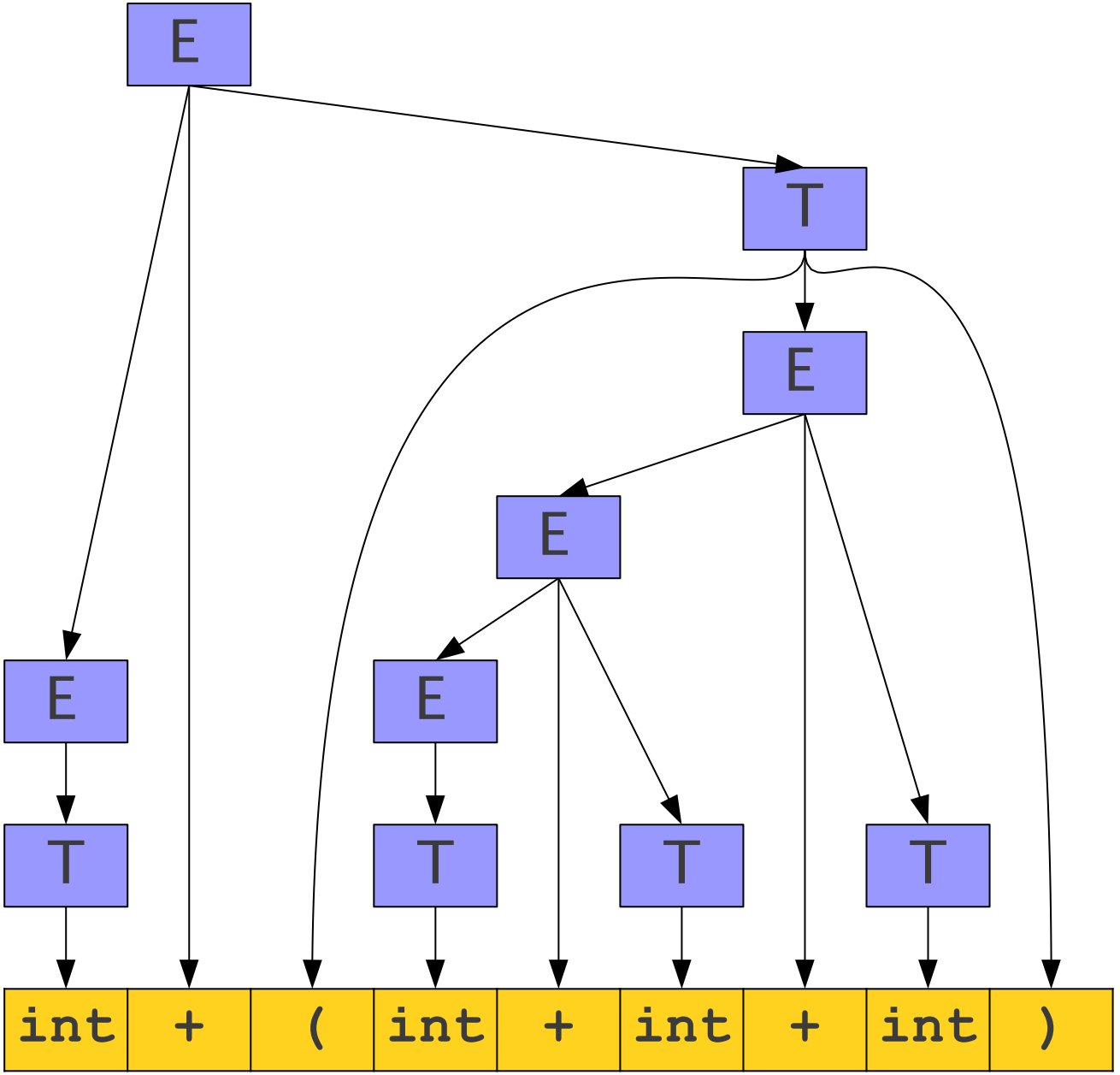
Parsing

$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow \texttt{int}$

$T \rightarrow (E)$



Bottom-Up Parsing - Example

$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow \texttt{int}$

$T \rightarrow (E)$

`int + (int + int + int)`

Bottom-Up Parsing - Example

$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow \texttt{int}$

$T \rightarrow (E)$

`int + (int + int + int)`

Bottom-Up Parsing - Example

E → T
E → E + T
T → int
T → (E)

int + (int + int + int)
⇒ T + (int + int + int)

Bottom-Up Parsing - Example

$E \rightarrow T$
$E \rightarrow E + T$
$T \rightarrow \texttt{int}$
$T \rightarrow (E)$

$\texttt{int} + (\texttt{int} + \texttt{int} + \texttt{int})$
$\Rightarrow T + (\texttt{int} + \texttt{int} + \texttt{int})$
$\Rightarrow E + (\texttt{int} + \texttt{int} + \texttt{int})$

Bottom-Up Parsing - Example

E → T		int + (int + int + int)
E → E + T	⇒	T + (int + int + int)
T → int	⇒	E + (int + int + int)
T → (E)	⇒	E + (T + int + int)

Bottom-Up Parsing - Example

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

Bottom-Up Parsing - Example

$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow \texttt{int}$

$T \rightarrow (E)$

$$\begin{aligned}
& \texttt{int} + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & T + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (T + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (E + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (E + T + \texttt{int})
\end{aligned}$$

Bottom-Up Parsing - Example

$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow \texttt{int}$

$T \rightarrow (E)$

$$\begin{aligned}
& \texttt{int} + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & T + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (T + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (E + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (E + T + \texttt{int}) \\
\Rightarrow\ & E + (E + \texttt{int})
\end{aligned}$$

Bottom-Up Parsing - Example

$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow \texttt{int}$

$T \rightarrow (E)$

$$\begin{aligned}
& \texttt{int} + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & T + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (T + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (E + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (E + T + \texttt{int}) \\
\Rightarrow\ & E + (E + \texttt{int}) \\
\Rightarrow\ & E + (E + T)
\end{aligned}$$

Bottom-Up Parsing - Example

$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow \texttt{int}$

$T \rightarrow (E)$

$$
\begin{aligned}
& \texttt{int} + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & T + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (T + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (E + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (E + T + \texttt{int}) \\
\Rightarrow\ & E + (E + \texttt{int}) \\
\Rightarrow\ & E + (E + T) \\
\Rightarrow\ & E + (E)
\end{aligned}
$$

Bottom-Up Parsing - Example

$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow int$

$T \rightarrow (E)$

$$
\begin{aligned}
& int + (int + int + int) \\
\Rightarrow\ & T + (int + int + int) \\
\Rightarrow\ & E + (int + int + int) \\
\Rightarrow\ & E + (T + int + int) \\
\Rightarrow\ & E + (E + int + int) \\
\Rightarrow\ & E + (E + T + int) \\
\Rightarrow\ & E + (E + int) \\
\Rightarrow\ & E + (E + T) \\
\Rightarrow\ & E + (E) \\
\Rightarrow\ & E + T
\end{aligned}
$$

Bottom-Up Parsing - Example

$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow \texttt{int}$

$T \rightarrow (E)$

$$
\begin{aligned}
& \texttt{int} + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & T + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (\texttt{int} + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (T + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (E + \texttt{int} + \texttt{int}) \\
\Rightarrow\ & E + (E + T + \texttt{int}) \\
\Rightarrow\ & E + (E + \texttt{int}) \\
\Rightarrow\ & E + (E + T) \\
\Rightarrow\ & E + (E) \\
\Rightarrow\ & E + T \\
\Rightarrow\ & E
\end{aligned}
$$

Overview of Bottom-Up Parsing

$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow \texttt{int}$

$T \rightarrow (E)$

```
   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.

Handles

- The basic steps of a bottom-up parser are
  - to identify a substring within a rightmost sentential form which matches the RHS of a rule.
  - when this substring is replaced by the LHS of the matching rule, it must produce the previous rightmost-sentential form.
- Such a substring is called a handle
- A left-to-right, bottom-up parse works by iteratively searching for a handle, then reducing the handle.

Finding Handles

- Where do we look for handles?

- How do we search for handles?
  - What algorithm do we use to try to discover a handle?
- How do we recognize handles?
  - Once we've found a possible handle, how do we confirm that it's correct?

Question One:

Where are handles?

A Sample Shift/Reduce Parse

E → F

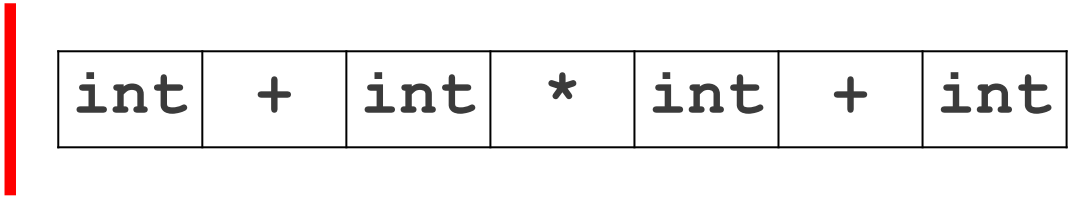
E → E + F

F → F * T

F → T

T → int

T → (E)



A Sample Shift/Reduce Parse

E → F

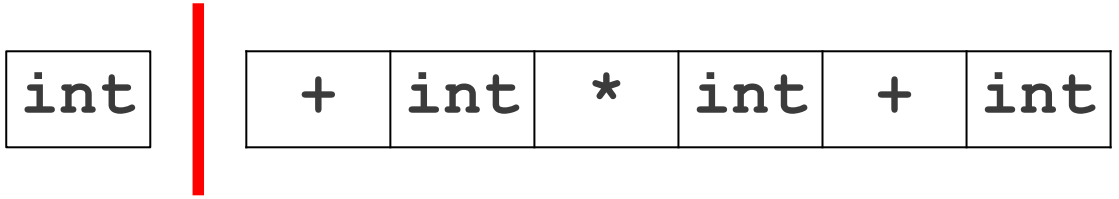
E → E + F

F → F * T

F → T

T → int

T → (E)



A Sample Shift/Reduce Parse

E → F

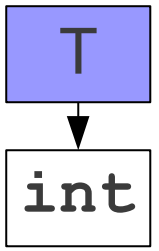
E → E + F

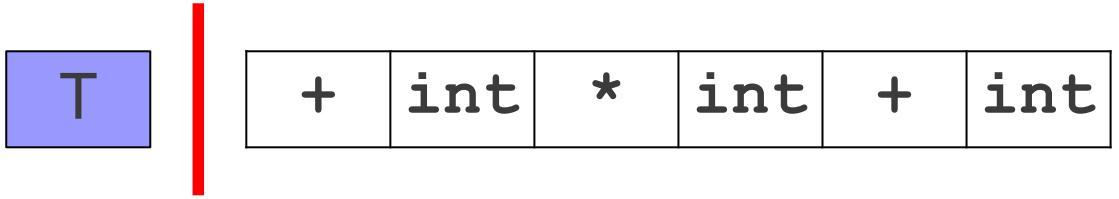
F → F * T

F → T

T → int

T → (E)





A Sample Shift/Reduce Parse

E → F

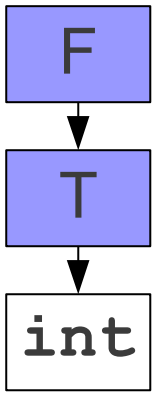
E → E + F

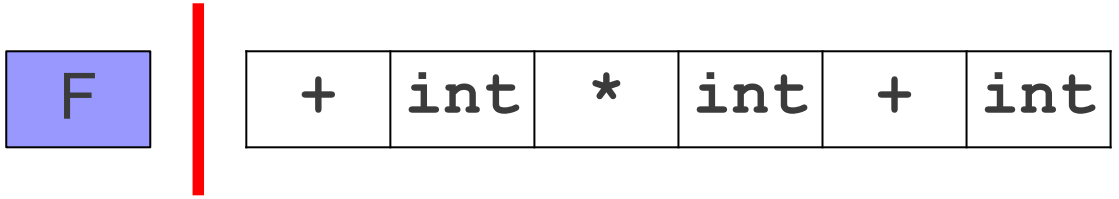
F → F * T

F → T

T → int

T → (E)





A Sample Shift/Reduce Parse

E → F

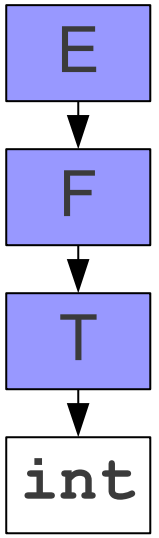
E → E + F

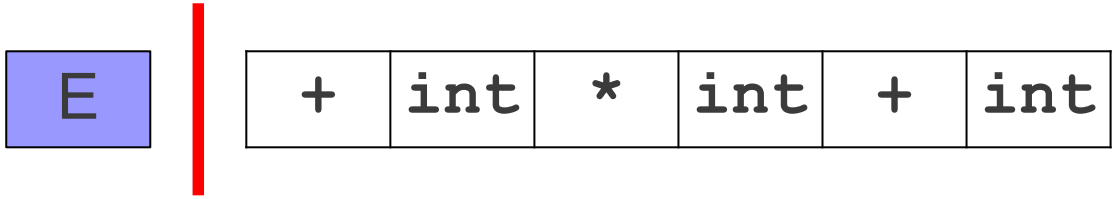
F → F * T

F → T

T → int

T → (E)





A Sample Shift/Reduce Parse

E → F

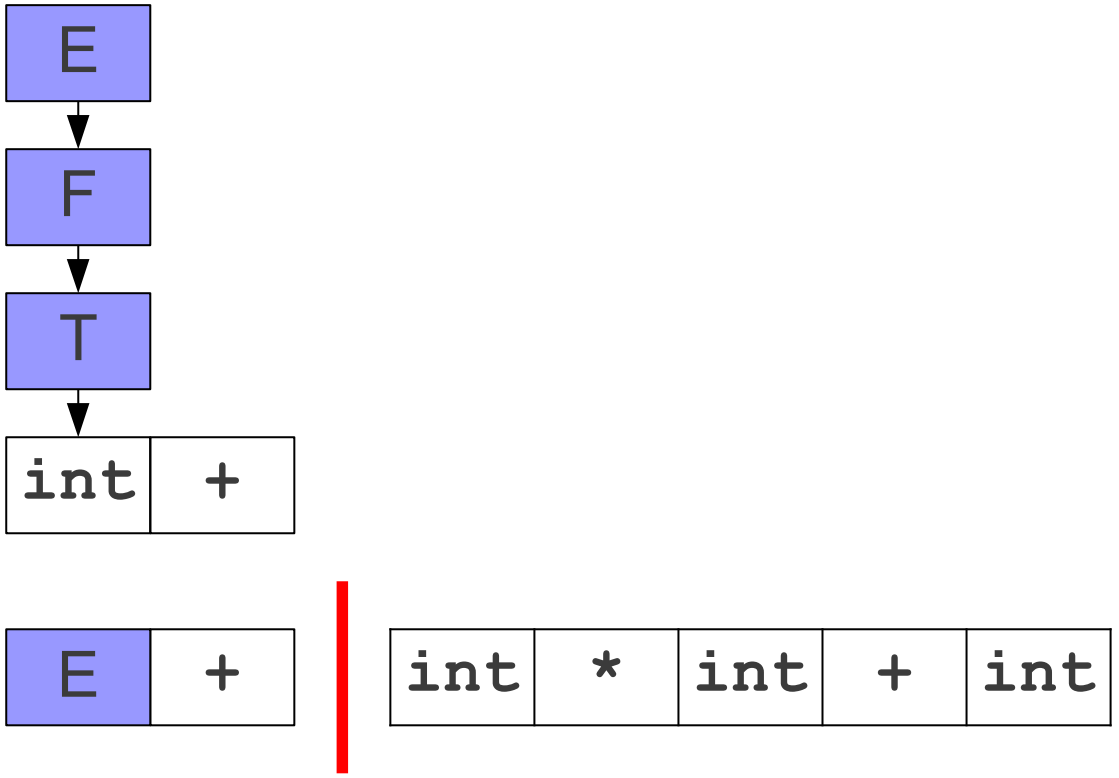
E → E + F

F → F * T

F → T

T → int

T → (E)



A Sample Shift/Reduce Parse

E → F

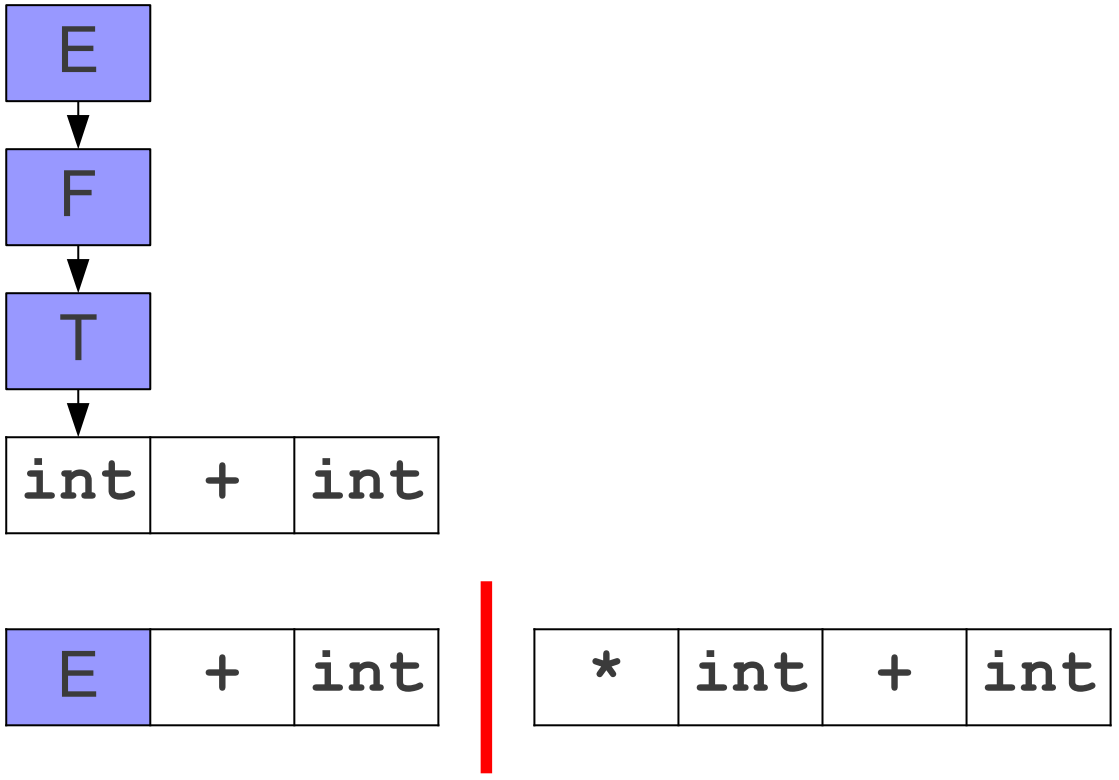
E → E + F

F → F * T

F → T

T → int

T → (E)



A Sample Shift/Reduce Parse

E → F

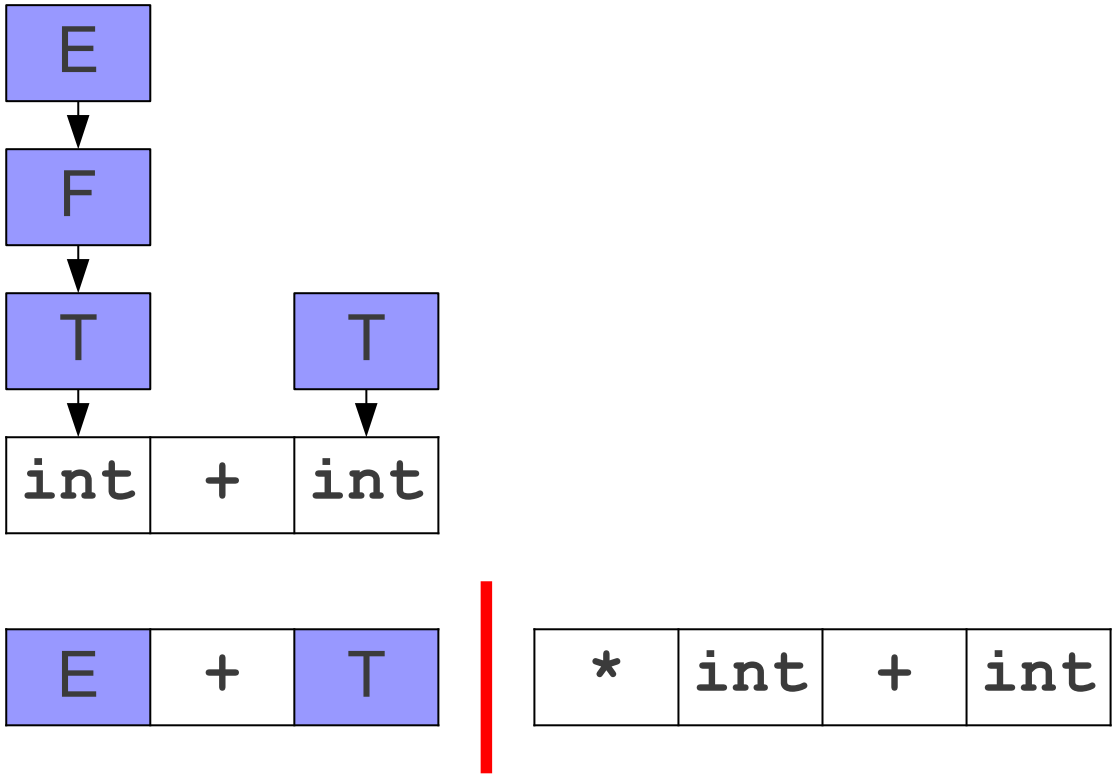
E → E + F

F → F * T

F → T

T → int

T → (E)



A Sample Shift/Reduce Parse

E → F

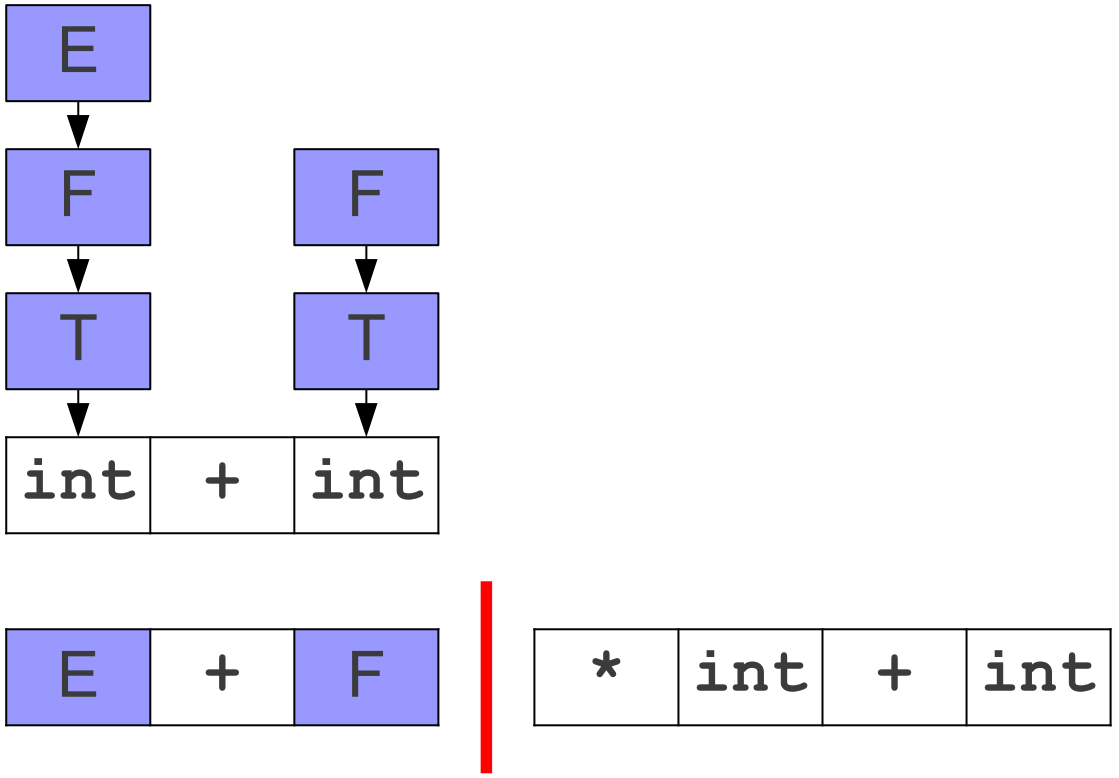
E → E + F

F → F * T

F → T

T → int

T → (E)



A Sample Shift/Reduce Parse

E → F

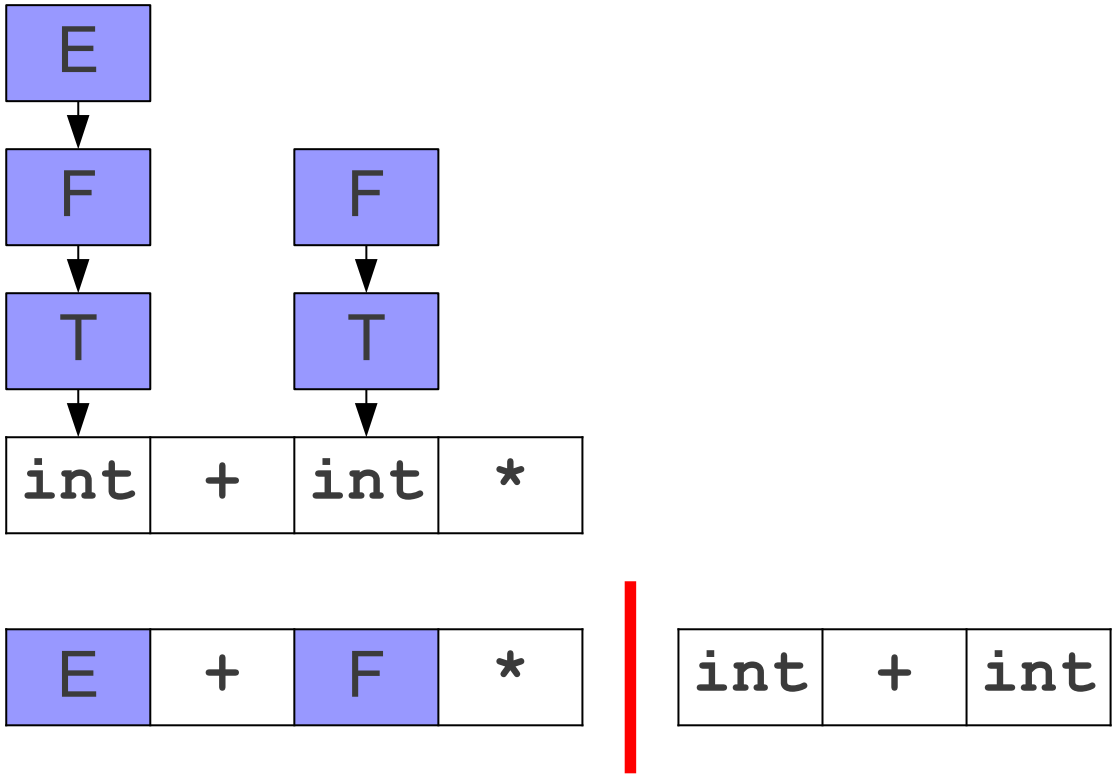
E → E + F

F → F * T

F → T

T → int

T → (E)



A Sample Shift/Reduce Parse

E → F

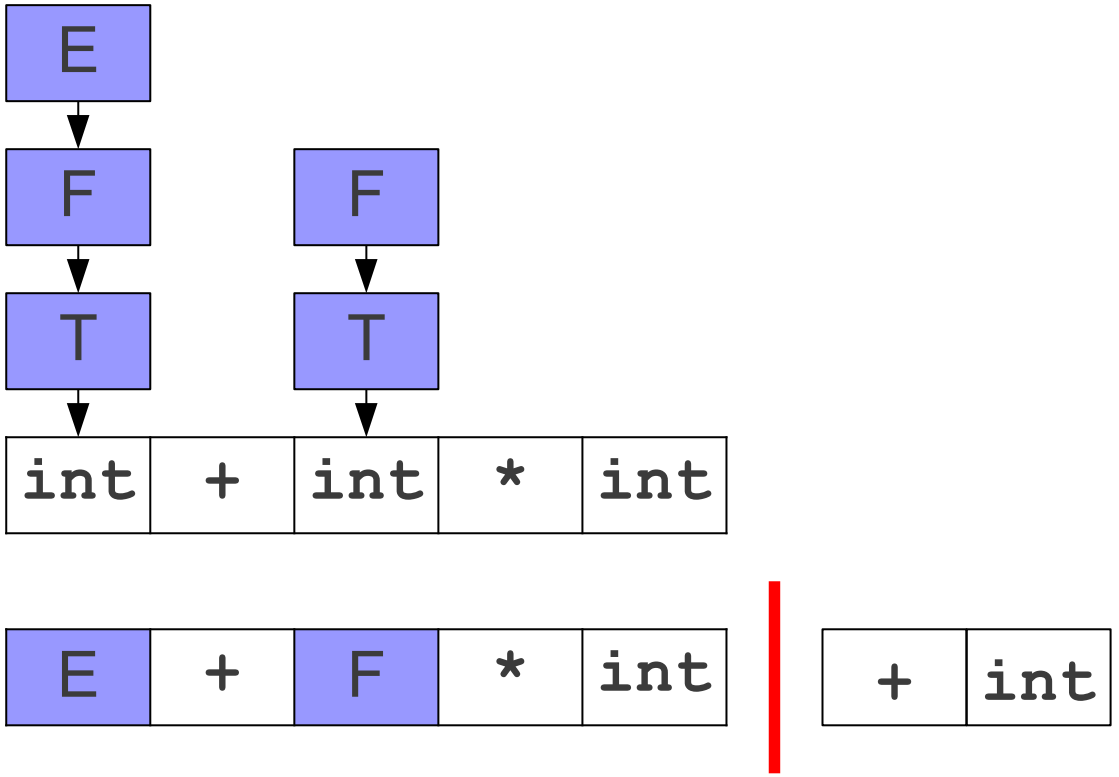
E → E + F

F → F * T

F → T

T → int

T → (E)



A Sample Shift/Reduce Parse

E → F

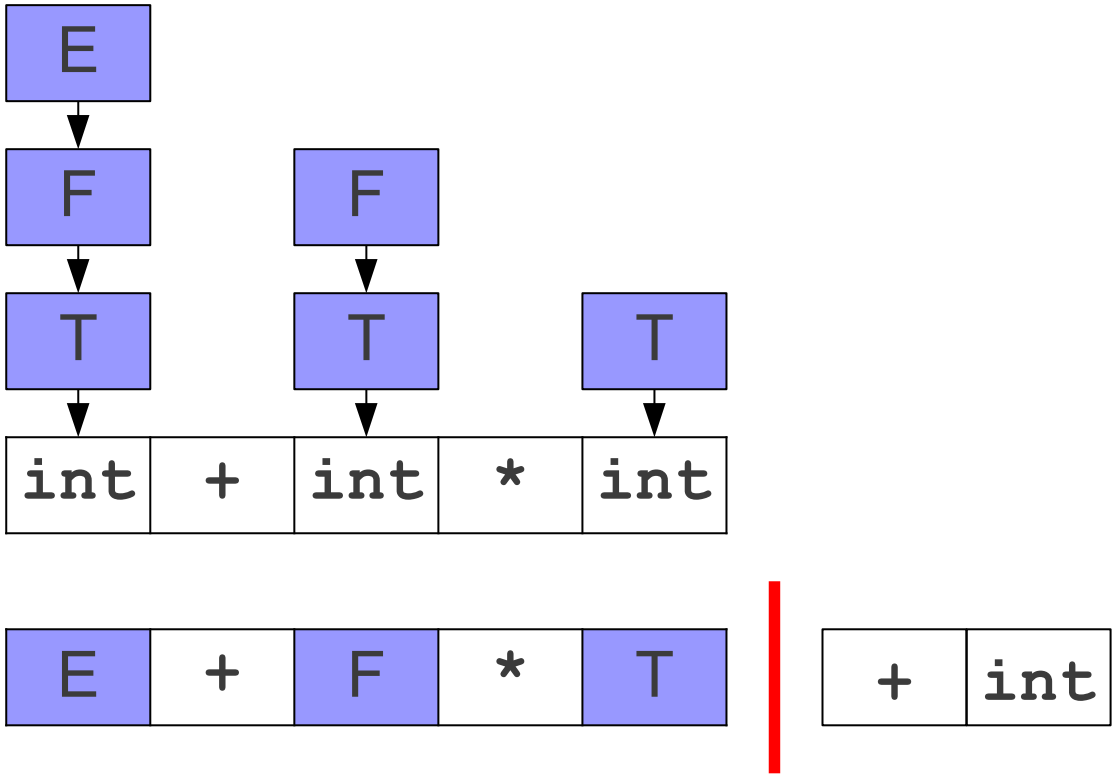
E → E + F

F → F * T

F → T

T → int

T → (E)



A Sample Shift/Reduce Parse

E → F

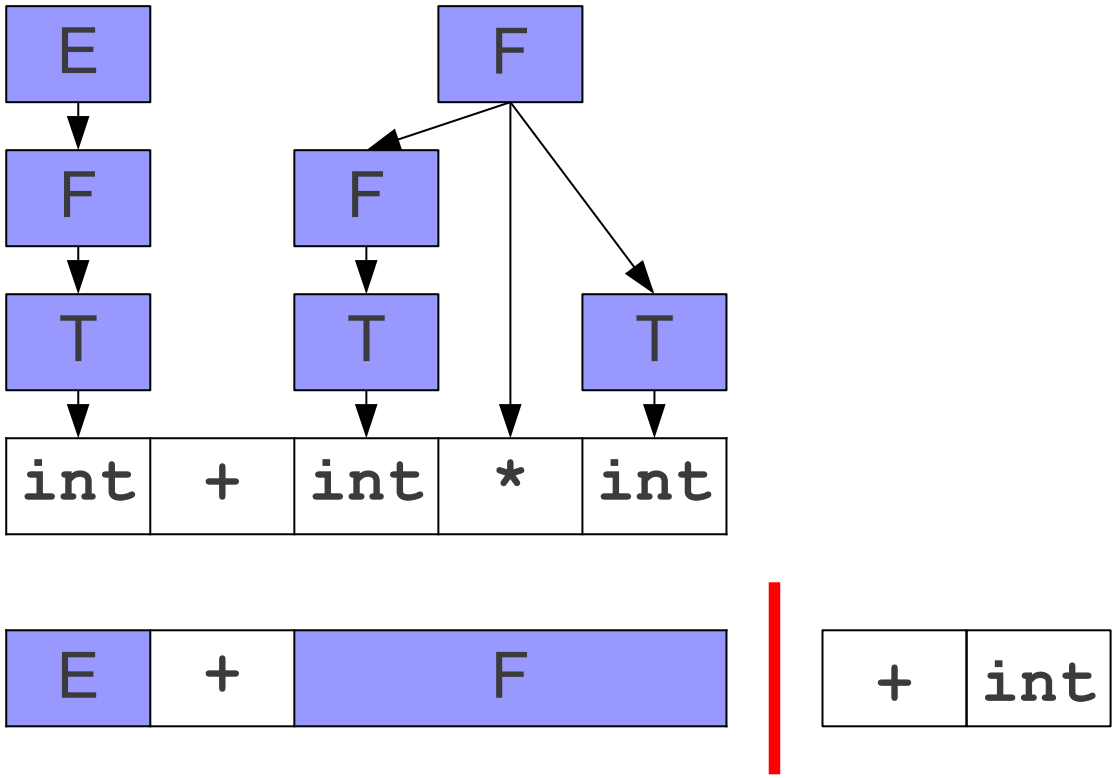
E → E + F

F → F * T

F → T

T → int

T → (E)



A Sample Shift/Reduce Parse

E → F

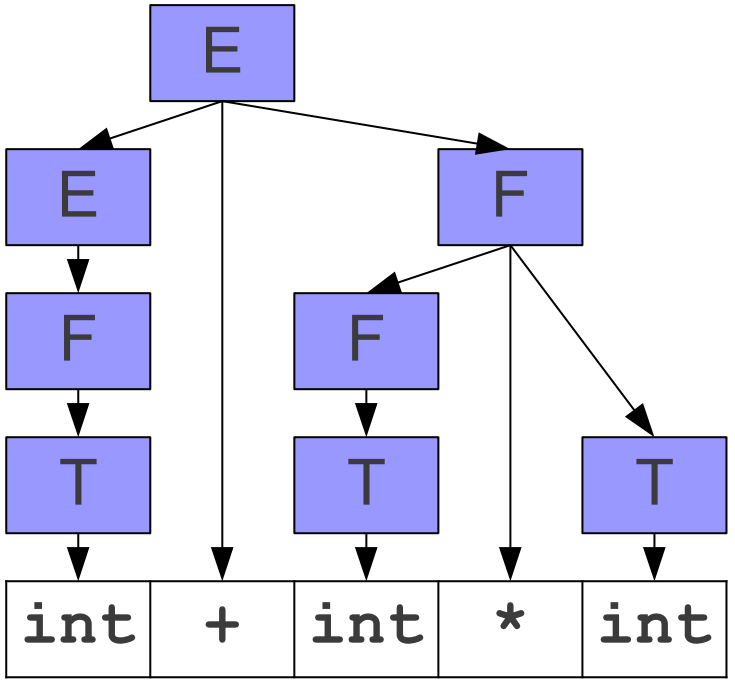
E → E + F

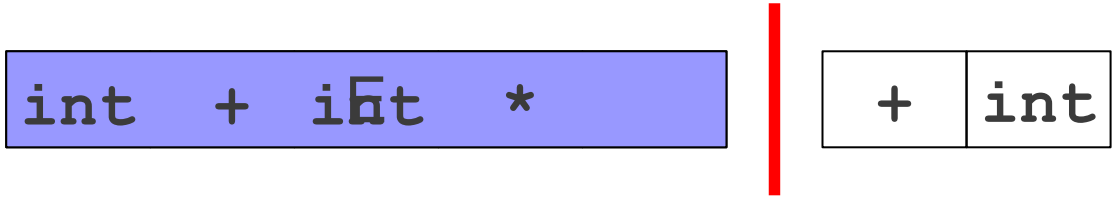
F → F * T

F → T

T → int

T → (E)





A Sample Shift/Reduce Parse

$E \rightarrow F$

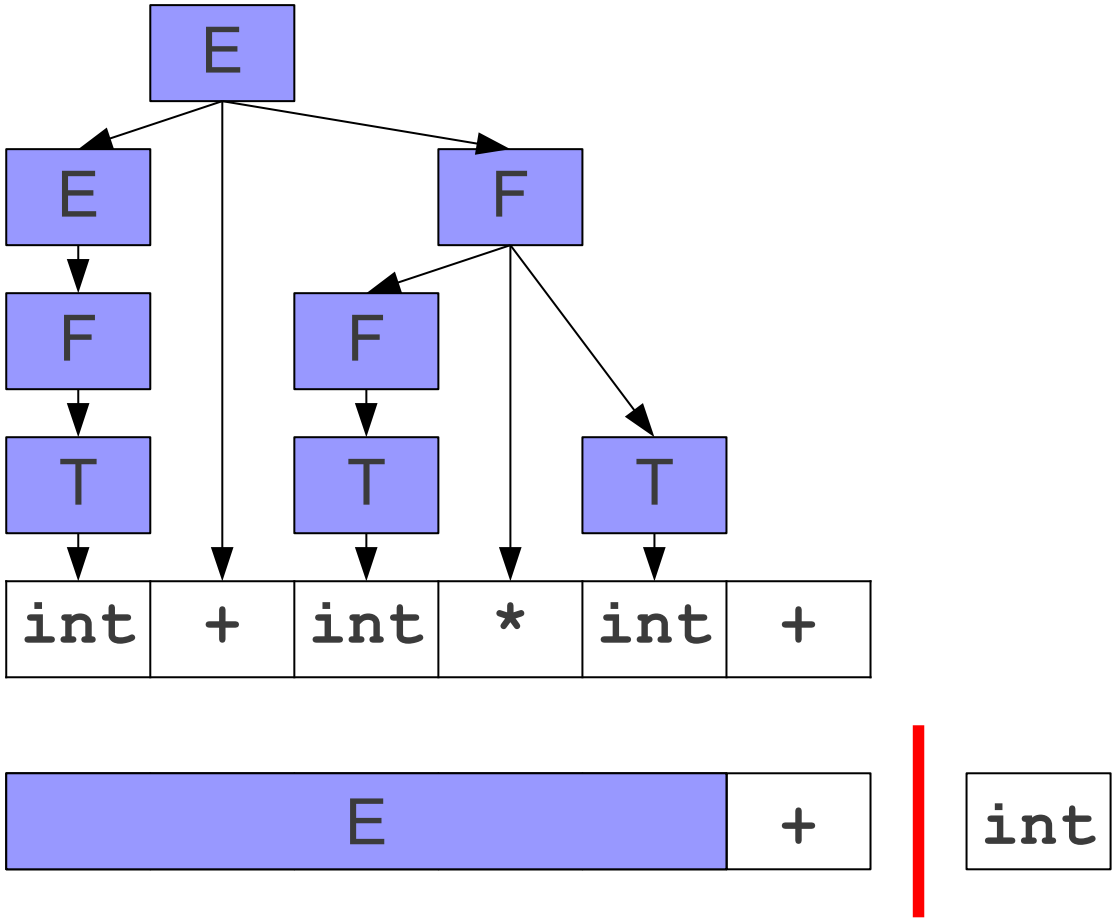
$E \rightarrow E + F$

$F \rightarrow F * T$

$F \rightarrow T$

$T \rightarrow \texttt{int}$

$T \rightarrow (E)$



A Sample Shift/Reduce Parse

$E \rightarrow F$

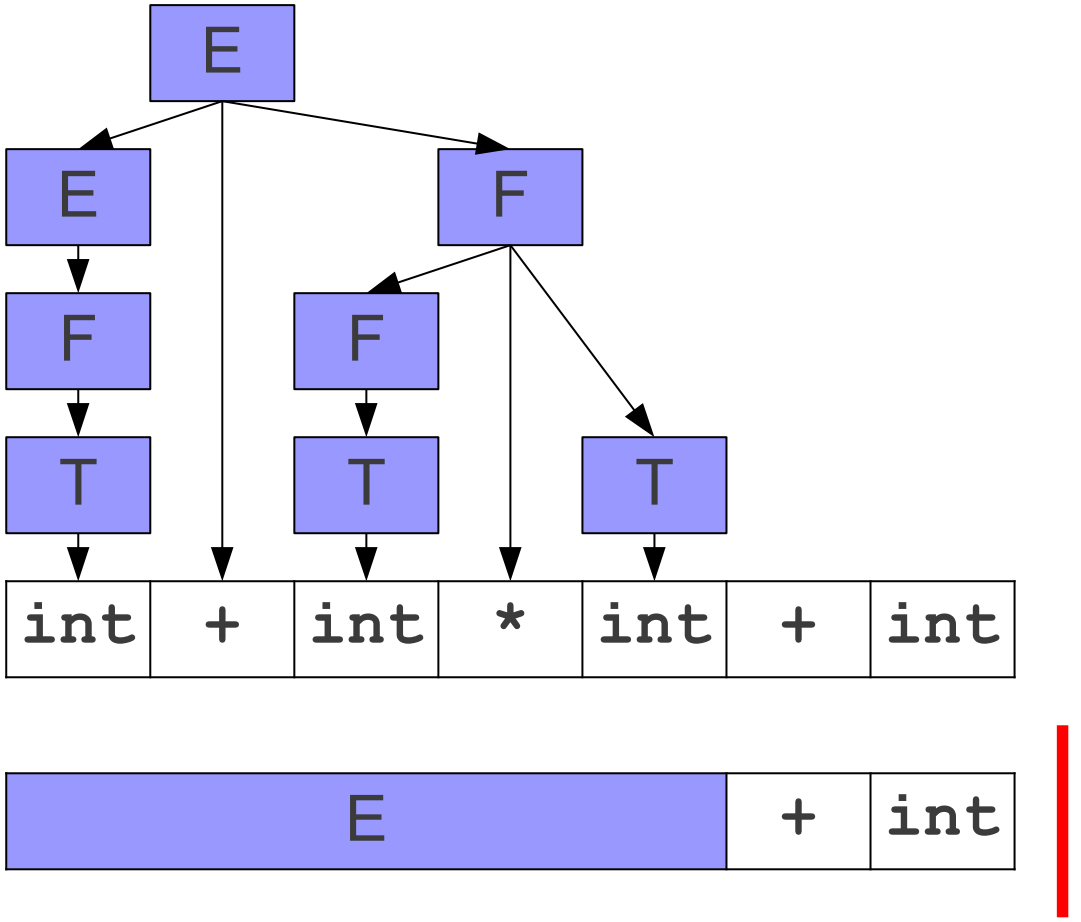
$E \rightarrow E + F$

$F \rightarrow F * T$

$F \rightarrow T$

$T \rightarrow \texttt{int}$

$T \rightarrow (E)$



A Sample Shift/Reduce Parse

$E \rightarrow F$

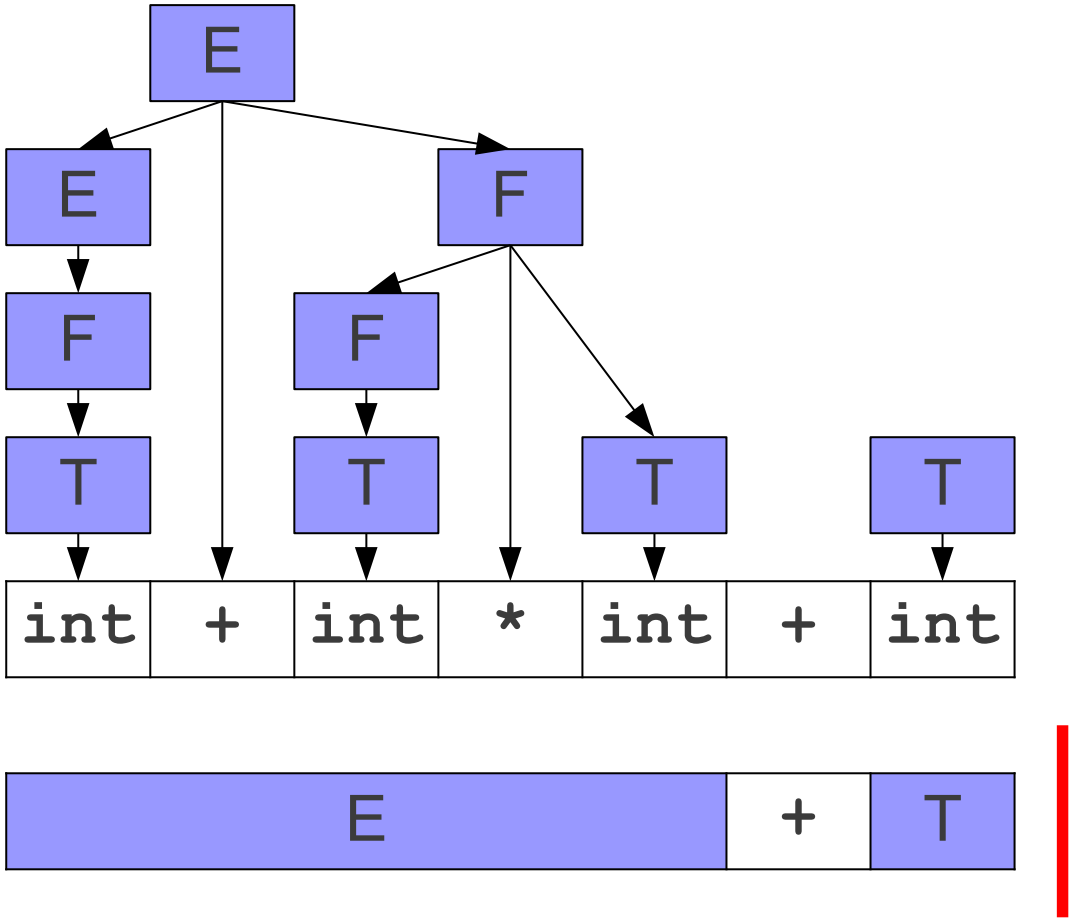
$E \rightarrow E + F$

$F \rightarrow F * T$

$F \rightarrow T$

$T \rightarrow \texttt{int}$

$T \rightarrow (E)$



A Sample Shift/Reduce Parse

E → F

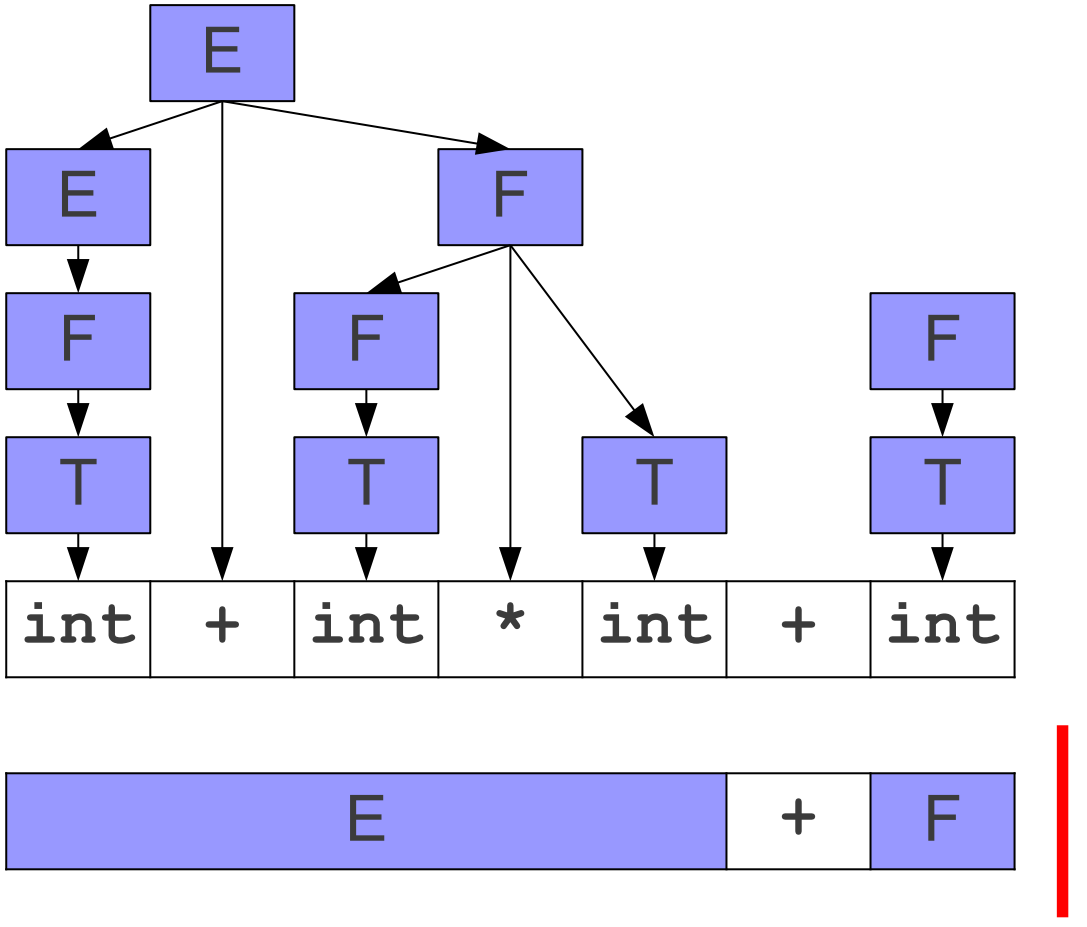
E → E + F

F → F * T

F → T

T → int

T → (E)



A Sample Shift/Reduce Parse

E → F

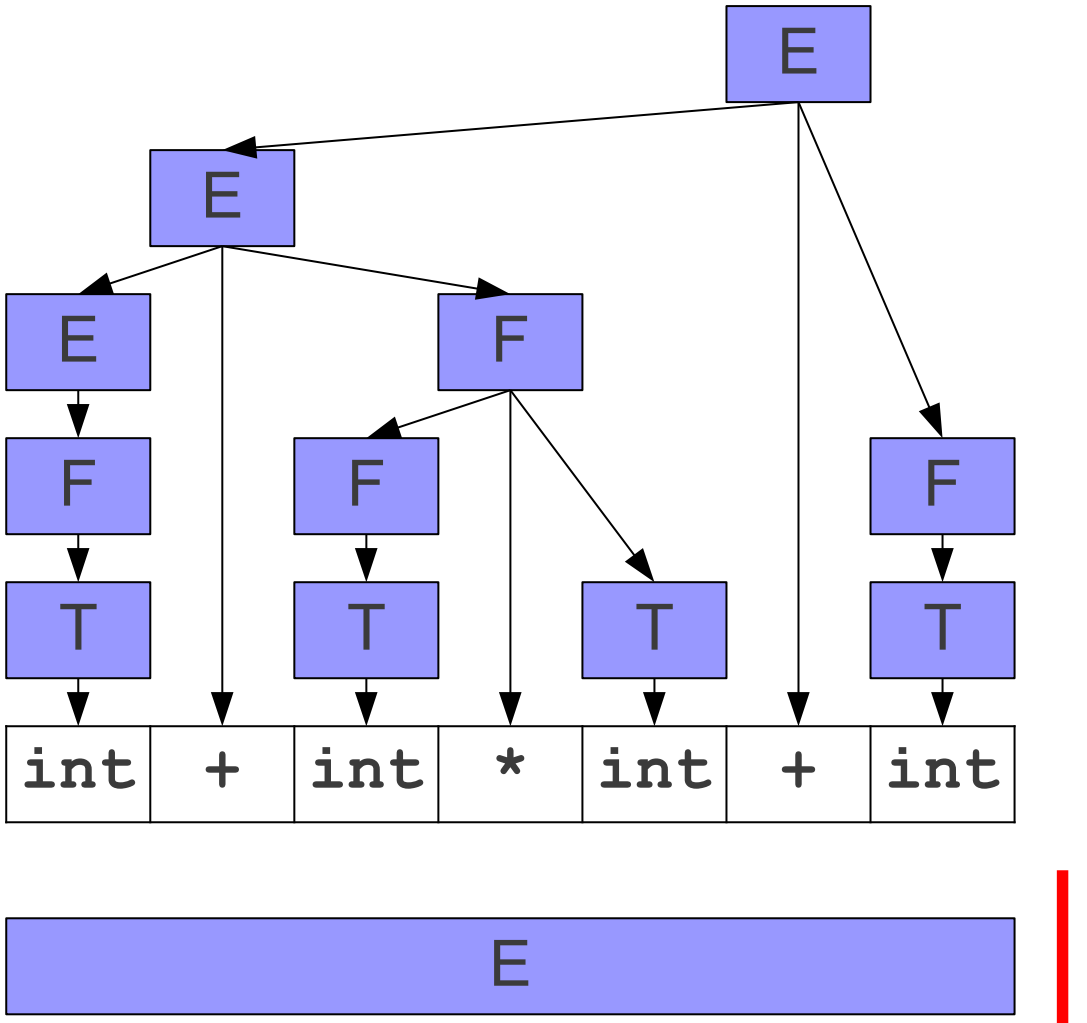
E → E + F

F → F * T

F → T

T → int

T → (E)



Shift/Reduce Parsing

- Shift/reduce parsing means

  Shift: Move a terminal from the right to the left area.

  Reduce: Replace some number of symbols at the right side of the left area.

Finding Handles

- Where do we look for handles?
  - **At the top of the stack.**
- How do we search for handles?
  - What algorithm do we use to try to discover a handle?
- How do we recognize handles?
  - Once we've found a possible handle, how do we confirm that it's correct?

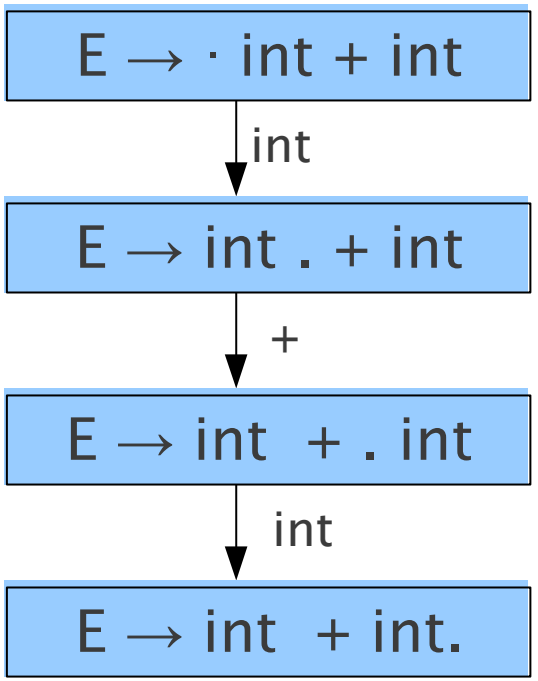
Question Two:

How do we search for handles?

Exploring the Left Side

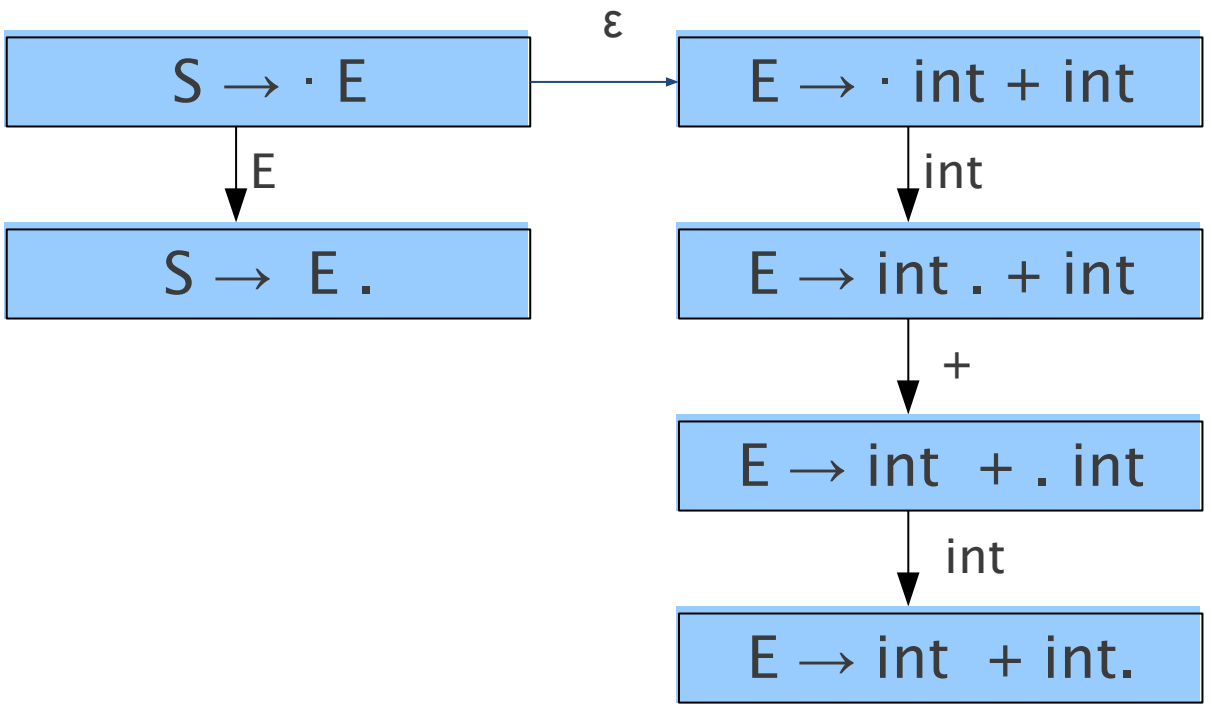
- The handle will always appear at the end of string in the left side of the parser.
- Can *any* string appear on the left side of the parser, or are there restrictions on what sorts of strings can appear there?
- If we can find a pattern to the strings that can appear on the left side, we might be able to exploit it to detect handles.

How to Track Handles?



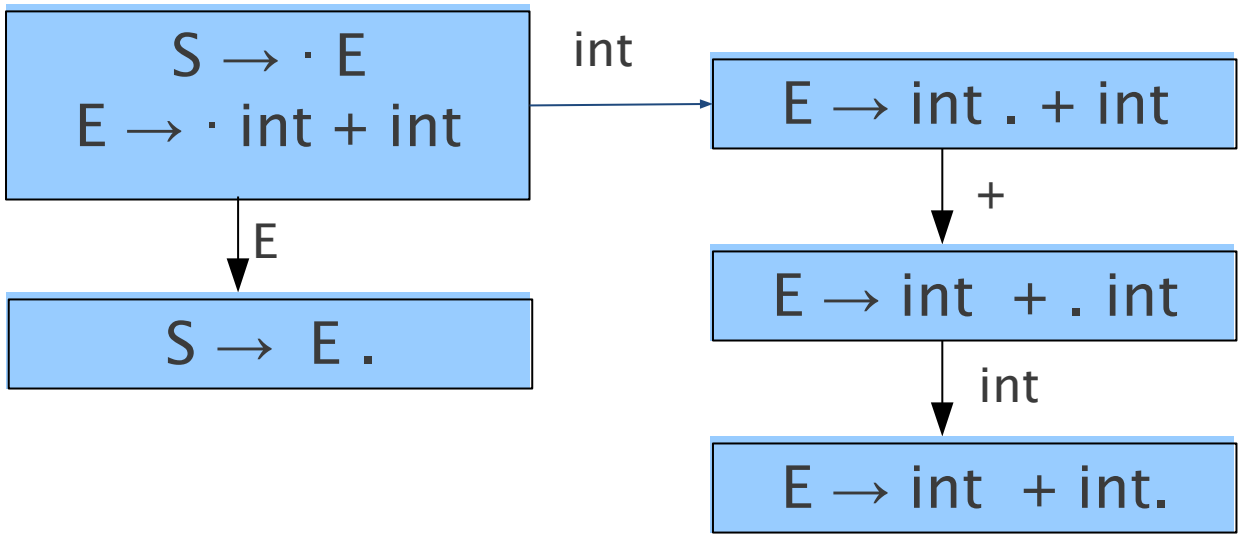
E → int + int

How to Track Handles?



S → E
E → int + int

How to Track Handles?



S → E
E → int + int

A Deterministic Automaton

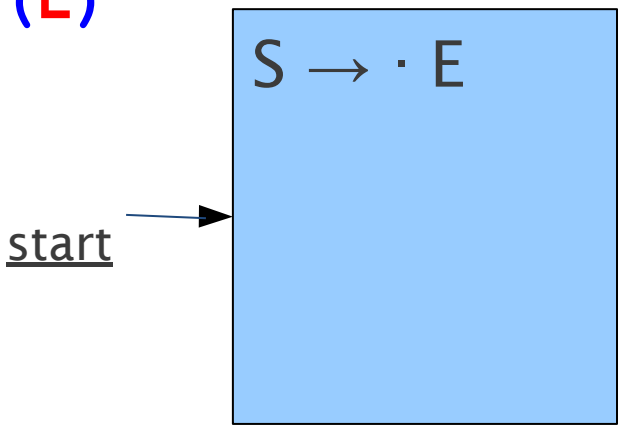
S → E

E → T;

E → T + E

T → int

T → (E)



A Deterministic Automaton

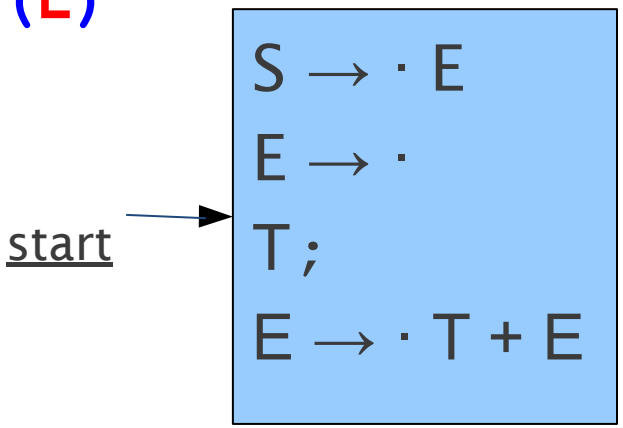
S → E

E → T;

E → T + E

T → int

T → (E)



A Deterministic Automaton

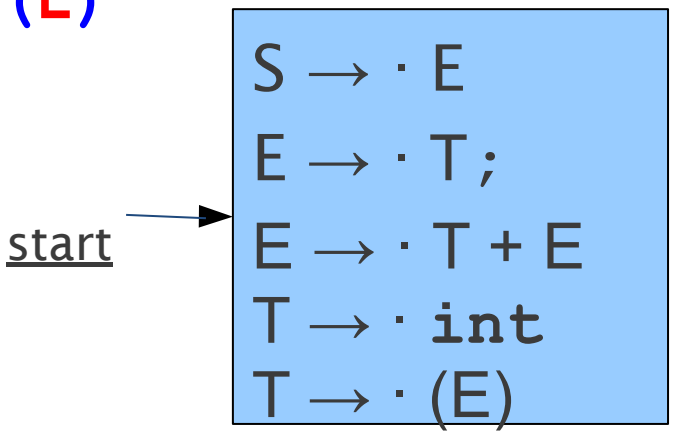
S → E

E → T;

E → T + E

T → int

T → (E)

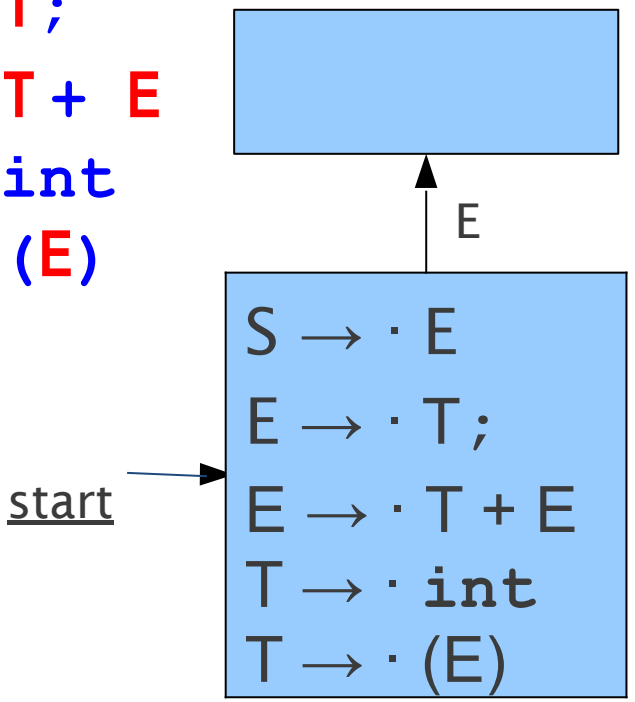


Constructing the Automaton

- Begin in a state containing $S \rightarrow \cdot A$, where **S** is the augmented start symbol.
- Compute the **closure** of the state:
  - If $A \rightarrow a \cdot B\omega$ is in the state, add $B \rightarrow \cdot \gamma$ to the state for each production $B \rightarrow \gamma$.

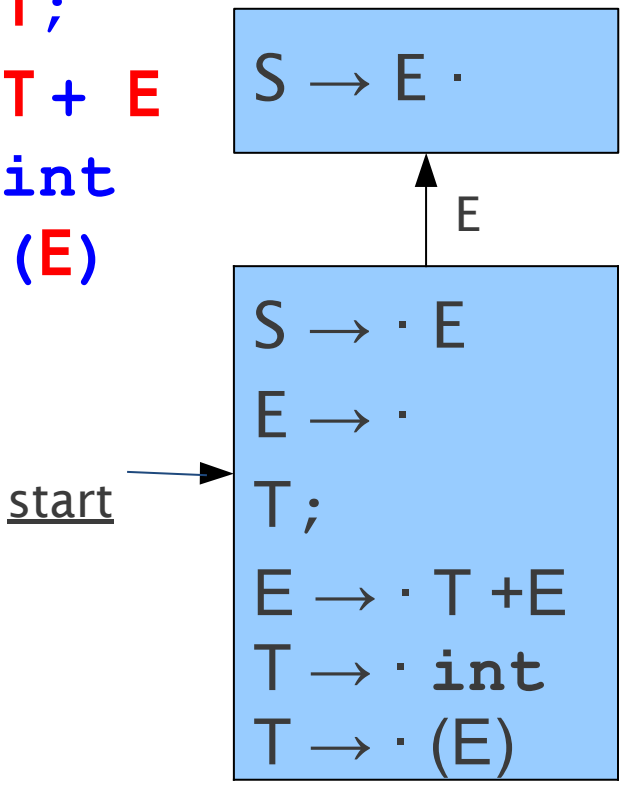
A Deterministic Automaton

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



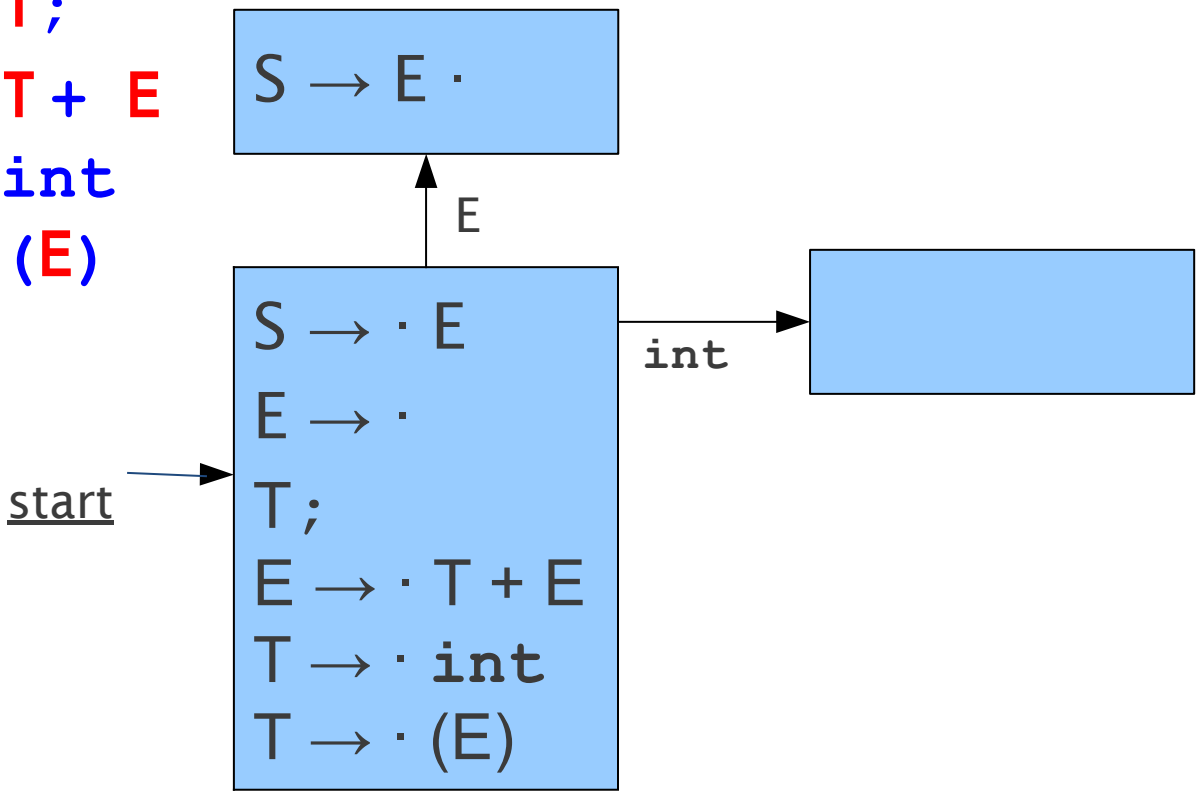
A Deterministic Automaton

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



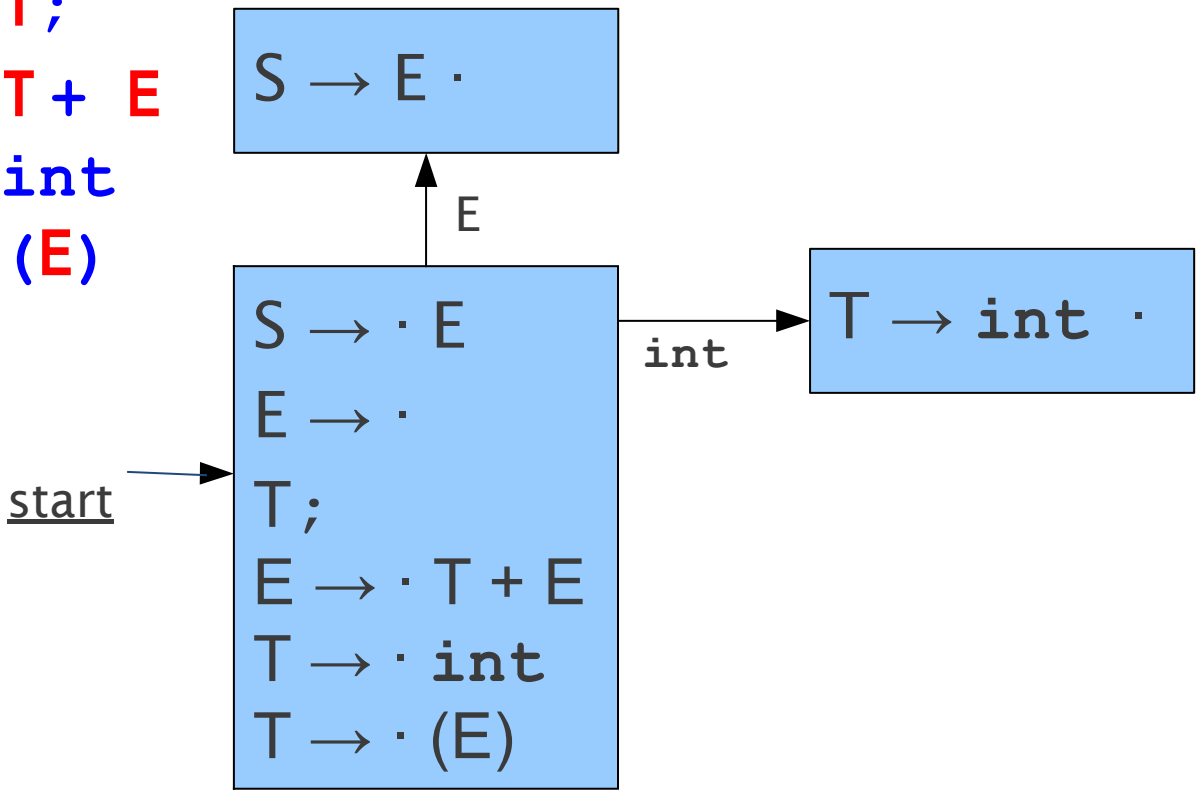
A Deterministic Automaton

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



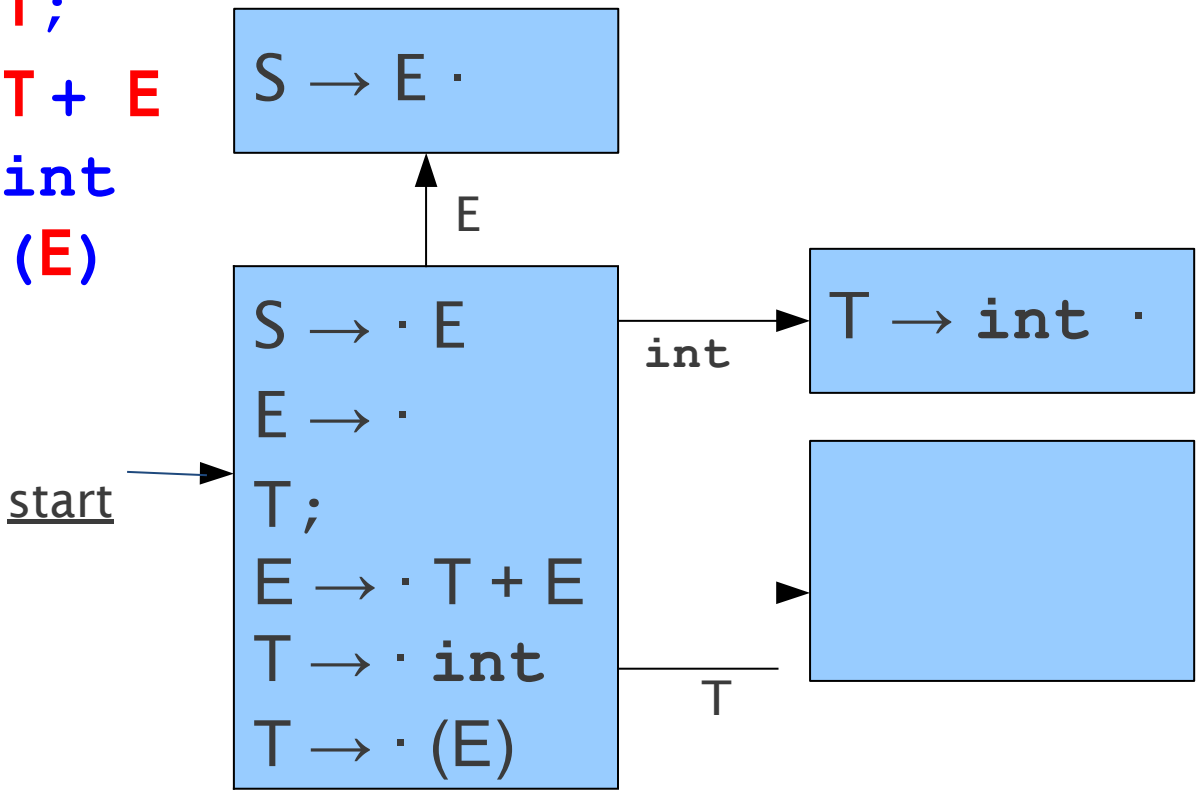
A Deterministic Automaton

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



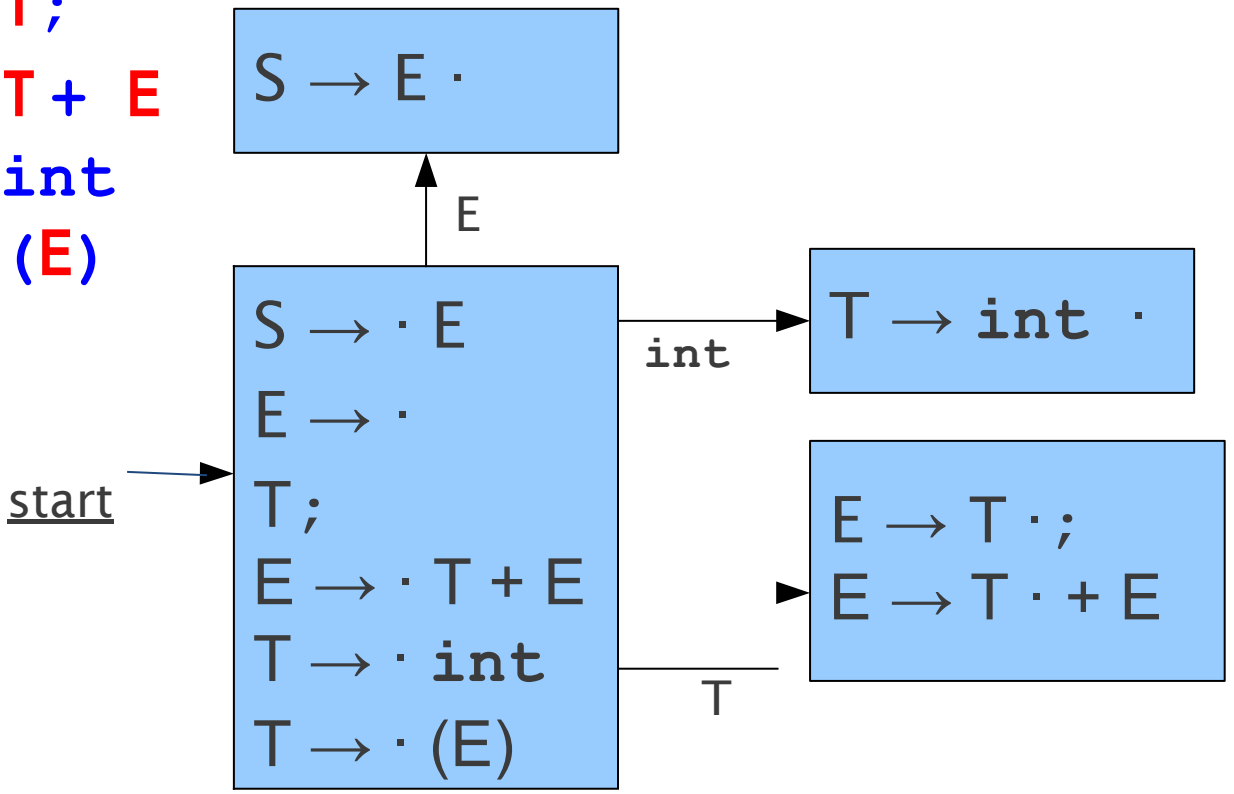
A Deterministic Automaton

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

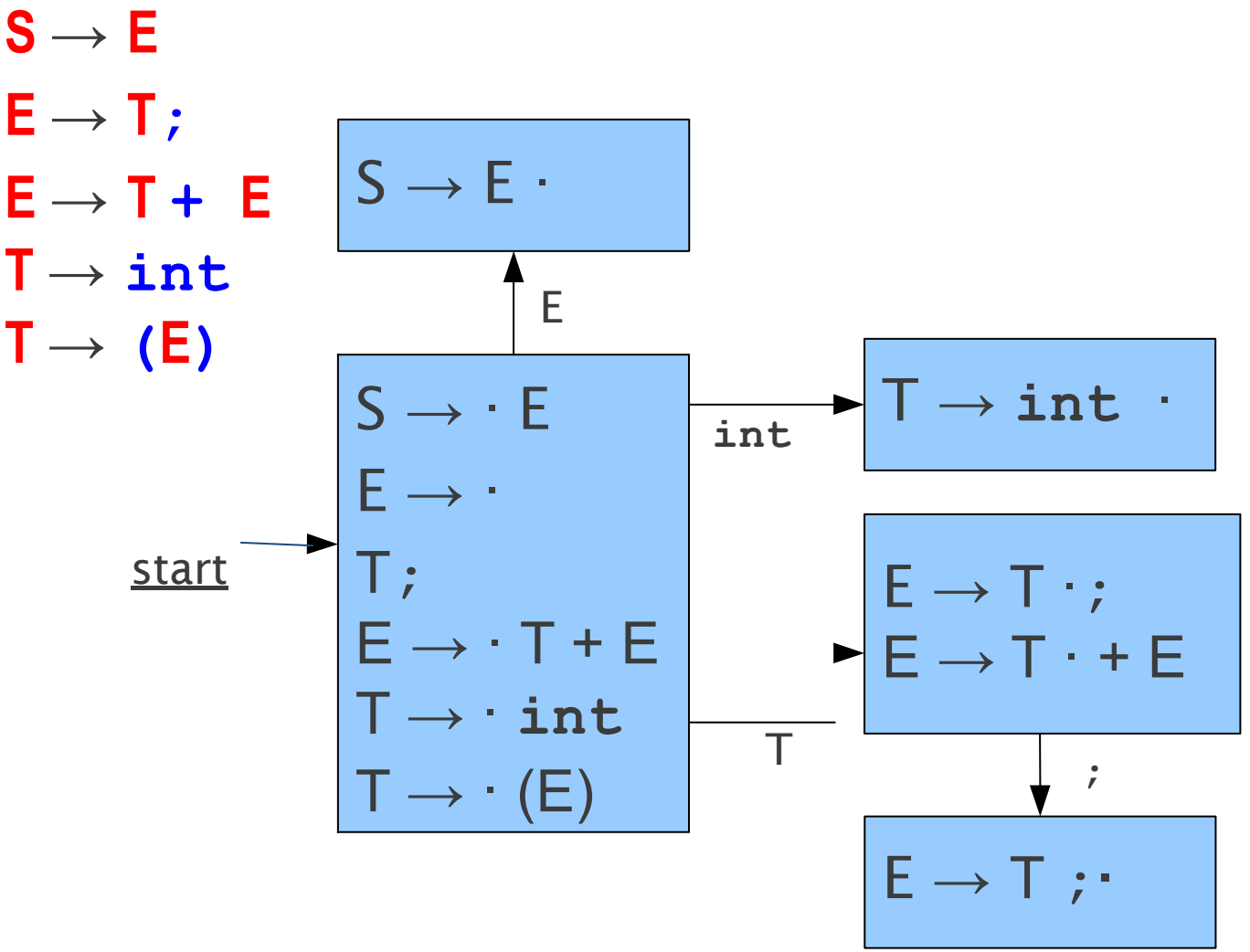


A Deterministic Automaton

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



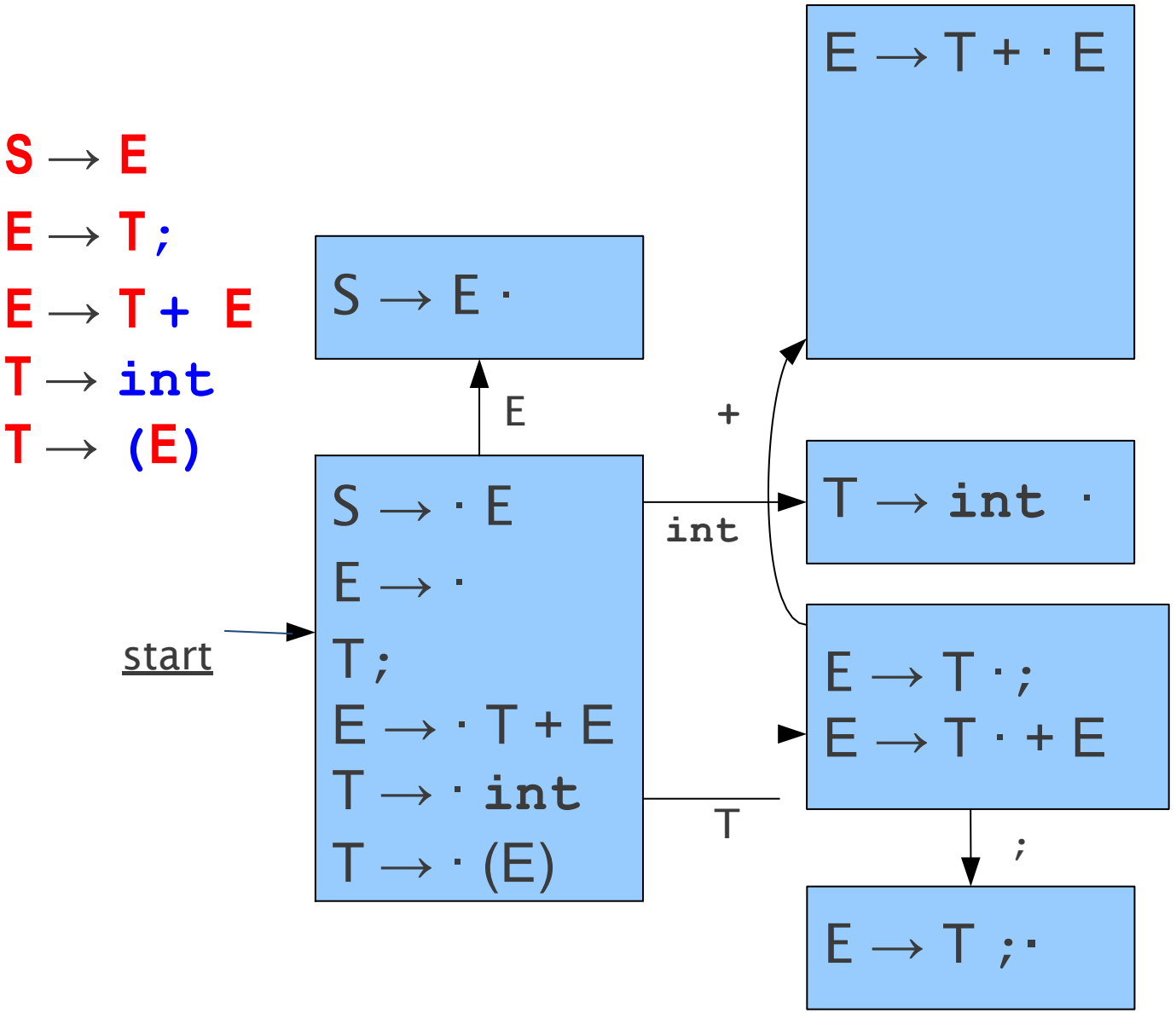
A Deterministic Automaton



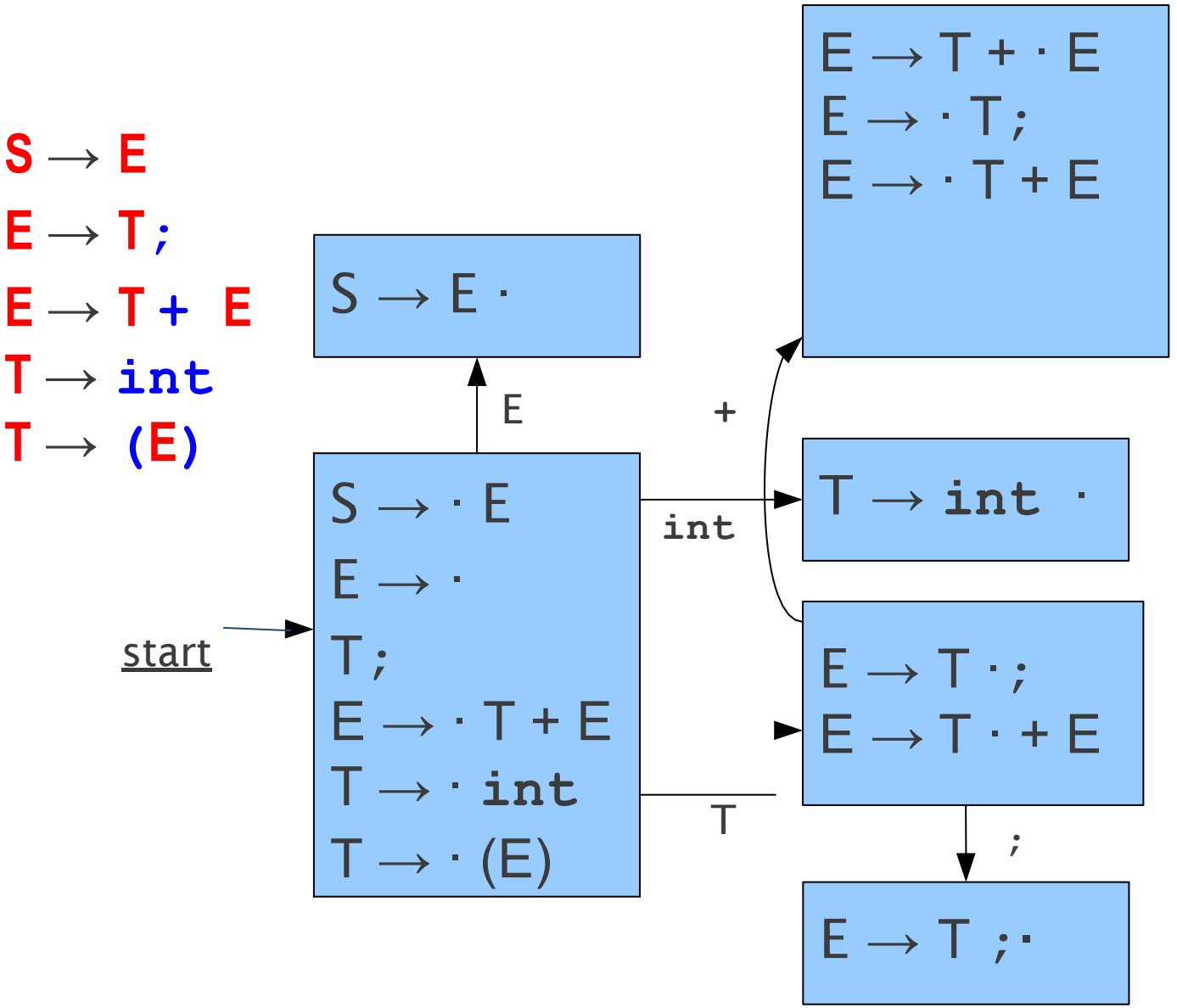
Constructing the Automaton

- Repeat until no new states are added:
  - If a state contains a production $A \rightarrow a \cdot x\omega$ for symbol x, add a transition on x from that state to the state containing the closure of $A \rightarrow ax \cdot \omega$

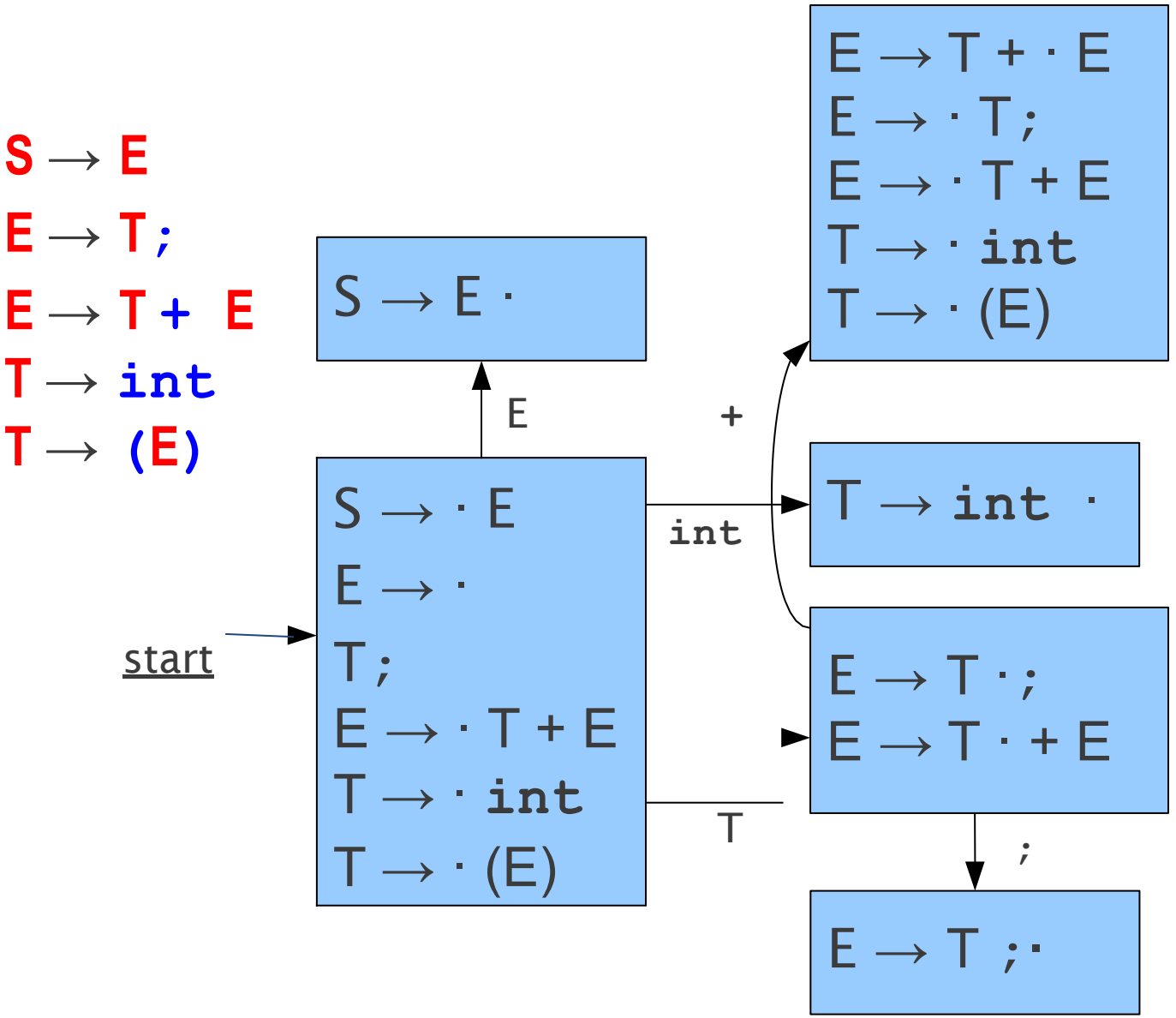
A Deterministic Automaton



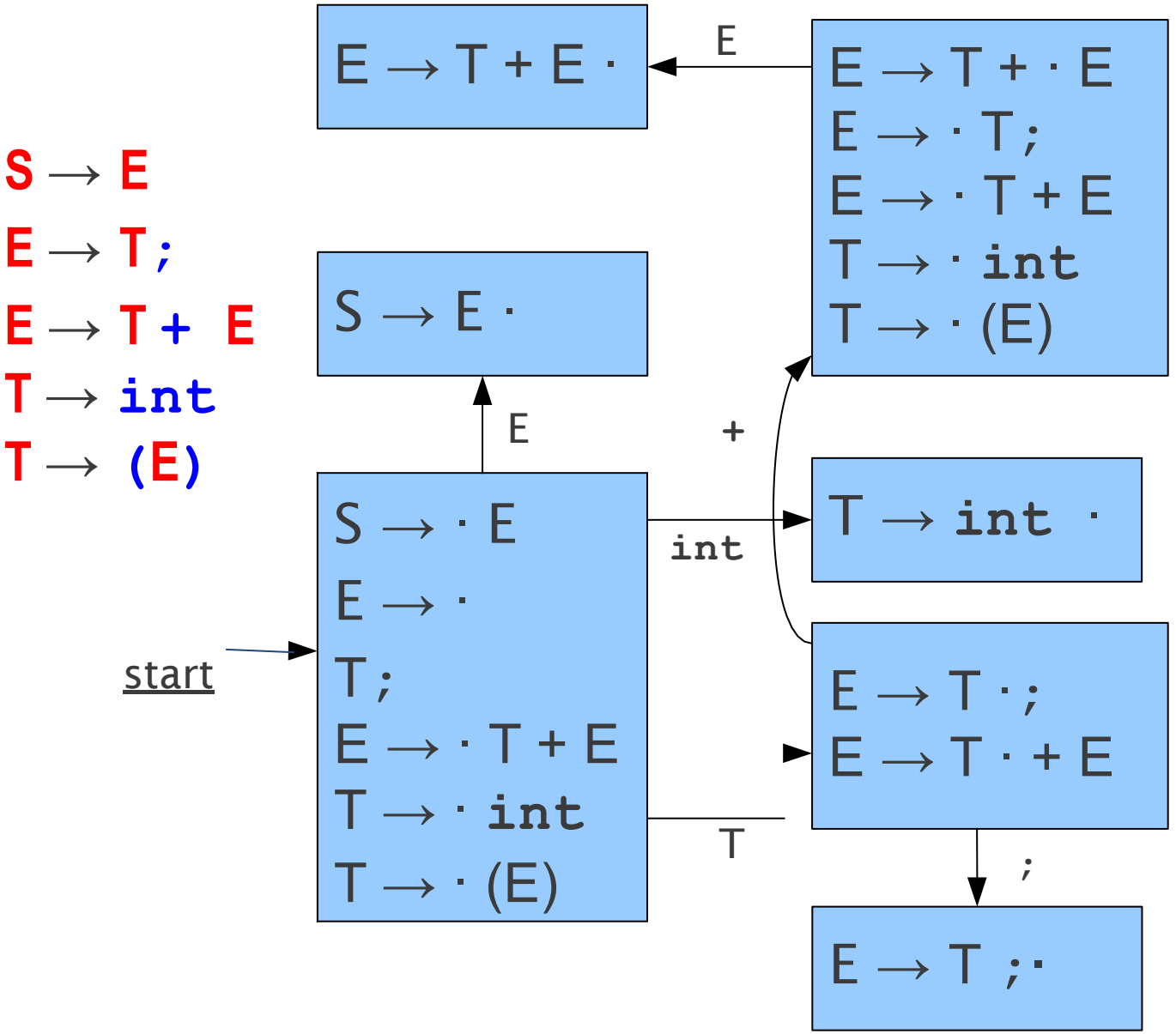
A Deterministic Automaton



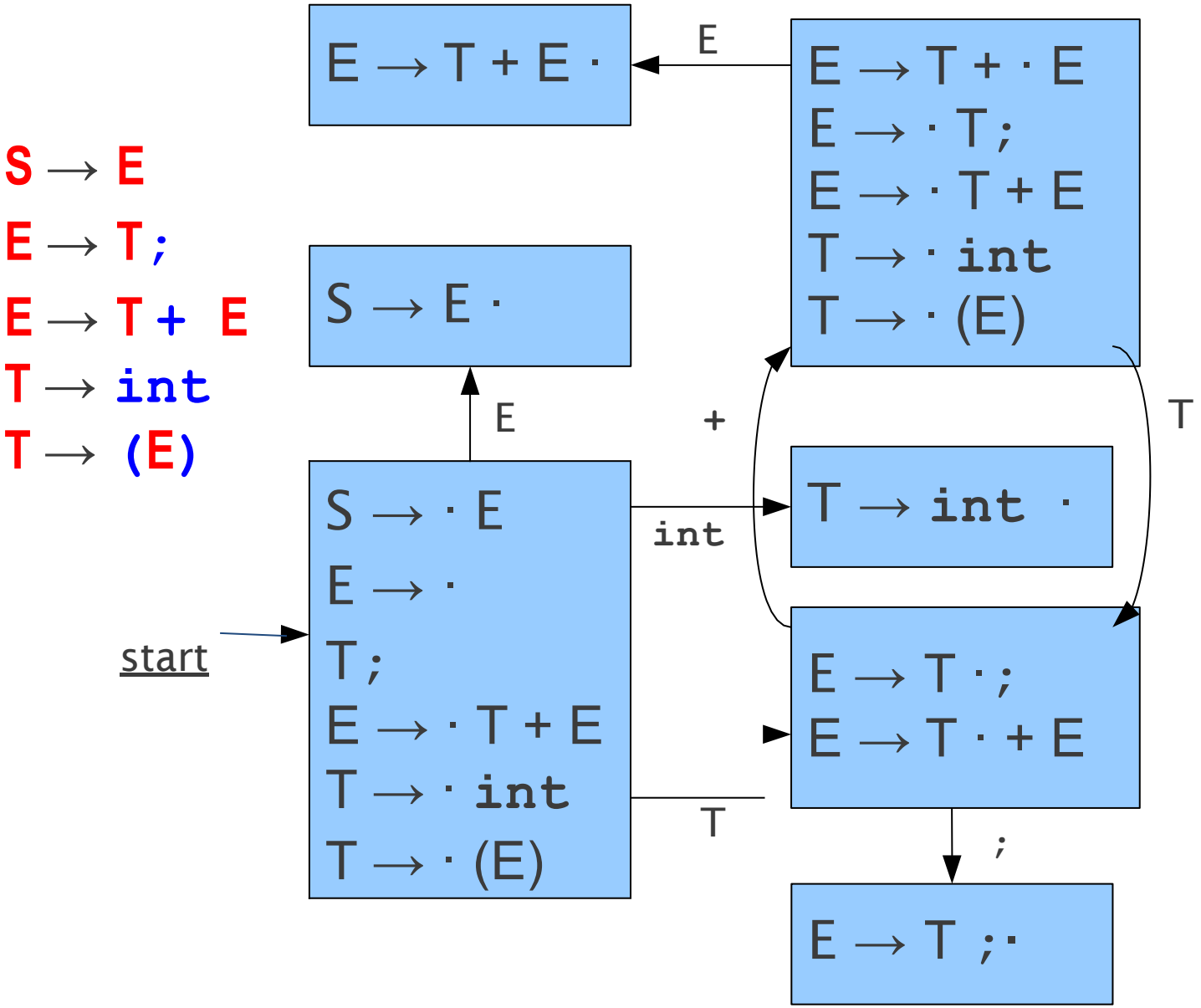
A Deterministic Automaton



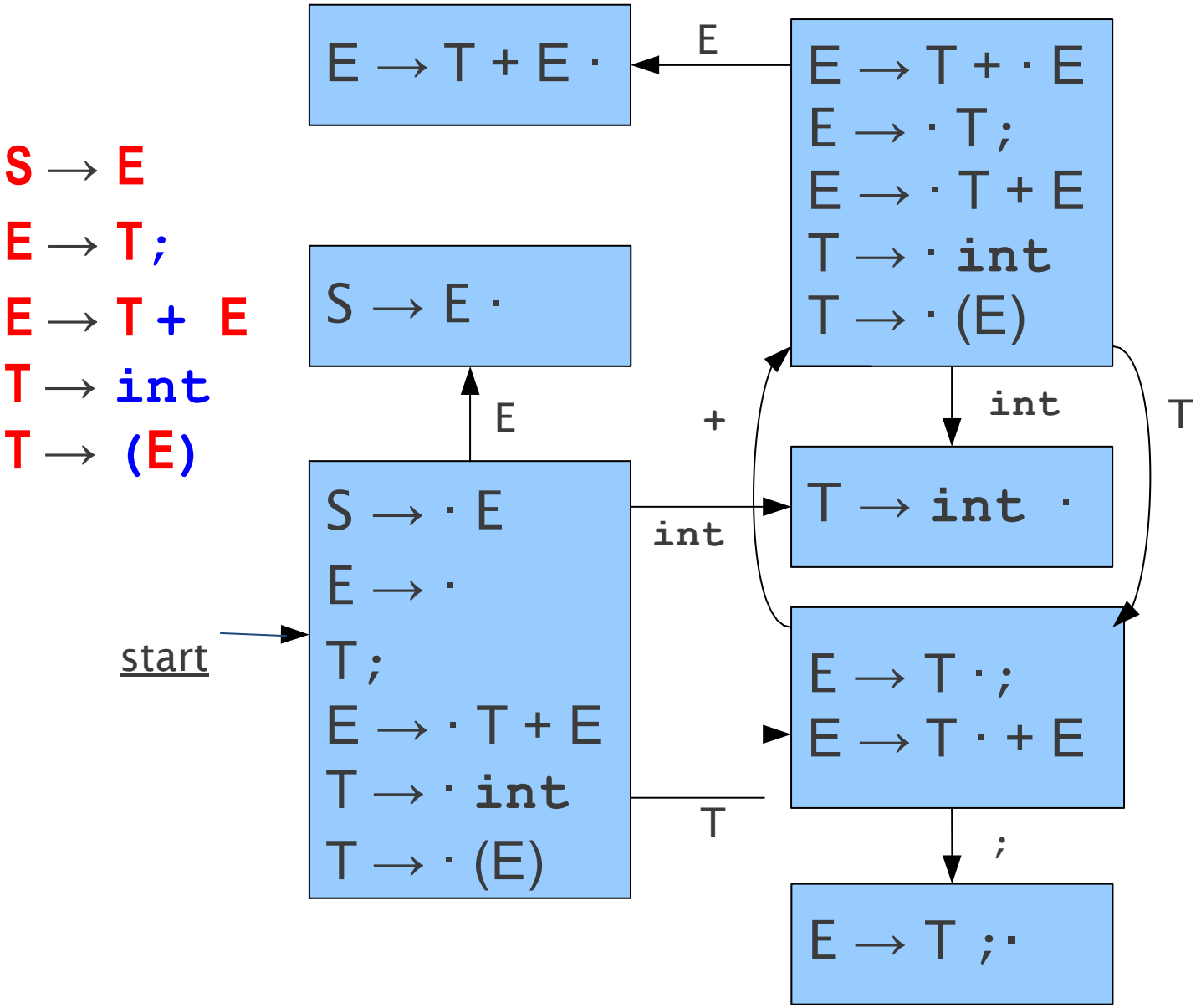
A Deterministic Automaton



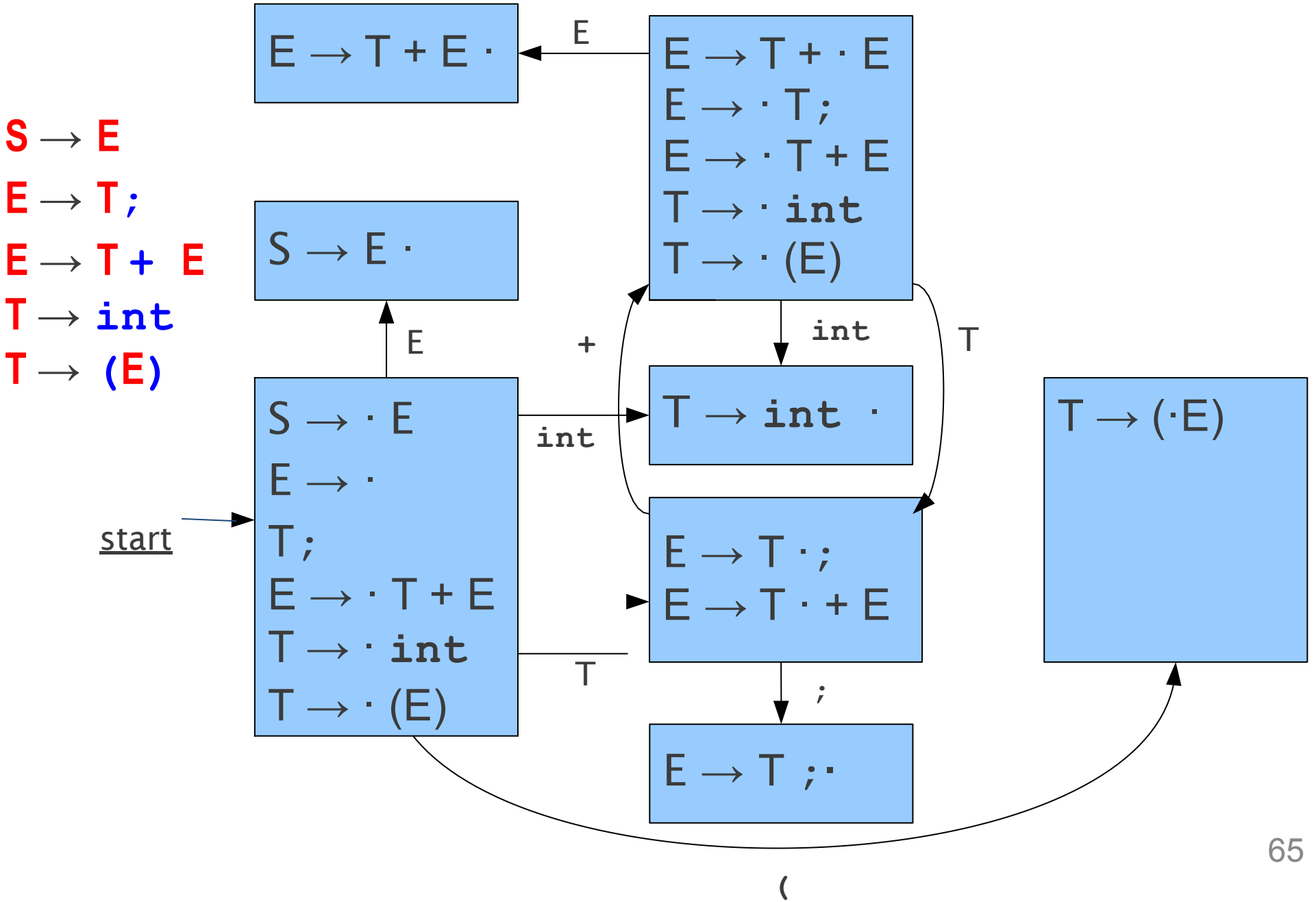
A Deterministic Automaton



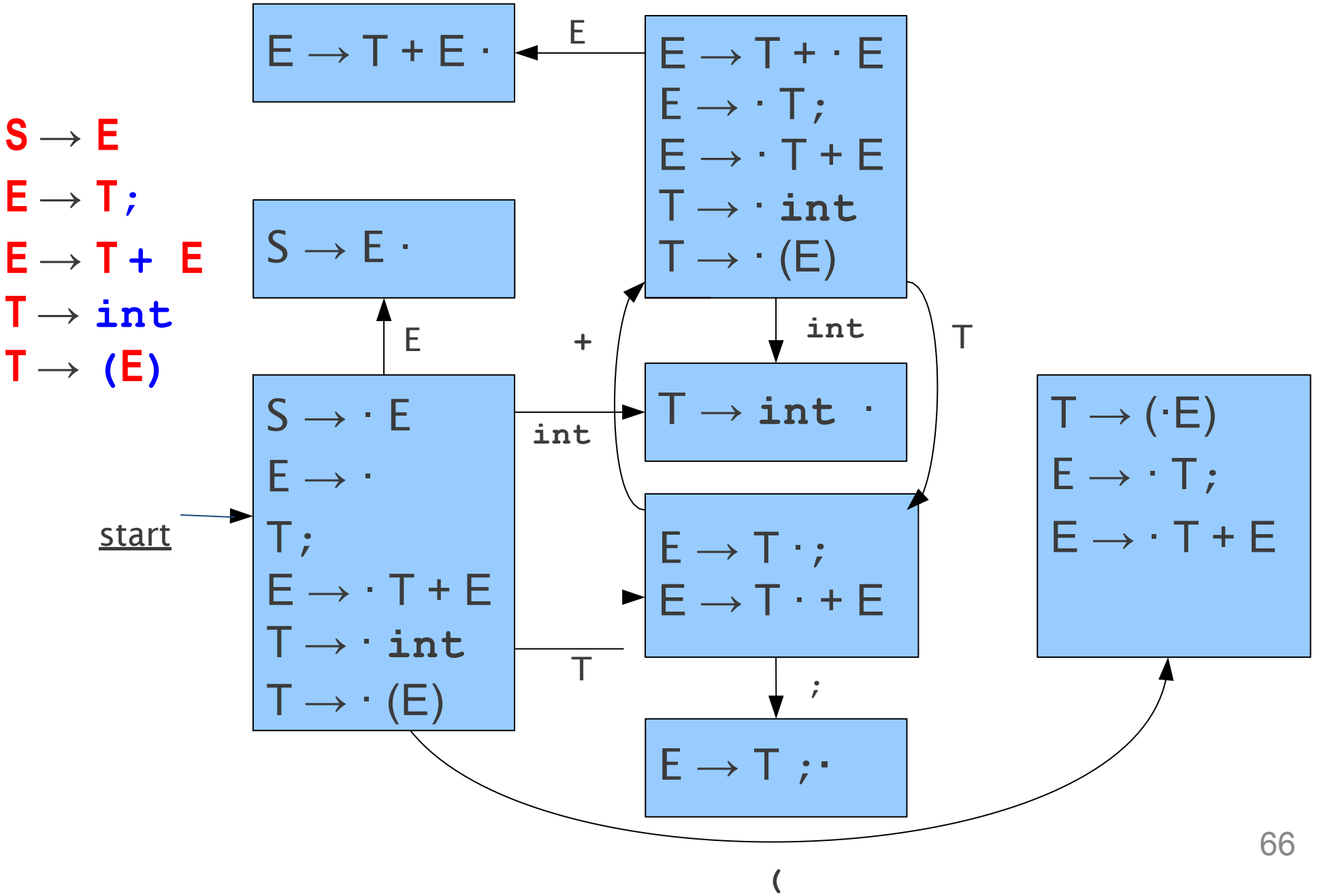
A Deterministic Automaton



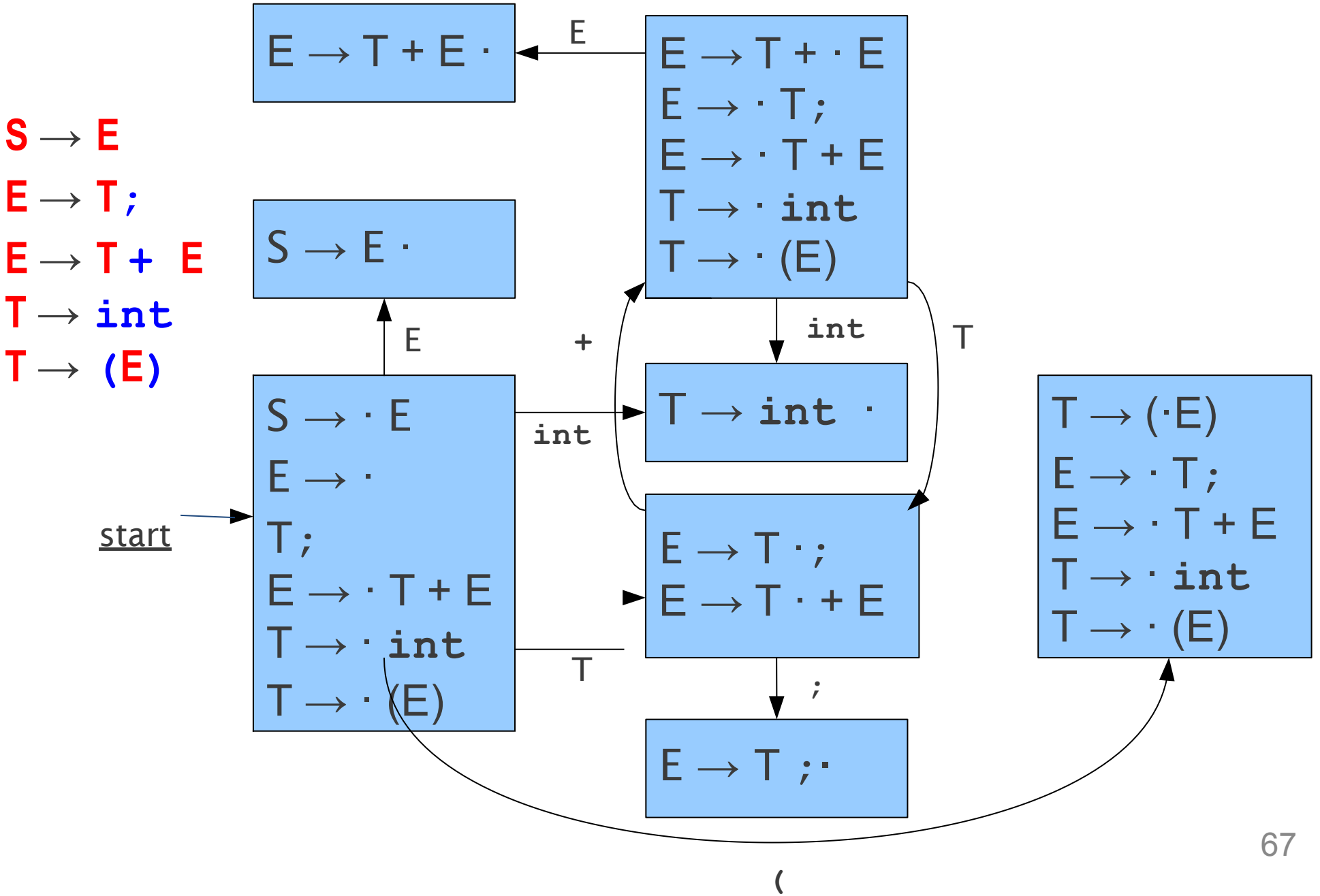
A Deterministic Automaton



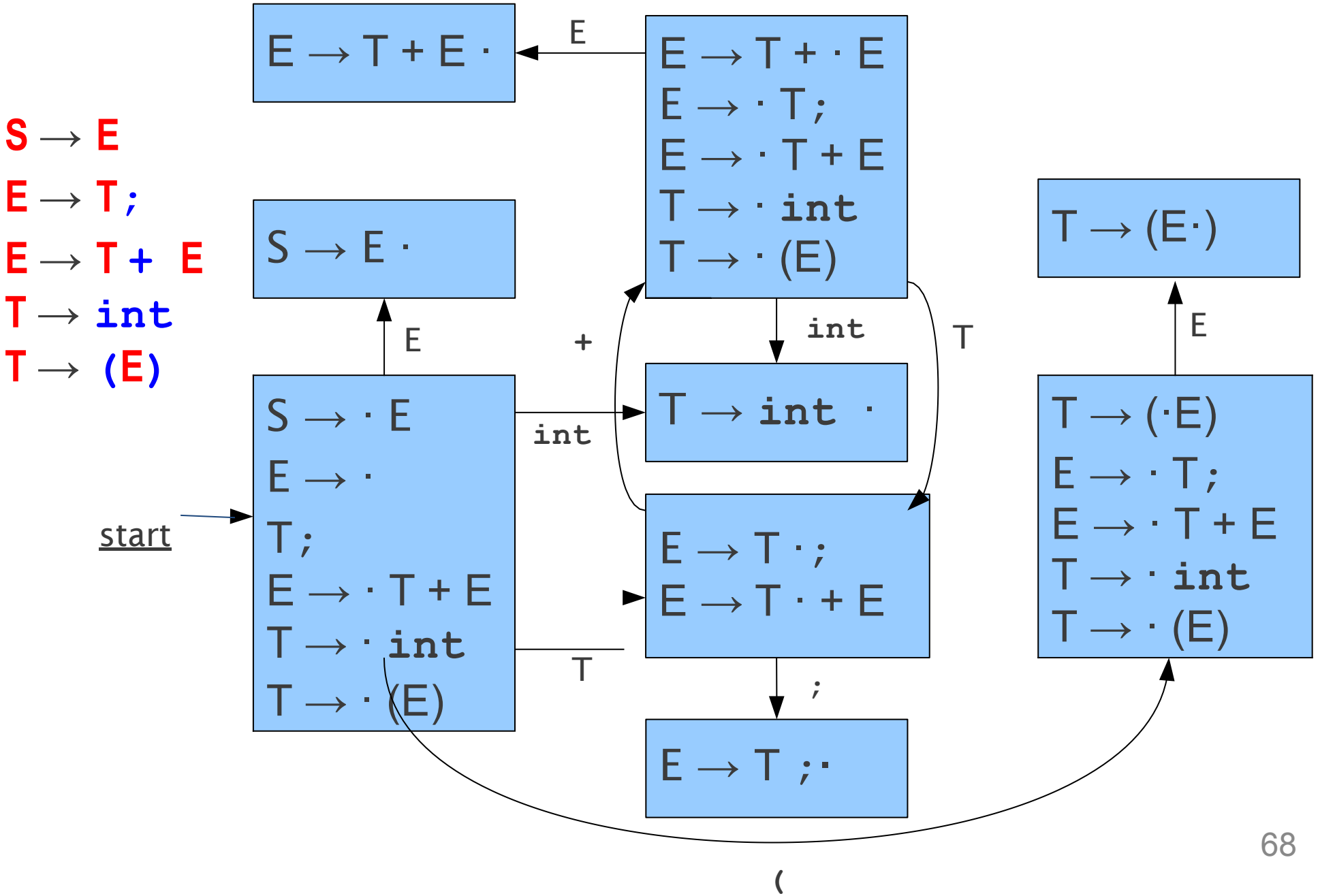
A Deterministic Automaton



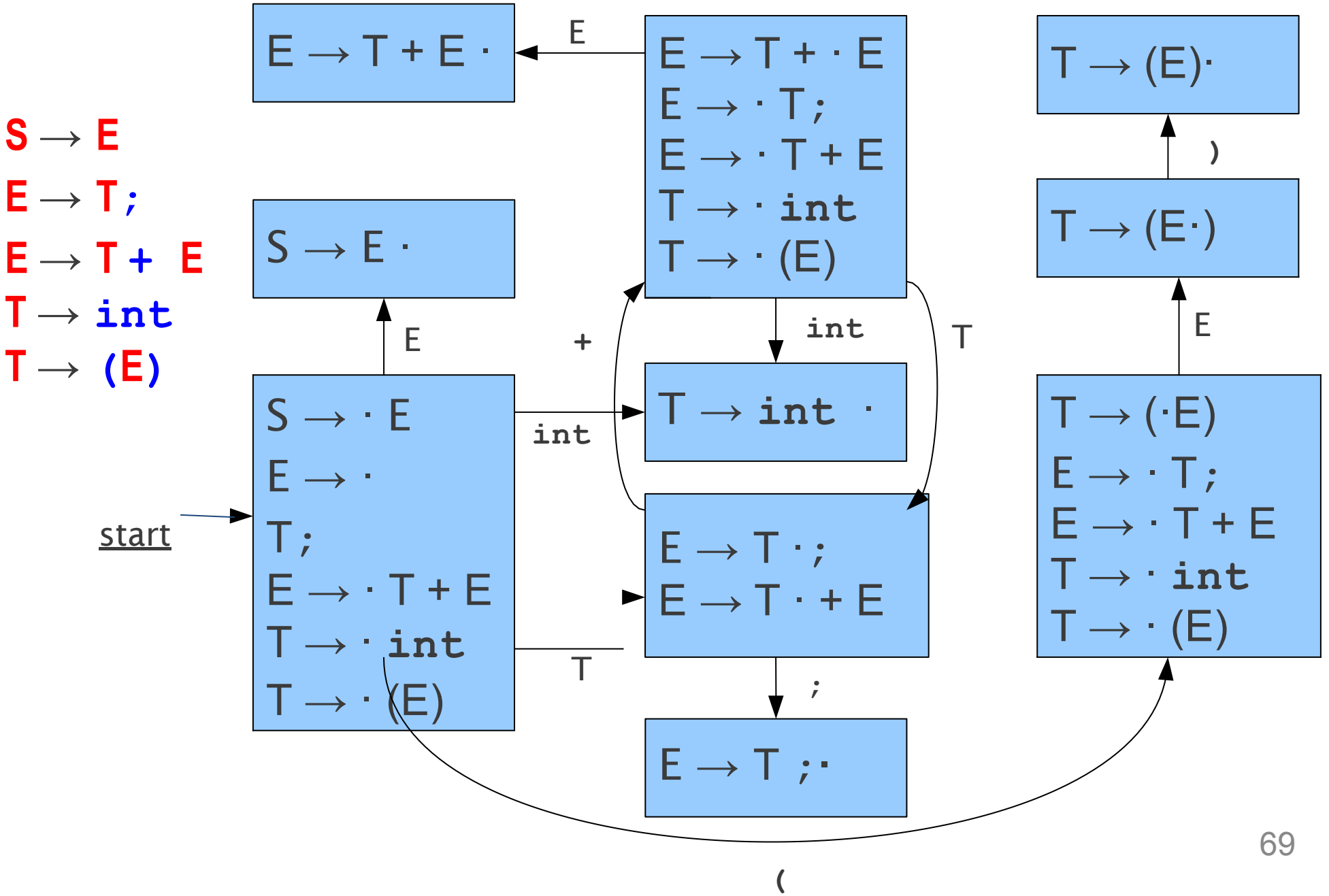
A Deterministic Automaton



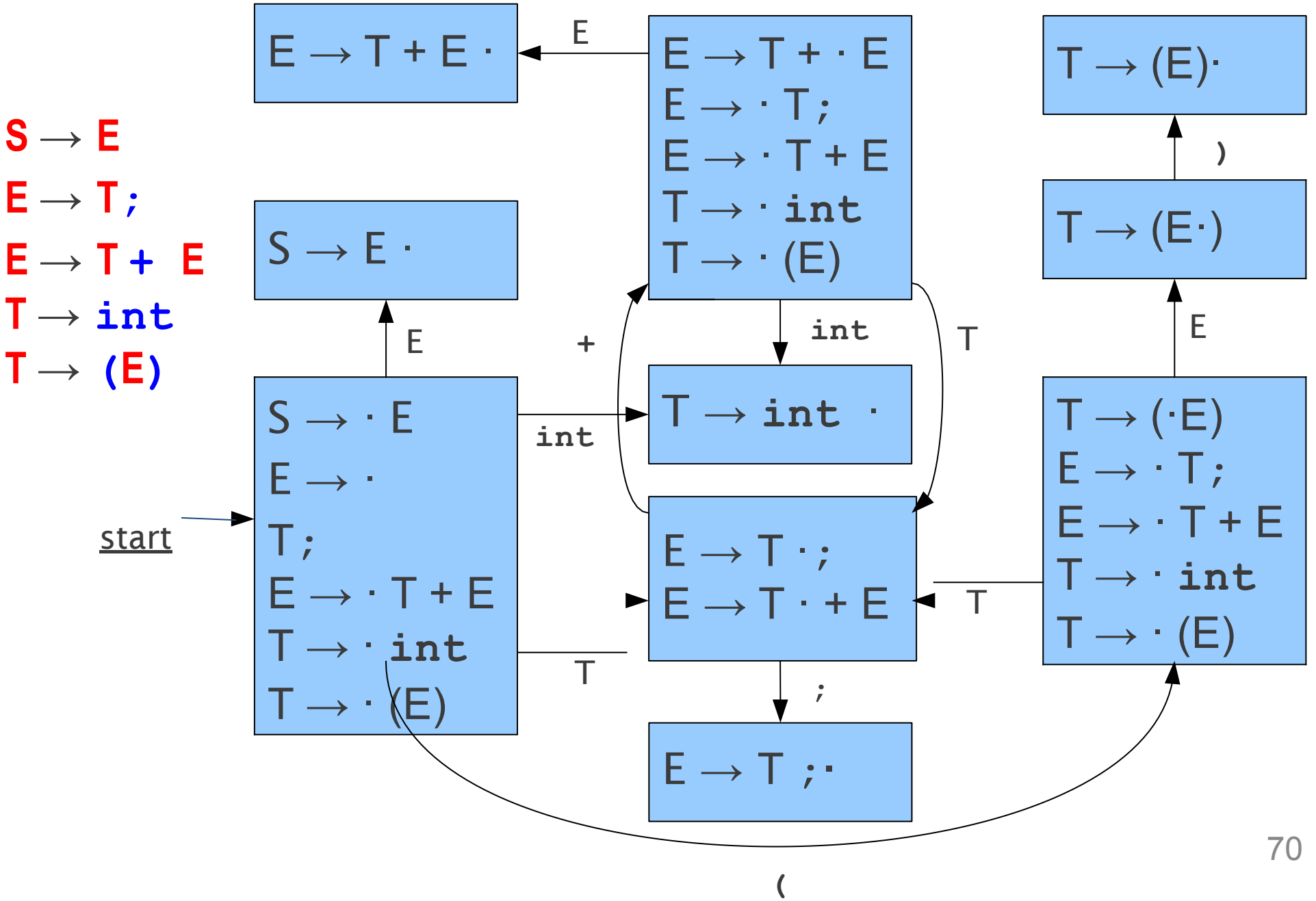
A Deterministic Automaton



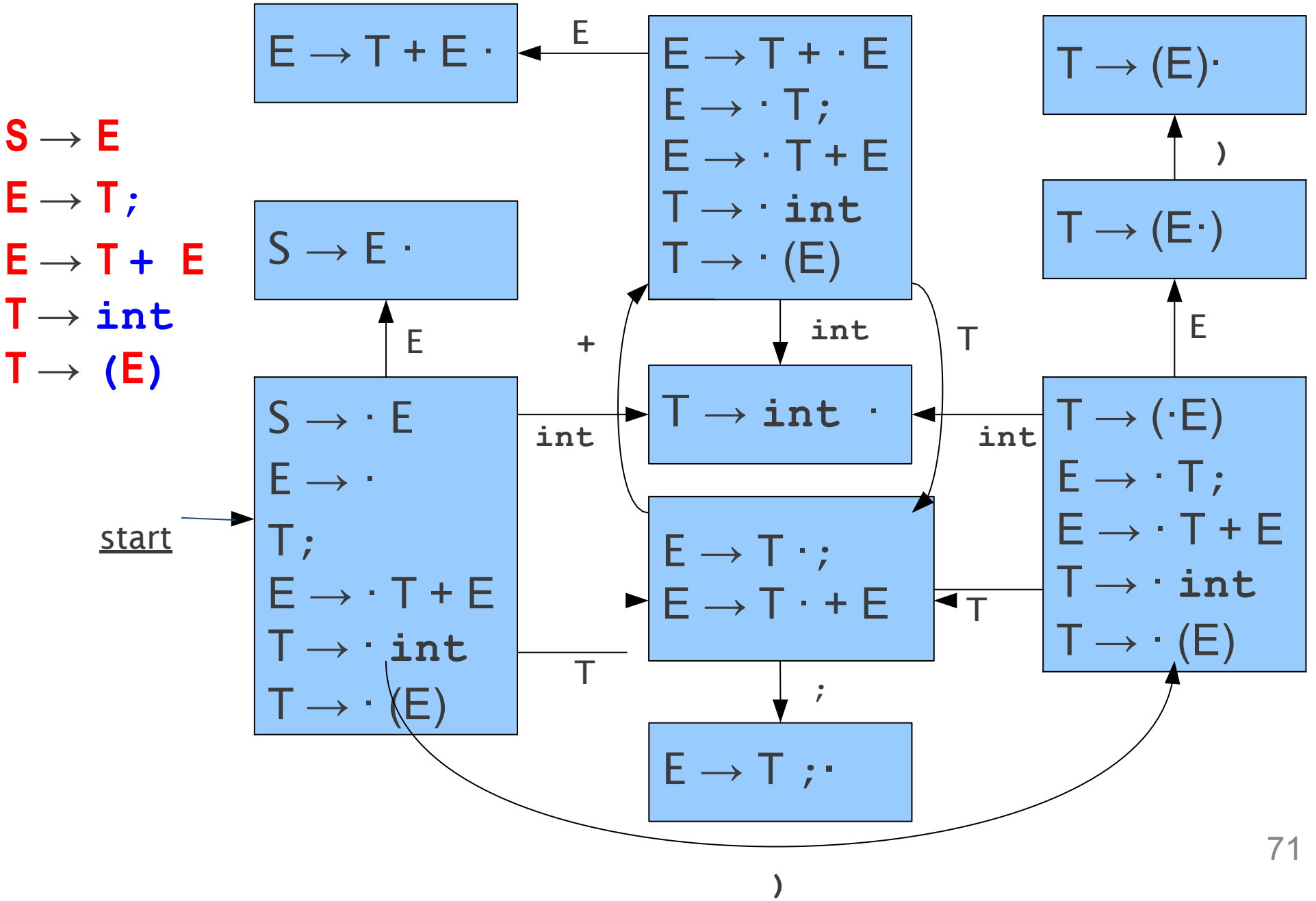
A Deterministic Automaton



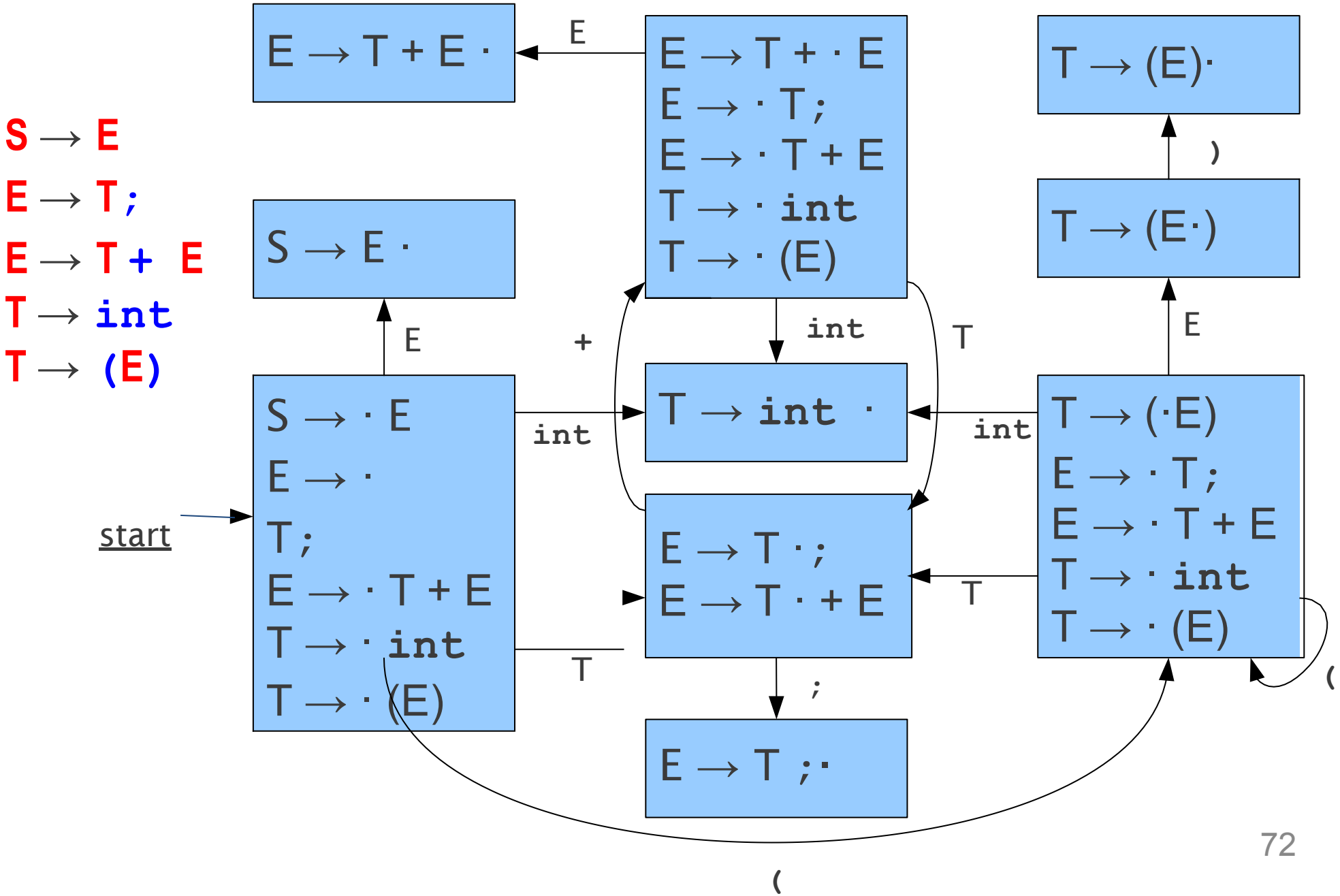
A Deterministic Automaton



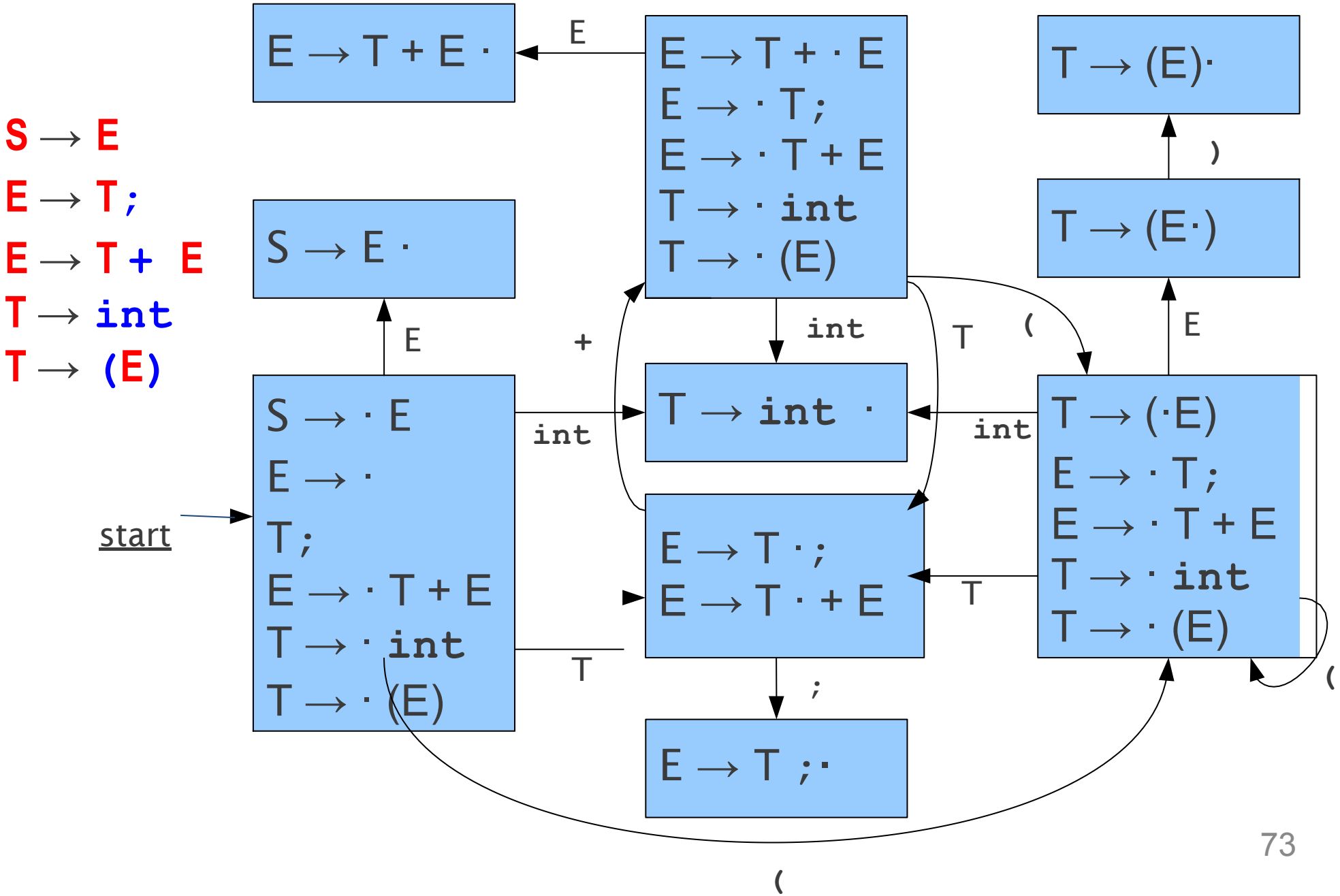
A Deterministic Automaton



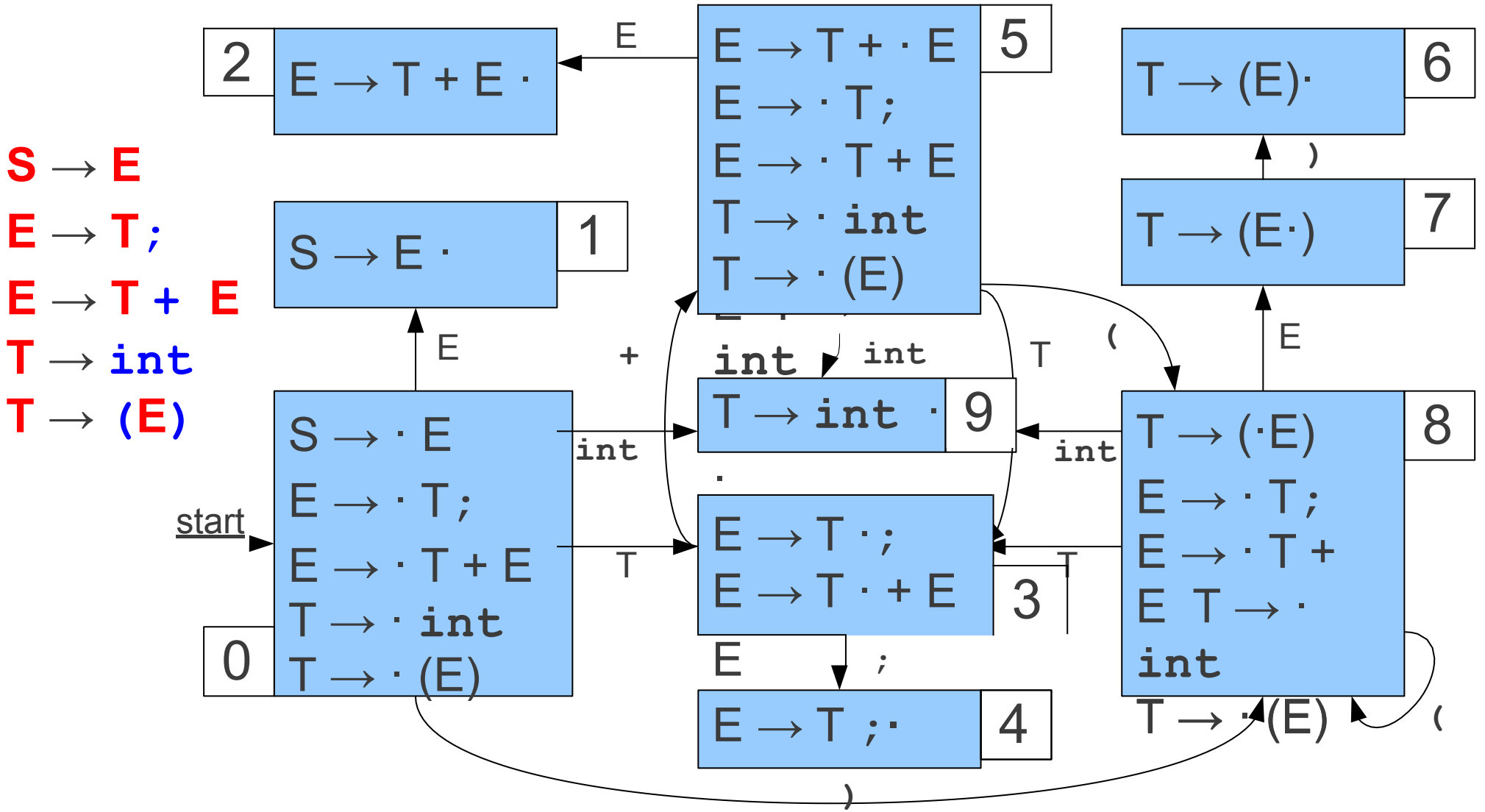
A Deterministic Automaton



A Deterministic Automaton



A Deterministic Automaton



A Deterministic Automaton

	`int`	`+`	`;`	`(`	`)`	E	T
0	S9			S8		S1	S3
1							
2							
3		S5	S4				
4							
5	S9			S8		S2	S3
6							
7					S6		
8	S9			S8		S7	S3
9							

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

$$A \rightarrow \omega \cdot$$

- Then we might be looking at a handle.
- This automaton can be used to discover possible handle locations!

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?

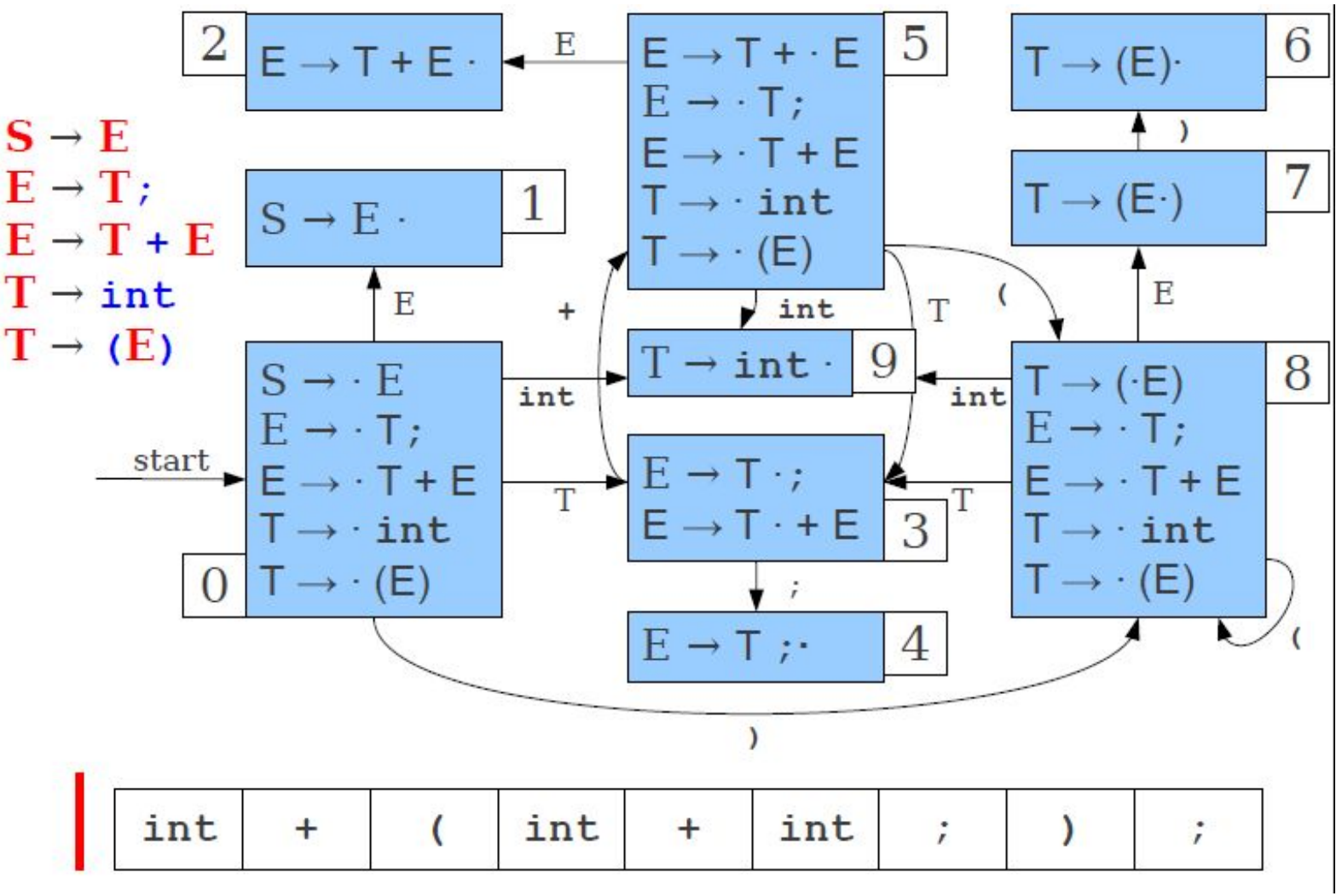
Handle Recognition

- 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.

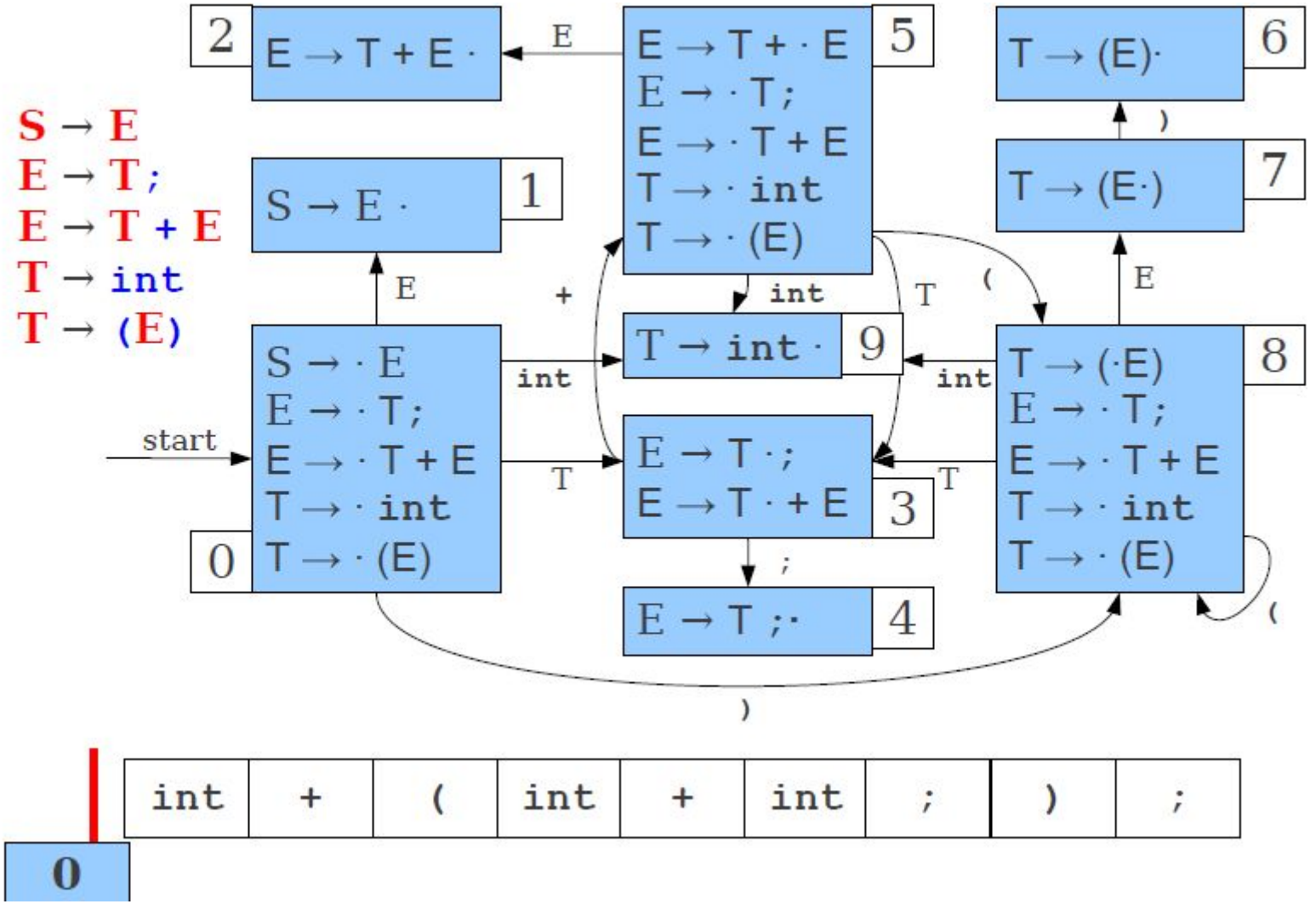
Our First Algorithm: LR(0)

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

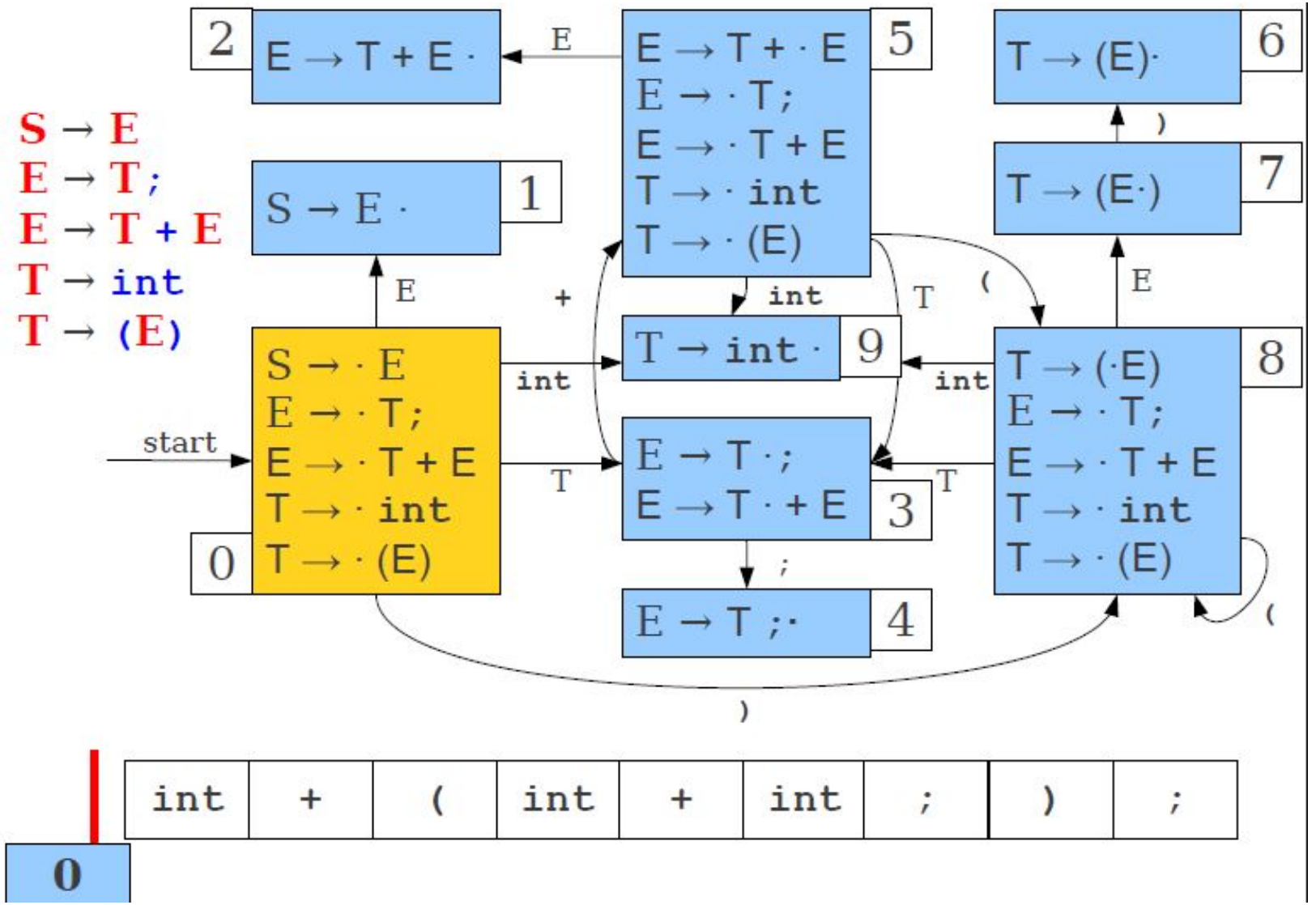
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
6
T → (E)·
)
1
S → E ·
7
T → (E·)
E
+
int
T
(
E
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
int
T → int ·
9
int
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
start
T
E → T ·;
E → T · + E
3
T
0
;
E → T ;·
4
)
(
int + (int + int ;) ;

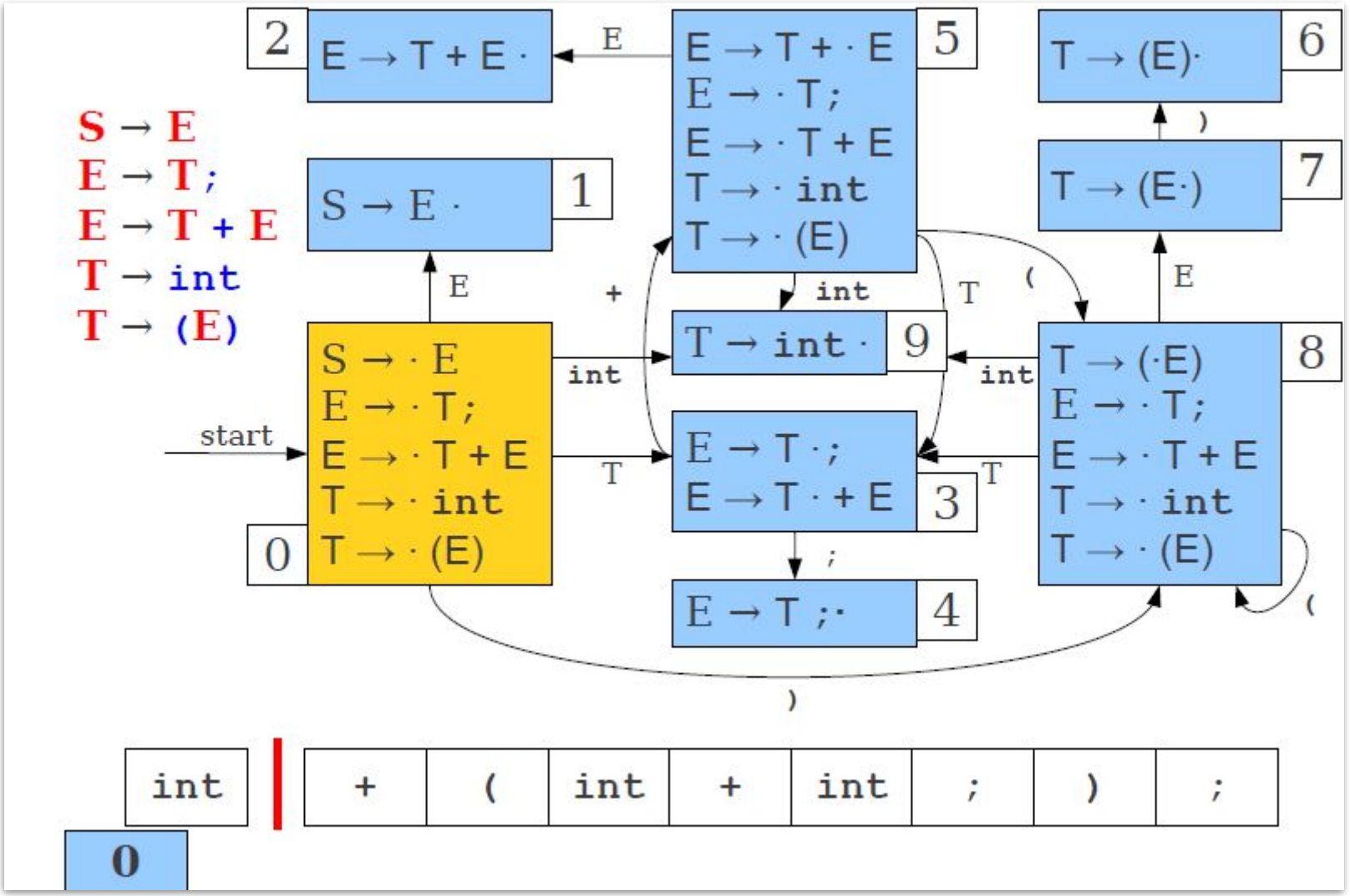
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
S → E ·
1
E → T + E ·
2
E → T · ;
E → T · + E
3
E → T ; ·
4
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
T → (E·)
7
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
T → int ·
9
E
T
int
+
(
)
;
int | + | (| int | + | int | ; |) | ;
0

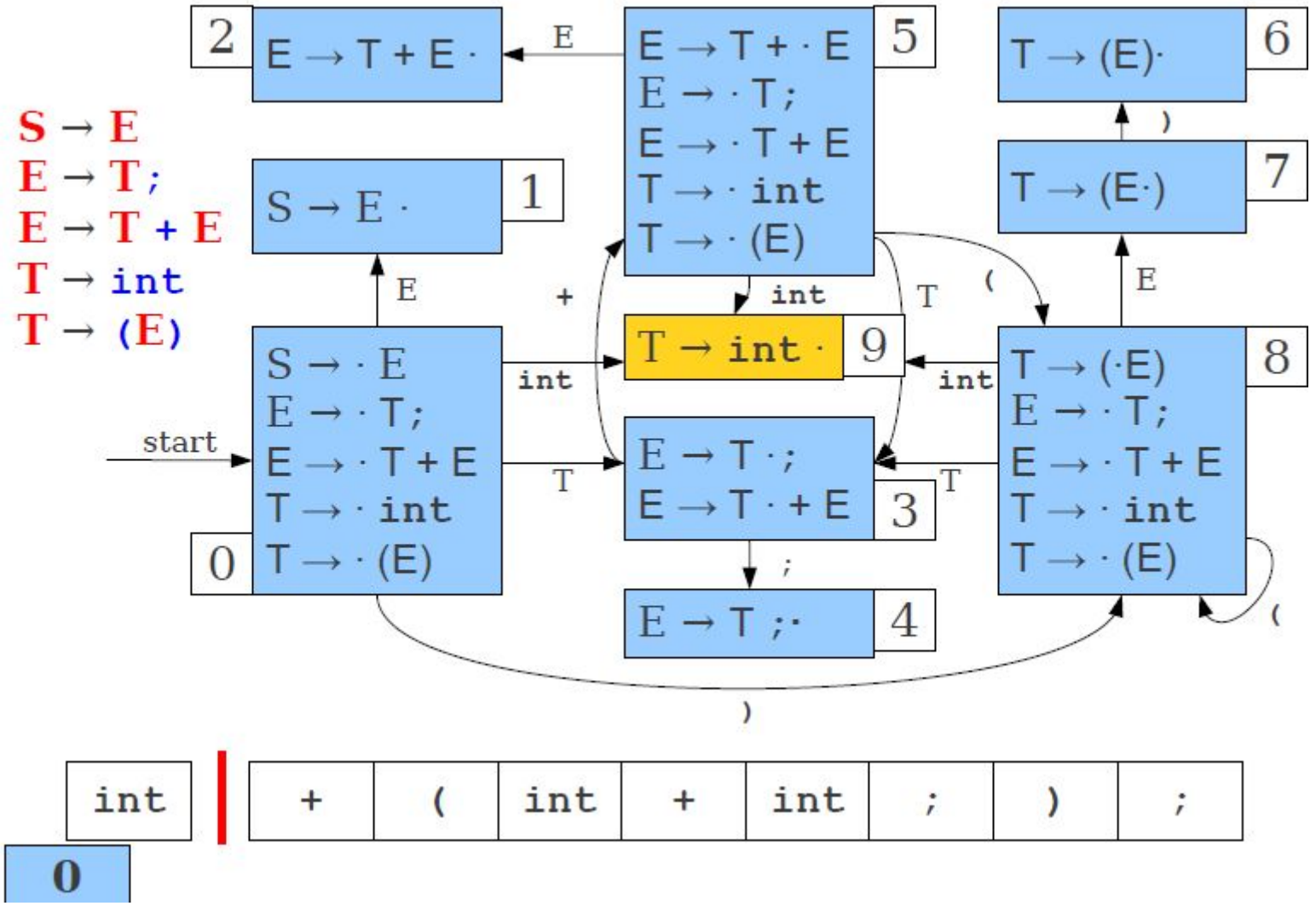
LR(0) Parsing


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

LR(0) Parsing

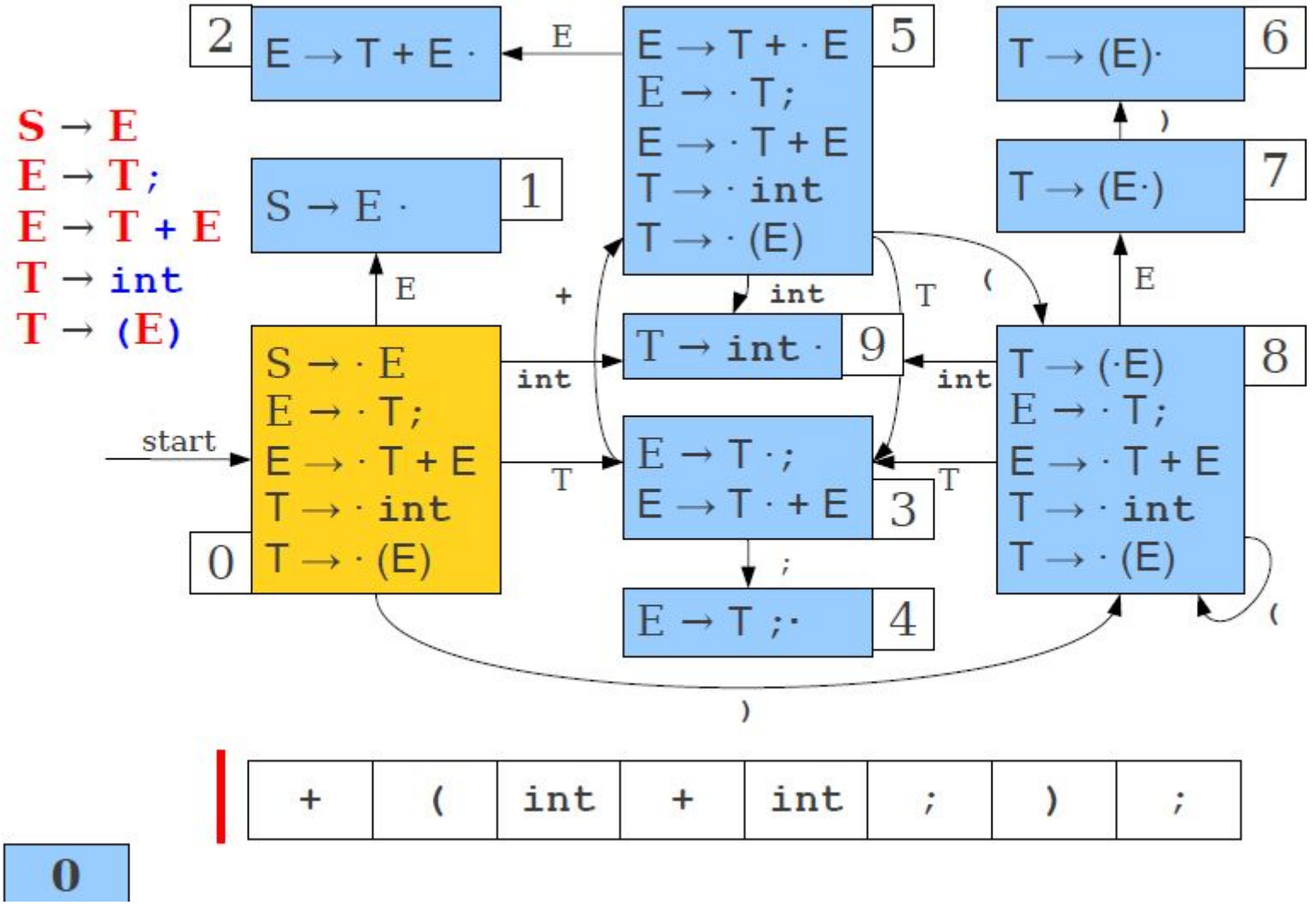

S → E
E → T;
E → T + E
T → int
T → (E)
0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
1
S → E ·
2
E → T + E ·
3
E → T · ;
E → T · + E
4
E → T ; ·
5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
6
T → (E)·
7
T → (E·)
8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
9
T → int ·
start
E
T
int
+
(
)
;
int
+
(
int
+
int
;
)
;
0

LR(0) Tables

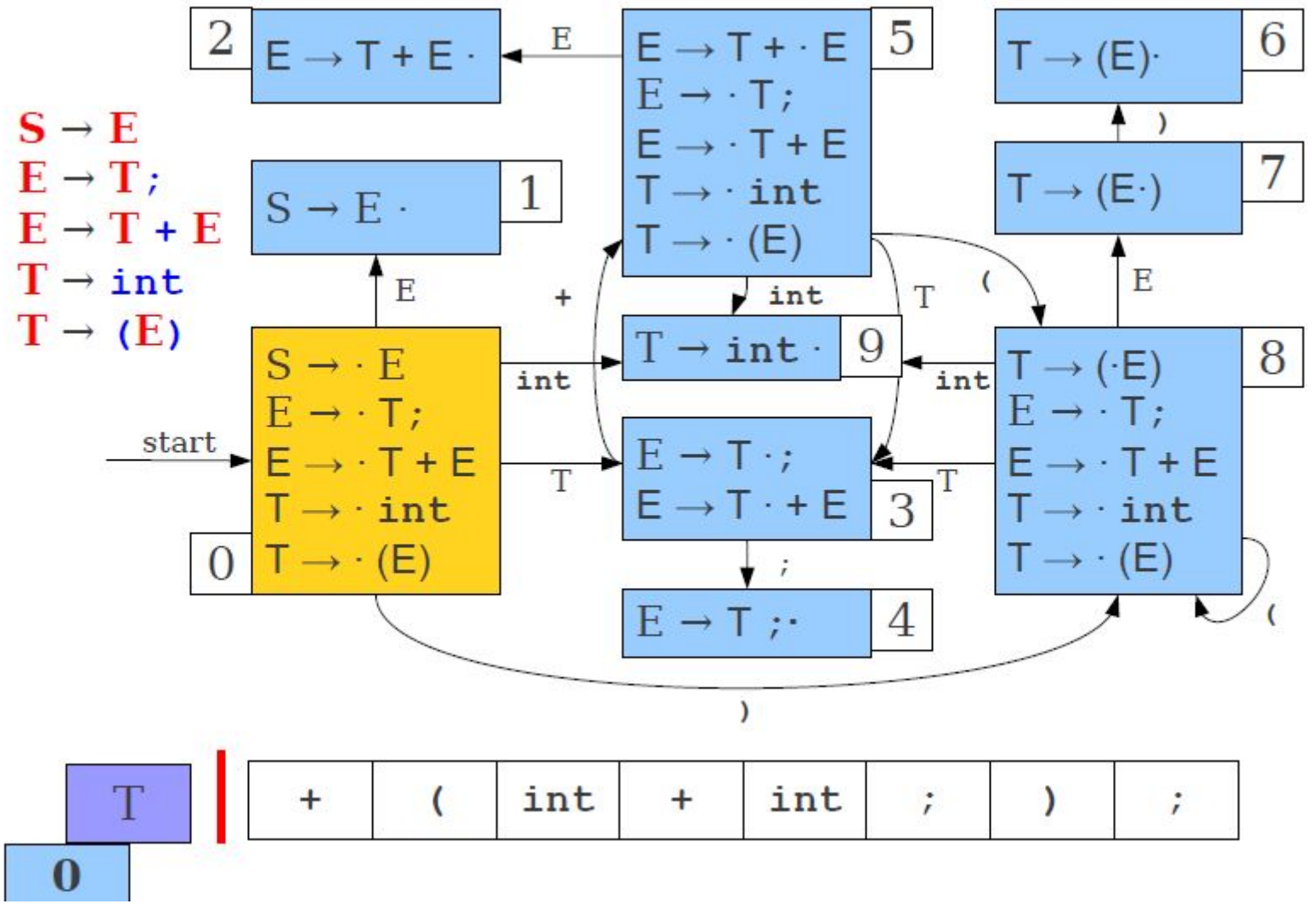
(1) S → E
(2) E → T;
(3) E → T + E
(4) T → int
(5) T → (E)

	int	+	;	(	)	E	T
0	S9			S8		S1	S3
1							
2	r3	r3	r3	r3	r3		
3		S5	S4				
4							
5	S9			S8		S2	S3
6							
7					s6		
8	S9			S8		S7	S3
9	r4	r4	r4	r4	r4		

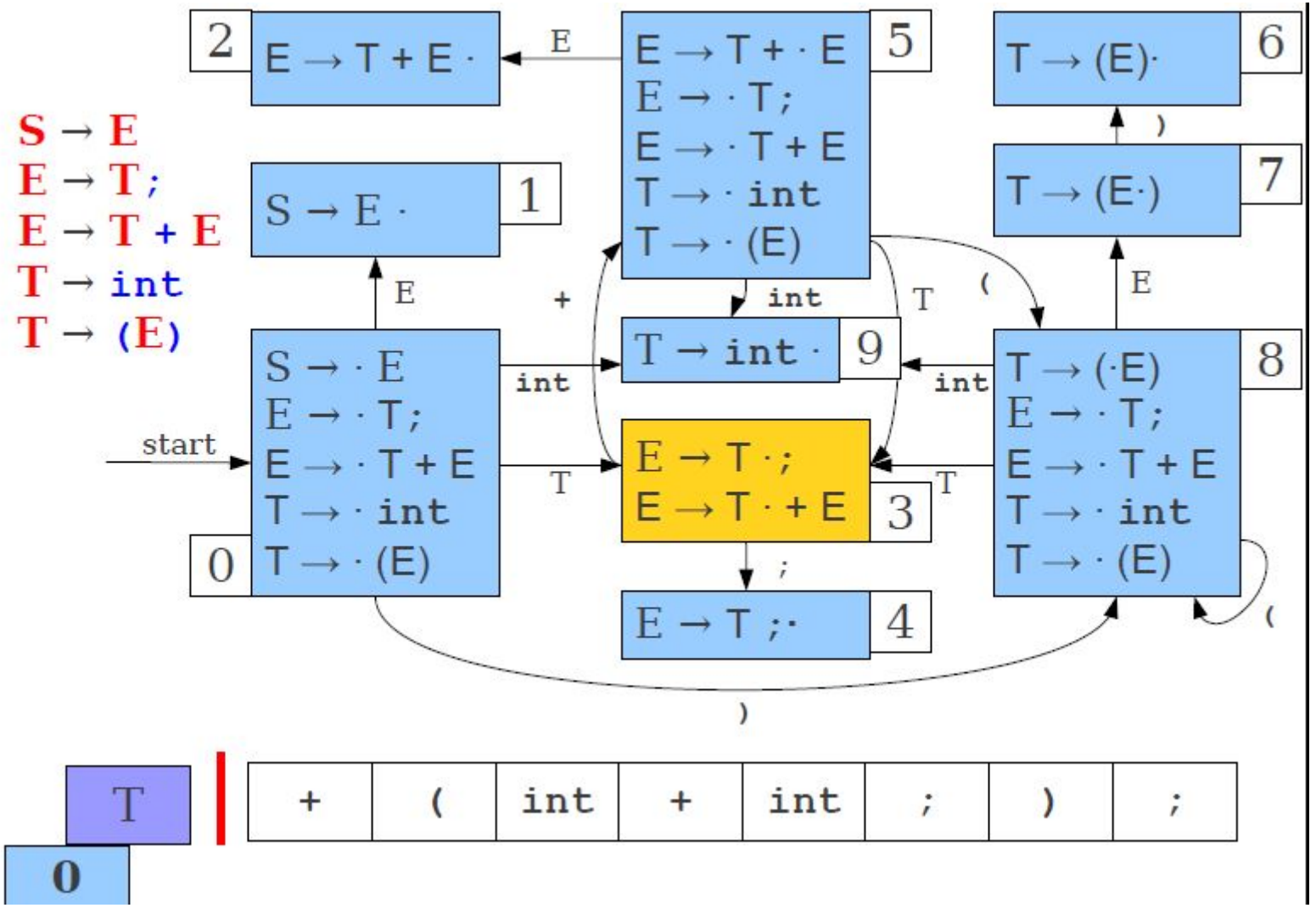
LR(0) Parsing


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

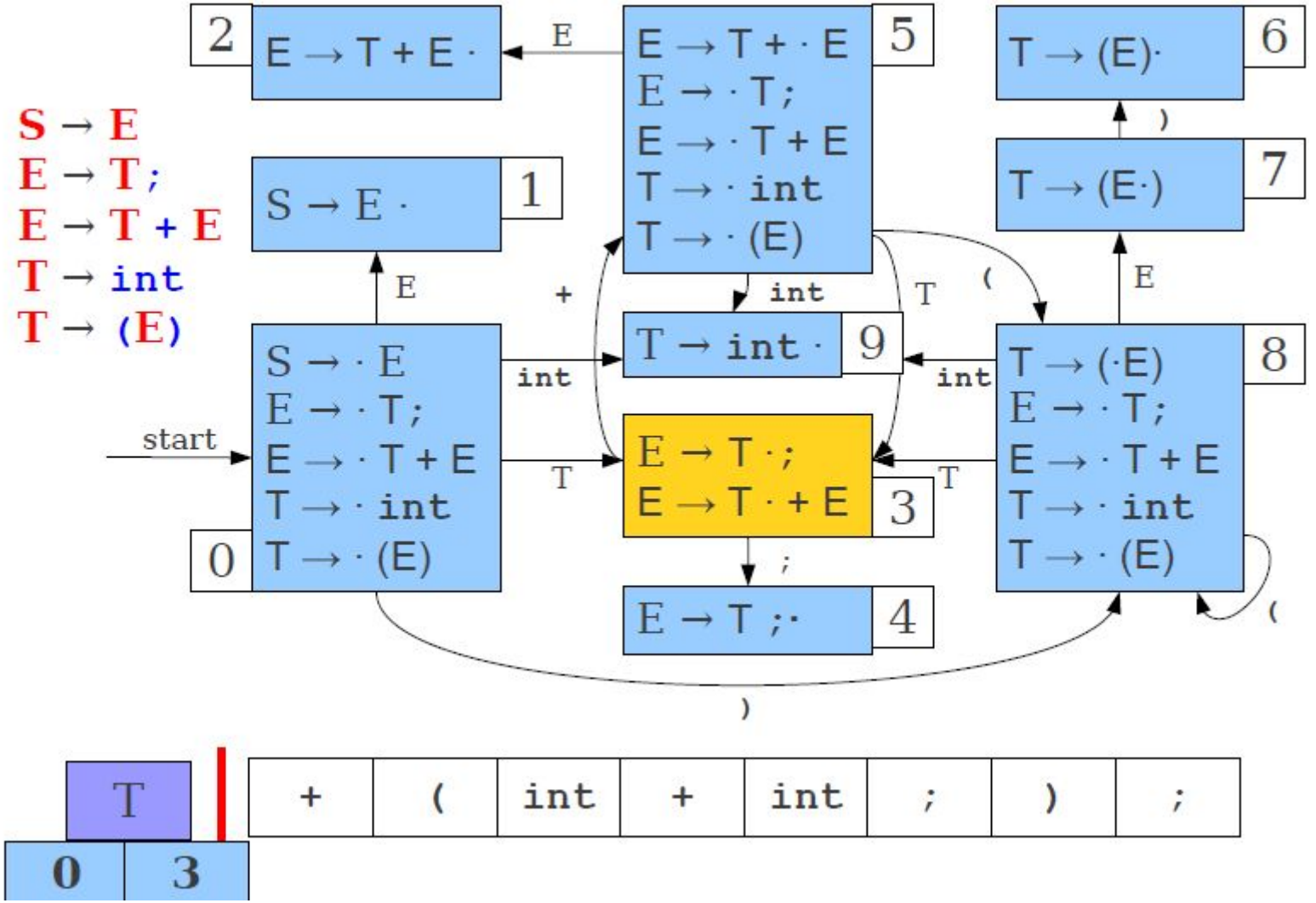
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
6
T → (E)·
)
7
T → (E·)
1
S → E ·
E
+
int
T
(
E
9
T → int ·
int
int
8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
T
T
E → T ·;
E → T · + E
3
T
;
E → T ;·
4
)
(
T
0
+
(
int
+
int
;
)
;

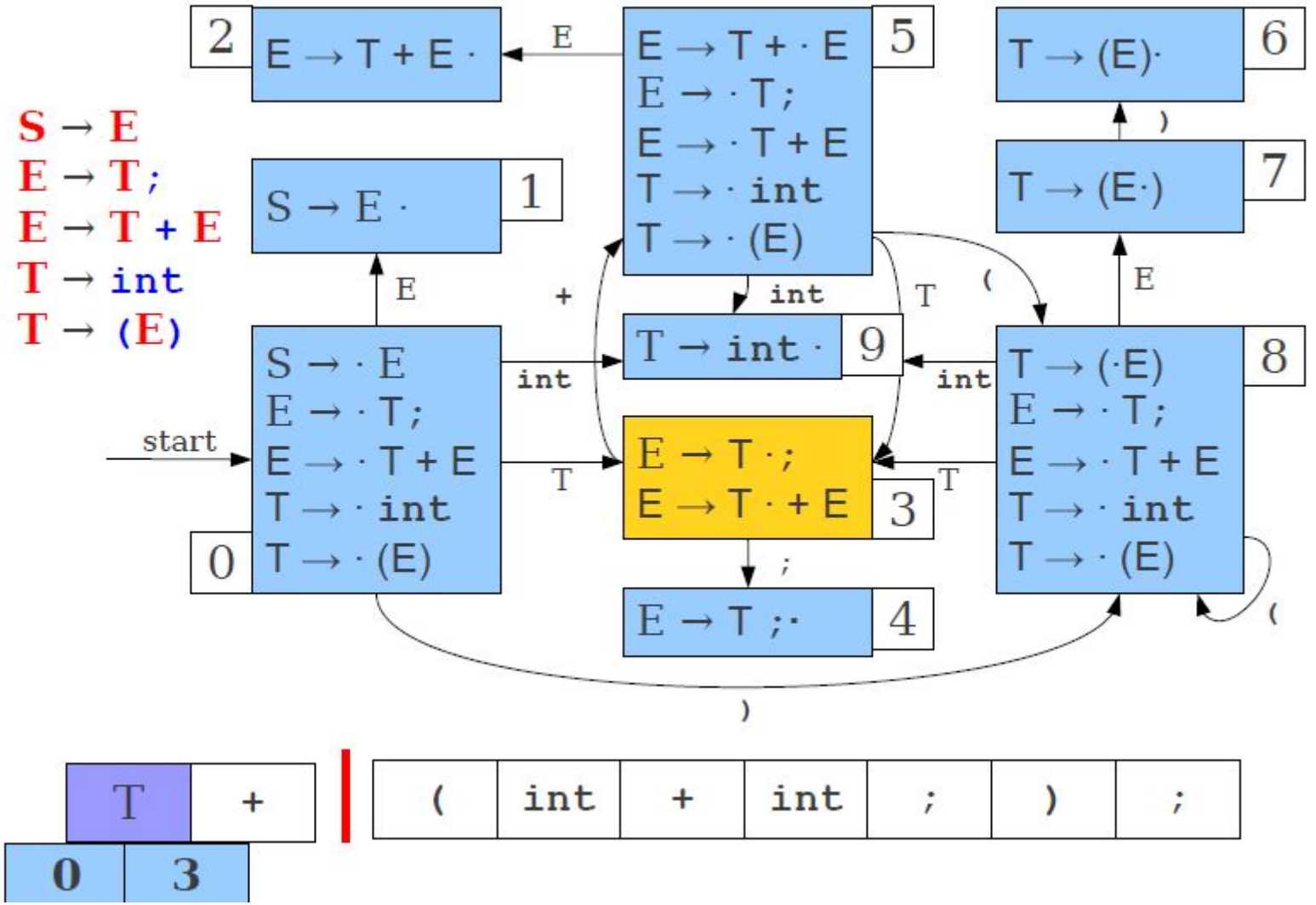
LR(0) Parsing


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

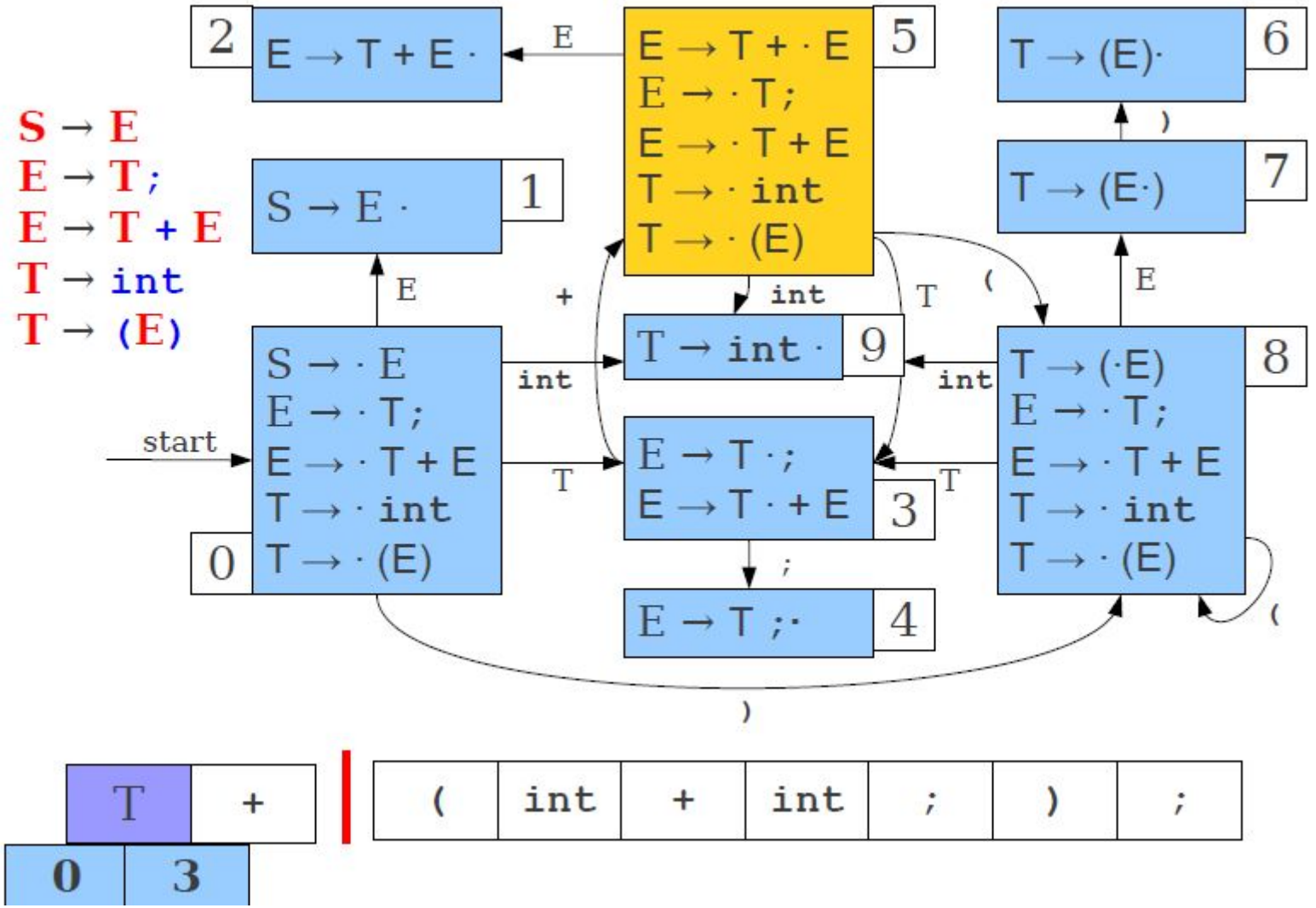
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
start
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 ·
E
T
int
+
(
)
;
T
0
3
+
(
int
+
int
;
)
;

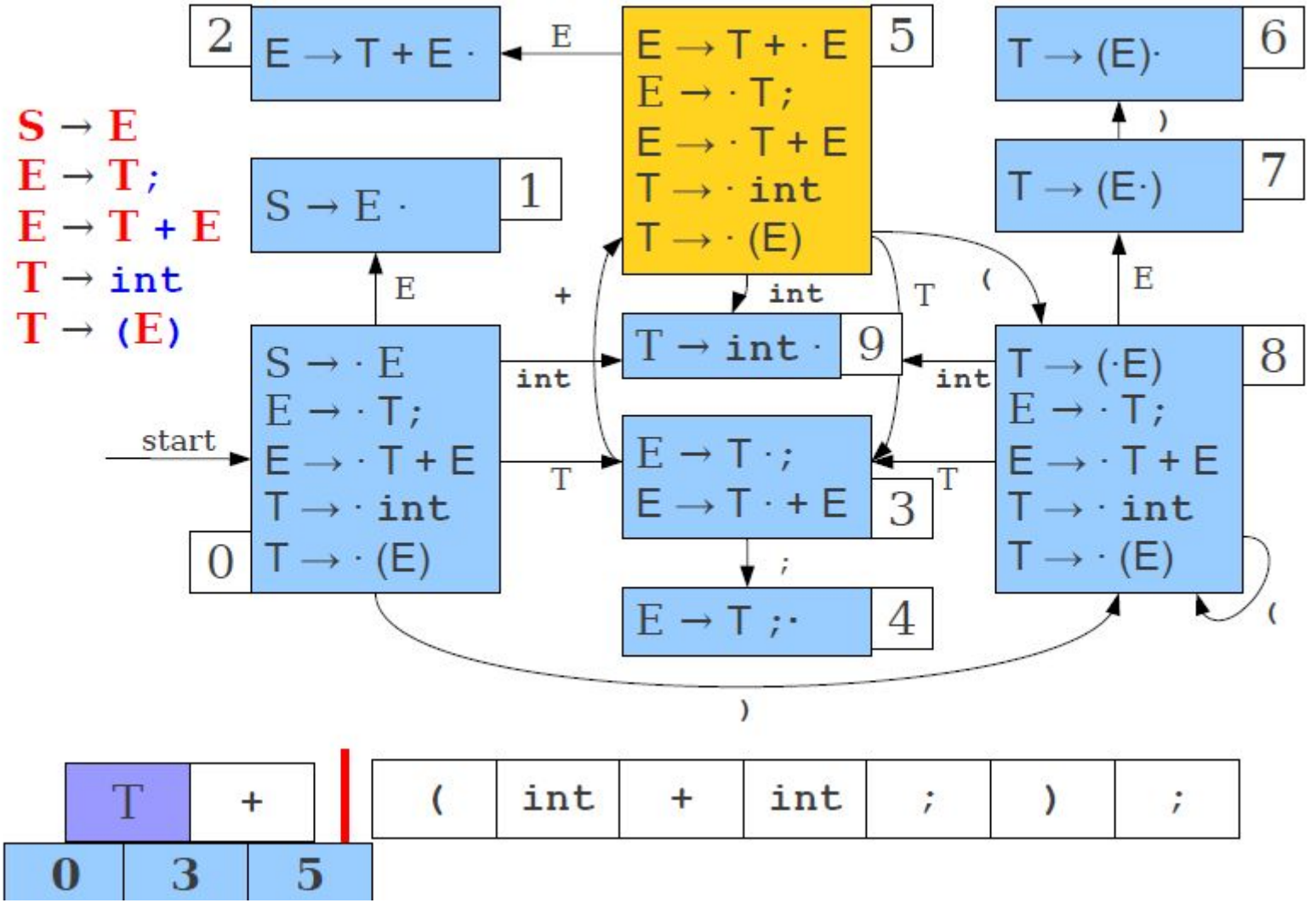
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
start
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 ·
E
T
int
+
;
(
)
T
+
0
3
(
int
+
int
;
)
;

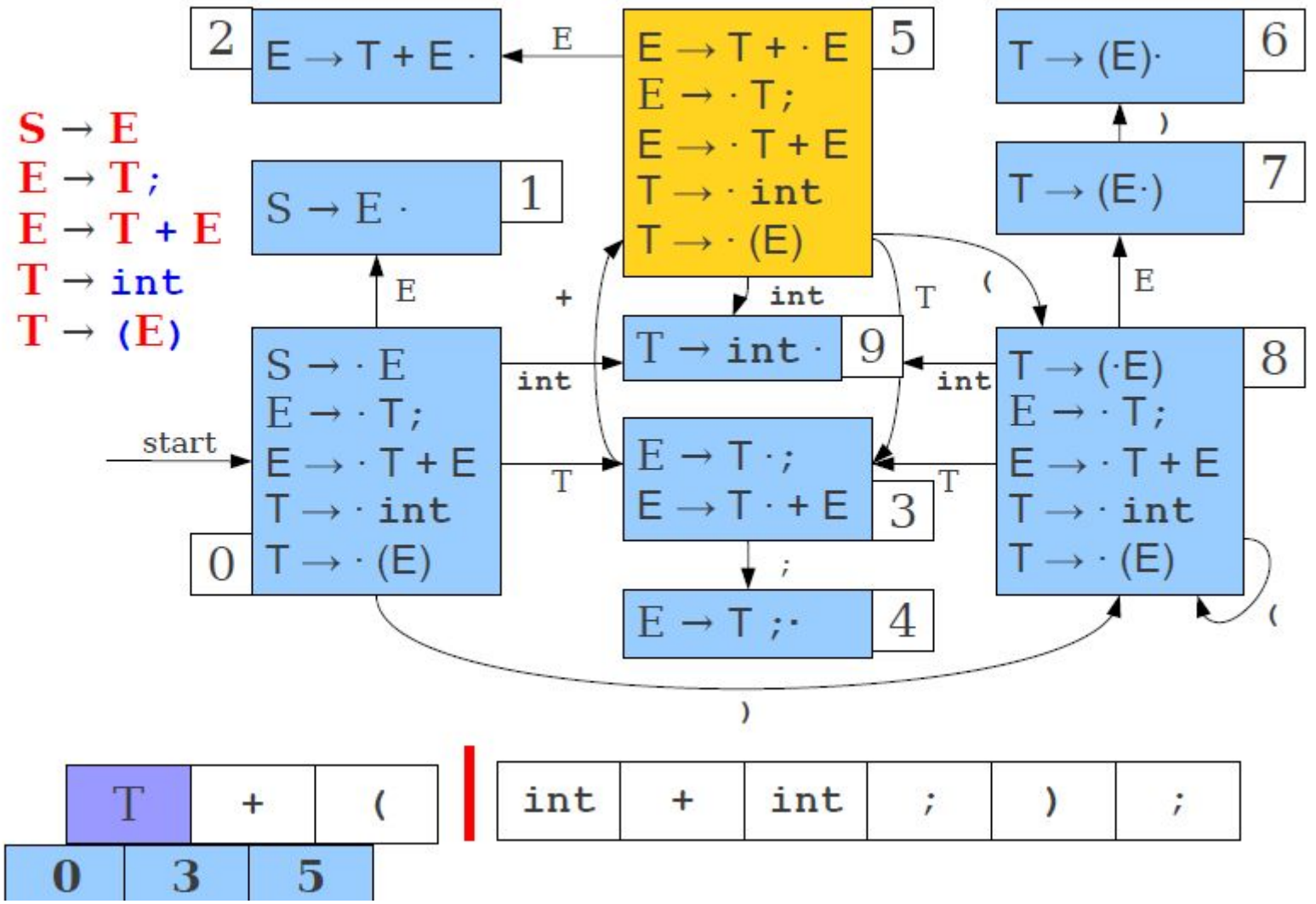
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
start
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 ·
E
+
int
T
(
)
;
T
+
0
3
(
int
+
int
;
)
;

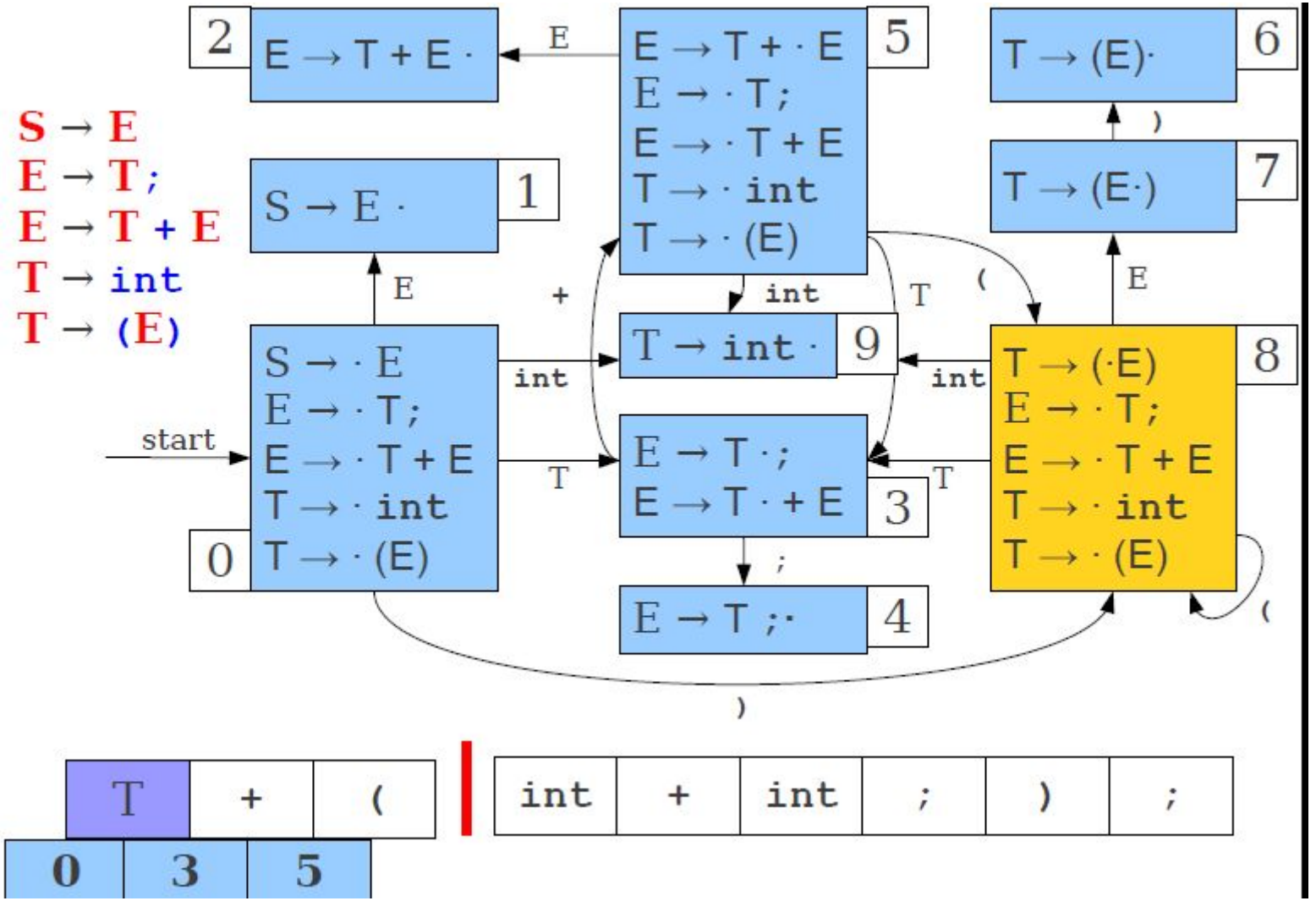
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
start
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 ·
E
+
int
T
(
)
;
T
+
0
3
5
(
int
+
int
;
)
;

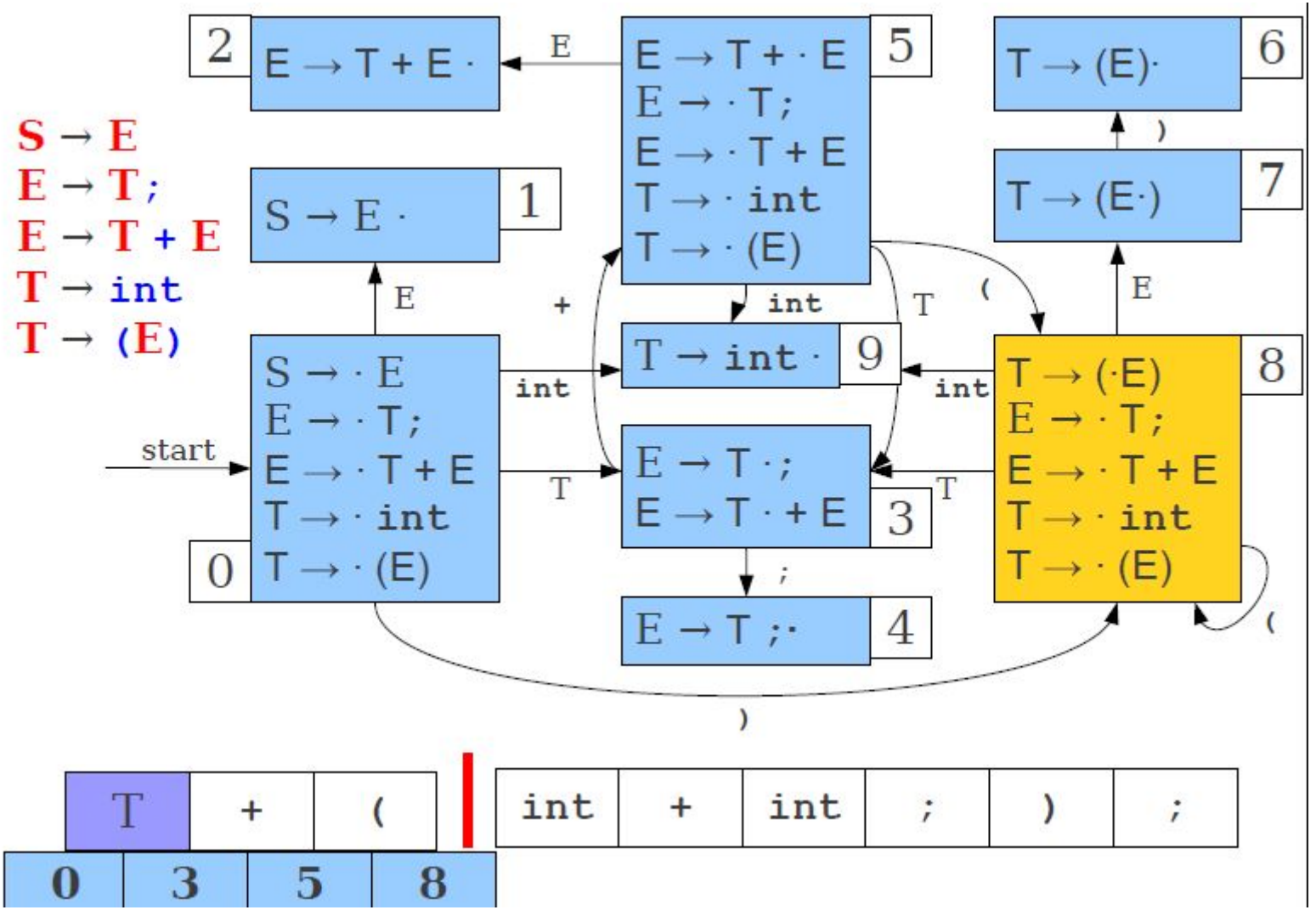
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
T → int ·
9
int
int
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
T
E → T ·;
E → T · + E
3
;
E → T ;·
4
(
)
T
+
(
0
3
5
int
+
int
;
)
;

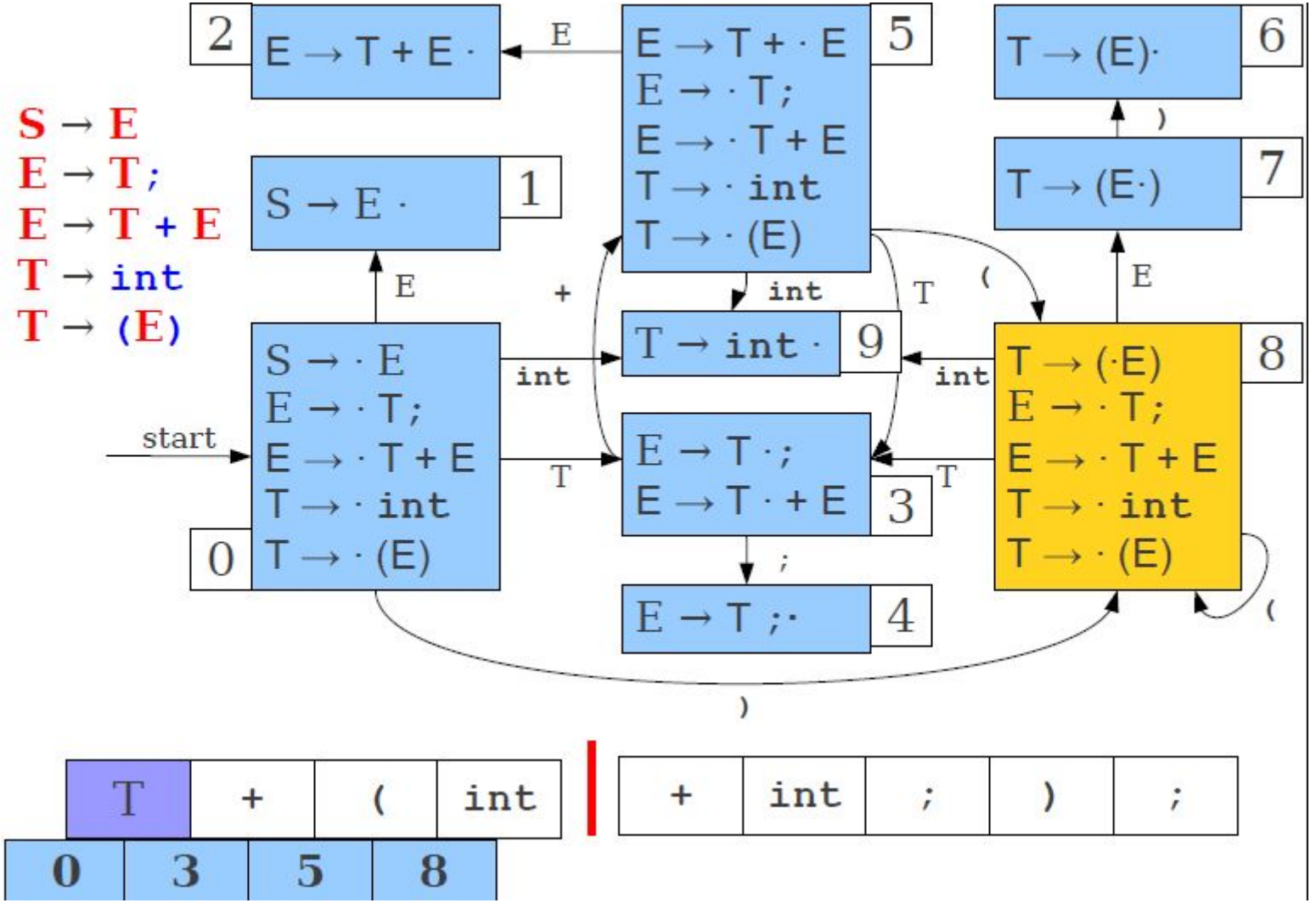
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
S → E ·
1
E → T + E ·
2
E → T ·;
E → T · + E
3
E → T ;·
4
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
T → (E·)
7
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
T → int · 9
E
+
T
int
(
)
;
T
+
(
0
3
5
int
+
int
;
)
;

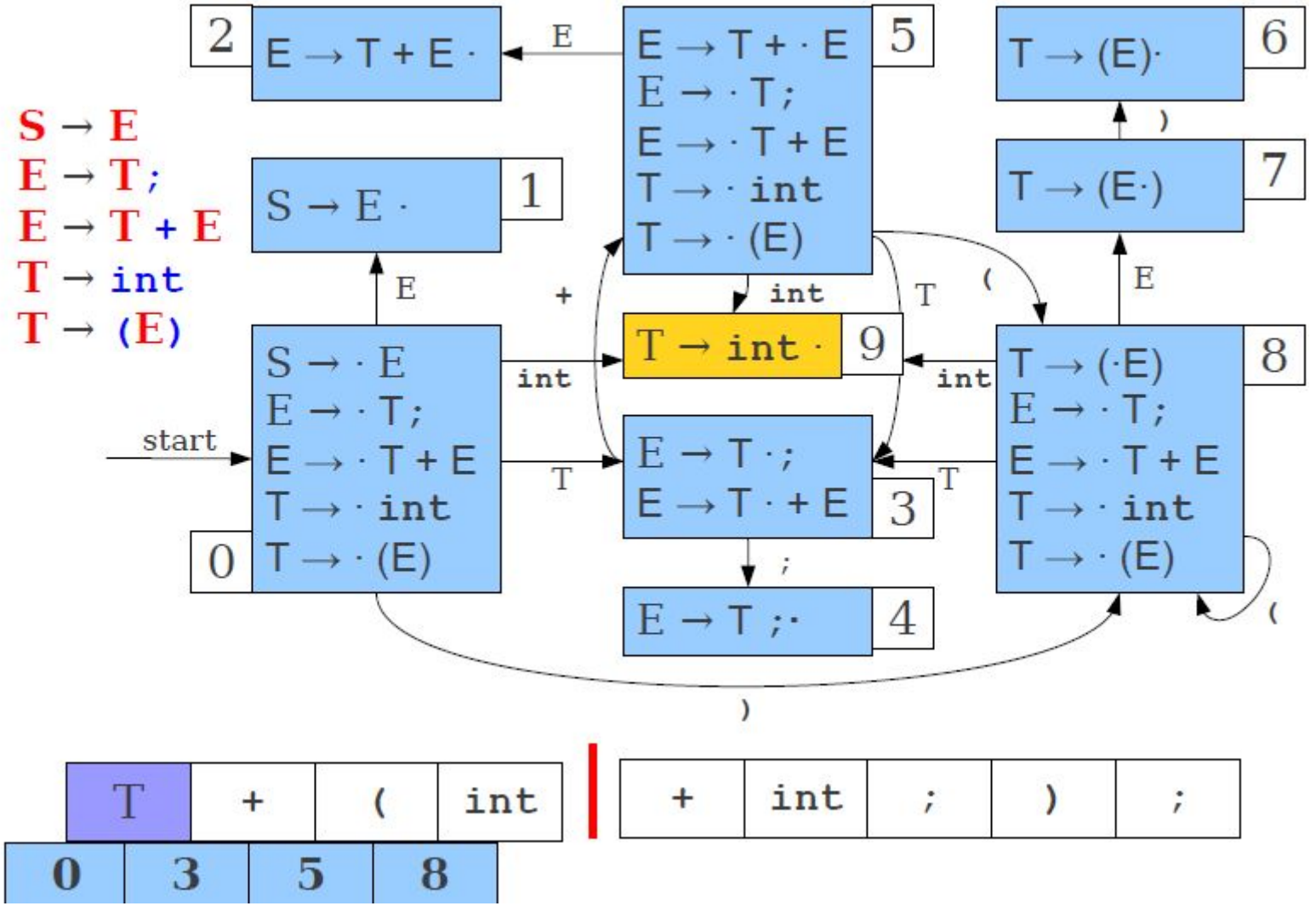
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
start
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 ·
E
+
T
int
(
)
;
T
+
(
0
3
5
8
int
+
int
;
)
;

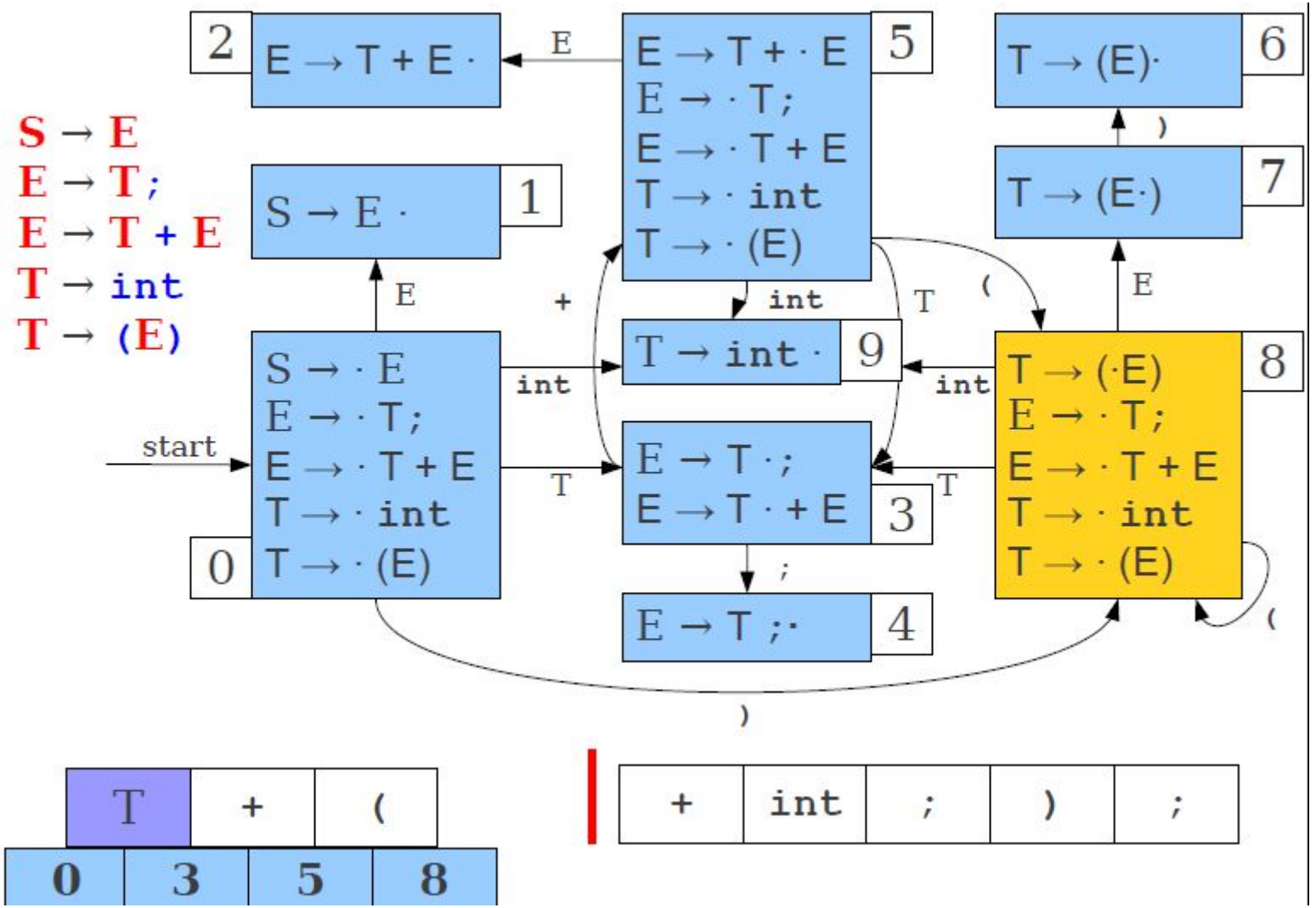
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
S → E ·
1
E → T + E ·
2
E → T · ;
E → T · + E
3
E → T ;·
4
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
T → (E·)
7
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
T → int ·
9
E
T
int
+
(
)
;
T + (int
0 3 5 8
+ int ;) ;

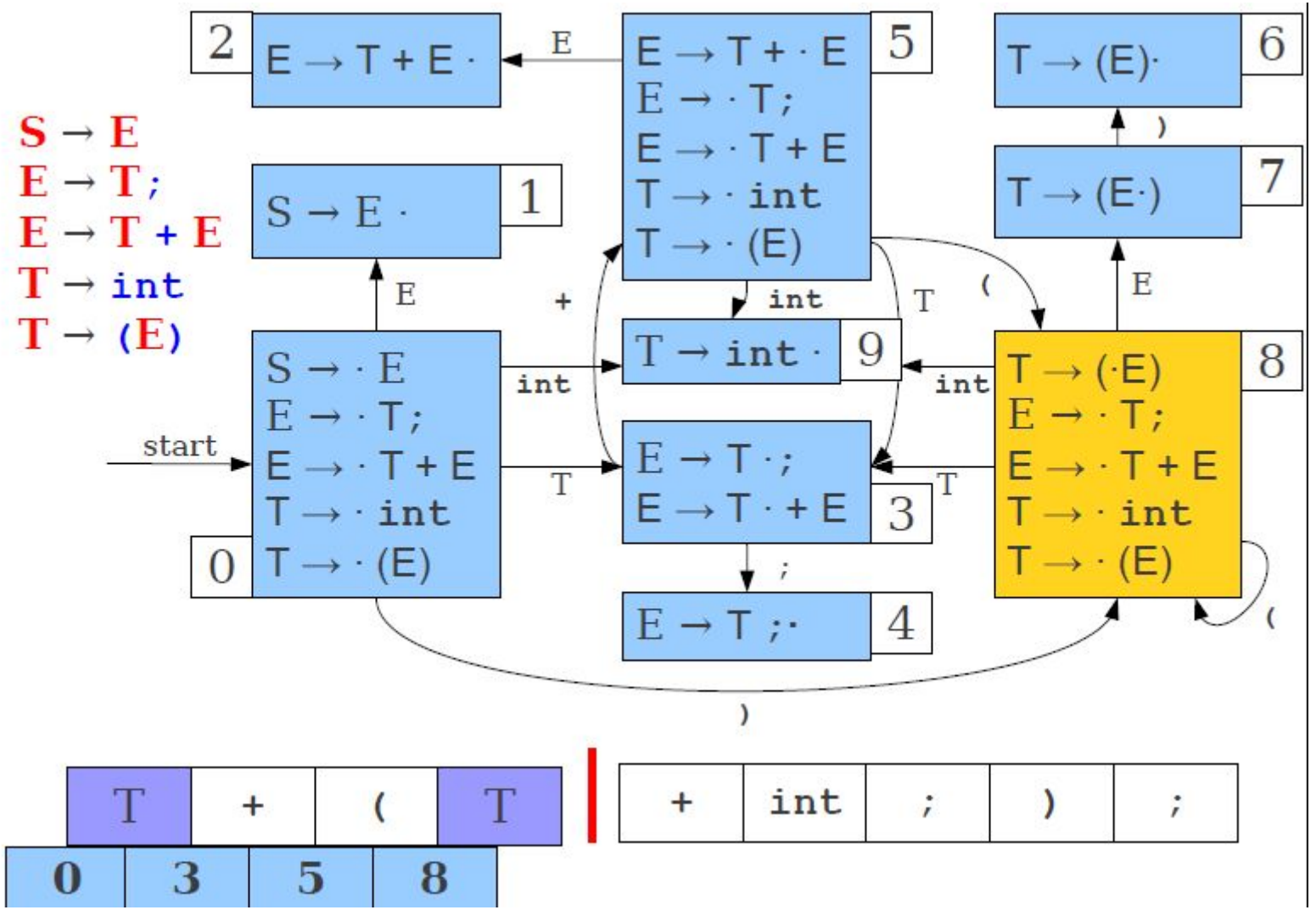
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
start
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 ·
E
T
int
+
;
(
)
T
+
(
int
0
3
5
8
+
int
;
)
;

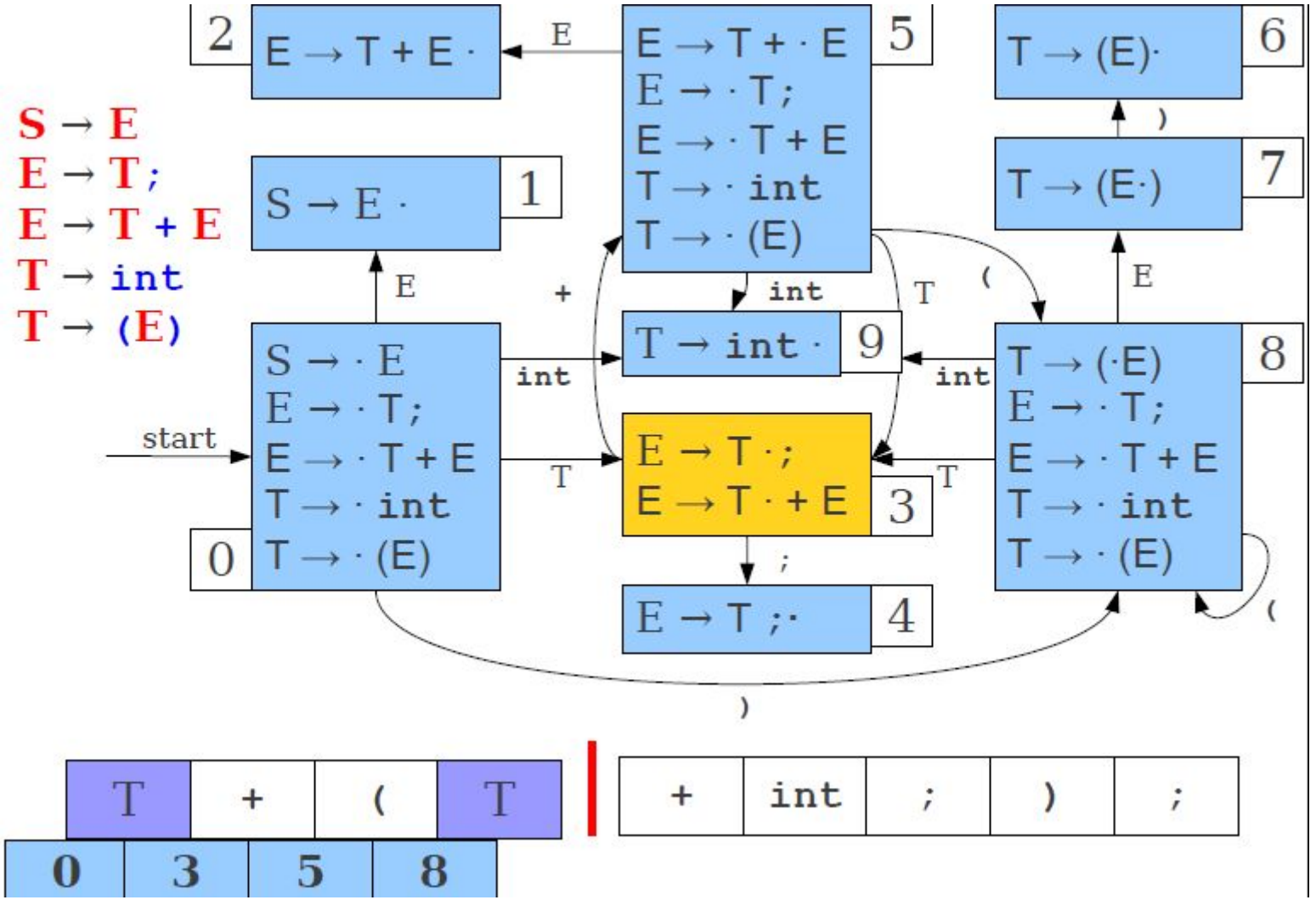
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
start
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 ·
E
T
int
+
;
(
)
T
+
(
0
3
5
8
+
int
;
)
;

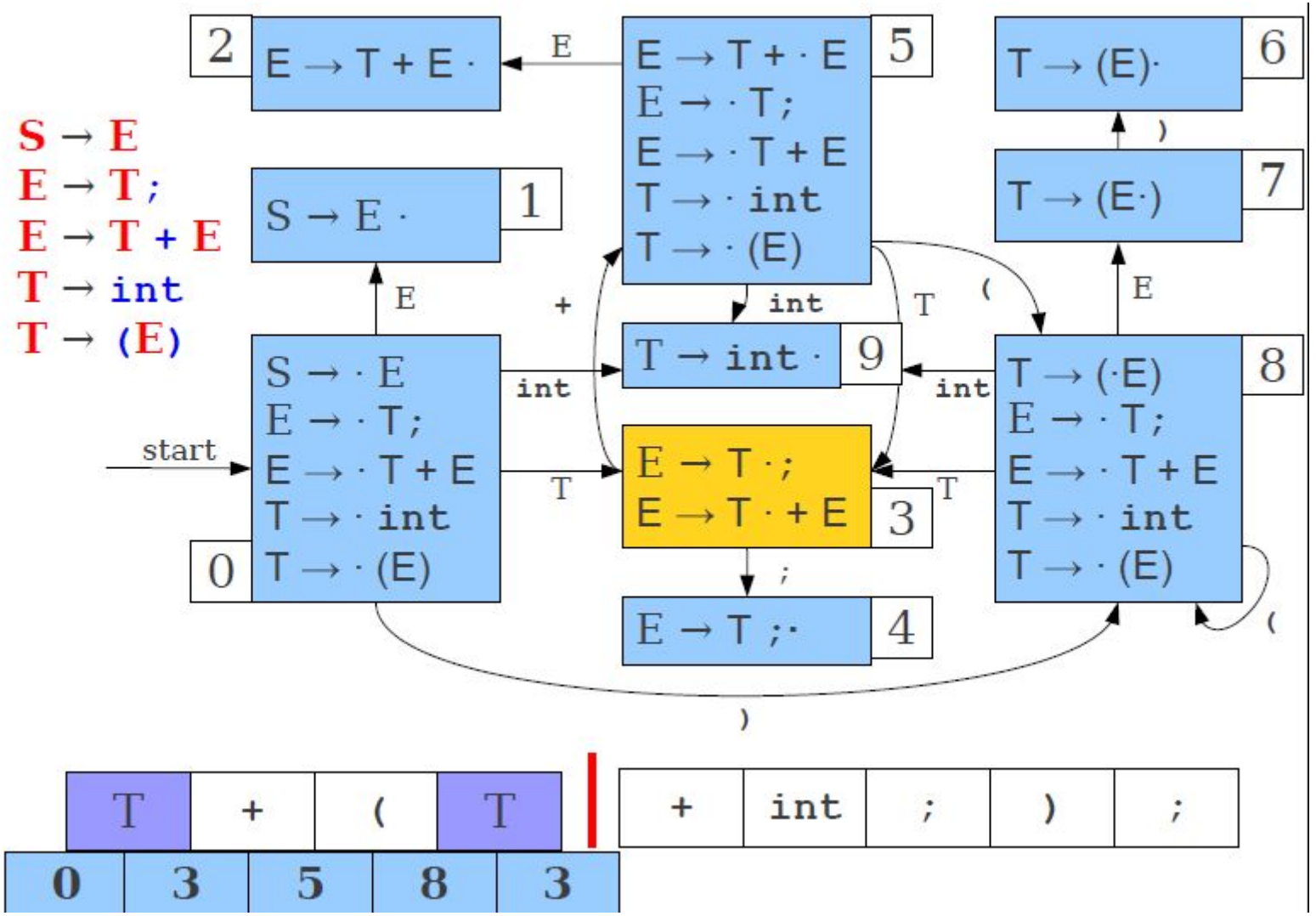
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
T → int ·
9
int
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
int
T
E → T ·;
E → T · + E
3
T
;
E → T ;·
4
)
(
T
+
(
T
0
3
5
8
+
int
;
)
;

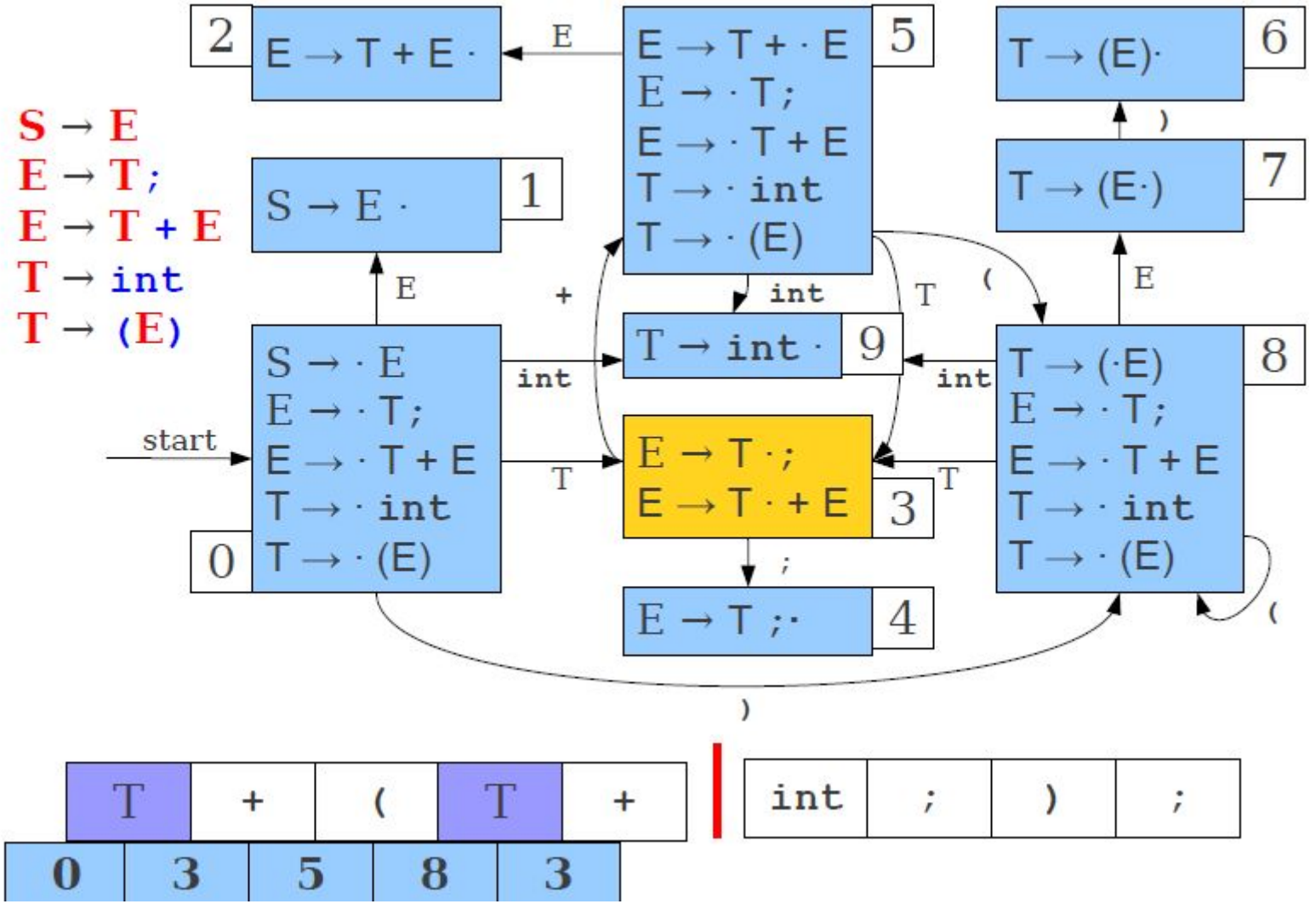
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
S → E ·
1
E → T + E ·
2
E → T ; ·
4
E → T · ;
E → T · + E
3
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → int ·
9
T → (E)·
6
T → (E·)
7
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
E
+
int
T
(
)
;
T + (T
0 3 5 8
+ int ;) ;

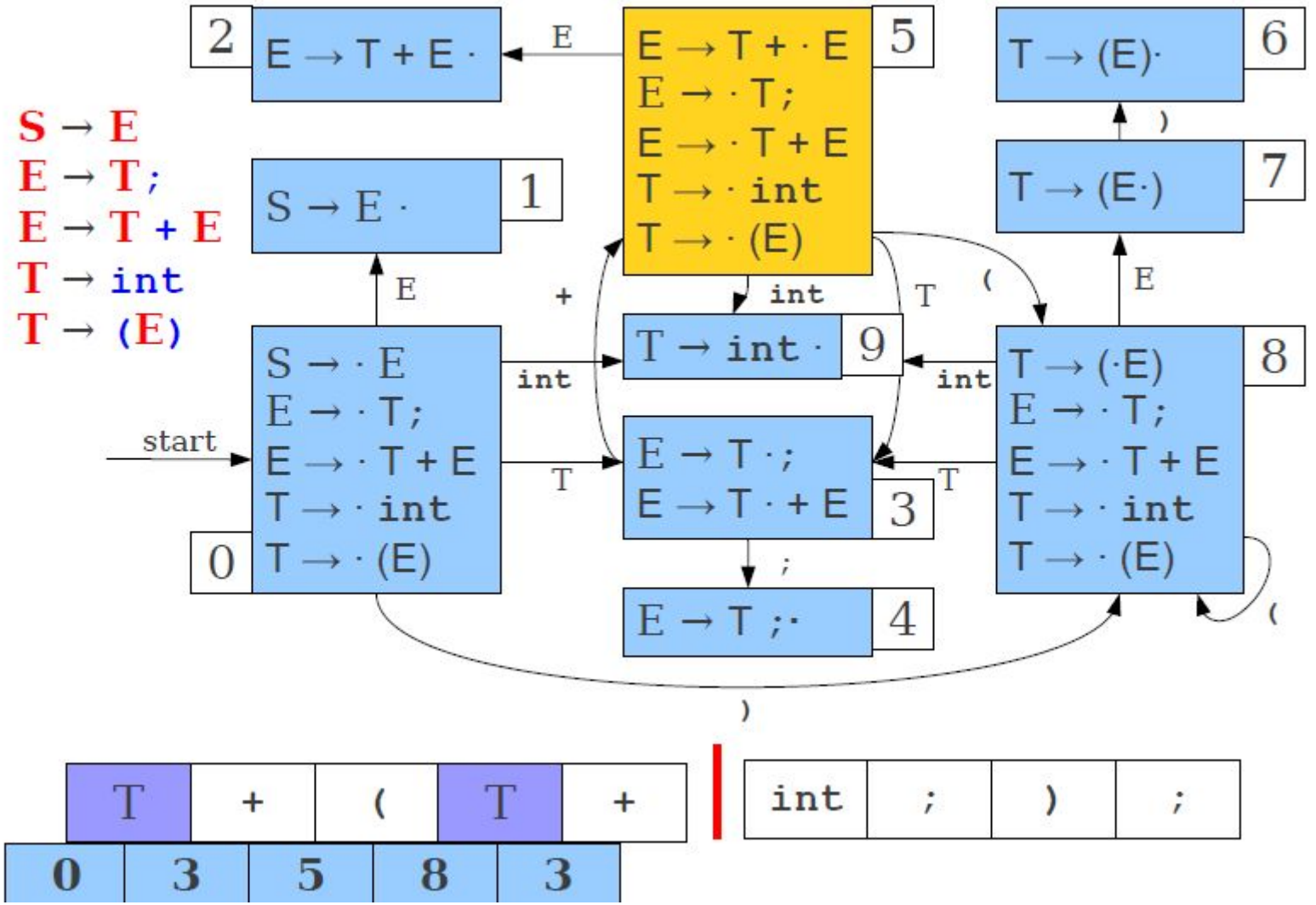
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
T → int ·
9
int
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
T
E → T ·;
E → T · + E
3
T
;
E → T ;·
4
)
(
T
+
(
T
+
int
;
)
;
0
3
5
8
3

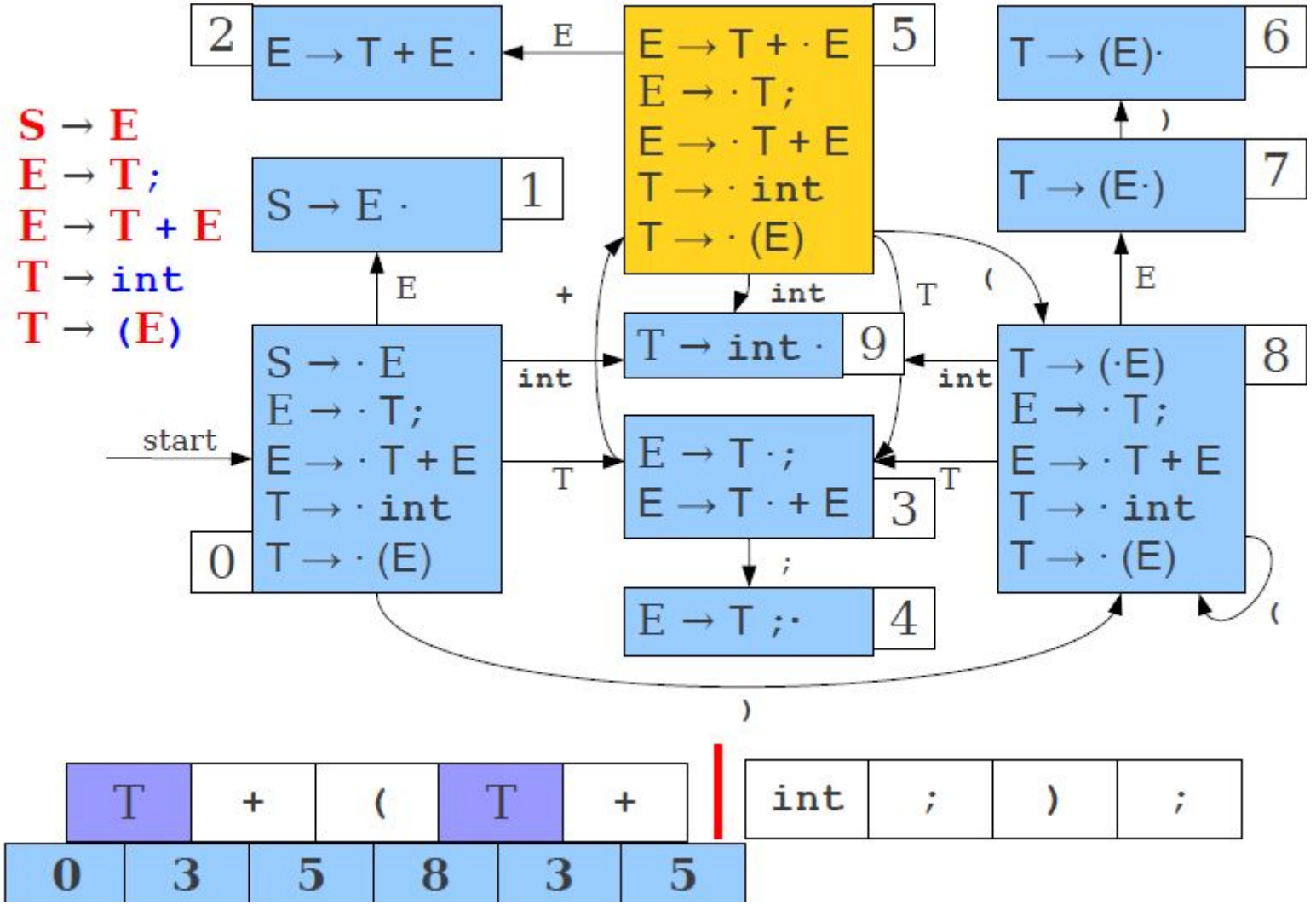
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
T → int ·
9
int
int
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
start
0
T
E → T · ;
E → T · + E
3
T
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
;
E → T ;·
4
)
T
+
(
T
+
0
3
5
8
3
int
;
)
;

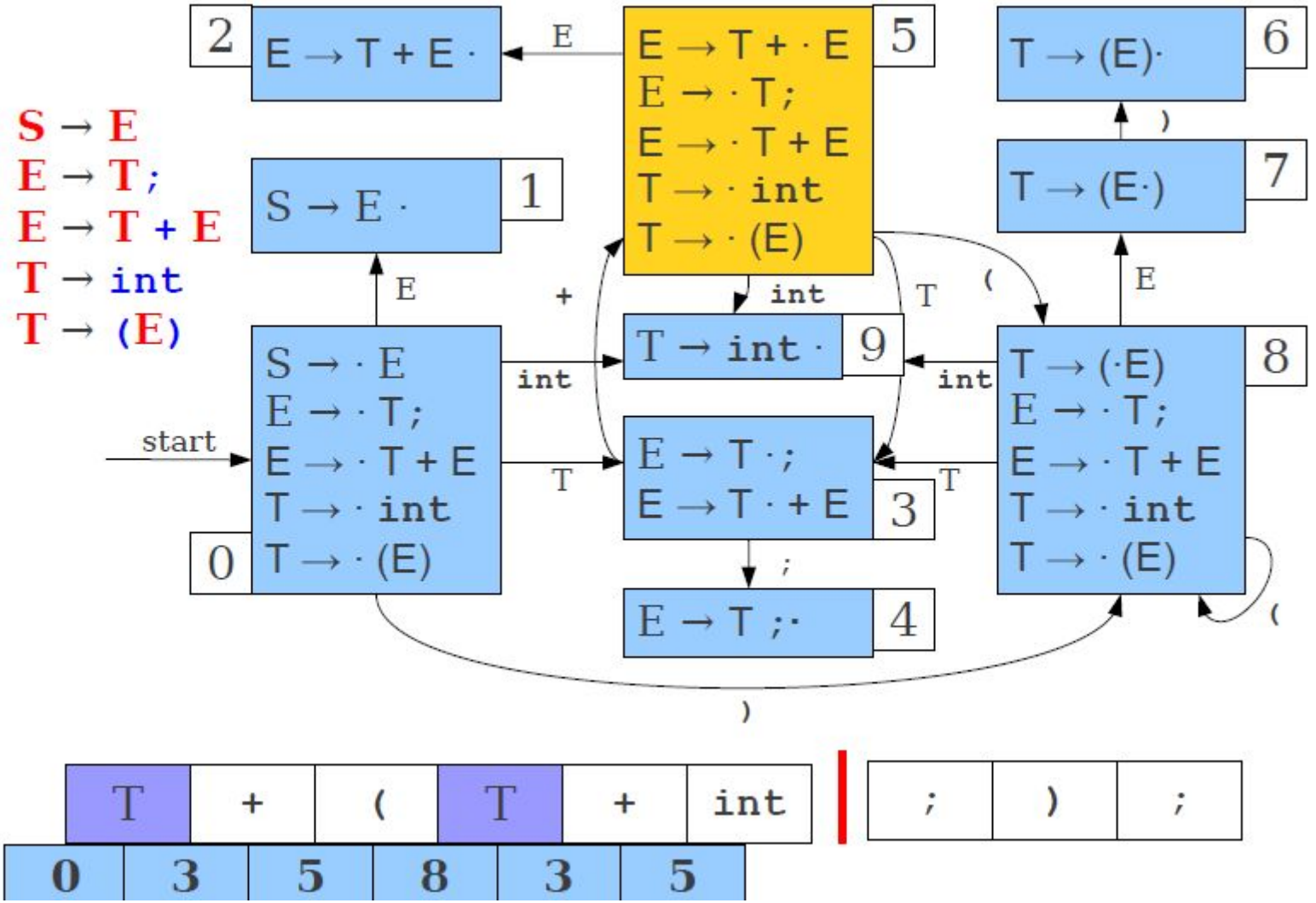
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
start
int
T → int ·
9
int
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
T
E → T ·;
E → T · + E
3
;
E → T ;·
4
)
(
T
+
(
T
+
0
3
5
8
3
int
;
)
;

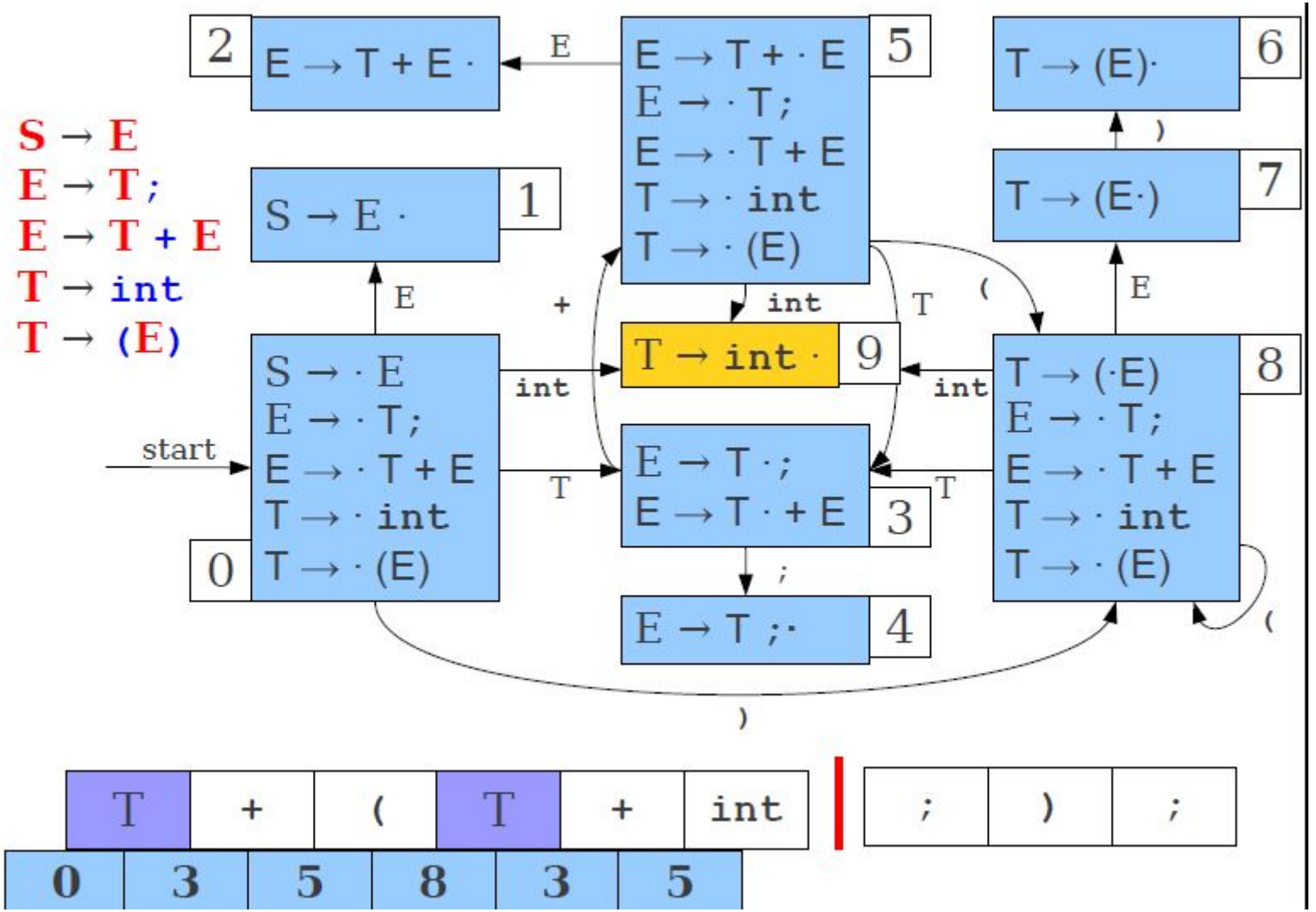
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
start
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 ·
E
+
int
T
(
)
;
T + (T + int ;) ;
0 3 5 8 3 5

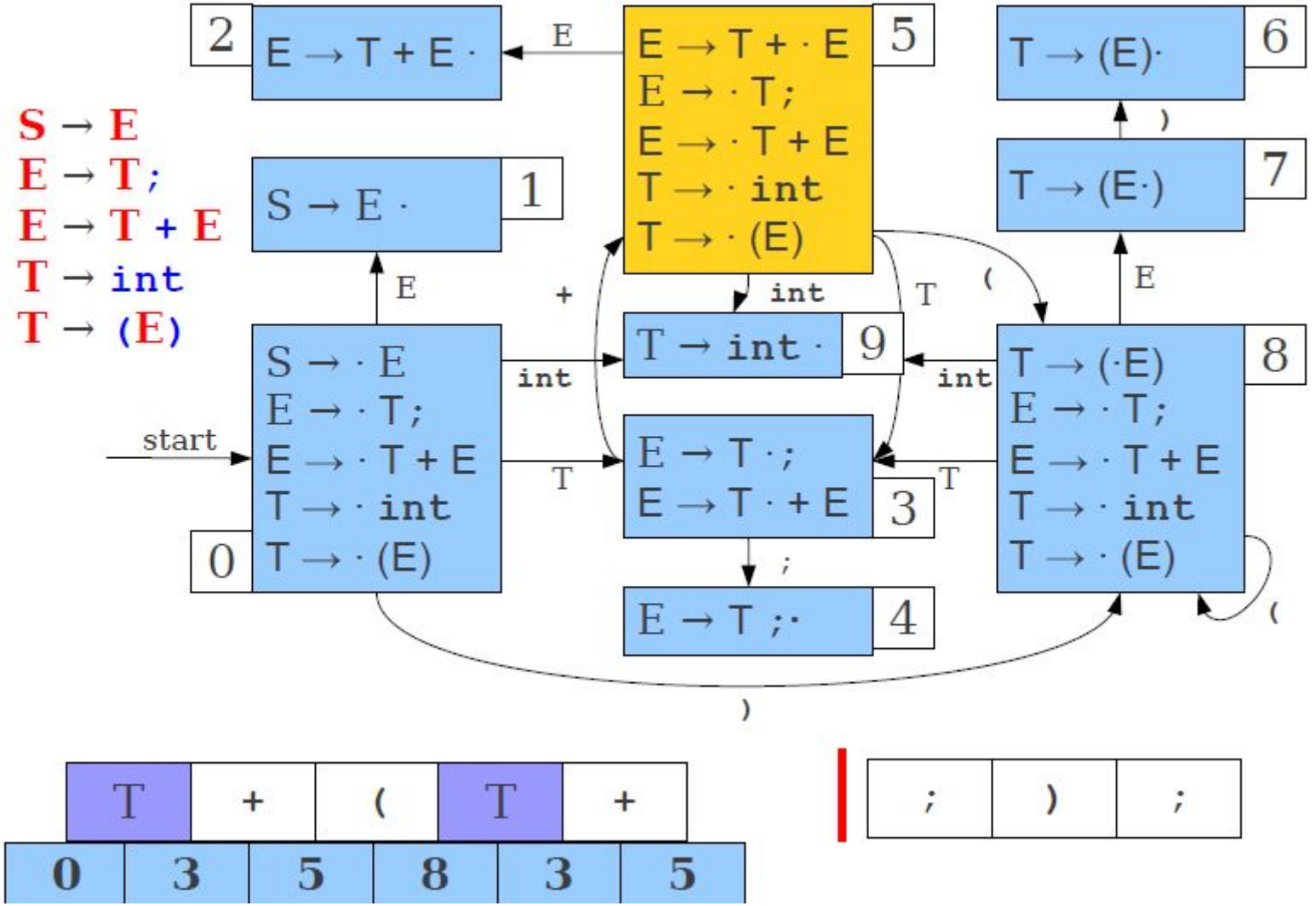
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
start
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 ·
E
+
int
T
(
)
;
T + (T + int
0 3 5 8 3 5
;) ;

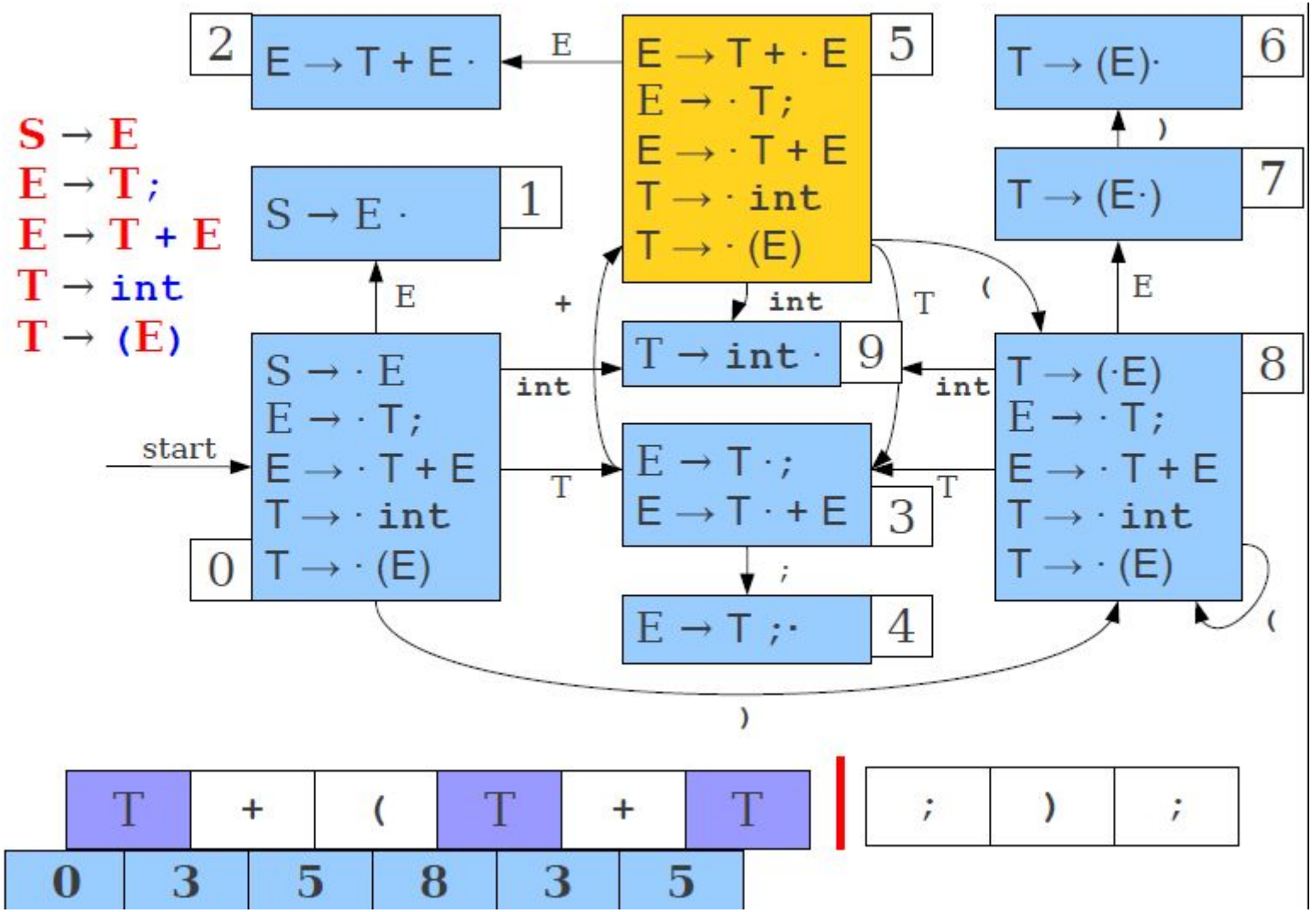
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
T → int ·
9
int
int
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
T
E → T ·;
E → T · + E
3
T
;
E → T ;·
4
(
)
T
+
(
T
+
int
;
)
;
0
3
5
8
3
5

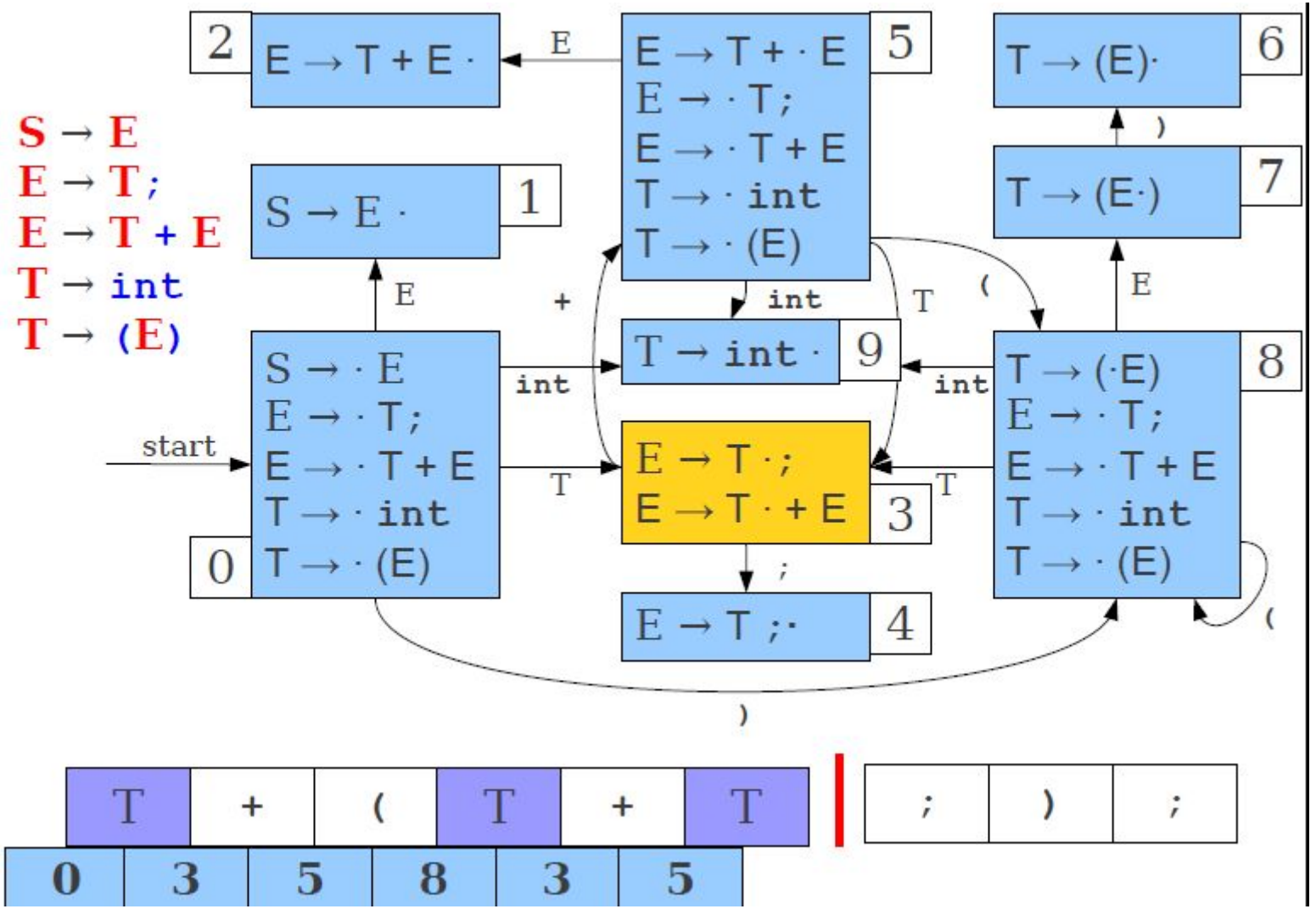
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
T → int ·
9
int
int
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
T
E → T ·;
E → T · + E
3
T
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
;
E → T ;·
4
)
(
T
+
(
T
+
0
3
5
8
3
5
;
)
;

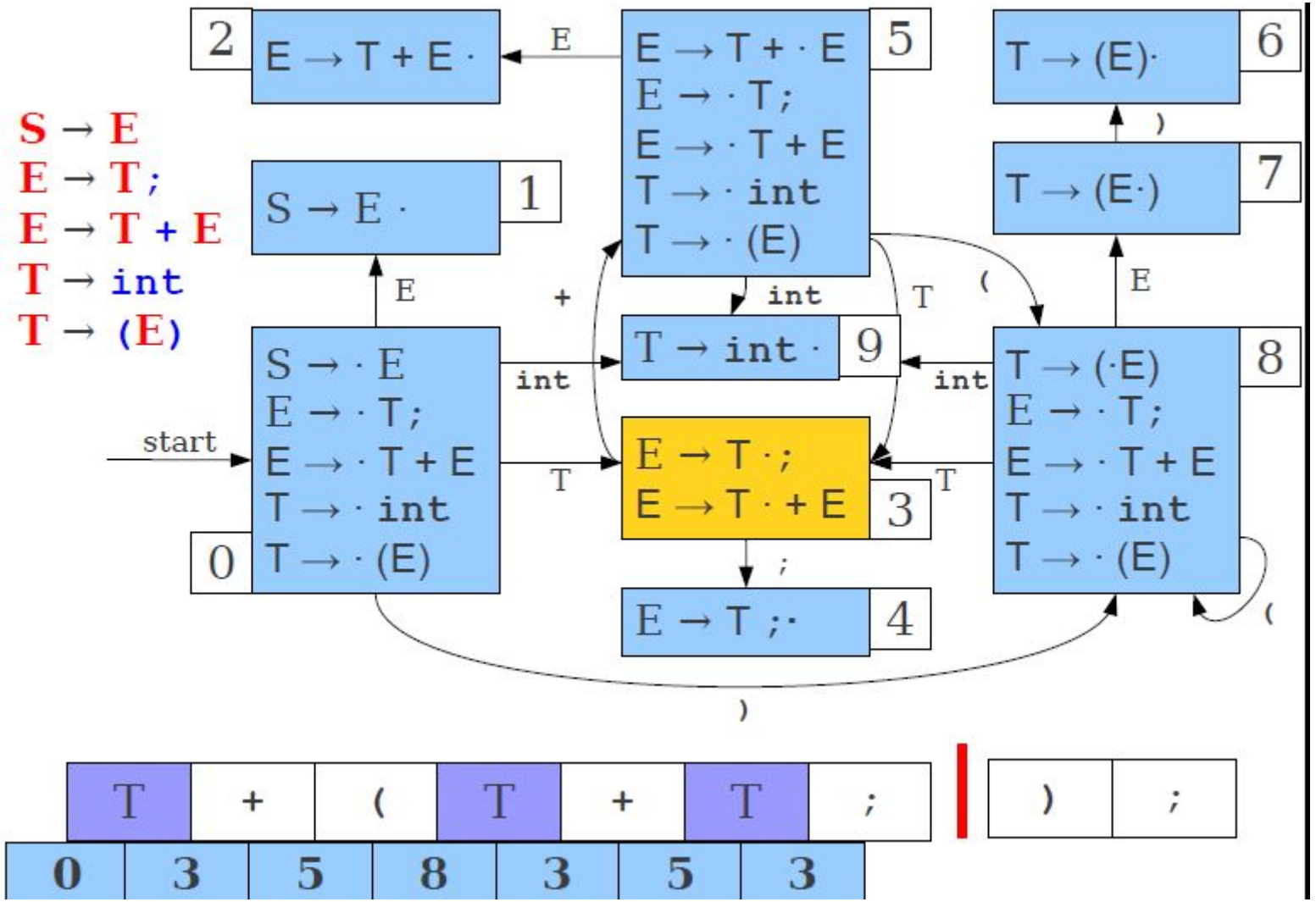
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
start
int
T → int ·
9
int
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
T
E → T ·;
E → T · + E
3
;
E → T ;·
4
)
T + (T + T ;) ;
0 3 5 8 3 5

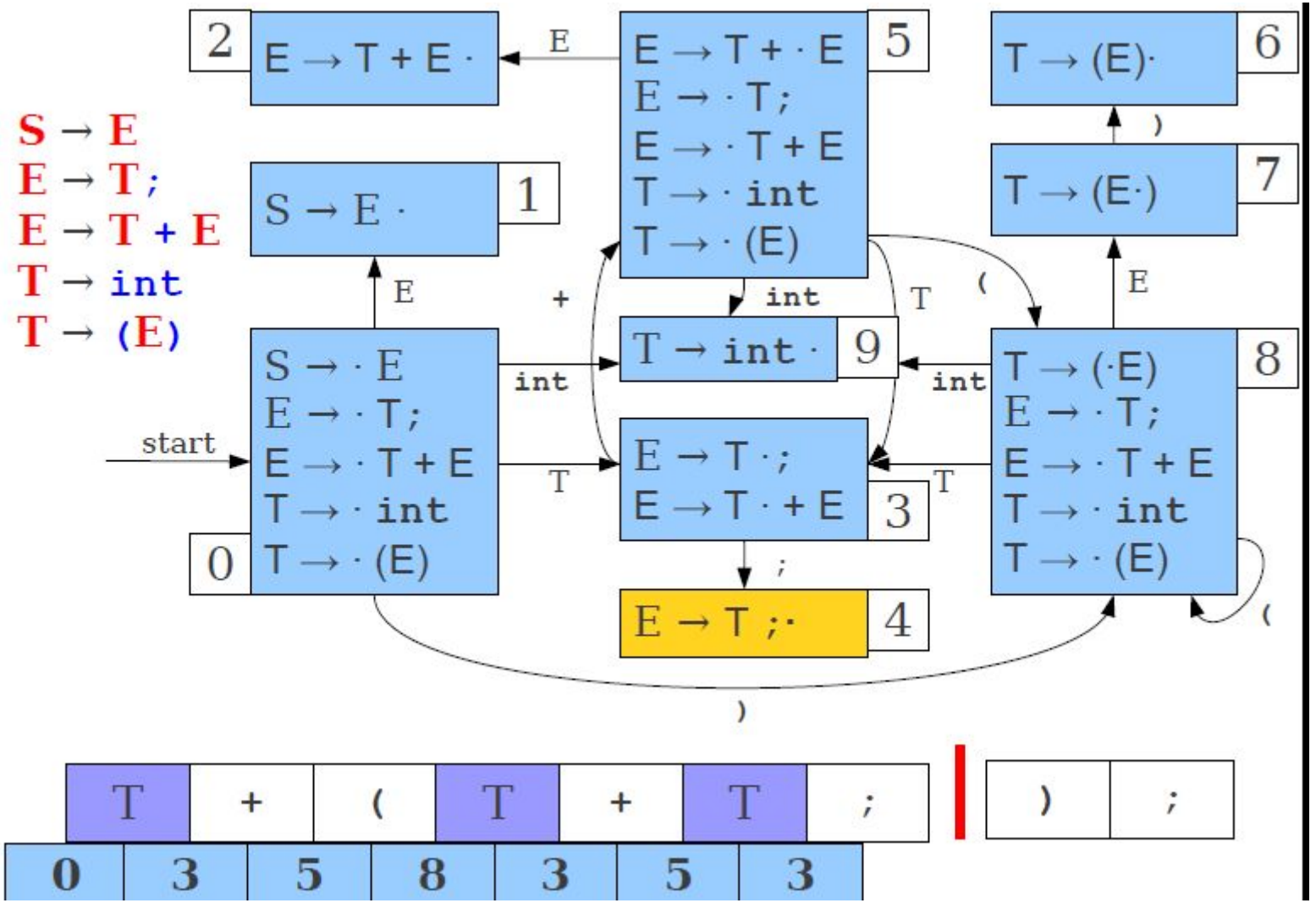
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
start
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 ·
E
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)
0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
start
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 ·
E
+
T
int
(
)
;
T + (T + T ;) ;
0 3 5 8 3 5 3

LR(0) Parsing

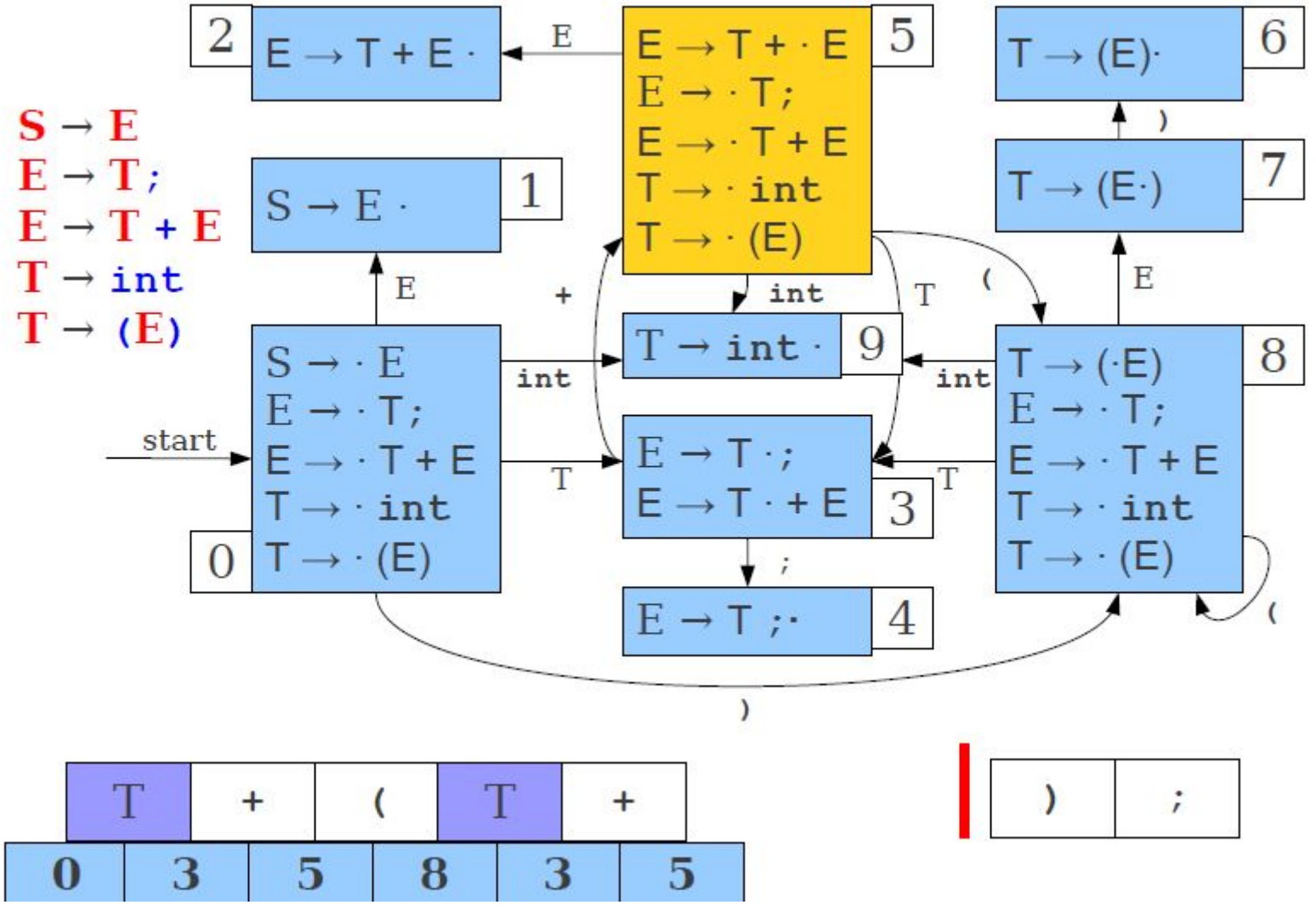

S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
6
T → (E)·
)
7
T → (E·)
1
S → E ·
E
+
int
T
(
E
9
T → int ·
int
int
8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
T
E → T ·;
E → T · + E
3
;
4
E → T ;·
)
T + (T + T ;) ;
0 3 5 8 3 5 3

LR(0) Tables

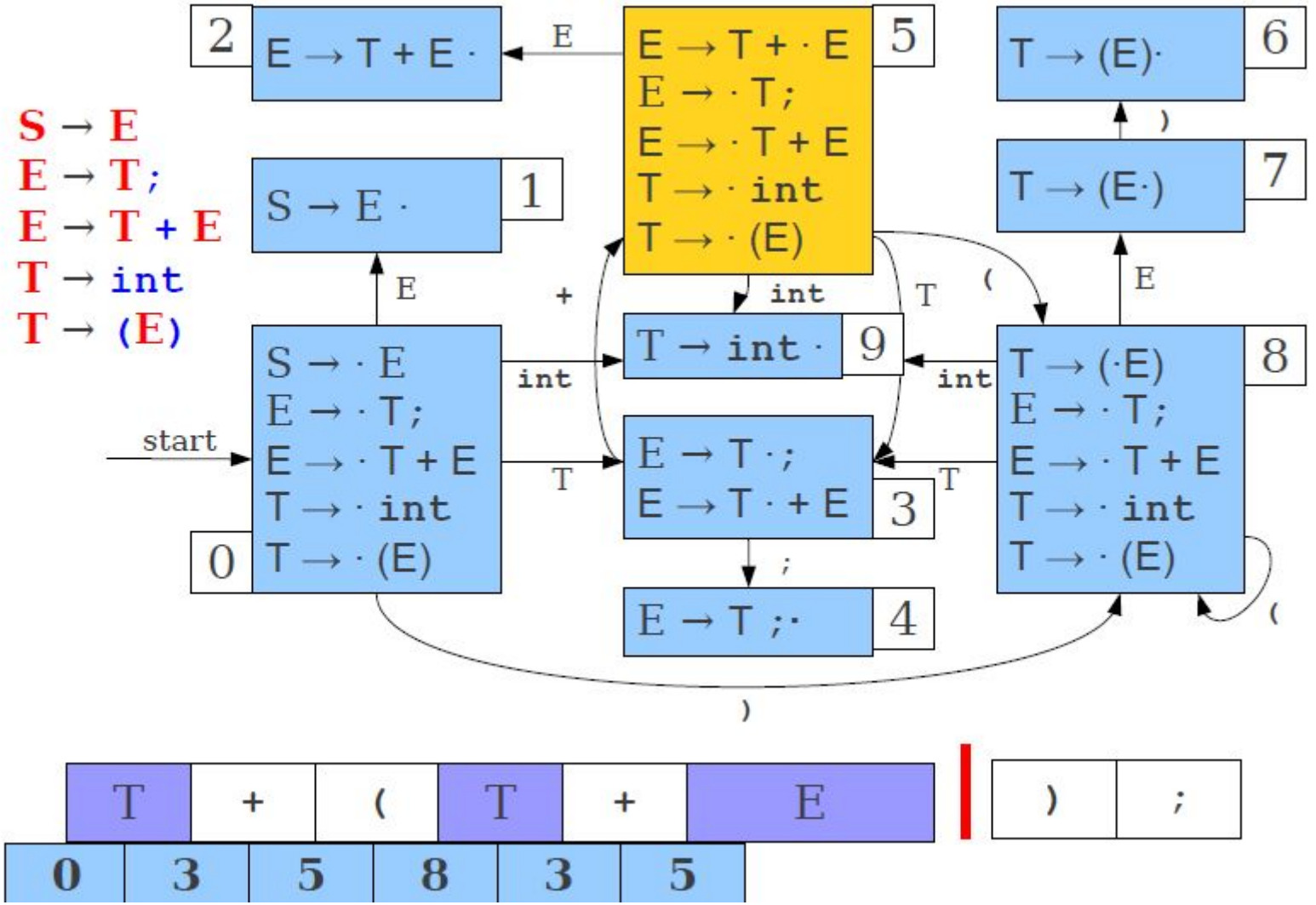
(1) S → E
(2) E → T;
(3) E → T + E
(4) T → int
(5) T → (E)

	int	+	;	(	)	E	T
0	S9			S8		S1	S3
1							
2	r3	r3	r3	r3	r3		
3		S5	S4				
4	r2	r2	r2	r2	r2		
5	S9			S8		S2	S3
6							
7					s6		
8	S9			S8		S7	S3
9	r4	r4	r4	r4	r4		

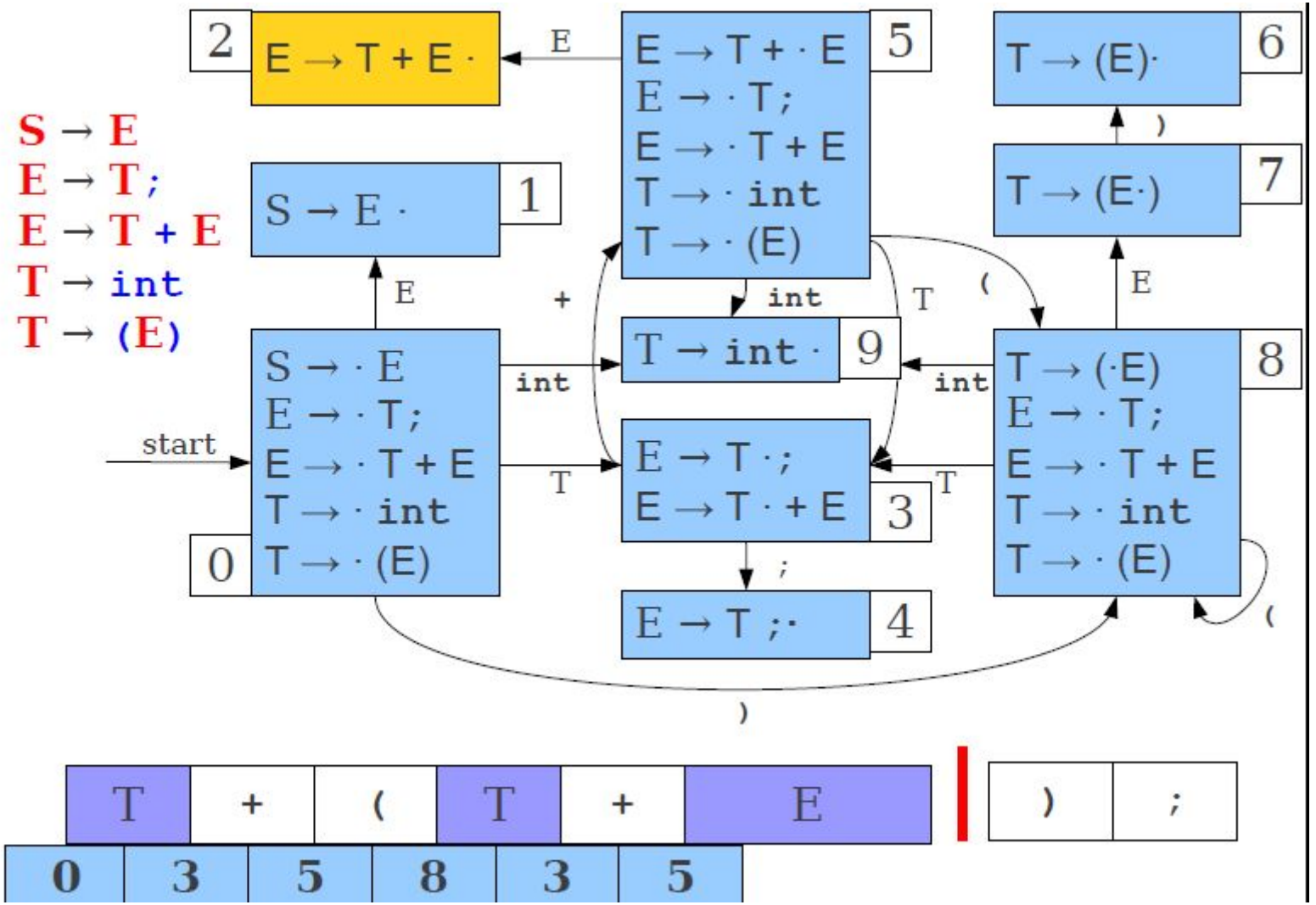
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
start
int
T → int ·
9
int
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
T
E → T ·;
E → T · + E
3
T
;
E → T ;·
4
)
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 ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
T → int ·
9
int
int
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
start
0
T
E → T ·;
E → T · + E
3
T
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
(
;
E → T ;·
4
)
T
+
(
T
+
E
)
;
0
3
5
8
3
5

LR(0) Parsing

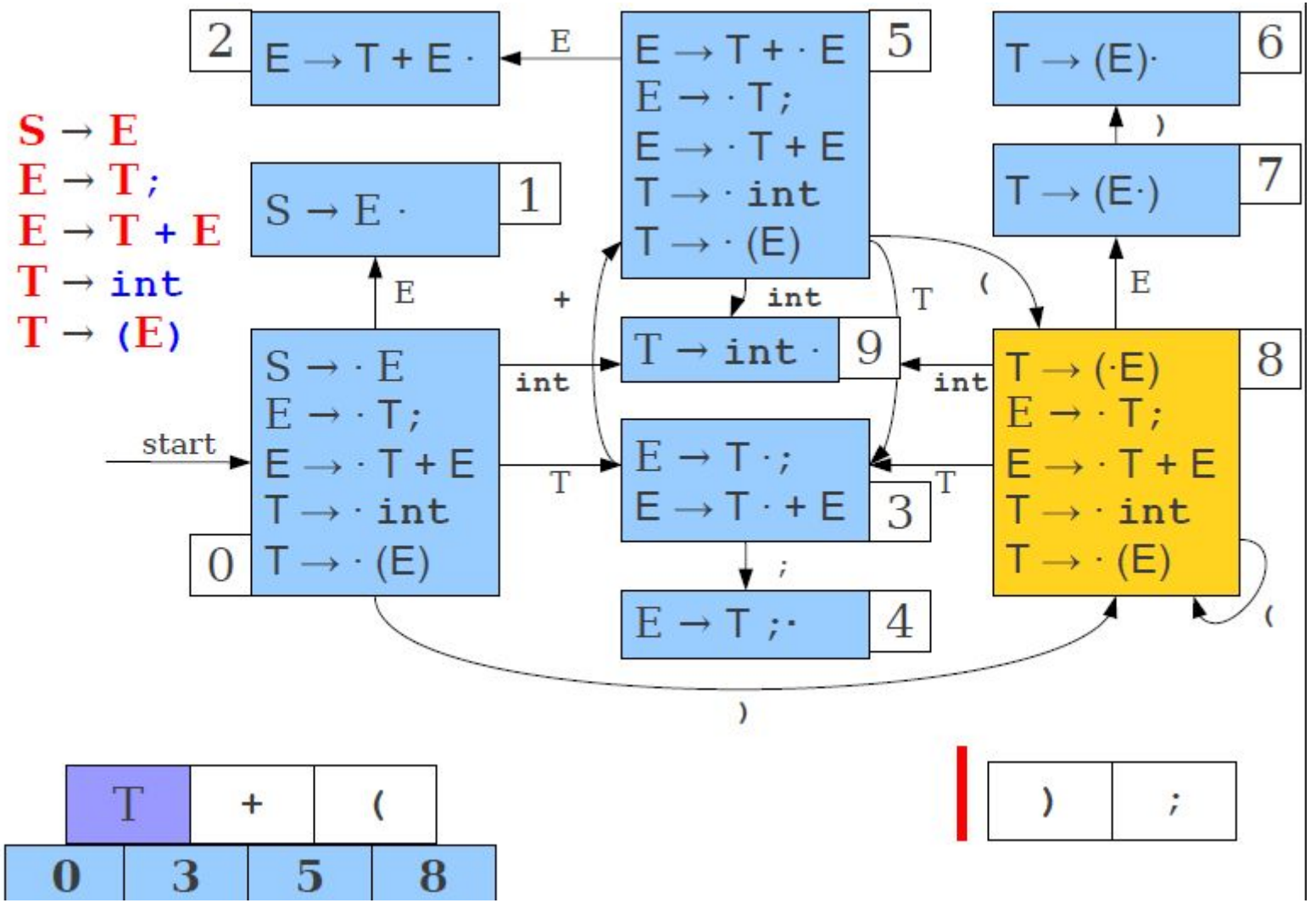

S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
6
T → (E)·
)
7
T → (E·)
1
S → E ·
E
+
int
T
(
E
9
T → int ·
int
int
8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
int
T
E → T ·;
E → T · + E
3
T
;
E → T ;·
4
)
(
T
+
(
T
+
E
)
;
0
3
5
8
3
5

LR(0) Tables

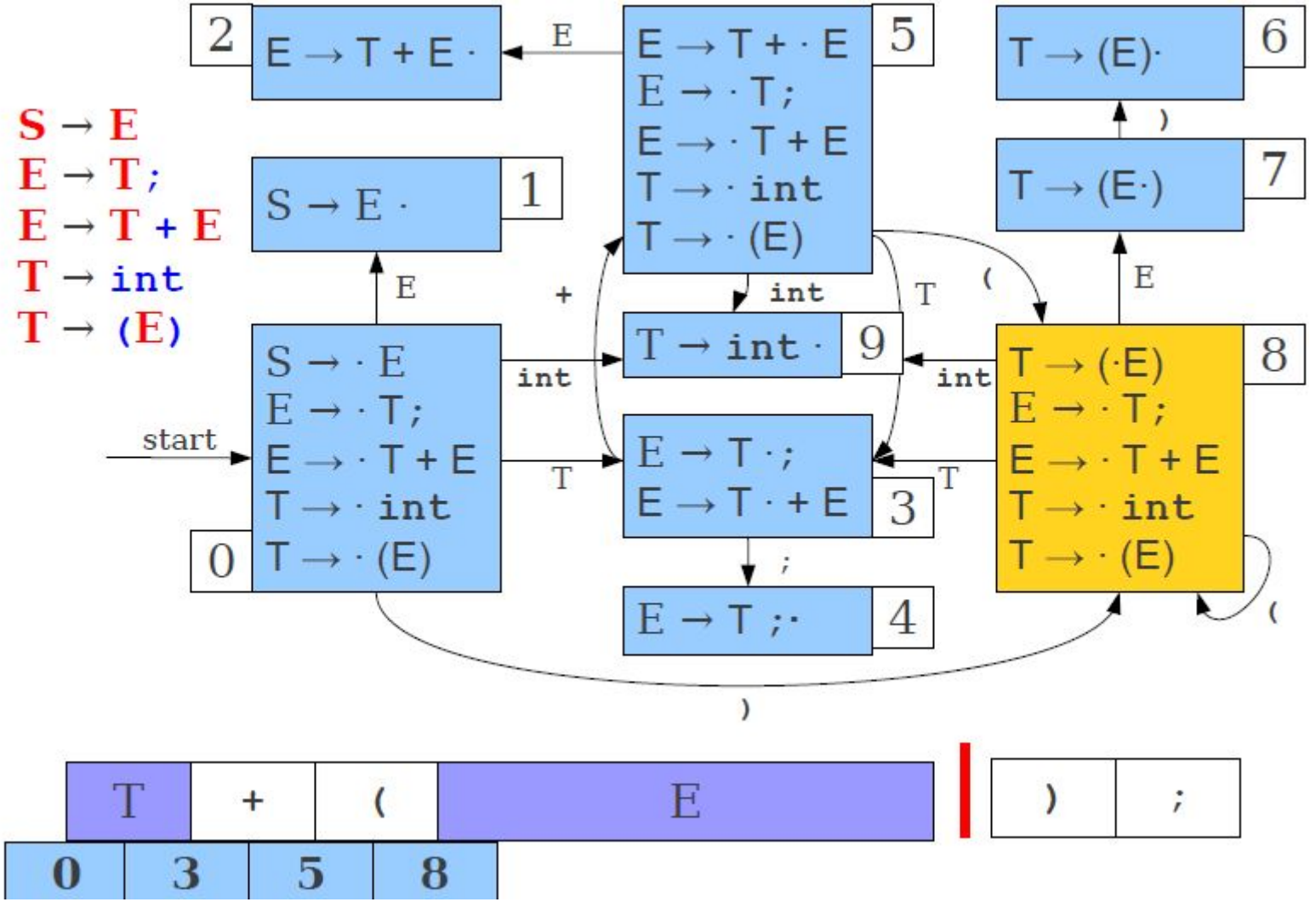
(1) S → E
(2) E → T;
(3) E → T + E
(4) T → int
(5) T → (E)

	int	+	;	(	)	E	T
0	S9			S8		S1	S3
1							
2	r3	r3	r3	r3	r3		
3		S5	S4				
4							
5	S9			S8		S2	S3
6	r5	r5	r5	r5	r5		
7					s6		
8	S9			S8		S7	S3
9	r4	r4	r4	r4	r4		

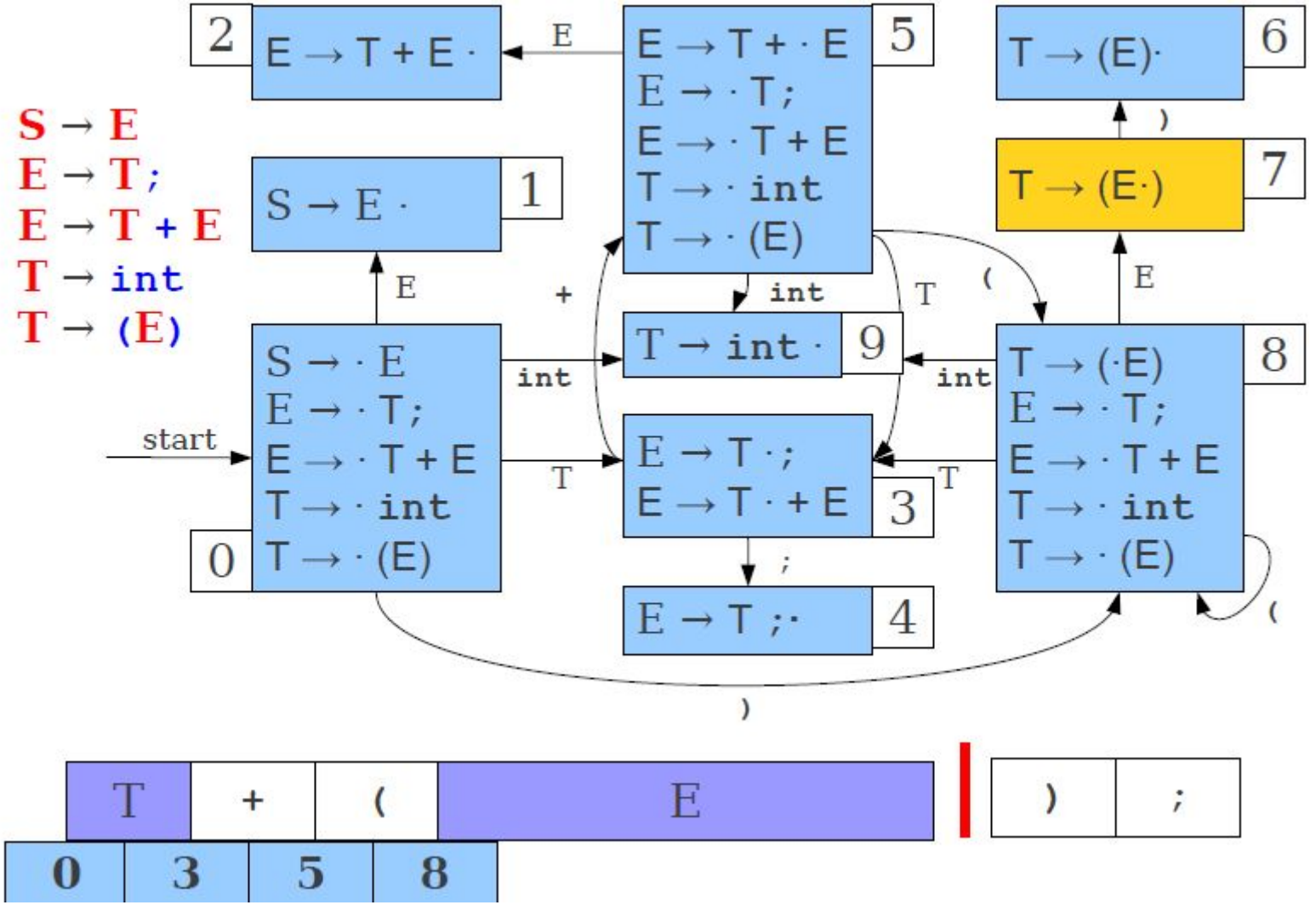
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
T → int ·
9
int
int
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
T
E → T ·;
E → T · + E
3
T
T
;
E → T ;·
4
(
)
T
+
(
0
3
5
8
)
;

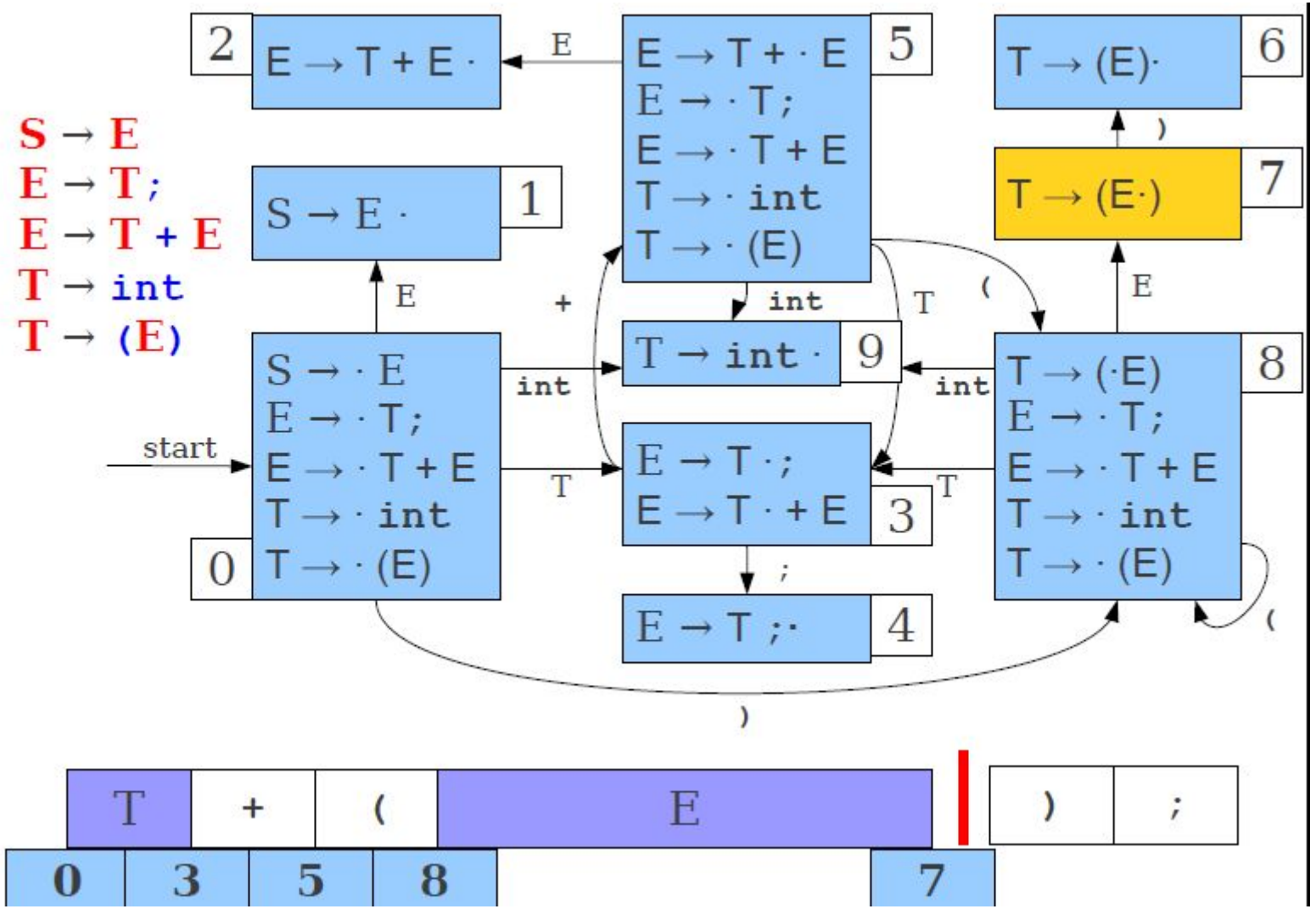
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
S → E ·
1
E → T + E ·
2
E → T · ;
E → T · + E
3
E → T ;·
4
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
T → (E·)
7
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
T → int · 9
int
+
E
T
(
)
;
T
+
(
E
)
;
0
3
5
8

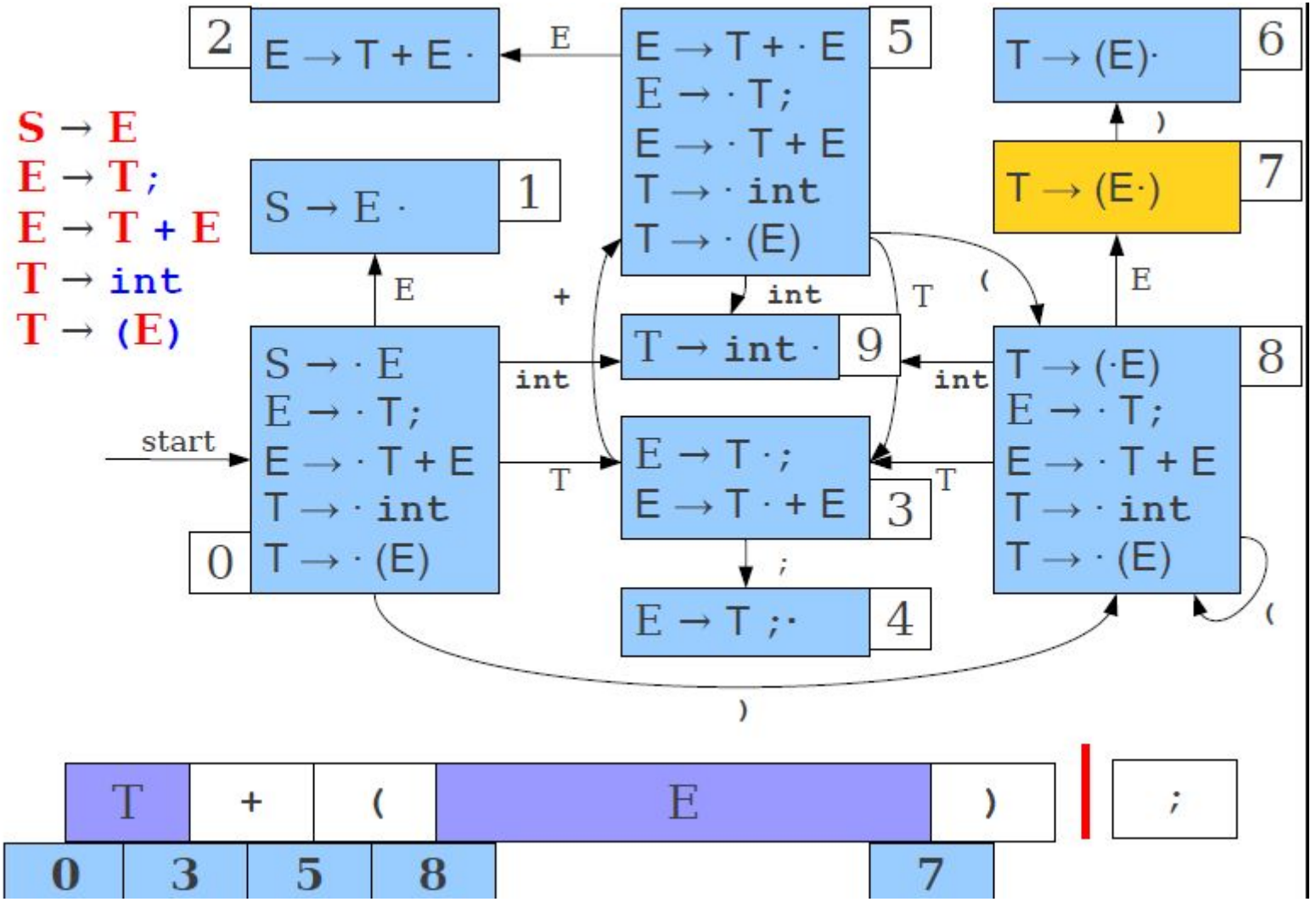
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
S → E ·
1
E → T + E ·
2
E → T · ;
E → T · + E
3
E → T ;·
4
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
T → (E·)
7
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
T → int · 9
E
+
T
int
(
)
;
T
+
(
E
0
3
5
8
)
;

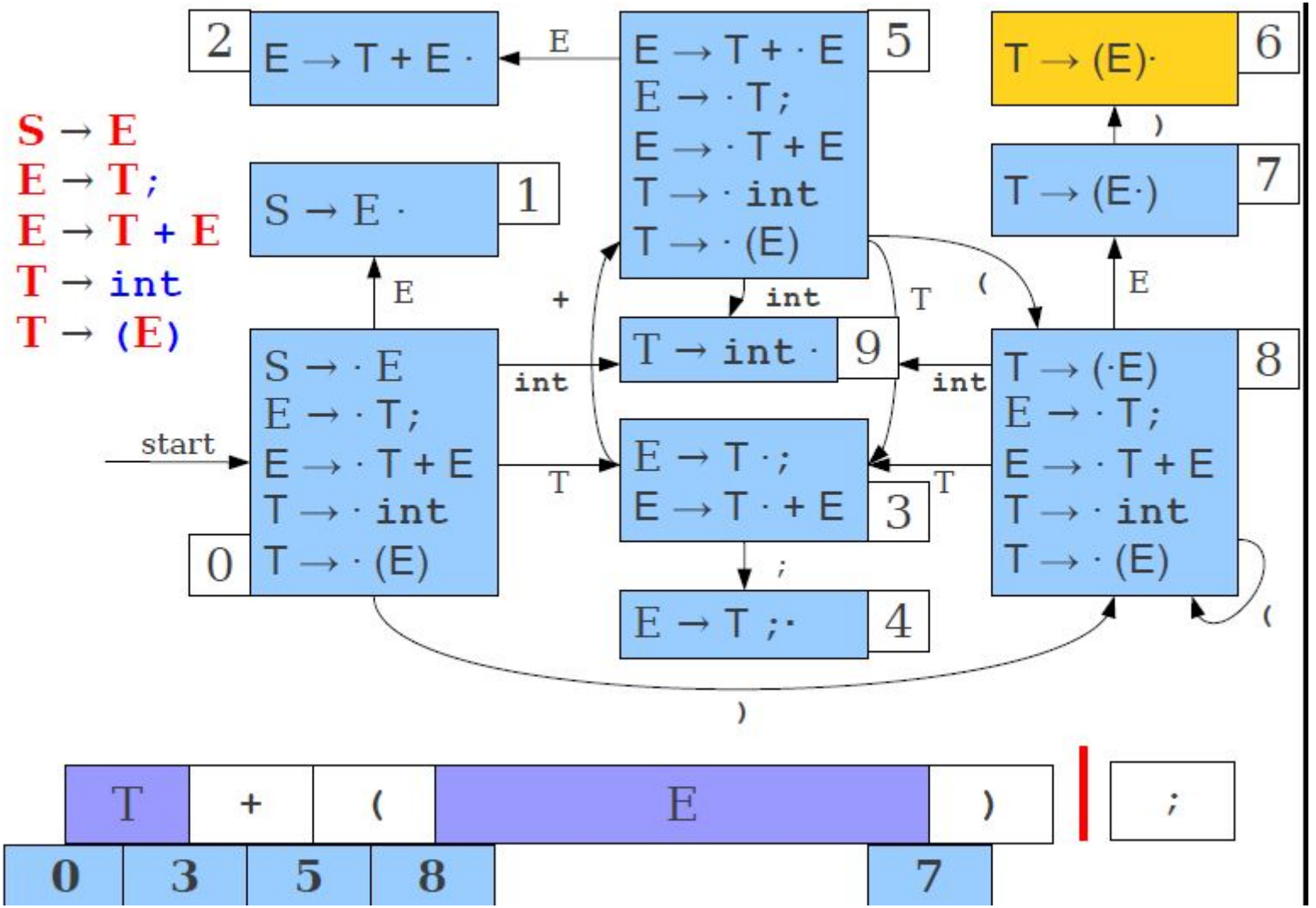
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
T → int ·
9
int
int
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
start
S → · E
E → · T;
E → · T + E
T → · int
0
T → · (E)
T
E → T ·;
E → T · + E
3
T
;
E → T ;·
4
(
)
T
+
(
E
)
;
0
3
5
8
7

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)
0
S → E ·
1
E → T + E ·
2
E → T ; ·
4
E → T · ;
E → T · + E
3
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → int ·
9
T → (E)·
6
T → (E·)
7
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
E
+
T
int
(
)
;
T
+
(
E
)
;
0
3
5
8
7

LR(0) Parsing

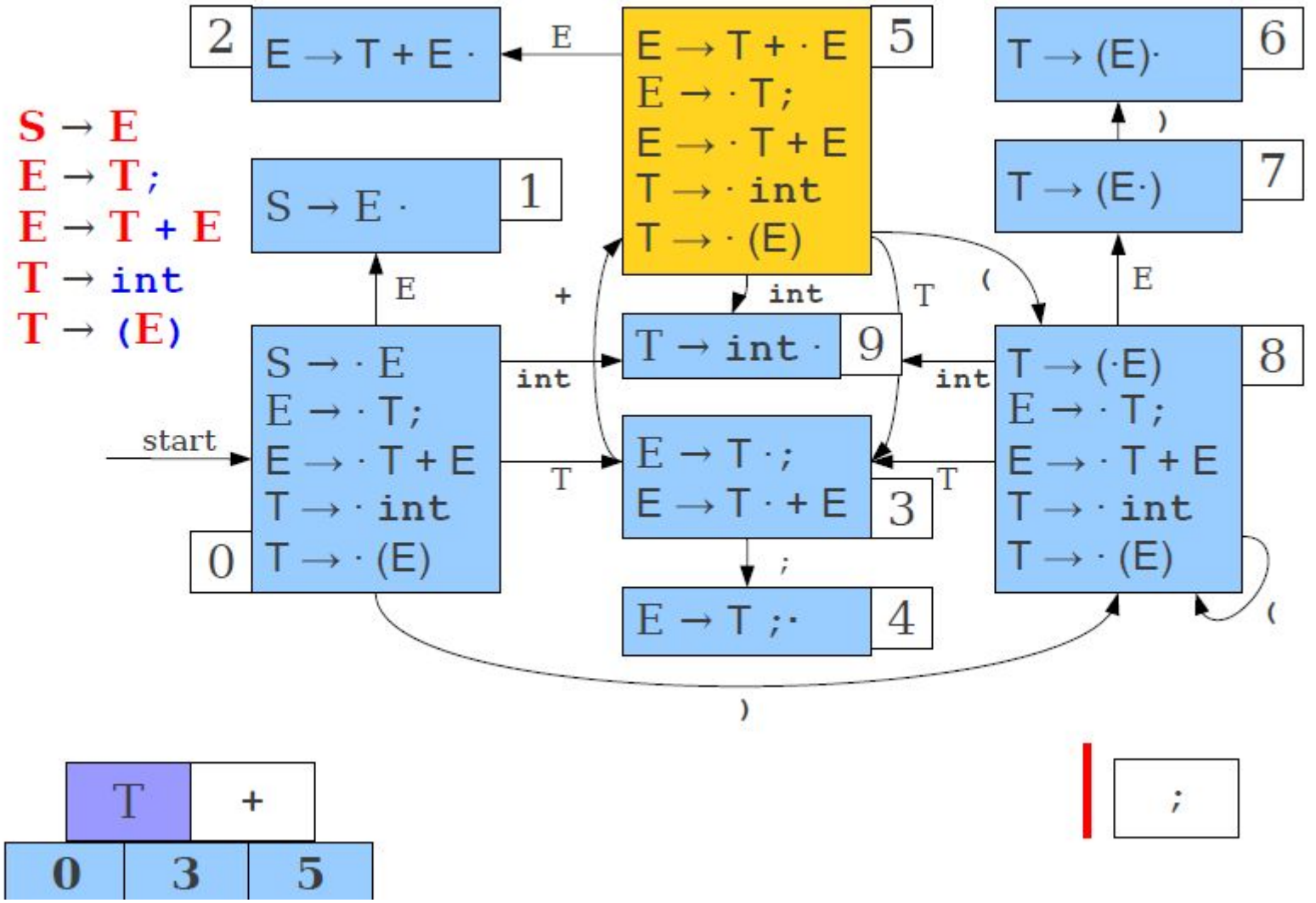

S → E
E → T;
E → T + E
T → int
T → (E)
0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
start
1
S → E ·
2
E → T + E ·
3
E → T · ;
E → T · + E
4
E → T ;·
5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
6
T → (E)·
7
T → (E·)
8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
9
T → int ·
int
+
T
E
(
)
;
T
+
(
E
)
;
0
3
5
8
7

LR(0) Tables

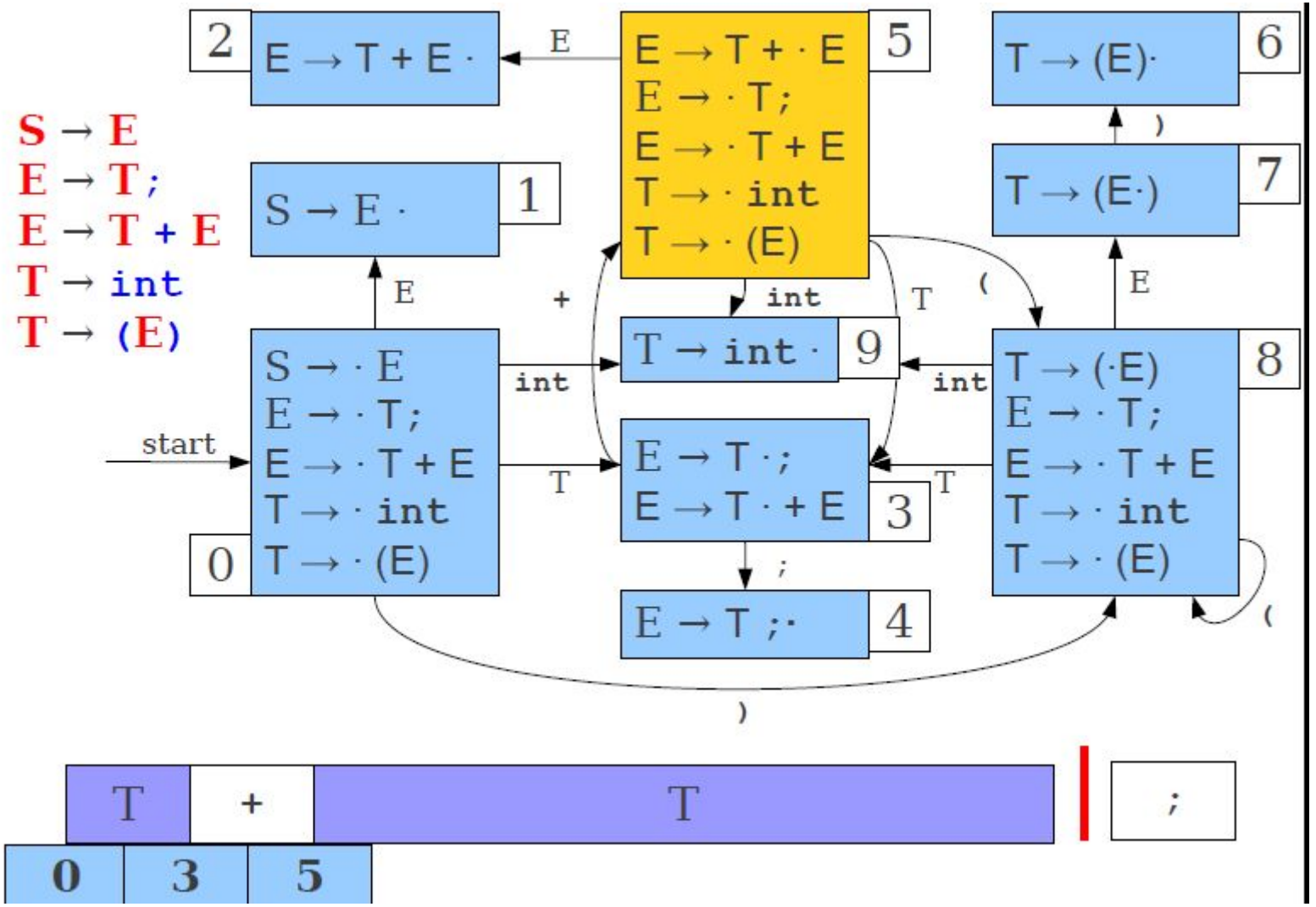
(1) S → E
(2) E → T;
(3) E → T + E
(4) T → int
(5) T → (E)

	int	+	;	(	)	E	T
0	S9			S8		S1	S3
1							
2	r3	r3	r3	r3	r3		
3		S5	S4				
4	r2	r2	r2	r2	r2		
5	S9			S8		S2	S3
6	r5	r5	r5	r5	r5		
7					s6		
8	S9			S8		S7	S3
9	r4	r4	r4	r4	r4		

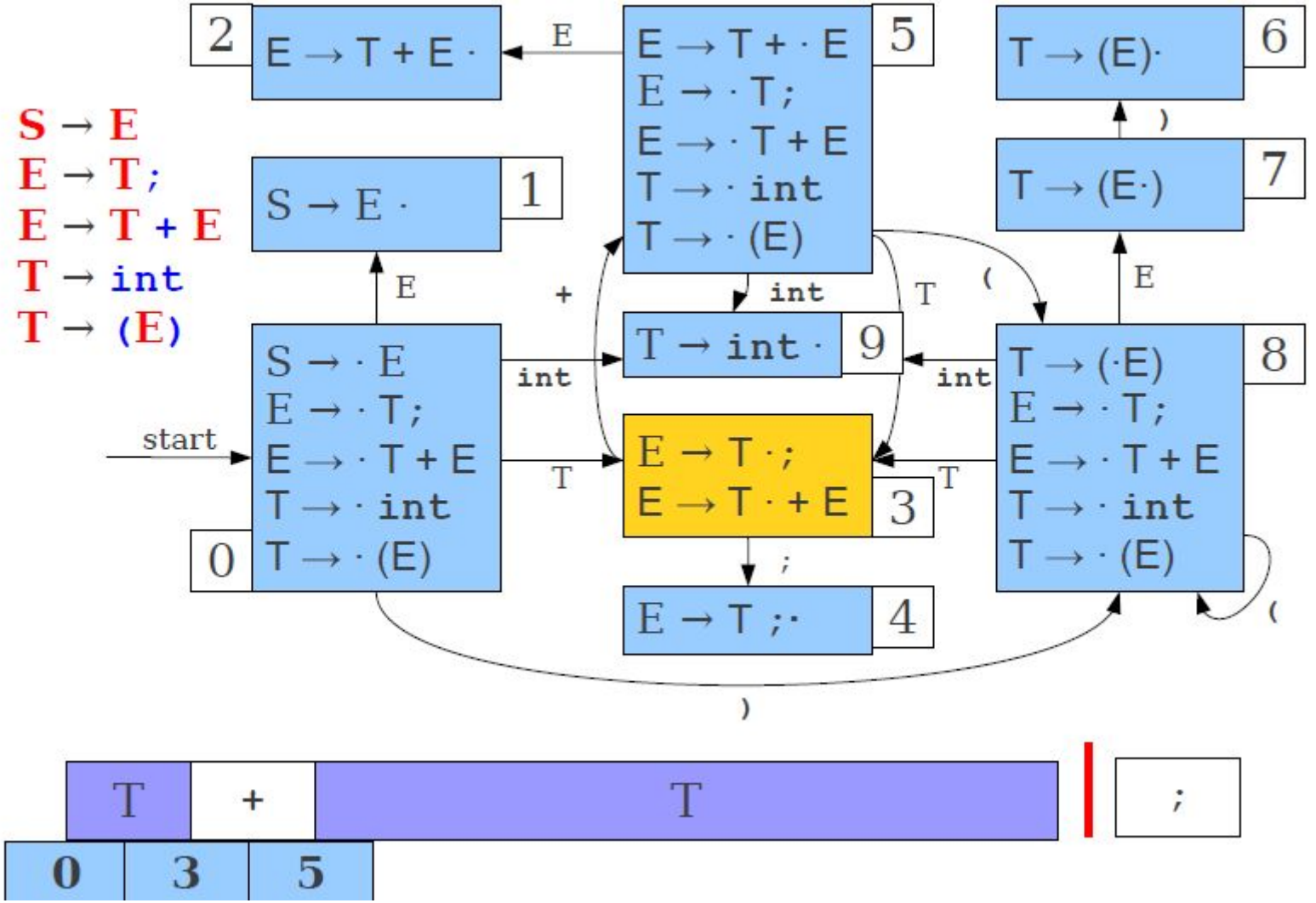
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
T → int ·
9
int
int
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
T
E → T ·;
E → T · + E
3
T
;
E → T ;·
4
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
(
)
T
+
0
3
5
;

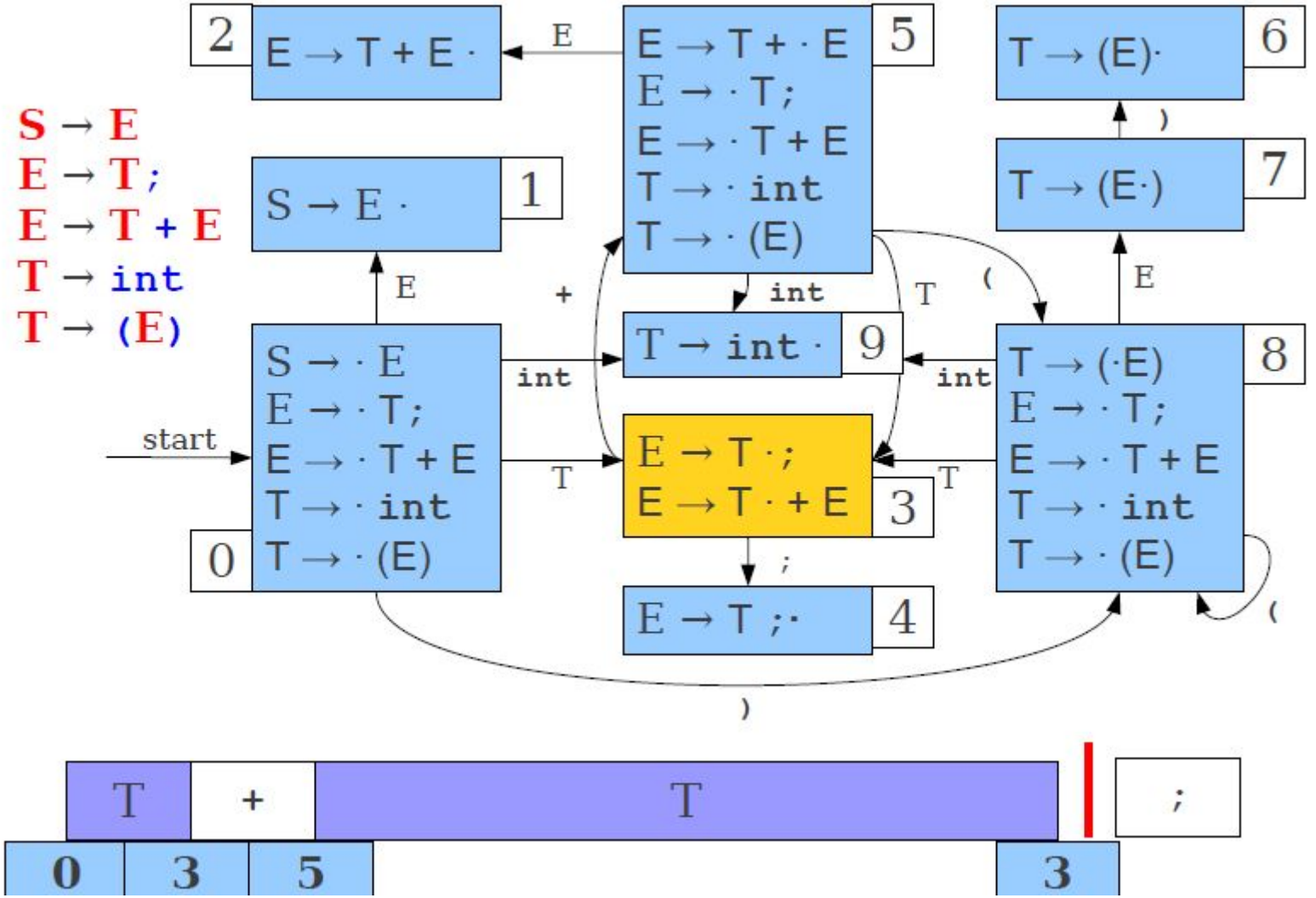
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
S → E ·
1
E → T + E ·
2
E → T ·;
E → T · + E
3
E → T ;·
4
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
T → (E·)
7
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
T → int ·
9
E
T
int
+
(
)
;
T
+
T
;
0
3
5

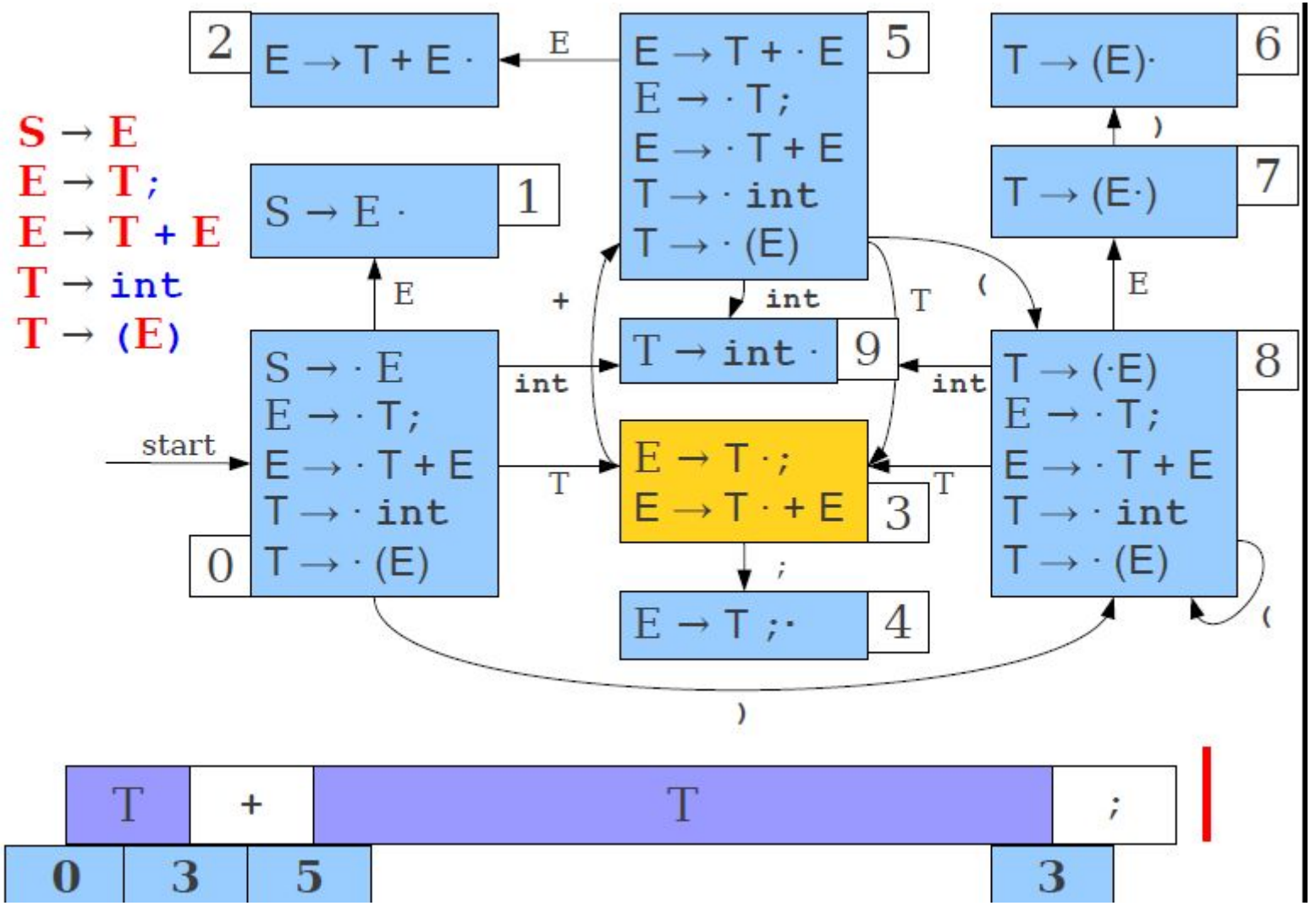
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
start
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 ·
E
+
T
int
(
)
;
T
+
T
;
0
3
5

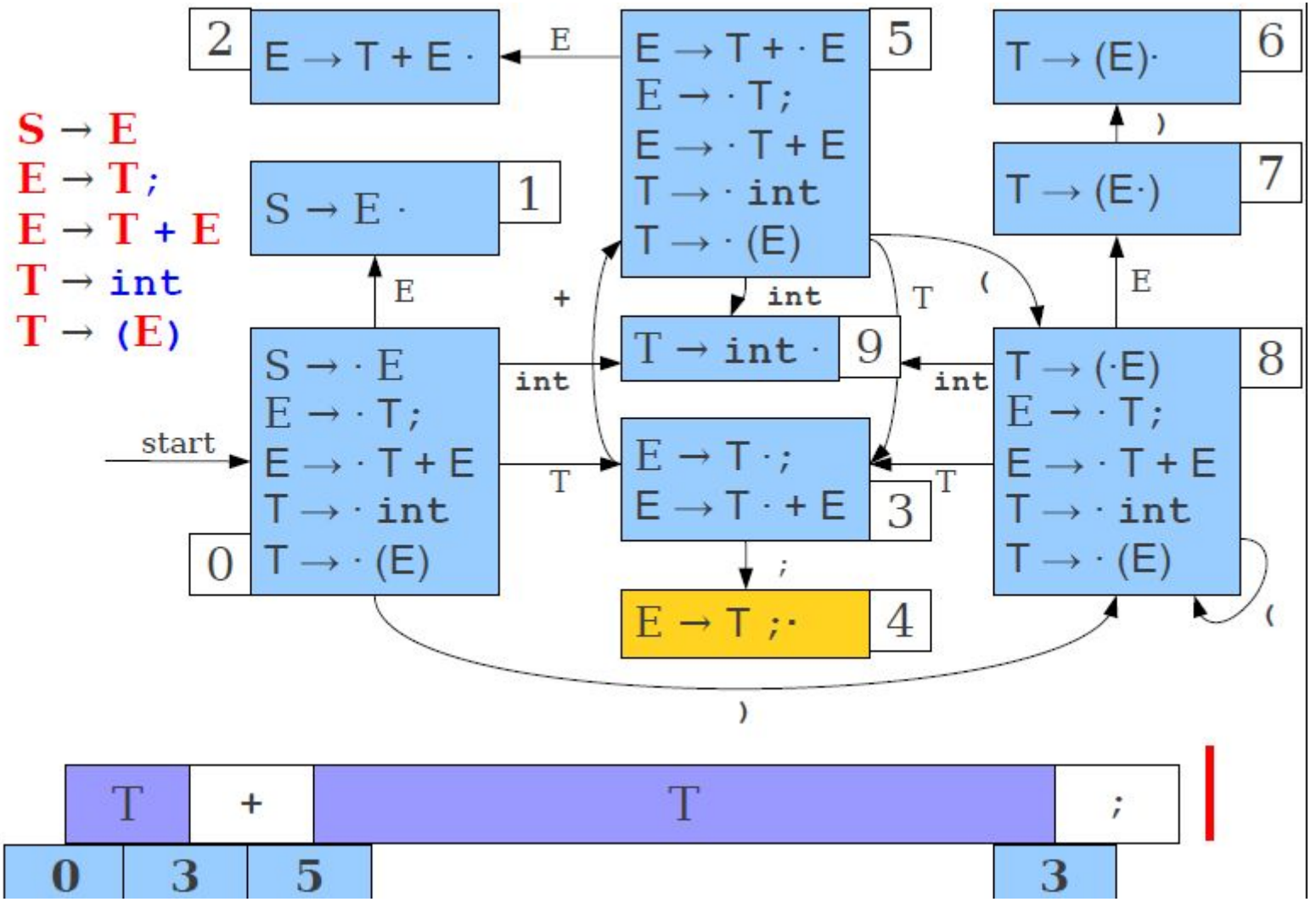
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
T → int ·
9
int
int
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
T
E → T ·;
E → T · + E
3
T
E → T ;·
4
;
)
(
T
+
T
;
0
3
5
3

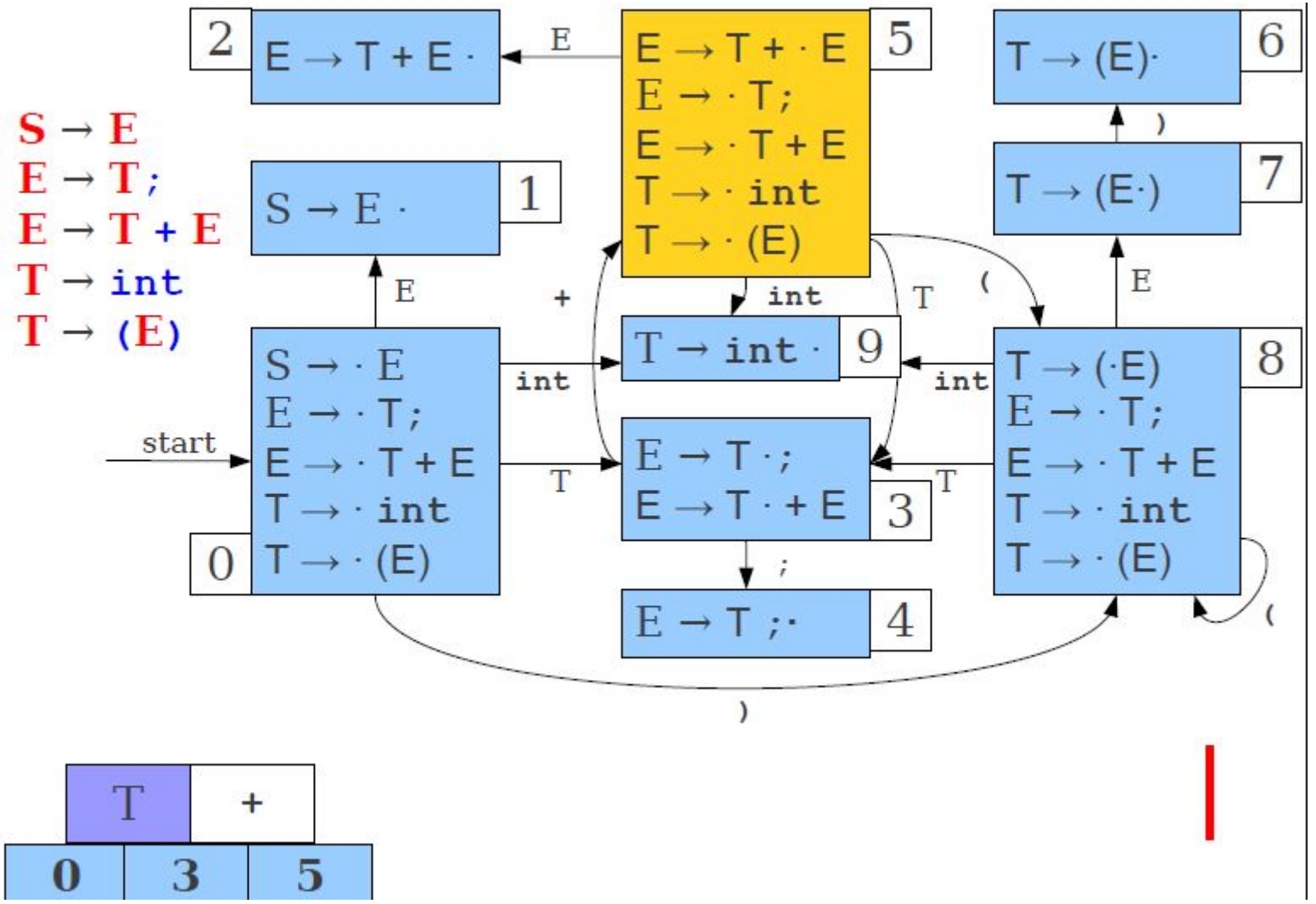
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
T → int ·
9
int
int
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
T
E → T ·;
E → T · + E
3
T
E → T ;·
4
;
)
(
T
+
T
;
0
3
5
3

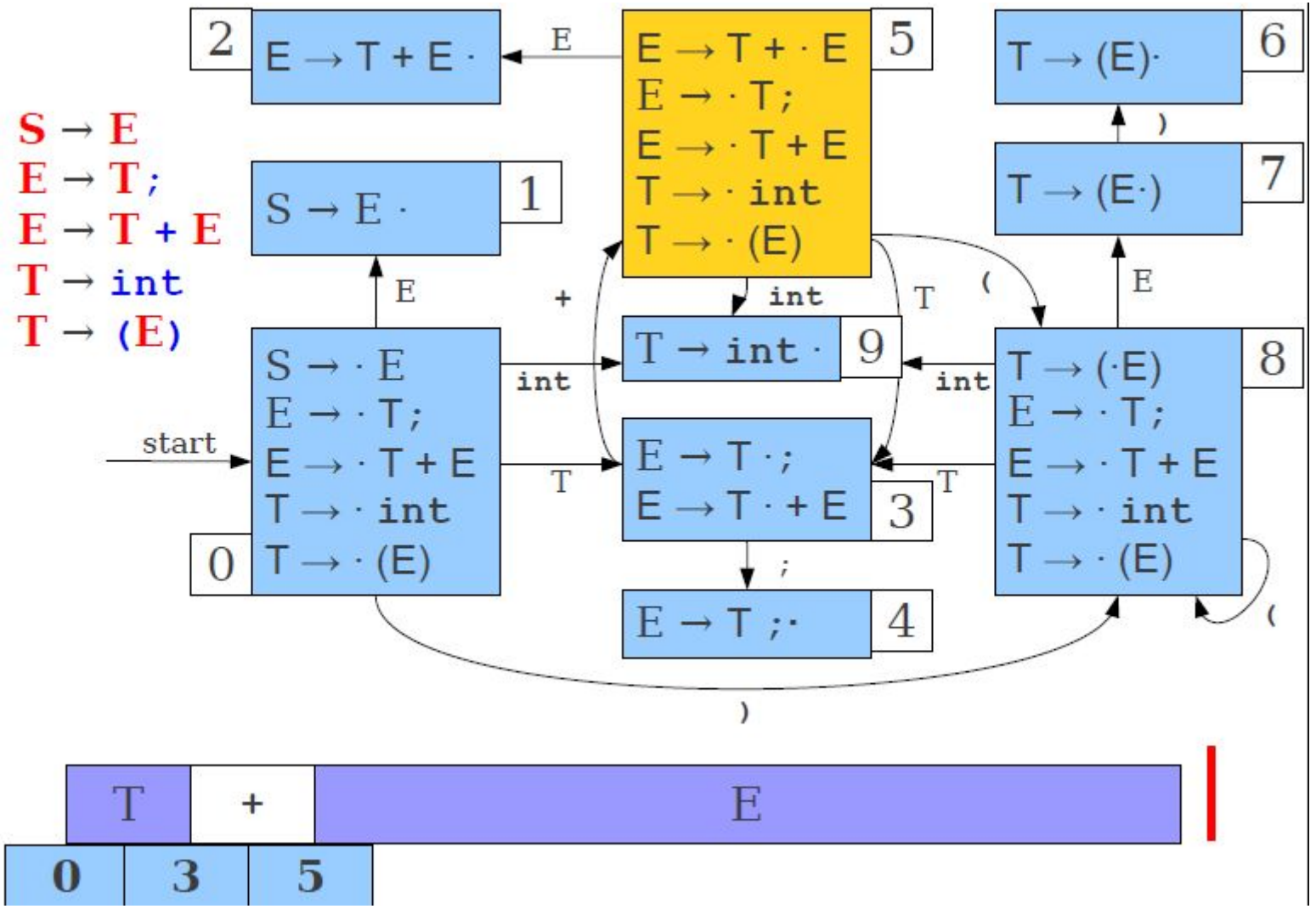
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
start
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 ·
E
+
T
int
(
)
;
T
+
T
;
0
3
5
3

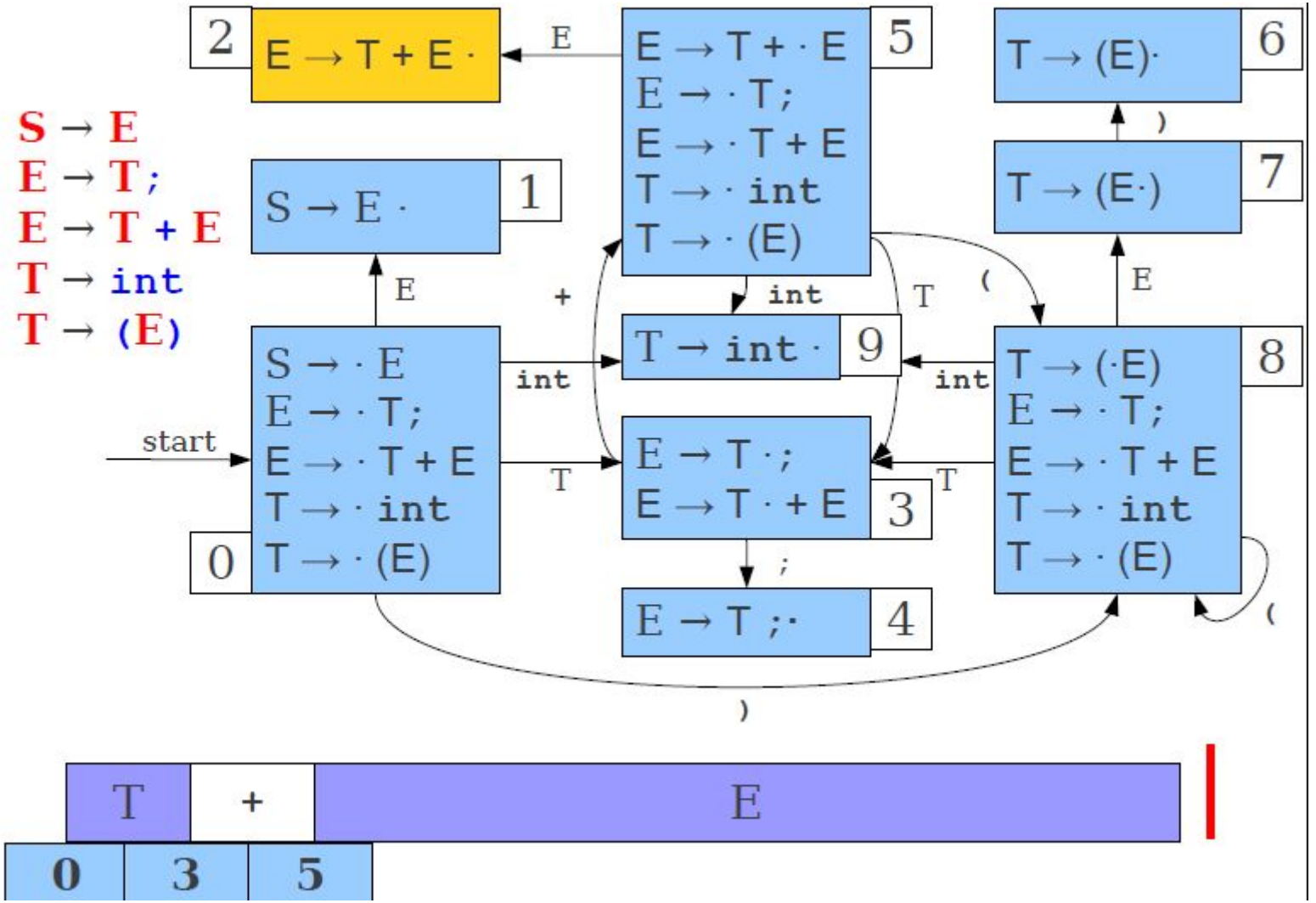
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
T → int ·
9
int
int
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
start
0
E → T ·;
E → T · + E
3
T
T
;
E → T ;·
4
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
(
)
T
+
0
3
5

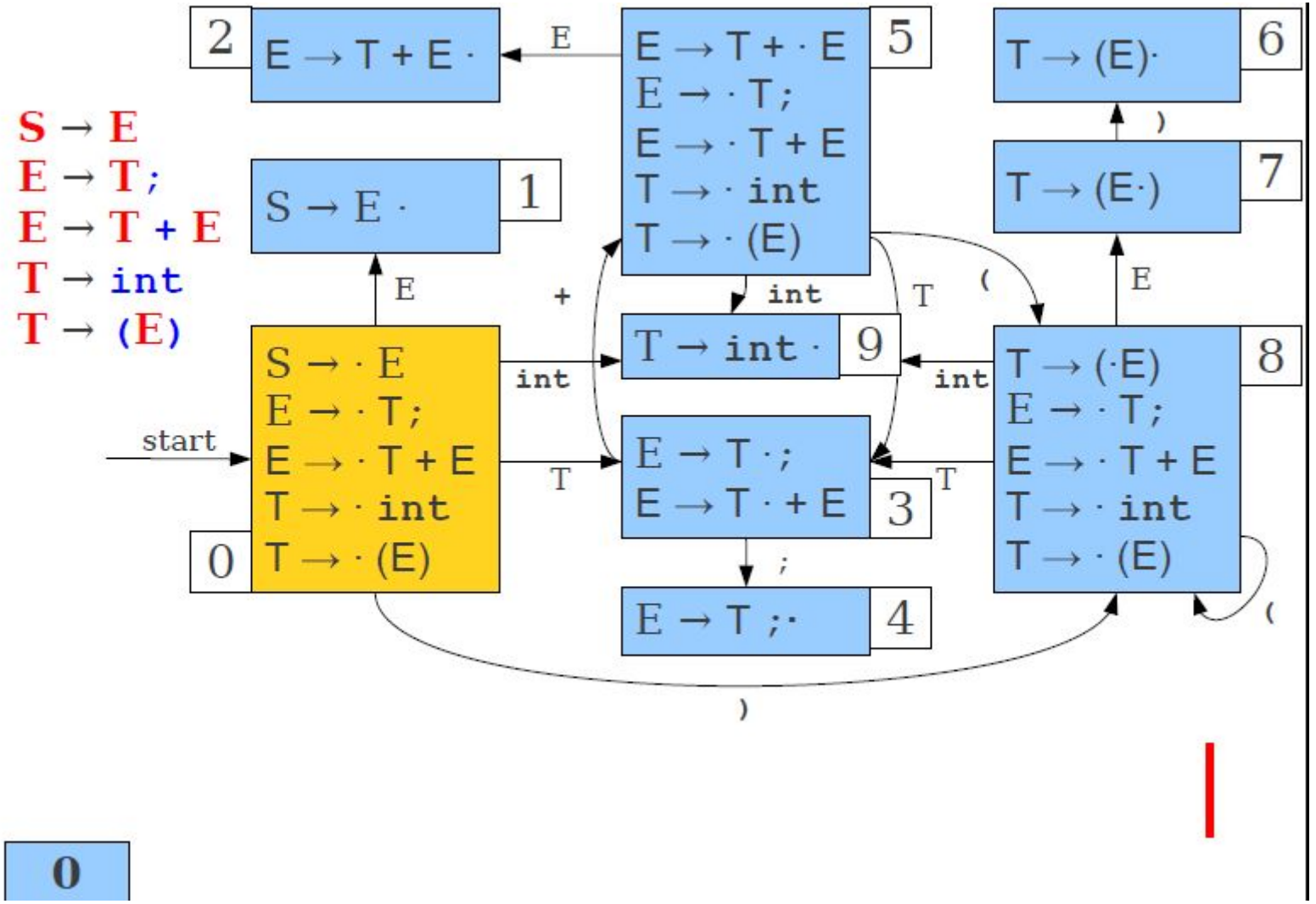
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
start
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 ·
E
T
int
+
;
(
)
T
+
E
0
3
5

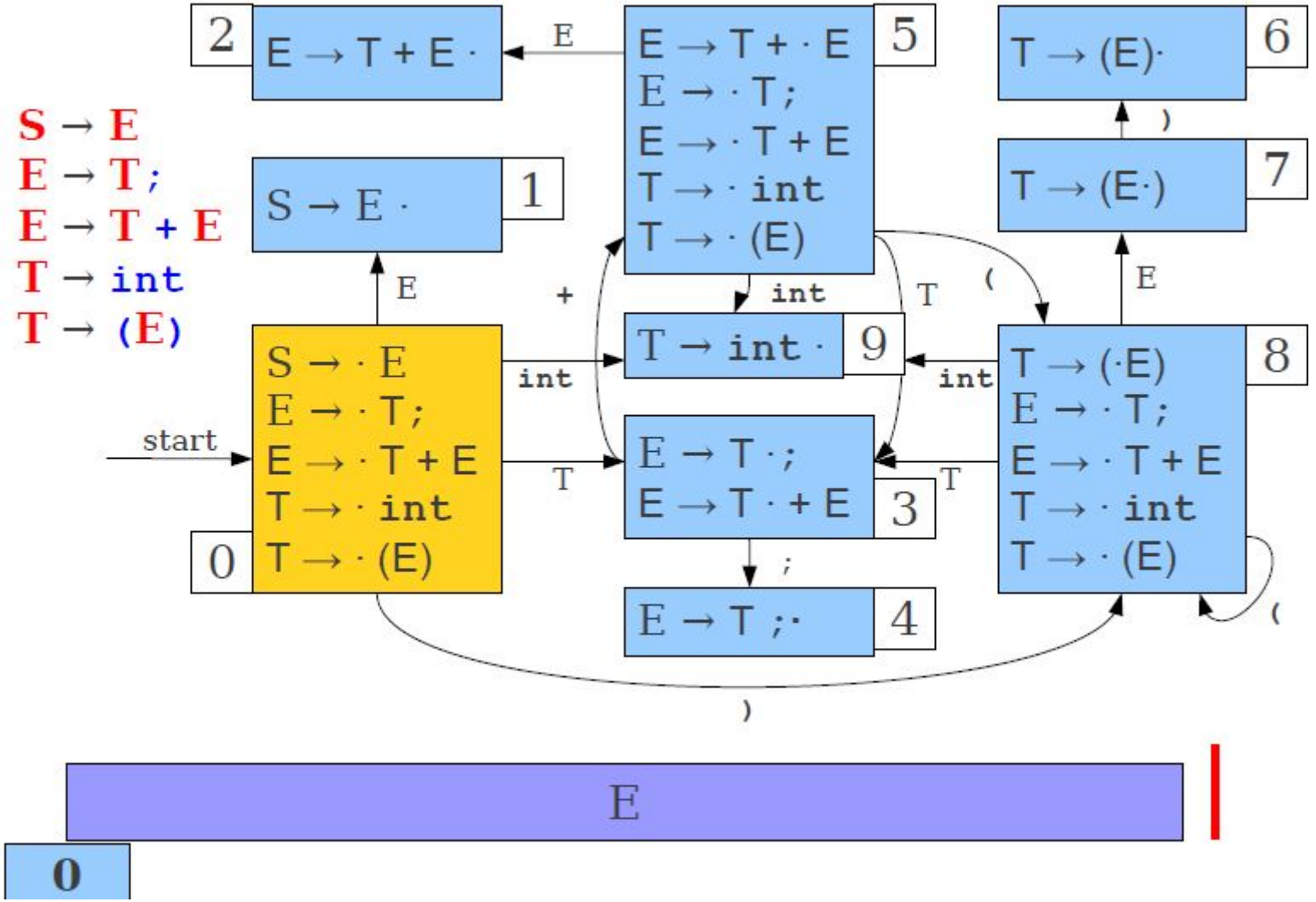
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
S → E ·
1
E → T + E ·
2
E → T ·;
E → T · + E
3
E → T ;·
4
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
T → (E·)
7
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
T → int ·
9
E
+
T
int
(
)
;
T
+
E
0
3
5

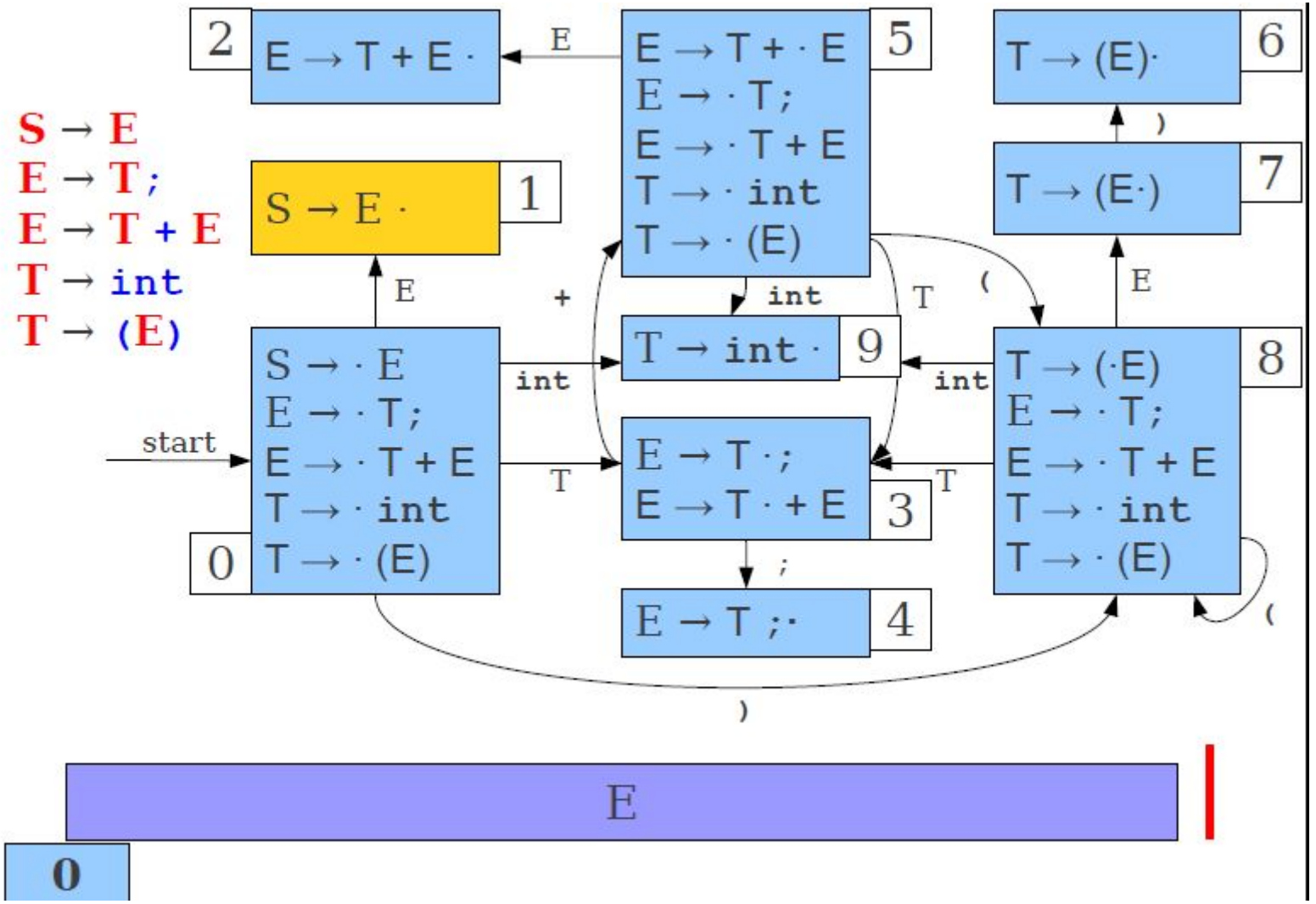
LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
6
T → (E)·
)
1
S → E ·
7
T → (E·)
E
+
int
T
(
E
0
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
start
int
9
T → int ·
int
8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
T
T
3
E → T ·;
E → T · + E
T
;
4
E → T ;·
)
(
0

LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
T → int ·
9
int
int
start
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
T
E → T ·;
E → T · + E
3
T
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
(
;
E → T ;·
4
)
E
0

LR(0) Parsing


S → E
E → T;
E → T + E
T → int
T → (E)
2
E → T + E ·
E
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
5
T → (E)·
6
)
T → (E·)
7
S → E ·
1
E
+
int
T
(
E
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
0
start
int
T → int ·
9
int
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
8
T
E → T ·;
E → T · + E
3
T
;
E → T ;·
4
)
E
0

LR(0) Tables

(1) S → E
(2) E → T;
(3) E → T + E
(4) T → int
(5) T → (E)

	Action					Goto	
	int	+	;	(	)	E	T
0	S9			S8		S1	S3
1	Acc	Acc	Acc	Acc	Acc		
2	r3	r3	r3	r3	r3		
3		S5	S4				
4	r2	r2	r2	r2	r2		
5	S9			S8		S2	S3
6	r5	r5	r5	r5	r5		
7					s6		
8	S9			S8		S7	S3
9	r4	r4	r4	r4	r4		

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** $A \rightarrow \omega$, which performs reduction $A \rightarrow \omega$.
  - Any state of the form $A \rightarrow \omega \cdot$ does that reduction; everything else shifts.
- The **goto** table maps state to a next state.
  - This is just the transition table for the automaton.

LR(0) Tables

(1) S → E
(2) E → T;
(3) E → T + E
(4) T → int
(5) T → (E)

	Action					Goto	
	int	+	;	(	)	E	T
0	S9			S8		S1	S3
1	Acc	Acc	Acc	Acc	Acc		
2	r3	r3	r3	r3	r3		
3		S5	S4				
4	r2	r2	r2	r2	r2		
5	S9			S8		S2	S3
6	r5	r5	r5	r5	r5		
7					s6		
8	S9			S8		S7	S3
9	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** $A \rightarrow \omega$:
    - Remove $|\omega|$ 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.